VMV 2007

H. P. A. Lensch, B. Rosenhahn,
H.-P. Seidel, P. Slusallek, J. Weickert (Eds.)

Vision, Modeling, and Visualization 2007

Proceedings
November 7-9, 2007
Saarbrücken, Germany

Saarbrücken Berlin

Bibliographic information published by *Die Deutsche Bibliothek*

Die Deutsche Bibliothek lists this publication in the *Deutsche Nationalbibliografie*; detailed bibliographic data is available on the Internet at http://dnb.ddb.de

Printing and Binding: Digitaldruck Pirrot GmbH, Saarbrücken, Germany

ISBN 978-3-940739-00-1
ISBN 978-3-89838-085-0

Preface

Vision, modeling, and visualization are rapidly converging disciplines. Often the three fields are treated separately, each with their own methodology and terminology. However, in many applications they must work hand in hand to evoke cross-fertilization and new research directions. Prominent examples are image and video-based rendering, 3D-TV, and medical visualization. Also, due to the proliferation of digital photography and high-speed graphics accelerators, image-based modeling and the corresponding vision and visualization techniques have become a highly relevant research topic.

The *Vision, Modeling, and Visualization 2007* workshop addresses the entire spectrum of techniques and applications in this combined field, from data acquisition over processing to visualization, including perceptional issues. This workshop is the twelfth in a series of annual meetings organized by different research centers and groups. After highly successful meetings at various locations, this year's event takes place in Saarbrücken, Germany.

The call for papers drew 49 contributions from 15 different countries. Each submission was anonymously reviewed by two members of the Program Committee and two external reviewers. These proceedings contain the 27 papers which were accepted. They present an excellent cross section of ongoing research in the fields of vision, modeling, and visualization. The papers are organized in 7 sessions: Visualization, GPU, 3D Acquisition and Processing, 2D/3D Image Processing, Mesh Processing, Learning and Recognition, and Medical Visualization.

In addition, we are happy that three internationally renowned experts have accepted our invitation to present keynote speeches:

- Thomas Vetter, University of Basel, Switzerland
- Michael Goesele, TU Darmstadt, Germany
- Ramesh Raskar, Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA, USA

Furthermore, the workshop features a special session entitled *Gigapixel displays, a challenge for visualization?* organized by the BW-FIT reserach cluster on gigapixel visualization.

The conference is organized by the Computer Graphics Group at the Max Planck Institute for Computer Science and by Saarland University in cooperation with the Max Planck Center for Visual Computing and Communication. Carsten Stoll and Conny Liegl have managed the conference web-site and the conference management system. The proceedings have been assembled by Oliver Schall and Wolfram von Funck. Sabine Budde is the conference secretary and Christel Weins is the financial and social events coordinator.

Saarbrücken, November 2007

Hendrik Lensch, Bodo Rosenhahn, Hans-Peter Seidel, Philipp Slusallek, and Joachim Weickert

General Chair
Hans-Peter Seidel
Philipp Slusallek
Joachim Weickert

Technical Program Chairs
Hendrik P. A. Lensch
Bodo Rosenhahn

Proceedings
Oliver Schall
Wolfram von Funck

Publicity
Christel Weins

Website
Conny Liegl

Local Arrangements/Registration
Sabine Budde

Social Event
Christel Weins
Sabine Budde

Finances
Christel Weins

Conference Secretary
Sabine Budde

Technical Program Committee

Volker Blanz	Heinrich Müller
Mario Botsch	Heinrich Niemann
Andrés Bruhn	Carol O'Sullivan
Hans Burkhardt	Dietrich Paulus
Joachim Denzler	Pieter Peers
Oliver Deussen	Konrad Polthier
Phil Dutré	Bernhard Preim
Peter Eisert	Stefan Roth
Thomas Ertl	Szymon M. Rusinkiewicz
Dieter Fellner	Christoph Schnörr
Michael Goesele	Hans-Peter Seidel
Günther Greiner	Mike Sips
Eduard Gröller	Philipp Slusallek
Baining Guo	Oliver Staadt
Peter Hastreiter	Marc Stamminger
Hans-Christian Hege	Eckehard Steinbach
Nicolas Holzschuch	Matthias Teschner
Kai Hormann	Holger Theisel
Jan Kautz	Michael Wand
Leif Kobbelt	Joachim Weickert
Reinhard Koch	Daniel Weiskopf
Torsten Kuhlen	Rüdiger Westermann
Jochen Lang	Tim Weyrich
Marcus Magnor	Simon Winkelbach

Contents

Invited Talk

Morphable Models for Faces, Skulls and Cars

Thomas Vetter

Department of Computer Science
University of Basel, Switzerland

Abstract

Morphable models constitute a unifying framework for the analysis and synthesis of images. In the field of Computer Graphics, they are applied to model photo-realistic face images; in the domain of Computer Vision, they are used in face recognition applications compensating variations across pose, illumination and facial expressions. Morphable face models draw on prior knowledge of human faces in the form of a general face model, learned from examples of other faces. By exploiting the correspondences between all examples, these models introduce a vector space structure on the examples that allows to synthesize novel photo-realistic images. Image analysis can be performed by fitting such a flexible model to novel images. Then, the model parameters yielding the optimal reconstruction are used to code or analyze the face depicted. In this talk, I start with a quick review on morphable face models and will discuss some of its limitations for face recognition applications based on skin detail analysis. In a second part I will report on current work on building morphable models of human bones and skulls. Here we use a novel data registration technique that does not require a parametric representation of the example data. Finally, I will conclude with presenting a Morphable Model for the design and modification of automotive shapes.

Surface Glyphs for Visualizing Multimodal Volume Data

Timo Ropinski[1], Michael Specht[1,2], Jennis Meyer-Spradow[1], Klaus Hinrichs[1], Bernhard Preim[2]

[1]Visualization and Computer Graphics Working Group (VisCG), University of Münster
[2]Department of Simulation and Graphics (ISG), University of Magdeburg
Email: {ropinski, spech_m01, spradow, khh}@math.uni-muenster.de,
preim@isg.cs.uni-magdeburg.de

Abstract

In this paper we present concepts for integrating glyphs into volumetric data sets. These concepts have been developed with the goal to make glyph-based visualization of multimodal volumetric data sets more flexible and intuitive. We propose a surface-based glyph placement strategy reducing visual clutter as well as image-space glyph aggregation. Thus the user is not distracted by unwanted clustering, and his focus of attention can rather be guided using appropriate visual appearances. Furthermore, we present techniques to make the setup of glyph-based visualizations more intuitive. These concepts have been integrated into a user interface which supports easy configuration and comparison of different glyph setups. Based on the chosen setup a visual legend is generated automatically to make a step towards quantitative visual analysis. We will discuss the placement strategy as well as the glyph setup process, explain the used rendering techniques and provide application examples of multimodal visualizations using the proposed concepts.

1 Introduction

Multimodal volume visualization has to deal with the proper integration of data obtained from different sources. In the medical domain acquisition of different modalities is about to become a daily routine since modern scanners such as PET/CT, PET/MRT or SPECT/CT can be used to capture multiple registered volume data sets. PET data sets which usually have a lower resolution than CT data sets represent a functional image, e.g., metabolism activity, while CT data sets provide a detailed morphological image. To benefit from both modalities, they have to be visualized simultaneously in an integrated manner. In contrast to these modalities additional data can be derived from a given volume data set and visualized with our technique. For instance, cardiac wall thickness or wall motion can be calculated from time-varying medical data sets. The use of multimodal volumetric data sets is manifold not only in medicine but also in areas like meteorology and seismology. Furthermore, many physical simulations, e.g., fluid dynamics or quality insurance simulations, produce multimodal volumetric data sets.

A common approach to visualize volumetric data sets containing two modalities is to generate a fusion image by blending between these. For instance, modern medical workstations support such a fusion imaging to explore registered PET/CT data sets and allow the user to control the degree of blending by using a slider. This fusion imaging has two drawbacks. First, this form of user control introduces an additional degree of freedom which makes it difficult to find adequate visualization parameters. Second, quantification becomes more difficult since the blending has a major influence on the colors used to represent intensities stored within the data sets, and the colors may interfere with the surface shading. While these issues may be manageable when dealing with only two modalities, the blended fusion image is insufficient if more than two modalities have to be taken into account.

Using glyphs in volume visualization is not new [6, 7, 8]. However, most of the proposed work has rather focussed on ways glyphs can be used to visualize information than on how to flexibly setup this information and make it quantifiable. The contribution of this paper is to make glyphs more usable as a tool for multimodal volume visualization. Therefore we propose a surface-based glyph placing strategy which decreases unwanted glyph clustering and reduces clutter by minimizing

glyph occlusion. Furthermore, we will introduce a user interface, which allows an easy configuration of a glyph setup, i.e., the definition how properties are represented by glyphs and how the glyphs are placed. Based on this setup we describe how to generate a visual legend which is a necessary step towards a quantitative visual analysis of glyph visualizations.

The paper is structured as follows. In the next section related work is discussed. The so called surface glyphs and details regarding their placement are discussed in Section 3. In order to support a flexible and intuitive setup of a glyph-based visualization we propose some techniques accompanied with appropriate user interface concepts in Section 4. The used rendering technique is briefly described in Section 5, while application examples are given in Section 6. The paper concludes in Section 7.

2 Related Work

This section briefly discusses some work related to the topic of this paper, but does not intend to give a complete overview of glyph visualization techniques used in scientific as well as information visualization. Moreover, we focus on those techniques which are related to our approach for visualizing volumetric multimodal data sets.

Ward [18] states that glyph-based visualization is a powerful method for providing multimodal information, by adding iconic glyphs to a scene in order to display various variables through various properties such as shape or color. In his work he describes and classifies different glyph placing strategies and proposes rules for their usage in the context of information visualization.

In the field of scientific visualization diffusion tensor imaging (DTI) is probably the domain where the usage of glyphs has been most intensively investigated [10, 17, 8]. Most of the work published in this area focuses rather on choosing an appropriate glyph shape in order to transmit information [5] than on positioning of the glyphs. For instance, in [8] superquadrics are used to convey the principal eigenvectors of a diffusion tensor in order to depict the microstructure of white-matter tissue of the human brain. The distinct glyphs are placed in a regular grid and controlled by a fractional anisotropy threshold in order to minimize visual clutter. Jankun-Kelly and Mehta [6] have used superquadric ellipsoid glyphs to visualize traceless tensor data.

To achieve a beneficial glyph visualization, not only the shape of the used glyphs but also their placement is essential. In [10] a stochastically jittered placing of glyphs is described with the goal to eliminate the possibly distracting effects of a grid placement. In 2006 Kindlmann and Westin [7] have proposed a glyph packing strategy allowing a texture like appearance of a glyph aggregation. In contrast to the approach presented in this paper only one modality, i.e., DTI data, is used, while our glyph placing strategy is based on the fusion of multiple modalities. In [13] spherical glyphs are exploited to visualize cardiac MR data in order to allow an exploration of the structure as well as the function of the myocard.

Besides in the medical domain glyphs are also used in other areas dealing with multimodal data. Nayak et al. [12] have used glyphs in order to visualize seismic data representing the measured, time-dependent, 3D wave field of an earthquake recorded by a seismic network. Reina and Ertl [14] have proposed a technique for depicting molecular dynamics by means of hardware-accelerated glyph rendering, and Saunders et al. [15] have proposed the use of circular glyphs for visualizing nano particles in formation.

Besides using glyphs there are other strategies to visualize multi-field volume data. Recently Akiba et al. [1] have presented a novel user interface concept to support the identification of correlation. Their technique exploits linked views as well as parallel coordinates and is demonstrated by visualizing the hurricane Isabel data set. Similar to the concepts presented in [4] their approach supports brushing, to allow the user to formulate the current interest. Blaas et al. have introduced a technique which also exploits linked views [3]. Besides a physical view they use a feature-space view to visualize multi-field data.

3 Surface Glyphs

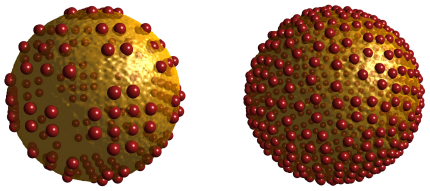
Glyph placement is crucial in order to achieve meaningful glyph-enhanced visualizations. Glyphs that occlude each other or occlude big portions of the volume can easily counteract the information increase intended by the use of glyphs. Therefore it

must be ensured that glyph occlusion is minimized, and that not too many glyphs are placed.

A basic approach for placing glyphs is to superimpose a regular three-dimensional grid on the volumetric data set and to place the glyphs at the grid points (see Figure 1(a)). However, this method has two drawbacks. First, the introduced grid structures do not necessarily match the structure of the underlying data set, e.g., in the example the underlying data set shows a round, smooth sphere. Second, placing glyphs in a regular grid may lead to the false impression of a *glyph aggregation* resulting from the glyph positions in image space. These aggregations attract the user's attention, although the user's focus could be better guided by the visual appearance of the glyphs, i.e., by the glyph properties and user-defined glyph property mapping functions (see Section 6).

To eliminate these drawbacks, we propose a surface-based glyph placement strategy. It achieves a feasible glyph distribution that fulfills both of the requirements stated above by placing glyphs on isosurfaces of a volumetric data set. A straightforward way of defining such an isosurface is to specify an isovalue as it is done in isosurface rendering. The effect of isosurface placement is that glyphs are placed at points that are exactly located on an isosurface (see Figure 1(b)). Isosurface placement leads to fewer glyph occlusions and also avoids misleading glyph aggregation in image space.

A possible approach to realize isosurface glyph placement is to calculate a polygonal mesh using the marching cubes algorithm [11], and then to randomly choose polygons and place glyphs on them.



(a) Regular grid placement. (b) Isosurface placement.

Figure 1: Comparison of the regular grid and isosurface glyph placement methods. The result of the isosurface placement is visually more pleasing because the glyphs are evenly distributed on the surface and occlusion is minimized.

Uniform distribution could be achieved by calculating the surface area of each polygon and taking polygon surface area distribution into account when polygons are chosen randomly. Apparently, the contiguous surface resulting from the marching cubes algorithm is not necessary for isosurface glyph placement. Distinct glyphs should be placed at certain locations near an or even exactly on an isosurface, but it does not matter whether two locations of interest are directly connected by an isosurface. This makes the marching cubes approach inappropriate, especially if one considers the complexity and ambiguous special cases.

Therefore we propose a simpler method, which proceeds in two steps. The first step resembles closely the first step of the marching cubes algorithm. A three-dimensional boolean array having in each direction the size of the volumetric data source minus one is allocated. Each boolean array element corresponds to a cell which is determined by an octuple of adjacent voxels from the data set. In the first step, the *marking step*, the array is traversed, and for each cell the data source is evaluated at each of its eight vertices. A cell is marked if the values indicate that the isosurface passes through it, i.e., if the eight values are not all greater than or not all less than the specified isovalue. In the second step, the *placement step*, marked cells are repeatedly extracted from the array until none are left. A glyph is placed at every marked cell and adjusted inside that cell in such a way that it is located exactly on the specified isosurface while remaining as close as possible to the center of the cell. This is achieved by subdividing the cell into eight sub-cells and querying the data source at the eight vertices of each subcell. Among all sub-cells the isosurface is determined to run through, the sub-cell with the smallest distance to the root cell's center is chosen and recursively subdivided. Finally, all cells within a user-specified world space distance from the current cell are cleared in order to create a certain amount of empty space around each glyph.

Although the images resulting from the isosurface placement method are good for simple, generic data sets, a problem arises when real-world data is used. As depicted in Figure 2(a), it is possible that glyphs seem to be placed below the surface (this effect can be seen in the region of the ear) or placed not at all because isosurfaces may be very close and run parallel to each other, so that glyphs are dis-

tributed on both surfaces. Figure 2(b) reveals that the missing and misplaced glyphs have been placed on the inside surface of the skull.

In order to provide a solution for misplaced glyphs, the isosurface placement method has been restricted to visible isosurfaces only. Visible isosurface placement assumes that isosurfaces that are not visible to the viewer can not be of any interest and hence, they should be ignored in the cell marking step. This additional condition for the marking step is implemented by casting axis-aligned rays into the volume data set, starting at each outer cell and directed into the data set. When one cell has been marked by a ray, this ray is terminated so that in the end only cells visible from the outside have been marked. With this restriction we resolve ambiguities as those shown in Figure 2.

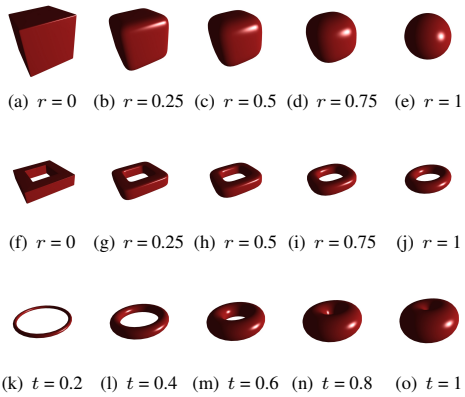
4 Glyph Visualization Setup

We have identified the following work flow for creating glyph-enhanced volume visualizations. Initially, after the user has loaded the contributing data sets the glyph modeling has to be carried out. Therefore the user selects a glyph prototype and specifies a mapping function which maps the information contained in the volume data sets to glyph properties, e.g., color or size. Finally the desired glyph placement method is chosen and the glyphs are arranged within a volume data set to represent information located at their position. Our proposed user interface assisting during the glyph setup consists of two parts, one window for glyph modeling and an overlayed graphical legend supporting evaluation of glyph-enhanced visualizations.

4.1 Glyph Shapes

Each glyph prototype is characterized by a set of properties through which information can be visualized. Depending on the desired visualization and the glyph properties, the suitability of glyph prototypes varies. Therefore the user has to choose a suitable glyph prototype for a certain task. The basic properties which are shared by all glyph prototypes are color, opacity and size, while more sophisticated glyph prototypes offer further possibilities to convey information by additional glyph properties.

The glyph prototypes described in this paper are based on the superquadric shapes presented



(a) $r = 0$ (b) $r = 0.25$ (c) $r = 0.5$ (d) $r = 0.75$ (e) $r = 1$

(f) $r = 0$ (g) $r = 0.25$ (h) $r = 0.5$ (i) $r = 0.75$ (j) $r = 1$

(k) $t = 0.2$ (l) $t = 0.4$ (m) $t = 0.6$ (n) $t = 0.8$ (o) $t = 1$

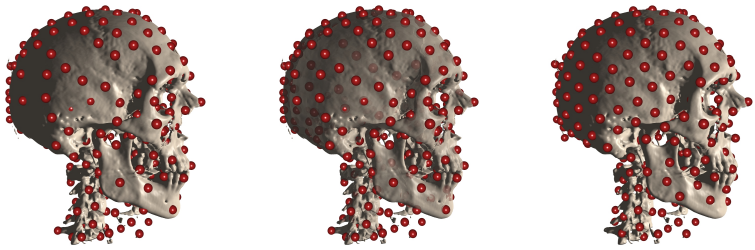
Figure 3: The supersphere and supertorus prototypes with varying parameters for roundness r (upper two rows), and the supertorus prototype with varying parameters for thickness t (lower row).

in [2]. In contrast to cuboid or ellipsoid glyphs, superquadrics have the advantage that a multitude of parameters can be mapped unambiguously. For instance, when using ellipsoids or cuboids an awkward viewing direction may result in visual ambiguity, i.e., different shapes are not distinguishable after projected onto the view plane [8]. For the sake of simplicity, some parameters of the original superquadric shapes have been made constant or combined in order to obtain intuitive shapes that can be interpreted easily: the supersphere and the supertorus. In contrast to the original superquadric ellipsoid which takes three radii as parameters, the supersphere has a fixed radius of 1. Glyph properties defined by the supersphere prototype are the scalar parameters α and β, which are derivations from the original superquadric ellipsoid α' and β'. The purpose of this simplification is a more easy setup as well as interpretation process. Thus it is more intuitive to define the roundness of the surface because their default value is 0 and the perceived change in roundness and edge sharpness resulting from adjusting α and β is linear. The conversion from supersphere α and β to the original α' and β' is given by:

$$\alpha' = 2^{\alpha} \tag{1}$$

$$\beta' = 2^{\beta} \tag{2}$$

In the original equations, the base shape (plain



(a) (b) (c)

Figure 2: Glyphs are distributed on front- and back-facing surfaces (a), semi-transparent isosurfaces reveal the hidden glyphs (b). Application of visible isosurface placement (c).

sphere or torus) results from $\alpha' = \beta' = 1$. Less roundness can be achieved with $0 > \alpha' < 1$ and $0 > \beta' < 1$, sharper edges can be achieved with $\alpha' > 1$ and $\beta' > 1$. With the conversion applied, the base shape results from $\alpha = \beta = 0$, less roundness is achieved with $\alpha < 0$ and $\beta < 0$, sharper edges are achieved with $\alpha > 0$ and $\beta > 0$, and the change in sharpness or roundness is linear and thus more suitable for user interaction. However, these prototypes are still rather complex because the interpretation of the α and β values is not very intuitive. For example, two supertori perceived as 'not round' can have different parameters. In order to resolve such ambiguities, simplified versions of both supersphere and supertorus prototypes have been created in which the α and β parameters are combined into a parameter r that represents the roundness of the object expressed by a value in the range from 0 (angular) to 1 (round). The conversion from r to α and β is defined by the equation

$$\alpha = \beta = 5 \cdot r - 5. \tag{3}$$

This results in a value between 0 and 5 serving as good values for round resp. non-round superquadrics (see Figure 3).

The supertorus prototype is defined by an additional scalar t that represents the thickness of the torus. In comparison to the original superquadric shapes, the supersphere and supertorus prototypes are easier to interpret because an estimation of the roundness r seems rather trivial when compared to an estimation of the parameters α and β used in the original equations. Even for users with no mathematical background it should be easy to interpret the roundness and thickness values.

The complex glyph modeling task can be done with the help of the user interface depicted in Figure 5, whereas the interface elements are arranged according to the work flow of the glyph modeling process. After all data sets have been loaded, a glyph prototype has to be chosen from a list of available prototypes. The currently chosen glyph prototype is shown in the prototype window (see upper overlay in Figure 5). To save screen space, glyph properties of interest can be expanded or hidden, e.g., in Figure 5, the roundness and thickness properties are currently expanded.

4.2 Glyph Property Mapping

For specifying the property mapping function, which maps input values from scalar data sources to scalar glyph property values, the user can control a set of mapping keys. Each mapping key defines a pair of source and destination values. The specification of the mapping function is similar to specifying a transfer function. An example is shown in Figure 4, where two keys are used to define the mapping behavior. In order to provide the possibility to put emphasis on certain glyphs, steps can be introduced in a mapping function by splitting keys and defining different destination values for points to the left and to the right of the key (see Figure 4). If the mapping function is evaluated for a source value other than those defined by the keys, linear interpolation between the keys to the left and the right of the queried location is used. If the mapping function is queried for locations that do not lie between two mapping keys, the value of the nearest mapping key is returned.

In the mapping function window (see lower over-

lay in Figure 5), a mapping function can be defined by modifying mapping keys within the canvas spanning the complete range of possible input and output values. In addition to the mapping function, a scalar data source must be chosen that should be linked to the glyph property and that the mapping function should be applied to. The mapping canvas supports the user by providing the possibility to display the histogram of the current data source. After the glyph properties have been adjusted, a glyph placement method is chosen along with a data source the glyph placement is applied to. When the glyph prototype and the placement method have been set up, the glyph modeling is finished and the glyphs can be rendered.

4.3 Glyph Legend

To allow the interpretation of a glyph-enhanced visualization, it is necessary that the glyph mapping process is transparent, such that the user can derive the source information from a glyph's representation. Therefore we propose a glyph legend which represents the mapping function graphically. Since only those glyph properties that are dependent on data sources actually convey information, only these properties are of interest within a glyph legend. Thus a legend can be constructed from a number of rows each depicting how a property of interest relates to its underlying data source. If multiple properties depend on the same data source, these properties can be combined into a single row because for each of these properties, the data source is queried at exactly the same location and thus returns exactly the same value.

For generating the glyph legend, all data sources that are linked through a mapping function are taken into account. Thus, for each data source involved, a row is added to the legend that indicates how the

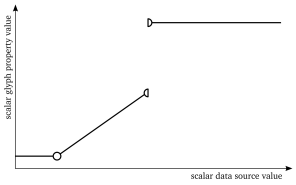


Figure 4: An example mapping function with two mapping keys. The second key is split.

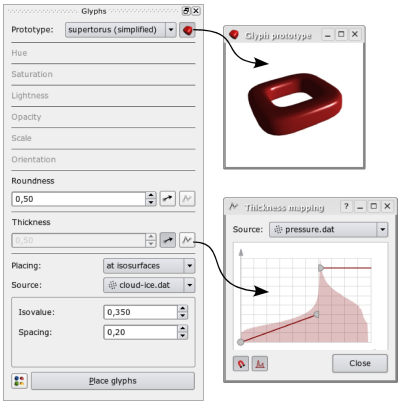


Figure 5: The glyph modeling user interface with the glyph prototype window and the graphical glyph mapping function editor displaying the histogram of the current scalar data source.

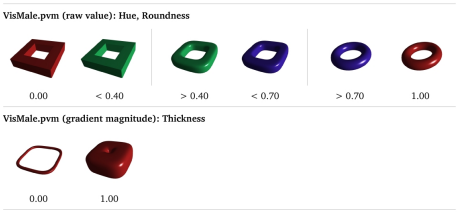


Figure 6: In the minimal glyph legend only key glyphs are displayed.

data source influences the visual appearance of the glyph. All glyph properties that are connected to the current row's data source through the mapping function are determined and printed in the row caption to clarify which properties are conveyed by the glyph icons. Furthermore, for all mapping keys an icon of the glyph prototype is added to the row, showing the glyph prototype with the resulting representation inherently defined by the mapping function. In the cases of split mapping keys, two glyph icons representing both left and right mapping key destination values are rendered and separated by a vertical line to emphasize the fact that a split mapping key is displayed. A legend created in such a way can be seen in Figure 6.

A more comprehensive legend can be created by inserting supplementary keys in cases where the differences between two consecutive mapping keys'

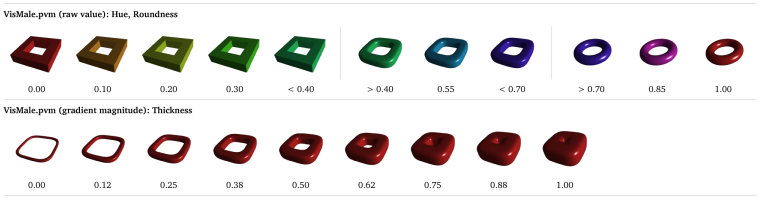


Figure 7: In the the extended glyph legend, glyphs are inserted at certain locations in order to bridge large differences.

destination values are large. A possible approach to extend the legend is to find pairs of consecutive mapping keys for which the change of at least one property exceeds a certain threshold and to add the appropriate glyph representations to the legend. Such an extended legend is shown in Figure 7.

5 Rendering

To allow simultaneous display of volumetric data and polygonal glyphs, we have extended the GPU-based ray casting technique [9], which is solely applicable to render volumetric data sets. The integration of opaque glyphs is quite easy. In order to combine opaque geometry with GPU-based ray casting, it is sufficient to modify the end points of each ray by drawing the additional polygons on top of the proxy geometry's back faces (see Figure 8). Thus each ray terminates as soon as it hits a polygon. Glyphs can be integrated by initially rendering them to the background and afterwards blending the ray-casting results using the modified exit parameters shown in Figure 8(c).

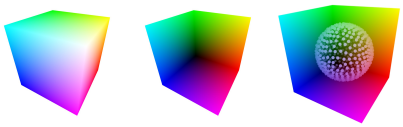


Figure 8: The entry and exit parameters for GPU-based volume ray-casting (left,middle). The exit parameters are modified in order to integrate opaque glyphs positioned on a sphere (right).

6 Application Examples

This section discusses some examples of the described glyph-enhanced volume visualization applied to the NCAT PET/CT data set [16] as well as the hurricane Isabel data set.

In contrast to Computed Tomography which reveals the inner structure of a subject, Positron Emission Tomography is used to get information about metabolic activity inside a living subject. Because PET images are of much lower resolution than CT data sets, it is difficult to tell where points in the image are located within the subject because the morphological context as provided by Computed Tomography is missing. To avoid such problems, a combined CT/PET scan can be performed. The result of such a scan are a PET data set and a CT data set. The latter provides contextual information and is registered with the PET data.

A method of displaying CT and PET data simultaneously is PET/CT fusion imaging, in which a color gradient is applied to the PET image and subsequently both images are combined into a single fusion image. However, this fusion makes quantization difficult, since the PET color mapping is influenced by the color of the blended CT data.

The visualizations presented in Figure 9(a) and in Figure 9(b) are alternatives to common PET/CT fusion imaging. A volume rendering of the CT data provides the context for the PET data which is conveyed by glyphs. In order to demonstrate that in glyph-enhanced volume visualization the glyphs are completely independent of the type of volume rendering applied to the context data, both direct volume rendering (see Figure 9(a)) and X-ray simulation (see Figure 9(b)) are shown. In both examples, different attempts to direct the viewer's attention to important regions of the volume are demonstrated. In Figure 9(a), the supertorus thickness is

used to highlight the lesion on the left ventricle evident from the PET data. The thickness is mapped in a way that the user's attention is directed towards regions of unusually low metabolic activity, which are shown by using thicker supertori. Furthermore, the glyph orientation is adjusted to match the normal vectors of the heart surface, resulting in minimal occlusion and the effect that the glyphs look like glued to the surface. In Figure 9(b) regions of high PET activity are visualized in an unobtrusive manner in order to highlight regions of low PET activity.

The previous example could have been visualized to a certain extent by using non-glyph techniques. For instance a simple color coding could have been used in order to visualize the PET activity. However, in order to make this visualization quantifiable, the shading should not influence the shown color. Therefore the data set would have been rendered without shading, which would destroy the shape through shading cue. Another possibility would be using stippling. The CT data set could be rendered using regular phong shading and the PET intensity could be depicted by superimposing a stippling pattern, which intensity is altered based on the PET intensity. While this approach allows an integration of the PET information into a CT visualization without loosing the shape from shading, it is limited to the usage of two modalities.

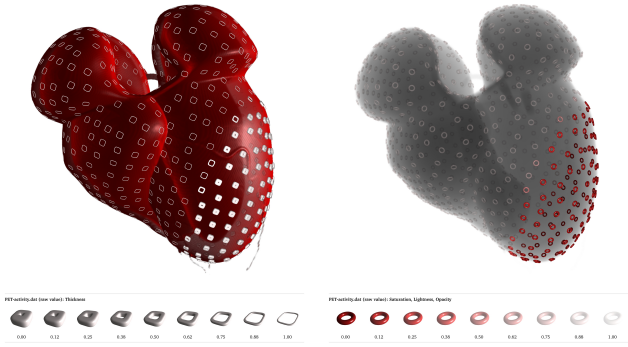
The hurricane Isabel data used for the visualization presented in Figure 10 consists of many modalities including cloud water, cloud ice, graupel, rain, snow, vapor, pressure, temperature and wind direction. The goal is to visualize some of these variables simultaneously in order to allow visual exploration of the value distribution and to identify correlations. Two of these variables plus the sum of three further variables are depicted by the glyphs in the image. Temperature is depicted by hue in a range from blue (cold) to red (warm). Air pressure is depicted by the supertorus thickness, a combination that allows for intuitive interpretation because of the analogy to a bicycle tire. The amounts of graupel, rain and snow are accumulated to the total amount of precipitation and depicted by the roundness of the glyphs.

Similar to applying color mapping, the glyph-based visualization allows to get a quick overview of the value distribution. For instance it can be seen that the temperature is highest within the eye of the storm decreasing with increasing distance to it. Additionally we can see that the pressure is very low inside the eye and increasing with increasing distance to the eye. Since we are using supertorus glyphs, besides the thickness we can also use the roundness to represent a variable. With the precipitation mapped to roundness, it can be seen that it is highest in the vicinity of the eye of the storm and it becomes clear that it is not uniformly decreasing with the distance to the eye. Moreover the precipitation is highest behind the eye of the storm (the direction of movement is towards the upper left).

Although this example shows the potential of glyph-based visualizations it also demonstrates the limits when applying it to multimodal data having many variables. The remaining glyph properties that could be used to depict more scalar variables are saturation, lightness, opacity and scale. Saturation cannot be used to convey detailed information if hue is used to depict other data at the same time, because when the saturation value is approaching 0, correct interpretation of the hue value becomes increasingly difficult. When lightness is used, the interpretation of the hue value becomes difficult when the lightness value approaches 0 or 1. These effects lead to the conclusion that in most cases, only one data source can be depicted by the color properties, although simultaneous display of three completely unrelated variables seems possible. Furthermore, size is potentially inappropriate for conveying data as well because the data-determined size of a glyph conflicts with its size due to perspective projection. Additionally, shape perception becomes very difficult when glyphs are small.

Besides the glyph property mapping, the surface used for glyph orientation has to be defined. When having modalities which provide contextual information, e.g. a CT scan, this definition is quite easy. However, in the general case this task needs more attention. Sometimes it might even be desirable to have a dynamic surface, i.e., a 2D slice moving through the data set.

In both presented examples a non-continuous mapping can be used to further emphasize differences, e.g., when reaching a certain threshold. In the first example this can be used to show only glyphs representing abnormal PET intensities, while the one representing normal ones can be omitted by assigning transparency or minimal size.



(a) PET/CT DVR. (b) PET/CT x-ray.

Figure 9: Cardiac wall motion and activity, derived from PET/CT data, depicted with different glyph setups.

7 Conclusion and Future Work

In this paper we have presented concepts for easy setup and interpretation of glyph-based visualizations. We have introduced modifications of superquadric glyphs and have discussed a surface-based glyph placement strategy which binds glyphs to a surface of interest. To make a step towards a quantifiable glyph-based visualization, we have introduced a visual glyph legend.

In the future more improvements for generating glyph configurations could be considered. An important outcome of this development could be design guidelines specifying which glyph properties are best suited for depicting certain information. In some cases also non-symmetric glyphs could be helpful, e.g., for visualizing velocity or a direction of movement. In addition, during glyph modeling the user should be notified about glyph properties potentially shadowing each other.

Acknowledgements

This work was partly supported by grants from the Deutsche Forschungsgemeinschaft (DFG), SFB 656 MoBil Münster, Germany (project Z1).

References

[1] Hiroshi Akiba and Kwan-Liu Ma. A tri-space visualization interface for analyzing time-varying multivariate volume data. In *Proceedings of Eurographics/IEEE VGTC Symposium on Visualization*, May 2007.

[2] Alan H. Barr. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications*, 1(1):11–23, January 1981.

[3] Jorik Blaas, Charl P. Botha, and Frits H. Post. Interactive visualization of multi-field medical data using linked physical and feature-space views. In *Eurovis 2007, proceedings of the joint Eurographics / IEEE VGTC Symposium on Visualization*, pages 123–130, 2007.

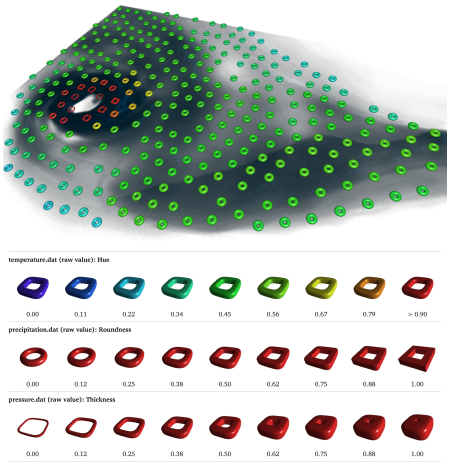
[4] Helmut Doleisch, Philipp Muigg, and Helwig Hauser. Interactive visual analysis of hurricane isabel with simvis. In *IEEE Visualization Contest*, 2004.

[5] D. S. Ebert, R. M. Rohrer, C. D. Shaw, P. Panda, J. M. Kukla, and D. A. Roberts. Procedural shape generation for multi-dimensional data visualization. In *Data Visualization '99*, pages 3–12. Springer-Verlag Wien, 1999.

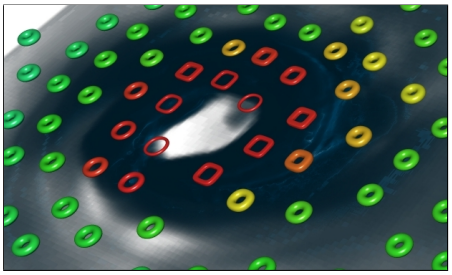
[6] T.J. Jankun-Kelly and Ketan Mehta. Superellipsoid-based, real symmetric traceless tensor glyphs motivated by nematic liquid crystal alignment visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1197–1204, 2006.

[7] Gordon Kindlmann and Carl-Fredrik Westin. Diffusion tensor visualization with glyph packing. *IEEE Visualization 2006)*, 12(5):1329–1135, September-October 2006.

[8] Gordon L. Kindlmann. Superquadric tensor



(a) Glyph visualization of the hurricane Isabel data set in conjunction with a legend.



(b) A detailed view of the eye of the hurricane.

Figure 10: Glyphs are used to visualize temperature, precipitation and air pressure. In this example the benefits of the supertorus glyph become present, since the three modalities are mapped to the color, thickness and roundness parameters.

glyphs. In *Joint Eurographics - IEEE TCVG Symposium on Visualization*, pages 147–154, 2004.

[9] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *IEEE Visualization 2003*, pages 38–43, 2003.

[10] David H. Laidlaw, Eric T. Ahrens, David Kremers, Matthew J. Avalos, Russell E. Jacobs, and Carol Readhead. Visualizing diffusion tensor images of the mouse spinal cord. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 127–134. IEEE Computer Society Press, 1998.

[11] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA, 1987. ACM Press.

[12] A. M. Nayak, K. Lindquist, D. Kilb, R. Newman, F. Vernon, J. Leigh, A. Johnson, and L. Renambot. Using 3D Glyph Visualization to Explore Real-time Seismic Data on Immersive and High-resolution Display Systems. *AGU Fall Meeting Abstracts*, pages C1208+, December 2003.

[13] Lydia Paasche, Steffen Oeltze, Frank Grothues, Anja Hennemuth, Caroline Kühnel, and Bernhard Preim. Integrierte visualisierung kardialer mr-daten zur beurteilung von funktion, perfusion und vitalität des myokards. In *Bildverarbeitung fr die Medizin*, pages 212–216, 2007.

[14] G. Reina and T. Ertl. Hardware-Accelerated Glyphs for Mono- and Dipoles in Molecular Dynamics Visualization. In K. W. Brodlie and D. J. Duke and K. I. Joy, editor, *Joint Eurographics - IEEE TCVG Symposium on Visualization*, pages 177–182, 2005.

[15] Patrick Coleman Saunders, Victoria Interrante, and S.C. Garrick. Pointillist and glyph-based visualization of nanoparticles in formation. In *Joint Eurographics - IEEE TCVG Symposium on Visualization*, pages 169–176, 2005.

[16] W.P. Segars, B.M.W. Tsui, A.J. Da Silva, and L. Shao. Ct-pet image fusion using the 4d ncat phantom with the purpose of attenuation correction. In *IEEE Nuclear Science Symposium Conference Record 2002*, pages 1775–1779, 2003.

[17] A Sigfridsson, T Ebbers, Heiberg, and L Wigström. Tensor field visualization using adaptive filtering of noise fields combined with glyph rendering. In *Proceedings IEEE Visualization 2002*, pages 371–378, Boston, Massachusetts, 2002.

[18] Matthew O. Ward. A taxonomy of glyph placement strategies for multidimensional data visualization. *Information Visualization*, 1(3/4):194–210, 2002.

Illustrative Rendering of Seismic Data

Daniel Patel[1,2], Christopher Giertsen[1], John Thurmond[3], Eduard Gröller[4,2]

Christian Michelsen Research, Bergen, Norway[1]
University of Bergen, Bergen, Norway[2]
Norsk Hydro, Bergen, Norway[3]
Vienna University of Technology, Austria[4]
Email: daniel@cmr.no, chrisgie@cmr.no
john.thurmond@hydro.com, groeller@cg.tuwien.ac.at

Abstract

In our work we present techniques for illustrative rendering of interpreted seismic volume data by adopting elements from geology book illustrations. We also introduce combined visualization techniques of interpreted and uninterpreted data for validation, comparison and interdisciplinary communication reasons. We introduce the concept of smooth transitions between these two semantical levels. To achieve this we present transfer functions that map seismic volume attributes to 2D textures that flow according to a deformation volume describing the buckling and discontinuities of the layers of the seismic data.

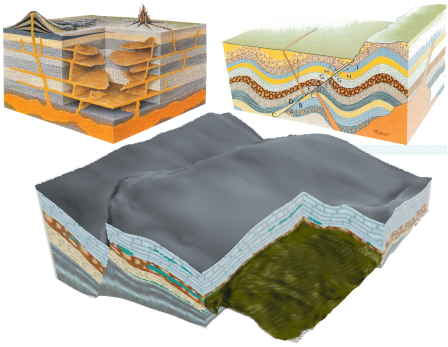


Figure 1: Geological and rendered illustrations. Top left: A cutout with extruding features. Top right: Textured layers with a fault discontinuity in the middle. Pictures are from Grotzinger et al. [6]. Bottom: Illustration rendered with our techniques.

1 Introduction

In geology faults and horizons are central subsurface structures. The earth has a layer-like structure and horizons are defined as the surfaces that separate one layer from another. Tension in the crust of the earth deforms the layers over time and creates cracks. These so called faults are more or less vertical discontinuities of the layers.

Geological illustrations in text books try to convey faults, horizons and other structures of the earth by using different artistic techniques as seen in the top of Figure 1. The illustrator draws a cubical subsection of the earth defining the area of interest. The horizons and faults are represented by textures flowing inside the layers that are discontinuous across faults. The textures are drawn on the exterior side faces of the cubical subsection whose extent we hereby refer to as the roaming box. Axis-aligned cutouts with textures on the interior side faces are used to show features inside the cubical subsection. The cutouts sometimes contain extruding 3D features. Our illustrative renderings adopt all these techniques as seen in the bottom of Figure 1.

Figure 2 presents the flow from data acquisition to data visualization. The faults, horizons and other subsurface structures are discovered by geoscientists interpreting volumetric descriptions of the subsurface. These volumetric descriptions are typically obtained in geophysical surveys by processing the reflections of waves sent into the surface. The volume storing the reflection data is called the reflection volume. In a time consuming process the faults and horizons are manually found from the reflection volume and stored as surfaces. Several seismic attributes can be computed from the reflection data such as acoustic impedance (Ai) and the ratio be-

tween the pressure and shear wave (Vp/Vs). We will refer to these volumes as seismic attributes.

Coming up with a good visualization of interpreted data can be difficult, therefore we propose to use illustrative techniques. Illustrations are being used when there are certain high level aspects of a complex image, such as interpreted information, that need to be communicated in a simple way. Rendering of interpreted seismic data as illustrations has several advantages. It simplifies the visualization and emphasizes the elements of interest in order to disseminate gained knowledge from the interpretation process. Making a good illustration for scientific purposes takes time. Being able to render geological illustrations is advantageous both for quickly creating static images to illustrate geological books and for interactive oil exploration when interpreted survey data needs to be communicated as part of decision making.

Interpreting seismic data is a time consuming manual process and it is important to verify the interpretation with the underlying data source. By combining visualizations of interpreted and uninterpreted data it is possible to perform comparisons and look for deviations. This is another goal in our work. We propose to visualize the interpreted data as geological illustrations and to visualize uninterpreted data using color coded cutting planes and regular volume rendering. We present how to combine these two representations. The user can control the balance between these two visualization styles to fit his or her needs. For interdisciplinary communication reasons visualizations can be made to have the right balance between interpreted data which contains semantical information understandable by lay men to uninterpreted data which contains the information-rich underlying data material understandable by domain experts.

To our knowledge the concept of creating automatic illustrations of seismic data has not been thoroughly explored before, neither in the geophysics nor in the visualization research community. We also believe this applies to combined rendering of interpreted and uninterpreted seismic data.

We start with related work in Chapter 2. After an overview in Chapter 3 we describe the calculation of the texture flow in chapter 4. In chapter 5 we use the calculated flow in combination with texture transfer functions to texturize the cutting planes on the side faces of the cubical subsection and on the cutout. In chapter 6 we describe volume rendering for displaying the cutout and the surroundings and we specify how this is integrated with the rendering of textures during ray casting. Finally future work and conclusions are presented in chapter 7. The bottom half of Figure 2 shows a high level overview of the paper.

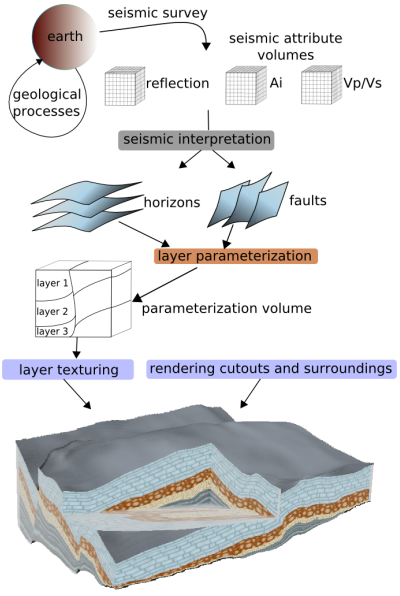


Figure 2: Overview of the process from data collection to visualization. The paper covers the lower three colored rectangles in chapter 4, 5 and 6.

2 Related work

We first review work dealing with illustrative techniques and then review work in the field of seismic visualization. Illustrative rendering is a non-photo realistic visualization technique using the advantages of conveying information through illustrations. In recent years several illustrative rendering techniques, mainly in the domain of anatomical visualization, but none in the domain of seismic visualization, have been proposed. Some of these techniques deal with applying textures from reference images.

Owada et al. [11] present an interactive system for texturing arbitrary cuts through polygonal objects. The user defines the texture flow by spec-

ifying a flow field and a distance field on the cut which is used in the texture synthesis to create a texture on the cut that follows the flow. Their method is general and therefore requires user interaction to specify the deformation and the texture. We calculate a parameterization up front so texturing can be achieved quickly and without the need for texture synthesis. In our approach many of the parameters defining the visualization are known prior to rendering, therefore less user specification is required

There are also several papers dealing with textures in medical volume rendering. Lu and Ebert [9] generate illustrative renderings of color and scalar 3D volumes by applying textures sampled from illustrations and photographs. 3D textures are created by combining color information from the illustrations with 3D volume data of a corresponding area. Finally the 3D textures are made tileable with Wang Cubes. With segmented volume data they apply the corresponding 3D textures on each segment. With unsegmented scalar data they use a transfer function to map scalar voxel values to the 3D textures in a similar way to what we propose. They do not deal with multi-attribute texture transfer functions and with deforming the textures to follow the underlying flow of the data as we do. In addition their method of calculating the textures is tailored to handle 3D textures whereas we use 2D textures.

Dong and Clapworthy [4] present a technique that achieves 3D texture synthesis following the texture orientation of 3D muscle data. Their algorithm has two steps. First they determine the texture orientation by looking at the gradient data of the volume and by using a direction limited Hough transform. Second they perform a volumetric texture synthesis based on the orientation data. In our work, instead of considering the volume for evaluating texture flow, we consider the geometric layers. In addition the texture synthesis of Dong and Clapworthy has the drawback of not working on textures with large interior variation as textures in geologic illustrations commonly have.

Wang and Mueller [14] use 3D texture synthesis to achieve sub-resolution zooming into volumetric data. With 2D images of several zoom levels of a tissue, they synthesize 3D volume textures for each level and use constrained texture synthesis during zooming to blend smoothly between the levels. They address the issue of sub-resolution details but do not consider texture flow.

In the domain of seismic processing GeoChron [10] is a formal model for parameterizing the layers defined by faults and horizons. The GeoChron model allows for several inputs which act as constraints to the parameterization. It considers the physical processes behind the deformation whereas our parameterization is fully defined by the fault and the horizon data. We believe that for illustration purposes a less physically accurate and computationally less intensive algorithm requiring a minimal amount of input and expertise such as our parameterization is preferable. However since our visualization algorithm is decoupled from the parameterization, it would also accept a GeoChron parameterization.

Cutouts on seismic data and interaction in VR was presented in the work by Ropinski et al. [13] where they use volume rendering with two transfer functions. One transfer function is used for the volume inside a user defined box and one transfer function is used for the volume outside. We incorporate and extend this concept in our work. Several papers on visualizing seismic data exist. Some deal with automatic horizon extraction [2] or fault extraction [2, 7], others deal with handling large volumes [2, 12], but none deal with illustrative rendering. Somewhat related is the dissertation of Frank [5] where the GeoChron parameterization [10] is used as a lookup to unfold and flatten a seismic data volume. In commercial systems seismic attribute data is presented with volume rendering and geometric surfaces are used to present horizons and faults.

3 Overview of the rendering process

Our methods render interactively the illustrative features found in geological images. Texturing is achieved by rendering deformed 2D textures on the exterior side faces of the roaming box and on the interior side faces of the cutout. For each layer the user assigns a texture and the texture's horizontal and vertical repeat rate. To also represent seismic attributes the user can assign textures and opacities to intervals of the seismic attribute values. These attribute textures are then blended and laid over the layer textures. We represent extruding features in the cutouts by volume rendering using a color transfer function together with a depth based opacity transfer function. The opacity is a function of the

layer depth and the transparency can be set to restrict volume rendering to certain layers or to certain depths within a layer. Also in the surrounding area outside the cutout we perform volume rendering that can be restricted to certain layers or to certain depths within a layer. In the surrounding area the voxel colors are equal to the average color of the 2D texture used in the layer the voxel is in. This gives a consistent coloring with the cutting plane textures as can be seen in the bottom of Figure 2 and the top of Figure 10. There we render opaquely the top and bottom horizon with the average color of the 2D texture in the horizons. To visualize the uninterpreted seismic data we render the cutting planes and the surrounding volume with the color transfer function used for the cutout volume rendering. The user can smoothly change between the uninterpreted data rendering and the interpreted illustrative textured rendering by changing the blending factor. An overview of the texturing process can be seen in Figure 6 while the lower part of Figure 2 shows how the texturing fits into the 3D visualization. In the next three chapters the details of the visualization process described above is presented.

4 Layer parameterization

We parameterize the volume to render 2D planar textures following the flow of the layers and to achieve depth controlled volume rendering. Using the coordinate system shown in Figure 3d we define horizons as non-intersecting surfaces stacked in the z-direction of the type $z = H(x, y)$ and faults as non intersecting surfaces stacked in the x-direction of the type $x = F(y, z)$ or in the y-direction of the type $y = F(x, z)$. The faults, horizons and the side faces of the roaming box divide the volume into subvolumes which we will refer to as slabs. Conversely, each of these slabs is horizontally confined by what we will refer to as the upper and lower horizon and are laterally confined by fault surfaces and the side faces of the roaming box (see Figure 3a).

There exists no unique solution to parameterize a volume. We have designed the parameterization so that it represents the slabs in a flattened version where horizons and the layer between are planar. Figure 3d shows the parameterization coordinate system (u, v, w) embedded in the world coordinate system (x, y, z).

The parameterization consists of several steps. First the upper and lower horizon of the slab is extended by extrapolation (see dotted lines in Figure 3a). We do this extension to get a correct volumetric parameterization close to the vertical slab borders. Then the lower horizon surface is parameterized and the depth parameter w is calculated for the volume, (see curves in Figure 3b). Finally the 2D parameterization of the lower horizon is projected into the volume along the gradient field of the w parameter (blue curves in Figure 3c), resulting in a 3D parameterized slab.

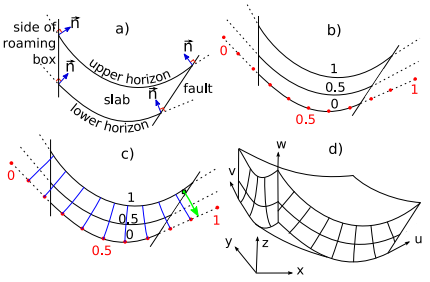


Figure 3: A 2D version of the steps needed for parameterizing a slab is shown in a-c. The world and the parameter coordinate system is shown in d.

Let $w_p = \{x, y, z\} \in \mathbf{R}^3$ be a point in W(orld space), and $p_p = \{u, v, w\} \in \mathbf{R}^3$ be the corresponding point in P(arameter space). We represent the mapping from W to P as $\mathbf{P} : W \rightarrow P$ where

$$p_p = \mathbf{P}(w_p) = \{P_u(w_p), P_v(w_p), P_w(w_p)\}$$

Let $min_{upper}(w_p)$ and $min_{lower}(w_p)$ express the Euclidean distance from w_p to the closest point on the upper and lower horizon respectively. The w parameter, or layer depth, is defined as:

$$P_w(w_p) = \frac{min_{lower}(w_p)}{min_{lower}(w_p) + min_{upper}(w_p)}$$

min_{upper} and min_{lower} are found by discretizing the upper/lower horizon into a point cloud. For discretizing we linearly subsample the horizon grid four times, and store the points in a kd-tree for efficient searching. Note that P_w does not express the distance to the closest surface as found by a distance transform, but the relative distance between the upper and lower horizon, it maps the lower horizon to 0, the upper horizon to 1 and is linear in between. In effect it flattens the layer and defines a local depth measure on it. See curves in Figure 3b.

We now have a w parameterization of the slab. The (u, v) values in the slab are found by projecting the (u, v) values from the parameterized lower horizon, which is described in 4.2. Projections into the volume is done along the streamlines seen as blue curves in Figure 3c which are defined by the vector field ∇P_w. ∇P_w is calculated using central differences. For each voxel we trace along the streamline in the opposite gradient direction toward the lower horizon, seen as a green arrow in Figure 3c. We assign to the voxel the (u, v) value of the intersected point on the lower horizon.

Assigning (u, v) values for the voxels inside the slab that are close to the vertical slab borders might result in streamlines leaving the slab and entering an area where P_w has not been calculated. See green arrow in Figure 3c. We have extended the horizons with the method described in 4.1 prior to the w parameterization and prior to the (u, v) parameterization of the bottom horizon. By doing this we have gradient data outside the slab as well as a parameterized surface outside the lower horizon which makes it possible to calculate streamlines leaving the slab. The parameterization procedure ensures that the (u, v) parameterization is orthogonal to the w parameterization which in turn will result in angle preservation in the 2D textures. The parameterization works well for surfaces of low curvature and without folds as seen in this application but would require some extension for handling other types of surfaces.

The parameterization is done on each slab and is stored in an RGB volume consisting of the (u, v, w) parameters. The parameters of each slab are all in the $[0, 1]$ range. To encode segmentation information for each slab we scale and shift the w and u parameter values. Each layer's w values are scaled and shifted such that values go from 0 at the lower horizon in the bottom layer to 1 at the upper horizon in the top layer with each layer having equally sized intervals. Similarly, the u values are scaled and shifted on each side of the fault. At the left side of the fault in Figure 4 the u values are between 0 and 0.5 and on the right side they are between 0.5 and 1. The segmentation information is used for having different textures in different layers and possibly on different sides of faults. The parameterization is not meant to be geologically accurate but to act as a tool for 2D texturing and depth dependent volume rendering. The goal is to achieve

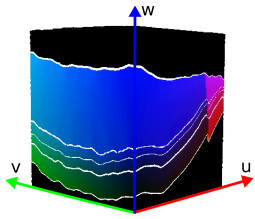


Figure 4: The parameterization RGB volume. White lines have been added on horizons. There is a color change across the fault due to shifting and scaling of the u parameter.

images with illustrative quality. The two following sections will describe the horizon extrapolation and the bottom horizon parameterization which was assumed to be done prior to the layer parameterization but were not explained in detail.

4.1 Horizon extrapolation

For parameterization of areas close to the vertical slab borders we need surface information beyond the horizon borders as described earlier. To achieve this we carry out a simple surface extrapolation in all directions. First we extrapolate the surface in positive and negative x direction by considering the surface as a collection of curves parallel to the x axis and extending the endpoints of the curves in tangential direction. See normals and dotted lines in Figure 3a. We then do the same procedure on the resulting surface in positive and negative y directions. Finally we crop the horizons so that their projections to the xy plane are rectangular and so that sufficient data exists beyond their original borders. On our data we ended up with a heuristic extension of 20 percent of the horizon length in each direction to correctly parameterize the areas close to the vertical slab borders.

4.2 Surface parameterization

For the (u, v) parameterization of the lower horizon surface we calculate a parameterization that locally minimizes the area distortion. Red dots on Figure 3b show the corresponding 1D version. The parameterization is created with the CGAL library [1] using the discrete authalic parameterization [3]. The parameterization defines (u, v) values for each

vertex on the surface. The parameterization is constrained by giving initial values to the surface borders whose projection to the xy plane forms a rectangle due to the surface extrapolation. For the initial values we clockwise assign the border vertices with values (0,0) (0,1), (1,0) and (1,1) and interpolate the values along each edge with equidistant spacing. We now have a (u, v) parameterization of the lower horizon and a w parameterization of the slab.

4.3 Interpolation problem around horizons and faults

The parameterization volume is a discrete specification of our parameterization function. With trilinear sampling we get a smoother function which however leads to invalid interpolation in cells on slab boundaries where the eight cell corners are in different slabs. We calculate new values for the invalid corners by extrapolating from valid neighbor values. Then we perform the trilinear interpolation for the new corner values on the GPU. The extrapolation will try to assign a new value for invalid corners by considering the corner's two neighbors in positive x direction. If both are valid then their values are linearly extrapolated and assigned to the corner. If not, then the search continues in negative x direction and then similarly in y and z direction. In rare occasions the procedure fails to extrapolate all the invalid corner values and an erroneous interpolation is performed. The resulting artifacts will be noticeably only at the slab borders and will be of the same size as a voxel in the parameterization volume. The procedure improves the quality of the renderings as can be seen in Figure 5.

4.4 2D texture mapping on axis-aligned cutting planes

Our parameterization volume now makes it possible to apply an undeformed 3D texture stored in parameter space and deform it into world space for texturing voxels in our layers. However this would require to first generate 3D textures which is a research topic in its own as investigated by Lu and Ebert [9]. Since we are going to texture axis-aligned cutting planes as done in geological illustrations we can reduce the problem to a 2D texturing problem. This has several advantages. 2D tileable textures are easy to generate, take little space, and can be sampled from illustrations directly. With our

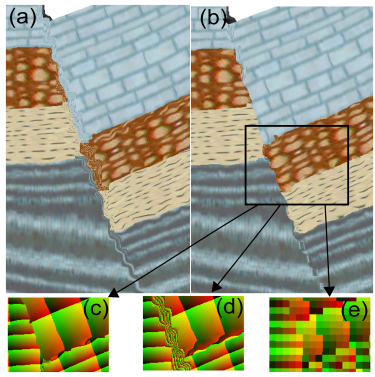


Figure 5: Fault and interpolation problems. In a) linear interpolation is used. In b) extrapolation as described in 4.3 improves the quality. In c) we see the parameterization of the zoomed-in rectangle with extrapolation as opposed to without in (d). In (e) we see the parameterization with nearest neighbor interpolation showing the resolution of the parameterization.

method the 2D textures maintain coherency when moving the cutting planes and we have better control over the repetitive appearance than for 3D textures. However we need to define a transformation from 3D parameter space (u, v, w) to 2D parameter space (u', v'). The mapping is straightforward. For texturing in the xz plane we use the (u, w) values, for texturing in the yz plane we use (v, w) and for texturing in the xy plane we use (u, v) values. The mapping conserves the angle preservation property of the 3D parameterization.

5 Layer texturing

This chapter presents three transfer functions that are being used together to texture and color cutting planes. First we present the layer texture transfer function, abbreviated as layer TTF. It assigns textures to each layer. Then we present the scalar texture transfer function, abbreviated as scalar TTF. It assigns textures and opacities to regions having seismic attribute values in certain ranges. The resulting scalar TTF texture for a cutting plane is blended according to its opacities with the layer TTF texture using the over operator. The combined results are cutting planes with deformed textures similar to the ones in geology illustrations.

Finally we present the concept of smoothly moving from illustratively rendered cutting planes to color coded cutting planes. Here seismic attribute values are mapped to colors using a color transfer function, abbreviated as color TF. See Figure 6 for an overview and Figure 7 for a texture example. Visualizing horizon, fault, deformation and seismic

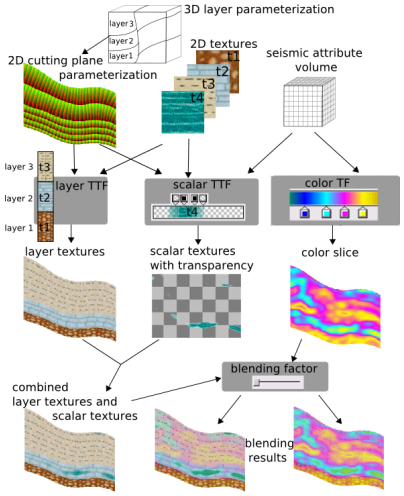


Figure 6: Overview of how textures are combined. Layer TTF, scalar TTF and color TFs are explained in 5.1, 5.2 and 5.3 respectively.

attribute information through textures has several advantages. By looking at Figure 7 one sees that textures communicate the id, orientation and compression of layers on a local scale. An example is given with the two small patches in the black circles in Figure 7. The texture of a patch reveals its layer id. The angles in the texture express the orientation of the layer in that area. Compression is presented through the vertical texture repeats in a layer. Since the vertical texture repeats are constant throughout the layer (there are always 16.5 bricks stacked in the height in layer 1 in Figure 7), compression will be high where the layer is thin and low where the layer is thick. It is possible to see that the texture patch in the left circle is slightly more compressed than the texture patch in the right circle of Figure 7. Finally, by letting both the horizontal and vertical texture repeat rate be a function of an underlying scalar value, scalar data can be presented in the texture as seen Figure 8. All this information is communicated with textures even on zoom scales where no horizons are visible, and also when zooming beyond the resolution of the seismic attribute volume. On such sub-resolution scales color transfer functions yield blocky or monotonous colored results whereas textures give aesthetically pleasing results. One can imagine zooming past the attribute volume resolution when inspecting overlaid high resolution data, such as bore well core data.

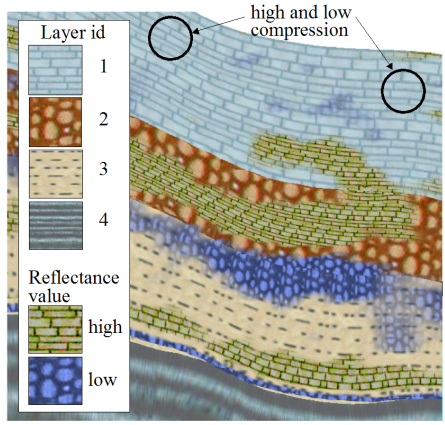


Figure 7: Combination of layer TTF and scalar TTF for the reflectance volume. The brown brick texture shows areas of high scalar values and the violet texture shows areas of low scalar values.

5.1 Layer texture transfer function (TTF)

To texture the cutting planes we use 2D RGB textures with wrap-around and bilinear filtering. They are taken from geological illustrations, and mapped on the layers. We use a layer TTF that maps from a voxel's w parameter value to a texture id, a horizontal and vertical texture repeat rate and an opacity value: $layer_{ttf}(w) = \{layerId, h_{rep}, v_{rep}, \alpha\}$. The opacity value is only used later in the cutout volume rendering. Since w varies in distinct intervals for each layer, each layer can have its own texture assigned. Texture variations within one layer such as having different textures in the top and bottom half of the same layer or having different textures on each side of a fault is also possible.

5.2 Scalar texture transfer function (TTF)

While the layer TTF represents the interpreted horizons as textures, the scalar TTF represents uninter-

preted seismic attribute data as textures. The scalar TTF is equal to the layer TTF except that it is the seismic attribute values that are used as look up values. This makes it possible to control the textural appearance for regions on the cutting plane which have seismic attribute values in certain ranges. The scalar TTF texture is overlaid on the layer TTF texture using the over operator. The α value defines its transparency in the various regions. This combined view expresses how individual seismic attributes relate to layers, i.e., if intervals of an attribute are confined within certain layers or change significantly (or subtly) between layers. It also represents a unified visualization of layer data and seismic attribute data through textures.

Typically the repeat rates of a scalar texture are taken from the layer it is drawn on. However the user can set multiplicative factors in the repeat values in the scalar TTF to change this. We do this by default so textures can maintain the same repeat factors when crossing layers of different thicknesses. For a layer twice as thick as another one the vertical repeat of the thick layer's texture will be half of the thin layer's. A scalar texture crossing the layers would abruptly change its repeat rates. To have consistent repeats across layers as can be seen for the brown brick texture in Figure 7, the user can change in the layer TTF the vertical repeat for the thin layer to half of what it is for the thick layer.

The selection of repeat rates for the textures is highly dependent on the degree of zoom. When zooming out, textures will be perceived as being too high frequent and when zooming in they will be perceived as being too low frequent. For this reason we multiply all the repeat factors with a global user definable repeat factor which is manually set according to the zoom level.

5.3 Rendering uninterpreted and interpreted data

To inspect the uninterpreted data directly on cutting planes we apply the color TF on the scalar values of a seismic attribute. We also introduce the concept of a continuous transition from illustrative rendering of interpreted data to rendering of uninterpreted data. The transition is done by smoothly blending from visualizing textured cutting planes to visualizing cutting planes colored by the color TF with seismic attribute values. This not only gives a smooth transition from one mode to the other but also introduces an intermediate rendering mode where interpreted data is superimposed on the uninterpreted data. The balance between the two data sources can be adjusted to get what the user perceives as an optimal balance between the rendering techniques. See Figure 10.

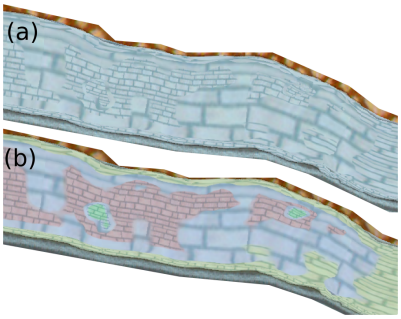


Figure 8: Layer TTF, scalar TTF and color TF combined. Instead of using different textures on intervals of the scalar values we use the same texture with four different repeat rates. It is difficult to discern the textures in a). In b) we blend in colors from the color TF to more easily discern the textures.

6 Rendering cutouts and surroundings

We implement volume rendering with one transfer function for the cutout and another one for the surroundings to support different rendering styles. By doing this we can achieve rendering of extruding features in the cutout and opaque ground rendering in the surroundings as seen in geological illustrations.

For volume rendering in the cutout we use the color TF on seismic attribute data introduced earlier. To specify transparencies in the volume rendering we extend the color TF with an α channel. By multiplying a voxel's α value from the color TF with the α value from the layer TTF we can adjust the transparencies based on the w value of the sample. Now we can do volume rendering on selected layers by manipulating the α in the layer TTF and making layers transparent or semitransparent.

For visualizing the surroundings we do volume rendering where each voxel is given the average

color of the 2D texture at the voxel position. The average color is precalculated for each 2D texture. The opacity is controlled by a separate opacity transfer function for the surroundings. It maps the w parameter to opacities enabling a layer oriented volume rendering of the surroundings. The opacity can then be set for instance to render certain horizon surfaces or layers semitransparently. When performing smooth transitions from rendering of interpreted data to rendering of uninterpreted data we go from using the average color of the 2D texture at the voxel position to using the voxel's color according to the color TF and the seismic attribute value at that position. In the images of this article we render the top and bottom horizon opaquely to get an opaque ground as seen in geological illustrations.

In the following paragraphs we describe how volume rendering is combined with texturing of the cutting planes. We perform ray casting with empty space skipping as suggested by Krüger and Westermann [8]. The entry and exit point of each ray is further clipped to the roaming box. Volume rendering with the transfer function for the surrounding is performed outside the cutout, while the transfer function for the cutout is used inside the cutout. Texturing is done at points where the parameterization volume intersects the exterior of the roaming box or the interior of the cutout box. See Figure 9 for a 2D depiction which acts as a reference to the following description. Texturing is done at the entry

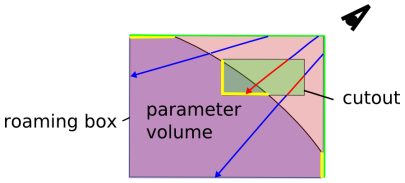


Figure 9: Combining volume rendering with texture rendering. The green line depicts the entry points. The yellow lines show where texturing is done. Red ray segments show where the cutout transfer function is used and blue ray segments show where the surrounding transfer function is used

point if the entry point is inside the parameter volume (green/yellow border). If not volume rendering with the transfer function for the surrounding is performed until the end point (upper blue ray) or until the cutout is intersected. If the cutout is intersected then volume rendering with the transfer function for the cutout is used (red segments) until the cutout exit point is reached. If the cutout exit point is inside the parameter volume texturing is performed (yellow border). If not, ray casting with the transfer function for the surrounding is performed until the exit point. A ray is always terminated if opacity reaches 1.

By doing volume rendering only on selected layers we can easily achieve the effect seen in geological illustrations of extruding layers in the cutouts. For exploration of the seismic data this is useful when the user wants to consider only one layer at a time. For instance the oil reserves are typically trapped between horizons in so called reservoirs. If the expert wants to perform volume rendering to explore such a reservoir it would be natural to confine the volume rendering to the layer the reservoir is in. See the bottom of Figure 1 for an example of volume rendering in a cutout limited to a layer. It shows a combined texture and volume rendering with an extruding layer. Volume rendering is performed only for layer 3 with brown color to mimic a geological illustration. The layer discontinuity is due to a fault. Turquoise patches on the textures show areas with high reflection values. Figure 10 shows a smooth transition from illustrative rendering to seismic attribute rendering.

The texture calculation and volume rendering is performed on the GPU in a single pass. With a Geforce 8800 GTX graphics card and an image size of 800×800 we achieve 5 frames per second. Without the extrapolation as described in 4.3 the frame rate is doubled. The three component parameterization volume is of size $128 \times 128 \times 128$ and the reflectance volume of size $240 \times 271 \times 500$. The Ai volume is of size $96 \times 96 \times 500$ and covers a smaller area than the reflectance volume. The 2D textures are each of size 64×64. A filmclip can be seen on

`http://www.cg.tuwien.ac.at/research/publications/2007/patel_daniel_2007_IRSD/patel_daniel_2007_IRSD-movie%20clip.avi`

7 Conclusions and future work

We have presented a technique for illustrative rendering of interpreted geological data. We have also shown how to create combined visualizations of interpreted and uninterpreted seismic data for valida-

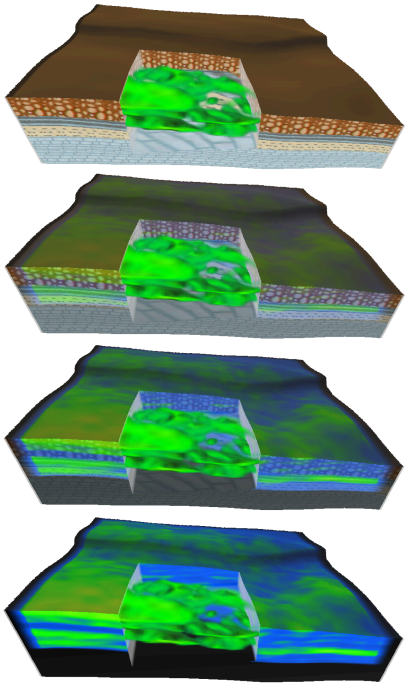


Figure 10: Blending from illustrative rendering to uninterpreted data rendering of the Ai attribute. Volume rendering is performed in areas with high Ai values. On the right of the cutout one can see how the green area having high Ai values corresponds to a layer. The black areas contain no data.

tion and comparison reasons and for creating visualizations that can be targeted to anyone from laymen to domain experts. On the technical side we have presented the concept of 2D texture transfer functions with deformed textures.

Illustrative techniques can make it faster to evaluate large oil prospects. It can also improve communication between different stakeholders and towards media, public sector and politicians. In the future we will look into methods making it possible to do illustrative rendering of uninterpreted data.

References

[1] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

[2] L. Castanie, B. Levy, and F. Bosquet. Volumeexplorer: Roaming large volumes to couple visualization and data processing for oil and gas exploration. *Proc. of IEEE Visualization '05*, pages 247–254, 2005.

[3] M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. *Computer Graphics Forum*, 21:209–218, 2002.

[4] F. Dong and G. Clapworthy. Volumetric texture synthesis for non-photorealistic volume rendering of medical data. *The Visual Computer*, 21(7):463–473, 2005.

[5] T. Frank. *Advanced Visualisation and Modeling of Tetrahedral Meshes*. PhD in geosciences, Institut National Polytechnique de Lorraine, 2006.

[6] J. Grotzinger, T. H. Jordan, F. Press, and R. Siever. *Understanding Earth*. W. H. Freeman and Company, 1994.

[7] W.-K. Jeong, R. Whitaker, and M. Dobin. Interactive 3d seismic fault detection on the graphics hardware. *Volume Graphics*, pages 111–118, 2006.

[8] J. Krüger and R. Westermann. Acceleration techniques for gpu-based volume rendering. *Proc. of IEEE Visualization '03*, pages 38–47, 2003.

[9] A. Lu and D. S. Ebert. Example-based volume illustrations. *Proc. of IEEE Visualization '05*, pages 83–92, 2005.

[10] R. Moyen. *Parametrisation 3d de lespace en geologie sedimentaire: le modele geochron*. PhD in geosciences, Institut National Polytechnique de Lorraine, 2005.

[11] S. Owada, F. Nielsen, M. Okabe, and T. Igarashi. Volumetric illustration: designing 3d models with internal textures. *SIGGRAPH '04*, pages 322–328, 2004.

[12] J. Plate, M. Tirtasana, R. Carmona, and B. Fröhlich. Octreemizer: a hierarchical approach for interactive roaming through very large volumes. *Proc. of VISSYM '02*, pages 53–64, 2002.

[13] T. Ropinski, F. Steinicke, and K. H. Hinrichs. Visual exploration of seismic volume datasets. *Journal Proc. of WSCG '06*, 14:73–80, 2006.

[14] L. Wang and K. Mueller. Generating subresolution detail in images and volumes using constrained texture synthesis. *Proc. of IEEE Visualization '04*, pages 75–82, 2004.

Visualizing large-scale IP traffic flows

Florian Mansmann, Fabian Fischer, Daniel A. Keim, Stephen C. North

University of Konstanz, Germany
AT&T Research, USA
Email: {mansmann,fischerf,keim}@inf.uni-konstanz.de, north@research.att.com

Abstract

Hierarchical Network Maps are a scalable approach to the presentation of IP-related measurements on the global Internet. This study focuses on how to extend them for emphasizing the source destination relationship of network traffic aggregated on IP prefix, autonomous system, country, or continent. Edge bundles consisting of several spline curves visually group traffic that shares common ancestor nodes along the IP/AS hierarchy.

1 Introduction

Today, signature-based and anomaly-based intrusion detection is considered the state-of-the-art for network security. However, fine-tuning parameters and analyzing the output of these methods can be complex, tedious, and even impossible when done manually. If this situation was not challenging enough, current malware trends suggest an increase in security incidents for the foreseeable future. The health of the network infrastructure clearly depends on the effectiveness of both manual and automated methods to analyze, comprehend, and disseminate understanding of large network data sets.

Hierarchical Network Maps (HNMaps) are an approach to the presentation of IP-related measurements on the global Internet. They are based on a hierarchy ($prefix \rightarrow AS \rightarrow country \rightarrow continent$) on top of all Internet subnet prefixes and displayed using a space-filling visualization technique. This pixel-conservative approach is appropriate as display space is a scarce resource when displaying about 200,000 IP prefixes at once.

In previous work, we considered network statistics observed at a single vantage point (arriving at or leaving from a particular gateway) and displayed it either by its source or destination in the IP address space. In this study, we consider traffic being transferred through multiple routers, such as in service provider networks. The use of *edge bundles* enables visually displaying and detecting patterns through accumulative effects within nodes of the AS/IP hierarchy. The novelty of this approach lies in its capability to support the formation of a mental model that places each autonomous system or IP prefix on a map while linking nodes according to the traffic under consideration, and at the same time limiting visual clutter.

The rest of this paper is structured as follows: we briefly discuss the employed database technology to speed up the backend of our application and review related work. Our HNMap appraoch is then discussed and extended through so-called edge-bundles. Afterwards, we examine generation of random data and apply the presented methods before assessing the overall contribution.

2 Efficient querying of large IP-related data sets

To support visualization that is fast enough for interactive data exploration, we need not only to consider efficient rendering techniques, but also be aware of database technology as precondition for flexibility in querying different data sets as well as for speed. We therefore briefly regard the *multi-dimensional data model* ([13]) which stores large amounts of *facts* with associated numerical *measure* in *data cubes* that are particularly well-suited for data analysis (in contrast to storing of transactional data). Queries aggregate measure values over a range of dimensional values to provide results such as the number of security events aggregated on each AS in a given country (dimension IP hierarchy) at a certain day (dimension time). Figure 1 shows the user interface to specify the database query and visualization parameters.

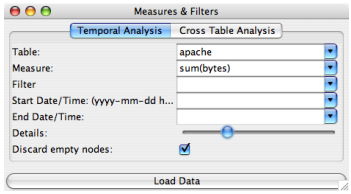


Figure 1: Specifying the database query and visualization parameters.

Using the snowflake schema, a separate table is created for each level of the dimension IP address and the tables' entries are linked accordingly. For example, each IP address is linked to the tuple of its advertised IP prefix, which in turn is linked to its AS, etc. Pre-joining the lowest level of the IP hierarchy with all its upper level entries avoids expensive join operations during interactive analysis and thus considerably speeds up queries. *IP addresses* are grouped by *IP prefix* → *autonomous system* → *country* → *continent* and we thus obtain the hierarchy as shown in Table 1.

Table 1: IP/AS hierarchy

Level	Name	Entries
1	continents	7
2	countries	190
3	autonomous systems	23054
4	prefixes	197427

The snowflake schema is not limited to the dimension IP hierarchy, but can be - depending on the analysis field - easily extended to various dimensions, such as time, protocol, ports, or type of event.

3 Related Work

A common way of displaying hierarchical data is in layouts where child nodes are placed inside parent node boundaries. Such displays provide spatial locality for nodes under the same parent, and visually emphasize the sizes of sets at all levels in the hierarchy. Usually, leaf nodes may have labels or additional statistical attributes that may be encoded graphically as relative object size or color.

The most important layouts of this type are *Treemaps* – space-filling layouts of nested rectangles, of which there are several main variants. The earliest variant was the *slice-and-dice Treemap* [8]. Here, display space is partitioned into slices whose sizes are proportional to to the sum of the nodes they contain. At each hierarchy level this procedure is repeated recursively, rendering child nodes inside parent rectangles while alternating between horizontal and vertical layouts. This is not difficult to program and can run efficiently, but long, thin rectangles arise, which are hard to perceive and compare visually.

Squarified Treemaps [3], remedy this deficiency by using rectangles with controlled aspect ratios. Rectangles are prioritized by size, so large ones are treated as the most critical ones for layout. This improves the appearance of Treemaps, but does not preserve the input node order, which is also a problem in some applications. This drawback was noticed, and *Ordered Treemaps* [2] were introduced to address it.

An alternative, non-Treemap layout algorithm which is not space filling was applied in [7] to the visualization of computer security data. Their proposed method maps hosts to rectangles, and subnets to larger enclosing rectangles. In contrast to this method, the approach proposed here has the goal of integrating both geographic and abstract layouts in the same view, and scaling up to the entire IPv4 address space.

Another related area is recent work on rectangular layouts in cartography, such as rectangular cartograms [5] which optimize the layout of rectangles with respect to area, shape, topology, relative position, and display space utilization. A genetic algorithm has been applied to find a good compromise between the objective functions describing the above mentioned properties. This technique renders layouts offline, not interactively and does not yet exploit hierarchical structures.

HNMap [11] supports the formation of a mental model of measurements reflecting the global Internet. It combines several layout techniques to cope with a large, multilevel AS/IP hierarchy in a tool that runs fast enough for interactive response. In another study [10], we evaluated alternative layouts and reported a case study with network data from a web server, a university network gateway, and an intrusion detection system (IDS) from a service provider to gain deeper insight into these large data sets.

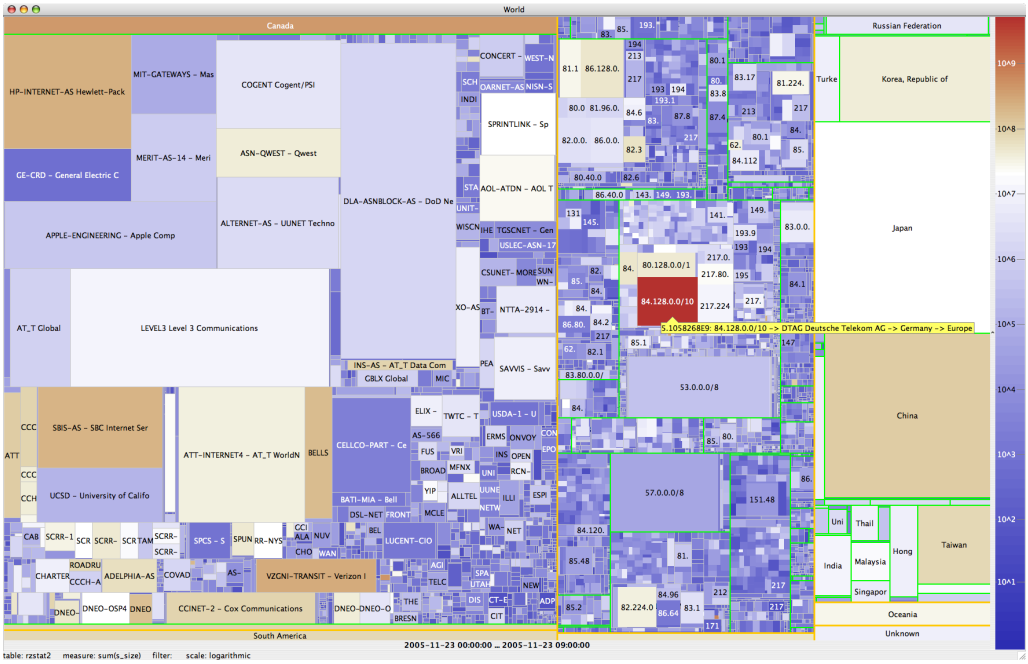


Figure 2: Multi-resolution HNMap approach to display aggregated IP-referenced measurements of prefixes (middle), ASes (left), countries (right), and continents (bottom) using a bi-color scale.

Drawing network traffic as lines on top of maps is well-known and has been studied in the cartographic community. Unfortunately, highly interconnected nodes lead to a lot of visual clutter and make it difficult to recognize any structure. Becker et al. [1] attack the problem by inventing *line shortening* and visualizing an adjacency matrix instead of geographic objects. Brushing has also been recommended [15].

A further approach to show the movement of objects from one location to another are so-called *flow maps*. Traditionally, these flow maps were hand drawn to reduce visual clutter introduced by overlapping flows. Phan et al. presented a method for generating well-drawn maps, which allow users to see the differences in magnitude among the flows while minimizing the amount of clutter [14]. However, interpretation of flow maps with multiple vantage points stays challenging.

A recent study [6] handles the problem of visual clutter elegantly: a hierarchical classification of nodes is exploited for bundling lines that connect leaf nodes, to visually emphasize correlations. Inspired by this work, we draw *edge bundles* on top of HNMaps in this paper. This enables us to view end-to-end relationships in network data sets, instead of limiting our analysis to the outgoing or incoming traffic from a single vantage point.

4 Hierarchical Network Map

In visualizing times series of network statistics, setting node (rectangle) sizes proportional to a time series variable leads to confusing displays, due to repositioning of nodes between HNMap frames. Figure 2 shows the multi-resolution HNMap approach addressing spatial memory by fixing node positions to facilitate tracing through time. A reasonable approach to this is to make node sizes proportional to the number of IP addresses contained, which is static in our experiments.

The general visualization paradigm of our technique places child nodes within the bounds of their parent node's rectangle. This results in a grouping operation which adds semantic meaning to the otherwise unstructured data. The benefit is obvious for

the parent child relationship among the continent and country as well as the AS and prefix nodes. However, the relationship between countries and ASes is not always clear. For this, we rely on statistics for each IP within an AS, which we obtained from a commercial GeoIP database [12]. Unfortunately, the country information within the registration services of ARIN, RIPE, AFRINIC, APNIC, LACNIC as obtained from [16] were uncomplete and sometimes misleading (a lot of ASes are registered to EU rather than a country).

The upper two levels of the AS/IP hierarchy are geographic entities. In general, geographic visualization is often very compelling– two-dimensional maps are familiar to most people as a convention for representing three-dimensional reality. Remarkably enough, mental models derived from maps are effective for many tasks even when extreme scales and nonlinear transformations are involved. Many approaches have been investigated for showing geographically-related as well as more abstract information on maps [4].

As a similarity measure for the AS level, we calculated the middle IP address of each AS by averaging the weighted middle IP address of all its advertised IP prefixes. This information is only meaningful to a certain extent as ASes sharing similar prefixes are positioned next to each other, but it does not take connectivity among ASes into account.

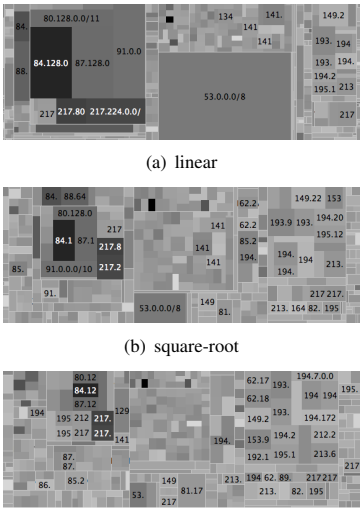
The lowest level of the hierarchy consists of the prefixes which have a clearly defined order. Our system therefore offers the capability to display this level using the Ordered Strip Treemap [2] with a line-wise sorting order. However, the HistoMap 1D method scales better to the large hierarchy data at hand with respect to visibility of small prefixes, layout preservations, and rendering performance (cf. [10]). The hierarchy levels can be interactively drilled-down or rolled-up. The mouse cursor highlights the measure for one particular hierarchy node and labels for the current hierarchy level as well as all parent nodes. When loading a data set, pruning nodes with little or no traffic frees space for the current analysis and retaining layout stability at the same time becomes an important aspect [10]. Since this paper is a continuation of our previous work, we discard nodes with no traffic in the HNMaps presented in this study.

Large differences in sizes between IP prefixes, ASes, countries, and even continents turn visual comparison of the respective rectangles into a challenge, especially when dealing with ordinary computer displays as opposed to wall-sized displays. We opted for a compromise by scaling the IP prefix sizes (number of contained IP addresses) and therefore indirectly also the upper level aggregates using square-root or, alternatively, logarithmic scaling:

$$f_{sqrt}(x) = \sqrt{x} \tag{1}$$

$$f_{log}(x) = \log x + 1 \tag{2}$$

The effects of node size scaling are illustrated in Figure 3. Note that the size of the upper level rectangles is determined through the sum of the scaled child nodes.



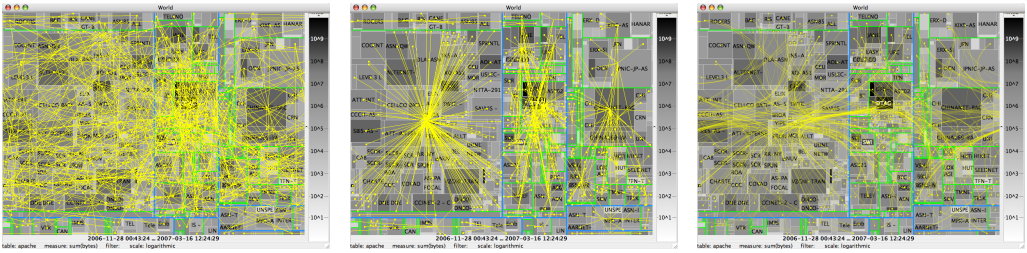
(a) linear

(b) square-root

(c) logarithmic

Figure 3: Effects of scaling on the space-filling layout demonstrated on some IP prefixes in Germany.

In the proposed visualization, the value of a measure of interest is encoded as color, using a fixed color scale and logarithmic normalization: $colorindex(v) = log(v+1)/log(v_{max}+1)$. The user is free to move the transition point (white) in the bi-color scale (blue to red) to focus his analysis on a range of values. When drawing edge-bundles, we substitute the blue-red color scale with a white-black color scale for the HNMap in the background to improve visibility of the bundles.



(a) Straight connection lines (b) Exploiting hierarchical structure (c) Compromise through edge bundles

Figure 4: Comparison of different strategies to draw adjacency relationships among nodes of the IP/AS hierarchy on top of the HNMap.

In some cases, analyzing an absolute measure of network traffic (e.g., number of connections or bytes transferred) provides hardly any insight in the time-varying dynamics of the data. Therefore, we calculate and display the change over time in some analysis scenarios.

5 Drawing routed traffic for multiple sources and destinations

Figure 4 shows three different approaches to draw adjacency relationships among nodes of the IP/AS hierarchy. (a) The problem of visual clutter is obvious when straight lines are drawn. (b) Simply connecting with the upper level nodes of the hierarchy removes visual clutter, but adds a lot of ambiguity, e.g. the lines connecting the left star (U.S.) and the stars in Europe (middle) – one for every country – are overdrawn several hundred times. (c) An interesting compromise is to use so-called edge bundles and transparency effects to combine the advantages of the latter approaches.

5.1 Edge Bundles

A recenty study proposed a way of using spline curves to draw adjacency relationships among nodes organized in a hierarchy [6]. Figure 5 illustrates how we use hierarchy structure to draw a spline curve between two leaf nodes. P_{start} (green) is the center of the source rectangle representing a node in the IP/AS hierarchy and P_{end} (red) the center of the destination rectangle. All points P_i (green, blue, and red) from P_{start} over $LCA(P_{start}, P_{end})$ (least common ancestor) to P_{end} form the cubic B-spline's control polygon. Because many splines share the same LCA (in this case the center point of the world rectangle), we do not use them for the control path.

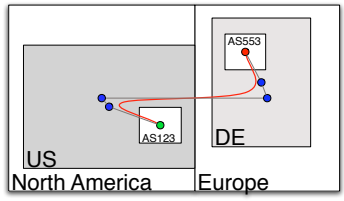


Figure 5: Spline (red) with control polygon (gray)

The degree of the B-spline used to control the bundling strength. In our experiments, a degree of 6 has proven to be a good choice for the cases where we have enough control points.

5.2 Edge Coloring

In general, we considered two options for coloring the edges, namely (a) use of color to convey the amount of traffic transferred and (b) use of color to distinguish edges. For the first case (see Fig. 6), we employed a heat map color scale from yellow to red using 50 to 0 percent alpha blending to visually weight the high traffic links. Naturally, the less important splines were drawn first. For the second case (see Fig. 7), we chose a HSI color map with constant saturation and intensity which is silhouetted against the background. The largely varying colors make tracing of single splines easier, but we noticed that users were strongly distracted by the colorful display.

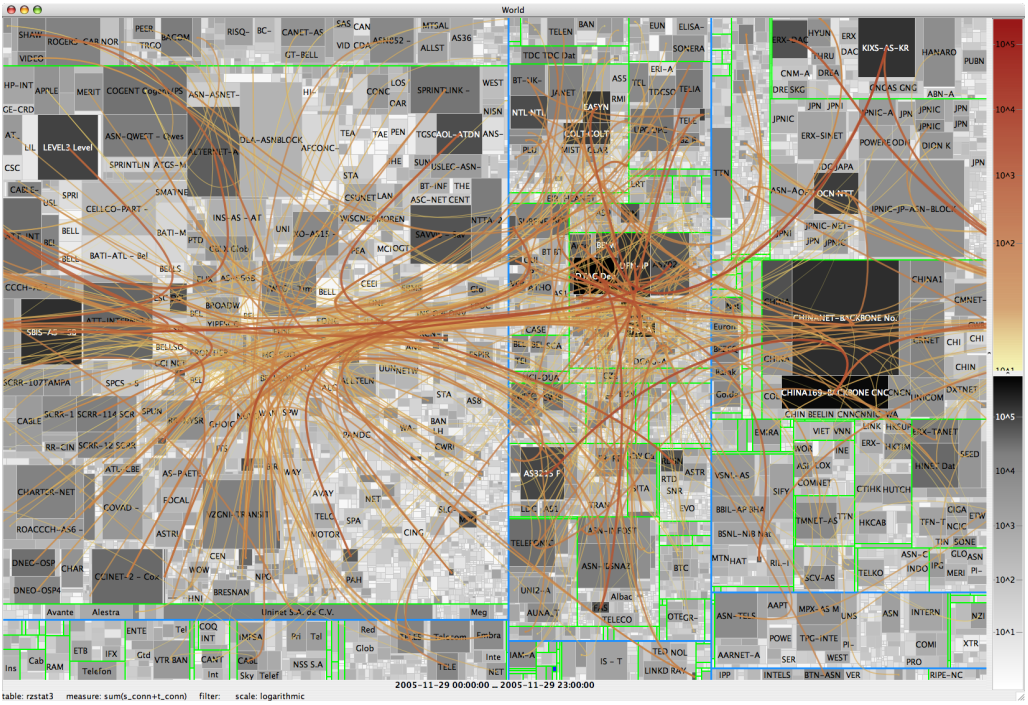


Figure 6: HNMap with edge bundles showing the 500 most important connections of one-day incoming and outgoing traffic of our university gateway (the anonymized destination/source is semi-randomly chosen). Color combined with transparency and line width communicate the amount of traffic of each spline.

5.3 Interaction design

In our opinion, the task of tracing splines is not solvable by means of coloring as soon as their number excels about one hundred nodes. We therefore tried to tackle the problem through interaction. Basically, there are two possible interaction scenarios. The first scenario is that a particular spline or region is selected, refined, and visually highlighted. Selecting a spline or a region by the start and end points of the splines is relatively intuitive to implement using the spatial data structure of the HNMap application.

However, the second scenario comprises marking a whole bundle of splines and was discarded since the implementation becomes tedious at this place (probably a point in polygon test for each pixel of all splines).

After specifying the start or end points of the splines using mouse interaction, we redraw all splines using their previous RGB color values – the not selected splines with higher alpha blending and the selected ones without transparency effects on top. This allows the user to easily trance the few highlighted splines to their end.

5.4 Data Simulation

As it was impossible to obtain real traffic data from service providers for publishing, we settled on using real netflow data from our university gateway and substituted the internal source IP prefix with a randomized one according to algorithm 1.

This randomization schema is based upon the assumption that nodes with more traffic are more likely to communicate with several other nodes, whereas the opposite applies to low traffic nodes.

Figures 6 and 7 demonstrate the outcome of our randomization schema based upon a one-day traffic load of our university gateway. The images highlight the high-level connectivity information while still being able to recognize the low-level relations

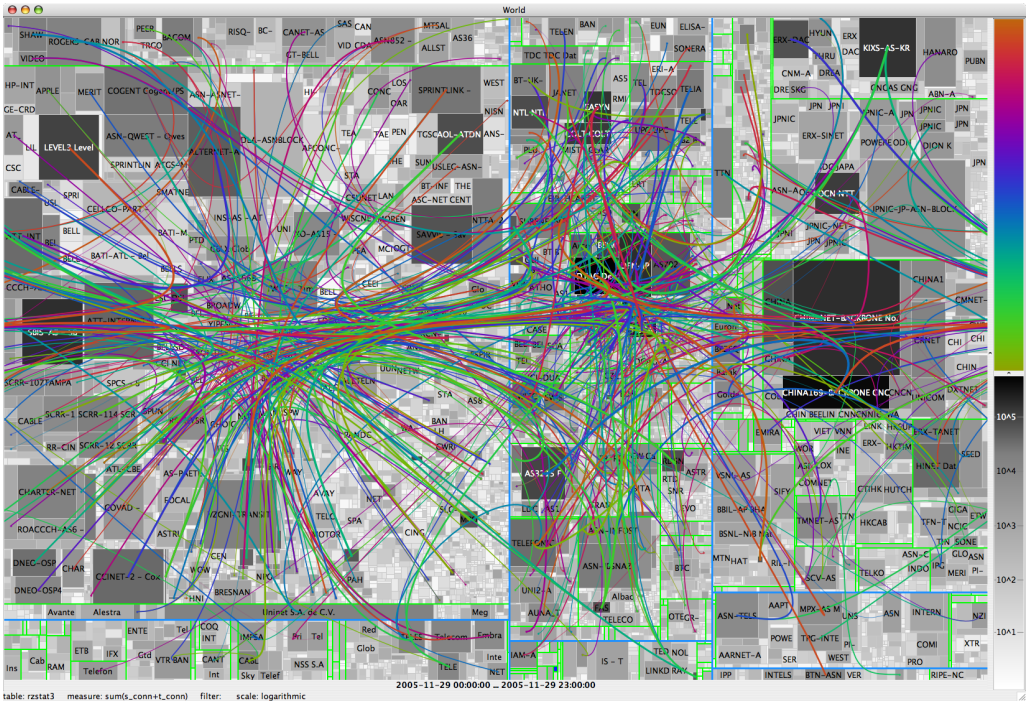


Figure 7: HNMap with randomly colored edge bundles makes splines more distinguishable. The amount of traffic is expressed only through spline width.

begin
  $SortedList \leftarrow$ all unprocessed nodes and weights
  while $|SortedList| > 0$ **do**
    $(n_{min}, t_{min}) \leftarrow SortedList.removeMin()$
    $(n_{rand}, t_{rand}) \leftarrow SortedList.RandomSample()$
    $drawSpline(n_{min}, n_{rand}, t_{min})$
    $SortedList.update(n_{rand}, t_{rand} - t_{min})$
end

Algorithm 1: Randomization schema to simulate backbone provider traffic.

to a large extent. The strong bundling effects in North America (center left) are explainable through the proximity of two control points to each other (North America, United States).

6 Findings and Evaluation

Edge bundles offer the possibility to convey source destination relationships on top of the HNMap and thus leverage the application from a purely measurement based analytical approach to a more complete view on large-scale network traffic. Since it is very challenging to trace single splines as soon as a certain volume of traffic links are placed on the map, we experimented with the RGB and alpha values of the spline colors. Mouse interaction is used to select splines at their start or end points in order to silhouet them from the remaining splines.

When monitoring larger networks, focusing the analysis on a particular type of traffic, for example communication of hijacked computers of a botnet, helps to significantly reduce the number of nodes and connections to be displayed.

During the analysis, further detailed information of other attributes of the data set at hand can be displayed using bar charts or the Radial Traffic Analyzer [9]. One major drawback of our approach is

that conveying significance of splines or to distinguishing them through color disqualifies the visual variable color for being used to indicate direction of traffic flows.

7 Conclusion

HNMaps visually represent network traffic aggregated on IP prefixes, ASes, countries, or continents and can be used for exploration of large IP-related data sets. In this study, we extended HNMaps through edge bundles to emphasize the source destination relationship of network traffic. Rather than representing new visualization or analysis techniques, we combined two existing methods and applied them to large-scale network traffic to gain deeper insight.

In order to facilitate tracing of splines, which represent source destination relationships among networks or abstract nodes of the IP hierarchy, we compared two alternative coloring schemes. Moreover, mouse interaction was proposed to visually enhance network traffic links of interest.

References

[1] R. A. Becker, Stephen G. Eick, and A. R. Wilks. Visualizing network data. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):16–21, March 1995.

[2] Benjamin B. Bederson, Ben Shneiderman, and Martin Wattenberg. Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. *ACM Trans. Graph.*, 21(4):833–854, 2002.

[3] M. Bruls, K. Huizing, and Jarke J. Van Wijk. Squarified treemaps. In *Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, pages 33–42, 2000.

[4] Martin Dodge and Rob Kitchin. *Atlas of Cyberspace*. Addison-Wesley, 2001.

[5] Roland Heilmann, Daniel A. Keim, Christian Panse, and Mike Sips. RecMap: Rectangular Map Approximations. In *InfoVis 2004, IEEE Symposium on Information Visualization, Austin, Texas*, pages 33–40, October 2004.

[6] Danny Holten. Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, 2006.

[7] Takayuki Itoh, Hiroki Takakura, Atsushi Sawada, and Koji Koyamada. Hierarchical visualization of network intrusion detection data. *IEEE Computer Graphics and Applications*, 26(02):40–47, 2006.

[8] B. Johnson and Ben Shneiderman. Tree-maps: A space filling approach to the visualization of hierarchical information structures. In *VIS '91: Proceedings of the 2nd IEEE Conference on Visualization*, pages 284–291, 1991.

[9] Daniel A. Keim, Florian Mansmann, Jörn Schneidewind, and Tobias Schreck. Monitoring network traffic with radial traffic analyzer. In *Proc. of IEEE Symposium on Visual Analytics Science and Technology 2006 (VAST 2006)*, pages 123–128, 2006.

[10] Florian Mansmann, Daniel A. Keim, Stephen C. North, Brian Rexroad, and Daniel Shelehedal. Visual analysis of network traffic for resource planning, interactive monitoring, and interpretation of security threats. *IEEE Transactions on Visualization and Computer Graphics*, 13(6), 2007.

[11] Florian Mansmann and Svetlana Vinnik. Interactive Exploration of Data Traffic with Hierarchical Network Maps. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1440–1449, 2006.

[12] Maxmind, LLC. Geoip database, 2007. `http://www.maxmind.com`.

[13] T. B. Pedersen and C. S. Jensen. Multidimensional database technology. *IEEE Computer*, 34(12):40–46, 2001.

[14] Doantam Phan, Ling Xiao, Ron Yeh, Pat Hanrahan, and Terry Winograd. Flow map layout. In *INFOVIS '05: Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*, page 29, Washington, DC, USA, 2005. IEEE Computer Society.

[15] Ben Shneiderman and Aleks Aris. Network visualization by semantic substrates. *IEEE Trans. Vis. Comput. Graph.*, 12(5):733–740, 2006.

[16] Team Cymru. IP to ASN Lookup Page, April 2007. `http://www.cymru.com/BGP/asnlookup.html`.

Image-Space GPU Metaballs for Time-Dependent Particle Data Sets

C. Müller, S. Grottel, T. Ertl

Visualisierungsinstitut der Universität Stuttgart
Email: {mueller,grottel,ertl}@vis.uni-stuttgart.de

Abstract

Molecular dynamics simulations are today a widely used tool in many research fields. Such simulations produce large time-dependent data sets, which need to be interactively visualised allowing efficient exploration. On the other hand, commonly used point-based rendering of the individual particles usually fails to emphasise global contiguous structures like particle clusters. To solve this issue, we want to visualise these data sets using metaballs. Free particles form individual spheres while clustered particles result in larger closed shapes. Using image-space hardware-accelerated techniques provides interactive frame rates and high visual quality. We present two approaches evaluating the metaballs shapes on the graphics hardware. The first approach uses a vicinity data structure stored in graphics memory to evaluate the metaballs shape in a single rendering pass. The second approach uses multiple rendering passes to approximate the metaballs shape on a per pixel basis.

1 Introduction

A large number of technical and scientific problems can nowadays be solved using molecular dynamics simulations. One such example is nucleation, i. e. the state change from a vapour into the liquid phase. This process is found in many physical phenomena, e. g. the formation of atmospheric clouds or the processes inside steam turbines, where a detailed knowledge of the dynamics of condensation processes can help to optimise energy efficiency and avoid problems with droplets of macroscopic size. The liquid phase emerges through spontaneous density fluctuations in the vapour which lead to the formation of molecular clusters, the predecessors of liquid droplets. The figures 1, 9, and 10 show such nucleation data sets. For the calculation of key properties like the nucleation rate, it is essential to make use of a meaningful definition of molecular clusters, which currently is a not completely resolved issue.

The visualisation of the simulation results, especially the emphasis of molecular clusters, is therefore essential for analysing simulation results and cluster detection algorithms. Point-based rendering of such data sets is a common practice and delivers fast visualisations of large amounts of particles, but does not convey the detailed shapes of clusters and obscures their real extents — an effect the users do not desire (see figure 1). On the other side, with metaballs [1] a much more suitable technique for solving this issue exists, since they form a closed and compact surface. However, current metaball rendering techniques either require costly isosurface extraction or are based on time-consuming raycasting, which prevents interactive visualisation of time-based data sets.

This paper presents our efforts to enable interactive visualisation of clusters in molecular dynamics data sets as metaballs. We think that image-space approaches are more promising in this context since they do not rely on preprocessing, like isosurface extraction, and hence should permit rendering of time-dependent data sets. Furthermore, image-space techniques, which include raycasting, should achieve an optimal visual quality without a prohibitively fine tessellation of the isosurfaces.

The remainder of this document is structured as follows: Section 2 briefly describes the theoretical background of metaballs and existing rendering algorithms. In section 3, we describe two approaches for implementing metaballs in image-space, one using additional texture memory and one using multiple rendering passes for solving the occlusion problem. Section 4 presents the results of our performance measurements and is followed by some final conclusions.

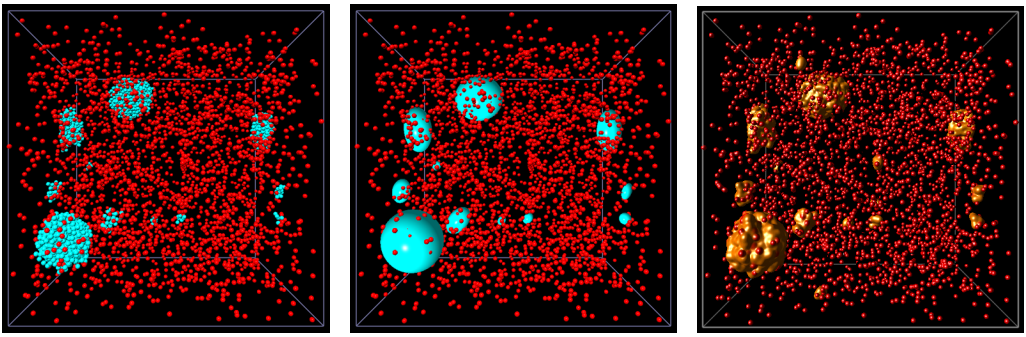


Figure 1: Point-based visualisation of an argon nucleation simulation data set. The left image highlights molecular clusters using different colours, the middle image uses ellipsoids and the right one uses metaballs. The metaball visualisation shows a closed surface of the molecule clusters while not exaggerating their size.

2 Related Work

The metaball technique was introduced by Blinn [1] for displaying molecular models and was originally called *blobs*. The idea is visualising molecules as isosurfaces in the simulation of an electron density field. Blinn uses the exponential function

$$D(r) = \exp(-ar^2)$$

as field function, which is derived from the density function of a hydrogen atom, with r being the distance of the current sampling point from the centre of the current metaball. The sum of the density functions of all atoms parameterised with the distance from the sampling point then yields the value of the density field at this point.

The notion of *metaballs* for the same concept goes back to Nishimura et al. [11][12], who use a piecewise quadratic field function. Several other field functions have been proposed later on, like a degree six polynomial by Wyvill et al. [16] or the degree four polynomial

$$D(r) = \left(1 - \left(\frac{r}{R}\right)^2\right)^2 \qquad (1)$$

with R being the radius of the influence sphere of the current metaball by Murakami et al. [9][12]. For a radius $r > R$, $D(r) = 0$ is assumed.

Two typical approaches of displaying metaballs are raycasting the density field and thus rendering the isosurface directly as suggested by Blinn or extracting the surface using the Marching Cubes Algorithm [7] and rendering it like any other mesh.

Using state-of-the-art graphics hardware, Nvidia showed how to use vertex and geometry shaders to evaluate the density field function and generate geometry on the GPU [13]. Kooten et al. [6] employed a point-based method for visualising metaballs on the GPU. Their goal is to render fluid surfaces made up of metaballs, e. g. water in a glass. To achieve this goal, they render a large number of particles which are forced on the implicit surface of the metaballs by velocity constraints and use a spatial hashing method for evaluating the density field on the GPU. Using repulsion forces between the particles, these are uniformly distributed over the surface and thus finally cover the whole metaballs, if their number is sufficiently large.

Earlier, Microsoft showed in their DirectX SDK [8] an image-based approach for achieving a metaball-like effect. They accumulate the density function and the surface normals via additive blending in off screen buffers during multiple rendering passes. As the approach has no depth information, it is limited to metaballs on a single plane.

Our framework for particle data visualisation [3], which the extension for displaying metaballs presented in this paper is intended for, uses glyphs based on the ellipsoid splatting technique presented by Gumhold [4], or more precisely, the approach by Klein et al. [5], which unfolds each glyph from a single point sprite vertex. The algorithm uses e. g. for an ellipsoid the centre point, the three radii and an orientation quaternion to compute the outline of the projected glyph in the vertex shader. The size of the projection determines the size of the

point sprite. The fragment shader then raycasts the glyph for each pixel the point sprite generates. This method does not only allow for raycasting spherical and ellipsoidal glyphs, but also more complicated ones like dipoles, arrows and *tubelets* [14].

3 Image-Based Metaball Rendering

The density function we use in our work is a scaled version of the function presented in equation 1:

$$D(r) = \frac{16}{9}\left(1 - \left(\frac{r}{2R}\right)^2\right)^2 \qquad (2)$$

with R being the radius of the sphere forming the metaball, $r \geq 0$, and assuming $D(r) = 0$ for $r > 2R$. The idea behind these factors is to generate a visually coherent impression when switching from raycasted spherical glyphs to metaball rendering or when mixing different glyphs and the metaball rendering. We choose the influence radius to be the doubled particle size of our data sets, and the threshold t for the isosurface in the density field is chosen such that a sphere, which does not form a metaball, looks identical to raycasted sphere glyphs. The scaling of the function with $\frac{16}{9}$ results in the density being one at the distance of the original sphere radius, i. e. $D(R) = 1$. Figure 2 shows a plot of our density function. However, our approaches are not limited to these choices. With minor changes to the used shader programs, it is possible to use any other density function, as long as the function has a finite support, any other influence radius, and any threshold value.

The two approaches presented in sections 3.1 and 3.2 have the common goal to produce a metaball visualisation from spheres, which are solely specified through their centre and radius, in image space without generating geometry. They therefore also share the same problem of occluded fragments, which will be described more detailed in the following section.

Our first idea to overcome this problem was making the missing information available to the fragment shader via a *Vicinity Texture*. The metaballs then can be rendered as point sprites in a single pass without processing the whole image area but at the expense of an increased number of expensive texture accesses. This approach is closely related to the glyph raycasting in our framework and described in section 3.1.

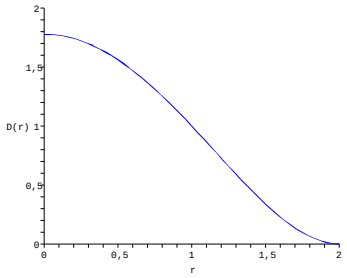


Figure 2: The density function from equation 2 with $R = 1$.

Our second approach, which is elaborated in section 3.2, goes quite in the opposite direction: instead of generating fewer but more expensive fragments, it constructs the metaballs from a lot of relatively cheap fragments in several rendering passes. Reminding of depth peeling [2], the final image results from a viewport aligned imaginary plane being moved through the data set. In each pass, the density function is evaluated for each pixel and the plane is moved by a distance value provided by an oracle. If these steps are adequately long and enough of them have been made, the surface of all metaballs should have been found.

3.1 Vicinity Texture

The main problem when trying to evaluate 3D metaballs in image space is occlusion, which prevents an accumulation of the density function by simply rendering the projection of the influence spheres of the particles. E. g. even if one knew in advance that $P1$ in figure 3 will not contribute to a metaball and therefore only rendered the surely opaque sphere with radius R instead of the whole influence sphere with radius $2R$, none of the fragments generated by $P3$ would be visible. Consequently, when accumulating the density within the projected influence sphere of $P2$, the contribution of $P3$ and therefore also the dotted metaball surface connecting $P2$ and $P3$ would be missing.

One would have to know for every fragment generated by an influence sphere which other spheres contribute to it, i. e. this information would have to be available on the graphics card. Kooten et al. [6] use a reversed version of the spatial hash table pre-

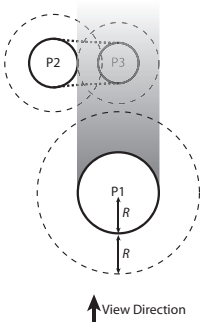


Figure 3: Occluded fragments hamper the image space computation of metaballs: the dashed influence sphere of $P3$ is completely occluded by $P1$. Therefore, it is not possible to accumulate the density function correctly in image space. As the contribution of $P3$ is completely missing, the dotted metaball surface cannot be found and $P2$ appears as normal sphere.

sented by Teschner et al. [15] to obtain for a certain area of the screen all relevant particles. Depending on the size of the grid subdividing the screen space, this hash table can become quite big, and as we intend to use the metaball visualisation in combination with glyphs raycasted on the GPU as described by Klein et al. [5] we chose to attach the vicinity information not to the screen position but to the vertices defining the object-space centre of the spheres. This has also the advantage that our map is view-independent and must only be updated, if the position of the vertices changes. Thus, we construct in a pre-processing step a texture which holds for each sphere the object-space position and the radius of all other spheres which are near enough to contribute to the density field. Before each set of influencing spheres, an extra entry holding the number of positions to follow is added. The coordinates of this counter are set as texture coordinates of the vertex. All spheres which do not have neighbours with which they potentially form a metaball are omitted in order not to waste texture memory.

Having the above-mentioned lookup table for the influencing spheres, the rendering of the metaballs happens in a single pass: the centre positions of the spheres are rendered as point sprites and the vertex shader adjusts the screen space size of the point to hold all the pixels of the projected sphere. For that, the desired object-space sphere radius is passed as homogenous coordinate of the vertex. The sign of the homogenous coordinate is additionally used to determine, whether the current sphere potentially forms a metaball. If it can form a metaball, its radius is doubled in order to make the sphere enclose the whole space where our density function (equation 2) does not vanish. In the fragment shader, the sphere is then raycasted, and as long as it surely cannot be part of a metaball the intersection point between the viewing ray and the sphere just has to be lit to complete the pixel [5].

If, however, the current sphere can form a metaball, we must determine whether it actually does so, and where the threshold value t is reached for the first time. For these purposes, we sample the density field within the influence sphere, whose boundary we just have computed, while advancing on the viewing ray until the end of the influence sphere. At each sampling point, the density function is evaluated on-the-fly for the current sphere and the ones in their neighbour list. If the sum of the densities, which is the final density at the sampling point, is sufficiently close to the isovalue t, the sampling ends.

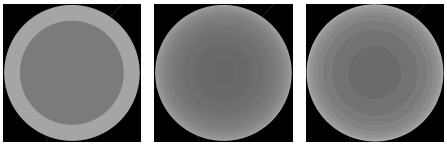


Figure 4: Influence of the recursive refinement of the isosurface. The colour denotes the length of the ray from the surface of the influence sphere to the isosurface in the density field. The leftmost image shows the result for only ten sampling points, the middle one for ten sampling points and four additional refinement steps, and the rightmost one for 50 sampling points.

It requires, however, a prohibitively large number of samples per pixel in order to find a sufficiently good isosurface in the density field. But the number of required samples can be effectively limited if the sampling is stopped once the density is above the threshold for the first time and the desired point on the isosurface is then searched by bisecting the previous sampling step recursively. Figure 4 shows this effect for four recursive refinement steps.

Once the isosurface of the metaball in the density field has been found, the OpenGL-conform depth of the fragment can be computed like for a normal sphere or any other glyph. For lighting the metaball surface, the normals are approximated as the sum of the normals of the contributing spheres weighted with the respective density contribution [8]. Since the accumulated density on the surface results to one, the weighted summation of normal vectors is equivalent to the use of barycentric coordinates and thus a legitimate interpolation.

3.2 The Walking Depth Plane

As we found that the texture lookup in the vicinity texture poses heavy load on the graphics card (see section 4), we tried an alternative approach without using a texture-based data structure. The goal was to remove the texture lookup as bottleneck and additionally to avoid restraints on data set sizes due to the available texture memory. The main idea of this approach is to use multiple rendering passes to approximate the correct isosurface through a moving plane of depth values.

Two frame buffer objects with 16 bit float RGB channels are used to store the needed data. The first buffer, called λ-buffer from now on, is used to store the depth of the pixel in image space, which approximates the targeted isosurface in means of the distance from position of the camera in world space coordinates. In a second channel of this buffer the maximum distance value for the pixel is stored as information for the termination criterion. The second buffer — we will refer to this one as density buffer — is used to evaluate the density field at the depths provided by the λ-buffer. Using these two buffers, the algorithm works as follows:

3.2.1 Initialising the λ-Buffer

The first rendering pass (Step 1 in figure 5) is used to initialise the λ-buffer. The starting and maximum depth values over all spheres for each pixel are calculated by raycasting the influence spheres of each particle. Depending on whether the starting or the maximum values are calculated, the front or back side of the spherical glyphs are raycasted. The minimum or maximum over all glyphs is determined using OpenGL depth tests. The pixels which are not set by any fragment of these glyphs are initialised using the clear colour value. Since these pixels will never be part of a metaball, their values must only trigger the loop termination criterion and have not to be comparable to real λ values calculated using the initialisation sphere glyphs.

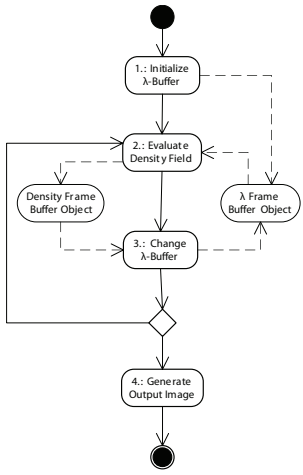


Figure 5: The different rendering passes of the *Walking Depth Plane* approach, together with the two frame buffer objects involved. The solid lines represent the control flow, while the dashed lines represent access to the frame buffer objects.

3.2.2 Evaluation of the Depth Field

After the initialisation step, the approximation of the metaball surface is done by repeating steps 2 and 3 (see figure 5). This loop and its termination are controlled by the CPU.

The evaluation of the density field in step 2 is very similar to the initialisation pass. The individual spheres are uploaded as points with the additional information of the influence radius of the density function. But instead of raycasting an implicit surface of the sphere, the density function for the sphere is evaluated at the position calculated from the viewing ray of the given pixel and the depth value stored in the λ-buffer. All density portions of all spheres are accumulated by performing additive blending on the float frame buffer the values are written to. To improve the upcoming update of the λ-buffer the density field can be evaluated at multiple depths at one time by summing the density at a specific depth in a separate channel of the

frame buffer object. Using a single RGB colour attachment allows for evaluating the density field at three positions at a time, which allow for a more substantiated guess of the next sampling position. A similar use of colour channels was presented in [10] for performing RBF-based volume rendering through splatting on up to four slices in one pass.

3.2.3 Moving the Depth Plane

The third step is to change the values of the λ-buffer and thus to move the surface described by these values closer to the targeted isosurface. Since the values are stored in a float colour attachment, they can be increased and decreased using alpha blending and fragments with a colour value describing the change step. It is also possible to use two λ-buffers used alternately. This way the calculation of the λ values for the next pass could not only consider the evaluated depth value, but also the current λ values. However, since we wanted to minimise the use of textures, we chose to use a rather simple oracle for computing the step size and direction from only the last λ value.

The direction of the step is directly given by the density value of the current pixel. If the density is below the threshold, the step must go forward into the viewing plane, and if the density is above the threshold, we already missed the isosurface and must step back. While this is intuitive, choosing a step size is not. The radius of the smallest sphere can be used as maximum step size. It is then still possible to miss a rather thin connection between spheres, but this is an intrinsic problem of any sampling. However, in all of our data sets this problem does not occur unless rendering time-dependent interpolated data sets with changing molecule sizes.

To approximate the targeted isosurface with sufficient precision, the step size must be decreased as the density reaches the threshold value. This decrease is controlled by an oracle. Since the density is a summation from several density functions, it is not possible to precisely determine the required step size. Therefore, we use two simple heuristics as our oracle. We can use the fact that in our molecular dynamics data sets the spherical particles only interpenetrate each other a little to our advantage here. Hence, the summed density functions characteristics does not differ dramatically from the individual function. As approximation to these characteristics we used quadratic attenuation for steps into the viewing plane and linear attenuation for steps out of the viewing plane: the step size $|\Delta|$ is

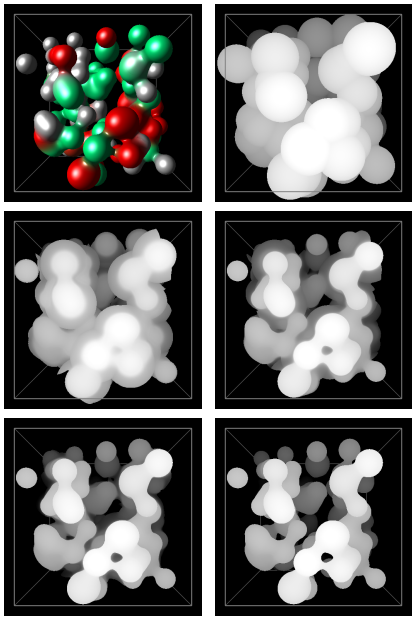


Figure 6: Metaball isosurface approximation after different number of iterations. Upper left image shows final output image of the *Walking Depth Plane* approach. The other images show the content of the λ-buffer after different numbers of iterations (from upper right to lower left): directly after initialisation, after 5, 10, 15, and 31 (final result) iterations.

$$|\Delta| = \begin{cases} \Delta_{max}(1 - \tilde{D}(\vec{x})^2) & \text{for } \tilde{D}(\vec{x}) < t - \epsilon \\ \frac{1}{2}\Delta_{max}(\tilde{D}(\vec{x}) - 1) & \text{for } \tilde{D}(\vec{x}) > t + \epsilon \\ 0 & \text{otherwise} \end{cases}$$

with Δ_{max} being the maximum step size, t the threshold, ϵ the approximation tolerance, and $\tilde{D}(\vec{x})$ the summed density over all metaballs in the data set at the position $\vec{x}$. Experimental results showed that this oracle is quite effective. Figure 6 shows the evolution of the λ values using the oracle with a data set consisting of rather huge metaballs formed by several individual spheres, making it hard to predict the correct step size as already mentioned above.

3.2.4 Terminating the Loop

The last issue to be solved is to define and to evaluate the criteria for terminating the iteration loop. As the maximum step size is known, the required iterations can be computed using this value and the size of the bounding box. This, of course, is only true as long as the oracle does not decrease the step size adaptively. Therefore, the maximum number of iterations must be increased, e. g. by a factor of two. For the data set shown in figure 6, the calculated value is 16 resulting in a maximum number of iterations of 32, which is a very good prediction. However, if the sphere radii are quite small, as in most of the data sets we present, the maximum number of iterations gets overestimated. An overall iteration maximum is used to prevent the application from visually freezing. As second mechanism ensuring the responsiveness of the application, a maximum execution time for the iteration loop is used: a minimum frame rate can be defined, which will be reached under all circumstances by early termination of the loop at the cost of worse approximation of the isosurface in the background of the data set. In our test runs we used a minimum frame rate of 1 FPS.

The last termination criterion of our software is the approximation of the isosurface, which is directly given by the density value calculated in step 2. If this value is sufficiently close to the threshold, the pixel is marked as finished, by setting the value of the depth buffer of both frame buffer objects to zero, which is done in one pass since both frame buffer objects share the same depth attachment. If a given percentage of all pixels is marked as finished, the iteration loop is terminated. Figure 7 shows such terminated pixels after a few iterations. The straight forward approach to evaluate this criterion would be using occlusion queries and count the created fragments relative to the size of the viewport. However, this is not possible on many systems, including all computers we tested our software on, because occlusion queries seam not to count fragments generated into a frame buffer object. As fall back solutions, one could count all texels with a value of zero in an additional rendering step or reading the whole image back and count them on the CPU. However, the overhead for both solutions is so big that it actually is the best idea to abandon this criterion and to rely on the other ones, which perform sufficiently well.

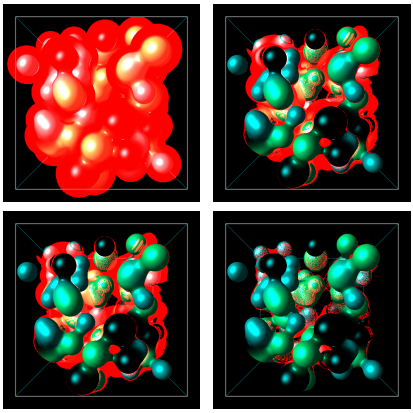


Figure 7: The metaball of the same data set as shown in figure 6 after 5, 10, 20, and 30 iterations (from upper left to lower right). The red colour channel encodes the finished pixels black (showing the isosurface in cyan) and unfinished pixels red.

3.2.5 Shading

The final output image is generated by deferred shading in step 4. After the iteration loop terminated, the λ-buffer holds the approximated distance values of all required points on the targeted isosurface. In an additional rendering pass, the density field is again evaluated at the positions stored in the λ-buffer for computing the surface normal vectors and the colours similarly as described in section 3.1. A final rendering into the real window's frame buffer uses the values from the frame buffer objects to write fragments with OpenGL-conform depth and a colour calculated by a phong lighting using the colours and normal vectors.

4 Results

As already mentioned, our goal was to interactively visualise molecular dynamics data sets with our metaball methods. Tables 1 and 3 show performance measurements for six different data sets. The *Argon* data set (figure 1) is a nucleation simulation of argon with 5000 molecules. The *Ethane* data set (figure 9) — the largest in our tests — consists of 25000 molecules and shows a nucleation process in a supersaturated configuration. Renderings of these two data sets using the *Walking Depth*

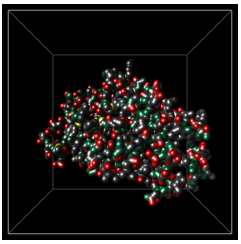


Figure 8: Disulfide Bond Formation Protein from the Protein Data Base consisting of 1454 particles.

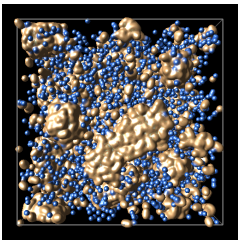


Figure 9: A nucleation simulation of supersaturated ethane consisting of 25000 elements.

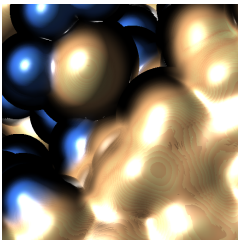


Figure 10: Undersampling artifacts when rendering the ethane data sets with the *Moving Depth Plane* and only 100 steps.

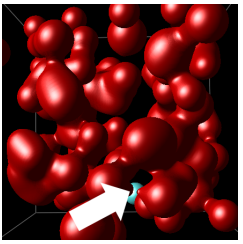


Figure 11: Missed surface parts (arrow) due to undersampling by the *Vicinity Texture* approach.

Plane approach are shown in the video accompanying this paper, captured in real time from the desktop. The data sets three to five, *Sim-1* (figure 12, bottom left image), *Sim-2* (figure 12, top images) and *Sim-3* (figures 6, 7 and 11), are generated test data sets used while developing our software. They were created using a random placement, only controlling that the overlapping of spheres is limited. The sixth data set, *1A2J* (figure 8), is a Disulfide Bond Formation Protein imported from the Protein Data Base as example of the applicability of our method to other kinds of data sets.

Table 1: Rendering Performance of *Vicinity Texture* implementation. The processing time specifies the time needed to build up the required data structure texture (view-independent) using a plane-sweep algorithm.

Data Set	# of Spheres	Processing Time	FPS
Argon	5000	113 ms	0.1
Ethane	25000	2568 ms	0.2
Sim-1	100	$<$1 ms	61.0
Sim-2	500	1 ms	4.0
Sim-3	100	$<$1 ms	4.0
1A2J	1454	12 ms	0.14

Table 1 shows timing results for the *Vicinity Texture* implementation. It is striking that the approach shows an extremely poor performance for data sets with large metaball clusterings as our real-world data sets are. We attribute this to the tremendous number of texture fetches required during the on-the-fly evaluation of the density function. Especially noticeable is the fact that the frame rate for the *Sim-1* data set is 15 times higher than for the *Sim-3* data set, although both consist of 100 spheres. Table 2 points out, why: it shows, how many spheres are potentially part of a metaball, i. e. have a non-empty vicinity groups, and how many are normal spheres. As for the *Sim-3* data set nearly all spheres are potential metaballs, the number of texture accesses is much higher than for *Sim-1*. The most extreme data set is the 1A2J data set, which

Table 2: Number of potential metaball members in the different data sets: Non-empty groups are the spheres which have at least one neighbour which is close enough to form a metaball with. Spheres which can never form a metaball are counted as empty groups. The last column shows the minimum, average and maximum vicinity group size in the data set.

Data Set	Non-empty Groups	Empty Groups	Min/Avg/ Max Size
Argon	3684	1316	1/39.3/125
Ethane	24838	162	1/85.2/178
Sim-1	40	60	1/1.7/4
Sim-2	305	195	1/1.6/7
Sim-3	99	1	2/8.0/17
1A2J	1454	0	2/6.9/13

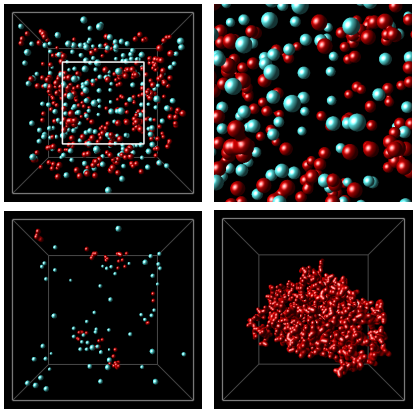


Figure 12: Results of the potential metaball detection during the preprocessing: metaballs and potential metaballs are coloured red, normal spheres cyan. The upper left image shows classification for the 500-element Sim-2 data set, the one right of that shows a close up view of the center area. One can see that actually only very few metaballs are generated. The lower row shows the Sim-1 and the 1A2J data set. In the last-mentioned one, all spheres form one large metaball.

forms one large metaball and therefore has no normal sphere (Figure 12, bottom left image).

One interesting fact about the *Vicinity Texture* approach is that it scales quite well with the image resolution. The 1600×1200 rendering of the *Sim-1* data set reaches the same frame rate as on a 512^2 viewport. We attribute this to the fact that the texture fetching capability of the graphics hardware is not completely utilised for the small number of fragments generated in the small viewport. For the *Sim-2* data set the frame rate drops from 4.0 to 2.0 FPS, because the number of filled pixels, which cause time-consuming texture fetches, increases while texture access posed heavy load on the GPU already for the small picture.

Table 3 shows performance values for the *Walking Depth Plane* approach. Since the approach does not only depend on the complexity of the data set, but also on the maximum number of iterations, we provide rendering performance values for all six data sets using exactly 100 iterations. Increasing or decreasing the maximum number of iterations intuitively decreases or increases the frame rates. No

Table 3: Rendering Performance of *Walking Depth Plane* Implementation for different maximum iteration counts.

Data Set	# of Spheres	Maximum # of Iterations	FPS
Argon	5000	100	13
Ethane	25000	100	5
Sim-1	100	100	61
Sim-2	500	100	15
Sim-2	500	80	19
Sim-3	100	100	15
Sim-3	100	31	61
1A2J	1454	100	21
1A2J	1454	250	11

rendering, except for the rendering of *1A2J* with 100 iterations, created any visual artifacts due to insufficient number of iterations (see figure 10). The *Vicinity Texture* approach can also suffer from typical undersampling artifacts like holes the in metaball surface and from jittering of the surface normals (see figure 11) as we often can afford only very few sampling steps per pixel.

We ran our performance tests on an Intel Core2 Duo 6600 processor with 2.40 GHz, 2 GB memory, and an NVidia GeForce 8800 GTX graphics card with 768 MB graphics memory. The viewport size was 512^2 for all tests.

5 Conclusions

In this paper, we showed how to interactively render metaballs from large particle data sets. We tried two different techniques, of which the first one turned out to perform disappointingly with real-world data sets. The tremendous number of texture accesses required by this approach simply poses a too heavy load on the graphics card, but the evaluation of the density function only for fragments generated by the influence spheres limits the impact of the viewport size after all. As the current linear layout of the vicinity texture is probably unfavourable with regard to caching, it should be investigated whether spatial grouping of the entries can improve the rendering performance.

We described a second approach using multiple rendering passes to approximate the surfaces of the

metaballs. Its rendering performance is quite good on state-of-the-art GPUs, and as it does not require any pre-processing it is even possible to visualise time-dependent data sets on the fly. However, the rendering speed is tightly connected to the viewport size and the method may generate visual artifacts if the number of iterations is insufficient. The current termination criteria and estimation of required iterations cannot effectively prevent these if the size of a molecule changes over time and falls below the pre-computed maximum step size.

Another problem is noticeable in figure 7 when comparing the upper right and lower left images. Although ten iterations lie between these images, the differences are almost indiscernible. We attribute this to the fact that the empty space between the spheres cannot efficiently be skipped without additional effort. Therefore, empty-space-skipping is a feature we want to implement in the future.

We also hope that this and further optimisations can improve the rendering performance to interactively handle data sets consisting of hundred thousands of particles, which are not uncommon in the application area of our software, and to support larger output sizes.

6 Acknowledgements

This work is partially funded by the state of Baden-Württemberg in context of the BW-FIT project *Interaktive Visualisierung für Gigapixel Displays* and by Deutsche Forschungsgemeinschaft (DFG) as part of SFB 716. We want to thank Guido Reina for inspiring discussions on the different approaches.

References

[1] J. F. Blinn. A Generalization of Algebraic Surface Drawing. In *ACM Transactions on Graphics*, 1(3): 235–256, 1982.

[2] C. Everitt. Interactive Order-Independent Transparency. Nvidia Technical Report.

[3] S. Grottel, G. Reina, J. Vrabec, and T. Ertl. Visual Verification and Analysis of Cluster Detection for Molecular Dynamics. In *Proceedings of IEEE Visualization*, 2007, to appear.

[4] S. Gumhold. Splatting Illuminated Ellipsoids with Depth Correction. In *Proceedings of Vision, Modeling, and Visualization 2003*, 245–252, 2003.

[5] T. Klein and T. Ertl. Illustrating Magnetic Field Lines using a Discrete Particle Model. In *Proceedings of Vision, Modeling, and Visualization 2004*, 387–394, 2004.

[6] K. van Kooten, G. van den Bergen and A. Telea. Point-Based Visualization of Metaballs on a GPU. In *GPU Gems 3*. Addison-Wesley, 2007, to appear.

[7] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *International Conference on Computer Graphics and Interactive Techniques*, 163–169, 1987.

[8] Microsoft. *DirectX SDK*. http://msdn.microsoft.com/directx

[9] S. Murakami and H. Ichihara. On a 3D Display Method by Metaball Technique. In *Electronics Communication Conference*, 1607–1615, 1987.

[10] N. Neophytou, K. Mueller. GPU accelerated image aligned splatting In *Volume Graphics*, 197–242, 2005.

[11] H. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirakawa and K. Omura. Object Modeling by Distribution Function and a Method of Image Generation. In *Electonics Communication Conference*, 718–725, 1985.

[12] T. Nishita and E. Nakamae. A Method for Displaying Metaballs by using Bézier Clipping. In *Computer Graphics Forum*, 13(3): 271–280, 1994.

[13] Nvidia. *Direct3D SDK 10 Code Samples*. http://developer.download.nvidia.com/SDK/10/direct3d/samples.html

[14] G. Reina, K. Bidmon, F. Enders, P. Hastreiter, and T. Ertl. GPU-Based Hyperstreamlines for Diffusion Tensor Imaging. In *Proceedings of EUROGRAPHICS - IEEE VGTC Symposium on Visualization 2006*, 35–42, 2006.

[15] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets and M. Gross. Optimized Spatial Hashing for Collision Detection of Deformable Objects. In *Proceedings of Vision, Modeling, and Visualization 2003*, 47–54, 2003.

[16] B. Wyvill, C. McPheeters and G. Wyvill. Data structure for soft objects. In *The Visual Computer*, 2(4): 227–234, 1986.

Sequential Data Compression of Very Large Data in Volume Rendering

Roland Fraedrich*, Michael Bauer†, Marc Stamminger*

* Friedrich-Alexander-University, Erlangen-Nuremberg,
† Fraunhofer Institute for Integrated Circuits, Erlangen-Tennenlohe
Email: {roland.fraedrich, marc.stamminger}@informatik.uni-erlangen.de,
michael.bauer@iis.fraunhofer.de

Abstract

Improving rendering speed in the visualization of very large volumetric data without loss of information is still a challenge. State of the art methods focus on data reduction by omitting content which does not contribute to the final image. In order to further decrease the amount of data, we present a general approach for applying common sequential data compression schemes to blocks of data, which can be decoded on-the-fly by the graphics processing unit (GPU) in a single render pass. Subsequently, the block can be rendered with common axis-aligned texture based rendering techniques. For our implementation, we use Huffman coding as compression scheme utilizing the capabilities of modern graphics hardware to implement the decoding step in a fragment shader. As a result of our approach, up to 3.2 times more data can be transmitted to or held in texture memory of graphics hardware without loss of information.

1 Introduction

Due to the enormous development of commercial graphics hardware in the recent years, today it is possible to render three-dimensional datasets in very high quality at interactive frame rates [8, 12, 20, 25]. However, this applies only to data which completely fits into texture memory (today up to 1.5 GB), so that the full performance of the graphics processing units (GPUs) can be exploited. For large datasets which exceed this limit, the volume has to be divided into smaller *bricks* which, each for itself, fit into video memory. If the whole dataset can not reside in video memory, however, all visible bricks have to be transferred to the graphics hardware for the rendering of every single frame. Due to the bandwidth limitations of the corresponding transmission channel (for PCI Express theoretically up to 4 GB/s), conventional techniques lead to frame rates which are far behind interactivity [3]. Unfortunately, neither the transmission channel nor the graphics memory is expected to increase to such an extent that this problem will be solved. On the contrary, the development of medical imaging devices and the computational power for numerical simulations will lead to datasets of further increasing size.

In the recent years, various approaches have been published to tackle this problem (see Section 2). As the transmission channel is the bottleneck of the rendering pipeline, the general idea is to reduce the amount of data which has to be transferred to the graphics hardware. So far, the primary focus has been on omitting content which does not contribute to the final image. By means of culling and multi-resolution techniques, the amount of data transfer can be significantly decreased, but in most cases these algorithms do not suffice to gain interactive frame rates without accepting loss of information. These approaches, however, reduce the amount of data merely by selecting its relevant portions, but finally transmit those in an uncompressed format. Regarding the various compression techniques which are used in computer science, there seems to be still much potential to further decrease the data amount by using other coding schemes.

Unfortunately, almost any traditional compression technique is not directly implementable on GPU. A major restriction is the sequential order of most CPU-based compression schemes, whereas the parallel architecture of GPUs is optimized for accessing the memory randomly in 2D or 3D. Exceptions are lossy approaches like vector quantization, which would be preferable due to their high compression ratios [22]. However, they comprise the severe problem that even the smallest error can be increased to an arbitrary size by a high-frequent

transfer function. Thus, for controlling this error it is mandatory to consider classification during compression, leading to the handicap that the expensive decoding step has to be performed at runtime.

In this work, we avoid this problem by means of a general lossless approach for applying sequential data compression schemes to bricks of volumetric data, which are decoded and rendered on-the-fly in a two-pass strategy. The basic idea of the decompression pass is that each fragment decodes a stripe of texels along one axis of a single block. Altogether, all slices of this block are decompressed, which can be rendered in a subsequent render pass using common 2D texture based techniques. As only one block is decoded at the same time and the other blocks can be held in video memory in the compressed format, larger datasets can be held on GPU than without compression. For even larger datasets, requiring a continuous data transfer, the compression leads to a higher data throughput. As a proof of concept, we apply Huffman coding on volumetric data and implement its decoding step in a fragment shader by exploiting features of modern graphics hardware. To further improve the compression ratio, we combine decorrelation techniques like predictive coding with the Huffman coder.

The remainder of this paper is structured as follows. After a review of related work in Section 2, we describe the basic idea of applying sequential data compression techniques on volumetric data and our general two-pass strategy for decompression and rendering. In Section 4, we go into details of the implementation of Huffman encoding and decoding. Subsequently, we present some decorrelation techniques for improving the compression ratio. In Section 6, we show and discuss our experimental results. Finally, we present our conclusions and give some ideas of future work.

2 Related Work

While compression in general is an old and well explored area of research, its application on volume data for the purpose of rendering is a rather new field of study. Using various different techniques like wavelet transforms, early publications have focused on the data reduction in main memory and on storage devices [10, 18, 21]. In these approaches, however, the dataset has to be decoded before it is transmitted to the graphics hardware. Thus, the amount of data which has to be transferred to and stored in video memory is unaffected.

One of the first ideas to tackle this particular problem has been *level-of-detail (LOD) rendering* (also known as *multi-resolution rendering*) [13, 24] in which parts of the volume are rendered in lower resolution in order to decrease the amount of data. Based on this concept various approaches have been published, which differ in the division of the volume and the algorithm for selecting an appropriate LOD for each block [5, 7, 15, 16]. In addition, all proposed approaches are combined with some sort of data compression, but again only with the intent of reducing the amount of data in main memory.

The first approach including some kind of decompression on graphics hardware has been texture packing [11, 14]. The basic idea is to reorganize the non-empty content such that it fits into a smaller texture. The original volume is then replaced by one or more indexing textures, in which the packed blocks are referenced. To be more suitable to the transfer function, the sub-blocks can also be grouped in different textures according to their value range instead of their local coherence.

Exploiting the texturing capabilities of modern graphics hardware in a similar fashion, Schneider and Westermann [22] present an implementation of vector quantization on GPU. In order to achieve better quality, the data is divided into 4^3 blocks, which are decomposed into a multi-resolution representation. With this approach, compression factors of up to 64:3 can be achieved, but in contrast to the lossless packing techniques it leads to a significant loss of information. Depending on the transfer function, these errors result in more or less visible artifacts.

Furthermore, there are vector quantization techniques which are directly supported by modern GPUs and which can be specified during texture definition (e.g. S3 texture compression). However, the compression schemes are very simple and the quality is not acceptable for volume rendering.

Similar to vector quantization, we focus on the explicit encoding of volume data. However, by employing lossless compression schemes we avoid its primary problem. Regarding all other publications, our approach could be used in combination with these, as they transmit the volume data in an uncompressed format. Besides the visualization of static data, this applies to many methods for rendering time-varying datasets as well [1, 6, 23].

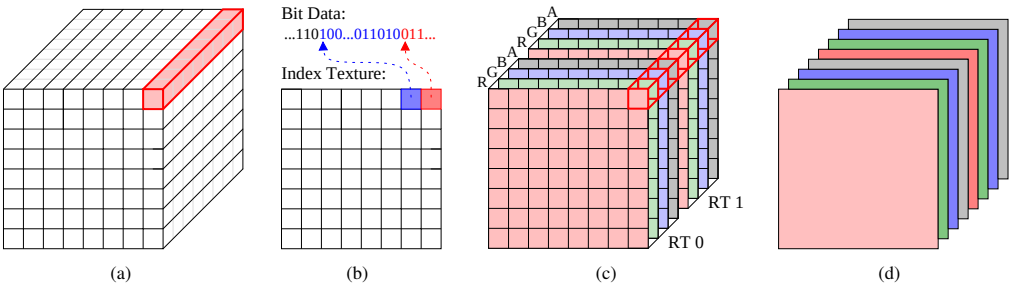


Figure 1: For data encoding, the block is decomposed into parallel stripes of voxels (a). The bit data of the encoded stripes is stored within a single bit stream and an index texture is used for referencing the first bit of each stripe (b). For decoding, a quad is rendered with the size of the index texture and each fragment decodes one stripe of the block. The decoded scalar values are written into different color channels of multiple render targets (c), which are used as textures for volume rendering in a subsequent render pass (d).

In reference to our implementation, the strategy is most similar to deferred filtering [4]. In contrast to its iterated decompression, however, we can decode a whole block in a single render pass. Apart from that, our approach has the same benefits such as continuous reconstruction.

3 General Approach for the Use of Sequential Data Compression Schemes on GPU

Sequential data compression schemes gain high compression factors by packing information into a bit stream, which is encoded and decoded in serial order, e.g. if the end of a codeword determines the beginning of the subsequent one. Modern graphics hardware, on the other hand, owes its performance primarily on its highly parallel streaming architecture including independent processing of fragments within a single render pass. In order to exploit the benefits of modern GPUs as well as of sequential encoding algorithms, we divide every data block (e.g. a brick or a LOD-block) into parallel stripes of n voxels (see Figure 1(a)), which are independently decoded, but for itself in sequential order.

3.1 Encoding, Decoding, and Rendering

During encoding, which is performed only once on CPU in a preprocessing step, the stripes are packed into a single bit array and a *index texture* is used to store the references to the first bit of each stripe (see Figure 1(b)).

For the decoding of the according block on GPU, a quad is rendered having the same size as the index texture. By accessing this texture, each fragment retrieves the reference to the encoded data, from where it can start decoding of the corresponding stripe. As each decoded texel has to be intermediately stored in order to be available for the subsequent render pass, the length n of each stripe is limited by the writing capabilities of the GPU. Since the index structure represents some overhead compared to the original compression scheme, n should be maximized. Assuming that we have to deal with scalar data, we use all distinct color channels of a *frame buffer object* (FBO) with multiple render targets for storing the decoded texels (see Figure 1(c)). Hence, for modern graphics hardware with 8 render targets and 4 color channels, up to 32 texels can be stored, or in other words, up to 32 slices can be decompressed in a single render pass.

In the render pass, the render targets are bound as textures, which can be used for common axis-aligned texture based rendering. The only difference to the standard implementation is an additional selection of the corresponding color channel, as four slices are encoded within a single RGBA-texture. This can simply be implemented by using a uniform variable and a single dot product.

One disadvantage of axis-aligned rendering is that three different stacks of slices have to be available per block in order to reduce artifacts. As our

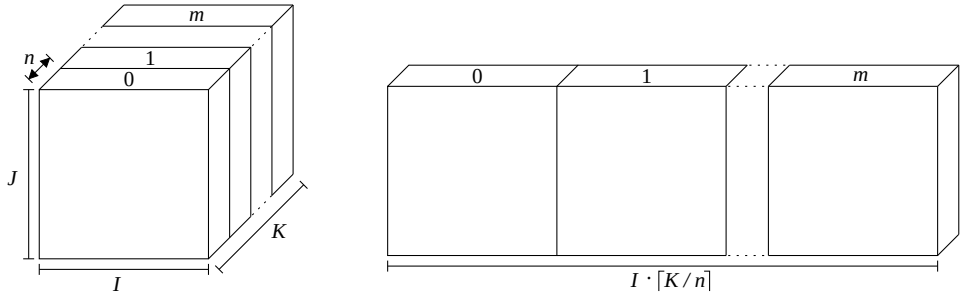


Figure 2: Our approach allows decoding of blocks with size $I \times J \times K$ ($K \leq n$), where n is the number of writable symbols per fragment in a single render pass. However, blocks with $K > n$ can also be decoded at once, if we decompose each into $m = \lceil K/n \rceil$ subblocks of depth n (a) and rearrange the data (b).

scheme is restricted to decode slices along its decomposition axis, this overhead applies to our approach as well. Due to the continuous data transfer, however, this additional amount of data solely affects the main memory, which is significantly larger than the texture memory.

3.2 Handling of Large Blocks

Resulting from our approach, it is possible to decode a volume of any size $I \times J \times K$ (with $K \leq n$) in a single render pass. For a combination with bricking or LOD-rendering this size is usually sufficient. However, the decompression of larger blocks with $K > n$ is possible as well by reordering all texels $(i, j, k) \in (I, J, K)$ as follows:

$$\begin{pmatrix} i \\ j \\ k \end{pmatrix} \rightarrow \begin{pmatrix} \lceil k/n \rceil + i \\ j \\ k \bmod n \end{pmatrix}$$

As can be seen in Figure 2, the rearranged block has a size of $(I \cdot \lceil K/n \rceil, J, n)$, which can directly be encoded and decoded as described above. In the rendering pass, we simply have to adapt the texture coordinates accordingly.

3.3 Data Structures

For the implementation of the decoding step on graphics hardware, the before mentioned data structures have to be mapped to data types of a fragment shader. The index texture can be stored as a 2D texture with dimensions $I \times J$ (or $I \cdot \lceil K/n \rceil \times J$ for rearranged blocks). By means of the novel *gpu_shader4* extension of *OpenGL*, the bit offsets can be directly stored in integer format, with a resolution of either 16 or 32 bits. Since we address single bits, 16 bits suffice for blocks of size L^3 with $L \leq 16$ and 32 bits for $L \leq 512$.

According to the one-dimensional nature of the *bit data*, the best fitting data structure would be a *buffer texture*, a new data structure which is nothing but a simple array. However, in our implementation on a GeForce 8800 GTX (driver: 100.14.11) we have achieved better performance by mapping the bit data on a two-dimensional texture. We use RGBA8-values as internal format, since our experiments have shown that on the given hardware it is the best trade-off between the reduction of texel fetches and the costs for extracting a sequence of bits.

4 Huffman Coding for Texture Compression on GPU

Based on the previously described concept, the decoding step of sequential data compression schemes is generally implementable on graphics hardware. For our experimental implementation, we have decided to use *Huffman coding* [9] as basis for our compression scheme. Huffman coding is the most effective codeword based entropy encoding scheme and is used for many different compression algorithms (e.g. DEFLATE) and popular multimedia codecs (e.g. JPEG and MP3).

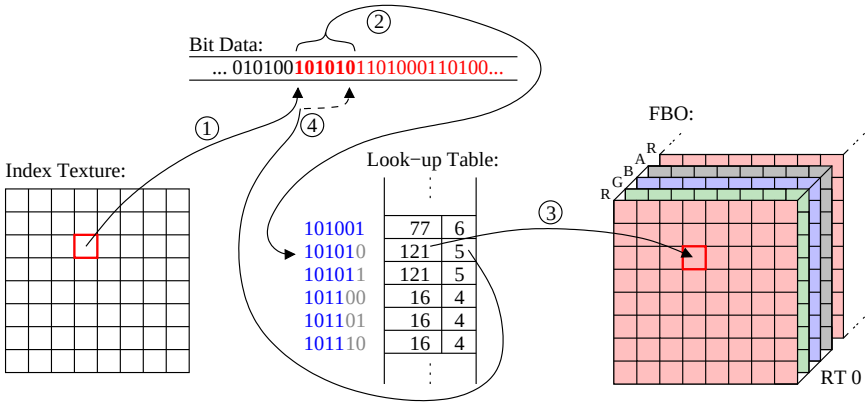


Figure 3: The shader pass for Huffman decoding is performed as follows: First, the pointer to the stripe's bit data (red bits) is initialized by fetching the offset from the index texture. From the current position, the next $l_{\max}$ (e.g. 6) bits are extracted and used as index for accessing the look-up Table (2). The according entry (blue bits represent the codeword) provides the decoded symbol (luminance value), which is stored in the current color channel of the frame buffer object (3), and the length of the decoded codeword (alpha-value), which is used to update the read position of the bit data (4). The steps (2) to (4) are repeated n times until every element of the stripe is decoded and stored in the multiple render targets.

Like any codeword based algorithm, it defines a new alphabet of prefix-free codewords, such that the length of the codes is proportional to the self-information of the encoded symbols. During compression, the symbols of equal length are replaced by the bit-strings of the new alphabet and for decompression the codewords are transferred back to the original values.

4.1 Encoding on CPU

As for any codeword based entropy coding scheme, compression of some given data is divided into three successive steps: Calculation of weights (most usual probabilities) for each symbol, generation of a prefix-free alphabet based on these weights, and finally encoding of data by replacing the original symbols with the bit-strings of the codewords.

In our implementation, the weights are computed by counting the frequency of occurrence for the distinct values in the whole dataset. The prefix-free alphabet is generated by successively creating a binary tree according to the *optimum coding* procedure of Huffman [9]. Finally, each block is separately encoded for the three decomposition axes. The encoding itself is performed stripe by stripe, so that we have a bit stream and a two-dimensional indexing array representing the data as described in Section 3.

Without a doubt, higher compression factors would be obtainable with a local codebook per block. However, as the according information has to be transferred to the graphics hardware as well, it is much more appropriate to use a global one, which can reside in video memory all the time.

4.2 Additional Data Structures

Besides the bit data and the index texture (see Section 3.3), we need information for storing the prefix-free alphabet on the graphics hardware. Due to performance issues, this is done by means of a *look-up table* (LUT), with $2^{l_{\max}}$ entries representing the possible bit string of length $l_{\max}$, where $l_{\max}$ is the maximal length of a codeword in the alphabet. For the creation of the look-up table, the indices of the entries are regarded as bit strings. For each entry the codeword is determined, which is prefix to this bit string. The corresponding symbol and the length of the codeword are stored in the table (see Figure 3). Thus, reading the next $l_{\max}$ bits as index, we can decode the current symbol and determine the bit offset for the read position by a single look-up.

As we have to store two components, the LUT-entries are represented by luminance-alpha values with a bit-resolution according to the given volume data (8- or 16-bit). Note that the look-up table may be rather large. For $2^{l_{\max}}$ entries with 2 color channels and 2 bytes per channel for 16-bit data, it requires $2^{l_{\max}+2}$ bytes. Depending on the actual size of the video memory, it might be better to adapt the generated alphabet in extreme cases, so that $l_{\max}$ is restricted to some predefined value.

4.3 Decoding on GPU

By means of the introduced data structures, Huffman decoding can be directly implemented in a single render pass. For this purpose, a quad with the size of the index texture is rendered into a framebuffer object with multiple render targets, as already described in Section 3.1. The indexing texture is bound as texture to the quad, so that each fragment can read bit data offset of the corresponding stripe. The decoding pass is then performed as follows:

```
init read position of the bit data;
for each element to be decoded {
  read next l_max bits;
  get according LUT-Entry;
  write symbol of LUT-Entry;
  update read position;
}
```

This procedure is illustrated in Figure 3. First, the pointer to the first bit is fetched from the index texture according to the fragment's texture coordinates. After this initialization, the next $l_{\max}$ bits are read from the bit texture according to the bit pointer. This is done incrementally, i.e. in each step only those texels are fetched, which have not been read so far.

The extraction of the index for the look-up table is done by a few shifting, swizzling, and arithmetic operations. With this index as texture coordinate, the according LUT-entry is fetched. Its alpha value is added to the bit pointer in order to update the read position and its luminance value is the decoded symbol. In order to avoid difficult branching, all symbols are stored in a intermediate array and finally written to the render targets at the end of the shader.

5 Improving the Compression Ratio

For improving the compression ratio, we suggest the use of *decorrelation* techniques, i.e. some preprocessing which concentrates the information to a small part of the data whereas rather large parts contain only a low amount of self-information. Due to the characteristics of volumetric data, it seems obvious that this is possible by exploiting the local coherence of neighboring voxels.

For this purpose, we have implemented several predictive schemes. That means, we try to estimate the value of a symbol by some given rule and the information of the previously decoded symbols, and store instead of the original value v_i its difference d_i to a predicted one p_i ($d_i = p_i - v_i$).

In the decoding step, the same rule is applied to predict the value and the difference is added in order to obtain the original value. For appropriate schemes, the differences d_i are in most cases very small and thus, have a frequency distribution which is much better suited for coding schemes like Huffman coding than the original values v_i.

The following schemes have been implemented:

- *Constant Prediction*
  Each voxel is estimated to be have the same value as its predecessor along a single stripe and the divergence is Huffman encoded.

- *Linear Prediction*
  Values are predicted by linear extrapolation of the two previous elements and the differences are compressed using Huffman coding.

- *Least Square Prediction*
  A linear function is computed to approximate the three previous voxel values by the least square method and the function is evaluated for the element, of which the value is estimated. The divergence is encoded using Huffman coding.

- *Low Resolution Approximation*
  Use of a low resolution volume with a quarter of the resolution for each dimension to approximate the data. The differences are encoded using Huffman coding.

One benefit of the suggested schemes is their easy integration into the decoding shader as we just have to add a few additional arithmetic operations.

However, in the former three schemes the first one to three symbols can not be estimated using the according predictor. Thus, we either have to treat these elements as a special estimation case or have to load them as initial data. As the original values and the differences are different kinds of data considering their probability distribution, we suggest to store the first slice as initial data separately in an uncompressed format. In order to avoid an additional texture and an additional look-up, this information can be encoded in the index texture. For predictive schemes, which require more than one previous element, the prediction is simplified for the first values.

Another issue of using predictive schemes is that we have to encode a different alphabet, since a prediction can lead to rational and negative numbers. As a consequence, the new alphabet may have much more elements, which can have negative effects when applying Huffman coding on the data, e.g. if we have to deal with a larger look-up table (see Section 4.2). In order to circumvent this problem, we round any predicted value before computing the difference to the original one. Hence, the original data range of $[0, \max]$ can be at most increased to $[-\max, \max]$. Note that this step does not include any loss of information. We only have to take care, that we round the prediction in the same manner during encoding and decoding.

6 Results

Subsequently, we present performance measurements according to a prototype implementation of our algorithm. As we deal with data compression, we focus on the compression efficiency as well as the times for encoding and decoding. Note that our approach is lossless, so that the image quality is identical to the original dataset. All results have been obtained by using a GeForce 8800 GTX with 768 MB local video memory on a 3.40GHz Intel Pentium 4 HT with 2GB main memory under Linux.

For the validation of our approach and the suggested decorrelation techniques, the different compression schemes have been applied to volumetric data of different modalities (CT, MRI), applications (medicine, industry), and quality (noise). The first dataset is *Bruce Gooch's Brain*, which can be regarded as a rather good MRI scan of a human head. The knee dataset, known from the *Transfer Function Bake-off* [19], is a MRI scan of a human knee containing a high amount of noise. The third dataset is an industrial CT scan of an engine block with two cylinders, which has been taken by General Electric, USA. Like most industrial scans, the examined object does consist of just a few materials – in this case, two materials plus air. The abdomen dataset is a CT scan of a belly in prone orientation in usual image quality, which has been captured at the Walter Reed Army Medical Center, USA. The last dataset is the piggy bank from the Computer Graphics Group at the University of Erlangen, Germany, which is a very clean CT scan containing only a few materials (ceramic piggy bank, wooden plate, and chocolate coins) and a large amount of air. Size and resolution of the datasets are shown in Table 1.

Dataset	Mod. / bits	Resolution	Size
Brain	MRI / 12	$256^2 \times 156$	20 MB
Knee	MRI / 12	$512^2 \times 87$	44 MB
Engine	CT / 8	$256^2 \times 110$	7 MB
Abdomen	CT / 12	$512^2 \times 463$	232 MB
Piggy	CT / 12	$512^2 \times 134$	67 MB

Table 1: Datasets used for experiments.

6.1 Compression Efficiency

The compression factors (uncompressed size / compressed size) for the different schemes and datasets (brick size: 32^3) are shown in Table 2. For the predictive schemes (constant, linear, and least square) the initial slice was stored in an uncompressed format. Due to our focus on reducing the data transfer, the compressed size includes all data but the look-up table, which resides in video memory during rendering.

As can be seen, compression factors up to 3.2 have been achieved depending on the compression scheme and the dataset. Comparing the different decorrelation techniques, constant prediction obtains the highest compression for three of the five datasets. Furthermore, it is near the maximal ratio for the brain dataset and only for the piggy bank significantly smaller than more complex coding schemes. Even if it is the most simple idea, constant prediction seems to cope best with the high-frequent noise of typical datasets. When these discontinuities are less (e.g. if a dataset is smoothed due to visualization purposes), linear and least square prediction yield even higher compression. Besides the

Algorithm	Brain	Engine	Knee	Abdomen	Piggy
Without Decorrelation	2.43	*1.46*	*1.45*	*1.74*	*1.90*
Constant Prediction	2.50	**1.83**	**1.65**	**2.01**	2.84
Linear Prediction	*2.29*	1.67	1.64	1.94	**3.20**
Least Square Prediction	2.36	1.67	1.59	1.92	2.97
Low Resolution Approx.	**2.56**	1.67	1.52	1.91	2.14

Table 2: Compression factors (uncompressed / compressed size) for the different decorrelation techniques and datasets (brick size: 32^3). The highest and lowest compression factor are emphasized (bold / italic).

afore mentioned decorrelation techniques, we have also experimented with wavelet transforms. Without a quantization of data, however, it did not prove to be beneficial.

With reference to the different properties of the datasets, the amount of noise, the portion of air, and the number of different materials have major impact on the compression ratio. Note that the rather low compression factor for the engine dataset is due to its 8-bit resolution compared to the 12-bit precision of all other datasets, which have to be stored with 16-bit as a 12-bit format is not supported by GPU.

6.2 Decoding Performance

Besides the compression ratio, the decoding efficiency is of major importance for our approach. Applying the algorithm without decorrelation or with constant prediction as described above, the decompression rates vary between 35 and 48 MB/s for the different datasets. This is a considerably lower data throughput compared to the bandwidth of PCI Express (usually is up to 1 GB/s in practice), i.e. our approach has a worse performance than bricking without compression. Based on some benchmarking, however, we found out that a major hindrance of performance was the use of all 8 render targets. By using only 4 render targets, which has been already supported by the previous hardware generations, the decoding throughput has been improved to a range of 570 to 750 MB/s for our testing datasets.

Thus, even if the current driver (version: 100.14.11) might not yet be fully optimized regarding the new GPU features, the performance of our algorithm is currently only in the range of the actual transmission bandwidth to the video memory. The major bottleneck of the shader performance is the high number of dependent texture look-ups for Huffman decoding. The decoding of each symbol depends on the previous one in the stripe, so that besides the prediction no operations can be executed until the next LUT-entry is fetched. Hence, the performance is dominated by the memory latency of graphics hardware.

Hence, using Huffman Coding is optimal with respect to codeword based entropy encoding, but not optimal regarding the decoding performance on graphics hardware. In order to achieve decoding performance significantly above the transmission bandwidth to the video memory, there is need for other coding schemes which are more suitable for the implementation on GPU. In addition, optimizations are imaginable to speed up the performance for a part of the stripes, for example by regarding properties of the current transfer function.

6.3 Encoding Performance

The encoding of data has to be applied only once for each dataset. Afterwards it can be held on a server in the compressed format and directly be used for rendering. Thus, the encoding performance is of rather low importance and its implementation was no target of significant optimizations.

Regarding the different prediction schemes (constant, linear and least square) and Huffman without decorrelation, the compression of the data along one stack of slices took up to 5 seconds for the datasets brain and engine, up to 14 seconds for the piggy bank and the knee, and up to 74 seconds for the abdomen. The data throughput (measured by original size / compression time) is in the range of 3.2 to 6.5 MB/s. According to its higher complexity, low resolution approximation had a lower throughput ($1.4 - 4.1$ MB/s) and took up to three times longer than the other compression schemes.

6.4 Comparison to Other Approaches

The comparison to other compression approaches can be seen twofold. On the one hand, we can compare our results with the approach of Schneider and Westermann [22]. Their vector quantization scheme can yield compression factors up to 64:3, which is far beyond our results. However, this compression scheme contains a significant loss of information and thus, cannot be directly compared to our approach.

On the other hand, we can compare our algorithm with lossless CPU-based compression schemes, like *GNU zip*, which is based on Lempel-Ziv coding [26] of bytes, and *bzip*, which is based on the Burrows-Wheeler transform [2] and Huffman coding. The former one compresses Bruce Gooch's brain in about 4.6 seconds to a factor of 2.42. Decompression took only 0.36 seconds, resulting in a data throughput of 56.8 MB per second. The latter one compresses the dataset by a factor of 3.34 within 6.4 seconds. The time required for decoding is 2.6 seconds, resulting in a throuput of 18.1 MB/s. Compared to these results, our algorithm outperforms both schemes regarding the data throughput for decompression and achieves in addition higher compression factor (2.50) than GNU zip.

Nevertheless, even better compression ratios would definitely be possible by means of algorithms which are specialized on grey-level images (e.g. *Glicbawls* [17]) or images in general (e.g. *JPEG2000*). However, these compression schemes are highly specialized and far too complex for an implementation on graphics hardware for the purpose of interactive rendering.

7 Conclusion and Future Work

In this paper, we presented a novel approach for assigning sequential data compression schemes to volumetric data for the purpose of volume rendering. The general algorithm uses the capabilities of modern graphics hardware to decode blocks of data in a single render pass. The decoded data is stored in multiple render targets, which can be used as textures for rendering in a subsequent render pass. Our experimental implementation of using Huffman coding along with prediction schemes has gained compression factor up to 3.2, which can be used to decrease the transmission time for blocks as well as to increase the amount of information which can be held in the video memory.

The decoding throughput was in the range of the transmission rate of PCI Express in practice. Hence, our experimental implementation represents so far no better alternative. However, as LOD-rendering is quite a well-explored field of research and real data compression has been so far restricted to vector quantization, we believe that our general approach represents a new possibility to implement various compression schemes on GPU. With Huffman coding, representing the optimal codeword based entropy encoding scheme, we have shown what compression factors are possible without loss of information and how expensive it is to achieve such ratios.

Due to this results, future work will be twofold. On the one hand, we will focus on increasing the performance of our approach. This includes the development of better suited coding schemes as well as additional optimization, e.g. by considering the transfer function. On the other hand, there is need to develop schemes which achieve higher compression ratios. This will not be possible without loss of information. In contrast to vector quantizations, however, it is possible to keep the error of quantization below some given threshold. Alternatively, it may be possible to predefine some properties of the transfer function which are beneficial for encoding the data, e.g. to define a data range associated with air.

In summary, we believe that this approach provides various possibilities to find an optimal tradeoff between quality and rendering speed. Thus, it is an optimal complement to the various multiresolution and texture packing techniques, which have so far transferred data in an uncompressed manner.

References

[1] A. P. D. Binotto, J. Comba, and C. M. Dal Sasso Freitas. Real-Time Volume Rendering of Time-Varying Data Using a Fragment-Shader Compression Approach. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics,* 69–76, 2003.

[2] M. Burrows and D. J. Wheeler. A Block-Sorting Lossless Data Compression Algo-

rithm. Technical Report, Digital Equipment Corporation, Palo Alto, CA, 1994.

[3] K. Engel, M. Hadwiger, J. M. Kniss, C. Rezk-Salama, D. Weiskopf. *Real-Time Volume Graphics*, 2006.

[4] N. Fout, H. Akiba, K.-L. Ma, A. Lefohn, and J. M. Kniss. High-Quality Rendering of Compressed Volume Data Formats. In *Proceedings of EuroVis 2005*, 77–84, 2005.

[5] S. Guthe and W. Straßer. Advanced Techniques for High-Quality Multi-Resolution Volume Rendering. *Computers & Graphics*, 28(1):51–58, 2004.

[6] S. Guthe and W. Straßer. Real-Time Decompression and Visualization of Animated Volume Data. In *IEEE Visualization*, 349–356, 2001.

[7] S. Guthe, M. Wand, J. Gonser, and W. Straßer. Interactive Rendering of Large Volume Data Sets. In *IEEE Visualization*, 53–59, 2002.

[8] M. Hadwiger and C. Sigg and H. Scharsach and K. Bühler and M. Gross. Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces, In *Proceedings of Eurographics '05*, 303–312, 2005.

[9] D. A. Huffman. A Method for the Construction of Minimum Redundancy Codes. *Proceedings of the IRE*, 40(10):1098–1101, 1952.

[10] I. Ihm and S. Park. Wavelet-Based 3D Compression Scheme for Interactive Visualization of Very Large Volume Data. *Computer Graphics Forum*, 18(1):3–15, 1999.

[11] M. Kraus and T. Ertl. Adaptive Texture Maps. In *Proc. SIGGRAPH/EG Graphics Hardware Workshop '02*, 7–15, 2002.

[12] J. Krüger and R. Westermann. Acceleration Techniques for GPU-based Volume Rendering. In *Proceedings IEEE Visualization 2003*, 38–44, 2003.

[13] E. LaMar, B. Hamann, and K. I. Joy. Multiresolution Techniques for Interactive Texture-Based Volume Visualization. In *IEEE Visualization*, 355–36, 1999.

[14] W. Li and A. E. Kaufman. Texture Partitioning and Packing for Accelerating Texture-based Volume Rendering. In *Graphics Interface*, 81–88, 2003.

[15] X. Li and H.-W. Shen, Time-critical Multiresolution Volume Rendering using 3D Texture Mapping Hardware. In *VolVis*, 29–36, 2002.

[16] P. Ljung, C. Lundstrom, A. Ynnerman, and K. Museth. Transfer Function Based Adaptive Decompression for Volume Rendering of Large Medical Data In *VolVis*, 25–32, 2004.

[17] B. Meyer and P. E. Tischer. Glicbawls - Grey Level Image Compression by Adaptive Weighted Least Squares. In *Data Compression Conference*, 503 ff, 2001.

[18] K. G. Nguyen and D. Saupe. Rapid High Quality Compression of Volume Data for Visualization. *Computer Graphics Forum*, 20(3):49–56, 2001.

[19] H. Pfister, W. E. Lorensen, C. L. Bajaj, G. L. Kindlmann, W. J. Schroeder, L. Sobierajski Avila, K. Martin, R. Machiraju, and J. Lee-The Transfer Function Bake-Off. *IEEE Computer Graphics and Applications*, 21(3):16–22, 2001.

[20] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl, Interactive Volume Rendering on Standard PC Graphics Hardware using Multi-Textures and Multi-Stage Rasterization. In *HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, 109–118, 2000.

[21] F. F. Rodler. Wavelet Based 3D Compression with Fast Random Access for Very Large Volume Data. In *Pacific Conference on Computer Graphics and Applications,* 108–117, 1999.

[22] J. Schneider and R. Westermann. Compression Domain Volume Rendering. In *IEEE Visualization*, 293–300, 2003.

[23] B.-S. Sohn, C. L. Bajaj, and V. Siddavanahalli. Volumetric Video Compression for Interactive Playback. *Computer Vision and Image Understanding*, 96(3):435–452, 2004.

[24] M. Weiler, R. Westermann, C. D. Hansen, K. Zimmerman, and T. Ertl. Level-of-Detail Volume Rendering via 3D Textures. In *Volume Visualization and Graphics Symposium 2000*, 7–13, 2000.

[25] R. Westermann and T. Ertl. Efficiently using Graphics Hardware in Volume Rendering Applications. In *ACM SIGGRAPH '98*, 169–177, 1998.

[26] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 23(3), 337–343, 1977.

GPU Rendering of Secondary Effects

Kai Bürger, Stefan Hertel, Jens Krüger and Rüdiger Westermann

Technische Universität München
Email: {buergerk,hertel,jens.krueger,westermann}@in.tum.de

Abstract

In this paper we present an efficient data structure and algorithms for GPU ray tracing of secondary effects like reflections, refractions and shadows. Our method extends previous work on layered depth cubes in that it uses layered depth cubes as an adaptive space partitioning scheme for ray tracing. We propose a new method to efficiently build LDCs on the GPU using geometry shaders available in Direct3D 10. The overhead of peeling the scene multiple times can thus be avoided. We further show that the traversal of secondary rays is greatly accelerated by exploiting a two level hierarchy and the adaptive nature of the LDC. Due to the computational and bandwidth capacities available on recent GPUs our method enables high-quality rendering of static *and* dynamic scenes at interactive rates.

1 Introduction and Related Work

Since the early years of computer graphics there has been interest in ray tracing due to its potential for the accurate rendering of complex light phenomena. Over the last few years, there was an ever growing interest due to the observation that interactive ray tracing can now be achieved on custom hardware [8, 23, 22], or by using a cluster of custom computers [19, 26]. Recent advances in hardware and software technology, including specialized ray tracing chips [25] as well as advanced space partitioning and traversal schemes [27, 30], have even shown that ray tracing is potentially suited for real time applications like computer games and virtual environments.

Simultaneously, considerable effort has been put into the implementation of ray tracing on programmable graphics hardware. Inspired by the early work of Purcell et al. [21] and Carr et al. [2], in a number of succeeding implementations it was shown that the capabilities of recent GPU stream architectures including parallelism across stream elements and low-latency memory interfaces can effectively be used for ray tracing [17]. While these approaches were solely based on uniform space partitioning schemes, recent work has also demonstrated the possibility to build and traverse adaptive spatial hierarchies on the GPU [15]. Hereafter, Foley and Sugerman [5] as well as Popov et al. [20] independently examined stack operations—a feature not well-supported by the GPU—and they reported a significant performance gain by using a stackless traversal algorithm for kd-trees. Alternatively, Carr et al. [3] represented surfaces as geometry images and introduced linked bounding volume hierarchies to avoid conditionals and stack operations. By taking advantage of the GPU to construct these hierarchies, for the first time the authors could demonstrate real-time GPU ray tracing of dynamic scenes.

Despite all the advancements in GPU ray tracing, including efficient approximations for ray-object intersection using pre-computed environment imposters [14] and ray-object penetration depths [31], it can still not be denied that high quality ray tracing using optimized CPU codes performs favorable or even faster than many GPU implementations. The main reason why rasterization hardware is not perfectly suited for ray tracing is the inability of current GPUs to efficiently determine ray-object intersections for rays others than view rays. This makes it difficult to accurately simulate secondary effects like reflections, refractions, and shadows, as such effects require parts of the scene to be rendered multiple times under different projections, i.e. onto different receivers. Although possible in principle, it was shown by Wand and Straßer [28] for the rendering of underwater caustics that this approach is not practicable in general.

On the other hand it can be observed that a significant part of the render time in typical 3D applications is shading, and it is well accepted that the GPU outperforms the CPU in this respect. Appar-

Figure 1: Method demonstration: Cubemap reflections (left), our method (middle) and software ray tracing (right). On a single Geforce 8800 GTX our method renders the scene into a 1280x1024 image at 3 fps.

ently, due to the GPUs inability to effectively exploit adaptive space partitioning schemes this advantage is entirely amortized in ray tracing. Our motivation is thus to overcome GPU limitations in finding ray-object intersections, at the same time exploiting the intrinsic strength of these architectures to shade billions of fragments in real-time.

1.1 Contribution

The main contribution of this paper is a new GPU approach for ray tracing of secondary effects like reflections, refractions and shadows. This is achieved by using an adaptive spatial data structure at extreme resolution, and by providing novel methods to construct and traverse this data structure on the GPU. Just as proposed by Lischinski and Rappoport [13], our data structure represents the scene as a set of layered depth images (LDI) [24, 16, 6] along three orthogonal projections. According to Lischinski and Rappoport [13] we will refer to this structure as the layered depth cube (LDC). We extend LDCs in the following ways:

- LDIs are constructed via depth peeling [4], but we employ Direct3D 10 functionality so that the scene only has to be peeled once to generate LDIs along multiple viewing directions. To accelerate the LDC construction every polygon is rendered only into the LDI capturing the scene from the direction most perpendicular to this polygon (see Figure 5). To accommodate deferred shading the LDC stores not only fragment depth, but also interpolated colors and texture values as well as normals. The process leads to a view independent scene representation using a minimal number of samples.
- For each LDI a two level hierarchy is built. This method is similar to the empty space skipping structure used by Krüger and Westermann [11]. Entries in this low resolution representation of one LDC direction store for each bundle the minimum and maximum distances from a reference plane perpendicular to the direction of projection.
- We present an efficient ray-object intersection test using LDCs. As in each LDI the samples lie on a 2D raster we perform ray traversal in these rasters alternatively. This method is similar to screen-space ray tracing described by Krüger et al. [10], but it has a major advantage: The hierarchical representation is used to skip regions in these rasters not containing any structures a ray could intersect with.

As both the LDC construction and the traversal are performed on the GPU the rasterization capacities of recent architectures can effectively be exploited. In particular, since our approach does not require any pre-process to modify the initial scene representation it can be used in the same way to render dynamic scenes or scenes created or modified on the GPU. Some results of our approach together with a comparison to cubemap reflections and software ray tracing are shown in Figures 1 and 2.

The remainder of this paper is organized as follows: In the next chapter we will discuss the LDC construction in particular the use of Direct3D 10 features to speed up this process. We will then describe how this structure can be efficiently traversed on the GPU. Next, we explain the integration of the LDC construction and the ray traversal into the rendering algorithm. Finally, we analyze the perfor-

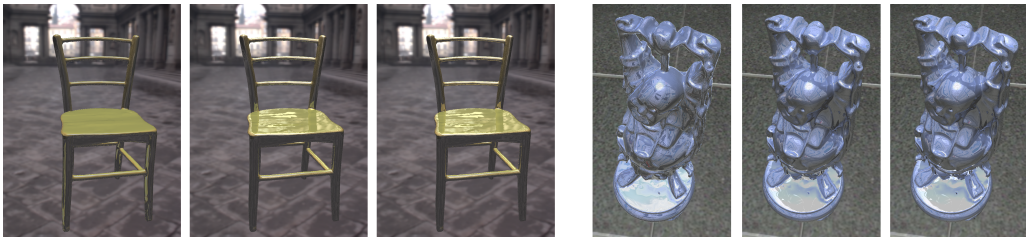


Figure 2: Method demonstration: Cubemap reflections (left), our method (middle) and software ray tracing (right). On a single Geforce 8800 GTX our method renders the scene into a 1280x1024 image at 5 and 3 fps, respectively.

mance of the major components of our system, and we conclude the paper with some remarks about future research in this field.

1.2 LDC Construction

To efficiently construct LDCs on the GPU we first employ depth peeling [4] to generate LDIs along three orthogonal viewing directions. Depth-peeling requires multiple rendering passes. For each pixel, in the n-th pass the $(n-1)$-th nearest fragments are rejected in a fragment program and the closest of all remaining fragments is retained by the standard depth test. A floating point texture map—the depth map—is used to communicate the depth of the surviving fragments to the next pass. The number of rendering passes is equal to the objects depth complexity, i.e. the maximum number of object points falling into a single pixel. This number is determined by rendering the objects once and by counting at each pixel the number of fragments falling into it during rasterization. The maximum over all pixels is then collected in a log-step reduce-max operation [12].

Although more efficient depth peeling variants exist, for instance the method proposed by Wexler et al. [29] showing linear complexity in the number of polygons compared to quadratic complexity of standard depth peeling, in the current work we favor the more complex approach. This is because it does not require any pre-processing and thus can be used for the processing of dynamic scenes and scenes modified or generated on the GPU.

1.2.1 Construction using Geometry Shader

In this chapter we describe how to efficiently generate an LDC that captures the entire scene. From the text it should become clear that the same approach can of course be used to generate LDCs for separate objects in the scene. In particular for rigid objects this allows us to pre-compute the LDC once and to exclude it from depth peeling in successive frames. LDC construction greatly benefits from latest graphics APIs and hardware as explained in the following paragraphs.

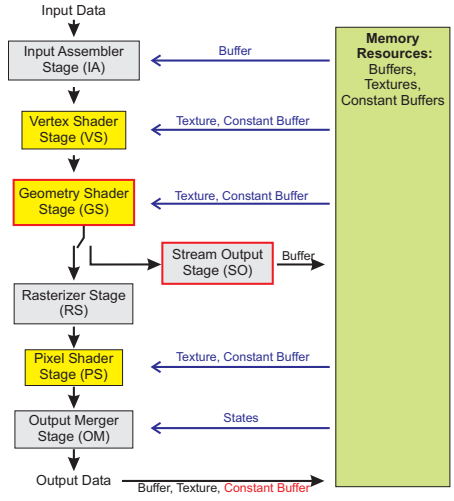


Figure 3: The Direct3D 10 rendering pipeline. Programmable stages are drawn in yellow. Note in particular the new programmable Geometry Shader stage after the Vertex Shader.

One of the key novelties of Direct3D 10 capable hardware [1] is a new programmable stage in the rendering pipeline. This stage—the Geometry

Shader—is placed directly after the Vertex Shader stage (see Figure 3). In contrast to the Vertex Shader the Geometry Shader takes as input an entire graphics primitive (e.g. a triangle or a triangle and its neighbors) and outputs zero to multiple new primitives. This is achieved by letting the Geometry Shader append the new primitives to one or multiple output streams. The LDC construction algorithm makes use of this feature and binds multiple output streams called render targets (MRTs) to the Geometry Shader. Note that these MRTs are different from the ones known in the Pixel Shader stage.

While MRTs in the Pixel Shader are used to output multiple color values at the very end of the pipeline, to each Geometry Shader MRT its own rasterizer, Pixel Shader stack, depth and color buffers are associated. One can even assign another stack of Pixel Shader MRTs to each of these separate pipelines (see Figure 4).

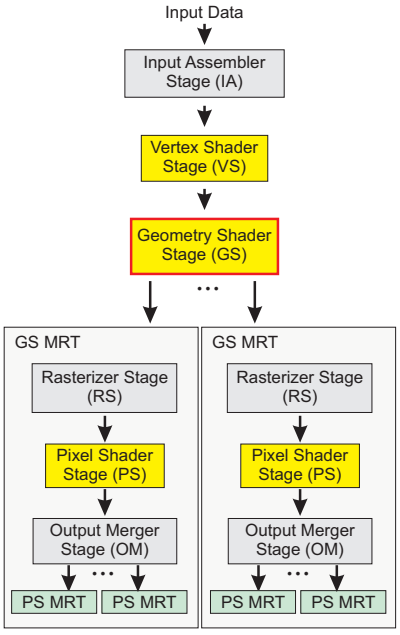


Figure 4: This figure depicts the difference between MRTs used in the Geometry Shader and in the Pixel Shader stage.

Without the functionality provided by the Geometry Shader LDC construction requires the entire scene to be rendered three times. By using the Geometry Shader MRTs we only need to process and rasterize the scene once. This is achieved by letting the Vertex Shader perform the geometry transformation excluding the viewing transformation. Transformed vertices are then combined to triangles and sent to the Geometry Shader stage. Here, face normals are computed for every triangle and compared against the three directions along which peeling is performed. Triangles are then rasterized into the MRT which captures those parts of the scene most perpendicular to its projection direction (see Figure 5). In this way every triangle has to be processed and rasterized only once. This method not only reduces GPU load to one third, but it also reduces the depth complexity along all three directions since lesser primitives appear in either target.

Furthermore, as no primitive is rasterized into more than one MRT the process avoids storage of redundant samples in different projection directions thus leading to an optimal view independent scene representation. As mentioned earlier, each Geometry Shader MRT pipeline can have multiple Pixel Shader MRTs at its very end (see Figure 4), we use this additional possibility to capture multiple values at once. In particular we use two Pixel Shader MRTs, one target to store the normal and depth information of the fragments and a second target for texture and color information. In the final rendering step we use the later values to perform deferred shading of the reflected and refracted surfaces.

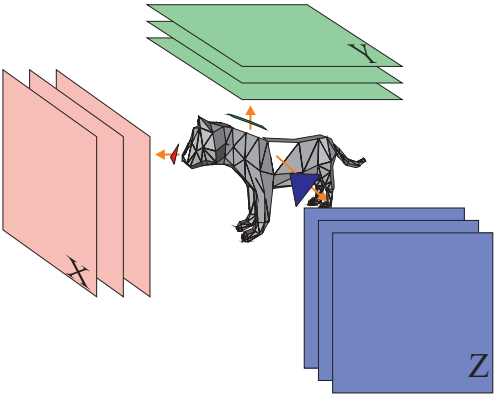


Figure 5: This figure shows how triangles are projected into only one LDI stack depending on their face normal orientation.

By using the Geometry Shader MRTs the complexity of our algorithm to construct the LDC is reduced to $\max(\text{depthComplexity})$ passes. How-

ever, with the advent of layered depth/color buffers at the end of the rendering pipeline we expect our construction procedure to become even more efficient. For a long time such a functionality is present on GPUs (e.g. ATI/AMDs FBuffer technology [9]) though it has not yet been exposed. On the other hand we perceive an ever growing demand for depth peeling in a number of applications ranging from order independent transparency [4] and CSG rendering [7] to volume rendering applications [18] and real-time rendering [10]. As a consequence we expect graphics APIs to support these features in the foreseeable future. Then, the generation of LDCs could even be performed in one single rendering pass.

1.3 Two Level LDC

An LDC exhibits an extremely adaptive and compact representation of the scene, yet it lacks the hierarchical nature of other space partitioning schemes such as octrees or kd-trees. It is on the other hand well known that such schemes can greatly accelerate ray tracing in that they allow for an efficient determination of ray-object intersections.

To generate a two-level LDC we first compute the LDC as described above. Then a single low resolution data structure is generated for every LDI. This structure is built only on the depth values whereas the the normal and texture stacks remain unchanged. For the generation of an $n \times m$ reduced empty space skipping texture we employ a custom pixel shader, that performs a max/min computation on a grid of $n \times m$ fragments from all LDI layers in one direction (see Figure 6). Such a texture can be seen as an axis aligned bounding box, which will be used later on to efficiently skip empty space in the scene.

Equipped with the acceleration texture as described we will now show how to exploit this data structure for GPU ray tracing.

1.4 Ray Traversal

To traverse rays through a two level LDI we employ an approach similar to the one proposed by Krüger and Westermann [11]. We perform a modified DDA algorithm to find the coarse level grid cells intersected by the ray. Therefore the ray is tested against the depth range stored in the cells of the acceleration texture being hit. If an intersection is found we

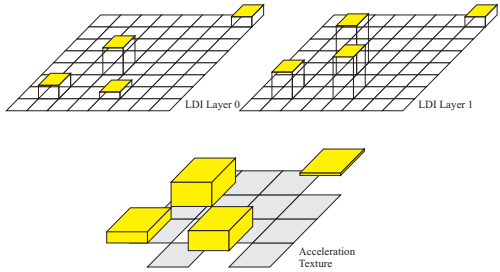


Figure 6: The image shows how two LDI layers (top) are combined into a single acceleration texture (bottom). This texture effectively stores bounding boxes around all LDI layers for a given $n \times m$ grid in this case 2×2.

step down to the LDIs and use the same DDA algorithm within the $n \times m$ block. If an intersection with one of the values stored in the LDIs is found we terminate the ray traversal, otherwise we continue at the next block in the coarser empty space skipping texture.

In the way described we subsequently process one peel direction after another. Note that if an intersection in one direction is found this intersection does not necessarily has to be the first intersection along the ray. However, in this case we can change the end point of the ray to the position of that intersection, continually reducing the remaining distance to be considered along the ray. After all layers have been traversed the intersection point between the ray and an object in the scene is found, or just the intersection with the LDCs bounding box. This information is used later on for shading the reflected sample point.

2 Rendering

After having described both the two level data structure used to represent a scene on the GPU and an efficient ray traversal scheme for this data structure we will now present the rendering algorithm that finally generates an image of the scene. This algorithm renders the scene in three passes. In the first pass—which itself consists of multiple subpasses—the LDC capturing the scene is generated. In the second pass “primary ray-object intersections” are determined by rasterizing the scene under the current viewing transformation. In the third

pass the secondary rays are traced in the LDC to simulate reflections, refractions and shadows.

While passes one and two are clear, pass three needs some further explanations.

2.1 Ray Tracing and Shading

After the second pass the scene as seen from the current view point, appropriately shaded but without any secondary effects, has been generated. In this image we need to find for every specular receiver the points in the scene from where to receive an additional radiance contribution. Therefore we proceed in two steps. In the first step we render reflective/refractive objects by employing a pixel shader program that computes for every fragment the following values:

- the reflection vector
- the intersection of this vector with the LDC
- the parallel projection of the reflection vector into the three orthogonal LDIs

The later two quantities are stored in three render targets, of which we use the first to store the intersection point and the remaining two to store projected directions (see Figure 7). In the second step we use these values to perform the LDC traversal as described in the previous section. This results in either an intersection point with an object in the scene or with the scene's bounding box. In the latter case we simply perform a lookup into an environment map to compute the color of the reflection. In the former case we lookup the normal and the color generated for the intersected point in the LDC construction, and we use this information to shade the reflected point. Finally, the color seen along the primary rays is combined with the color of the secondary ray and rendered into the color buffer.

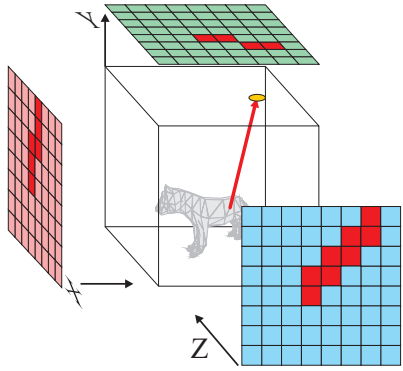


Figure 7: This image illustrates the four values generated prior to the ray traversal. Firstly, the intersection point between the reflected ray and the bounding box yellow) is computed. Next, the ray is projected into the three orthogonal LDIs (red, green, blue).

3 Results

We have tested the proposed GPU ray tracing technique in a number of different scenarios consisting of several thousands up to hundreds of thousands triangles. Images of such scenes together with ground truth images generated by a software ray tracer are given in Figures 1 and 2. At the end of this chapter some additional examples including shadowing and refractions are shown. In contrast to the first four examples, in the fifth example a dynamic object is rendered using vertex shader skinning. In this case the LDC is constructed on-the-fly in every frame of the animation (see Figure 10).

All of our tests were run on a single core Pentium 4 equipped with a NVIDIA Geforce 8800 GTX graphics card with 768 MB local video memory. In all of our tests $1K \times 1K$ LDIs were used to sample the objects along three orthogonal viewing directions. Note that this corresponds to a resolution of the spatial data structures of $1K^3$. However, as we store 16 Bit floating point depth values in every LDI this resolution is in fact significantly higher. As we only capture those areas in space where some objects are present, our approach requires considerably less memory than other techniques using a uniform grid structure to represent the scene.

All objects are encoded as indexed vertex arrays stored in GPU memory. In all our experiments an intersection was determined if the distance between a point on the secondary ray and a fragment coded in the LDC was less than 0.001 in world space. This tolerance was also used in all other examples throughout this paper. It can be seen in all our examples that the scene is adequately sampled by the LDC and GPU ray tracing produces almost exactly the same results as the software ray tracer running in double floating point precision on the CPU.

Representative timings in milliseconds (ms) for GPU ray casting of secondary effects in the four example scenes are listed in Table 1. The number of

LDIs required to capture the scenes adequately are 8, 12, 7, 6 and 6 respectively. The values in columns labeled (A) show the amount of time spent by the GPU for LDC construction. Columns labeled (B) show the time spent for ray tracing including rendering of the final result. Performance was measured using LDIs at 512×512 and $1K \times 1K$ resolution. All tests were rendered into a 1280×1024 viewport (see Figures 8 to 10) .

Figure 8: Two reflective teapots above a mirroring plane and the EG07 Phlegmatic Dragon with a reflective surface. Note the reflection of the dragons face on its nose.

As the timings show, by means of the proposed technique secondary effects in very complex scenes can be simulated at interactive rates and convincing quality. In particular it can be observed that the

	LDI resolution			
	512 x 512		1024 x 1024	
	A	B	A	B
Bunny (8192 tris)	7	61	19	115
Car (20264 tris)	9	82	25	250
Dragon (120K tris)	12	205	31	441
Max Planck (300K tris)	19	160	53	346
Tiny (animation) (1628 tris)	5	44	16	95

Table 1: Timing statistics (in ms) for different scenes.

LDC construction is fast enough to allow for on-the-fly capturing of reasonable scenes. This property makes it possible to render dynamic objects at high frame rates and quality.

4 Conclusions and Future Work

In this work we have described a technique for GPU ray tracing of secondary effects. By using a view-independent, two level LDI representation in combination with an adaptive ray traversal scheme on the GPU interactive rendering of such effects is possible. We have shown how to construct LDCs efficiently on the GPU by exploiting recent functionality on Direct3D 10 capable graphics hardware together with the intrinsic strength of these architectures to shade millions of points in real-time. As our timings indicate, the proposed techniques enable interactive ray tracing of complex scenes at high accuracy. In comparison to software ray tracing visual artifacts are marginal.

Due to the efficiency of the LDC construction it is in particular possible to integrate this process into the rendering pass. This enables the rendering of dynamic objects without any additional modifications of the proposed algorithm. As the construction process only requires objects to be available in any renderable format our method can also deal with objects being modified or constructed on the GPU.

In the future, we will investigate how to further improve the performance of the proposed rendering technique. In particular we will focus on the problem how to effectively exploit the internal RGBA pipeline on current GPUs. In principle this can be done in two ways: Firstly, by storing four LDIs into one RGBA texture target and thus by providing the

possibility to trace every ray in four depth layers simultaneously. Secondly, by tracing four rays simultaneously in one fragment program. As both optimizations are greatly influenced by the ability of recent GPUs to effectively exit shader programs, a detailed analysis of the performance gains for different scenes and architectures is required. Furthermore, we plan to compare our two-level acceleration structure to fully octree or kd-tree hierarchies.

5 Acknowledgments

The authors wish to thank the AIM@SHAPE project and all of its partners for providing most of the meshes used in this paper as well the Academy of Sciences of the Czech Republic, and CGG, Czech Technical University in Prague for providing the Phlegmatic Dragon data set.

References

[1] David Blythe. The direct3d 10 system. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 724–734, New York, NY, USA, 2006. ACM Press.

[2] N. Carr, J. Hall, and J. Hart. The ray engine. In *Proceedings of the ACM SIGGRAPH /EUROGRAPHICS conference on Graphics hardware*, pages 37–46, 2002.

[3] Nathan A. Carr, Jared Hoberock, Keenan Crane, and John C. Hart. Fast gpu ray tracing of dynamic meshes using geometry images. In *GI '06: Proceedings of the 2006 conference on Graphics interface*, pages 203–209, 2006.

[4] Cass Everitt. Interactive order-independent transparency. Technical report, NVIDIA White papers, 2001.

[5] Tim Foley and Jeremy Sugerman. Kd-tree acceleration structures for a gpu raytracer. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 15–22, 2005.

[6] Steven J. Gortler, Li-Wei He, and Michael F. Cohen. Rendering layered depth images. Technical Report Technical Report MSTR-TR-97-09, Microsoft Research, Redmond, WA, 1997.

[7] John Hable and Jarek Rossignac. Blister: Gpu-based rendering of boolean combinations of free-form triangulated shapes. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 1024–1031, 2005.

[8] Daniel Hall. The ar350: Today's ray trace rendering processor. SIGGRAPH / Eurographics Workshop On Graphics Hardware - Hot 3D Session, 2001.

[9] Mike Houston, Arcot Preetham, and Mark Segal. A hardware f-buffer implementatio. Technical report, Stanford University Computer Science Technical Reports, 2006.

[10] Jens Krüger, Kai Bürger, and Rüdiger Westermann. Interactive screen-space accurate photon tracing on GPUs. In *Rendering Techniques (Eurographics Symposium on Rendering - EGSR)*, pages 319–329, June 2006.

[11] Jens Krüger and Rüdiger Westermann. Acceleration Techniques for GPU-based Volume Rendering. In *Proceedings IEEE Visualization 2003*, 2003.

[12] Jens Krüger and Rüdiger Westermann. Linear algebra operators for GPU implementation of numerical algorithms. *ACM Transactions on Graphics (TOG)*, 22(3), 2003.

[13] Dani Lischinski and Ari Rappoport. Image-based rendering for non-diffuse synthetic scenes. In *Proceedings, Ninth Eurographics Workshop on Rendering*, pages 301–314, 1998.

[14] Szirmay-Kalos Lszl, Aszdi Barnabs, Laznyi Istvn, and Premecz Mtys. Approximate ray-tracing on the GPU with distance impostors. *Computer Graphics Forum*, 24(3), 2005.

[15] Günther Greiner Manfred Ernst, Christian Vogelgsang. Stack implementation on programmable graphics hardware. In *Vision Modeling and Visualization*, pages 255–262, 2004.

[16] Nelson Max. Hierarchical rendering of trees from precomputed multi-layer z-buffers. In *Proceedings of the eurographics workshop on Rendering techniques '96*, pages 165–174, London, UK, 1996. Springer-Verlag.

[17] Gabo Moreno-Fortuny and Michael McCool. Unified stream processing raytracer. Poster at GP2: The ACM Workshop on General Purpose Computing on Graphics Processors, 2004.

[18] Zoltan Nagy and Reinhard Klein. Depth-peeling for texture-based volume rendering. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applica-*

tions, 2003.

[19] W. Martin Parker, P.-P. Sloan, P. Shirley, B. Smits, and C. Hansen. Interactive ray tracing. In *Symposium on Interactive 3D Computer Graphics*, pages 119–126, 1999.

[20] Stefan Popov, Johannes Günther, and Philipp Slusallek. Stackless kd-tree traversal for high performance gpu ray tracing. In *Proceedings of EUROGRAPHICS annual conference*, 2007.

[21] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, 21(3):703–712, July 2002.

[22] J. Schmittler, S. Woop, D. Wagner, W.J. Paul, and P. Slusallek. Realtime ray tracing of dynamic scenes on an fpga chip. In *Graphics Hardware 2004*. Eurographics Association, 2004.

[23] Jörg Schmittler, Ingo Wald, and Philipp Slusallek. SaarCOR: a hardware architecture for ray tracing. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 27–36, 2002.

[24] Jonathan Shade, Steven Gortler, Li wei He, and Richard Szeliski. Layered depth images. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242, 1998.

[25] Jörg Schmittler Sven Woop and Philipp Slusallek. Rpu: A programmable ray processing unit for realtime ray tracing. In *Proceedings of ACM SIGGRAPH 2005*, July 2005.

[26] Ingo Wald, Carsten Benthin, and Philipp Slusallek. Distributed interactive ray tracing of dynamic scenes. In *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics (PVG)*, 2003.

[27] Ingo Wald, Thiago Ize, Andrew Kensler, Aaron Knoll, and Steven G Parker. Ray Tracing Animated Scenes using Coherent Grid Traversal. *ACM Transactions on Graphics*, pages 485–493, 2006. (Proceedings of ACM SIGGRAPH 2006).

[28] Michael Wand and Wolfgang Straßer. Real-time caustics. In P. Brunet and D. Fellner, editors, *Computer Graphics Forum*, volume 22(3), 2003.

[29] Daniel Wexler, Larry Gritz, Eric Enderton, and Jonathan Rice. Gpu-accelerated high-quality hidden surface removal. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 7–14, New York, NY, USA, 2005. ACM Press.

[30] Sven Woop, Gerd Marmitt, and Philipp Slusallek. B-KD Trees for Hardware Accelerated Ray Tracing of Dynamic Scenes. In *Proceedings of Graphics Hardware*, 2006.

[31] Chris Wyman. Interactive image-space refraction of nearby geometry. In *Proceedings of GRAPHITE*, pages 205–211, 2005.

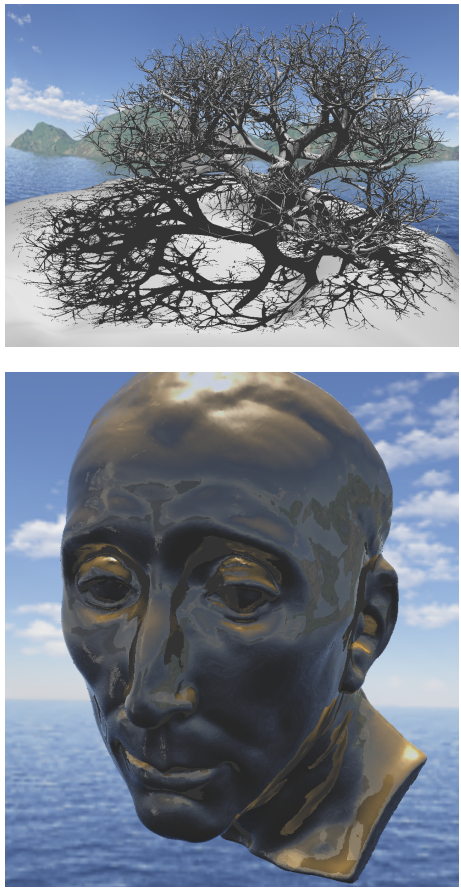


Figure 9: These Images depict from top to bottom. Shadows of a complex tree computed by our approach, and a reflective bust of Niccolo da Uzzano.

Figure 10: Various renderings using the proposed GPU ray casting are shown. Note that the rightmost figure in the middle row is one snapshot out of a GPU animation using vertex-skinning. In the last row we show cubemap reflections (left) and reflections rendered with our method (middle) and with a software ray tracer (right).

Frame-to-frame coherent GPU ray-casting for time-varying volume data

Sergi Grau, Dani Tost

GIE-CREB Universitat Politècnica de Catalunya
Email: {sgrau, dani}@lsi.upc.edu

Abstract

We present a new GPU-based ray-casting method for the visualization of time-varying structured grid volume datasets that uses frame-to-frame coherence in order to speed-up rendering. This method allows users to visualize the data through time steps of different sizes, forward and backward. At each frame, it avoids casting rays through pixels that have not changed their intensity in relation to the previously generated frame. To do so, we ray-cast a 3D texture that contains the current value of the voxels and the instant in which this value changes. Thus, as we sample a ray, we are able to compute when the pixel intensity will change. The 3D texture is updated at each frame on the GPU using a run-length encoding of the voxel values through time stored in a 2D texture. This method significantly speed-ups rendering of frame-to-frame coherent time-varying datasets.

1 Introduction

In the last years, GPU-based implementations of volume ray-casting have renewed the interest for this rendering technique which is accurate and flexible, but too slow to provide interactivity when it runs on the CPU. Several GPU-based ray-casters for structured grids have been proposed, first based on multiple rendering passes [9] and, more recently, thanks to the new dynamic branching functionality, on a single rendering pass [21].

Many different acceleration techniques have been proposed to accelerate classical CPU-based volume ray-casting of static volume data. They are mostly aimed at performing empty-space-leaping, i.e. avoiding sampling rays through empty regions of the volume. These techniques can roughly be subdivided into two groups. The first category includes the methods employing distance maps [4] [20] or spatial subdivisions of the volume such as minmax octrees [10] in order to adaptively sample regions and to skip empty ones. The second group is composed of methods that consider that, usually, users explore datasets by visualizing them several times, changing continuously the viewpoint around. They exploit the coherence between successive visualizations by using the position of the first significant voxel along a ray in order to compute where sampling should start at the current frame [5] [25]. The first group of techniques is more difficult to map to GPU ray-casting, but, for the second one, an efficient GPU driven implementation has been proposed [8].

Several attempts have been made to speed up ray-casting of time-varying datasets following two basic approaches similar to those used for static data: global data structuring or frame-to-frame coherence. The first strategy extends hierarchical data structures to the temporal dimension. The second strategy exploits frame-to-frame coherence in order to avoid casting unchanged rays. Shen and Johnson [19] use this idea and store incremental voxel models at each frame. They project the modified voxels and recast only the rays affected by these projections. Tost et al. [23] avoid the voxels projection using a double structure: in image-space, a temporal buffer that stores for each pixel the next instant of time in which the pixel must be recomputed, and in object-space a Temporal Run-Length Encoding of the voxel values through time.

All these time-varying data rendering techniques have proven to reduce significantly the rendering time. However, since they run on the CPU, they are not fast enough to provide continuous sequences of animation and to provide interactivity in time-varying data exploration. In this paper, we propose to fit the last frame-to-frame coherent ray-casting approach for time-varying data in the rendering pipeline of the GPU ray-casting. This way, our approach benefits both from software-based and hardware-based accelerations. It casts only modified rays and performs space-leaping on these rays.

It can be used to explore time-varying datasets through a static camera or with a moving camera if it is combined with a GPU-based space-leaping technique for static data [8].

2 Related work

2.1 Static Data and moving camera

When the dataset is static, the position of the first non-transparent voxels hit by the rays can be stored and reprojected for the next camera position in order to estimate the new ray sampling starting positions. The reprojection introduces a cost overhead that can be of minor importance if the space leaping gain is important, which is generally the case for isosurface rendering.

This idea was first proposed by Gudmundsen and Radén [5] for parallel projections and further developed by Yagel and Shi [25] for perspective projections. These authors store in an auxiliary coordinate buffer the coordinates of the first non-transparent voxel encountered by the ray emitted at each pixel. Wan et al. [24] found that, being point-based, this method can create artificial hole pixels, that can be corrected using a cell-reprojection scheme. Yoon et al. [26] reproject rays instead of reprojecting hitpoints or cells. Klein at al. [8] proposed a GPU single-pass implementation of the reprojection technique that uses an off-screen floating-point render target to store the hitpoints. Their approach achieves a speed-up of more than a factor of two for isosurface rendering. It is complemented by a a selective supersampling that reduces aliasing.

2.2 Frame-to-frame coherent rendering of time-varying volume datasets

The idea of taking advantage of frame-to-frame coherence has been exploited in time-varying polygonal scenes for visibility computations [6] and global illumination [7]. It has also been used in order to speed up isosurface extraction in indirect time-varying volume rendering [22].

For direct volume rendering, two main approaches have been proposed: to treat time-varying data as an $n-D$ model with $n = 4$ [14], or to separate the time dimension from the spatial ones. In the second approach, at each frame, the data values corresponding to that instant of time must be loaded. Reinhard et al. [15] have addressed the I/O bottleneck of time-varying fields in the context of ray-casting isosurfaces. They partition each time step into a number of files containing a small range of iso-values. They use a multiprocessor architecture such that, during rendering, while one processor reads the next time step, the other ones render the data currently in memory. Binotto et al. [2] propose to compress highly coherent time-varying datasets into 3D textures using a simple indexing scheme mechanism that can be implemented using fragment shaders. Youmesy et al. [27] accelerate data load at each frame using a differential histogram table that takes into account data coherence.

Temporal coherency can also be taken into account during rendering. Several authors have exploited it in various algorithms: ray-casting, shear-warp [1], texture-mapping [12] [16] [2] and splatting [27]. We herein focus on the first group. Ma et al. [13] construct a Branch-on Need Octree (BON) for every instant of time and merge the subtrees that are identical in successive BONs. They propose to ray-cast the first BON completely and only the modified subtrees of the following BONs. The Temporal Space Tree (TSP)[18] is a spatial octree that stores at each node a binary tree representing the evolution of the subtree through time. The TSP tree can also store partial sub-images to accelerate ray-casting rendering. It has also been used to speed up texture-based rendering [3]. In addition, a wavelet-based variant of the TSP (WTSP) has recently been published [17].

Shen and Johnson [19] exploit ray coherence when the property values inside the voxels change along time and the camera remains static. Given the initial data sets, this method constructs a voxel model for the first frame and a set of incremental models for the successive frames, composed of the coordinates of the modified voxels and their values. The first frame is computed from scratch. The next frames are computed by determining which pixels are affected by the modified voxels of the corresponding incremental file, updating the voxel model and recasting only the modified rays. This strategy produces a significant speed up of the animation if the incremental files are small, i.e., the number of modified voxels is low. Liao et al. [11] improve this technique by computing an additional differential file that stores the position of the changed pixels. At each frame, the rays are either computed

following Shen et al.'s strategy or using the differential file, i.e. avoiding the cost of projection of the modified voxels.

Finally, Tost et al. [23] use a double structure. In image-space, they compute a temporal buffer that stores for each pixel the next instant of time in which the pixel must be recomputed. In object-space, they codify the variation of the property values through time using a Temporal Run-Length Encoding (TRL). At each frame, only the rays corresponding to pixels that need to be updated at the current frame are cast. The TRL of the voxels visited along a ray are used to update the temporal buffer. Space leaping is also provided by storing in the temporal buffer the position of the first non-empty voxel.

Our approach maps this last method on the GPU. In addition, we extend it by allowing users to visualize the data not only from frame to frame but through time steps of different sizes, forward as well as backward. Moreover, we can combine our approach with the GPU-based reprojection technique, and thus we support changes in the camera position during the visualization of time-varying data.

3 Frame-to-frame coherent GPU ray-casting of time-varying volume data

Before giving details of the implementation, we present an overview on how the basic approach of [23] can be mapped to the GPU. The basic GPU ray-casting that we have implemented is based on a single pass using the dynamic flow control in the fragment processor available in recent GPUs [21]. We first render the front faces of the bounding box of the voxel model. Then, we cast only the rays inside the box, starting sampling at the projected positions. For each ray, we sample the volume, fetching the sample points from a 3D texture map that stores the density and the gradient values, taking advantage of the hardware-based 3D interpolation.

For time-varying data, since we want to avoid recasting rays through pixels that haven't changed their intensity in relation to the previously rendered frame, we need to store for each ray its intensity and the next and previous time step at which this intensity changes. We use these instants in the depth buffer in order to take advantage of the depth bound test to avoid casting unchanged rays. We also store the first non-empty hitpoint in order to know the starting sampling point or to compute it by reprojection, when the camera position changes. We call these data structures *Time Buffers (TB)*.

In order to compute the next instant at which the ray intensity changes, we use a run-length codification of the voxels values through time (Time Run-Length encoding, TRL). Each code is composed of a value and the number of time steps in which this value remains constant. This codification is computed in a pre-process according to a user-defined error. When a ray is cast, its next (or previous) instant of intensity change is computed using the minimum (or maximum) change instant of the current codes of all the visited voxels. In Section 3.1, we explain how we store the TRL in the GPU.

The basic algorithm is shown in Figure 1. Rendering starts at the first frame by projecting the front faces of the volume bounding box. Pixels outside the projection are defined as *inactive* and not cast. *Active* pixels can remain constant in relation to the previously rendered frame or they may have changed and need to be recomputed. In the former case we call them *fresh* pixels and, in the latter case, *expired pixels*. At the first frame all *active* pixels are set as being *expired* pixels. After the first frame has been rendered, while the user explores the time-varying data, the same process is repeated. At each iteration, several changes can occur: instant of time, camera, transfer function and lights. A change in time leads to update the 3D texture. A change in the camera activates the reprojection mechanism. Finally, changes that modify all the active pixels intensity yields to set all them to *expired*.

```
Hitpoints initialization
Set all active pixels as expired
Render expired active pixels
repeat {
    if frame changes
        Update 3DTexture
    if camera changes
        ReProjection
    if camera or transfer function or lights change
        Set all active pixels as expired
     Render expired active pixels
}
```

Figure 1: Main algorithm.

3.1 Data structures

The Time Buffers (TB) are implemented as a Frame Buffer Object (FBO) with a depth attachment and three image-size color attachments: *color texture*, *hitpoint texture* and *info texture*. All these textures have the FP32RGBA format with 32 floating-point for each channel (see Figure 2).

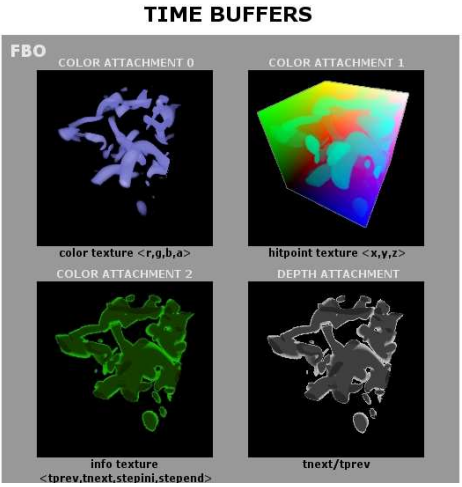


Figure 2: The time buffers FBO.

We store the current ray intensity (RGBA) in the *color texture*, the first non-empty hitpoint position in the *hitpoint texture*, and, in the *info texture*, the next *(tnext)* and previous *(tprev)* instant at which the ray intensity changes. This leaves us 2 data channels in this texture. We use them to enhance space-leaping. Often, users use the transfer function to select subsets of the volume, by assigning zero opacity to non-empty unselected regions. In this case, not only empty regions of the volume but also non-selected ones can be skipped (see Figure 3). To do so, we store in the two left channels of the *info texture* the distance between the first non-empty position and the first ($step_{ini}$) and last points ($step_{end}$) of the visible part of the volume.

The *Time Run-Length* (TRL) representation of the time-varying voxel model stores for every voxel v_i a sequence of codes composed of the voxel value and the number of time steps in which this value remains constant within a user-defined error: $codes(v_i) = < value_k, nframes_k >, k = 1 \ldots ncodes(v_i)$.

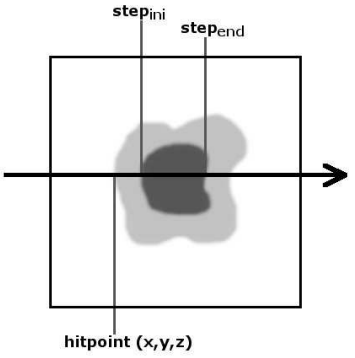


Figure 3: Space leaping empty regions (in white) and non-selected regions (in light gray). The values $step_{ini}$ and $step_{end}$ define the selected interval along the ray (in dark gray).

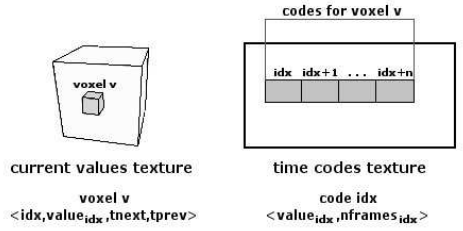


Figure 4: TRL structure in the GPU: current values 3D texture and 2D time codes texture.

This information is stored in a 2D texture *(time codes texture)* sorted using the voxel coordinates as a primary key and time as secondary key (see Figure 4). For ray-casting, we use a 3D texture *(current values texture)* that stores the current value of all the voxels and their next *(tnext)* and previous *(tprev)* instants of change. At each frame, this 3D texture is updated, also on the GPU, using the 2D texture. To do so, for all the voxels in the *current values texture*, we also store an index to the current code in the *time codes texture*.

3.2 Handling *expired* pixels

The mechanism used to prevent recasting rays through *fresh* pixels is the depth bounds test. This test restricts render to the pixels which depth value falls within a user specific range between 0 and 1.

We scale and bias into an interval of 0 to 1 the values of *tprev* and *tnext* in the *info texture*. Therefore, when data exploration is forward, i.e., positive increase in the time dimension, we write the *tnext* value into the depth buffer. Then, before rendering we set the depth bounds range to $[0 \dots f_{cur}]$, being f_{cur} the current normalized frame. This ensures that during rendering pixels changing further on time are not rendered.

When data exploration is backward (i.e. diminishing time), we write the *tprev* values into the depth buffer and set the depth bounds range to $[f_{cur+1} \dots 1]$. When the direction of data exploration changes, since the depth values correspond to the previous direction, they must be rewritten according to the new direction before rendering.

For the first frame and whenever the transfer function or the lights change, all the *active* pixels must *expire*. We write 0 for all the pixels of the depth buffer if the exploration if forward, and 1 otherwise.

3.3 Initialization

Let $cvt(v)$ denote the information stored in the *current values texture* at voxel v and let $value(i)$ and $nframes(i)$ represent the property value and the number of frames stored at code i in the *time codes texture*. The *current values texture* is initialized in the CPU and next transferred to the GPU. We use an auxiliary table that stores for each voxel v an index to its first code (idx_v) in the *time codes texture*. Thus, the value of a voxel v of the *current values texture* is computed as: idx_v, the property value of the code idx_v and the corresponding *tprev* and *tnext* values computed respectively as 1 and the number of frames of idx_v : $cvt(v) =< idx_v, value_{idx_v}, 1, nframes_{idx_v} >$.

As described above, the *hitpoint texture* is initialized on the GPU by projecting the front faces of the volume bounding box. For each projected pixel, a fragment shader is activated, that computes the first non-empty voxel.

3.4 Rendering

Render is performed on *expired* pixels only. For every pixel, the fragment shader shown in Figure 5 traverses the texture along the ray between $step_{ini}$ and $step_{end}$ until maximum opacity is reached. It computes the pixel intensity, and its *tnext* and *tprev* values.

```
uniform sampler3D Curr;
uniform sampler1D LUT;
uniform sampler2DRect Hitpoint;
uniform sampler2DRect Info;
uniform vec3 camera;
uniform vec3 stepsize;
uniform float forward;
void main(void)
{
    vec4 dst = vec4(0,0,0,0);
    vec3 position = texture2DRect (Hitpoint,gl_TexCoord[0].xy);
    vec3 direction = normalize(position - camera);
    vec3 step = direction * stepsize;
    float tnext = 1.0, tprev = 0.0;
    vec4 i = texture2DRect(Info,gl_TexCoord[0].xy);
    position = position + step * stepini(i);
    for(int b = 0; b < stepend(i); b++)
    {
        vec4 v = texture3D(Curr, position);
        vec4 src = texture1D(LUT,value(v));
        dst = (1.0 - dst.a) * src + dst;
        tnext = min(tnext,next(v));
        tprev = max(tprev,prev(v));
        position = position + step;
        if (dst.a > 0.98) break;
    }
    gl_FragData[0] = dst;
    gl_FragData[1] = vec4(tprev,tnext,stepini(i),stepend(i));
    if (forward)
        gl_FragDepth = tnext;
    else
        gl_FragDepth = tprev;
}
```

Figure 5: Coherent ray-casting fragment shader. For simplicity, this shader only computes volume rendering without illumination.

3.5 Updating the *current values texture*

At each new time step of the dataset, the 3D *current values texture* is updated using the 2D *time codes texture*. Before the 8th serie of the NVidia cards, it was not possible to update a 3D texture without transferring all the volume from the CPU. The only mechanism available to modify a 3D texture in the GPU was to codify the 3D texture as a 2D texture. A 2D texture can be attached to a FBO, so we can write on it in the GPU. With the new series and the *glFramebufferTexture3DEXT* extension, it is possible to attach a slice of a 3D texture to an FBO. We use this extension to update the 3D texture by slicing it by z-sorted planes. For each slice, we perform two steps: first, we determine the voxels that must be updated, and next we update these voxels

using the depth test to restrict the second step to only them.

A voxel must be updated, if the current frame is not between its *tnext* and *tprev* values in the 3D *current values texture*. The first step uses a fragment shader that checks this condition. For the second step, we need to take into account that the 3D texture holds absolute time steps (*tnext* and *tprev*) whereas the 2D texture stores relative time lengths of the codes *(nframes)*. Therefore, given a voxel v and its current texture value $< idx, value_{idx}, tprev, tnext >$, when this texture value is updated to the next code of the 2D texture it is set to: $< idx+1, value_{idx+1}, tnext, tnext + nframes_{idx+1} >$. When it is set to the previous code $idx-1$ it is: $< idx-1, value_{idx-1}, tprev - nframes_{idx-1}, tprev >$.

Thus, the second step uses a fragment shader that searches in the 2D texture the new current code of a voxel and it updates the 3D texture accordingly, as shown in Figure 6.

3.6 Reprojection

The *hitpoints texture* can be used to perform the reprojection as described by Klein et al. [8]. We use Pixel Buffer Object (PBO) extension to load the *hitpoints texture* values onto a Vertex Buffer Object (VBO) in the GPU. Then, the vertices are resent to the graphics pipeline having set the OpenGL matrices to the new view parameters. This avoids us the need to initilize the *hitpoint texture* at each camera movement.

3.7 Trilinear interpolation

For simplicity, in the previous explanation, we have assumed that *nearest neighbor* interpolation was applied in fetching values of the 3D texture. However, tri-linear interpolation of the property values is necessary to provide the desired smoothness and accuracy of the images. The problem is that the 3D texture holds frame identifiers (next/previous frame at which the pixel changes) as well as property values. The 3D interpolated frame identifier represents an average of the next (previous) instant of ray change, whereas they should represent its minimum (maximum). As a consequence, at the boundary between regions of change and regions that remain constant, some rays may not be recomputed. This is illustrated in Figure 7. In order to prevent this error,

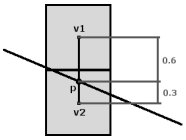
```
uniform sampler2D curr;
uniform sampler2DRect codes;
uniform float t;
void main(void)
{
    vec4 v = texture2D(curr,gl_TexCoord[0].xy);
    vec2 c;
    float idx = index(v);
    float tnext = next(v), tprev = prev(v);
    while((t < tprev) || (t >= tnext))
    {
        if (t < tprev)
        {
            idx- -;
            c = currCode(codes,idx);
            tnext = tprev;
            tprev = tprev - code(c);
        } else {
            idx++;
            c = currCode(codes,idx);
            tprev = tnext;
            tnext = tnext + code(c);
        }
    }
    gl_FragColor = vec4(idx,value(c),tprev,tnext);
}
```

Figure 6: Fragment shader that updates the 3D texture for random access through time. In case the access is sequentially, we can avoid to use the loop.

we propose two different strategies. First, during run-length encoding time-varying data we compute the time length in which the voxel's neighbor values remain constant instead of considering only the voxel value changes. This way, more codes are created, but the interpolated frame identifier are more conservative and will recompute all rays falling at the boundary between changed and unchanged regions. The second strategy keeps the run-length codification as explained in the previous section, but when computing the Z-buffer, it takes into account not only the time step stored at the corresponding pixel, but those within a neighborhood computed as the rasterized voxel edge length.

3.8 Memory management

The capacity of the GPU memory determines the size of the voxel model and the number of consecutive frames that can be handled in the GPU.



	value	tprev	tnext
v1 = <	5 ,	3 ,	8 >
v2 = <	8 ,	2 ,	6 >
p = <	7 ,	2.3 ,	6.6 >
p' = <	5 ,	3 ,	6 >

Figure 7: Time interpolation artifacts. The interpolated *tnext* value at p is 6.6. At frame 6, the ray through p will not be updated although it should be because v2 has changed and it affects the interpolated color at p.

Specifically, the GPU memory must fit the 3D *current values texture* plus the 2D *time codes texture*. The *current values texture* has an occupancy of 4 floats (index, value, *tnext*, *tprev*, gradient), 4 bytes each: $n * 5 * 4$ bytes, being n the number of voxels. Currently, the maximum size of an RGBA float 3D texture is 2048^3.

The size of the TRL is proportional to the number of codes of all the voxels: $\sum_i^n codes(i)$. However, for the voxels that are empty all time along, we store a unique common "empty code" with zero value and infinite number of frames, this way, the occupancy of the TRL is actually: $\sum_i^m codes(i)$, being $m \leq n$ the number of non-empty voxels of the model. A code occupies 3 floats (value, gradient, number of time steps). The maximum size for a 2D texture is currently 4096^2 RGBA, which is also the upper limit for the total number of codes that we can store at a time in one texture. Usually, the time-varying datasets are not very large in space. For instance, in our simulations, we have used datasets of size about $128^3 \times 100 frames$ having a total number of codes that fits entirely into one 2D texture. However, if the number of codes exceeds the 2D texture limit, we need to split the TRL into various 2D textures. The maximum number of textures that can be handled in the GPU is thus the GPU memory minus the occupancy of the 3D texture divided by 4096^2. For our datasets sizes, we can have up to 2 of these textures. Above this limit, we will need to read the texture from the CPU. In any case, this is far much faster than loading the voxel values at each frame.

4 Results

We have performed the simulations on a Pentium Dual Core 3.2 GHz with 3 GB memory and a NVidia GeForce 8800 GTX with 768 MB. We have used two datasets: *TJet* and *Vortex* from the NSF ITR repository (see Figure 8).

Table 1 shows the characteristics of the datasets: size and number of frames, total number of voxels values through time (nv) in millions, number of codes with an error of $\varepsilon = 10^{-5}$ (nc_1) and $\varepsilon = 10^{-3}$ (nc_2). From the table we can observe that the number of codes is very low in relation to the total number of voxels values (less than 8% for the Vortex dataset and 4% for the TJet dataset). This is due to the fact that the datasets are very coherence through time. We tried other error estimations, but the results do not differ so much. Because of this high coherency, the TRL provides an efficient compression of the data. We are able to store all the time steps in the GPU, whereas if we had stored the whole 3D model on the GPU, we would have been able to manage only few time steps. Specifically, in our case, for the Vortex dataset, we need 190MB to store the TRL for the 100 frames, whereas we would need 3.3GB to store all the 3D voxel models.

Dataset	Size	nv	nc_1	nc_2
Vortex	$128^3 x100$	205	15(14)	14(13)
TJet	$129^2 x104x150$	260	10(8.5)	7.5(7)

Table 1: Datasets: size and number of frames, total number of voxels values through time (nv) in millions, number of codes with an error of $\varepsilon = 10^{-5}$ (nc_1) and $\varepsilon = 10^{-3}$ (nc_2). Numbers in parentheses represent the number of codes when all the empty codes are codified as a unique one.

Table 2 and 3 show the results of the simulations for different image size animations. Times are expressed in frames per second (fps). In the second column, we can see the times for the visualization using each model separately. The third column contains the times with our method. Frames per second are computed as the average during a 300 frames data exploration session in which data varied at each frame, and the user moved freely the camera. For the non-coherent mode, we have only taken into account the transfer time from the CPU to GPU and the rendering, but not the transfer time from disk

to CPU. The results are satisfactory: we achieve factors of speed ups between 2 and 4 for all image sizes. We have observed that the whole process is very fast on the GPU. One of the none expected results is that the computation of the *tnext* and *tprev* values during the rendering stage is fairly time consuming. This is due to the fact that we use a branching (max/min operation) inside the loop, which is still slow on the current shader processors.

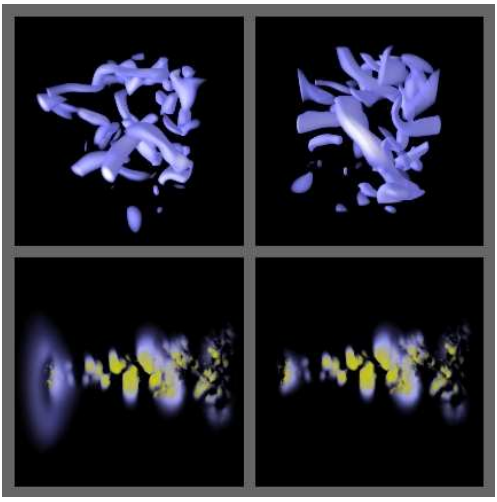


Figure 8: Two frames of Vortex(top) and TJet(bottom).

Image size	Vortex n3D	Vortex TRL
500x300	27.54fps	92.00fps
512x512	21.57fps	47.67fps
800x800	20.00fps	38.76fps
1250x900	19.45fps	26.50fps

Table 2: Times in frames per seconds of the simulation of the Vortex dataset: the first column shows the image size, the second column the times for the non-coherent method, and the last column the times for our method.

Figure 9 shows a set of two consecutive frames of the Vortex dataset, rendered separately, with our method and nearest neighbor interpolation and with trilinear interpolation. On the right, we show *inactive*, *expired* and *fresh* pixels (white/red/green). We can see that this dataset does not have much *inactive*

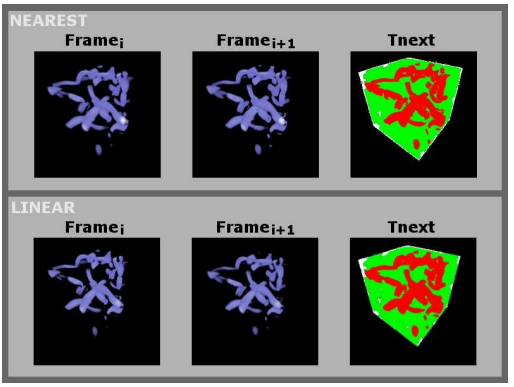


Figure 9: Two consecutive frames of the Vortex dataset: top nearest interpolation, bottom linear. From left two right: first frame, next frame, pixel status before the next frame rendering. *Inactive* (white), *expired* (red) and *fresh* (green) pixels.

Image size	TJet n3D	TJet TRL
500x300	30.67fps	132.74fps
512x512	21.67fps	89.82fps
800x800	12.54fps	44.38fps
1250x900	10.58fps	46.15fps

Table 3: Times in frames per seconds of the simulation of the TJet dataset: the first column shows the image size, the second column the times for the non-coherent method, and the last column the times for our method.

pixels, because most of the voxels have a non-empty value through time. On the contrary, there are many fresh pixels. Comparing the linear and the nearest interpolation, we see that some fresh pixels in the nearest interpolation become expired in the linear strategy.

5 Conclusions

We have presented a new frame-to-frame coherent GPU-based ray-casting for the visualization of time-varying structured grid volume. Our method allows users to visualize the data through time steps of different sizes, forward and backward, and moving the camera. It conveniently organizes the data on the GPU and reduces the amount of information

to be loaded by using run-length encoding through time. Depending on the degree of temporal coherence of the data, we achieve rendering times 2 to 4 times faster than with the basic non coherent raycasting.

In the future, we plan to use the TRL encoding in other direct volume rendering approaches, specifically texture mapping, which would be very similar to our current method, and splatting.

Acknowledgments This work has been partially funded by the project MAT2005-07244-C03-03 and the Institut de Bioenginyeria de Catalunya (IBEC). The authors thank Eduard Groeller for his valuable comments.

References

[1] K. Anagnostou, T. Atherton, and A. Waterfall. 4D volume rendering with the Shear-Warp factorization. *Symp. Volume Visualization and Graphics'00*, pages 129–137, 2000.

[2] A. P. D. Binotto, J. Comba, and C.M. Dal Sasso Freitas. Real-time volume rendering of time-varying data using a fragment-shader compression approach. *6th IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 69–76, 2003.

[3] D. Ellsworth, L. J. Chiang, and H. W. Shen. Accelerating time-varying Hardware volume rendering using TSP trees and color-based error metrics. In *IEEE Visualization'00*, pages 119–128, 2000.

[4] J. Freund and K. Sloan. Accelerated volume rendering using homogeneous region encoding. *IEEE Visualization'97*, pages 191–196, 1997.

[5] B. Gudmundsson and M. Randén. Incremental generation of projections of ct volumes. In *Proc. of the first conference on Visualization on biomedical computing 1990*, pages 27–34, 1990.

[6] V. Havran, J. Bittner, and H. P. Seidel. Exploiting temporal coherence in ray casted walkthroughs. In *ACM Spring conference on Computer graphics*, pages 149–155, 2003.

[7] V. Havran, C. Damez, and H. P. Myszkowski, K. Seidel. An efficient spatio-temporal architecture for animation rendering. In *Proc. 14th Eurographics workshop on Rendering*, pages 106–117, 2003.

[8] T. Klein, M. Strengert, S. Stegmaier, and T. Ertl. Exploiting frame-to-frame coherence for accelerating high-quality volume raycasting on graphics hardware. In *IEEE Visualization '05*, pages 123–230, 2005.

[9] J. Krüger and R. Westerman. Acceleration techniques for GPU-based volume rendering. In *IEEE Visualization'03*, pages 287–292, 2003.

[10] S. Lakare and A. Kaufman. Light weight space leaping using ray coherence. In *IEEE Visualization'04*, pages 19–26, 2004.

[11] S.K. Liao, Y.C. Chung, and J.Z.C. Lai. A two-level differential volume rendering method for time-varying volume data. *The Journal of Winter School in Computer Graphics*, 10(1):287–316, 2002.

[12] E. B. Lum, K. L. Ma, and J. Clyne. A Hardware-assisted scalable solution for interactive volume rendering of time-varying data. *IEEE Trans. on Visualization and Computer Graphics*, 8(3):286–301, 2002.

[13] K. Ma, D. Smith, M. Shih, and H. W. Shen. Efficient encoding and rendering of time-varying volume data. *Technical Report ICASE NASA Langsley Research Center*, pages 1–7, 1998.

[14] N. Neophytou and K. Mueller. Space-time points: 4D splatting on efficient grids. In *IEEE Symp. on Volume Visualization and graphics*, pages 97–106, 2002.

[15] E. Reinhard, C.Hansen, and S.Parker. Interactive ray-tracing of time varying data. In *EG Parallel Graphics and Visualisation'02*, pages 77–82, 2002.

[16] J. Schneider and R. Westermann. Compression domain volume rendering. In *IEEE Visualization'03*, pages 39–47, 2003.

[17] H. Shen. Visualization of large scale time-varying. *Journal of Physics: Conference series*, 1(46):535–544, 2006.

[18] H. Shen, L. Chiang, and K. Ma. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In *IEEE Visualization'99*, pages 371–377, 1999.

[19] H. W. Shen and C. R. Johnson. Differential volume rendering: a fast volume visualization

tech for flow animation. In *IEEE Visualization'94*, pages 180–187, 1994.

[20] M. Sramek and A. Kaufman. Fast ray-tracing of rectilinear volume data using distance transforms. *IEEE Trans. on Visualization and Computer Graphics*, 6(3):236–252, 2000.

[21] S. Stegmaier and al. A simple and flexible volume rendering framework for graphics-hardware-based raycasting. In E. Gröller and I. Fujishiro, editors, *Volume Graphics*, pages 187–195, 2005.

[22] P. Sutton and C. Hansen. Accelerated isosurface extraction in time-varying field. *IEEE Trans. on Visualization and Computer Graphics*, 6(2):98–107, 2000.

[23] D. Tost, S. Grau, M. Ferré, and A. Puig. Ray-casting time-varying volume data sets with frame-to-frame coherence. *VDA'06*, 2006.

[24] M. Wan, A. Sadiq, and A. Kaufman. Fast and reliable space leaping for interactive volume rendering. In *IEEE Visualization'02*, 2002.

[25] R. Yagel and Z. Shi. Accelerating volume animation by space-leaping. In *IEEE Visualization'93*, pages 62–69, 1993.

[26] I. Yoon and al. Accelerating volume visualization by exploiting temporal coherence. In *IEEE Visualization'97*, pages 21–24, 1997.

[27] H. Younesy, T. Möller, and H. Carr. Visualization of time-varying volumetric data using differential time-histogram table. In *Volume Graphics'05*, pages 21–29, 2005.

Using Quadtrees for Energy Minimization Via Graph Cuts

Cristina N. Vasconcelos[1], Asla Sá[2], Paulo Cezar Carvalho[3], Marcelo Gattass[1,2]

[1] Depto. de Informática - Pontifícia Universidade Católica (PUC-Rio).
Rua Marquês de São Vicente, 225. 22453-900 - Gávea, Rio de Janeiro, RJ, Brasil
[2] Tecgraf (PUC-Rio). Rua Marquês de São Vicente, 225. 22453-900 - Gávea, Rio de Janeiro, RJ, Brasil
[3] Instituto de Matemática Pura e Aplicada (IMPA).
Estrada Dona Castorina, 110. 22460 - Jardim Botânico, Rio de Janeiro, RJ, Brasil
Emails:{crisnv@inf.puc-rio.br, asla@tecgraf.puc-rio.br,
pcezar@impa.br, mgattass@tecgraf.puc-rio.br}

Abstract

Energy minimization via graph cut is widely used to solve several computer vision problems. In the standard formulation, the optimization procedure is applied to a very large graph, since a graph node is created for each pixel of the image. This makes it difficult to achieve interactive running times. We propose modifying this set-up by introducing a pre-processing step that groups similar pixels, aiming to reduce the number of nodes and edges present in the graph for which a minimum cut is to be found. We use a quadtree structure to cluster similar pixels, motivated by fact that it induces an easily retrievable neighborhood system between its leaves. The resulting quadtree leaves replace the image pixels in the construction of the graph, substantially reducing its size. We also take advantage of some of the new GPGPU concepts and algorithms to efficiently compute the energy function terms, its penalties and the quadtree structure, allowing us to take a step toward a real time solution for energy minimization via graph cuts. We illustrate the proposed method in an application that addresses the problem of image segmentation of natural images by active illumination.

1 Introduction

Many important problems in image analysis can be posed as optimization problems involving the minimization of some kind of energy function. For some of those problems, methods based on computing the minimum cut on graphs offer the possibility of finding global minimum for some classes of energy functions [3].

These methods explore the fact that algorithms for computing minimum cuts in polynomial time have been known for some time [1].

Much research has been done in setting the mathematical requirements for the energy functions that justify the use of Graph Cut minimization for both exact and approximate cases [1],[2],[3]. The applicability of the technique has also been shown by several papers in themes like image segmentation [7], foreground/background extraction [11], clustering [4], texture synthesis[10], photo composition[9] and so on.

However, the use of graph-cut methods for real-time applications has been limited by the size of the graph in which optimization must take place. In this paper we propose a pre-processing of the input images, in order to produce a new set of nodes and edges, instead of the image pixels and its neighborhood commonly used for the graph construction. The proposed sets are considerably smaller, inducing a significant reduction on the running time of the graph-cut procedure. We call Quad Cut the use

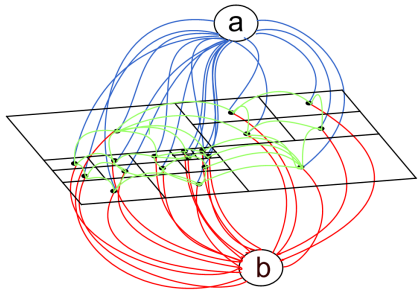


Figure 1: Quad Graph

of graph cut minimization in this modified way, the concept is illustrated in Figure 1.

The idea of the preprocessing is to group similar pixels, but in a way that creates a well known neighborhood system. For that reason, we choose to group them into Quadtree nodes. The metric used for grouping should be a similarity criteria appropriate to the context being analyzed by the energy function.

After constructing the quadtree, its leaves are used, instead of image pixels, as the basis for the construction of the graph. An appropriate energy function and neighborhood relationships are created to be used in this new procedure.

As we are interested in offering a fast approximation for the computer vision problems that rely on computing the minimum cut on an appropriately constructed graph, in addition to reducing the graph size, we also explore graphics hardware to efficiently compute energy function terms, its penalties and the Quadtree structure. Inspired by [15], we can take advantage of the Graphics Processing Unit (GPU) parallelism to compute all the preprocessing steps, including an efficient construction of a Quadtree with all the information needed for the optimization algorithm, leaving the CPU free to minimize the Graph constructed with the quad leaves.

As an application, we address the problem of foreground/background image segmentation aided by active illumination, in which graph cuts are used to compute an optimal binary classification, starting with an initial background/foreground separation, provided by the difference in intensity levels for two different illumination levels [11]. Figure 2 illustrates the application. Observe that the quality of the binary segmentation produced can be used for matting.

The paper is organized as follows: some applications that use energy minimization via graph cuts in vision are reviewed in the next Section; Section 3 briefly describes the basic concepts for energy minimization via Graph Cuts; then, in Section 4 we argue that grouping pixels into the Quadtree structure is useful to substantially reduce the nodes of the final graph to be cut. An GPU implementation to construct the quad tree structure is discussed in section 5. In Section 6 we present an illustrative implementation to accelerate the active illumination segmentation problem. Results are discussed in Section 6.4 followed by conclusions and future work.

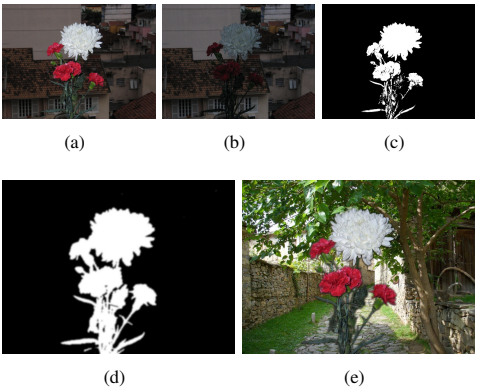


Figure 2: Example of minimization via Graph Cuts to the image segmentation of natural images aided by active illumination. In (a) and (b) the input images are shown. In (c) the initial segmentation provided by active illumination is compared to the final optimized segmentation shown in (d). The composition result (using parameters $\sigma_L = 0.25$, $\sigma_C = 0.05$) is shown in (e).

2 Related Work

In the Computer Vision and Graphics context, the graph cut method, can be interpreted as a clustering algorithm that works in a image feature space to produce spatially coherent clusters as result. Several recent works creatively models different applications as a labeling problem, then uses graph cuts to optimize the proposed labeling. This is the case in [9], where a framework for composing digital photos into a single picture, called digital photomontage, is described. Having n source images $S_1, ..., S_n$ to form a photo composition, the problem is posed as choosing a label for each pixel p, where each label represents a source image S_i. The proposed method extends the applicability of graph cuts to compute selective composites, photo extended depth of field, relighting, stroboscopic visualization of movement, time-lapse photo mosaics and panoramic stitching.

In [4], the spatial clustering problem is modeled as a labeling problem. The spatial coherence is guaranteed by the penalty imposed for neighboring pixels to have different labels, that are used as weights for the edges between neighbor pixels in the graph.

In [10], texture synthesis is modeled as labeling. The method generates textures by copying input texture patches into a new location, the graph-cut technique is used to find the optimal region inside the patch to be transferred to the output image. Such patch fitting step is a minimum cost graph cut problem using a matching quality measure for pixels from the old and new patch.

The problem of monochrome image colorization is modeled as a segmentation problem in [5]. The input image is partitioned interactively while the user specifies input colors, maintaining smoothness almost everywhere except for the sharp discontinuity at the boundaries in the image.

Image segmentation problem can also be solved by minimization via graph cuts. The main work lies in defining the energy function that models the specified application. In particular, background/foreground segmentation can be solved by means of Graph Cuts. In [6], [7] and [8] the user has to indicate coarsely the foreground and the background pixels, as initial restrictions for a minimization process. Then, graph cuts are used to find automatically the globally optimal segmentation for the rest of the image.

Similarly to our algorithm, [8] proposed the use of the image uniform regions as the nodes used in the graph construction in stead of the image pixels. They group similar pixels into such regions segmenting the original image using the watershed method. We believe that such segmentation do not provide a neighborhood system neither a boundary perimeter and area as easy to compute as the one presented in our proposal provided by the quadtree structure.

In this paper we will concentrate on applying graph cuts for image foreground-background segmentation aided by *active illumination*, as in [11]. Active illumination consists of using an additional light source in the scene that illuminates the foreground objects more strongly than the background. This gives *a priori* clues of the foreground. The information derived from this difference in illumination replaces the indication of object and background pixels by the user. These initial clues are then used as seeds for an optimization procedure in order to obtain a high quality segmentation. Potentially, the approach could be used for video capture, since a projector can be controlled to produce alternating illumination conditions at 60 Hz.

3 Basic concepts in Energy Minimization via Graph Cuts

In Computer Vision and Graphics, energy functions minimization is commonly computed using the min-cut/max-flow algorithms. The general goal for using the min-cut/max-flow algorithms is to find a labeling L, that assign each variable $p \in P$ (usually associated with the pixels of the input image) to a labeling $L_p \in L$, which minimizes the corresponding energy function.

The number of possible values assumed by the variables of the energy function is assumed finite, and modeled as a set of labels L, each label representing a possible output value.

The energy function to be optimized can be generally represented as [2]:

$$E(L) = \sum_{p \in P} D_p(L_p) + \sum_{p,q \in N} V_{p,q}(L_p, L_q), \quad (1)$$

Traditionally, $N \subset P \times P$ is a neighborhood system on pixels, $D_p(L_p)$ is a function that measures the cost of assigning the label L_p to the pixel p, while $V_{p,q}$ measures the cost of assigning the labels $\{L_p, L_q\}$ to the adjacent pixels p and q and is used to impose spatial smoothness.

The method of Graph Cuts to minimize (1) is applied by the creation of a graph normally containing nodes corresponding to each of the image pixels and some additional special nodes, called terminals, corresponding to each of the possible labels. There are two types of edges in the graph: n-links and t-links. N-links are the edges connecting pairs of neighboring pixels, representing the neighborhood system in the image, while t-links are edges connecting pixels with terminals nodes. All edges in the graph are assigned some weight or cost related to the energy function terms. The cost of a t-link corresponds to a penalty for assigning the corresponding label to the pixel, derived from the data term D_p in (1). The cost of a n-links corresponds to a penalty for discontinuity between the pixels. These costs are usually derived from the pixel interaction term $V_{p,q}$ in (1).

The Graph Cut finds a minimum of the energy function (1), providing an optimal labeling for the graph nodes [2].

4 Grouping Pixels into Quadtrees Leaves

When modeling computer vision problems as a energy-minimization problem, one can use different kinds of image features (e.g., luminance, color, gradient, frequency) and different metrics (e.g., statistical functions, differences between images, min/max relations). However, whatever the image feature or the metric used in the energy function, most natural images have areas of pixels presenting similar values according to them. Those pixels are expected to receive the same label in the energy minimization output. Our approach takes advantage of this fact, grouping pixels of such uniform areas, thus decreasing the graph size on which the min-cut algorithm is to be applied.

One more question arises here. If, on one hand, grouping pixels reduces the size of the graph, on the other hand, it may cause its adjacency topology to be more complex than the usual 4- or 8-connected pixel neighborhood systems. This may lead to spending considerable time both to find suitable clusters of pixels and to compute their adjacency relationships, overcoming the benefits by the smaller graph size.

Driven by these observations, we propose the use of a quadtree structure for grouping pixels into regions using a similarity criteria, while, at the same, creating a manageable neighborhood system between the quadtree leaves, in which adjacency relationships are easily retrievable.

In the next subsections we show how a graph for energy minimization can be constructed using quadtree leaves. The construction of the quadtree itself is discussed in section 5.

4.1 Graph Cuts using Quadtree Leaves

Using the quadtree leaves as the input data for the energy minimization via graph cuts, our goal is to find a labeling L, that assigns a label $L_t \in L$ to each leaf $t \in T$ of the quadtree, that minimizes the energy function adopted. The same set of the labels L may be used here. The modified energy function can be generally represented as:

$$E(L) = \sum_{t \in T} \alpha * D_t(L_t) + \sum_{t,u \in N} \beta * V_{t,u}(L_t, L_u), \quad (2)$$

Where $N \subset T \times T$ is a neighborhood system on the quadtree leaves, $D_t(L_t)$ is a function that measures the cost of assigning label L_t to leaf t, and $V_{t,u}$ measures the cost of assigning labels $\{L_t, L_u\}$ to the adjacent leaves t and u. The α and β terms are weights for balancing the energy function, explained below.

In such energy function model, the energy variables represent the quadtree leaves. Thus, graph cut minimization is applied to a graph containing nodes corresponding to each leaf of the quadtree and terminal nodes corresponding to each of the possible labels. Now, the n-links connect pairs of neighboring leaves, while t-links connect leaves with terminals nodes.

4.1.1 Weighting the Quadtree Nodes

The α and β factors were added to equation (2) in order to balance the energy metric according to leaves topology. The number of pixels inside a leaf t is $(2^{level(t)})^2$, while the number of pixels in the border between two neighboring leaves t and u is $2^{\min(level(t),level(u))}$. Therefore, we can rewrite (2) by taking α, that represents the weight for the regional term, as the leaf area, and β, that represents the weight for the boundary term, as the number of neighboring pixels between the two leaves.

With the suggested weights, we ensure that larger leaves have greater impact than smaller ones, while also enhancing the neighborhood influence of larger borders.

$$E(L) = \sum_{t \in T} (2^{level(t)})^2 * D_t(L_t) + \sum_{t,u \in N} 2^{\min(level(t),level(u))} * V_{t,u}(L_t, L_u), \quad (3)$$

5 Efficiently computing the Quadtrees

In this section we describe how the quadtree can be constructed efficiently using graphics hardware.

5.1 Quadtrees in GPGPU

The increasing use of the Graphics Processing Unit (GPU) for general-purpose computation (GPGPU) is motivated by its newest capability of performing more than the specific graphics computations which they were designed for.

In the context of our proposal, the GPU can be used for efficiently computing the energy function terms and also for constructing the quadtree whose leaves will be used as nodes in the graph cut minimization. For saving the partial results, we apply the useful concept of "Playing Ping-Pong with Render-To-Texture" [17], rendering to Frame Buffer Objects (FBO) [19] when 32-bit floating-point precision is necessary.

A solution for constructing a quadtree structure for general purposes in GPU is presented in [15]. A reduction operator is described that creates an image pyramid called QuadPyramid. The operator writes in each fragment of the pyramid texture whether it represents a grouping of similar pixels or if it should be threaded as a quadtree internal node, in this case saving the number of leaves covered by the region represented by the fragment.

In a second shader, they identify the quadtree leaves reading the pyramid texture repeatedly, simulating tree traversals from root to leaves. Relative counters, read from the pyramid texture, are used to control such traversals. The origin and size of the found leaves are saved in a output texture, organized as a point list. To construct such list for a quadtree of m leaves over a square image of N pixels, their algorithm may need $(m * \log(\sqrt(N)))$ texture accesses in the worst case.

For our purposes, the resulting quadtree leaves will be used in CPU for graph construction. In addition to the origin and size of the leaves, we will also need leaf values that are used as the graph weights. We propose a simpler image pyramid operator for quadtree construction than the used in [15] and a new algorithm for identifying leaves from the pyramid texture. Next sections explain our methods for quadtree construction and leaves identification.

5.2 Quadtree Construction

Once a similarity criteria has been selected, the input image should be transformed to the adopted metric space, previously to the quadtree construction. For example, when grouping pixels by luminance, the original image should be transformed to the luminance space.

Here, as in [15], the quadtree construction starts by a reduction operator, creating an image pyramid. For each fragment in the pyramid level being constructed, the operator reads four texture samples from the previous pyramid level, representative of its four children in the quadtree. If the samples represent similar nodes, then, the fragment is classified as a leaf, grouping them into a single node that receives its children mean value. Otherwise, the fragment is classified as a tree internal node. The reduction operator is performed until the pyramid top level (1×1 pixel dimension) is reached.

Our algorithm is simpler than the one presented in [15]. While grouping leaves, [15] also computes relative counters in fragments representing internal nodes. Those counters indicate how many leaves are covered by the internal node being processed. In our case, we do not count the existing leaves inside a internal node region because this information is not needed in our leaves isolation solution.

For our purposes, the pyramid texture is used for saving the grouping decision (alpha channel) and the leaves values (RGB channels). Figure 3 shows an image pyramid found using the example application of section 6.

Figure 3: image pyramid found using the example application

5.3 Identifying Final Leaves

In order to identify the quadtree leaves in the pyramid texture, we propose a leaf isolation method that does not require computing several texture transversals, as used in [15], and, as a consequence, does not impose the use of a GPU supporting several nested branches.

Using the pyramid image as input, this processing step produces a texture whose pixels contain the data corresponding to a quadtree leaf (its size, position and representative value), or a color associated with empty data. This texture saves all the data needed for building the graph *a posteriori*.

Our algorithm erases texels representing other than leaf nodes in the pyramid texture. For that, we use a new fragment shader that reads our pyramid texture and discards all fragments that should not be

leaf nodes in the final tree. This shader produces the output texture in a single rendering pass that makes at most two texture accesses per fragment.

The cleanup shader initially reads the fragment classification (leaf/non-leaf) from the alpha channel of the pyramid texture. If the sample is already classified as non-leaf, the fragment is immediately discarded. Otherwise, the pyramid texture is queried again, now on its corresponding parent texture coordinate. When the parent was classified as a leaf, this means that this fragment was grouped with its neighbors into a higher level leaf, so it can also be discarded. However, in the case of a non-leaf parent, this means that the previous shader could not group this node with its neighbors and that the fragment represents a leaf in the final tree.

The fragments that pass through those tests are considered as final quadtree leaves and are written in the output texture, saving in its channels all the data to be associated with the leaf that the fragment represents (see figure 4). By doing this, we guarantee that subsequent steps of the graph construction do not have to query any other texture.

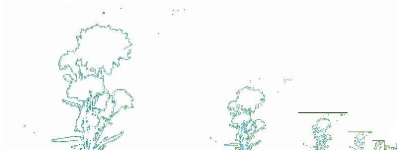


Figure 4: Quadtree Leaf Texture

All the information necessary for graph-cut computing is contained in this texture. For illustration, in figure 5 we reconstruct the entire quadtree using only the leaf texture shown in figure 4). Each leaf is painted according to its level.

6 Application to Active Segmentation

In this section we describe in detail an application of the proposed method to the problem of image segmentation by active illumination using graph cuts.

Segmentation using active illumination employs a single, intensity-modulated light source that stays in a fixed position between shots, as proposed in [11]. The two shots, differently illuminated, are used to obtain an initial segmentation used as a seed, referred as *segmentation seed*, and to attribute

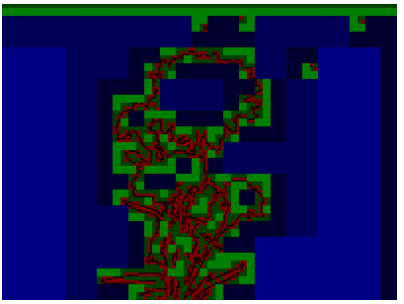


Figure 5: Found Quadtree (leaf color according with its level).

weights to the pixels that are used in graph cut optimization step to produce a improved final segmentation.

6.1 Energy Function Definition

The objective function adopted is the same proposed in [11]. The regional term considers the luminance difference between the two input images and the object color histogram as information that characterize the segmentation. The luminance difference for background pixels is considered to have Gaussian distribution, with density

$$\mathrm{p}_B(p) = \frac{1}{\sqrt{2\pi}\sigma_L}\exp\left(\frac{-|L_{I_2}(p) - L_{I_1}(p)|^2}{2\sigma_L^2}\right), \tag{4}$$

where σ_L is the standard deviation of the luminance differences, illustrated in figure 6 b.

The segmentation seed is defined as $O = \{p \mid \mathrm{p}_B(p) < t\}$, where t is a small threshold.

The color histogram of these initial foreground pixels are used to characterize the object as in [6]. In this work, only the components a and b of the Lab color systems are considered to characterize the object color distribution. For simplicity, the histogram is defined over a uniform partition.

The object distribution function is modeled as

$$\mathrm{p}_O(p) = \frac{n_k}{n_O} \tag{5}$$

where n_k is the number of pixels assigned to the bin k and n_O is the number of pixels in the object region O.

Observe that only one of the input images is used to construct the histogram information, since mixing different images may distort color information.

In most cases, we use the image corresponding to the lowest projected intensity.

The regional term of the energy function is:

$$R(x_p) = \begin{cases} -\log(\mathrm{p}_O(p)), & \text{if } x_p \text{ is 1} \\ -\log(\mathrm{p}_B(p)), & \text{if } x_p \text{ is 0} \end{cases} \quad (6)$$

where 1 is foreground and 0 is background.

The likelihood function for neighboring boundary pixels given by

$$\mathrm{B}(p,q) = 1 \; - \; \exp(\frac{-(||Lab(p) - Lab(q)||)^2}{2\sigma_C^2}), \quad (7)$$

where $Lab(p)$ denotes the color at point p and σ_C is the standard deviation of the L^2-norm of the color difference.

The boundary term for neighboring pixels p, q is given by $-|x_p - x_q| \log \mathrm{B}(p,q)$, where points q are the neighbors of p.

The final objective function combines both the regional and the boundary term and is given by:

$$E(\mathbf{X}) = \sum_{p \in I_1} R(x_p) - \sum_{p,q \in I_1} |x_p - x_q| \cdot \log \mathrm{B}(p,q), \quad (8)$$

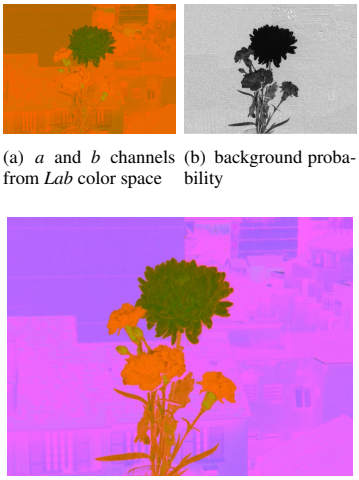
As shown in [11], the proposed energy function is regular, which means that it can be minimized by graph-cuts. This remains valid for the modified energy function defined on quadtrees leaves. As a consequence, Quad-Cuts can be applied to minimize the modified energy function.

6.2 Energy Function in GPU

The next sections describe how shaders can be used to compute efficiently the regional and boundary terms of the active illumination energy function applying GPGPU.

To pass the computed data efficiently across the algorithm we create what we call a *Stratified Texture*, illustrated in Figure 6.

The Stratified Texture is generated by saving, in its different channels, red, green, blue and alpha, all the data needed for the following steps of our algorithm. In this example application, the red and green channels are used for storing the a and b channels of the input image converted to Lab color space, the blue channel for storing the initial seed segmentation obtained by thresholding the luminance difference, and the alpha channel for storing the background distribution.



(a) a and b channels from *Lab* color space (b) background probability

(c) RGBA are respectively *a, b*, segmentation seed and background probability

Figure 6: Stratified Texture.

6.2.1 Color Space Conversion

The input images are converted from RGB to CIE Lab color space, to exploit metrics in a perception-based color space presenting orthogonality properties between luminance and chrominance information.

Shaders for color space conversion have been used intensively by GPGPU programs. However, in order to efficiently compute the RGB to Lab conversion with high precision we also take advantage of the concept of rendering to texture with 32 bit floating point internal format using frame buffer objects (FBO) [19]. We save the Lab a and b computed channels in the resulting texture r and g channels, as illustrated in figure 6(a).

6.2.2 Background Probability

The background probability is computed in GPU according to equation (4), measuring the distribution of the luminance difference of the lit and unlit images. The result is illustrated in Figure 6(b).

For efficiently using the GPU parallelism, we pre-compute the constants $1/\sqrt{2\pi}\sigma_L$ and $1/(2\sigma_L^2)$ of equation (4) for a fixed σ_L. Those values are passed to the shader, avoiding repeatedly calculating it for every fragment.

6.2.3 Computing the Color Distribution

In order to compute the object distribution function using equation (5), we construct the histogram of the a and b channels from Lab color space (saved in stratified texture red and green channels), distinguishing object pixels using the object seed (from the stratified texture blue channel).

Motivated by its performance in computing histograms with a large set of bins, we choose to adapt [12] to our application context. Originally, that approach was proposed for monochromatic histograms, computing the histogram bin selection in a vertex shader, by loading the texture using either vertex texture fetches or by rendering the input image pixels into a vertex buffer, according to the graphics hardware capability.

We propose to adapt [12] to a vertex shader that computes bin selection in a 2D mapping, modifying it to compute a histogram representing the frequencies of occurrence in both input channels. Our vertex shader computes the vertex position by reading the a and b channels, multiplying their normalized values by the number of bins in the corresponding dimension, and then transforming the resulting values to frame coordinates.

Observe that a histogram of a trichromatic image could also be computed in GPU using techniques for representing 3D arrays such as those proposed in [18].

6.3 Application pipeline

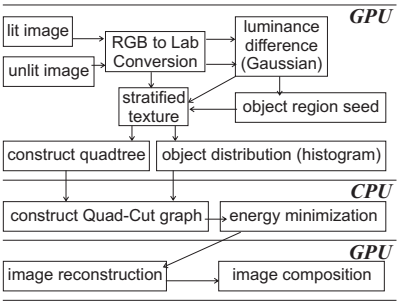


Figure 7: The proposed Quad-Cut method.

The main steps of the example application are illustrated in Figure 7.

The lit and unlit input images are converted to Lab color space. Then, another shader computes the background distribution texture. The result of those shaders are grouped in the stratified texture as described in section 6.2 and illustrated in figure 6.

The object distribution function is obtained by computing the object histogram of the a and b channels read from the stratified texture, using only pixels that failed the background threshold test (read from its blue channel). This histogram is saved in a texture to be used later in the energy function construction.

Then, the quadtree is created using our reduction operator through the stratified texture. Following the method in section 5.3, the resulting pyramid texture in cleaned, generating a texture that contains only the leaf nodes. that contains all information needed about each leaf: its level, from its relative position in the texture; its a and b from LAB conversion saved in the red and green channels; and the luminance distribution, saved in the blue channel.

All the above steps are computed in GPU. After them, the graph is constructed in CPU by reading the data from the leaf texture (fig. 4) and from the histogram textures.

In CPU we store the quadtree leaves in a pointer less representation, as a linear quadtree. The leaves are associated with location codes for fast neighbor search as in [16].

The graph is constructed using the leaf data, which stores the previously computed terms of the objective function, according to the method explained in section 4, which is minimized by the Graph-Cut minimization as in [1].

The solution of the minimization provides the classification of the quadtree leaves as background or foreground. So, using the position and size of each leaf, we reconstruct the resulting image that represents the alpha mask solution.

Back to the GPU, for the final composition, a smooth shader is applied to the computed alpha mask. Finally, a blending operator $\alpha F + (1 - \alpha)B$ is applied to the segmented foreground and the new background.

6.4 Results

Segmentation results using Quad-Cuts and the final compositions are shown in figures 2 and 8. To illustrate the considerable reduction in the number of variables in the minimization problem, both figures 2 and 8 are originally 800×600 (480,000) pixels, while the computed quadtrees have 9,556 (2%)

leaves and 30,036 (6%) leaves, respectively. Notice that the special characteristics of figure 8 (that presents many holes and thin structures) are automatically preserved through 15,992 leaves in the lowest level (1×1 pixel) and 8,718 in the next level (4×4 pixels).

(a) Segmentation seed

(b) Output α-channel

(c) Composite

Figure 8: Composition Result 2 (using $\sigma_L = 0.25$, $\sigma_C = 0.05$).

We also measured the execution time of an background/foreground segmentation using graph-cut and active illumination with a Quad-Cut implementation with its preprocessing steps computed in GPU. A NVIDIA GeForce 7900 graphic card was used for the timings shown in Table 1.

Table 1: Processing time

step	*in seconds*
Energy function on GPU:	
RGB to Lab	< 0.001
Background prob	< 0.001
Histogram	< 0.015
Quad on GPU:	
Pyramid Construction	0.047
Quad Leaves Isolation	< 0.001
Quad on CPU:	
Reading Texture to CPU	0.015
Leaf List	0.016
Neighborhood	0.014
Graph-cut Minimization	0.001
Answer Reconstruction	0.016

7 Conclusions

We propose to accelerate the computation of energy minimization using graph cuts by applying a preprocessing step for reducing the number of graph nodes and edges. In this pre-processing, pixels are grouped by a similarity criteria according to the problem context.

We argue in favor of using a quadtree structure for managing such clustering regions, motivated by the easily retrievable neighborhood system between its leaves. In order to support our claim, we present a general formulation of the energy function using the leaves as its variables, and we also presented a general graph-cut construction over the quadtree leaves.

We also show how the quadtree structure can be constructed using graphics hardware. Initially, we use a reduction operator for constructing an image pyramid that writes in each texel whether a similarity clustering was applied or not. Such shader is simpler than the one proposed in [15]. Then we propose a leaf isolation method that discards from the pyramid texture all the texels that do not represent a quadtree leaf, efficiently removing unneeded information of non-leaf nodes. The proposed method requires fewer texture readings than the method proposed by [15], due to fact that the algorithm that it employs for finding leaves does not compute tree traversals for discovering each leaf in the tree.

Our graph construction method does not compute a point list on GPU of the quadtree leaves, as [15] does. Instead, as explained before, we use the leaf texture data to save the weights of the computed energy function, and the leaf texture coordinates are used to set the leaf level, size and corner position. Saving all the data needed for the posterior steps into such leaf texture allows an efficient interplay between the result generated in GPU and the energy minimization on CPU.

We also presented an application of our method to the foreground/background segmentation prob-

lem. It can be observed from the presented results (figures 2 and 8) that the proposed method for grouping pixels into quad leaves conserved image fine grain details of the original image (by creating leaves as small as 1×1) while also featuring a good grouping rate, by creating large leaves in regions of similar pixels .

We also show that the efficient implementation of all preprocessing steps on GPU leads to reasonably fast processing rates. As a consequence, we believe that our method constitutes an important step towards real time segmentation and matting using active segmentation.

References

[1] Y. Boykov, O. Veksler and R. Zabih. Fast Approximate Energy Minimization via Graph Cuts. *IEEE Transactions on PAMI,* 23(11):1222-1239, 2001.

[2] Y. Boykov and V. Kolmogorov. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Computer Vision. In *Proc. of Int'l Workshop Energy Minimization Methods in Computer Vision and Pattern Recognition,* 2001.

[3] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *Proc. IEEE Transactions on Pattern Analysis and Machine Intelligence,* 26(2):147-159, 2004.

[4] R. Zabih and V. Kolmogorov. Spatially Coherent Clustering Using Graph Cuts. in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'04),* 2:437-444, 2004.

[5] J. Yun-Tao and H. Shi-min. Interactive Graph Cut Colorization. *The Chinese Journal of Computers,* 29(3):508-513, 2006.

[6] C. Rother, V. Kolmogorov and A. Blake. GrabCut - Interactive Foreground Extraction using Iterated Graph Cuts. *ACM Trans. Graph.,* 23(3):309–314, 2004.

[7] J. Wang, P. Bhat, R. A. Colburn, M. Agrawala and M. F. Cohen. Interactive Video Cutout. In *Computer Graphics Proceedings ACM SIGGRAPH,* 2005.

[8] Y. Li, J. Sun, C. Tang, H. Shum. Lazy Snapping. *ACM Trans. Graph.,* 23(3):303–308, 2004.

[9] A. Agrawala, M. Doncheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin and M. F.Cohen. Interactive Digital Photomontage. In *Computer Graphics Proceedings ACM SIGGRAPH,* 294-302, 2004.

[10] V. Kwatra, A. Schdl, I. Essa, G. Turk, A. Bobick. Graphcut Textures: Image and Video Syntesis using Graph Cuts. In *ACM Transactions Graphics, Proc. SIGGRAPH,* 22(3):277-286, 2003.

[11] A. Sá, M. B. Vieira, A. Montenegro, P. C. Carvalho and L. Velho. Actively Illuminated Objects using Graph-Cuts. In *Proceedings of SIBGRAPI,* 2006.

[12] T. Scheuermann and J. Hensley. Efficient histogram generation using scattering on GPUs. In *SI3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games,* 33–37, 2007.

[13] O. Fluck, S. Aharon, D. Cremers and M. Rousson. GPU histogram computation. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Research posters,* 53, 2006.

[14] S. Green. Image Processing Tricks in OpenGL. In *Game Developers Conference (GDC05), 2005.*

[15] G. Ziegler, R. Dimitrov, C. Theobalt and H.-P. Seidel. Real-time Quadtree Analysis using HistoPyramids. In *IS&T and SPIE Conference on Electronic Imaging,* 2007.

[16] S. F. Frisken and R. Perry. Simple and Efficient Traversal Methods for Quadtrees and Octrees. *Journal of Graphics Tools,* 7(3):1-11, 2002.

[17] D. Goddeke. Playing Ping Pong with Render-To-Texture. *http://www.mathematik.uni-dortmund.de/ĝoeddeke,* 2005.

[18] M. Harris, D. Luebke, I. Buck, N. Govindaraju, J. Kruger, A. Lefohn and T. Purcell. GPGPU: General-Purpose Computation on Graphics Hardware. In *Tutorial at ACM SIGGRAPH 2005,* 2005.

[19] S. Green. The OpenGL Framebuffer Object Extension. In *Games Developers Conference (GDC),* 2005.

Improving the Data Quality of PMD-based 3D Cameras

Alexander Sabov, Jörg Krüger

Fraunhofer Institute for Production Systems
and Design Technology (IPK)
Pascalstrasse 8-9, D-10587, Berlin, Germany
Email: alexander.sabov@ipk.fraunhofer.de

Abstract

Reproducing the environment as a virtual model in a computer is a requirement of many vision, modeling and vizualisation applications. 3D video range cameras are a promising approach since they provide direct 3D information of a recorded scene. This paper focuses on improving the data quality of such cameras by analysing and processing the raw values. The first algorithm focuses on the quality of the distance values. The optimal exposure times for the camera are predicted and the resulting distance values are combined to create an improved depth image. The second algorithm focuses on the quality of grayscale values. An illumination model is presented which is used to eliminate spotlight effects in the picture. An evaluation on test scenes demonstrates the effectiveness of the algorithms.

1 Introduction

3D-imaging is a technique for capturing and displaying three dimensional information of the environment. A common method is stereoscopic imaging where two pictures are taken which have a horizontal displacement similar to the view of the human eyes. The length of the displacement allows drawing conclusions about the depth information of the scene. Complex and time consuming algorithms are needed for finding correlating points in the pictures and calculating the distance values.
A 3D video range camera is a new promising alternative for capturing three dimensional information. The optical sensor is a Photonic Mixer Device (PMD) which provides direct depth information for each pixel of an image. High frame rates can be achieved allowing capturing and processing the data in real-time. Several companies provide range cameras with different specifications [1] [2].
The cameras still have some limitations that must be overcome to improve the quality of the 3D data.

The main limitations are:

- Low resolution (maximum: 160x120 pixels)
- Accuracy and noise of distance data
- Interference with background illumination (sunlight, fluorescent lamps)
- Low range (maximum: 7.5 m)

The mentioned limitations are the subject of current research activities [3] [4]. Since PMD is a new technology it can be assumed that there will be much improvement over the next years.
Additional requirements must be fulfilled depending on the application the camera is used for. For example, in the application of Autonomous Mobile Systems (AMS) the PMD sensor is used for navigation purposes [5]. The reliability of data and measurements for the accuracy is of high priority for such applications. In this paper two algorithms are presented that address the improvement of the data.

2 Background

The theoretical background on range cameras has been described in many publications [6] [7] [9]. The following sections give a brief description and a summary of the formulas needed to calculate the final distance values.

2.1 Range Camera

Range cameras are based on the time of flight (TOF) principle [6]. The simplest form of a TOF measurement is the emission of a light pulse. The light is reflected from the environment and a receiver close to the sender captures the reflected light pulse and measures the time the light needed to travel.
Since the speed of light is known it is possible to

calculate the distance the light travelled. The light travels to the target and back again. Therefore, the distance to the object is half of the light travel distance.
Such direct light measurements require very precise electronic components which are capable of measuring time differences of nano seconds. An alternative is to send a modulated wave with a long wavelength and to measure the phase difference between the sent and received modulated wave. Since the phase difference is only unique up to half of the wavelength the possible maximum distance measurement d_{max} is limited to this value as well.

$$d_{max} = \frac{c}{2 \cdot f_{mod}} \quad (1)$$

where
c: speed of light (299792458 m/s)
f_{mod}: modulated frequency of the wave

The camera system we used [1] contains a matrix of infrared LEDs (Figure 15) to send a carrier wave which is modulated by a 20 MHz signal (Figure 1).

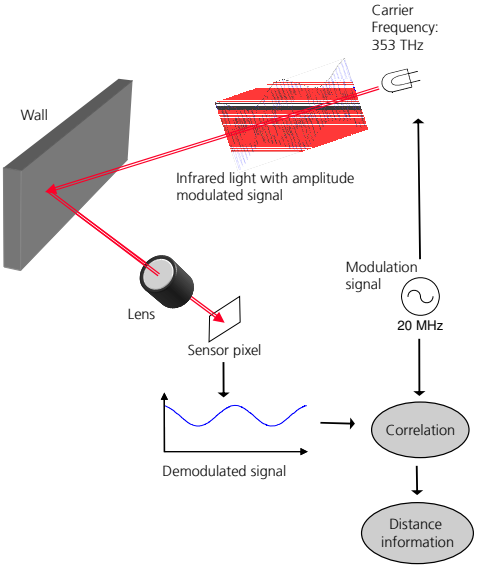


Figure 1: Principle of the PMD camera

This results in $d_{max} \approx 7,5m$. A lens captures the reflected light waves on the PMD sensor. The incoming modulated carrier signal is demodulated and cross-correlated with the original modulation signal to retrieve the phase difference.

2.2 Retrieving Distance Data

The distance information is retrieved by determining the phase difference between the modulation signal and the received demodulated signal [7]. For the derivation of the formulas we can assume an ideal sinusoidal modulation signal $ms(t)$ and a carrier signal $cs(t)$ (Figure 2):

$$ms(t) = cos(\omega \cdot t) \quad (2)$$
$$cs(t) = A_c \cdot cos(\Omega \cdot t) \quad (3)$$

with $\Omega >> \omega$

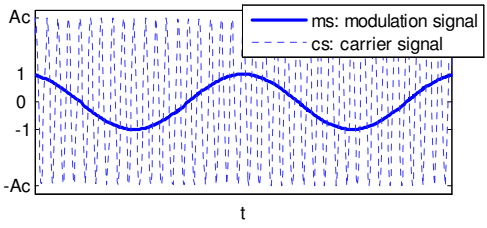


Figure 2: Modulation

An amplitude modulation of $ms(t)$ on $cs(t)$ results in (Figure 3):

$$mds(t) = (A_c + cos(\omega \cdot t)) \cdot cos(\Omega \cdot t) \quad (4)$$

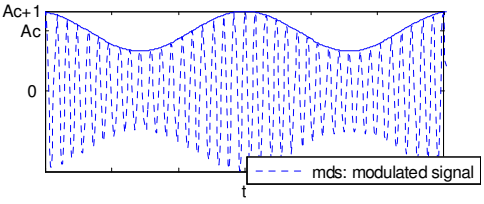


Figure 3: Modulated wave

The received signal $rs(t)$ contains the phase shift φ we want to determine. The energy loss of the reflected wave results in a descaling factor A for $rs(t)$

which affects the amplitudes of $rs(t)$ and the offset A_i of the amplitude modulation linearly (Figure 4):

$$rs(t) = (A_c + cos(\omega \cdot t - \varphi)) \cdot cos(\Omega \cdot t) \cdot A \quad (5)$$

$$ds(t) = (A_i + A \cdot cos(\omega \cdot t - \varphi)) \quad (6)$$

$$\text{with } 0 \leq A \leq 1 \text{ and } A_i = A \cdot A_c$$

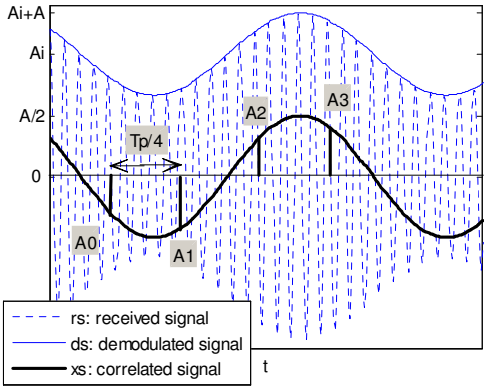


Figure 4: Signal demodulation and cross-correlation

The demodulated signal $ds(t)$ corresponds to the modulation signal by a shift of A_i, an Amplitude of A and a phase shift of φ. The resulting cross-correlation $xs(\tau)$ is calculated by:

$$xs(\tau) = ms(t) \otimes ds(t) = \lim_{T \to \infty} \frac{1}{T} \int_{-\frac{T}{2}}^{+\frac{T}{2}} ms(t) \cdot ds(t+\tau)\, \partial t = \frac{A}{2} \cdot cos(\omega \cdot \tau + \varphi) \quad (7)$$

While the offset information A_i is lost during the cross-correlation, the Amplitude A and phase φ can be determined from $xs(\tau)$. Since $xs(\tau)$ is a sinusoidal signal with known frequency the signal can be reconstructed by using only four sampling points $A_0 - A_3$ at an interval of one quarter of the modulation period T_p [8]:

$$A = \frac{\sqrt{(A_1 - A_3)^2 + (A_2 - A_0)^2}}{2} \quad (8)$$

$$\varphi = \arctan\left(\frac{A_1 - A_3}{A_2 - A_0}\right) \quad (9)$$

The phase φ in the interval $[-\pi, \pi]$ has a linear relationship to the distance d:

$$d = \left(d_{max} \cdot \frac{\pi + \varphi}{2 \cdot \pi} + \text{offset}\right) \bmod d_{max} \quad (10)$$

The offset describes the time delay based on the electronic components of the camera. The value is calibrated for each camera by the manufacturer. The offset addition is limited to the maximum distance by the modulo operator.

The camera we used [1] contains a sensor chip with two output channels. The correlation of the signals is done directly on each pixel. The incoming photons generate charge carriers which are transferred to the output channels. The modulation signal $ms(t)$ controls to which channel the electrons will be transferred [9]. The charge difference between the channels is provided by an A/D-converter. The values of the converter are the raw values which the camera provides on request. The values correspond to the sample values $A_0 - A_3$ plus a positive shift due to the electronic measurement principle of the converter. Since both A and φ are invariant to a shift the raw values can be used directly as sample points.

3 Multiple Integration

The measurements of the PMD chip are based on the absorption of photons and generation of charge carriers. The integration time is the time the chip is exposed to the incoming photons. It has a direct impact on the accuracy of the measurement. The aim is to retrieve as many measurements with acceptable accuracies as possible.

Figure 5 shows some measurements on a wall with varying wall color and real distance. The measured distance values only overlap with the real distance in a specific section which differs in position and length. While the measurement in Figure 5b seems to be stable above 5000 μs, Figure 5a returns wrong measurements after 15000 μs and Figure 5c after 30000 μs.

Two effects are responsible for the erroneous values. The erroneous values on the left are due to insufficient illumination for signal detection. The erroneous values on the right are due to a limitation of possible number of charge carriers on the integration capacity which leads to a saturation. Please refer to [13] for a detailed explanation.

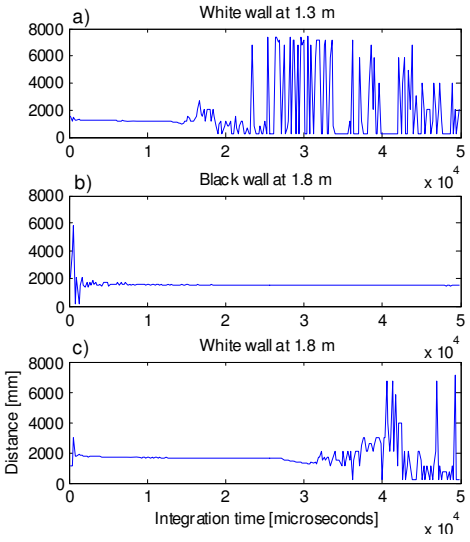


Figure 5: Distance measurements

3.1 Optimal Integration

The accuracy of a measurement can be determined directly by the following formula [10] [11]:

$$\Delta d = \frac{d_{max}}{\sqrt{8}} \cdot \frac{\sqrt{I}}{2 \cdot A} \tag{11}$$

I is the intensity of the light which corresponds to the mean amplitude $rs(t)$ and to the offset A_i of $ds(t)$ as well. Since the value of A_i is lost during the cross-correlation process and the camera does not provide the charge values of the channels separately the intensity can not be calculated exactly. Since A_i is proportional to the amplitude A of $ds(t)$ we can estimate I by a constant factor k:

$$I = k \cdot A \tag{12}$$

This results in:

$$\Delta d = k \cdot \frac{d_{max}}{\sqrt{8}} \cdot \frac{1}{2 \cdot \sqrt{A}} \tag{13}$$

We calibrate k by using the noise range of a pixel. The camera is pointed towards a wall at a fixed distance. We measure the amplitude of the pixel and use the standard deviation of the measured distance for Δd. This process can be repeated for other distances if needed.

The accuracy of the measurement in Equation 13 only depends on the amplitude of the signal. Figure 6 shows the corresponding amplitudes to the distance measurements of Figure 5. The graphs show a bump shape with a maximum value at 200. In practical experience the measurements already show saturation effects when using integration times on the right side of the maximum. Therefore, acceptable integration times are values left from the maximum with an amplitude value above a defined minimum. The optimal integration time is located at the maximum amplitude position.

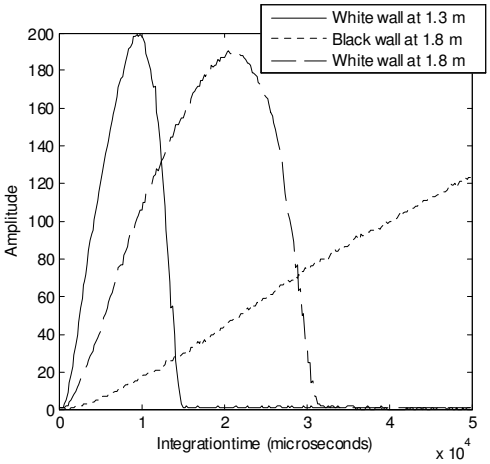


Figure 6: Amplitude measurements

3.2 Number of Measurements

For a complex scene a single measurement may not be enough. If the scene contains multiple walls and objects with different material properties and distances the optimal integration times will be different for each pixel.

Figure 7 illustrates the problem. The camera is pointed towards a wall and a box with a hole in the center is placed in front of it. The distance between the camera and the wall is 1.2 m. The distance between the camera and the wall is 2.35 m. The figures show the detected 3D points of the scene from a bird's-eye view with the wall on the right, the box on the left side and the camera (which is not shown) on the left of the box. In Figure 7a an optimal integration time has been chosen for the box and the

3D points of the box create a sharp shape. The wall behind it appears very scattered due to insufficient amplitude values. In Figure 7b an optimal integration time has been chosen for the wall. The wall appears in a sharp shape while the shape of the box can not be recognized any more. Only the combination of both measurements gives an adequate result for the complete scene (Figure 7c). The wall and box both appear sharp in the scene.

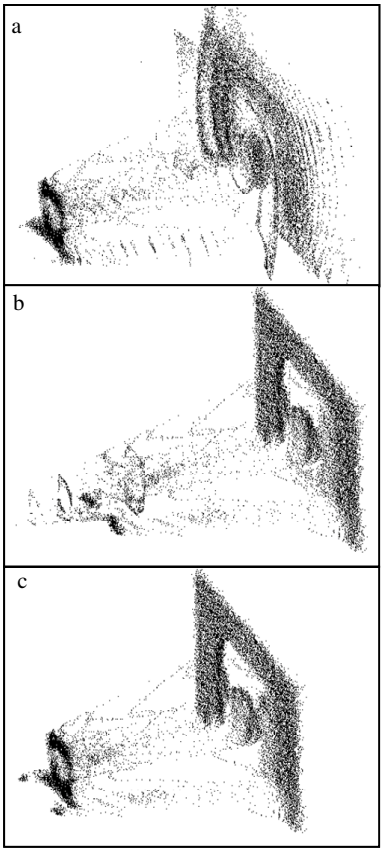


Figure 7: 3D points for different integration times (a,b) and after multiple integration (c)

Since each additional measurement requires time the aim is to cover a long depth range with as few measurements as possible. Also, for AMS applications a method is needed which determines the optimal integration times automatically.

3.3 Algorithm

We propose the following algorithm, which is illustrated in Figure 8:

- Perform three measurements $m_1 - m_3$ with the first three integration times $t_1 - t_3$ and determine the amplitude gradients A_l and A_r for each pixel.

$$A_l = \frac{m_2 - m_1}{t_2 - t_1} \tag{14}$$

$$A_r = \frac{m_3 - m_2}{t_3 - t_2} \tag{15}$$

  Since the integration times are very low the overall time should be low as well.
- Classify the section of the bump (1 to 4) by comparing the signs and values of the gradients and the absolute value of the amplitude. Please note that the different locations of the three measurements in Figure 8 are only for illustration. In reality the measurements are fixed on the left side and the graph is shifted.
- Case 1,3: Select the higher gradient to calculate the integration time IA_{max} with maximum amplitude A_{max}(200). The lower gradient could include a part of the floor line and lead to wrong results.
- Case 1,2,3: Approximate the left part of the bump by a straight line from zero to the point of maximum amplitude. Calculate the integration time IA_{min} with minimum acceptable amplitude A_{min}.

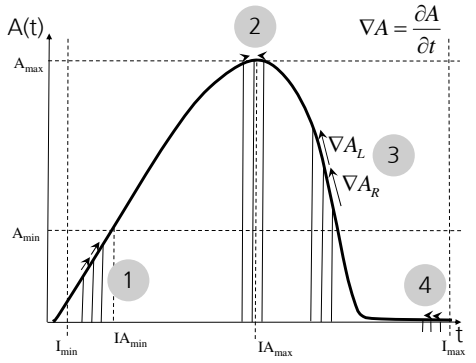


Figure 8: Determination of the amplitude maximum

After IA_{min} and IA_{max} values are determined for each pixel we can plot the distribution of optimal values A_{max} of the scene (Figure 9). We can identify the box in front of the wall by the peak of the second bar in the histogram. Most of the optimal A_{max} values for the wall lie above our maximum allowed integration time and are therefore all stacked at I_{max}, the last bar in the figure.

The two highest peaks could be considered as

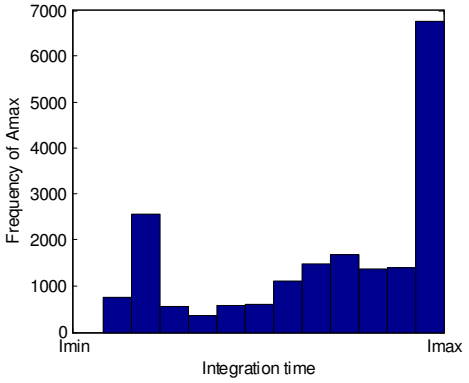


Figure 9: Histogram of integration times for maximum amplitudes

the result for a double integration and would result in 35% of the pixels having an optimal integration time for this scene. But having as many pixels with acceptable amplitudes as possible has a higher priority than having optimal amplitudes. The acceptable amplitudes for a pixel lie between IA_{min} and IA_{max} for each pixel. The corresponding histogram is shown in Figure 10. The best coverage of acceptable amplitudes is the highest bar in the figure (5th from the left). The related integration time is the first result of the algorithm.

For determining the next best integration time we need to exclude all pixels that are already covered by the first measurement. We create the histogram of acceptable amplitudes again (Figure 11). The figure shows now a breach at the position of the first result since all pixels having maximum amplitudes close to this position most likely also cover it and are therefore excluded. The 5th bar itself is empty since all pixels with maximum amplitude at this position have an acceptable amplitude at this position as well. The first bar in the figure (from the left) is

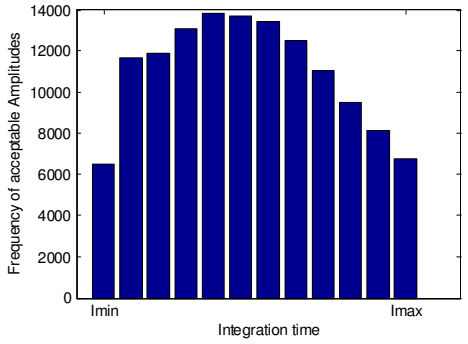


Figure 10: Histogram of integration times for acceptable amplitudes

now the second best coverage of acceptable amplitudes.

This process can be repeated until the maximum

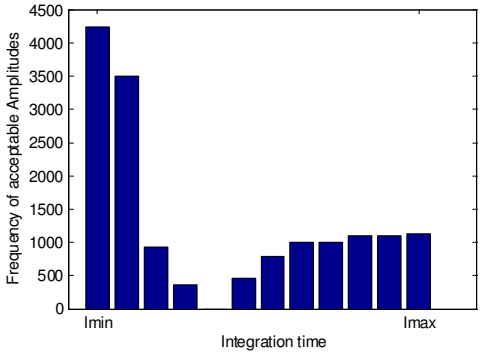


Figure 11: Remaining histogram for second integration

number of allowed multiple integrations is reached. The resulting pixels that are not covered can be assigned to one of the defined integration times. Since a very low amplitude on the left bump side is still better than a oversaturated amplitude they are assigned to the result closest to their maximum amplitude from the left side.

3.4 Evaluation

We evaluated the algorithm for the scene in Figure 7 for a double integration. We set A_{min} to a value of 50 and A_{max} to 200. We chose a step size of 4000 μs. The first three integration times were 4000, 8000 and 12000 μs. The minimum slope to classify case 1,2 or 3 was $2.4e^{-5}$.The possible integration times were limited by $I_{min} = 4000\ \mu s$ and $I_{max} = 50000\ \mu s$. The first optimal integration time found was 20000 μs. It corresponds to the background wall which has more pixel than the box in front of it. The second optimal integration time was 4000 μs and corresponds to the box. Table 1 shows the results of the evaluation and Figure 7c the resulting image.

Table 1: Number of pixels with acceptable accuracy after multiple integration.

Number of final measurements	2
Estimated number of pixels with acceptable amplitudes	94%
Measured number of pixels with acceptable amplitudes	83%

When measuring the speed of the camera capturing, data transfer and algorithm we noticed that most of the time is spent at camera capturing. Though the integration times for the first three measurements are very low, the camera still requires time values in the range of 100-200 ms for this process. This is irrelevant for offline processing but it is a bottleneck for real-time-capturing. We hope that these time values will improve in cameras of the next generation. The algorithm itself had no noticable calculation time. It was below 1 ms on a 2 GHZ PC.
We noticed an unsteadiness in straight walls when merging values from different integration times. When we investigated the wall from a fixed position with increasing integration times we observed that the position measurement of the wall changes constantly towards the camera. When merging two different integration times, two slightly different wall positions are merged as well. In future work we want to eliminate this effect by calibrating the camera for different integration times.

4 Illumination

The phase difference between the outgoing and incoming modulated wave allows the determination of the distance to the reflected points (refer to Equation 10).
Additionally, a grayscale image of the scene can be generated. The value I of Equation 12 (which is equivalent to A_i of the signal $ds(t)$ in Equation 6) describes the intensity of the modulated infrared light reaching a sensor pixel.

The following factors influence the intensity values:

- the radiant flux of the LEDs
- the radiation characteristics of the LEDs
- the orientation of the LEDs, target, and camera
- the target distance and reflectivity

An intensity image of a scene is shown in Figure 12a. The matrix of LEDs creates a spotlight in the center. The spotlight intensity decreases to the outer edges.
For AMS applications the texture of the walls can be of great importance. This requires grayscale images that have a more homogenous illumination like pictures from a digital camera taken by daylight. A wall with a homogenous color should also appear homogeneous in the grayscale image.
To eliminate the spotlight effect in the grayscale images we set up an illumination model which takes into account the different influencing factors except the target reflectivity. Then we divide the measured intensity by this value to obtain a grayscale image based on the target reflectivity.

4.1 Illumination model

Figure 13 shows the vector geometry used for the model. The illumination model is based on the following laws and functions:

1. The radiation characteristic $ra(\alpha)$ of the LEDs
2. The inverse-square law for light intensity: $I \propto \frac{1}{r^2}$
3. Lambert's emission law $I \propto cos(\theta)$ for lambertian surfaces
4. The angle between light beam and sensor $s(\beta)$

The resulting formula for the expected illumination for a target $\vec{p}$ is:

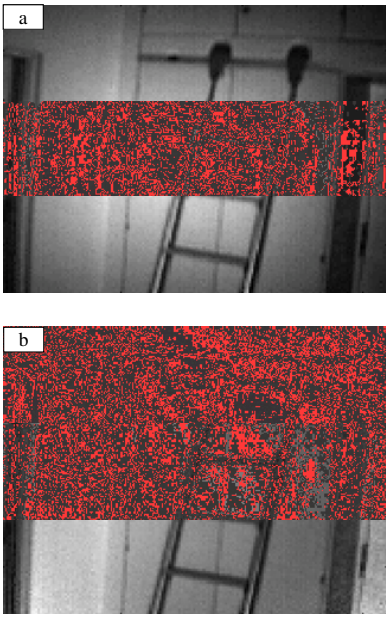


Figure 12: a) Grayscale image of intensity values; b) Grayscale image without spotlight

$$\text{radiance}(\vec{p}) \propto \sum_{i=0}^{n} \left(\frac{\text{ra}\,(\alpha) \cdot \vec{d_w} \cdot \left(\vec{l_i} - \vec{p}\right)}{\left|\vec{p} - \vec{l_i}\right|^2} \right) \cdot s(\beta) \quad (16)$$

$$\text{with} \quad \alpha = \text{acos}\left(\frac{\left(\vec{p} - \vec{l_i}\right) \cdot \vec{d_l}}{\left|\vec{p} - \vec{l_i}\right|} \right) \quad (17)$$

where
$\vec{d_l}$: normalized directional vector of LED
$\vec{d_c}$: normalized directional vector of camera
$\vec{d_w}$: normalized directional vector of wall
$\vec{l_i}$: position of LED with index i
n: number of LEDs
$s(\beta)$: dependency on angle between light beam and sensor

The radiation function is determined from the datasheet [12] of the LEDs (Figure 14). The function values are linearly interpolated using a lookup table. The LEDs are arranged in a matrix structure on both sides of the camera (Figure 15).

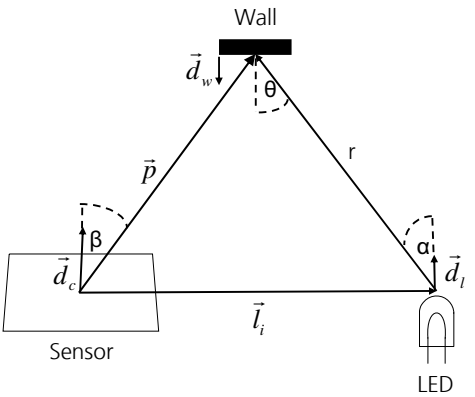


Figure 13: Vector geometry for illumination model

Figure 16 shows the intensity distribution of an imaginary horizontal plane going through the camera. The distribution is based on the first two emission laws/functions and the position of the LEDs.

The camera is located at x=0 and z=0 and points into the z direction. Red color represents high intensity and blue color low intensity. The spotlight appearance can be recognized in the figure.
The orientation of the wall must be known in order to apply Lambert's emission law. It can be determined by analysing the neighbour pixels. Eliminating Lambert's emission law from the grayscale values eliminates shading effects in the environment as well. This is very useful for extracting texture information but should not be done for grayscale images of the scene. Otherwise a scene containing only white walls would result in a completely white image.
The function $s(\beta)$ has been determined empirically. A white wall has been recorded which should result in a white image with equal values. The measured intensity values have been divided by the calculated intensity of Figure 16. The result is plotted in Figure 17.
It seems that there is a linear correlation with the angle between light beam and sensor. Only for very low distances the correlation seems not to be valid. A linear regression results in a line with a zero point at β_0=22°. Since law number 3 has not been taken into account the slope of the regression line is different for different scenes but the zero point seems to be stable. Therefore we can take the slopes of the

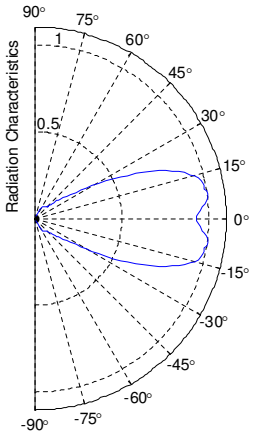


Figure 14: Radiation characteristics for LEDs

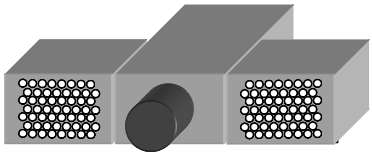


Figure 15: Camera with matrix of LEDs

regression line as the result for the multiplication factor $s(\beta)$:

$$s(\beta) = \frac{1}{\beta - \beta_0} \tag{18}$$

4.2 Evaluation

We use the simple scene with the white wall again for evaluating the illumination model. If the camera points perpendicular to the wall the resulting pixel values should be of equal value. For this scene the standard deviation is a measurement of the effectiveness of the illumination model. Ideally it should be zero.
Table 2 compares the standard deviation of the scene without and with the illumination model. It could be decreased by 39%. A resulting grayscale image for a complex scene is shown in Figure 12b.

The fact that the camera has its own light source appears to be a disadvantage at the beginning for

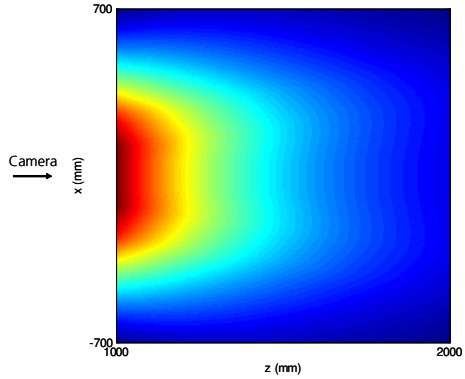


Figure 16: Distribution of horizontal radiation values (red: high, blue: low)

Table 2: Standard deviation of the measurements on a homogeneous white wall

Standard deviation of the normalized intensity values	0.2132
Standard deviation of the normed radiance values	0.1293
Reduction of the standard deviation by	39%

grayscale values. But after applying the illumination model it turns out to be a great advantage for retrieving the original texture data. Since the light source is located close to the camera no shadowing effects appear in the picture.

5 Conclusion

In this paper two algorithms have been proposed to improve the data quality of 3D video range cameras. The first algorithm used a method of multiple integration to overcome the low accuracy range of the camera. The second algorithm considered an illumination model to create grayscale images with homogenous illumination. The evaluation of the algorithms proved that they enhance the quality of the data. Future work on improvement will focus on the calibration of the camera system and distance data.

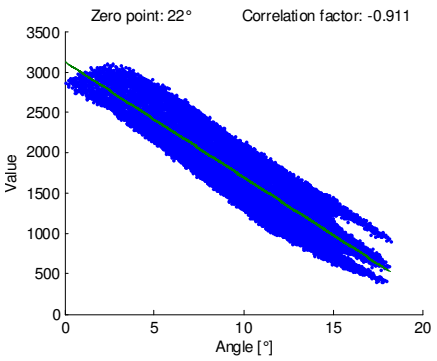


Figure 17: Dependency of pixel values on angle between light beam and sensor

References

[1] PMD Vision 19k - 3D video range camera. PMDTechnologies GmbH, 2007.

[2] SwissRanger CSEM SA.

[3] Universität Siegen. DFG Forschungspaket - Dynamisches 3D-Sehen (Dyn3D). 2007.

[4] M. Lindner and A. Kolb. Lateral and Depth Calibration of PMD-Distance Sensors. In *International Symposium on Visual Computing (ISVC06)*, 524–533, 2006.

[5] S. May, B. Werner, H. Surmann and K. Pervölz. 3D time-of-flight cameras for mobile robotics. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2006)*, 790–795, 2006.

[6] R. Lange. 3D Time-of-Flight Distance Measurement with Custom Solid-State Image Sensors in CMOS/CCD-Technology. *Dissertation University Siegen*, 2000.

[7] T. Oggier, R. Kaufmann, M. Lehmann, P. Metzler, G. Lang, M. Schweizer, M. Richter, B. Büttgen, N. Blanc, K. Griesbach, B. Uhlmann K.-H. Stegemann and C. Ellmers. 3D-Imaging in Real-Time with Miniaturized Optical Range Camera. In *6th Int. Conf. for Opt. Technologies (OPTO 2004)*, 89–94, 2004.

[8] 4-Phasenalgorithmus aus den Rohdaten. Internal paper. PMDTechnologies GmbH, 2004.

[9] T. Möller, H. Kraft, J. Frey, M. Albrecht and R. Lange. Robust 3D Measurement with PMD Sensors. In *Proceedings of the 1st Range Imaging Research Day*, 2005.

[10] T. Oggier, M. Lehmann, R. Kaufmann, M. Schweizer, M. Richter, P. Metzler, G. Lang, F. Lustenberger and N. Blanc. An all-solid-state optical range camera for 3D real-time imaging with sub-centimeter depth resolution (SwissRangerTM). In *SPIE–The International Society for Optical Engineering*, Volume 5249, 534–545, 2004.

[11] B. Büttgen, T. Oggier, M. Lehmann, R. Kaufmann, F. Lustenberger CCD/CMOS Lock-In Pixel for Range Imaging: Challenges, Limitations and State-of-the-Art. *CSEM,Swiss Center for Electronics and Microtechnology*, 2007.

[12] *Datasheet for High Power Infrared LED SFH 4259*, OSRAM, 2005.

[13] T. Krieger. Innovative Sensorkonzepte und Signalverarbeitungsstrategien zur Bewegungserkennung und Präsenzkontrolle von Personen. *Dissertation University Siegen*, 2002.

A Direct Numerical Approach to Perspective Shape-from-Shading

Oliver Vogel, Michael Breuß, Joachim Weickert

Mathematical Image Analysis Group, Faculty of Mathematics and Computer Science,
Saarland University, Building E1.1, 66041 Saarbrücken, Germany.
Email: {vogel,breuss,weickert}@mia.uni-saarland.de

Abstract

We introduce a novel numerical method for a recently developed perspective Shape-from-Shading model. In order to discretise the corresponding partial differential equation (PDE), Prados et al. employed the dynamical programming principle yielding a Hamilton-Jacobi-Bellman equation. We reduce that model to its essential, namely to the underlying Hamilton-Jacobi equation. For this PDE, we propose an efficient semi-implicit implementation. Numerical experiments show the usefulness of our approach: Besides reasonable computational times, the method is robust with respect to noise as well as to the choice of the numerical initial condition which is a delicate point for many SFS algorithms.

1 Introduction

The Shape-from-Shading (SFS) problem amounts to compute the 3-D shape of a surface from the brightness of exactly one given grey value image of that surface. It is a classical problem in computer vision, see e.g. [5, 6, 9, 21] and the references therein for an overview.

The modeling of the SFS problem via the use of a PDE was introduced by Horn [7, 8, 9], who also coined the name 'shape-from-shading'. The model of Horn is the basis of all later works in that field. As it is of importance in the context of our work, let us mention some relevant features of the model of Horn. On the modeling side, a distinguished ingredient is the use of an orthographic camera, i.e., the camera performs an *orthographic projection* of the scene of interest. Together with a point light source at infinity, the PDE

$$|\nabla u| - \sqrt{\frac{1}{I^2} - 1} = 0 \tag{1}$$

can be derived [1], where

- $u \equiv u(\mathbf{x})$ is the sought depth map,
- $|.|$ denotes the Euclidean vector norm,
- $I \equiv I(\mathbf{x}) = \frac{E(\mathbf{x})}{\sigma}$ is a normalised version of the image brightness, σ depends on the albedo of the surface and the intensity of the light source,
- $E \equiv E(\mathbf{x})$ is the brightness of the given grey-value image.

The *Eikonal equation* (1) constitutes the SFS-model widely studied in the literature. It is well-known that the corresponding problem is ill-posed, often shown via the so-called *convex-concave ambiguity*, see e.g. [4, 9] for related discussions.

In [2, 12, 13, 14, 15, 17, 18, 19, 20] a new PDE model for SFS is proposed. The setting of this new model is given by using a pinhole camera and a point light source at the optical center, thus incorporating a *perspective projection* instead of an orthographic one as in the classical case into the modeling process.

This perspective approach yields the *Hamilton-Jacobi equation*

$$\frac{I\mathsf{f}^2}{u}\sqrt{\frac{\mathsf{f}^2\left|\nabla u\right|^2 + \left(\nabla u \cdot \mathbf{x}\right)^2}{Q^2}} + u^2 = \frac{1}{u^2}\,, \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^2$ is in the image domain Ω as before, and

- f is the focal length relating the optical center of the camera and its retinal plane,
- $Q \equiv Q(\mathbf{x}) := \dfrac{\mathsf{f}}{\sqrt{|\mathbf{x}|^2 + \mathsf{f}^2}}$.

In order to obtain a viable numerical solver, Prados et al. use the dynamic programming principle. The resulting numerical solver incorporates the solution of an optimal control problem which can be quite intricate; for some details see [12]. In contrast, Tankus et al. rely on the level set method; see especially [20].

In a recent paper, Cristiani et al. [4] employed a semi-Lagrangian formulation of the above model to construct a fast SFS solver, however, as also shown in their paper, in contrast to our procedure their method is very sensitive to the choice of initial data.

Our contribution. We consider directly the PDE given in (2), and we show that it is possible to construct a *robust* and *efficent* numerical solver without the need to rely on dynamic programming, or to turn to the level set method. Another objective of us is that it is *easy to code* in comparison to other approaches in the field. As experiments show, the numerical routine we develop is also to a high degree *insensitive to perturbations* of initial data which can be a delicate point for SFS algorithms.

Organisation of this paper. In Section 2, we briefly review the modeling ingredients as well as some fundamental properties of the resulting PDE (2). In Section 3, we give a detailed description of our numerical scheme, focusing on its construction and the choice of the time step size. This discussion is followed by numerical experiments in Section 4. The paper is finished by concluding remarks.

2 Description of the model

In this paragraph, we briefly review the modeling process of (2), thereby illuminating the roles of its ingredients. For a more detailed description, see e.g. [15]. Figure 1 is adopted from there.

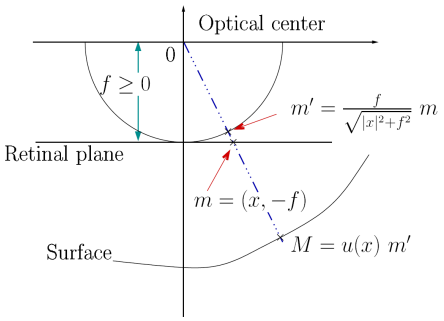


Figure 1: Perspective projection with a point light source located at the optical center.

Let Ω represent the rectangular image domain in $\mathbb{R}^2$. Consider then the surface $\mathcal{S}$, representing the object or scene of interest, parametrised by using the function $S : \bar{\Omega} \to R^3$ with

$$S(\mathbf{x}) = \frac{\mathsf{f} u(\mathbf{x})}{\sqrt{|\mathbf{x}|^2 + \mathsf{f}^2}} \, (\mathbf{x}, -\mathsf{f})^T \, . \tag{3}$$

As the two columns of the Jacobian $\mathcal{J}\left[S(\mathbf{x})\right]$ are tangent vectors to $\mathcal{S}$ at the point $S(\mathbf{x})$, their cross-product is a normal vector to $\mathcal{S}$. Thus, a normal vector $\mathbf{n}(\mathbf{x})$ at the point $S(\mathbf{x})$ is given by

$$\begin{aligned} \mathbf{n}(\mathbf{x}) \quad = \quad & \Bigg(\mathsf{f} \nabla u(\mathbf{x}) - \frac{\mathsf{f} u(\mathbf{x})}{|\mathbf{x}|^2 + \mathsf{f}^2} \mathbf{x} \, , \\ & \nabla u(\mathbf{x}) \cdot \mathbf{x} + \frac{\mathsf{f} u(\mathbf{x})}{|\mathbf{x}|^2 + \mathsf{f}^2} \mathsf{f} \Bigg)^T \, . \end{aligned} \tag{4}$$

Assuming that the surface is Lambertian, the brightness equation is

$$I(\mathbf{x}) = \frac{\cos\theta}{r^2} \, . \tag{5}$$

Thereby, θ is the angle between the surface normal vector and the (unit) light source direction $\mathbf{L}$,

$$\mathbf{L}\left(S(\mathbf{x})\right) = \frac{1}{\sqrt{|\mathbf{x}|^2 + \mathsf{f}^2}} \, (-\mathbf{x}, \mathsf{f})^T \, , \tag{6}$$

and r is the distance of the corresponding surface point to the light source. Employing the standard formula for cos in (5),

$$\cos\theta = \mathbf{L}\left(S(\mathbf{x})\right) \cdot \frac{\mathbf{n}(\mathbf{x})}{|\mathbf{n}(\mathbf{x})|} \, , \tag{7}$$

and evaluating the scalar product in (7), one obtains the sought formula (2).

For convenience, we assume for our numerical implementation, that the surface $\mathcal{S}$ is visible, i.e., it is in the front of the optical center, so that u is strictly positive. Then we use the change of variables $v = \ln(u)$, yielding the PDE

$$\frac{I \mathsf{f}^2}{Q} \sqrt{\mathsf{f}^2 \left|\nabla v\right|^2 + \left(\nabla v \cdot \mathbf{x}\right)^2 + Q^2} = e^{-2v} \, . \tag{8}$$

This equation is the basis of our numerical scheme.

Remarks. (i) As discussed in [12], the corresponding model was already considered in non-PDE-form in [10, 11]. (ii) The benefit of the above formulation is that the model is well-posed; for details see [12]. (iii) With different parametrizations of the surface $\mathcal{S}$, one arrives at different, yet to (8) equivalent PDEs.

The PDE (8) needs to be supported by boundary conditions, i.e., $v(\mathbf{x}) := \varphi(\mathbf{x})$ for $\mathbf{x} \in \partial\Omega$. In this setting, one can prove uniqueness of viscosity sub- and supersolutions, see especially [12].

We will be interested in computing a viscosity supersolution: this approach avoids the convex/concave ambiguity often encountered in SFS models. The corresponding theoretical setting can be described via $\varphi \equiv +\infty$; see the discussion in [12]. The consequence is, that the boundary condition becomes virtually unimportant, and we could implement it like Dirichlet boundary conditions given by a 'large' constant. In practice, we use Neumann boundary conditions in order to avoid the problem to set the latter, which works well. However, let us stress, that this particular consequence arises by an *Eulerian formulation* of the problem as given by (8). It does not hold, e.g., in the case of the recent semi-Lagrangian approach of Cristiani et al. [4]. We test the use of both types of boundary conditions in the section on numerical experiments.

3 Our numerical method

There are two main approaches in dealing with the PDE of interest (8):

1. Treat it like a usual boundary value problem and solve it directly.
2. Employ an additional 'time' variable and iterate until a steady state is reached.

We will follow the second path: This guarantees that we obtain an approximation of the viscosity supersolution. The logic is to choose an initial state above the supersolution and converge to this closest solution of (8) by iterating in 'time'. Thus, the PDE to discretise is

$$v_t = \underbrace{-\frac{I\mathsf{f}^2}{Q}\sqrt{\mathsf{f}^2\left|\nabla_{\mathbf{x}} v\right|^2 + \left(\nabla_{\mathbf{x}} v \cdot \mathbf{x}\right)^2 + Q^2}}_{=:A} + e^{-2v} \tag{9}$$

where $v \equiv v(\mathbf{x}, t)$ now, and where A is a useful abbreviation for later use.

3.1 Scheme construction

We employ the standard notation $v_{i,j}^n := v\left(ih_1, jh_2, n\tau\right)$, where h_1 and h_2 are the mesh widths and i and j the coordinates of the pixel (i, j) in $\mathbf{x}_1$- and $\mathbf{x}_2$-direction, respectively, and where τ is a time step size yet to be determined. The discretisation of $v_t(\mathbf{x}, t)$ we use is given by the *Euler forward* formula

$$v_t(\mathbf{x}, t)|_{(\mathbf{x},t)=(ih_1,jh_2,n\tau)} \approx \frac{v_{i,j}^{n+1} - v_{i,j}^n}{\tau} . \tag{10}$$

A *stable discretisation* of the spatial derivatives in $\nabla_{\mathbf{x}} v$ is given by the *upwinding* concept: As it is well-known for the kind of PDEs like (9), the use of central differences leads to artificial oscillations resulting in a blow-up of the numerical solution [16].

The upwinding concept boils down to use one-sided finite differences in the appropriate direction. These are determined by following the characteristics of the solution, thus realising wave-propagation in the physically correct direction. Following the derivation of Rouy and Tourin [16], one obtains

$$v_{\mathbf{x}_1}(\mathbf{x}, t)|_{(\mathbf{x},t)=(ih_1,jh_2,t)} \tag{11}$$
$$\approx \max\left(0, \frac{v_{i+1,j} - v_{i,j}}{h_1}, \frac{v_{i-1,j} - v_{i,j}}{h_1}\right),$$
$$v_{\mathbf{x}_2}(\mathbf{x}, t)|_{(\mathbf{x},t)=(ih_1,jh_2,t)} \tag{12}$$
$$\approx \max\left(0, \frac{v_{i,j+1} - v_{i,j}}{h_2}, \frac{v_{i,j-1} - v_{i,j}}{h_2}\right).$$

Note, that in (11)-(12), we have not specified the time level yet.

The reason for the latter is due to computational efficiency we would like to achieve. For this, we employ a *Gauß-Seidel-type idea* which works as follows. Let us introduce a linear numeration of pixels, i.e., we store the unknowns in a vector whose length is the total number of pixels. We then set the computational nodes in such a way, that we proceed according to the ordering in Figure 2.

1	2	3	$\ldots$
$n_x + 1$	$n_x + 2$	$n_x + 3$	$\ldots$
$2n_x + 1$	$2n_x + 2$	$2n_x + 3$	$\ldots$
$3n_x + 1$	$3n_x + 2$	$3n_x + 3$	$\ldots$
$\ldots$	$\ldots$	$\ldots$	$\ldots$

Figure 2: Pixel ordering for the Gauß-Seidel-type method.

Having a close look at formulae (11)-(12), we notice that the stencil of the method incorporates the data

$$\begin{array}{ccc} & v_{i,j+1} & \\ v_{i-1,j} & v_{i,j} & v_{i+1,j} \\ & v_{i,j-1} & \end{array} . \tag{13}$$

This means, at a pixel (i,j) and iterating through the pixel list as in Figure 2, we have already computed $v_{i,j+1}^{n+1}$ and $v_{i-1,j}^{n+1}$. Thus, for the computation of $v_{i,j}^{n+1}$, we can use these already updated values to achieve an accelerated convergence of our scheme.

To summarise, approximating spatial derivatives at time $t = n\tau$, the time levels within formulae (11)-(12) are set by us as

$$\max\left(0,\ \frac{v_{i+1,j}^{n}-v_{i,j}^{n}}{h_1},\ \frac{v_{i-1,j}^{n+1}-v_{i,j}^{n}}{h_1}\right),$$
$$\max\left(0,\ \frac{v_{i,j+1}^{n+1}-v_{i,j}^{n}}{h_2},\ \frac{v_{i,j-1}^{n}-v_{i,j}^{n}}{h_2}\right), \tag{14}$$

respectively. For clarity, let us stress once more, that the values from time level $(n+1)\tau$ in (14) are already *fixed* for the computation of $v_{i,j}^{n+1}$.

Turning to the discretisation of $I(\mathbf{x})$, $Q(\mathbf{x})$ and $\mathbf{x}$ in (9), we see that this issue amounts pixelwise to simple explicit term evaluations, so that there is no problem to deal with these terms.

We now turn, finally, to the source term e^{-2v} in (9). Source terms like this typically result in a very small time step size when evaluated *explicitly*, i.e., if at time level $t = n\tau$ we approximate it via $\exp\left(v_{i,j}^{n}\right)$. Especially, this results in iterates changing very slowly, leading to excessive computational times in reaching steady state solutions. Thus, we consider an *implicit* discretisation of it, writing

$$e^{-2v(\mathbf{x},t)}|_{(x,t)=(i,j,n\tau)} \approx e^{-2v_{i,j}^{n+1}}. \tag{15}$$

This component of our algorithm makes it necessary to employ in *each point* an iterative solver of the arising nonlinear equation: denoting by $\hat{A}$ the discretised version of term A from (9), we obtain by the Euler forward formula (10) pixelwise the update formula

$$v_{i,j}^{n+1} = v_{i,j}^{n} - \tau\hat{A} + \tau e^{-2v_{i,j}^{n+1}} \tag{16}$$

which has to be solved for $v_{i,j}^{n+1}$. We do this by employing the classical Newton-method, letting it iterate until convergence (which requires in practice three or four iterations). In this context, let us stress explicitly, that by (16) it does not become necessary to solve a nonlinear system of equations: the task amounts to solve *pixelwise* a quite harmless *one-dimensional* nonlinear equation, done efficently by the one-dimensional Newton-method.

To summarise, we propose a relatively simple-to-implement, semi-implicit method, where the source term is evaluated implicitly, and where already computed values are taken into account wherever possible accelerating convergence.

3.2 Choosing the time step size

We now discuss the most critical number that needs to be specified, i.e., the time step size τ. It is well-known, that implicitly discretised terms do not incorporate a restriction on the time step size. In fact, for a completely implicit scheme, where all data in (14) *and* (15) were from time level $(n+1)\tau$ there would theoretically not be a restriction on the allowed time step size at all. However, a completely implicit formulation results in a quite complicated nonlinear system of equations to solve numerically, which is contrary to the philosophy followed here to propose an efficient easy-to-code scheme. In our method, we discretise only the source term implicitly. Thus, the term in (15) does not impose a stability restriction, whereas the contribution due to $\hat{A}$ in (16) should imply a stability bound.

In practice, estimates for an upper bound on the time step size are often too restrictive for direct use if the underlying problem involves many nonlinearities. An automatic choice based on such an upper bound may thus result in unnecessarily long computational times. Nevertheless, it yields a good starting point for a user's choice. We now proceed in the line of these considerations, computing a reasonable candidate for an initial choice of the time step size. As indicated, we neglect for this the implicitly discretised source term.

A meaningful stability criterion for numerical methods for PDEs of the considered type is a *discrete maximum-minimum-principle*, i.e., ideally, the numerical solution of each time step shall not produce oscillations by over- or undershooting neighbouring data. This discrete stability criterion is closely related to the notion of viscosity solutions discussed in paragraph 2, as such viscosity solutions enforce the corresponding property on the level of the PDE-formulation [3].

Let us stress here, that it makes sense to establish a discrete minimum-maximum-principle neglecting in the corresponding computations the source term: This has the character of establishing a *necessary condition* for stability.

Let us have a close look at the terms of importance in the corresponding 'reduced form' of (16):

$$v_{i,j}^{n+1} = v_{i,j}^n - \tau \hat{A}\,. \tag{17}$$

Then the task arises to estimate

$$\begin{aligned}\left|\tau\hat{A}\right| \\ \overset{!}{\leq} \quad & \max\left(\frac{\left|v_{i+1,j}^n - v_{i,j}^n\right|}{h_1}, \frac{\left|v_{i-1,j}^{n+1} - v_{i,j}^n\right|}{h_1},\right. \\ & \left.\frac{\left|v_{i,j+1}^{n+1} - v_{i,j}^n\right|}{h_2}, \frac{\left|v_{i,j-1}^n - v_{i,j}^n\right|}{h_2}\right)\,. \end{aligned} \tag{18}$$

Let us note that, in (18), the values $v_{i-1,j}^{n+1}$, $v_{i,j+1}^{n+1}$ are already fixed so that it makes sense to incorporate these data from time level $(n+1)\tau$ into the computation.

For clarity, let us point out explicitly, that a discrete maximum-minimum-principle holds if (18) is satisfied: the right hand side amounts to the maximal absolute difference between $v_{i,j}^n$ and the other given data within the computational stencil. Thus, if (18) is met, the largest possible change due to an update yields the maximum or minimum of this set, respectively.

For abbreviation, let us now denote the quantity on the right hand side of (18) by δv. Employing the notation $\nabla\hat{v}$ for the discretisation of ∇v introduced in (11)-(12) and (14), we can compute the following estimates:

$$|\nabla\hat{v}|^2 \quad \leq \quad \left(\sqrt{2\delta v^2}\right)^2 \;=\; 2\delta v^2\,, \tag{19}$$

$$(\nabla\hat{v}\cdot\mathbf{x})^2 \quad \leq \quad (2\hat{\mathbf{x}}\delta v)^2 \;=\; 4\hat{\mathbf{x}}^2\delta v^2\,, \tag{20}$$

where in (20) we have used

$$\hat{\mathbf{x}} := \max_{i,j}\left(|i|\,h_1,\,|j|\,h_2\right)\,. \tag{21}$$

Note, that $\hat{\mathbf{x}}$ is a finite number but it can be quite large.

Plugging (19)-(20) into $\hat{A}$ yields

$$\begin{aligned}\hat{A} \quad &\leq \quad \frac{I\mathsf{f}^2}{Q}\sqrt{\mathsf{f}^2 2\delta v^2 + 4\delta v^2\hat{\mathbf{x}}^2 + Q^2} \\ &\leq \quad I\mathsf{f}\sqrt{\mathsf{f}^2+\hat{\mathbf{x}}^2}\sqrt{2\mathsf{f}^2\delta v^2 + 4\hat{\mathbf{x}}^2\delta v^2 + Q^2}\,. \end{aligned} \tag{22}$$

Up to (22), all steps involve rigorous estimates. As we only seek an estimate for choosing τ here, we may now employ the following simplification. As Q is a number in $(0,1)$ generally small in comparison with the other arising terms – which also comprise a 'pessimistic' estimation – we may neglect Q^2, arriving at

$$\max\hat{A} \approx \delta v I\mathsf{f}\sqrt{\mathsf{f}^2+\hat{\mathbf{x}}^2}\sqrt{2\mathsf{f}^2+4\hat{\mathbf{x}}^2}\,. \tag{23}$$

From (18) and (23), as $\left|\tau\hat{A}\right| \leq \delta v$ shall hold, we obtain after a few trivial manipulations the inequality

$$\tau < \frac{1}{2I\mathsf{f}\left(\mathsf{f}^2+\hat{\mathbf{x}}^2\right)}\,. \tag{24}$$

In SFS computations, the number on the right hand side of (24) is often very small, in a typical setting of our experiments around 10^{-5} to 10^{-7}. As indicated, this number can be relaxed by some factor as the estimation is pessimistic. Let us also note, while the discrete maximum-minimum-principle is enforced, on the other hand the theoretical maximum of local updates is allowed. This means, the absolute size of the number does not matter as much as it seems.

4 Numerical Experiments

In this paragraph, we show several numerical experiments on synthetic images in order to assess the performance of our algorithm.

The pyramid experiment. This experiment is very useful for investigating and visualising the influence of initial and boundary conditions. The task is to reconstruct the pyramid-shaped surface shown in Figure 3. Figure 4 shows a photograph of this surface with $\sigma = 1000$, $\mathsf{f} = 251.6$, $h_1 = h_2 = 1$, 256×256 pixels, where σ denotes the factor determined by light source intensity and surface albedo, and h_1, h_2 are as before the pixel widths in $\mathbf{x}_1$- and $\mathbf{x}_2$-direction, respectively. The rendering was done by ray-tracing the surface.

Note, as the surface consists of four triangles, only these triangles can be hit in the ray-tracing process. Consequently, the surface normal at the top of the pyramid does not point towards the camera, resulting in a maximum grey value of 228 instead of 255. Hence, we cannot expect the sharp top of the pyramid to be reconstructed perfectly.

As noted in the introduction, at the image boundary we employ Neumann boundary conditions. The algorithm was initialised with the constant $v \equiv \log 0.2$, which is larger than the actual solution.

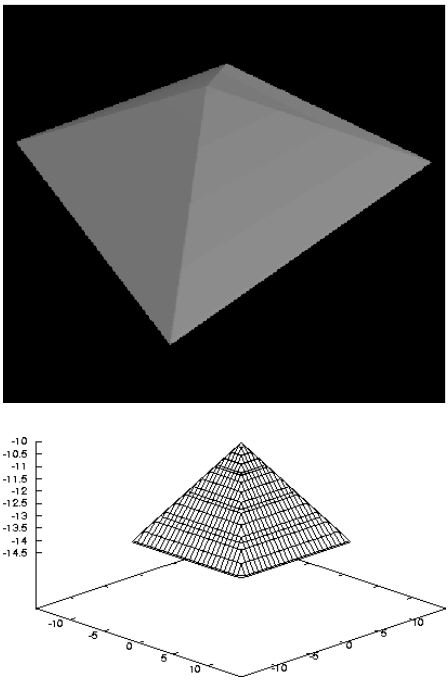


Figure 3: A pyramid shaped surface.

Let us point out here explicitly, that a constant v is equivalent to a spherical surface in the untransformed variable u. Figure 5 shows the surface corresponding to this constant initialisation.

Figure 6 shows a reconstruction of the surface using our method. As expected, the top of the pyramid is flat instead of sharp. Due to the boundary conditions, the reconstruction at the image boundary is a bit too round compared to the ground truth. However, overall the reconstruction of the pyramid is good, even the edges of the pyramid are reconstructed well.

As indicated before, our method does not rely on a specific initialisation. In the previous experiment, we initialised the algorithm with a constant v larger than the actual solution. We can choose any such constant, the reconstruction is always the same.

In fact, the method does not even need constant initialisation. Figure 7 shows an alternative initialisation for u with random data in the range $(0.18, 0.22)$. The result of the reconstruction is the same as shown in Figure 6.

Note that the speed of the reconstruction depends

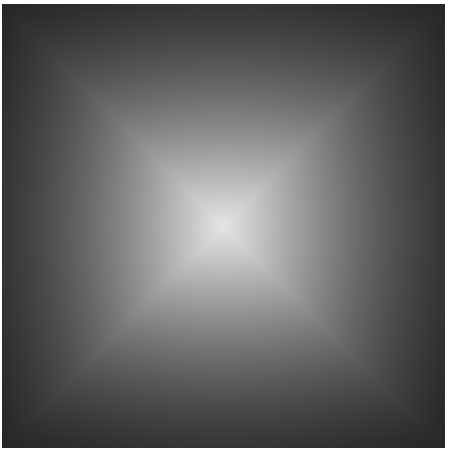


Figure 4: Input image for the pyramid surface.

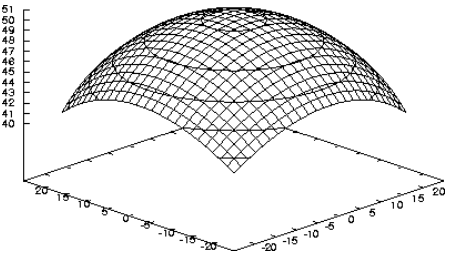


Figure 5: Surface with constant v.

on how far the initialisation is away from the actual solution and on the time step size. For ensuring stability, we need to use rather small time step sizes, see Section 3.2, so we should try to find an initialisation larger than, but as close as possible to the solution. In [12], it is shown that

$$v(\mathbf{x}) = -0.5 \log I f^2 \tag{25}$$

fulfils this. This initial image, however, is only defined if I is strictly positive everywhere, i.e., if there are no black pixels in the input image. With this initialisation, a good reconstruction of the pyramid image can be obtained in about one minute (C-code, Pentium 4, 3.2 GHz). Figure 8 shows the surface corresponding to this initialisation for the pyramid input image.

The discussed experiments were all done without any knowledge of the actual surface used within the

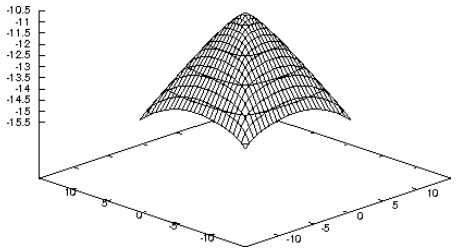


Figure 6: Reconstruction of the pyramid with Neumann boundary conditions.

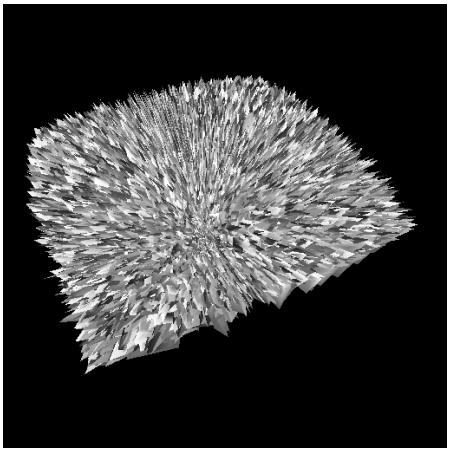


Figure 7: Surface with random v.

algorithm. If we use exact Dirichlet boundary conditions, i.e., if we set the values at the image boundary to those of the ground truth, we obtain a nearly perfect reconstruction, which is shown in Figure 9.

Prados et al. [12] suggest to use state constraints boundary conditions, i.e., Dirichlet boundary conditions with a very large constant at the boundary. Since this means that very large gradients are generated at the boundary, it can easily be seen by taking into account (14), that state constraints boundary conditions are practically the same as Neumann boundary conditions (this is also true for the algorithm of Prados et al.).

We implemented the algorithm from [12]. Figure 10 shows a reconstruction obtained by using this algorithm. The result seems to be about as good as the one computed by our method.

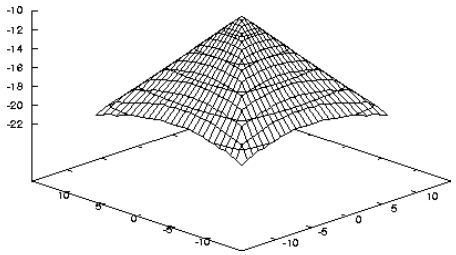


Figure 8: Initialisation with $v = -0.5 \log If^2$ for the pyramid image.

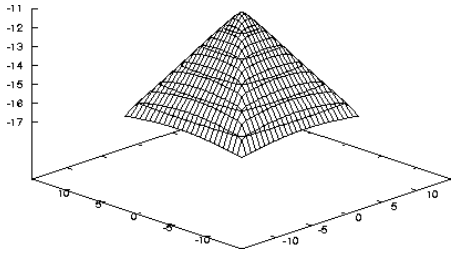


Figure 9: Reconstruction of the pyramid with exact Dirichlet boundary conditions.

Our algorithm is also robust with respect to noisy input data. In order to verify this claim, we have added Gaussian noise with standard deviation 10 to the input image. Figure 11 shows the corresponding reconstruction using our method. The result is nearly as good as without noise. The same holds true for the scheme of Prados et al.. Using the latter method, it is, however, necessary to reduce the time step size a bit to ensure stability of the method. The estimate given in [12] works very well for "continuous" input data, but for "discontinous" input data this estimate tends to be a bit too large, especially near very dark pixels, which may affect the stability of the scheme. Our method is perfectly stable using the estimate from equation (24).

Quantitative comparison with the scheme of Prados et al.. We compare our results to those of the algorithm of Prados et al. [12]. We compare both reconstruction quality and run time. For a quantitative comparison of both algorithms, we compare

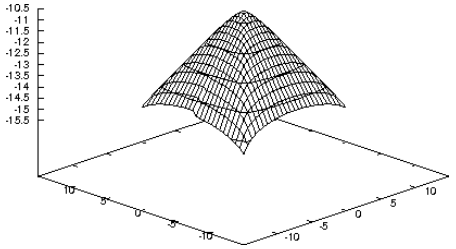


Figure 10: Reconstruction of the pyramid with the algorithm of Prados et al.

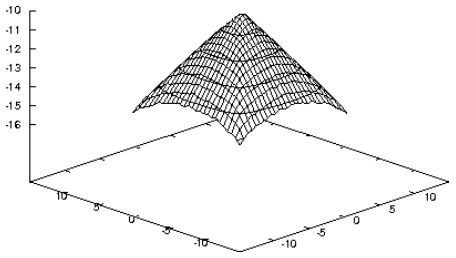


Figure 11: Reconstruction of the pyramid with Gaussian noise added.

the average L_1-error

$$\frac{1}{n_x n_y} \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} |u(i,j) - \hat{u}(i,j)|\,, \qquad (26)$$

where $n_x \times n_y$ is the image size, u the reconstructed depth and $\hat{u}$ the ground truth.

Table 1: L_1 errors of u for the pyramid experiment.

Noise	Our method	Prados et al.
Without	0.0069	0.0070
Gaussian, $\sigma = 5$	0.0071	0.0072
Gaussian, $\sigma = 10$	0.0076	0.0074

Table 1 shows the errors of both algorithms for the pyramid image with and without noise. Both algorithms are quite robust under noise, the results only get slightly worse. The quality of the result is about equal. As a stopping criterion we employed here a maximum change of a pixel value (in v) of less than 10^{-6}, which is fairly low, good results may as well be achieved with a larger value in less computation time.

Table 2: Run times for the pyramid experiment.

Method	Time
Prados et al.	322s
Our method, global τ	102s
Our method, local τ	41s

In Table 2, we compare the run times of both algorithms for the pyramid input image. Using a constant time step size according to the estimate in equation (24), we achieve a significantly better performance than Prados et al.. However, note that our implementation of the scheme in [12] is not optimal in the sense that it makes use of several (expensive) functions like atan, sin, cos, log, and exp in every iteration (while our algorithm only uses exp in the Newton step). One might speed it up, e.g. by using lookup tables for those functions. Still, our algorithm will be faster, and can be accellerated even more by evaluating the estimate from equation 24 in every pixel instead of using a constant time step size globally. For the pyramid image, we notice a speedup of roughly a factor 2.5 this way. Prados et al. make also use of a local choice of the time step size. All the run times in Table 2 are measured using a C-implementation of both methods on a Pentium 4, 3.2 GHz, 2 GB RAM running Linux, and both algorithms have been initialised like in equation (25).

Summary of the pyramid experiment. We have shown that our method works independently of the particular choice of initial and boundary values. In the simple setting of the pyramid experiment, the scheme yields good results. The new algorithm has proven robust to Gaussian noise added to the input image. Compared to the method of Prados et al., we observe a significant speedup with a comparable reconstruction quality.

The Mozart experiment. The Mozart experiment is a well-known benchmark in the SFS area. The ground truth is depicted in Figure 12.

As input image, we use the input image used in [12, 15], which is shown in Figure 13. Parameters for the reconstruction are $\mathsf{f} = 250$, $h_1 = h_2 = 1$, 256×256 pixels. Unfortunately, [12] does not give

the value for σ, we just assume $\sigma = 4 \cdot 10^4$, which should be in a realistic range.

As a particular difficulty, the input data involve a slight 'break of the rules', as it does not satisfy the underlying modeling assumption that the face is completely visible, which is obvious by the shadows on its left and right hand side. These shadows are indeed black pixels in the image, hence $I(\mathbf{x})$ vanishes at these pixels. However, $I = 0$ also means our method will not move towards the solution, but remain at the initialisation values. To overcome this, we change all pixels in the input image with grey value smaller than 5 to a grey value equal to 5, this way, those pixels will also have the ability to move away from the initialisation. Nevertheless, the reconstruction of the Mozart image at these pixels is very difficult.

Figure 14 shows a reconstruction of the Mozart image using our method (with constant initialisation) and Neumann boundary conditions. The face and the shoulders of Mozart are reconstructed very well. The reconstruction is somewhat flat, but this is the case for nearly every reconstruction of the Mozart face. The (too high) reconstruction of the background makes the reconstruction appear even more flat.

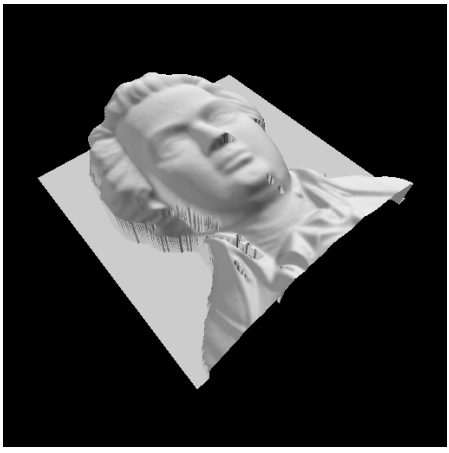


Figure 12: Ground truth for the Mozart image.

At the boundary of the face, the reconstruction proves to be very difficult, as we expected: Some (dark) pixels are far off the ground truth. The background is not recovered very well, this is caused by the boundary conditions (a sphere-like shape is assumed at the image boundary) and by the difficulties with the reconstruction of the face boundary.

Figure 13: Mozart input image.

The reconstruction using the method of Prados et al. is again comparable to the one using our scheme, so we do not give an additional corresponding figure here. Since there are dark pixels in the input image, we need to use a smaller time step size once again to ensure convergence of the method of Prados et al..

Concluding the Mozart experiment. Comparing our result with those in [12] which can be considered as the state-of-the-art, the reconstruction quality of the face is similar. At the face boundary our method performs a bit better while we have a much smaller amount of outliers, yet the reconstruction of the background is comparable. Altogether, our method seems to do very well here, considering we had to guess σ.

5 Concluding remarks

To conclude, we have shown that the constructed numerical methods satisfies the intended goals:

- robustness of the scheme with respect to the choice of initial data,
- moreover, robustness with respect to the implementation of boundary conditions,
- robustness with respect to noisy data,
- numerical results are of the accuracy of state-of-the-art methods in the field, yet the numeri-

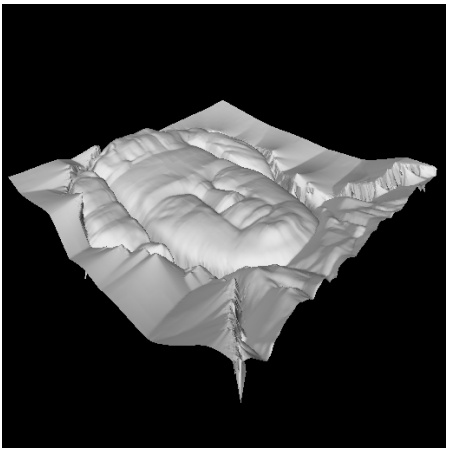


Figure 14: Reconstruction of the Mozart surface.

cal solver is much simpler,

- the implementation of the numerical scheme is of modest programming effort,
- computational times are reasonable.

Taking these aspects altogether, we have devised a highly competitive method in the field, as well as a good basis for further developments.

The work of us in the near future will be in two areas: (i) Further improvement and analysis of the numerical scheme, especially with respect to the choice of the time stepping method, and (ii) the investigation of real-world applications of the model in conjunction with the numerical scheme.

References

[1] A. R. Bruss. The eikonal equation: Some results applicable to computer vision. *Journal of Mathematical Physics*, 23(5):890–896, May 1982.

[2] F. Courteille, A. Crouzil, J.-D. Durou, and P. Gurdjos. Towards shape from shading under realistic photographic conditions. In *Proc. 2004 International Conference on Pattern Recognition*, volume 2, pages 277–280, Cambridge, MA, USA, August 2004.

[3] M. G. Crandall, H. Ishii, and P.-L. Lions. User's guide to viscosity solutions of second order partial differential equations. *Bulletin of the American Mathematical Society*, 27(1):1–67, 1992.

[4] E. Cristiani, M. Falcone, and A. Seghini. Some remarks on perspective shape-from-shading models. In F. Sgallari, A. Murli, and N. Paragios, editors, *Scale Space and Variational Methods in Computer Vision*, volume 4485 of *Lecture Notes in Computer Science*, pages 276–287, Berlin, May-June 2007. Springer.

[5] R. T. Frankot and R. Chellappa. A method for enforcing integrability in shape from shading algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(4):439–451, July 1988.

[6] B. K. P. Horn. *Shape from Shading: A Method for Obtaining the Shape of a Smooth Opaque Object from One View.* PhD thesis, Department of Electrical Engineering, MIT, Cambridge, MA, 1970.

[7] B. K. P. Horn. Understanding image intensities. *Artificial Intelligence*, 8:201–231, 1977.

[8] B. K. P. Horn and M. J. Brooks. Shape and source from shading. In *Proc. International Joint Conference on Artificial Intelligence*, pages 932–936, Los Angeles, CA, USA, August 1985.

[9] B. K. P. Horn and M. J. Brooks. *Shape from Shading*. Artificial Intelligence Series. MIT Press, Cambridge, MA, USA, 1989.

[10] T. Okatani and K. Deguchi. Reconstructing shape from shading with a point light source at the projection center: Shape reconstruction from an endoscope image. In *Proc. 1996 International Conference on Pattern Recognition*, volume 1, pages 830–834, Vienna, Austria, August 1996.

[11] T. Okatani and K. Deguchi. Shape reconstruction from an endoscope image by shape from shading technique for a point light source at the projection center. *Computer Vision and Image Understanding*, 66(2):119–131, May 1997.

[12] E. Prados. *Application of the theory of the viscosity solutions to the Shape from Shading problem*. PhD thesis, University of Nice Sophia-Antipolis, Oct. 2004.

[13] E. Prados and O. Faugeras. Perspective shape from shading and viscosity solutions. In *Proc Ninth International Conference on Computer Vision*, volume 2, pages 826–831, Nice, France, October 2003. IEEE Computer Society Press.

[14] E. Prados and O. Faugeras. A generic and provably convergent shape-from-shading method for orthographic and pinhole cameras. *International Journal of Computer Vision*, 65(1-2):97–125, 2005.

[15] E. Prados and O. Faugeras. Shape from shading: A well-posed problem? In *Proc. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 870–877, San Diego, CA, June 2005. IEEE Computer Society Press.

[16] E. Rouy and A. Tourin. A viscosity solutions approach to shape-from-shading. *SIAM Journal of Numerical Analysis*, 29(3):867–884, 1992.

[17] A. Tankus, N. Sochen, and Y. Yeshurun. A new perspective [on] shape-from-shading. In *Proc. Ninth International Conference on Computer Vision*, volume 2, pages 862–869, Nice, France, Oct. 2003. IEEE Computer Society Press.

[18] A. Tankus, N. Sochen, and Y. Yeshurun. Perspective shape-from-shading by fast marching. In *Proc. 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 43–49, Washington DC, USA, June-July 2004. IEEE Computer Society Press.

[19] A. Tankus, N. Sochen, and Y. Yeshurun. Reconstruction of medical images by perspective shape-from-shading. In *Proc. 2004 International Conference on Pattern Recognition*, volume 3, pages 778–781, Cambridge, UK, Aug. 2004.

[20] A. Tankus, N. Sochen, and Y. Yeshurun. Shape-from-shading under perspective projection. *International Journal of Computer Vision*, 63(1):21–43, June 2005.

[21] R. Zhang, P.-S. Tsai, J. E. Cryer, and M. Shah. Shape from shading: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):690–706, 1999.

An iterative framework for registration with reconstruction

Thales Vieira[1,2,3], Adelailson Peixoto[2], Luiz Velho[3], Thomas Lewiner[1]

[1] Department of Mathematics, PUC—Rio de Janeiro
[2] Department of Mathematics, UFAL—Maceió
[3] Visgraf Lab, IMPA—Rio de Janeiro

Abstract

The core of most registration algorithms aligns scan data by pairs, minimizing their relative distance. This local optimization must generally pass through a validation procedure to ensure the global coherence of the resulting alignments. This work introduces an iterative framework to guarantee the global coherence of the registration process. The iteration alternates registration and reconstruction steps, including alignments with the proper reconstructed surface, until the alignment of all the scans converges. The framework adapts to different contexts by choosing which scans are aligned and which are used for the reconstruction. This choice is based on the alignment and reconstruction errors. Derivations of this framework are presented with a rough automatic registration, increasing its robustness.

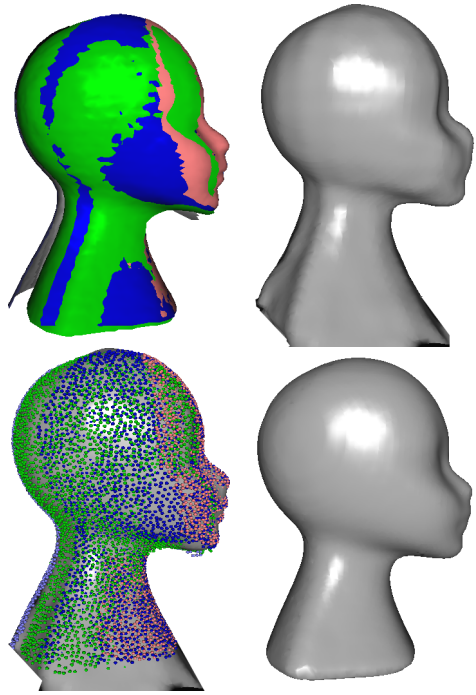
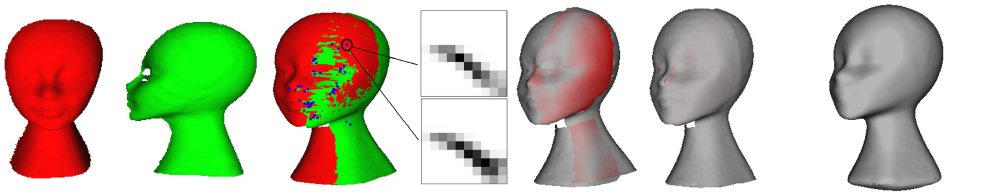


Figure 1: Correcting alignment with the reconstruction: The rough, pair-wise positioning accumulates errors (top left), which leads to false features at scan boundaries in the reconstruction (top right). Realigning with this reconstruction ensures the global coherence (bottom left), significantly improving the final reconstruction (bottom right).

1 Introduction

Three-dimensional scanning builds virtual models from several views of the same real object. Each view or *scan* generates a range image, i.e. a set of points in 3D with its own coordinate system. The registration process defines an optimal common coordinate system for all the scans, which is a necessary pre-processing of shape reconstruction algorithms. This optimal alignment is usually determined by minimizing over all the rigid transformations a distance between the overlapping parts of the scans. The basic optimization algorithm is the Iterative Closest Point (ICP) [1, 2], which aligns each pair of scans separately. Since the scans are views of a unique rigid object, a global optimal alignment must exist. However, the scanning process is prone to noise [15], and the local minima must then be checked for global consistency.

In this work, we propose to use and schedule intermediate reconstructed models to improve the registration process, extending the early work of Jin *et al.* [24]. This reconstruction step provides a feedback on the current alignment quality and on how to correct it for the final reconstruction: We align a selected subset of the scans with the reconstruction, generating a new alignment that is used for a subsequent reconstruction. The reconstruction generates



(a) A pair of scans. (b) Spin-Images of corresponding points on the two scans. (c) Fine alignment with ICP. (d) Reconstructed model.

Figure 2: The basic elements of a reconstruction pipeline: from the several scan data in their own coordinate systems (a), a local geometry descriptor is used to derive a rough position (b). This alignment is refined by distance minimization (c). The registered scans are finally merged into a single surface (d). The color in (c) codes the maximal distance between two scans in the view direction.

an updated model to align with, repeating the process until this virtuous loop converges (Figure 1). The alternation of alignment and reconstruction on different selection of scans adapts the general registration process to different contexts, such as dynamic or multi-resolution registration. In particular, it can improve the registration precision (basic framework), achieve delicate registration (overlapping maximization), improve the registration robustness (divergence correction), decrease the total execution time (multi-resolution) or further integrate the whole scanning/reconstruction process (dynamic registration).

Related works. Usual automatic registration involves two steps: initial positioning and refinements of this alignment with global coherence validation.

The registration process searches for rigid transformations, which are low-dimensional solutions (6 dimensions per scan), from high dimensional data (3 dimensions per scan point). Using global optimization [17, 21] or statistical analysis [19, 18], one can directly align all the scans simultaneously. The problem can also be considered partially by aligning the scans pair-wise. To avoid working with all the points of a scan at once, several techniques use local descriptors that are invariant under rigid motions. In this work, we use spin images [11] as a shape descriptor, similarly to previous works [22, 4]. Robust descriptors can also be combined with the point selection [5]. Further references on this initial positioning can be found in [3].

This positioning is generally refined by local minimization algorithms, such as the classical Iterated Closest Point (ICP) algorithm [1, 2]. This algorithm has been improved in speed [23], accuracy and robustness [12, 9, 6]. Further references on the ICP variations can be found in [14].

In order to avoid local minima of the ICP, the current alignment is checked for consistency. This consistency usually comes either from a global optimization [21] or from a dependency graph of the pair-wise alignments [17, 25, 22]. In this work, we propose to use and schedule reconstructions steps in the registration pipeline to guarantee this global consistency.

The introduction of reconstruction in the registration process was first described in [24], which used the pioneering reconstruction method of [8]. They reconstruct the surface from all the aligned scans during the registration, and align all the scans with the reconstructed surface. Further constraints can be added by aligning the scans with a pre-defined model [10]. We propose here a generalization of these ideas by scheduling which scan is aligned or reconstructed at each step. Among the many surface reconstruction methods, we test our framework with two of them: the Multiple Partition of Unity implicits of [20], and the Poisson inversion of [13].

Contributions. This work introduces a general framework for the use of reconstruction inside the registration pipeline. This reconstruction provides a global feedback on the quality of the alignment. Since the reconstructed surface is generally a mesh which approximates the scan, the original scans can be aligned with it using robust algorithms such as ICP derivatives. This leads to a globally coherent registration without global optimization or consistency graph. Moreover, the final registration is automatically optimized for the reconstruction, avoiding false feature at misaligned regions [15].

Figure 3: Improvement of the global alignment using the reconstructed surface: misaligned scan data (left) is realigned with the reconstruction to improve the final alignment (right).

We incorporate this reconstruction step to a framework built on three basic steps: 1) initial positioning, 2) local refinement of an alignment and 3) reconstruction (Section 2). By scheduling these steps and choosing on which scans they are applied, we open this registration framework to different contexts such as dynamic or multi-resolution registration (Section 3). We implement this framework with simple algorithms for each step (Section 4), significantly improving the global robustness of the registration (Section 5).

2 Framework Elementary Steps

In this section, we recall basic examples of scan descriptors for the initial positioning, alignment refinement and surface reconstruction (Figure 2). These elements are representative of the three elementary steps of our registration framework.

Spin Image Descriptor. Spin-Images [11] describe the surface shape in a short range around a reference point (Figure 2(b)). Given a reference point on the surface, the near-by surface points are projected on its tangent plane, and encoded in a bi-dimensional radial coordinate system that is invariant to rigid transformations. The spin image at the given point is the gray-scale image representing the density of points in this coordinate system. From the invariance of the system, corresponding points in different meshes generate similar spin-images even with clutter and occlusions.

Iterative Closest Point. The Iterative Closest Point (ICP) algorithm [1, 2] iteratively refines an initial alignment of two meshes (Figure 2(c)). It converges to a local minimum that can be the expected rigid transformation, depending on the initial condition. According to [14], the original algorithm and its many variations are composed of six stages: scan points' selection, matching, correspondences generation and filtering, error metric definition and minimization over the rigid transformation. The rigid transformation that minimizes the error is then applied to the scan points and the process is repeated until the rigid transformation is close enough to the identity.

Surface Reconstruction. Surface reconstruction consists in defining a continuous surface representation from a set of isolated points in space [8] (Figure 2(d)). Registration techniques are often used as a pre-processing step for reconstruction [16]. In this work, surface reconstruction serves as a global check for the registration. We illustrate this concept with one global-from-local and one global reconstruction algorithms: the Multiple Partition of Unity (MPU) [20] and the Poisson surface reconstruction [13]. The first one fits a tri-variate polynomial on the scan points contained in each leaf of an adapted octree, and blends these implicit representations by means of radial functions centered at the near-by leaves. The second one builds the characteristic function of the volume inside the surface. To do so, it solves a global linear system whose equations match the pseudo-derivatives of this characteristic function to the normal at each scan point, generating one linear equation for each derivative kernel at the center of an adapted octree.

Error Measures. In order to schedule in our framework which scans are aligned and which are reconstructed, some error measures are derived from each step of the pipeline. The initial positioning error is measured by the geometric consistency of spin image correspondences, i.e. the difference of the distances between the correspondences on each scan. The ICP error, i.e. average of distances between correspondences measures, is used to estimate the alignments accuracy. The reconstruction error is estimated directly from the reconstruction algorithm: the fitting error in each leaf for the MPU case, or the linear solver error in the Poisson case.

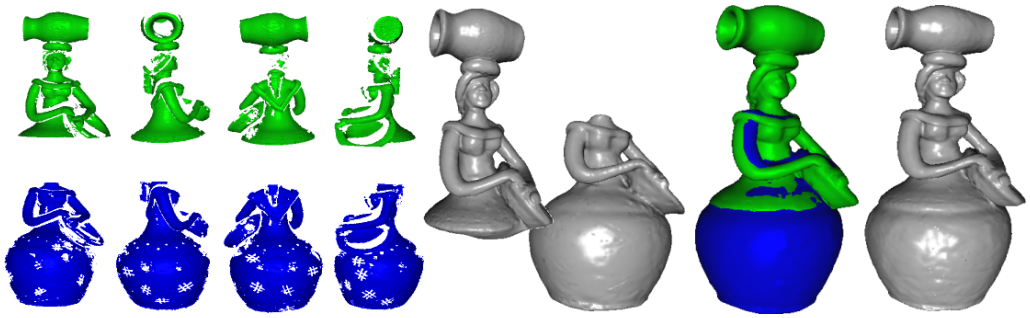


Figure 4: Derivation of our framework for overlapping maximization on a complex model, whose upper and lower parts are acquired in two separate sessions. Registering the models reconstructed separately from the upper and lower parts increases the overlapping parts, improving the stability of the alignment.

3 Registration with Reconstruction Framework

The usual registration pipeline starts with a rough positioning of the scans, which are then aligned by minimizing a distance between them. The aligned scan is then piped into a reconstruction algorithm to produce the final surface. We propose here a general framework to improve the alignment using the reconstructed surface, introducing a feedback in the registration process (Figure 3). This reconstructed surface serves a triple purpose. First, it allows measuring the quality of the registration of each single scan by computing its distance to the reconstructed surface. Second, aligning a scan with the reconstruction improves the registration and optimizes it for reconstruction. Third, this re-alignment guarantees a global coherence of the registration without global optimization, avoiding fake features due to misalignment.

Basic framework. Our framework is built on top of three basic steps:

1. Rough positioning of two distant meshes.
2. Fine alignment of two almost aligned meshes.
3. Reconstruction from several aligned meshes.

The reconstruction step serves to check and improve the alignments in different manners, depending when and how it is scheduled within the framework. For example, the iterative refinement of [24] consists in alternating the two last steps with all the scans at once, aligning them with the last reconstructed mesh.

Note that in steps 1 and 2, one of the meshes to be aligned can be the (partially) reconstructed surface itself. This is the key to transform the local problem into a global one. Furthermore, depending on the scheduling, i.e., which meshes are aligned and reconstructed at each stage, this framework adapts registration to different usages. We comment hereafter derivations of our framework in four different registration contexts: overlapping maximization, dynamic registration, divergence correction and multi-resolution.

Overlapping maximization. Complex objects are generally scanned in several sessions, typically obtained by rotating the objects in front of the scanner (Figure 4). While the overlapping between two consecutive scans of the same session is likely to be high, the overlapping between the scans of two different sessions may be too small to find a stable optimal alignment. Moreover, the rotation angle used inside a session provides a good relative initial positioning of the scans, while the alignment between sessions is not given *a priori*. For example on Figure 4, the upper and lower parts of the object overlap on a very small horizontal band. However, the overlapping of all the scans of one session with all the scans of another session is necessarily more extended. By reconstructing the aligned scan of each session and aligning only the reconstructed meshes, we benefit from this bigger overlapping.

More generally, we can choose at each step to reconstruct with all or only part of the scans, and align pairs of scans, mixed pairs of scan/reconstructed mesh, or only reconstructed meshes. This choice

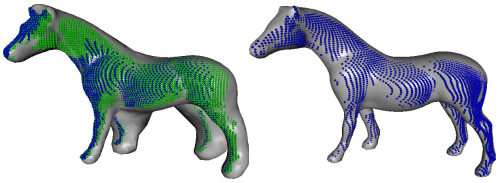


Figure 5: Derivation of our framework for dynamic registration. Partial reconstructions help the alignment, producing a complete reconstruction from only 5 of the 7 scans. (Left) Reconstruction from 2 scans with the Poisson method already generates the two rear legs. (Right) The 6^{th} scan aligned with the reconstruction from 5 scans.

may depend on the available processing time and on the error of each pair-wise ICP, such as the one described at Section 2.

Dynamic registration. When registering on-the-fly during the scanning process, or when computing the initial alignment incrementally, the scan added last must be aligned with all the previous ones (Figure 5). This process can be costly and unstable, in particular if this last scan overlaps with only a few of the previous ones. Since the reconstructed mesh should contain the details of each scan, aligning the new scan with it maximizes the overlapping area, reducing the number of incorrect correspondences. We can thus derive our framework to align the scan added last with the reconstruction of the previous one. The reconstruction can be performed after each scan addition, or when the error between the scans added last and the current reconstruction is bigger than a threshold, which means the aligned scans and the reconstruction are geometrically inconsistent. To improve the stability, the initial scans can be periodically re-aligned with the reconstruction.

Moreover, the registration of the previous scans can be improved by regularly scheduled steps of pair-wise alignments. This strategy is best used in real-time data acquisition for 3D reconstruction.

Divergence correction. Our general framework is an iterative process that converges depending on the initial alignment. When starting from a bad positioning, the process may oscillate between wrong alignments. This can be easily checked by tracking the transformations applied to a specific scan at each stage and the final error after a fixed number of ICP iterations (Figure 8). If these transformations remain far from the identity or if the error does not diminish, the scan can be re-aligned from scratch, using the rough positioning with the mesh reconstructed from all but this scan.

The convergence of our general framework can be improved by alternating global and local alignments: aligning all the scans with the reconstructed mesh (global) and aligning pairs of scans in sequence (local). Each alignment procedure returns an error (cf Section 2), which can be used to correct eventual divergent behaviors.

Multi-resolution. The resolution of the reconstructed mesh can be adjusted, either directly in the reconstruction algorithm tuning or explicitly by mesh simplification and refinement operations. Aligning a low-resolution reconstructed mesh with decimated scans produces a quick and coarse registration. Increasing the resolutions then incrementally improves the alignments. (Figure 6). This coarse-to-fine registration fills the gap between the rough positioning of distant meshes and the ICP alignment, reducing considerably the execution time. Moreover, this strategy improves the robustness of the registration on very noisy objects, avoiding local noise to be considered as features. This way, we obtain good results with only 2 or 3 different resolutions. For example, using two resolutions of 900 and 7000 points for each of the 6 scans, the total execution time for registration of the low resolution, reconstruction of the high one and aligning with the reconstructed surface is 47 seconds, while the conventional registration procedure lasts 256 seconds.

4 Implementation

The overall implementation of our framework is simple provided the basic elements described in Section 2, since meshes are the natural representation of both the original scans and the reconstructed surfaces. We implement the basic steps of the framework by using an automatic pair-wise alignment method for the first step, point-to-point ICP for the second step. For the third step, we use the available implementations of the MPU and Poisson reconstructions *as is*.

Model	# views	# points $\times 10^3$	Frame--work	Posit--ioning secs.	Recons--truction	secs.	ICP secs	# iter.	Error $\times 10^{-4}$
bunny	12	13	corrective	14	MPU	1.6	8.1	9	2.0
head	5	16	basic	41	MPU	1.8	7.2	9	1.8
hand	6	21	mutlires	7.0	MPU	3.5	3.1	2	9.2
horse	7	22	dynamic	4.6	Poi	3.3	4.8	2	3.2
lady bot	8	54	overlap	322	Poi	125	0	0	2.0
lady top	12	174	overlap	660	Poi	1340	75	2	1.5
lady	2	228	overlap	72.5	Poi	2486	0	0	2.0

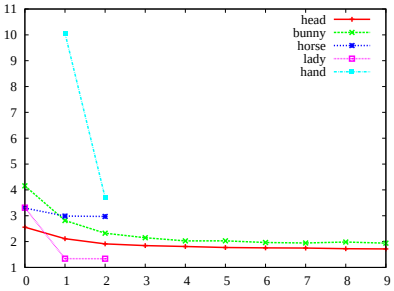


Figure 7: Some experiments on real objects (head, lady) and virtual models (horse, bunny, hand). The graph maps the maximal error from the number of iterations. The timings are averages per iteration, where one iteration corresponds to an alignment and a reconstruction step in the different frameworks. The error corresponds to the maximal ICP error of the last step.

The initial alignment of the first step can be either manual, deduced from the calibrated scanner position, or automatic. For this last case, we use an automatic alignment strategy similar to [22, 9]. Using the terminology of [3], it consists in selecting feature points based on their curvature in each scan, representing them by spin images and ranking matching images by the difference of their pixels. Groups of matching are valid if the distance between them inside each scan is similar. Valid groups can be discarded if they induce a too small overlap. In our experiments, the bin size of the spin-images took values between 2 and 4, and the spin-image width between 10 and 15.

The ICP variant used can be described in the terminology of [14] by: all points selection, matching based on Euclidean closest point using kd-trees, filtering sets of correspondences according to a distance threshold, squared error metric and minimization using quaternions following [7]. Finally, to emphasize the generality of our framework, we use two different reconstruction techniques, respectively [20, 13].

5 Experiments

We consider two sets of examples: artificially generated scans such as the *horse* model (Figure 5), where the correct alignment is the identity, and scanned objects such as the *head* and the *lady* (Figures 1, 2 and 4) to test the robustness to real scanner noise. We forced extreme cases with wrong initial alignments on an artificial scan of the *bunny* (Figure 8), or with flat overlapping area on the low resolution *hand* model (Figure 6).

Precision. In the different frameworks proposed, we obtain a significant improvement compared to a single alignment/reconstruction step involving all the scans. Detailed results on the illustrations of this work are reported on Figure 7. Observe that, for small or decimated models, the several reconstruction steps count only for a fraction of the total time, while it gets a higher proportion on bigger models. Even with initial alignment far from the correct position (Figure 8) the registration converges in a few iterations. Except for the extreme case in Figure 5, results obtained using MPU and Poisson reconstructions are similar.

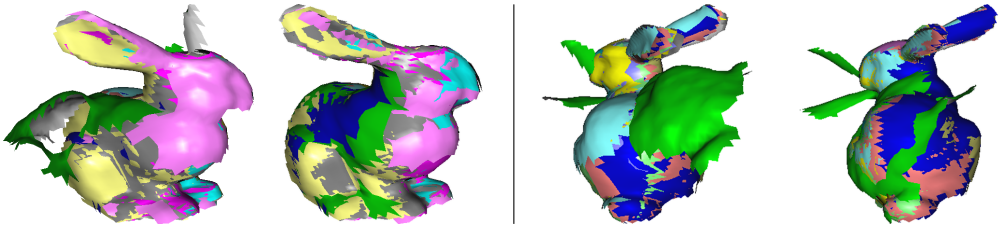


Figure 8: Derivation of our framework for divergence correction. This method improves the robustness of the global registration, and minimizes the impact of bad initial alignments: The error induced by artificially rotating one scan by 60 degrees is progressively corrected in 3 reconstruction/alignment iterations (left). Without the divergence correction, the ICP fits the scan to another position due to its symmetry (right).

Limitations. The proposed framework is very efficient in term of precision and adaptation to diverse contexts. However, since we did not optimize the reconstruction and alignment steps for consecutive calls, the overhead in execution time is still significant on big models without the multiresolution strategy. The resolution settings of the reconstruction have an impact on this execution time, and also on the quality of the alignment, in particular since we use point-to-point ICP. Except for the multiresolution of Figure 6, we let the resolution of all the reconstructions to their default parameters.

6 Conclusions

In this work, we propose a novel framework for registration, inserting and scheduling a reconstruction step to improve the final alignment of the scans. Variations of this scheduling adapt this framework to different contexts. These reconstruction steps provide a feedback to intermediate alignments and guarantees the global coherence of the alignment. They further optimize the alignment for the final reconstruction, avoiding fake features at the frontier of misaligned scans. This framework extends the work of Jin *et al.* [24], adapting global registration to different contexts. Moreover, it gives a simple way for registration to benefit from the recent advances in surface reconstruction.

References

[1] K. S. Arun, T. Huang, and S. Blostein. Least-squares fitting of two 3D point sets. *Pattern Analysis and Machine Intelligence*, 9(5):698–700, 1987.

[2] P. J. Besl and N. D. Mckay. A method for registration of 3D shapes. *Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb. 1992.

[3] B. Planitz, A. Maeder, and J. A. Williams. The correspondence framework for 3D surface matching algorithms. *Computer Vision and Image Understanding*, 97(3):347–383, 2005.

[4] N. Brusco, M. Andreetto, A. Giorgi, and G. M. Cortelazzo. 3D registration by textured spin-images. In *3D Digital Imaging and Modeling*, pages 262–269, Washington, 2005. IEEE.

[5] N. Gelfand, N. J. Mitra, L. J. Guibas, and H. Pottmann. Robust global registration. In *Symposium on Geometry Processing*, pages 197–206. ACM/Eurographics, 2005.

[6] N. J. Mitra, N. Gelfand, H. Pottmann, and L. J. Guibas. Registration of point cloud data from a geometric optimization perspective. In *Symposium on Computational Geometry*, pages 23–32. ACM, 2004.

[7] B. Horn. Closed-form solution of absolute orientation using unit quaternions. *Optical Society of America*, 4:629–642, 1987.

[8] H. Hoppe, T. DeRose, T. Duchamp, J. A. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Siggraph*, pages 71–78. ACM, 1992.

[9] Y. Liu. Improving ICP with easy implementation for free-form surface matching. *Pattern Recognition*, 37(2):211–226, 2004.

[10] Y. Liu, H. Pottmann, and W. Wang. Constrained 3D shape reconstruction using a combination of surface fitting and registra-

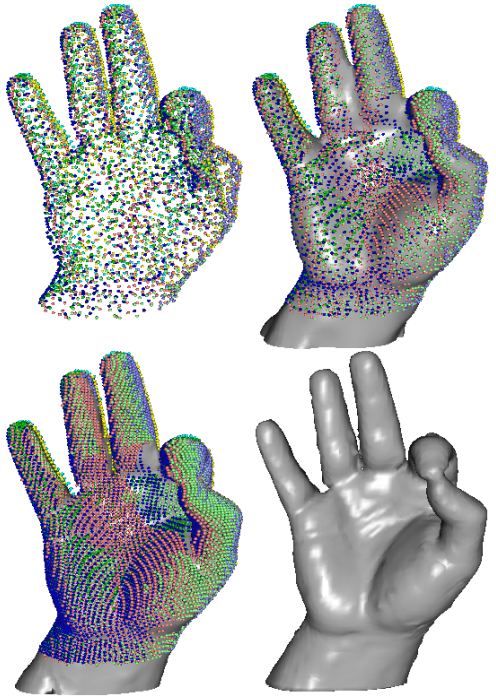


Figure 6: Derivation of our framework for multiresolution registration: aligning decimated scans generates a quick and rough initial positioning (top left), which is reconstructed (top right) and refined by increasing the resolutions of both the scans and the reconstructed surface (bottom). In this example, this process was five times faster than a direct registration.

tion. Technical report, Hong Kong University, 2005.

[11] A. Johnson. *Spin-Images: A Representation for 3D Surface Matching*. PhD thesis, Carnegie Mellon University, Pittsburgh, Aug. 1997. Advised by Martial Hebert.

[12] G. Blais and M. Levine. Registering Multiview Range Data to Create 3D Computer Objects. *Pattern Analysis and Machine Intelligence*, 17(8):820–824, 1995.

[13] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Symposium on Geometry Processing*, pages 61–70. ACM/Eurographics, 2006.

[14] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *3D Digital Imaging and Modeling*, page 145, Los Alamitos, 2001. IEEE.

[15] T. Weyrich, M. Pauly, R. Keiser, S. Heinzle, S. Scandella, and M. Gross. Post-processing of Scanned 3D Surface Data. In *Point Based Graphics*, pages 85–94. Eurographics, 2004.

[16] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Siggraph*, volume 30, pages 303–312. ACM, 1996.

[17] P. Neugebauer. Geometrical cloning of 3D objects via simultaneous registration of multiple range images. In *Shape Modeling and Applications*, page 130, Washington, 1997.

[18] C.-S. Chen, Y.-P. Hung, and J.-B. Cheng. RANSAC-based DARCES: A new approach to fast automatic registration of partially overlapping range images. *Pattern Analysis and Machine Intelligence*, 21(11):1229–1234, 1999.

[19] Y. Chen and G. Medioni. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155, 1992.

[20] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. In *Siggraph*, volume 22, pages 463–470, New York, 2003. ACM.

[21] S. Krishnan, P. Y. Lee, J. B. Moore, and S. Venkatasubramanian. Global registration of multiple 3D point sets via optimization on a manifold. In *Symposium on Geometry Processing*, pages 187–196. ACM/Eurographics, 2005.

[22] D. Huber and M. Hebert. Fully automatic registration of multiple 3D data sets. *Image and Vision Computing*, 21(7):637–650, 2003.

[23] D. Simon. *Fast and Accurate Shape-Based Registration*. PhD thesis, Carnegie Mellon University, Pittsburgh, Dec. 1996.

[24] H. Jin, H. Hoppe, T. Duchamp, J. A. McDonald, K. Pulli, and W. Stuetzle. Surface reconstruction from misregistered data. In *Human Vision and Electronic Imaging*, pages 324–328. SPIE, 1995.

[25] K. Pulli. Multiview Registration for Large Data Sets. In *3dim*, page 160, Los Alamitos, 1999.

3D Reconstruction of Reflection Nebulae from a Single Image

Andrei Linţu[1], Lars Hoffmann[2], Marcus Magnor[2], Hendrik P. A. Lensch[1], Hans-Peter Seidel[1]

[1]MPI Informatik, Germany [2]TU Braunschweig, Germany

Abstract

This paper presents a method for reconstructing the 3D distribution of dust densities in reflection nebulae based on a single input image using an analysis-by-synthesis approach. In a reflection nebula, light is typically emitted from a central star and then scattered and partially absorbed by the nebula's dust particles. We model the light transport in this kind of nebulae by considering absorption and single scattering only. While the core problem of reconstructing an arbitrary 3D volume of dust particles from a 2D image would be ill-posed we demonstrate how the special configuration of light transport paths in reflection nebulae allows us to produce non-exact but plausible 3D volumes. Our reconstruction is driven by an iterative non-linear optimization method, which renders an image in each step with the current estimate of dust densities and then updates the density values to minimize the error to the input image. The recovered volumetric datasets can be used in astrophysical research as well as planetarium visualizations.

1 Introduction

3D models and visualizations of astronomical objects are becoming more and more important nowadays. They are widely used as a tool to prove existing astrophysical theories and to determine various properties of a given astronomical object [6, 15]. They are further used in todays modern planetariums, in which the number of digital video projectors available increased significantly. Realistic 3D flybys of astronomical objects in planetarium shows are becoming rather the rule than the exception. However, most of these animations are created by talented visual artists; in many cases the structure and morphology of the original objects is widely unknown, and therefore the created volume has little in common with astrophysical certainties.

In this paper we address the problem of reconstructing a plausible 3D volume of a reflection nebula (see Figure 1) using only a single image as input, which can be captured by any telescope. We assume that all light reflected by the nebula originates from a single central star, and that the reflected intensity is due to single scattering and the absorption along the path. Scattering and absorption is correlated to the dust volume density which we aim to reconstruct. Besides the simplified light transport model we do not pose any further restrictions on the nebula, most importantly we do not assume any kind of symmetry as has been formulated in previous reconstruction approaches from single images [12, 10].

Figure 1: The NGC 2023 reflection nebula in Orion which (from [4], courtesy Robert Gendler). The cloud of dust is illuminated by the central star.

Of course, the reconstruction of 3D volumes from 2D images is an under-determined problem. We generate plausible reconstructions by making use of two insights: the intensity of each observed pixel is actually the combined effect of the volume densities along all light paths from the central star towards the viewer. Each voxel of the volume we are reconstructing therefore has a non-local effect on the final image. This implicitly regularizes the

possible solution to some extent. A second result of the specific light transport is that the estimated density for voxels close to the central star have a significant impact on the appearance of a large number of pixels while the area of influence decreases drastically with distance. We account for this effect by optimizing the volume densities in concentric shells, starting with the innermost sphere.

The optimization is driven by an analysis-by-synthesis approach. We make use of a hardware-accelerated volume renderer to determine the appearance given the current estimate of the dust densities. The error to the input image is then minimized based on Powell's algorithm [14]. While we do not claim to correctly reconstruct the original nebula's 3D volume we at least reconstruct a plausible dust distribution that matches the input image. Compared to an initial naive traversal of the volume during optimization, we obtain a better match to the given image and a more plausible 3D volume by following an improved traversal during optimization. Plausible reconstructions can be used for visualization purposes or as starting point for further, more complex physical simulations that incorporate additional measurements.

The structure of the remainder of the paper is as follows: we present related work in the next section. The physics of reflection nebulae and interstellar dust are shortly described in Section 3 and the used lighting model in Section 4. Section 5 presents our reconstruction approach as well as several optimization steps for this specific task. The results of the described reconstruction method are presented in Section 6 and we conclude the paper and present future work in Section 7.

2 Related Work

Recently, there have been large efforts to produce 3D simulations of astronomical nebulae, particularly the Orion nebula [13], where a highly detailed model of the nebula was created, based on data from astrophysical research papers. Time consuming renderings on supercomputers were generated for the final fly-through animation.

There are tomographic reconstruction algorithms that consider scattering in the reconstructed volumes [3] or even diffuse propagation [1]. However, they always build on observations from different projection directions. In our case, due to the large distance to the reflection nebulae only a single projection direction is typically available.

A method for reconstructing the 3D structure of another type of astronomical objects, planetary nebulae, from a single image has been proposed in [12]. The appearance of planetary nebulae is mostly influenced by the self-emission of ionized gas, slightly simplifying the reconstruction problem. Furthermore, the authors apply axial symmetry as a constraint to reduce the complexity of the reconstruction process from 3D to reconstructing a 2D density map, which is rotated around the axis of symmetry to obtain a volumetric dataset. Building on the same idea of axial symmetry the work has been extended to incorporate both emission as well as scattering and absorption found in planetary nebulae with non-negligible dust distributions [10].

In contrast to these methods, our approach is not considering any symmetry in the nebulae to be reconstructed. Instead, we make use of the implicit coupling of volume densities induced by the light transport in reflection nebulae and present an optimization strategy that produces unconstrained, plausible 3D volumes.

For physically-based, realistic volume rendering of synthetic reflection nebula datasets a hardware-accelerated approach has been presented in [11]. The authors of this paper give special attention to the physical correctness of their rendering model. Lacking the availability of real-world 3D volume data sets, the examples presented in the paper are synthetic nebulae which however strongly resemble the possible appearance of real reflection nebulae. We apply and adapt this rendering approach in Section 5.1 to our optimization procedure.

The main contribution of this paper is that we recover a physically plausible 3D volume of an astronomical nebula from a single input image. The obtained datasets can be used as starting approximations by astrophysicists working with 3D models of reflection nebulae as well as for educational purposes in modern day planetarium shows.

3 Reflection Nebula Physics

Reflection nebulae are clouds of dust surrounding one or more young, recently formed stars [2]. The stars in these nebulae are not hot enough to ionize the gas around them but their light is strong enough, so that the reflected light can be observed (see Fig-

ure 1). Their wonderful colors are due to the light of the central star which is scattered and attenuated by the dust surrounding it.

The blue colors seen in these nebulae are mostly due to scattering. This is due to the fact that light at blue wavelengths scatters much more than light at the red end of the visible electromagnetic spectrum. The best example which demonstrates this is the color of the sky; it is blue because it consists of sunlight scattered by the particles in the atmosphere. All reddish colors visible in reflection nebulae account for absorption, the light from the central star gets attenuated and reddened on its way to an observer on earth.

3.1 Interstellar Dust

The physics behind reflection nebulae is actually the physics of the interaction of light with interstellar dust particles. We create a physically-based model of light scattering in interstellar dust using a set of scalar parameters. One is the *albedo* α, which indicates how much light the particles reflect: 0 for total absorption, i.e. black dust, and 1 for the case where all incident light is scattered. In our calculations, we set $\alpha = 0.6$ [5]. In addition, the presence of dust also attenuates any light shining through the dust region, which can be described by an *extinction* parameter τ.

The scattering is further dependent on the angle θ between the incident and the reflected light direction. It is described by the *single particle scattering probability* modeled using the Henyey-Greenstein phase function [7]:

$$\phi(\theta) = \frac{1 - g^2}{2 \cdot (1 + g^2 - 2 \cdot g \cdot \cos\theta)^{3/2}}, \quad (1)$$

where g is an anisotropy factor for forward and backward scattering. Observations and measurements show [5], that equation 1 describes the scattering properties of interstellar dust with a value for $g \approx 0.6$. The effect of varying the anisotropy factor g is described in more detail in [11].

4 Lighting Model

Our volume renderer is based on the following lighting model: the observed radiance $L(x, y)$ at a camera at position c is a function of the aggregated extinction $\tau(v)$, and the albedo $\alpha(v)$ that depend

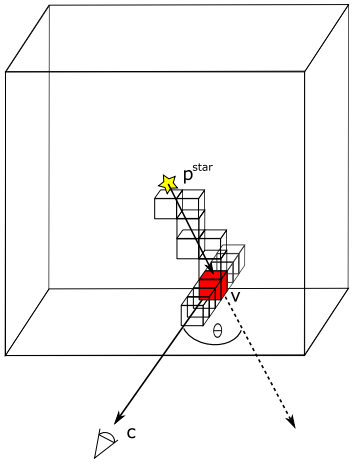


Figure 2: The intensity of the light reflected due to single scattering caused by the red voxel depends on the absorption along the path from the central star to the voxel, the voxel's albedo, the phase function, as wells as on the accumulated absorption along the remaining path towards the viewer.

on the dust density of every voxel v along the ray towards the camera pixel (x, y):

$$L(x, y) = \int_c^\infty e^{-\int_c^v \tau(w)dw} \cdot \alpha(v) \cdot S(v)dv, \quad (2)$$

where $S(v)$ is the total inscattering to the voxel v towards the camera due to the emission L_e^{star} of the nebula's central star(s) at position p^{star}. Considering single scattering only, $S(v)$ is computed as

$$S(v) = \phi(c, v, p^{star}) L_e^{star} \cdot e^{-\int_{p^{star}}^v \tau(w)dw} \quad (3)$$

incorporating the extinction on the way from the star to the voxel as well as the scattering phase function ϕ from the star to the voxel into the direction of the camera c. We assume the same phase function for all voxels. ϕ, L_e^{star}, and p^{star} are assumed to be known. We consider the star to be positioned in the center of the volume. L_e^{star} can be looked up in astrophysical data bases for a specific nebula.

Due to the multiple scattering within a single voxel, the extinction $\tau(v)$ and the albedo $\alpha(v)$ depend non-linearly on the dust density $d_{dust}(v)$. The dust density is the quantity we actually attempt to reconstruct. We precomputed the effective extinction and albedo for a the selected size of a voxel and

all possible dust densities using Monte Carlo simulation, even considering multiple scattering within one voxel. We perform the precomputation for a discrete set of densities spaced $\Delta\varrho$ apart and linearly interpolate for intermediate densities. Our goal is to determine $d_{dust}(v)$ up to a scale factor.

5 3D Reconstruction

Our analysis-by-synthesis approach is driven by the non-linear Powell optimization algorithm [14]. During optimization, we render the current dust density distribution using the equations described in Section 4 and the rendering framework in Section 5.1, and, at every step, we compare the rendering with the given input reflection nebula image.

Before starting the reconstruction process, the input images are cropped to a square and converted to grayscale. One has to make sure that the central star is placed in the center of the image. Since we do not consider self-emission foreground stars are masked out in order to avoid interference with the reconstruction.

The reconstruction process is based on minimizing the following error functional

$$Err = \sum_{x,y=1}^{n} ||L(x,y) - L_{inp}(x,y)||^2 \qquad (4)$$

which is computed using the currently rendered $L(x,y)$ (Equation 2) and the input image L_{inp}.

As can be seen in Figure 2, each voxel has a non-local effect on the rendered image, resulting in some implicit regularization of the optimization. However, this is not constraining the under-determined problem entirely. In the Powell algorithm each voxel is optimized independently. During optimization, the order in which the voxels are processed is very important for the speed and quality of the reconstruction. A naive traversal in a loop over the x, y and z coordinates yields poor reconstruction results, as demonstrated in Figure 3.

The reconstruction artifacts are due to the fact that the radiance reflected by voxels at the outer rim of the volume depends on the dust density values of the inner voxels. During optimization, changes to the inner voxels require updates of the outer voxels which can only be performed after the entire volume has been optimized in one iteration step.

We obtain much better results (Figure 5) when the traversal is done starting from the center to the outside. Voxels closest to the central star are traversed first, resulting in a set of concentric spheres with center at the point of the central star (see Figure 4).

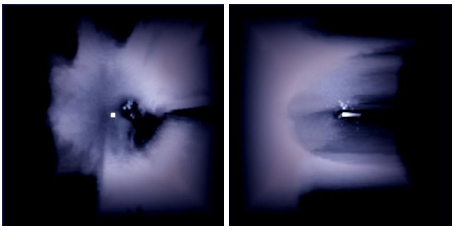


Figure 3: Inadequate optimization results of NGC 1999 by processing all voxels in scan-line order. Note the decreasing reconstruction quality with increasing distance to the central star. In the side-view (right), an unnatural distribution along the z-axis is visible: the dust is concentrated in the region furthest away from the observer.

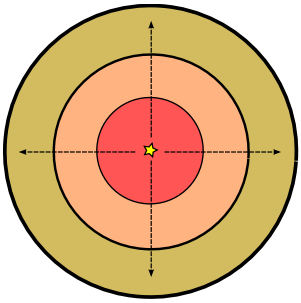


Figure 4: During optimization the volume is traversed starting from the center, in concentric shells.

Another observation is that it is better not to start the reconstruction process with an empty volume. It is filled homogeneously with dust densities as small as the smallest $\Delta\varrho$ with which we increment or decrement the voxel dust densities during optimization.

5.1 Rendering and Runtime Optimization

As the lighting calculations are the most computationally expensive, we can obtain a great speedup by accelerating them. We have developed two different options:

To render full frames, and to visualize the results of the reconstruction, we use an OpenGL-based renderer similar to that used for reflection nebulae visualization by Magnor et al. [11]. It is a volume rendering application based on a real time ray-caster implemented on graphics hardware. The implemented algorithm uses the idea described by Krüger et al. [9] to color code the direction of the viewing rays using a bounding box. The algorithm exploits the capabilities of modern graphics hardware to step along the lines of sight querying 3D textures, using a fragment shader to accumulate the inscattering along each ray while considering absorption.

While the hardware-based renderer is significantly faster than a software renderer for computing a full frame, we observe that during the optimization of individual voxels only fractions of the images are actually affected. During optimization, while constantly changing the density of one voxel at a time, only those pixels in the rendered image have to be recomputed which are affected by the new voxel value. The footprint of each voxel is precomputed since it only depends on the ray geometry and is independent on the actual volume density.

A further speed-up as well as an additional means to avoid local minima during the optimization process is achieved by an iterative multi-resolution approach. The reconstruction starts with a lower resolution volume which is scaled up after the end of every iteration. This is done by subdividing each voxel into eight sub-voxels with the same dust density value and applying a 3D Gaussian filter on the dataset for smoothing the resulting high frequencies. We measure a speed-up of factor 2 in the final reconstruction compared to optimizing performed directly at the highest resolution.

6 Results

We present the results of our proposed reconstruction approach for several reflection nebulae: NGC 1999 (Figure 5), the Iris Nebula (Figure 6, top row) and the Cocoon Nebula (Figure 6, bottom row). As already mentioned in section 5, the reconstruction is performed for a grayscale flux image only, speeding up the computation. The here presented results were rendered using spectrally dependent absorption and scattering coefficients. Since our lighting model does not include self-emission we manually removed all stars in the input images since they cannot be recovered. To produce slightly more realistic final renderings we sometimes added artificial star fields approximating the original image. Reconstruction times for the presented results are between 1-2 days on a 2.4GHz PC with 4GB memory.

For all nebulae we presented the original image, the reconstructed volume rendered from the same view, as well as a rendering of the volume rotated by 90 degrees around the vertical axis. In the frontal views, one can see that the large-scale features are very well reproduced. Some detail has been lost though because the reconstruction has been performed at a maximum resolution of 64^3 voxels. In the bottom right corner of Figure 5, we further present an image of the relative differences between the captured and the reconstructed image, which are overall relatively small. The values are normalized, with 1 corresponding to a difference of 255 in grayscale values.

In the frontal view, the largest error is observed close to the central star. This is because of the lack of self-emission in our framework and can be explained by the algorithm trying to compensate for the high intensity values in the center of the input image (where the central star is situated) by incorrectly adjusting the dust density.

Looking at the side views, one sees a reconstruction that radiometrically agrees with the provided input data, and thus is physically plausible. However, the reconstructed distribution along the z-axis might not necessarily match the expected statistical distribution. It looks slightly too smooth compared to the frontal view. However, it still resembles a reasonable nebula. A good reconstruction is achieved for the Cocoon Nebula (Figure 6, bottom row). We attribute this effect to the slightly less inhomogeneous distribution in the input image. We also have to mention that this nebula has also an emission part, thus not being best suited for reconstruction as a pure reflection nebula.

The most prominent artifacts of our reconstruction are possibly the diagonal features visible in all three side views. While we are not exactly sure, this might be caused by the way how the aggregated extinction $\tau(v)$ and albedo $\alpha(v)$ are precomputed for a single voxel and for every density. During evaluation the voxel is assumed to be perfectly isotropic which of course contradicts the anisotropic shape of a cube, with the largest deviation exactly along the diagonal.

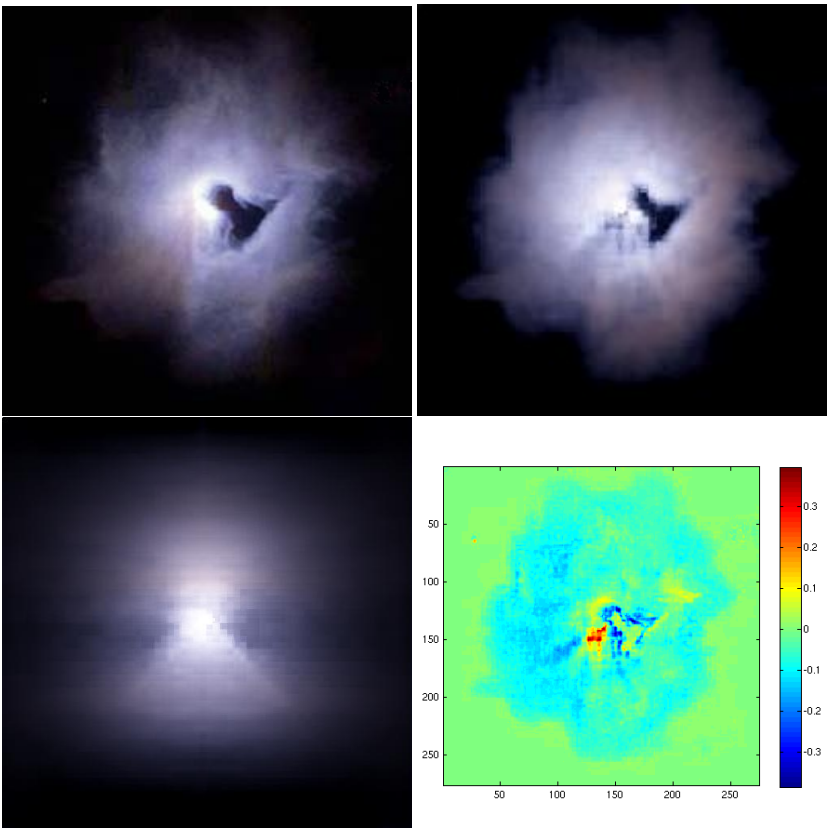


Figure 5: Results for the NGC 1999 nebula. The top row shows the input image (left) and the rendering of the reconstruction from the same viewpoint (right). The bottom row shows a side view of the recovered dust density (left). A color coded difference image between the input image and the rendering of the recovered dust density is presented in the bottom right corner, indicating the good quality of the reconstruction.

In Figure 7, we demonstrate the dependence of the reconstruction results on the intensity of the central stars. We reconstructed the same input image of the nebula NGC 1999 with three different values for L_e^{star}. In order to reproduce the same input pixel intensity, a fainter central star leads to higher dust density values. Conversely, a brighter central star leads to smaller dust density values. With the increasing dust density the different reconstructions also show an increased reddening as explained in Section 3. This effect could actually be used to optimize for the intensity of the central star as well, if wavelength dependent effects are considered during the optimization.

7 Conclusion and Future Work

We presented a reconstruction method for physically plausible volumetric models of reflection nebulae given only single input images. We do not pose any geometric constraints on the shape of the nebulae, such as symmetry which has been applied in previous reconstruction methods. Using an analysis-by-synthesis approach we perform a nonlinear optimization to recover the dust density values. The recovered datasets can later be visualized using a custom renderer and can also be used as starting point for planetarium shows or further physical simulations. While we so far concentrated on the reconstruction of reflection nebulae, it would

Figure 6: From left to right: input image, reconstructed frontal and rendered side view. The top row shows the reconstruction for the Iris Nebula and in the bottom row we present renderings of the Cocoon Nebula. Note, how the structures of the input image are well reproduced in the frontal view. The side views indicated that plausible volumes have been reconstructed.

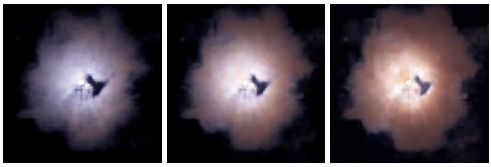


Figure 7: Reconstruction results for different intensity values of the central star, decreasing from left to right. A smaller luminance level leads to higher dust density values and to reddening of the nebula.

be interesting to apply our reconstruction algorithm to other volumetric phenomena that feature single scattering and absorption.

One meaningful extension of our method would be to incorporate multi-wavelength input images; the renderings of the recovered dust density would have to match multiple input images taken with different band filters. Besides the ability of estimating the exact brightness of the central star, as indicated in the previous section, one might be able to obtain a more precise volumetric reconstruction since additional independent constraints are given.

Another promising future research direction would be to add further constraints to the optimization in order to force the statistical distribution of densities along the z-axis to match the statistics of the input image. One could apply for example histogram matching or similar techniques which recently has been successfully applied in the context of solid texture synthesis [8]. Adding such an additional constraint should be rather easy in our current optimization framework.

8 Acknowledgements

Our thanks go to Christian Fuchs for his valuable comments on early drafts of the paper. We would also like to thank the peer reviewers for their valuable comments and constructive critique to improve the final version of the paper. This work has

been partially funded by the Max Planck Center for Visual Computing and Communication (BMBF-FKZ01IMC01).

References

[1] S. Arridge. Optical tomography in medical imaging. *Inverse Problems*, 15:R41–R93, 1999.

[2] James Binney and Michael Merrifield. *Galactic astronomy*. Galactic astronomy / James Binney and Michael Merrifield. Princeton, NJ : Princeton University Press, 1998. (Princeton series in astrophysics), 1998.

[3] Andrei V. Bronnikov. Numerical solution of the identification problem for the attenuated Radon transform. *Institute of Physics, Electronical Journals*, 1999.

[4] Rob Gendler. The universe in colors. available from `www.robgendlerastropics.com`, 2007.

[5] K. D. Gordon. Interstellar Dust Scattering Properties. In A. N. Witt, G. C. Clayton, and B. T. Draine, editors, *Astrophysics of Dust*, volume 309 of *Astronomical Society of the Pacific Conference Series*, pages 77–+, May 2004.

[6] K. D. Gordon, K. A. Misselt, A. N. Witt, and G. C. Clayton. The DIRTY Model. I. Monte Carlo Radiative Transfer through Dust. *The Astrophysical Journal*, 551:269–276, April 2001.

[7] Louis G. Henyey and Jesse L. Greenstein. Diffuse Radiation in the Galaxy. *Astrophysical Journal*, 93:70–83, 1941.

[8] Johannes Kopf, Chi-Wing Fu, Daniel Cohen-Or, Oliver Deussen, Dani Lischinski, and Tien-Tsin Wong. Solid texture synthesis from 2d exemplars. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3):to appear, 2007.

[9] Jens Krüger and Rüdiger Westermann. Acceleration techniques for GPU-based volume rendering. In Greg Turk, Jarke J. van Wijk, and Robert Moorhead II, editors, *14th IEEE Visualization 2003 Conference (VIS 2003), 19-24 October 2003, Seattle, WA, USA*, pages 287–292. IEEE Computer Society, 2003.

[10] Andrei Linţu, Hendrik P. A. Lensch, Marcus Magnor, Sascha El-Abed, and Hans-Peter Seidel. 3D Reconstruction of Emission and Absorption in Planetary Nebulae. In Hans-Christian Hege and Raghu Machiraju, editors, *IEEE/EG International Symposium on Volume Graphics*, September 2007. to appear.

[11] Marcus Magnor, Kristian Hildebrand, Andrei Linţu, and Andrew J. Hanson. Reflection Nebula Visualization. In C.T. Silva, E. Gröller, and H. Rushmeier, editors, *Proceedings of the IEEE Conference on Visualization (VIS'05)*, pages 255–262, Minneapolis, USA, October 2005. IEEE.

[12] Marcus Magnor, Gordon Kindlmann, Charles Hansen, and Neb Duric. Constrained inverse volume rendering for planetary nebulae. In *Proc. IEEE Visualization 2004, Austin, USA*, pages 83–90, October 2004.

[13] David R. Nadeau, Jon D. Genetti, Steve Napear, Bernard Pailthorpe, Carter Emmart, Erik Wesselak, and Dennis Davidson. Visualizing stars and emission nebulas. *Comput. Graph. Forum*, 20(1):27–33, 2001.

[14] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, MA, 2. edition, 1992.

[15] K. Wood, J. S. Mathis, and B. Ercolano. A three-dimensional Monte Carlo photoionization code for modelling diffuse ionized gas. *Monthly Notices of the Royal Astronomical Society*, 348:1337–1347, March 2004.

Invited Talk

Images, Images, Billions of Images . . .

Michael Goesele

GRIS
TU Darmstadt, Germany

Abstract

Today, there are literally billions of images available online, indexed by search engines according to image content, geographical location, and other features. Graphics and vision researchers recently started to incorporate these images into their algorithms as a valuable source of real-world imagery. In this talk, I will first give an overview over some recent work in this area. I will then introduce our work on dense geometry reconstruction from Internet images using multi-view stereo techniques. I will close with an outlook on some open questions in the field.

Filtered Blending: A new, minimal Reconstruction Filter for Ghosting-Free Projective Texturing with Multiple Images

Martin Eisemann, Anita Sellent, Marcus Magnor

Computer Graphics Lab, TU Braunschweig
Email: {eisemann,sellent,magnor}@cg.tu-bs.de

Abstract

Whenever approximate 3D geometry is projectively texture-mapped from different directions simultaneously, annoyingly visible aliasing artifacts are the result. To prevent such ghosting in projective texturing and image-based rendering, we propose a new GPU-based rendering strategy and a new, view-dependent definition of ghosting. The algorithm is applicable to any kind of image-based rendering method, or general projective texture mapping, and adapts to arbitrary camera setups. It is able to cope with imprecise 3D geometry. Ghosting artifacts are efficiently eliminated at real-time rendering frame rates on standard graphics hardware. With the proposed rendering technique, better-quality rendering results are obtained from fewer images and coarser 3D geometry.

1 Introduction

Approximate geometry, camera calibration inaccuracies, and subcritical sampling are the reason for ghosting artifacts in light field rendering [15], lumigraph rendering [11], and view-dependent projective texture mapping [8]. In fact, ghosting/aliasing/double images[1] are a problem common to all image-based modeling and rendering applications whenever recorded image footage is to be merged with available geometry into one consistent representation. For highly accurate, laser-scanned geometry, a number of strategies have been devised for how to register photographs to 3D geometry in the presence of calibration inaccuracies [27, 3, 14], as well as how to generate a globally consistent texture map from multi-view footage [22, 2, 28]. While impressive digital models of real-world objects have been created this way, they come at the price of considerable user interaction, and not all approaches are suitable for reproducing view-dependent reflectance effects as the created texture map is usually not view-dependent.

In this paper, we present a new algorithm to achieve aliasing-free rendering results directly from a set of photographs in conjunction with some arbitrarily coarse geometry proxy. The contributions of this paper are

- a new view-dependent, anisotropic reconstruction filter that is able to take camera distribution, sampling density, the new virtual viewpoint and geometry inaccuracy into account and that is applicable to any kind of projective texturing;
- a real-time implementation of the proposed filter on standard graphics hardware;
- a new, conservative, yet more general definition for the causes of ghosting in projective texturing which takes the current viewpoint into account and provides a new upper bound on the highest representable frequency without ghosting, which results in more details in the output image.

Our main goal is to improve the visual quality of existing image-based modeling and rendering methods as well as to simplify the use of image-based approaches for representing real-world objects.

Our paper is organized as follows. After reviewing relevant previous work in Section 2 we examine the underlying problem of ghosting artifacts in multi-image projective texture mapping, Section 3. In Section 4 we describe *filtered blending* as a way to eliminate ghosting. Implementation details are given in Section 5, and experimental evaluation results are presented in Section 6 before we conclude with Section 7.

[1] Throughout the paper, we use the terms "ghosting", "double images" and "aliasing" synonymously if not stated otherwise.

Figure 1: Images from our test data sets: for the synthetic *Bunny* and the real-world captured *Garfield*, approximated 3D geometry models are available. For the synthetic light fields *Buddha* and *Dragon*, a planar surface must suffice as geometry proxy, see Table 1 for more information on our test data sets.

	Bunny	**Garfield**	**Buddha**	**Dragon**
# geometry primitives	948	1280	1	1
Total images	49	24	256	256
Pixels per image	512^2	768×576	256^2	256^2
Uncertainty offset	0.63%	1.18%	7.07%	8.13%
Band-limit filter support	12 pixels	10 pixels	12 pixels	10 pixels
Viewport	$360° \times 360°$	$360° \times 180°$	$90° \times 90°$	$90° \times 90°$
Output resolution (pixels)	512^2	512^2	512^2	512^2
Type	synthetic	real-world	synthetic	synthetic

Table 1: Information concerning our test data sets shown in Figure 1. The uncertainty offset along the viewing ray in positive and negative direction, in comparison to the diagonal of the geometries bounding box.

2 Related Work

Image-based rendering (IBR) methods are able to achieve highly realistic rendering results of real-world objects or scenes from a collection of calibrated photographs. While some IBR methods rely solely on a large number of input images to minimize aliasing artifacts [15, 18], most IBR approaches make additional use of scene depth [11, 13, 4, 29], or full 3D geometry [8, 5, 26, 23, 24]. Potential sources for aliasing artifacts during rendering are (1) image calibration inaccuracies, (2) subcritical sampling in conjunction with insufficient pre-filtering [6, 16], and possibly (3) imprecise depth maps or inexact geometry.

Image-based modeling (IBM) extends the notion of IBR in that high-quality 3D geometry scans of an object are augmented with a collection of photos to capture the visual appearance [22, 27, 2, 14, 28]. Finite scanner resolution and tolerances, registration inaccuracies, and camera calibration errors all degrade overall image-to-texture mapping accuracy.

Different reconstruction filters for IBR have been investigated in the literature. Based on an analysis of the sampling problem in frequency and geometry space by Chai *et al.* [6] and Lin *et al.* [16], respectively, one can apply a low-pass filtering to the input/output images for ghosting-free "band-limited reconstruction" [25]. Isaksen *et al.* [13] propose a "wide-aperture reconstruction filter" which increases the spatial support or aperture size of the reconstruction filter. A combination of the two approaches is proposed by Stewart *et al.* [25]. Alternatively, Liu *et al.* [17] estimate scene geometry dynamically using a color similarity-based plane sweeping algorithm.

These approaches effectively reduce ghosting, but several issues remain unsolved. Not all of these approaches can be applied to general projective texturing. Many details are lost because the filtering operations are usually performed as a pre-processing step based on the maximum disparity, not taking the current viewing position into account. Not all approaches are able to preserve view-dependent reflectance characteristics. And a lot of user-interaction may be needed. In some approaches new artifacts may be introduced due to random color similarities. With wider camera baselines, these mismatch artifacts increase disproportionately.

Warping Techniques which establish dense correspondences between pixels in different images can produce most accurate results [7, 19]. However, specular and occluding surfaces pose a great challenge for the necessarily precise automatic camera calibration and depth acquisition. On the other hand, using only a sparse feature set, is usually problematic as feature detectors may select significantly different features in different images of the same scene. In addition many feature detectors are specialized to certain environments [1], and may not work as well for other settings.

3 Problem Description

In this section, we take a closer look at the causes of ghosting in projective texture mapping and light field rendering. We show that ghosting is solely dependent on the maximum disparity of a projected scene point to its real position in texture space. Therefore, ghosting can be detected even if only a single input camera and the virtual camera is taken into account.

Every pixel of the input images can be seen as the weighted integral of the light arriving at the image plane of the camera. Every scene point L of sufficiently small size, compared to the camera resolution, therefore contributes exactly to one pixel in the recorded images (for more details see [16]). During rendering, these are then reprojected onto an approximated surface. Lin *et al.* [16] state that the intensity contributions of each scene point L must at least touch each other in the output image to avoid ghosting. This is true for light field approaches if virtual and capturing cameras have the same resolution and the user is restricted to stay outside the convex hull defined by camera and focal plane. However, other rendering approaches, like projective texturing and geometry-assisted IBR, demand a more general definition of ghosting. Thus we propose that every scene point L must provide a single, resolution independent intensity maximum in the output image, Figure 2.

As one example, let's consider the viewing ray C_vL , Figure 2. By projecting the input image of camera C_1 onto the approximate surface S, the contribution of L will not appear at point F_0, but rather at point F_1, revealing a disparity of d. If d, projected into the input image, is larger than half a pixel, ghosting artifacts may appear, as the

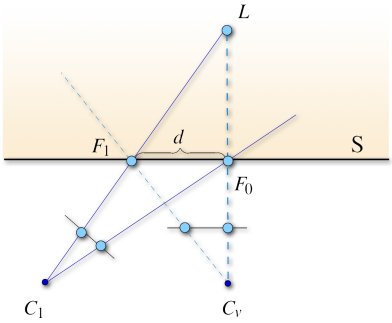


Figure 2: Ghosting in projective texture mapping, view-dependent texture mapping and light field rendering: The actual scene point L, as recorded from cameras C_1 and C_v, is projected to two different points F_1 and F_0 on the approximate object surface S. If the projected distance $d = F_0 - F_1$ is larger than half a pixel in the input images, ghosting occurs.

main contribution of L might not appear at F_0 after blending different input image values together.

This definition of ghosting is applicable to IBR in general because it gives room for view-dependent filtering of the input images, based on the current viewpoint, as we will show in Section 4. Note that our definition reduces to the one proposed by Lin *et al.* if their preconditions are fulfilled.

4 Filtered Blending

To motivate our approach, consider the diagram in Figure 3. The scene point L lies on the line of sight of the viewing ray $C_vL_{p_0}$, somewhere within the interval of maximum depth uncertainty d_{max} from the approximate geometry, which for this analysis we assume to be known. The line segment $L_{p_1}L_{p_2}$ projected into the texture space of camera C_1 reveals another line segment $T_{L_{p_1}}T_{L_{p_2}}$, which we call the line of disparity. Any value on this line could be the correct texture value. This is in fact similar to an epipolar geometry constraint [12].

We solve this uncertainty problem in a resampling process. Recall that ghosting is prevented if the projected disparity d in the texture images is less than $1/2$ texel (see Section 3). In frequency domain, let ω_t be the highest representable frequency in the texture function t, which is given by

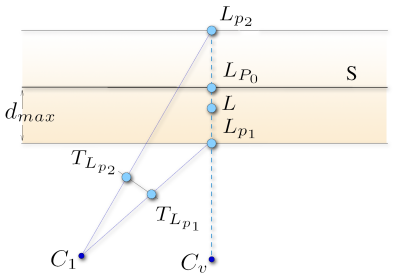


Figure 3: Scene point estimation. The scene point L observed from viewpoint C_v can only be estimated to lie somewhere between L_{p_1} and L_{p_2}, which are defined by the maximum depth uncertainty d_{max}. Its correct color value observed by camera C_1 lies somewhere between the projected texture coordinates $T_{L_{p_1}}$ and $T_{L_{p_2}}$.

its resolution and defined by the *Nyquist Theorem*. Then it follows that if we want to remove ghosting, frequencies higher than $\frac{1}{2d}\omega_t$ have to be removed. But as we are dealing with discrete values, simply removing these frequencies is insufficient. One also has to consider appropriate sampling positions. Choosing $\frac{1}{2}T_{L_{p_1}}T_{L_{p_2}}$ as the sampling position and $|T_{L_{p_1}}T_{L_{p_2}}| - \epsilon$ as the sampling distance, with $\epsilon \rightarrow +0$, we anisotropically resample the texture function along the line of disparity at the highest possible frequency which assures that no ghosting will appear. This way we effectively avoided ghosting, since the correct texture values always contribute to the corresponding output pixels. As we take the current viewpoint into account, the closer the virtual camera is to one of the input cameras, the fewer frequencies are cut off from that input image and the output image will contain much more details than in a band-limiting approach. If the input camera and virtual camera coincide, all detail is preserved. This way, we implicitly take the input camera distribution into account, as the size of our filter is based on the geometric uncertainty and position of the input cameras.

The depth uncertainty itself can be established in different ways. In two-plane parameterized light field rendering, it is usually the difference along the z-axis from the focal plane, which is orthogonal to this axis by definition. For synthetic scenes the value of uncertainty is usually known in advance, it is to be estimated for real world scenes. Then L_{p_1} and L_{p_2} can be calculated by intersecting every viewing ray with the plane at Z_{min} and Z_{max}, which are the minimum and maximum z-values in the scene, respectively. In a more general setup, one could decide to either create an offset along the normal of the approximate surface with which the viewing ray is intersected, or the offset is created along the viewing ray. In the first case the calculated disparity becomes very large at objects silhouettes, as the normal is almost perpendicular to the viewing ray's direction. This leads to strong and distracting blurring artifacts. We therefore chose to calculate the offset along the viewing ray. This results in a small blur for images close to the current viewpoint and larger blur for those farther away. This is similar to an angular metric, and no sudden jumps at triangle borders appear.

Since the support of the applied low-pass filter can theoretically become arbitrarily large, we take two simple steps to alleviate the needed effort. First, we make strong use of GPU processing power. The whole filtering algorithm is implemented as a pair of vertex and fragment shaders. Second, we trade off detail for speed by applying a multi-resolution technique. We set a threshold ν for the filtersize μ in texture space. If this threshold is exceeded we use the n-th level of the input image-pyramid computed in a preprocess instead of the image itself, with $n = \log_2(\frac{\mu}{\nu})$. Note that this approach has almost no effect on the visual quality of the output, since a large filter size implicates a small weighting factor for an input camera and therefore only a small contribution to the output image, but speeds up the whole rendering process by a factor of roughly three.

Interestingly, depending on the movement, the varying change in blur in the output image can evoke the impression of repeatedly changing speed, even if a movement is in fact constant. We can solve this perceptual problem by applying a simple motion blur technique. If the viewpoint does not change, the image quickly converges to the optimally filtered solution.

Note that our algorithm is independent of the weighting scheme used for the image synthesis step, where the acquired texture values are combined to reveal the final pixel value, and can be used in conjunction with quadralinear interpolation [15], the unstructured lumigraph weighting scheme [4] or angular distance measures [9, 21].

5 Implementation

We implemented our algorithm on a NVidia GeForce 8800GTX graphics card using OpenGL/GLSL. For the filtered blending approach we add/subtract the estimated or apriori known depth uncertainty offset along the viewing ray from the vertex position. Reprojecting the new positions into the different input images yields the texture coordinates.

For the resampling process during filtered blending, we implemented the Mitchell-Netravali cubic B-spline filter as a fragment shader program [20]. We compared different filters, e.g., also truncated Gaussian and box filter, and found that the Mitchell-Netravali filter yields the visually most convincing rendering results.

6 Results, Discussion

Our test data sets include one real-world object, *Garfield*, one synthetically created 3D object, *Bunny*, and the two well-known Stanford light fields *Buddha* and *Dragon*. Figure 1 shows examples of the test data sets. Additional data, like the used depth uncertainty or size of the band-limiting filter, is listed in Table 1. To evaluate rendering quality, we compare our rendering strategies to direct (quadra-)linear interpolation as well as to preprocessed band-limited filtering. For band-limited filtering, the filter support is set to the smallest possible value to prevent ghosting. In projective texture mapping, we always select the three nearest cameras for interpolation as described in [21].

Our first test scene *Bunny* consists of 49 images rendered from randomly selected viewing directions. The upper two rows in Figure 4 depict the results obtained by the different rendering approaches. For better rendering quality assessment, some of the details are enlarged. In the first row, the virtual camera is close to one of the input cameras, in the second it is placed right between them. When comparing the two leftmost images, which correspond to standard linear blending (Figure 4a) and band-limited filtering (b) to our result on the right(c), note how the ghosting is smoothed away by the filtered blending, while discontinuities of the checkerboard texture are much better preserved. We achieve 342 fps when using our filtered blending approach.

To acquire the calibrated images for the *Garfield* scene a computer-controlled turntable and a digital camera with a lever arm was used to record 24 images in hemispherical configuration. The geometry is estimated using the voxel-based approach by Eisert *et al.* [10]. Rendering results are shown in Figure 4 third row. Again, the noticeable ghosting artifacts along the pupil in the image on the left are eliminated in our approach, while some of the details at the nose could be preserved. The *Garfield* model is rendered at approximately 334 fps.

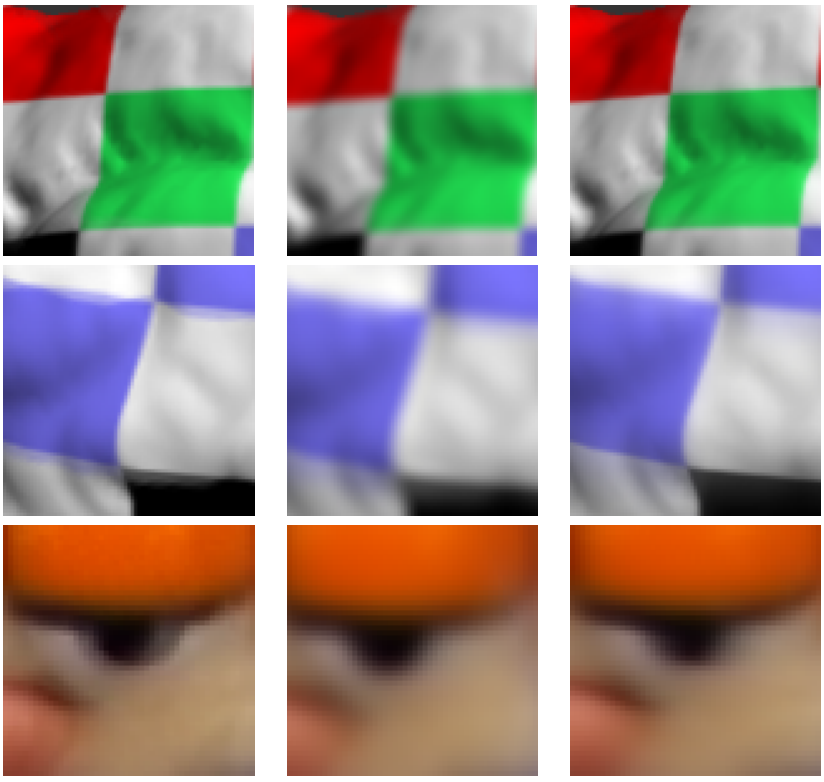
We also tested our "ghost-busting" approach for light field rendering using the *Buddha* and *Dragon* data sets (Figure 1). Rendering results are shown in Figure 5. Notice how ghosting is prevented in our approach, Figure 5(c), while ghosting artifacts are obvious in standard quadralinear interpolation (a). At the same time, much finer detail is preserved than if pre-processed band-limited filtering is used (b). In conjunction with light field rendering, we achieve around 105 fps with our approach.

7 Conclusions

We have presented an approach to achieve ghosting-free rendering results with viewpoint optimized, minimal, low-pass filtering for subcritically sampled light fields, as well as for general projective texture mapping with approximated geometry. In contrast to conventional methods based on prefiltering/band-limiting, our algorithm efficiently eliminates ghosting and preserves texture details considerably better, because our filtered blending takes the current viewpoint into account. Our definition of ghosting is able to conservatively establish the bounds for ghosting. Real-time rendering performance is achieved on a standard GPU. And the approach can be easily adapted to various different image-based rendering scenarios.

However there are certain limitations. If the input samples are too sparse, the image will still look blurry. Warping techniques might help to visually increase the resulting image even more. We are currently working on this.

Filtered blending greatly eases the constraints of image-based rendering: coarser 3D geometry, and fewer input images are sufficient to still achieve convincing rendering results. With this useful generalization, image-based rendering will hopefully find many new practical applications.



(a) Linear interpolation (b) Band-limited (c) Filtered Blending

Figure 4: Projective texture mapping of the *Stanford Bunny* (rows 1 and 2) and of the real-world data set *Garfield* (row 3). In row 1 the virtual camera is close to one of the input cameras, in row 2 and 3 it is set in-between two of them: (a) Linear interpolation reveals strong ghosting around high frequency details (row 2 and 3). (b) Band-limited reconstruction removes ghosting, but the result is excessively blurred. (c) Our filtered blending approach preserves discontinuities considerably better and completely removes aliasing. Notice the much sharper edges on the bunny in row 1 and 2 and the preserved brown stripe on the Garfield's nose.

References

[1] Daniel G. Aliaga, Dimah Yanovsky, Thomas Funkhouser, and Ingrid Carlbom. Interactive Image-Based Rendering Using Feature Globalization. In *I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 163–170, New York, NY, USA, 2003. ACM Press.

[2] Adam Baumberg. Blending images for texturing 3d models. In Paul L. Rosin and A. David Marshall, editors, *Proceedings of the British Machine Vision Conference*, 2002.

[3] Fausto Bernardini, Ioana M. Martin, and Holly Rushmeier. High-quality texture reconstruction from multiple scans. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):318–332, 2001.

[4] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured Lumigraph Rendering. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 425–432, 2001.

[5] J. Carranza, C. Theobalt, M. Magnor, and H. P. Seidel. Free-viewpoint video of human actors. In *SIGGRAPH '03: Proceedings of the 30th annual conference on Computer graphics and interactive techniques*, pages 569–577, 2003.

[6] Jin-Xiang Chai, Shing-Chow Chan, Heung-Yeung Shum, and Xin Tong. Plenoptic Sampling. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 307–318, 2000.

[7] Shenchang Eric Chen and Lance Williams. View Interpolation for Image Synthesis. Number Annual Conference Series, pages 279–288, 1993.

[8] Paul Debevec, George Boshokov, and Yizhou Yu. Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping. *9th Eurographics Rendering Workshop*, pages 105–116, 1998.

[9] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 11–20, 1996.

[10] Peter Eisert, Eckehard Steinbach, and Bern Girod. Multi-hypothesis volumetric reconstruction of 3-d objects from multiple calibrated camera views. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP'99)*, pages 3509–3512, 1999.

[11] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The Lumigraph. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54, 1996.

[12] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2 edition, 2003.

[13] Aaron Isaksen, Leonard McMillan, and Steven J. Gortler. Dynamically Reparameterized Light Fields. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 297–306, 2000.

[14] Hendrik P. A. Lensch, Jan Kautz, Michael Goesele, Wolfgang Heidrich, and Hans-Peter Seidel. Image-based reconstruction of spatial appearance and geometric detail. *ACM Trans. Graph.*, 22(2):234–257, 2003.

[15] Marc Levoy and Pat Hanrahan. Light Field Rendering. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42, 1996.

[16] Zhouchen Lin and Heung-Yeung Shum. A Geometric Analysis of Light Field Rendering. *International Journal of Computer Vision*, 58(2):121–138, 2004.

[17] Yang Liu, George Chen, Nelson Max, Christian Hofsetz, and Peter McGuiness. Undersampled Light Field Rendering by a Plane Sweep. *Computer Graphics Forum*, 25(2):225–236, June 2006.

[18] W. Matusik and H. Pfister. 3D TV: A scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes. In *SIGGRAPH '04: Proceedings of the 31st annual conference on Computer graphics and interactive techniques*, pages 814–824, 2004.

[19] Leonard McMillan and Gary Bishop. Plenoptic Modeling: An Image-Based Rendering System. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 39–46, 1995.

[20] Don P. Mitchell and Arun N. Netravali. Reconstruction Filters in Computer-Graphics. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 221–228, 1988.

[21] Kari Pulli, Michael Cohen, Tom Duchamp, Hugues Hoppe, Linda Shapiro, and Werner Stuetzle. View-Based Rendering: Visualizing Real Objects from Scanned Range and Color Data. In Julie Dorsey and Phillipp Slusallek, editors, *Proceedings of the Eurographics Workshop on Rendering*, pages 23–34, 1997.

[22] Claudio Rocchini, Paolo Cignoni, Claudio Montani, and Roberto Scopigno. Multiple textures stitching and blending on 3D objects. In *Proceedings of the Eurographics Workshop on Rendering*, pages 119–130, 1999.

[23] N. Snavely, S. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. In *SIGGRAPH '06: Proceedings of the 33rd annual conference on Computer graphics and interactive techniques*, pages 835–846, 2006.

[24] J. Starck and A. Hilton. Surface capture for performance based animation. *IEEE Computer Graphics and Applications*, 27(3):21–31, 2007.

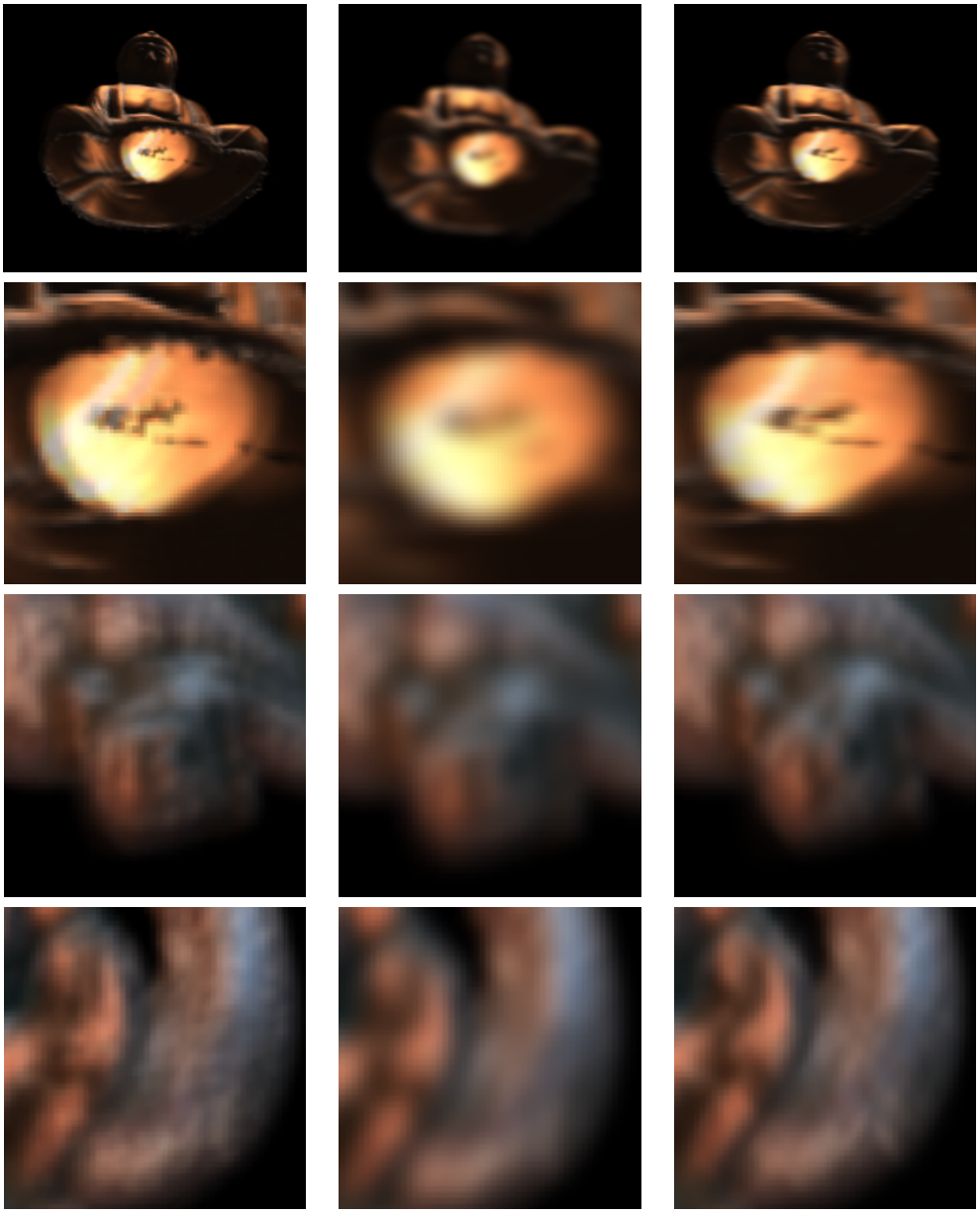
[25] J. Stewart, J. Yu, S. J. Gortler, and L. McMillan. A New Reconstruction Filter for Undersampled Light Fields. In *Proceedings of the Eurographics Workshop on Rendering*, pages 150–156, 2003.

[26] S. Vedula, S. Baker, and T. Kanade. Image based spatio-temporal modeling and view interpolation of dynamic events. *ACM Transactions on Graphics*, 24(2):240–261, April 2005.

[27] Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. Surface light fields for 3d photography. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 287–296, 2000.

[28] Kun Zhou, Xi Wang, Yiying Tong, Mathieu Desbrun, Baining Guo, and Heung-Yeung Shum. Texturemontage: Seamless texturing of arbitrary surfaces from multiple images. In *SIGGRAPH '05: Proceedings of the 32nd annual conference on Computer graphics and interactive techniques*, pages 1148–1155, 2005.

[29] C. Zitnick, S.B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. In *SIGGRAPH '04: Proceedings of the 31st annual conference on Computer graphics and interactive techniques*, pages 600–608, 2004.



(a) Quadra-linear interpolation (b) Band-limited (c) Filtered Blending

Figure 5: Comparison for the two sub-critically sampled light fields *Buddha* (top rows) and *Dragon* (bottom rows): (a) Quadralinear interpolation cannot suppress ghosting artifacts. (b) Band-limiting the entire light field leads to excessively blurry results. (c) Our filtered blending approach preserves most of the details while ghosting is completely avoided.

Freehand HDR photography with motion compensation

Nicolas Menzel, Michael Guthe

Philipps-Universität Marburg
FB12, Graphics and Multimedia Programming
Email: {menzel,guthe}@informatik-uni-marburg.de

Abstract

In theory HDR photography generated from exposure sequences is limited to pictures of still scenes preferably taken with the camera mounted on a tripod. Current state-of-the-art HDR imaging software relaxes this limitation by aligning the pictures with affine transformations. However, two pictures cannot be aligned with a single affine transformation if objects in the scene move or the position of the camera changes which causes parallax effects in the image.

In this paper, a method for the non-linear alignment of a picture sequence for HDR imaging is presented. By using techniques for motion detection and compensation, the approach is not only capable to correctly align pictures containing parallax effects, but also able to compensate for the movement of objects in the scene. This way, taking freehand HDR pictures from non-static scenes becomes possible with off-the-shelf digital cameras.

1 Introduction

The dynamic range of an image is the perceived ratio between the darkest and the brightest details of the depicted scene. This ratio easily exceeds the capabilities of a common photograph taken with a single exposure. From shadows to fully lit regions the radiance of objects in a single image can easily span several orders of magnitude. In sunlit scenes, dynamic ranges of $1 : 100,000$ or more are not unusual. It is obvious that this range cannot be captured by an ordinary digital camera with 256 quantization levels and a dynamic range of about $1 : 100$ for compact cameras and $1 : 300$ to $1 : 1,000$ for SLR cameras. Either dark, bright, or medium lit areas can be stored in full detail, all other radiance values are lost after the acquisition process.

The full dynamic range of a scene can be obtained by taking multiple photographs, each one with a different exposure time. This way it is guaranteed that shadowed areas as well as very bright areas are captured in full detail. The next step prior to combining the images is the reconstruction of the camera response function that relates color values to light intensities. The discrete response function is a non-linear curve that can differ for each color channel and is stored as a lookup table containing 256 values. After this step, the radiance map is computed as a weighted sum of pixel intensities from the images with different exposure times.

For the recovery of the dynamic range it is crucial for all pixel locations to be exactly aligned. This constraint originates in the assumption that the irradiance of each pixel remains constant throughout the whole exposure sequence. For this purpose it is advisable to create exposure sequences using a tripod and in addition, a remote controlled release. Besides avoiding undesirable vibrations, the scene has to be static as well, which makes capturing of naturally moving objects, such as clouds, leaves or waves, difficult to impossible. Up to now, these constraints limit high dynamic range (HDR) photography to professionals or very dedicated hobby photographers. This restriction can only be lifted by a tool that is able to align the images of an exposure sequence by removing any motion origination from camera or object movement.

To construct a robust freehand motion compensation algorithm, it is important to understand which effects may occur besides translation of the image:

- When photographs are taken by hand, rotation is inevitable, though very small angles of typically less than 20 degree can be expected.
- Even the slightest translation may cause severe parallax effects, e.g. when looking through a window or straight down a wall.

- In addition to the parallax, occlusion can also occur whenever the camera or an object moves.
- In contrast to traditional image matching, it cannot be assumed that finding matching patterns between different exposures is always possible. Detailed areas in one image are possibly saturated or at noise level in the next or previous image of the sequence, making it impossible to find a direct matching. This can be interpreted as missing data problem.

Considering these problems, we propose a two step algorithm. In the first step, the images are aligned as faithfully as possible and in the second step, the occlusion and missing data problems are handled by a robust HDR reconstruction method including a ghost removal step. The contributions of this paper are the following:

- A hierarchical non-linear alignment algorithm that is robust with respect to missing data due to black level noise and saturation.
- A robust HDR reconstruction algorithm that removes the remaining artifacts originating from occlusion while preserving information that is not contained in the other images of the sequence.

2 Related Work

Initially, Debevec and Malik proposed using multiple exposures to recover the full dynamic range [4] of a scene. Their proposal arises from the observation that if one pixel has twice the value of another, it cannot be concluded that it received twice the irradiance. Instead, the irradiance of a pixel is determined by an unknown, nonlinear function called the response function or response curve. Once the response curve is reconstructed, it can be re-used for the same camera type. However, as the response curve is badly conditioned at extreme intensities, Debevec and Malik introduced a linear weighting function to assure that values near saturation have a lower contribution to the final image. The HDR image is then computed as a weighted average of each pixel, thus containing information captured by each of the images. Later, Robertson et al. introduced a smoother gaussian weighting function [10] that better fits the accuracy range of a CCD sensor.

The generation of HDR images from exposure sequences requires perfectly aligned pictures since the intensity is calculated on a per pixel basis. If this is not the case, the pictures need to be aligned before HDR reconstruction. A fast and robust method for image alignment developed by Ward [12] is based on a median cut in combination with an XOR fitness function and is able to resolve translations between images in the sequence. Grosch extended this method to rotations [6] using a downhill simplex solver and exploiting graphics hardware to improve performance. The drawback of this method is that, despite its robustness to moving objects, only an alignment of static parts of the scene is possible. The remaining artifacts, originating from object movement or parallax, are removed in a second, semi automatic phase at the cost of a reduced dynamic range. The ghost removal proposed by Khan et al. [8] improves this step by not only considering the probability that a pixel is correctly exposed, but also the background probability of the pixel using an estimation scheme based on a d-variate Gaussian density function. For each pixel in every input image a small neighbourhood is considered to determine the contribution to the final image. Although the results are convincing, mere ghost removal cannot replace image alignment, since it reduces the dynamic range in areas where only a single image contributes to the final result.

If not only the static background, but also moving foreground objects are to be aligned, a single linear transformation is not sufficient. A similar non-linear matching problem needs to be solved for state-of-the-art video codecs like MPEG-4 [13]. In this case, so called macroblocks from one image need to be found in a corresponding image. This matching problem is typically solved using optical flow techniques [1, 2].

In the context of temporal image processing, Kang [7] presented a method to create a sequence of HDR images from a stream of LDR input images. For this purpose, the input images are captured with alternating exposure times using a programmable capture device. To generate an output HDR image for each input LDR image, neighbouring images have to be aligned, combined and tone mapped. Subsequent images are registered by estimating an affine transform that maps one onto the other. A gradient-based optical flow is used to compute a dense motion field to form a local correlation. Our approach is related to this in the sense that always three neighboured images are registered. In-

stead of an affine transform we use macroblocks for the alignment. For motion estimation, a very accurate and – with respect to different luminance – robust methods is the cross-correlation [9]. Another possibility to solve the non-linear alignment based on optical flow would be the use of dense gradient matching [3, 11]. However, this technique only works well if the luminance in both images is approximately identical. While this could be achieved by mapping an image to the dynamic range of the reference image, the method is not robust with respect to noise and thus will fail at dark and bright regions of the reference image which are however the most important parts of the other images as they contain most of the additional information.

3 Overview

The first step of the algorithm is the non-linear image alignment using macroblocks and maximizing the cross-correlation. The alignment is performed hierarchically to assure that matching blocks as well as non-matchable blocks are moved into a consistent direction. An in depth description of this process is given in section 4. After the alignment, most pixel locations match exactly but occlusion and parallax effects inside a single leaf level macroblock cannot be resolved, leading to ghosting artifacts. This is solved in the HDR reconstruction phase described in section 5 by calculating a per-pixel confidence value and marking non-plausible pixels such that they do not contribute to the final image.

4 Hierarchical Matching

For the non-linear matching, first an anchor image r is chosen by simply selecting the one containing the highest entropy and then all other images are aligned to this one. One way to capture the input images freehand is to use the automatic exposure bracketing (AEB) function of the camera. The AEB function induces the camera to capture a sequence of optimal exposed, underexposed and overexposed images. We expect the optimal exposed image to contain the highest entropy and set this image as anchor for every macroblock. Since capturing more than three images seems impractical for ad hoc purposes, our current implementation is limited to three input images, though the full dynamic range of outdoor scenes might not be covered with three input images only. In addition, if more than three images are used, finding an optimal anchor image might be a difficult task, since optimal correlation can expected only for neighbouring images. The basic idea for our motion estimation is that for a given macroblock M in the anchor image, the displaced matching macroblock in image i will have the locally maximal cross correlation $C_i(M, \delta)$. Note that in contrast to the original formulation, we do not normalize the vectors before calculating the cross correlation. While this leaves the location of the maximum unchanged, it additionally delivers a confidence for the match.

$$C_i(M,\delta) = \frac{\sum_{p\in M} \tilde{\mathbf{c}}_i(p+\delta)\cdot\tilde{\mathbf{c}}_r(p)}{N(M)\sqrt{\sum_{p\in M}\|\tilde{\mathbf{c}}_i(p+\delta)\|^2}}$$

with

$$\tilde{\mathbf{c}}_i(p+\delta) = \mathbf{c}_i(p+\delta) - \begin{pmatrix}1\\1\\1\end{pmatrix}\frac{\sum_{p\in M}\|\mathbf{c}_i(p+\delta)\|_1}{3N(M)}$$

$$\tilde{\mathbf{c}}_r(p) = \mathbf{c}_r(p) - \begin{pmatrix}1\\1\\1\end{pmatrix}\frac{\sum_{p\in M}\|\mathbf{c}_r(p)\|_1}{3N(M)},$$

where $\mathbf{c}_k(p)$ is the RGB color vector of image k at pixel p, and $N(M)$ is the number of pixels contained in the macroblock. To emphasize local contrast, the edges in all input images are enhanced using the following filter kernel F:

$$F = \begin{pmatrix} 0 & -\frac{1}{4} & 0 \\ -\frac{1}{4} & 2 & -\frac{1}{4} \\ 0 & -\frac{1}{4} & 0 \end{pmatrix},$$

where pixels at the boundaries are duplicated for consistent filtering.

Unfortunately, choosing a good macroblock size that works for every input sequence is not possible. If the size is too large, local movements cannot be compensated for, while too small macroblocks cannot always be matched reliably in the absence of local detail. Therefore, the image i is aligned in a hierarchical manner starting with a single centered square macroblock of size 2^n that is large enough to contain the whole image. After the best match for this root block is found, it is divided into four sub blocks and for each of them the best match is searched recursively while re-using the alignment of the previous level as initial guess. Hierarchical

guessing is prone to be less robust towards large motion of small objects. The problem is that while the confidence value of a large macroblock might by high, the motion of smaller objects cannot be estimated until a finer subdivision level is reached. In order to find the object, the algorithm then has to increase the search radius instead of actually reducing it in order to converge. Here, a trade-off between robustness of motion estimation and computing speed has to be found. To prevent random displacement when the macroblock contains noise only, a block is not displaced if its cross correlation less than 10% of its parent block's cross correlation. If this value is lower for the current block, it gradually decreases from level to level. During recursive search, the maximum search radius is reduced by a factor of two at each subdivision until it reaches two at a block size of 16×16 pixel which allows to also compensate for rotations of up to 14 degree.

At each recursion step, the displacement is bilinearly interpolated between the four adjacent parent nodes. Then an iterative search is performed for each macroblock starting with its interpolated displacement and that of its up to eight neighbors. For each initial displacement, the next local maximum is found using a gradient descent method with a fixed step size of one pixel, while for the root block, a step size of four pixels is used in order to find the global maximum. The search from the current start displacement is stopped when either the local maximum is found, or the number of iterations reaches the search radius of the current level. When the local maxima are found for each of the nine starting displacements, the final displacement for the current block is chosen to be that one with the highest cross correlation. If the maximum is reached for more than one different displacement, the closest one of them to the initial interpolated displacement is chosen to prevent displacement of macroblocks that do not contain local detail. Figure 1 shows the hierarchical alignment process for an input image where the clouds had moved with respect to the reference image.

After calculating the displacement vectors for the finest level macroblocks, the transformed image is assembled. For each macroblock, the corresponding pixels – without edge enhancement – are copied from the displaced position to their original position in the reference image. This way, the features of the current image become aligned with

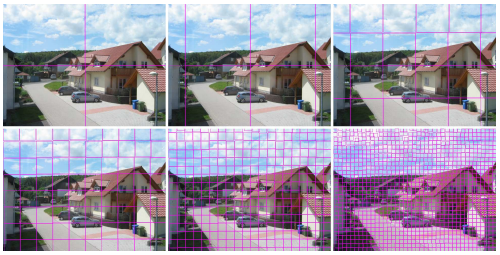


Figure 1: Hierarchical alignment of an image where the clouds had moved upwards and right during the exposure sequence. The magenta squares depict the matching macroblocks.

their corresponding features in the reference image. To prevent the well known blocking artifact of macroblock based motion compensation, the displacement is bilinearly interpolated per pixel for the transformation of the image. While this might introduce some distortions in badly aligned region, the overall image quality is greatly improved. Figure 2 shows the difference of the example image after alignment to the reference image mapped into the same exposure. Notice that most of the difference is due to highlights that are over saturated in the aligned image. The movement of the clouds and the slight foreground parallax have been corrected.

Figure 2: Difference of aligned image and reference image converted into same exposure. The darker the color, the higher is the difference

5 HDR Image Synthesis

The HDR reconstruction is based on a per-pixel weighted average of the measured logarithmic irradiance. Thus, the irradiance has to be reconstructed from each image as first step by calculating the camera response function f using the least squares solver proposed by Debevec and Malik [4]. The accumulated irradiance $E(p)$ at the current pixel p is

then calculated from the color values c_i and the exposure times t_i of each image i for each color channel in the following way:

$$\log E(p) = \frac{\sum_i w_i(p) \log f(c_i(p)) - \log t_i}{\sum_i w_i(p)}$$

The key to both, faithful reconstruction and artifact removal now lies in an appropriate weighting function. The first weighting function proposed by Debevec and Malik [4] was a simple piecewise linear function:

$$w_i(p) = \begin{cases} c_i(p) & : \quad c_i(p) \leq 127.5 \\ 255 - c_i(p) & : \quad c_i(p) > 127.5 \end{cases}$$

The drawbacks of this function are the non-zero derivatives at both ends and the relatively high weighting of extreme values. These introduce discontinuities and can reduce the dynamic range of the final image. While the first problem was solved with the gaussian weighting function of Robertson et al. [10] the range reduction is even more apparent. To solve both problems, we propose a quadratic spline weighting function that has zero values and derivatives at both ends:

$$w_i(p) = \begin{cases} 8\left(\frac{c_i(p)}{255}\right)^2 & : \quad \frac{c_i(p)}{255} < \frac{1}{4} \\ 1 - 8\left(\frac{c_i(p)}{255} - \frac{1}{2}\right)^2 & : \quad \frac{1}{4} \leq \frac{c_i(p)}{255} \leq \frac{3}{4} \\ 8\left(1 - \frac{c_i(p)}{255}\right)^2 & : \quad \frac{c_i(p)}{255} > \frac{3}{4} \end{cases}$$

Due to the low weight for extreme values, the result might however become noisy if the pixel is almost black or white in all images of the exposure sequence. Therefore, the weight is set to one in the longest exposure image if the color value is below 127.5 and in the shortest exposure image if the value is above 127.5 since these contain the most accurate information. Figure 3 shows the resulting weight functions for a sequence of five images with two f-stops between each successive pair.

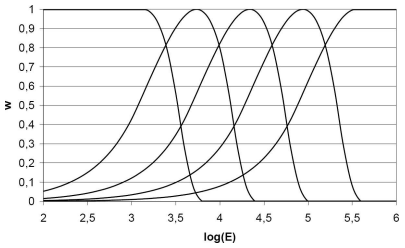


Figure 3: Weight functions for a sequence of five images with an exposure stepping of two f-stops and a gamma-2.2 camera response curve.

5.1 Ghost Removal

While these intensity based weighting functions only account for the accuracy of the CCD sensor, Grosch [6] additionally proposed to test the luminance and color value of a pixel against the corresponding one in an anchor image for plausibility. If the difference between expected value and sampled value exceeds some error threshold, the pixel is treated as ghost and its weight is set to zero. This plausibility test however is prone to inaccuracies of the camera response function and thus the confidence map often needs to be edited by the user.

Instead, we propose the following ghost removal technique: For each pixel in the all images, except the anchor image r, a confidence value, derived from the cross-correlation, is calculated. A small neighborhood N_p around the pixel p is extracted from each image i and the confidence $K_i(p)$ is calculated based on these:

$$K_i(p) = \frac{\sum_{j \in N_p} \mathbf{c}_i(j) \cdot \mathbf{c}_r(j)}{\sqrt{\sum_{j \in N_p} \|\mathbf{c}_i(j)\|^2} \sqrt{\sum_{j \in N_p} \|\mathbf{c}_r(j)\|^2}}$$

By considering the neighboring pixels instead of the camera response curve, the result is much more robust and less sensitive to noise. Note, that in contrast to the cross-correlation used for matching, the mean is not subtracted from the blocks since this would prevent detecting differences of the average luminance. If the confidence of a pixel falls below a threshold t_K, its weight is set to zero. The only exception is made for the shortest and longest exposure images: if the only information about a pixel is contained in one of these images – i.e. the color values are below the black noise level t_b or above saturation t_s in all other images – the confidence test is skipped and thus the weight is not altered. Altogether, this leads to the final weight:

$$w_i^*(p) = \begin{cases} 0 & : K_i(p) < t_K, t_b < c_i(p) < t_s \\ w_i(p) & : \text{else} \end{cases}$$

In the actual implementation, we used a neighborhood of 11×11 pixel centered at the current pixel and $t_K = 0.95$, $t_b = 0.05$, and $t_s = 0.95$ as threshold values.

Figure 4 shows the regions identified as non-plausible in the aligned example image from section 4. The non-matchable lens reflection, as well as some less accurately aligned features are removed.

Figure 4: Ghost pixels in an aligned image.

6 Results

Figure 5 shows the three exposure sequences used as test data to compare our approach with linear alignment. The first sequence is included to demonstrate that our method delivers as least as good results as linear alignment. Figure 6 shows a comparison of tone-mapped images generated with the two techniques. As tone mapping operator, gradient domain HDR compression [5] is used. The difference between the two images is almost invisible and ghost removal is required for neither of them.

The second example demonstrates the superiority of our approach in the presence of moving objects. Figure 7 shows a tone-mapped version of the HDR

Figure 5: Image sequences used for evaluation: The top row shows a sequence that can be aligned using linear transformations, while the middle sequence contains moving objects (the clouds) and the bottom one significant parallax effects and occlusions.

Figure 6: Tone-mapped results generated from the first sequence using linear alignment (left) and our non-linear alignment (right). Both are generated without ghost removal.

image generated with our approach in comparison with linear alignment. While the ghost removal is capable to remove most of the artifacts of the linear alignment, except for the still slightly blurry clouds (see magnifications), the dynamic range of the image is degraded. The improved alignment of our method does not only reduce the misalignment artifacts, but also exhibits a higher dynamic range of the final image after ghost removal, e.g. in the highlights of the roof tiles.

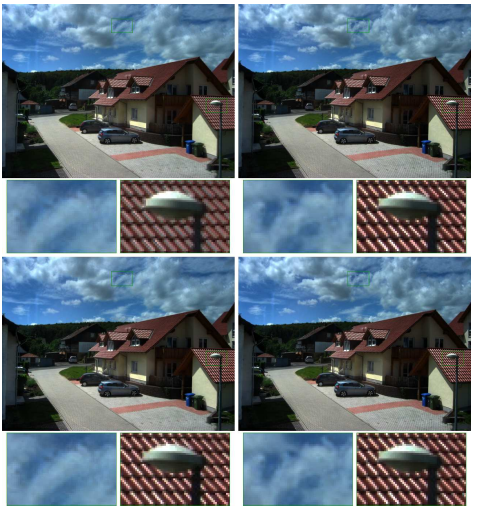


Figure 7: Tone-mapped results generated from the second sequence using linear alignment (left) and our non-linear alignment (right). The top row is without and the bottom with ghost removal.

The final example is chosen to exploit the limitations of our technique. Due to the significant parallax and large occluded areas, an alignment is not possible for all parts of the image sequence. Thus

the final HDR image (a tone-mapped one is shown in figure 8) contains many artifacts in these areas. While the ghost removal is capable of removing almost all artifacts – except for a region in the upper right part of the image that was completely white in two of the images and occluded in the third one – it degrades the dynamic range of the image. Nevertheless, the result with our technique is significantly better than with linear alignment, where even the ghost removal is not able to resolve all ambiguities.

Figure 8: Tone-mapped results generated from the third sequence using linear alignment (left) and our non-linear alignment (right). The top row is without and the bottom row with ghost removal.

7 Conclusion and Future Work

We have presented a method to generate HDR images from picture sequences that are taken without a tripod or automatic exposure sequence and can even contain moving objects (e.g. clouds, foliage, and people). Motion is identified using a hierarchical macroblock matching based on cross-correlation. This does not only compensate for moving objects, but also for movements of the camera that lead to parallax effects. While the current implementation has a runtime of less than 15 seconds for a sequence of three pictures and a resolution of about one mega pixel on a 2.4 GHz CPU, an implementation on current graphics hardware will be able to align high resolution pictures within a few seconds.

A limitation of the method is that it cannot compensate for large occluded areas that were visible in the reference image and thus solely relies on the ghost removal technique during the second step in such cases. If no data is available in the other images (i.e. the region is completely black or white) the irradiance cannot be reconstructed. Another limitation is that the technique can only compensate small rotations about the z-axis of the camera. Thus a possible extension would be to allow rotation the the macroblocks during the alignment phase.

References

[1] J. L. Barron, D. J. Fleet, S. S. Beauchemin, and T. A. Burkitt. Performance Of Optical Flow Techniques. In *CVPR,* 236–242, 1992.

[2] J. L. Barron, S. S. Beauchemin, and D. J. Fleet. On Optical Flow. In *6th Int. Conf. on Artificial Intelligence and Information-Control Systems of Robots (AIICSR),* 3–14, 1994.

[3] U. Clarenz, M. Droske, and M. Rumpf. Towards Fast non–rigid registration. In *Proc. of the AMS,* 2002.

[4] P. E. Debevec and J. Malik. Recovering High Dynamic Range Radiance Maps from Photographs. *Computer Graphics,* 31(Annual Conference Series):369–378, 1997.

[5] R. Fattal, D. Lischinski, and M. Werman. Gradient domain high dynamic range compression. *Computer Graphics,* 36(Annual Conference Series):249–256, 2002.

[6] T. Grosch. Fast and Robust High Dynamic Range Image Generation with Camera and Object Movement. In *Vision, Modeling and Visualization (VMV),* 2006.

[7] S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High dynamic range video. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2003),* 22(3):319–325, 2003.

[8] E. Khan, O. Akyuz, and E. Reinhard. Ghost removal in high dynamic range images. *Proceedings of ICIP'06,* 2006.

[9] J. P. Lewis. Fast Normalized Cross-Correlation. In *Vision Interface,* 120–123, 1995.

[10] M. Robertson, S. Borman, and R. L. Stevenson. Dynamic Range Improvement Through Multiple Exposures. In *Int. Conf. on Image Processing,* III:159–163, 1999.

[11] R. Strzodka, M. Droske, and M. Rumpf. Image registration by a regularized gradient flow - a streaming implementation in DX9 graphics hardware. *Computing,* 73(4):373-389, 2004.

[12] G. Ward. Fast, robust image registration for compositing high dynamic range photographs from hand-held exposures. *Journal of Graphics Tools,* 8:17–30, 2003.

[13] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra. Overview of the H.264/AVC Video Coding Standard. *IEEE Trans. on Circuits and Systems for Video Technology,* 13(7):560–576, 2003.

Sketch Based Image Deformation

Mathias Eitz Olga Sorkine Marc Alexa

TU Berlin
Email: {eitz,sorkine,marc}@cs.tu-berlin.de

Abstract

We present an image editing tool that allows to deform and composite image regions using an intuitive sketch-based interface. Users simply draw the outline of the source image region and sketch a new boundary shape onto the location where this region is to be pasted. The deformation optimizes a shape distortion energy, and we use Poisson cloning for subsequent compositing. Since the correspondence between the source and target boundary curves is not known a priori, we propose an alternating optimization process that interleaves minimization of the deformation energy w.r.t. the image region interior and the mapping between the boundary curves, thus automatically determining the boundary correspondence and deforming the image at the same time. Thanks to the particular design of the deformation energy, its gradient can be efficiently computed, making the entire image editing framework interactive and convenient for practical use.

Figure 1: The fish is deformed by applying a single sketch and seamlessly placed in an underwater scene.

1 Introduction

Gradient domain approaches have been successfully used for image manipulation. Preserving the gradients of the image *domain* leads to image deformation techniques [2, 5, 6, 10], which allow rotating and possibly scaling the image, but minimize deformation. Preserving the gradients of the image *function* (i.e., the colors) can be used to seamlessly paste a region of an image into another image, a technique called Poisson image editing [9]. Choosing the right region to be copied and pasted is not always easy, and several ways of using optimization to identify the region in the source and target images have been proposed [1, 7, 8].

Our idea is to allow deformation of the image region in this process. This would be an important improvement for several of the automatic approaches, as the source and target boundary curves could be different in shape. The shape can also be controlled by the user, who would simply sketch the boundary curves of the image part to be copied in the source image and the corresponding region in the target image. This simple and intuitive user interface is what we advocate in this work.

Starting with the two boundary curves, we try to minimize the deformation of the corresponding image domain. For this purpose, we define a rotation-invariant deformation energy (Section 3), essentially following the ideas of the gradient domain image deformation techniques mentioned above. Note that we are not assuming that the mapping between the two boundary curves is known. Instead, minimizing the deformation energy determines both, the mapping of the boundary curves onto each other, and the mapping of the region enclosed by the

boundary curves. We solve this minimization problem by interleaving the optimization of the interior region and optimization of the boundary curve mapping. In Section 4 we explain how this can be done by computing the gradient of the energy function with respect to the boundary vertices and then projecting these gradients onto the tangents of the boundary curve.

Using a least squares approach for enforcing the mapping of the boundary curves allows varying the degree of precision with which the deformation follows the boundary conditions, to enable gesturing of quick and crude sketches, as well as carefully drawn, precise curves. In the results section we show how this leads to a very easy to control image editing tool.

2 Framework

The user selects a region of the source image by drawing a simple closed curve. We represent this image region by a quad mesh $\mathcal{S} = \{V, E\}$ of a certain user-defined resolution, typically coarser than the original pixel grid in order to save computation time, similarly to [5, 10]. We will denote the original vertex positions of $\mathcal{S}$ by $\mathbf{v} \in \mathbb{R}^{2n}$, a $2n$-vector containing the coordinates of the n vertices (the n x-coordinates followed by the n y-coordinates). We refer to the position of the i-th vertex as $\mathbf{v}_i \in \mathbb{R}^2$ and the set of neighbors of vertex i is $\mathcal{N}(i)$.

The user then draws another simple closed curve into the target image to indicate placement and deformation of the selected image region. Depending on the input device, the sampling rate, and the expertise of the user, it might be necessary to filter out some noise in the sketched curve. After connecting the pixel midpoints with line segments, we apply the Douglas-Peucker polyline simplification algorithm [4] using a small ϵ. The result of this procedure is a piecewise linear target boundary curve, which we denote as γ.

The goal is to find displaced vertex positions $\mathbf{v'} \in \mathbb{R}^{2n}$ of the quad mesh that

1. minimize the distance of boundary vertices to γ and
2. minimize a deformation energy, which is invariant to rotation and isotropic scale.

In the following section we will briefly explain how we define the energy based on the results of gradient domain mesh deformation techniques [11].

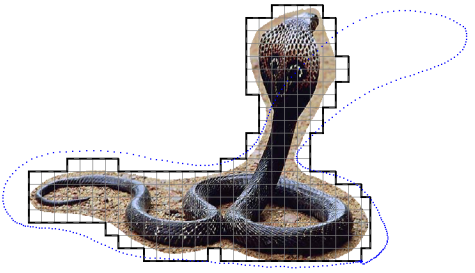


Figure 2: The user deforms an image region simply by drawing the desired new boundary curve γ, here shown as blue dots. The image region is represented by a quad mesh $\mathcal{S}$, in this case using a very coarse resolution (each quad is 20×20 pixels).

3 Deformation energy

It is common to measure deformation as a function of first and second derivatives of the mapping [3, 12]. The Laplacian of a mesh can be represented per vertex as

$$\boldsymbol{\delta}_i = \mathbf{L}(\mathbf{v}_i) = \mathbf{v}_i - (1/|\mathcal{N}(i)|) \sum_{j \in \mathcal{N}(i)} \mathbf{v}_j. \quad (1)$$

The mapping $\mathbf{v} \rightarrow \mathbf{v'}$ is a pure translation if all $\boldsymbol{\delta}_i$ remain unchanged. It is an isotropic scaling if the $\boldsymbol{\delta}_i$ are all scaled by the same factor. And it is a rotation if all $\boldsymbol{\delta}_i$ are rotated by the same angle. To measure deformation, we estimate isotropic scale and rotation locally (i.e., a similarity transformation), and compare the original $\boldsymbol{\delta}_i$ to the deformed ones.

A similarity transformation in 2D has the form

$$\mathbf{T}_i = \mathbf{T}_i(s, w) = \begin{pmatrix} s & w \\ -w & s \end{pmatrix}.$$

It can be estimated locally by considering each vertex and its neighbors, and then fitting the best similarity transformation in the least squares sense

$$\mathbf{T}_i(\mathbf{v'}) = \arg\min_{s,w} \sum_{j \in \{i\} \cup \mathcal{N}(i)} \|\mathbf{T}_i \mathbf{v}_j - \mathbf{v'}_j\|^2.$$

Note that $\mathbf{T}_i$ is a linear function in the $\mathbf{v'}$.

We use these local similarity transformations to transform the original Laplacians $\boldsymbol{\delta}_i$ and then compare them to the ones of the deformed mesh. The

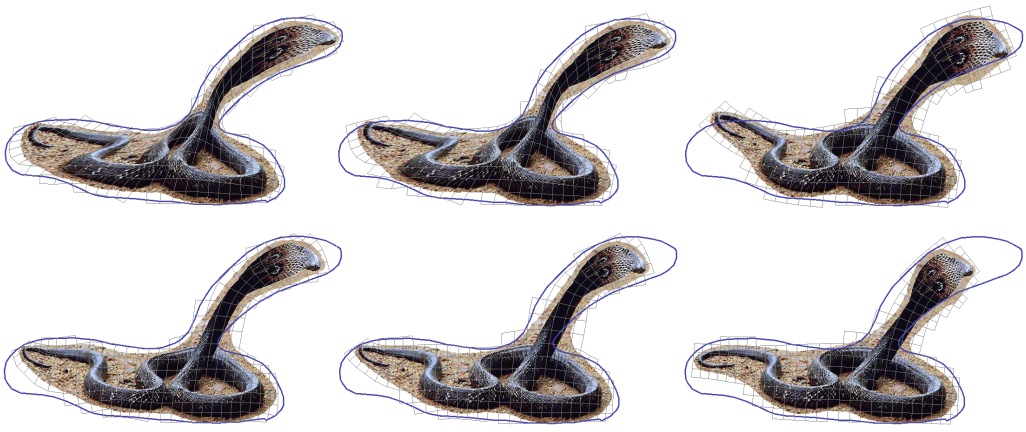


Figure 3: Comparison of deformation results using different handle weights ω (from left to right: $\omega = 1.0$, $\omega = 0.5$, $\omega = 0.1$). The top row shows results of the initial deformation while the bottom row shows the same results after applying our deformation minimization algorithm. The undeformed image region is shown in Figure 2.

squared difference of these vectors will be the deformation energy:

$$E_d(\mathbf{v}') = \sum_{i=1}^{n} \|\mathbf{T}_i(\mathbf{v}')\boldsymbol{\delta}_i - \mathbf{L}(\mathbf{v}'_i)\|^2 \qquad (2)$$

This is a quadratic expression in $\mathbf{v}'$. So it can be written as

$$\begin{aligned} E_d(\mathbf{v}') &= \|\mathbf{A}_d\mathbf{v}'\|^2 \\ &= \langle \mathbf{A}_d\mathbf{v}', \mathbf{A}_d\mathbf{v}' \rangle \\ &= \mathbf{v}'^{\mathsf{T}}\mathbf{A}_d^{\mathsf{T}}\mathbf{A}_d\mathbf{v}'. \end{aligned} \qquad (3)$$

It could be minimized by setting its partial derivatives to zero, leading to the equation $\mathbf{A}_d\mathbf{v}' = 0$. This means $\mathbf{v}' = 0$ is among the minimum energy states of $E_d(\mathbf{v}')$. The additional constraint that the boundary vertices of the mesh have to be close to the curve γ will yield non-zero solutions to the energy minimization.

4 Constrained energy minimization

We wish to find a deformation that minimizes the deformation energy so that all boundary vertices of the image region are close to the sketched curve γ. Assume we knew the target positions $\mathbf{h} \in \mathbb{R}^{2c}$ for the boundary vertices (again, $\mathbf{h}$ consists of the c concatenated x-coordinates and then the y-coordinates, and we denote the positions as $\mathbf{h}_i, i \in \mathcal{C}$). Then we could define an additional quadratic energy term

$$\begin{aligned} E_b(\mathbf{v}', \mathbf{h}) &= \sum_{i \in \mathcal{C}} \|\mathbf{v}'_i - \mathbf{h}_i\|^2 = \|\mathbf{A}_h\mathbf{v}' - \mathbf{h}\|^2 \\ &= \mathbf{v}'^{\mathsf{T}}\mathbf{A}_h^{\mathsf{T}}\mathbf{A}_h\mathbf{v}' - 2\,\mathbf{v}'^{\mathsf{T}}\mathbf{A}_h^{\mathsf{T}}\mathbf{h} + \mathbf{h}^{\mathsf{T}}\mathbf{h}, \end{aligned}$$

where $\mathbf{A}_h \in \mathbb{R}^{2c \times 2n}$ is a matrix that extracts the vector of constrained vertices out of $\mathbf{v}'$. Assume w.l.o.g. that $\mathcal{C} = \{1, \ldots, c\}$, then $\mathbf{A}_h$ has the simple form

$$\mathbf{A}_h = \begin{pmatrix} \mathbf{I}_{c\times c} & \mathbf{0}_{c\times(n-c)} & \mathbf{0}_{c\times n} \\ \mathbf{0}_{c\times n} & \mathbf{I}_{c\times c} & \mathbf{0}_{c\times(n-c)} \end{pmatrix}. \qquad (4)$$

Any weighted combination of the two energy terms

$$E_c(\mathbf{v}', \mathbf{h}) = E_d(\mathbf{v}') + \omega^2 E_b(\mathbf{v}', \mathbf{h}) \qquad (5)$$

is again a quadratic energy in $\mathbf{v}'$, i.e.

$$\begin{aligned} E_c(\mathbf{v}', \mathbf{h}) = \; & \mathbf{v}'^{\mathsf{T}}\left(\mathbf{A}_d^{\mathsf{T}}\mathbf{A}_d + \omega^2\mathbf{A}_h^{\mathsf{T}}\mathbf{A}_h\right)\mathbf{v}' - \\ & - 2\,\mathbf{v}'^{\mathsf{T}}\omega^2\mathbf{A}_h^{\mathsf{T}}\mathbf{h} + \omega^2\mathbf{h}^{\mathsf{T}}\mathbf{h}. \end{aligned} \qquad (6)$$

Now the resulting linear system defining the minimum energy state is nonhomogeneous, resulting in

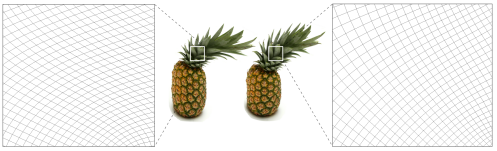


Figure 4: Comparison of an unoptimized (left) with an optimized mesh (right). The unoptimized mesh shows strong shearing artifacts which results in distortions of the image. After applying our energy minimization algorithm the resulting mesh is much more regular and thus visual distortions in the image are minimized. See Figure 6 for a higher resolution image.

a unique solution when more than two vertices are constrained and $\omega > 0$.

Of course, we don't have target positions $\mathbf{h}$ for the handle vertices a priori. All we know is that they are supposed to be on γ. So our approach to minimizing the energy is to compute target positions on the curve γ that minimize $E_d(\mathbf{v}')$, i.e., the positions are defined as

$$\arg\min_{\mathbf{h}\in\gamma} E_d(\mathbf{v}'). \tag{7}$$

The minimization is done iteratively, using a gradient descent approach. We first compute a rough initial mapping of the boundary vertices onto the user-sketch, i.e., matching the source boundary and the target sketch based on arc-lengths. We then take a small step into the negative direction of the gradients of the deformation energy with respect to the boundary vertices in order to find new handle positions resulting in smaller deformation energy and thus less distortion of the mesh. Since the handle positions are constrained to lie on the user-sketched boundary, we reproject the new handle positions onto the sketch and iterate the algorithm until a minimum is reached.

The deformation resulting from the rough arc-lengths based mapping is what we call "initial deformation" and compare our optimized results against.

4.1 Gradient on the boundary

Knowing the positions of the boundary vertices $\mathbf{h}$, we can compute $\mathbf{v}'$ as the minimum energy state of $E_c(\mathbf{v}', \mathbf{h})$ by taking the gradient of the energy w.r.t. $\mathbf{v}'$. From Eqn. (6) follows:

$$\frac{\partial E_c}{\partial \mathbf{v}'} = 2(\mathbf{A}_d^\mathsf{T}\mathbf{A}_d + \omega^2\mathbf{A}_h^\mathsf{T}\mathbf{A}_h)\mathbf{v}' - 2\omega^2\mathbf{A}_h^\mathsf{T}\mathbf{h}.$$

Setting the gradient to zero, we obtain

$$\mathbf{v}'(\mathbf{h}) = \omega^2(\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)^{-1}\mathbf{A}_h^\mathsf{T}\mathbf{h}, \tag{8}$$

where $\mathbf{A}_c \in \mathbb{R}^{(2n+2c)\times(2n)}$ is the combined rectangular matrix

$$\mathbf{A}_c = \begin{pmatrix} \mathbf{A}_d \\ \omega\mathbf{A}_h \end{pmatrix},$$

such that $\mathbf{A}_d^\mathsf{T}\mathbf{A}_d + \omega^2\mathbf{A}_h^\mathsf{T}\mathbf{A}_h = \mathbf{A}_c^\mathsf{T}\mathbf{A}_c$.

We proceed to compute the gradient $\mathbf{g}$ of the deformation energy functional with respect to the handles $\mathbf{h}$: $\mathbf{g} = \nabla E_d(\mathbf{v}'(\mathbf{h}))$. According to the chain rule,

$$\mathbf{g} = \nabla E_d(\mathbf{v}'(\mathbf{h})) = \left(\frac{\partial \mathbf{v}'}{\partial \mathbf{h}}\right)^\mathsf{T} \nabla E_d(\mathbf{v}').$$

Recall that the Jacobian matrix $\left(\frac{\partial \mathbf{v}'}{\partial \mathbf{h}}\right)$ is the matrix of partial derivatives:

$$\left(\frac{\partial \mathbf{v}'}{\partial \mathbf{h}}\right) = \begin{pmatrix} \frac{\partial v_1'}{\partial h_1} & \cdots & \frac{\partial v_1'}{\partial h_{2c}} \\ \vdots & \ddots & \vdots \\ \frac{\partial v_{2n}'}{\partial h_1} & \cdots & \frac{\partial v_{2n}'}{\partial h_{2c}} \end{pmatrix}.$$

Direct computation of the Jacobian is inefficient: from Eqn. (8) one can see that it requires explicitly having the inverse matrix $(\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)^{-1}$, which is expensive and would necessitate prohibitive storage, because the inverse matrix is generally not sparse. We describe how we circumvent this problem in Section 4.3.

4.2 Efficient gradient computation

Since the gradient descent algorithm requires several iterations until a minimum is reached, we are interested in computing $\mathbf{g}$ as efficiently as possible. The first part needed for the computation of $\mathbf{g}$ is $\nabla E_d(\mathbf{v}')$:

$$\nabla E_d(\mathbf{v}') = 2\left(\mathbf{A}_d^\mathsf{T}\mathbf{A}_d\right)\mathbf{v}'. \tag{9}$$

We can use the fact that $\mathbf{v}'$ is the solution of Laplacian mesh editing, which can be accelerated by computing the Cholesky factorization of the system

matrix $\mathbf{A}_c^\mathsf{T}\mathbf{A}_c$. Since this matrix does not change during our iterations, we can reuse this factorization; computing $\mathbf{v'}$ for a different right hand side then only requires a (fast) backsubstitution step.

4.3 Efficient Jacobian computation

From Eqn. (8) we can directly obtain the Jacobian of $\mathbf{v'}$, since $\mathbf{v'}$ is a linear function of $\mathbf{h}$:

$$\frac{\partial \mathbf{v'}}{\partial \mathbf{h}} = \omega^2 (\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)^{-1}\mathbf{A}_h^\mathsf{T}.$$

According to the structure of $\mathbf{A}_h$ (Eqn. (4)), it is easy to see that the Jacobian simply consists of the relevant columns of $(\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)^{-1}$. Let us denote the individual columns of $(\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)^{-1}$ by $\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_{2n}$. Then

$$\left(\frac{\partial \mathbf{v'}}{\partial \mathbf{h}}\right) = \omega^2(\mathbf{m}_1, \cdots, \mathbf{m}_c, \mathbf{m}_{n+1}, \cdots, \mathbf{m}_{n+c}).$$

To avoid explicitly computing the inverse matrix and then taking some of its columns, we observe what happens if we actually use *all* the columns. By "pretending" that the Jacobian actually equals $\omega^2(\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)^{-1}$, we obtain the following "overcomplete gradient":

$$\begin{aligned} \tilde{\mathbf{g}} &= (\omega^2(\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)^{-1})^\mathsf{T}(2(\mathbf{A}_d^\mathsf{T}\mathbf{A}_d)\mathbf{v'}) \quad (10) \\ &= 2\omega^2(\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)^{-1}\mathbf{A}_d^\mathsf{T}\mathbf{A}_d\mathbf{v'}. \quad (11) \end{aligned}$$

(Here we used the fact that since $\mathbf{A}_c^\mathsf{T}\mathbf{A}_c$ is symmetric, its inverse is also symmetric.) We can easily obtain $\tilde{\mathbf{g}}$ as the solution of the following linear system:

$$(\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)\tilde{\mathbf{g}} = 2\omega^2(\mathbf{A}_d^\mathsf{T}\mathbf{A}_d)\mathbf{v'}.$$

This system is quite efficient to solve: it only requires one more back-substitution, since the prefactorization of the system matrix $\mathbf{A}_c^\mathsf{T}\mathbf{A}_c$ is already given.

Finally, we observe that since the real gradient $\mathbf{g}$ only has $2c$ rows, and in order to compute it we need the $2c$ columns of the inverse matrix, we can actually obtain $\mathbf{g}$ from $\tilde{\mathbf{g}}$ simply by erasing the unnecessary rows and only keeping the rows $1, \ldots, c$, $n+1, \ldots, n+c$.

4.4 Reprojecting gradients

To minimize deformation energy we apply the gradient descent algorithm. Given the gradient $\mathbf{g}$ of the deformation energy with respect to the handle vertices $\mathbf{h}$, we compute new handle positions $\mathbf{h'}$ by taking a small step into the opposite direction of the gradient of $\mathbf{h}$:

$$\mathbf{h'} = \mathbf{h} - \lambda\mathbf{g}.$$

Recall that we require the boundary vertices $\mathbf{h}$ to lie on the user-sketched curve γ in order for the deformed shape $\mathcal{S}'$ to follow the user-sketch. The new handle positions $\mathbf{h'}$ that result from an optimization iteration generally do not lie on γ anymore. To fix this, we reproject the handles $\mathbf{h'}$ onto γ.

We have implemented the reprojection of $\mathbf{h'}$ onto γ as a simple orthogonal projection; i.e., for each handle vertex $\mathbf{v}'_c, c \in \mathcal{C}$, we find the nearest point on γ. We denote the vector of reprojected handle vertices by $\mathbf{h}'_{\mathbf{proj}}$. The reprojected handle $\mathbf{h}'_{\mathbf{proj}}$ is then used to compute the new energy gradient, and the algorithm is iterated until convergence. We found that approximately 50 iterations combined with a stepsize of $\lambda = 0.5$ yielded good visual results in all cases and thus used those values in all our examples .

5 Deforming and cloning

To finally compute the deformed image from the deformed quad mesh we apply texture mapping with bilinear interpolation, which can be quickly performed by graphics hardware. Lastly, the resulting image is composited onto the target background image with Poisson cloning as illustrated in Figure 1. For completeness, in the following we briefly describe the compositing algorithm.

The final image is a result of an optimization process that minimizes the squared difference between the gradients of the unknown region in the target image and the given gradients of the pasted image for target boundary conditions. This results in a solution that has gradients similar to the original image while the boundary constraints ensure that the resulting colors match the given boundary colors.

The optimization amounts to solving the Poisson equation on the pasted image domain, to reconstruct the three color channels. Using sparse linear solver libraries, e.g. TAUCS [13], we can do this very efficiently by factoring the Laplace matrix once and solving by backsubstitution. This even allows interactively moving the pasted region around the target image, since only the right-hand side of the system

changes, and thus the cloning can be recomputed very quickly.

6 Results

We have successfully applied the proposed method to various images and show comparison results between original, initially deformed and optimized images in Figure 6 and Figure 7. As can be seen in the resulting optimized images, visual distortions in the deformed image are minimized by our approach, while still following the user's sketch.

In Figure 8 we show the deformation energy gradients with respect to the handle vertices. The length of the gradient vectors is exaggerated by a factor of 10 for illustration purposes. Note that in the initially deformed mesh, the gradient vectors have large components tangential to the boundary curve, indicating that mesh deformation could be reduced by moving vertices along the boundary. As the minimization procedure progresses, tangential components become smaller and the procedure converges in all gradients being orthogonal to the sketch. Figure 4 shows another example of an optimized mesh. Applying our energy minimization algorithm removes the strong shearing artifacts clearly noticeable in the initially deformed mesh. The result is a mesh with a structure very close to the original mesh, leading to smaller visual distortion in the deformed image.

The performance of our approach depends on the size of the involved linear system, i.e. on the size of the underlying mesh that is to be deformed. See Table 1 for a comparison of execution time against various mesh sizes, taken from the example in Figure 5. While using coarse meshes provides interactive feedback within fractions of a second, higher resolutions take significantly longer to compute but result in less distortions in the final deformed image. Note that only the minimization operation (last column in Table 1) has to be executed when redrawing a sketch, all other parts are precomputed.

We show a comparison of the influence of varying handle weights on the deformation results in Figure 3. Note that exactly following the user-sketch using higher handle weights results in stronger deformations in the initial deformation result. Our proposed deformation minimization removes those distortions while still following the user-drawn outline.

vertices	init	fact.	∇E_d	min.
300	0.047	0.234	0.172	0.344
916	0.141	0.765	1.141	1.078
3491	0.438	2.562	4.204	4.39

Table 1: Performance data measured in seconds on an Intel Core 2 Duo 2.4 Ghz with 2 GBytes RAM. From left to right: Number of vertices in the underlying quad mesh, time to create the system matrix $\mathbf{A}_c$, time needed to compute a sparse factorization of $\mathbf{A}_c^\mathsf{T}\mathbf{A}_c$, time needed for computing the $\mathbf{A}_d^\mathsf{T}\mathbf{A}_d$ part of the gradient of the deformation energy and time needed to execute 50 iterations of our minimization algorithm in order to compute the optimized image.

7 Discussion

We have presented a fast and robust optimization technique that implements sketch based image deformation resulting in high quality deformation results without visible distortions.

Our technique is interactive up to medium mesh resolutions and provides quick feedback when redrawing a boundary since the most expensive computations (factoring the system matrix and computing $\mathbf{A}^\mathsf{T}\mathbf{A}$) only have to be done once. It is also easy to implement since computing the gradient with respect to the handle vertices, as shown in Eqn. (11), reduces to simply solving two linear systems.

The constrained energy minimum is usually not unique – depending on the initial boundary matching the gradient descent procedure converges to the closest local minimum. Also, as the deformation energy is invariant under rotations, near circular boundaries could lead to ambivalent minimum energy states. In practice, we have not observed this problem. In any case, the user could add information on corresponding points on the boundary to resolve ambiguities.

Our current implementation requires the user to resketch the whole boundary for each modification, which can be cumbersome. Future work may thus include boundary manipulation techniques, allowing the user to edit small parts of the boundary instead of doing a complete resketch. This may be achieved by using intuitive curve editing interfaces [6].

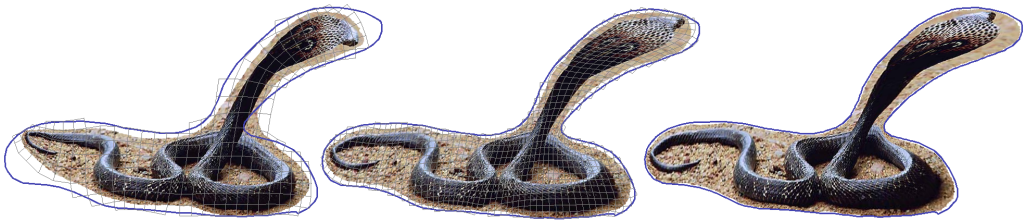
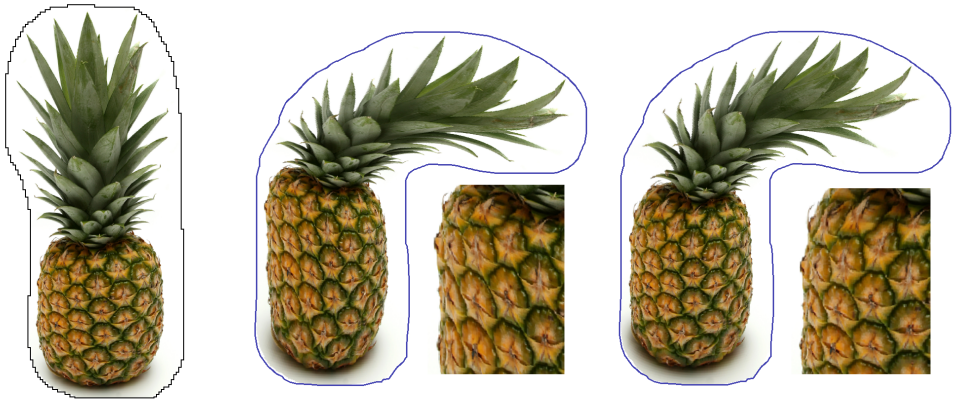
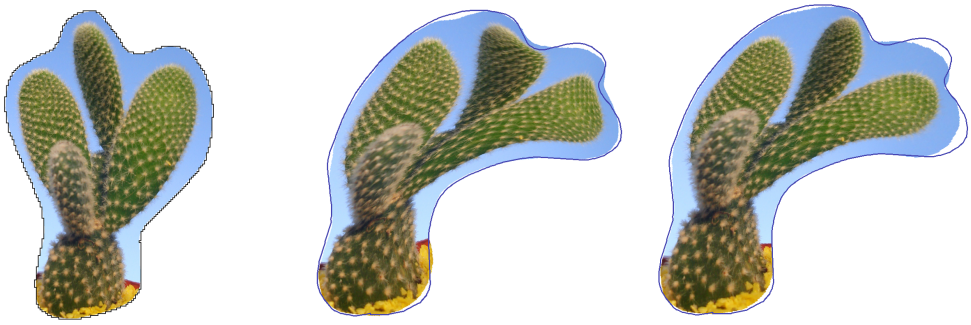


Figure 5: Deformation results using various mesh resolutions. From left to right: Using quad sizes of 20×20, 10×10 and 5×5 pixels, with constant handle weights $\omega = 1.0$.



(a) Original (b) Initial Deformation (c) Energy Minimized

Figure 6: Bending a pineapple. The result of the initial deformation (b) shows strong distortions. After minimizing the deformation energy, visible distortions are eliminated (c).



(a) Original (b) Initial Deformation (c) Energy Minimized

Figure 7: Bending the top of a cactus to the right. Again, the original, undeformed image is shown in (a), the initially deformed image in (b) and the optimized image after applying 50 iterations of our deformation minimization algorithm is shown in (c).

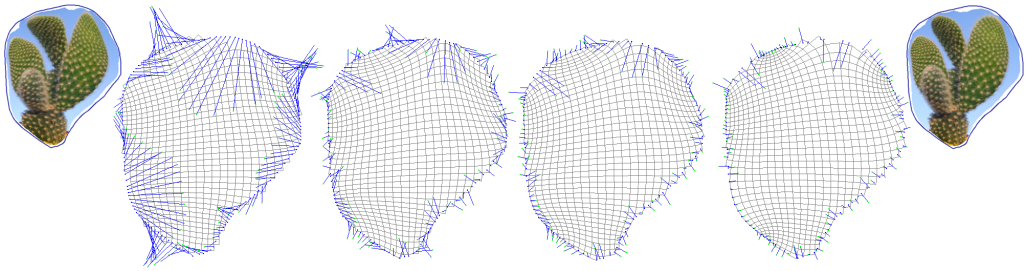


Figure 8: Development of a deformed mesh and its associated gradients of the handle vertices (blue lines) during the optimization process. The image on the top left corresponds to the leftmost, unoptimized mesh while the image on the top right corresponds to the optimized mesh on the right after applying 50 iterations of our optimization algorithm. Gradient vectors are scaled by a factor of 10 for illustration purposes.

Acknowledgements

We wish to thank Andrew Nealen for fruitful discussions and the anonymous reviewers for their valuable comments. This work was supported in part by the Alexander von Humboldt Foundation.

References

[1] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. *ACM Trans. Graph.*, pages 294–302, 2004.

[2] Marc Alexa, Daniel Cohen-Or, and David Levin. As-rigid-as-possible shape interpolation. In *Proceedings of ACM SIGGRAPH*, pages 157–164, 2000.

[3] Mario Botsch and Olga Sorkine. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*, 2007. To appear.

[4] D.H. Douglas and T.K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[5] Ran Gal, Olga Sorkine, and Daniel Cohen-Or. Feature-aware texturing. In *Proceedings of Eurographics Symposium on Rendering*, pages 297–303, 2006.

[6] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. *ACM Trans. Graph.*, 24(3):1134–1141, 2005.

[7] J. Jia, J. Sun, C.K. Tang, and H.Y. Shum. Drag-and-drop pasting. *ACM Trans. Graph.*, 25(3):631–637, 2006.

[8] Yin Li, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. Lazy snapping. *ACM Trans. Graph.*, pages 303–308, 2004.

[9] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Trans. Graph.*, 22(3):313–318, 2003.

[10] S. Schaefer, T. McPhail, and J. Warren. Image deformation using moving least squares. *ACM Trans. Graph.*, 25(3):533–540, 2006.

[11] Olga Sorkine, Yaron Lipman, Daniel Cohen-Or, Marc Alexa, Christian Rössl, and Hans-Peter Seidel. Laplacian surface editing. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 179–188. ACM Press, 2004.

[12] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *Proceedings of ACM SIGGRAPH*, pages 205–214, 1987.

[13] Sivan Toledo. TAUCS*: A Library of Sparse Linear Solvers, version 2.2*. Tel-Aviv University, Available online at `http://www.tau.ac.il/~stoledo/taucs/`, September 2003.

Non-iterative Camera Calibration Procedure Using A Virtual Camera

Tobias Elbrandt, Ralf Dragon, Jörn Ostermann

Institut für Informationsverarbeitung
Leibniz Universität Hannover, Appelstr. 9A, 30167 Hannover, Germany
{elbrandt/dragon/ostermann}@tnt.uni-hannover.de

Abstract

This article presents a method to improve camera calibration by separating determination of the non-linear calibration parameters from that of the linear ones. We measure correspondences between the distorted image on a camera target and a flexible undistorted calibration pattern displayed on a TFT monitor. Using these correspondences the image of the camera may be projected onto the plane of the calibration pattern to remove the distortion. We prove that this projection follows the principles of a pinhole camera and call it virtual camera. Using the undistorted images of the virtual camera for traditional camera calibration methods, linear camera parameters can be calculated with a 7.5% to 39.5% higher precision compared to Zhang/Heikkilä. The described procedure allows a camera calibration process in a non-iterative way.

1 Introduction

Every time a camera is used for geometric measurement, which is determination of the position or size of an object or its distance to another object, it is necessary to know the properties of the optical system of the camera.

Determining these properties is a classic research topic in computer vision [5][10][12] so the reader is refered to established literature (e.g. [4], chapter 6) for description of the parameters and how to work with them. The main parameters needed for measurement are the linear extrinsic and intrinsic camera parameters. Extrinsic parameters describe the position $\bar{C}_c$ (translation to the origin of the coordinate system[1]) and the orientation $\bar{R}_c$ (rotation) of the camera. Linear intrinsic camera parameters describe the linear properties of the optical system within the camera – i.e. the focal length f, aspect ratio of the pixels s_x/s_y, shear r, and displacement of the principal point (h_x, h_y). The linear parameters may be conveniently combined into the so called camera matrix M_c:

$$M_c = K_c R_c \tag{1}$$

with

$$R_c = \begin{bmatrix} \bar{R}_c & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{1} & -\bar{C}_c \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \bar{R}_c & -\bar{R}_c\bar{C}_c \\ \mathbf{0} & 1 \end{bmatrix} \tag{2}$$

$$K_c = \begin{bmatrix} fs_x & r & h_x & 0 \\ 0 & fs_y & h_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3}$$

Non-linear parameters describing radial, tangential, or prism distortion are also considered as intrinsic parameters (see [13] for their description).

Most calibration methods for single cameras (Tsai [10], direct linear transformation [4], Zhang [12], Heikkilä [5]) derive the parameters of the camera model on the basis of corresponding points in the world coordinate system and on the camera target. For this procedure, they need one or more camera images of a planar calibration pattern or calibration object with marks on the surface with known size or distance to each other. For each visible mark the position on the camera target is determined giving a list of correspondences between points P_i in world coordinates and points p_i in camera target coordinates, i.e. pixel. By using Equation (4) of the mapping of a pinhole camera, these correspondences are also modeled by algebraic means. The projective linear factor λ is calculated such that the third component of p equals 1.

$$p = \lambda M_c P \tag{4}$$

[1] Cartesian coordinates are marked here with a crossbar over the letter whereas there is no additional tag for homogeneous coordinates.

Traditional calibration methods determine the coefficients of the camera matrix M_c minimizing the sum of the errors d done by mapping the known world points to the measured camera points:

$$\hat{M}_c = \underset{M_c}{\operatorname{argmin}} \sum_i d(\lambda M_c P_i, p_i) \tag{5}$$

Prerequisite for this approach is that the correspondences between points in the world and the target can be described by the pinhole camera model. The image of a scene taken by a pinhole camera is *undistorted*. A real camera uses a system of different lenses and an aperture within the course of the light rays to bundle the light projecting a scene onto the camera target, so the image of the scene becomes *distorted*. The distortion by the optical system is represented here shortly with the function $l : (p_c \rightarrow p)$, $p_c, p \in (\mathbb{R}, \mathbb{R}, 1)^T$. With this function, mapping a point onto the camera target may be written in contrast to Equation (4) as:

$$p = l(\lambda M_c P) \tag{6}$$

To be able to determine the camera parameters although the mapping is distorted, many authors typically describe the distortion l using a non-linear mathematical model. The radial distortion describes the major part of the distortion. It is estimated by most of the established calibration methods like [12]. Some approaches additionally model tangential distortion [5] and prism distortion [13]. By naming the distortion model l_θ with the respective parameter set θ, the camera parameters are again determined by minimizing the sum of errors:

$$(\hat{M}_c, \hat{\theta}) = \underset{M_c, \theta}{\operatorname{argmin}} \sum_i d(l_\theta(\lambda M_c P_i), p_i) \tag{7}$$

Multi camera calibration methods work quite similar but they use corresponding points on camera images from different views. Silhouettes [8] or corresponding feature points [9], on the recorded scene may be used.

An analytical solution of Equation (7) cannot be calculated in general. For this reason, an initial set of distortion parameters is presumed to be able to estimate the camera matrix M_c. Then the distortion parameters are adjusted to minimize the projection error. Afterwards, the camera matrix is estimated again. This is done iteratively until the projection error is small enough or there are no improvements by adjusting the distortion parameters.

The basic premise for this approach is that the distortion of a real lens system can be completely described using algebraic formulas and that their degrees of freedom can be calculated. Both assumptions are usually not valid: The lenses are different to each other and normally have unknown characteristics. There are manufacturing tolerances in the fabrication process of both the lenses and the lens system; the lenses are not perfectly symmetrical, they are not exactly concentric and orthogonal to the optical axis, etc. In addition, the global optimum of Equation (7) cannot be calculated.

In this article we compensate the non-linear distortion l of the optical system prior to the camera calibration. The distortion is measured by associating every point of the camera target with a corresponding point on a known image plane. If it is known for every point of the camera target which point of the image plane it is the mapping of, then every camera image can be undistorted by reprojecting it onto that image plane.

A similar approach was presented in [7] where correspondences between camera target and image plane were measured using structured light shown on a plasma display panel and then used to undistort camera images. In contrast to that paper, we quantify exactly the projection error at each point of the camera target, whereas the other paper justs fits in a line for visual control of its results. Also, we proceed using the undistorted images for camera calibration.

In photogrammetry, distortions are sometimes calculated in a comparable manner using a réseau technique [2]. A small list of correspondences is build by measuring the position of grid points projected onto a film at exposure. The undistorted positions for the points of the image are then interpolated using these measured control points. Due to the control points being too sparsely distributed, this approach lacks accuracy.

We prove algebraically that by undistorting camera images with the presented method the real camera target is substituted by a *virtual target* with known measurements and without distortions. The usual iterative two-step camera calibration procedure is substituted by a three-step non-iterative procedure. The first step of the procedure results in an undistorted image of the scene. A second step ad-

justs the virtual target to an upright position. This step can be skipped if the undistorted images are used for photogrammetry only. After both steps, there is a complete mapping from the real camera to a virtual camera which is without distortions and which has an optical axis identical to that of the real camera.

In a third step, the images undistorted with this mapping are used for traditional camera calibration, skipping the estimation of the non-linear parameters, to obtain the linear camera parameters of the virtual camera. The big advantage is that the undistorted image of a calibration pattern follows the linear pinhole camera model, so the calibration procedure is much more stable and precise since estimation of the non-linear parameters can be skipped.

The following Section 2 describes the compensation method in detail. Subsection 2.1 explains the first step, the measurement and compensation of the distortions, and Subsection 2.2 explains the second step, the adjustment to an upright position. The last Subsection 2.3 discusses practical considerations of the proposed method. In the following Section 3, measurements of the accuracy of the method and their results are shown. The last Section 4 summarizes the paper.

2 Measurement and compensation of non-linear distortion

We assume that it is practically impossible to completely describe the properties of the distortion of an optical system with a parametrized algebraic model. Therefore, we determine for every point p on the camera target the exact coordinates of the corresponding point P_V on a real image plane V whose image is p. Having corresponding points for all points on the camera target we are able to reproject the camera image onto the plane V.

Figure 1 clarifies this approach. The object space is projected onto the camera target through the focus. Due to the camera lens system, the projected image is distorted which is indicated by the non-rectangular camera target. The light ray that projects the world point P_W onto the point p on the camera target also goes through point P_V of the virtual target. As the image plane is undistorted by definition, the camera image becomes undistorted by reprojecting it onto V, the virtual target[2].

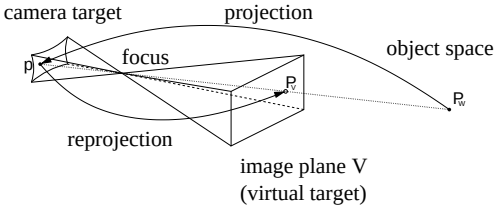


Figure 1: Mapping onto the virtual target

A set of points on the image plane is arranged on a rectangular grid. Within the coordinate system of the image plane, the position in homogeneous coordinates is $p_v = (u, v, 1)^T$. The image plane is at position $\bar{C}_I$ in world coordinates and is rotated in space with the rotation matrix $\bar{R}_I$. Both matrices may be combined to $R_I = \bar{R}_I \left[I | -\bar{C}_I \right]$. Naming the horizontal distance and the vertical distance between the points on the grid on the image plane with d_x and d_y, respectively, one can calculate the world coordinates P_V of the point p_v with $S_I(u, v, 1)^T = (d_x u, d_y v, 0, 1)^T$):

$$P_V = R_I S_I p_v \tag{8}$$

$$\text{with} \quad R_I = \begin{bmatrix} \bar{R}_I & -\bar{R}_I \bar{C}_I \\ \mathbf{0} & 1 \end{bmatrix}$$

$$\text{and} \quad S_I = \begin{bmatrix} d_x & 0 & 0 \\ 0 & d_y & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The function f maps all points $(u, v, 1)^T$ of the image plane onto the camera target:

$$p = l(\lambda_I M_c P_V) \tag{9}$$

$$= l(\lambda_I M_c R_I S_I (u, v, 1)^T)$$

$$= f((u, v, 1)^T) \tag{10}$$

By determining the correspondences Φ between points $(u, v)^T$ on the image plane and points $(x, y)^T$ on the camera target $f : (u, v, 1) \rightarrow (x, y, 1)$ can be ascertained. Thus, by building the inverse $f^{-1} : (x, y, 1) \rightarrow (u, v, 1)$ one gets the mapping (provided that l is bijective):

$$f^{-1}(p) = S_I^{-1} R_I^{-1} M_c^{-1} \lambda_I^{-1} l^{-1}(p) \tag{11}$$

[2]In the following text, the term *image plane* is used when talking about the source of the calibration points. The term *virtual target* is used when the target of the reprojection is meant. However, these two planes are the same.

By reprojecting the point P_W onto the virtual target one receives

$$\begin{aligned} & f^{-1}(l(\lambda_W M_c P_W)) \\ = \; & S_I^{-1} R_I^{-1} M_c^{-1} \lambda_I^{-1} l^{-1}(l(\lambda_W M_c P_W)) \\ = \; & \frac{\lambda_W}{\lambda_I} S_I^{-1} R_I^{-1} P_W \end{aligned} \tag{12}$$

Since S_I is not a square matrix, S_I^{-1} can be calculated using the pseudo inverse. Please note that the term $S_I^{-1} R_I^{-1}$ has size 3×4 like a homogeneous camera matrix; therefore it maps the point P_W onto the virtual target with the camera matrix $M_V = S_I^{-1} R_I^{-1}$ with $\lambda_V = \lambda_W / \lambda_I$. Hence,

$$p_v = \lambda_V M_V P_W \tag{13}$$

This is the reason why we call the reprojection onto the image plane a *virtual camera*. In contrast to Equation 6, this mapping is calculated without the distortion by function l.

If the image plane is orientated exactly orthogonal to the optical axis of the camera and the horizontal and vertical axes of the image plane and the camera target are parallel to each other, then the reprojected image is free of any distortion, no matter what physical origin it has. If the image plane was not adjusted exactly, then the reprojected image shows a certain slanted position. Subsection 2.2 describes how to compensate this slant.

2.1 Undistorted Virtual Camera

The first step of the compensation method is to determine the correspondences Φ between points $(u_i, v_i, 1)^T$ on the image plane V and points $(x_i, y_i, 1)^T$ on the camera target. The exact position and orientation of the plane in the world is relatively irrelevant for this step given that only points within the image plane are mapped onto the camera target and the camera image is sharp enough to determine the position of projected points on the camera target.

In order to determine the correspondences, points are displayed on the visible part of the image plane. The method used for that does not matter in general as far as the position is exact and well-known. A conventional TFT-monitor is used here (Subsection 2.3) because a single white pixel on a black background has the effect of a point light source. For every point $(u_i, v_i, 1)^T$, $u_i, v_i \in \mathbb{N}$ displayed on the image plane, which lies in the field of view of the camera, the position of its image on the camera target is determined with sub-pixel accuracy by calculating its centroid.

The list of correspondences $\Phi : (u_i, v_i, 1)^T \rightarrow (x_i, y_i, 1)^T$ of all observable points of the image plane describes the mapping function f with very fine granularity. We use f for two purposes:

- Determine the inverse function f^{-1} in order to convert camera target coordinates (2.1.1)
- Resample camera images onto the virtual target using a scanline-algorithm (2.1.2)

2.1.1 Building the Inverse Mapping Function

The points $(u_i, v_i, 1)^T$ on the image plane V build a dense grid at discrete equidistant coordinates, i.e. $u_i, v_i \subset \mathbb{N}$. The corresponding points $(x_i, y_i, 1)^T$ on the camera target are distributed densely, too, but not necessarily at equidistant locations, i.e. $x_i, y_i \subset \mathbb{Q}$. Inverting these correspondences directly would result in an inverse function $f^{-1} : \mathbb{Q}^2 \rightarrow \mathbb{N}^2$ of the optical mapping function f.

To be able to easily reproject coordinates measured on the camera target onto the virtual target, the inverse function f^{-1} should have a domain in $\mathbb{N}^2$ and its codomain in $\mathbb{Q}^2$. This means, for every point $(x_j, y_j, 1)^T$, $x_j, y_j \in \mathbb{N}$, on the camera target a point $(u_j, v_j, 1)^T$, $u_j, v_j \in \mathbb{Q}$ on the image plane has to be determined by interpolation.

This is done here with an image warping algorithm [11]. When calculating the inverse function it is assumed, that the mapping function is locally linear, which is reasonable as the points are very close to each other.

The resulting inverse function f^{-1} can then be used to convert camera target coordinates into undistorted coordinates of the virtual target.

2.1.2 Undistortion Using a Scanline Algorithm

The list of correspondences may also be used directly, i.e. without inverting, to undistort camera images.

It was determined for every point P_i of the visible image plane onto what coordinates p_i on the camera target it maps. Therefore, it is possible to reproject the camera image onto the image plane by taking the pixel value at position p_i for the point P_i on the image plane. As the coordinates of the points p_i are not discrete in general, the color/grey values have

to be interpolated, e.g. using a Sinc-function with a Blackman-Harris window[3]. The filters for the horizontal and vertical direction are separable, and their coefficients have to be calculated only once for a correspondence list. Therefore the calculation expense for the undistortion is within bounds for this method. However, a bilinear interpolation would be less expensive, but would result in images of lower quality.

2.2 Deskewed Undistorted Virtual Camera

If the image plane is not orthogonal to the optical axis of the real camera, the reprojection adds a slant to the image. To completely undistort reprojected images, this slant has to be compensated. As it can be seen in Figure 1, the real camera target and the virtual target have the same focus. This fact does not change if the virtual target is tilted. Two mappings that have the same focus are connected by a homography, i.e. one mapping can be transferred into the other. The mapping tilting the virtual target into the correct orientation is given by

$$p'_v = M'_V M_V^{-1} p_v \tag{14}$$

Each point p_v is reprojected into space using the inverse of the camera matrix M_V of the virtual camera to be projected onto the upright virtual camera using the camera matrix M'_V. The term $M'_V M_V^{-1}$ is denoted as homography.

The camera matrix M'_V is a short form for $K'_V R'_V$ and is unknown from the beginning. By using a traditional camera calibration method, the matrices K_V and R_V of the virtual camera are determined using undistorted camera images (see 2.1.2). The intrinsic camera matrix K'_V should have the same focal length and the same image resolution as the virtual target, but the principal point of the upright virtual target is moved to P', the center of the image.

To move the principal point to the image center, the optical axis of the virtual target has to be rotated. Figure 2 demonstrates this: On the left side, there is the real camera target with the optical axis A_c defined by R_c orthogonal on its center. The tilted virtual target is shown on the right side of the diagram. It is clear, that tilting the image plane may move the principal point outside of the actual image region. Now the principal point P is moved to

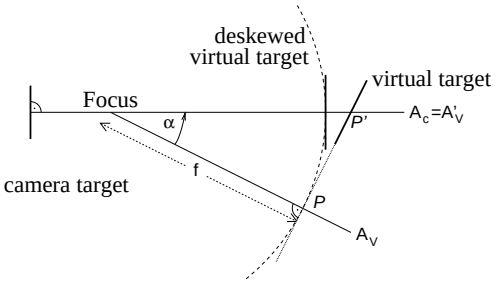


Figure 2: The virtual target is tilted into an upright position such that its principal point P moves to the center P' of the image.

the center P' of the image, rotating the optical axis of the virtual target to that of the real target. The angles α and β needed for that rotation can be calculated directly from the difference of P and P'. The optical axis A_V defined by R_V has to be rotated around these angles to obtain A'_V defined by R'_V, i.e. $R'_V = R_y(\beta)R_x(\alpha)R_V$ with $R_x(\alpha)$ and $R_y(\beta)$ being the rotation matrices around the x- and y- axis.

Finally, the homography $M'_V M_V^{-1}$ required to tilt the virtual target into an upright position according to Equation (14), is given by:

$$\begin{aligned} M'_V M_V^{-1} &= K'_V R'_V (K_V R_V)^{-1} \\ &= K'_V R_y(\beta) R_x(\alpha) R_V R_V^{-1} K_V^{-1} \\ &= K'_V R_y(\beta) R_x(\alpha) K_V^{-1} \end{aligned} \tag{15}$$

By applying this homography onto the camera matrix M_V of the virtual camera, the virtual camera gets an optical axis that is exactly orthogonal to the virtual target and which intersects it in its center. The deskewed virtual camera is completely described.

2.3 Practical Considerations

This subsection describes the technique to determine the correspondences required for the undistortion.

Firsts tests were done using a conventional TFT monitor. It is reasonable to assume that its pixels are equidistant in both directions, their distance can be measured and the horizontal and vertical axes are orthogonal to each other. The prerequisites for using it as a image plane are therefore given.

To measure the correspondences the camera is orientated towards the TFT-monitor such that the camera image only shows pixels of the monitor. To prevent reverberations from the border of the monitor, a certain distance to it has to be kept. A white pixel on the monitor illuminates some sensor elements $(\dot{x}_j, \dot{y}_j)$ of the camera target with the intensities $I(\dot{x}_j, \dot{y}_j)$. The center of gravity of the intensities is given by the sum of the positions of the elements each weighted with its intensity [1, Eq. 3.291]. For this method to work, the images of the points on the monitor must be relatively sharp.

To normalize the intensity values, a black and a white screen is shown and captured by the camera at the beginning. The brightness is adjusted to the best intensity range while preventing over- and underexposure. Next, the center, the direction of the axes, and the horizontal and vertical distances of the points on the camera target are determined by displaying three points in the center of the screen.

Now the actual measurement of correspondences is started. A pattern consisting of single points arranged uniformly with horizontal and vertical distances r_x and r_y, respectively, is shown full screen. The shorter the distance between the points, the more points are displayed at the same time and therefore, the less images have to be captured. On the other hand, the distance between the points has to be large enough such that the unsharp images of two pixels do not overlap. The tests for this article were done with a raster distance of $(r_x, r_y) = (32, 32)$.

Each point pattern is captured four times to reduce the effect of camera noise. Then the point pattern is displayed at a new position. After all $32 * 32 = 1024$ point patterns, the correspondence for every visible pixel on the monitor is measured; the correspondence list Φ is completely determined.

3 Experimental Results

In this section, results of our camera calibration method are presented and compared with a traditional calibration method. We used an IEEE1394 Prosilica EC 1380C single-CCD chip camera with a mounted Schneider-Kreuznach Cinegon 1.4/8-0512 industrial lens. The full-color RGB images used for the experiments were calculated using the high-quality AHD demosaicing algorithm [6]. The non-linear distortions were determined using a NEC MultiSync LCD1860NX TFT monitor.

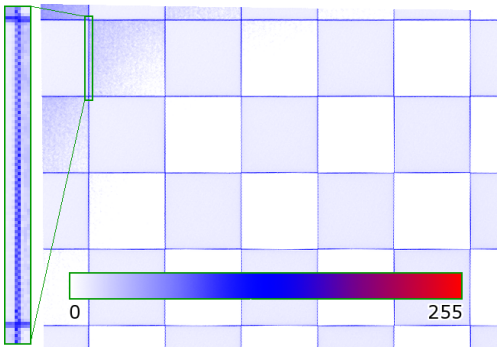


Figure 3: Upper left quarter of the difference between screen-shot and undistorted camera image of about 600×450 pixel size. White means no difference, blue medium difference and red means big difference. To be able to assess the colors an additional color wedge is added. On the left side, a zoom on the image shows that the line is perfectly undistorted and has constant width.

In the first step, the correspondence list was determined. Then, a checkerboard was displayed on the monitor without touching camera and monitor. The image captured by the camera was then undistorted using the scanline algorithm of Subsection 2.1.2. Now the difference between the undistorted camera image and the screen-shot was calculated and converted to a colored gradient to be able to assess the difference values. No difference is displayed as white, medium difference as blue and big difference as red. The upper left quarter of the resulting image is shown in Figure 3.

There are three main observations: First of all, one can recognize the checkerboard pattern. The reason for this is that the black of the screen-shot is really black (grey value 0) whereas the black as seen by the camera is only dark grey (grey value 10). The difference between black and dark grey can be observed. Secondly, the difference increases near the upper left corner. This can be traced to the fact that the pixel's luminosity on the TFT monitor lowers with a greater viewing angle. Both errors are due to an insufficient normalization of the intensity signal but have only a minor impact on the undistortion method.

The third observation is that one can clearly see the edges between the chessboards, due to a slightly

unsharp camera image. It is important that the edges are absolutely straight and have constant width. This means that the image was undistorted accurately. Otherwise, the width of the edges would differ.

A second test examines the accuracy of the undistortion more precisely. Camera images of the captured point patterns used for determining the correspondence list Φ are undistorted using the scanline algorithm from Subsection 2.1.2. These undistorted images are again used as input for determining a correspondence list. If the determination of the position of the points as well as the compensation of the optical mapping errors based on the first correspondence list was perfect, every point in the second correspondence list would be mapped to its original location.

Measuring the distributions of the reprojection errors in horizontal and vertical direction indicate that both are approximately Gaussian distributed. The standard deviation of the error in horizontal direction is $0.0344\,\mathrm{pixel}$ and in vertical direction $0.0305\,\mathrm{pixel}$. More than 99% of the errors are lower than $0.09\,\mathrm{pixel}$ and $0.078\,\mathrm{pixel}$, respectively.

The last experiment shows that by preprocessing camera images with the presented undistortion method, the accuracy of traditional camera calibration methods is substantially improved.

To estimate the intrinsic camera parameters of a traditional calibration method, we use the camera calibration tool of Jean-Yves Bouguet[3] which implements Zhang/Heikkilä [12][5]. Optionally, some non-linear distortion parameters, i.e. radial distortion of first to third order (κ_1-κ_3) and tangential distortion of first and second order (t_1, t_2) can be estimated. Pictures of a flat calibration pattern in 10 different positions and orientations are captured for this purpose.

The standard deviations of the parameters in Table 1 are given in pixel. In the first column the method used to undistort the camera images is listed. The second to fifth column give the standard deviations of the intrinsic camera parameters[4]. Finally, the reprojection error for the calibration points is given in the last two columns.

[3] Freely available under `http://www.vision.caltech.edu/bouguetj/calib_doc`.

[4] The used calibration toolbox outputs three times the parameters' standard deviation instead because it aims to show the uncertainties.

The first row shows the standard deviation of the linear intrinsic camera parameters without prior estimation of the non-linear distortion parameters. In the following five rows, the stabilities of the intrinsic parameters are listed if the non-linear distortion parameters are estimated using the toolbox. The second to last row shows the results of the estimated parameters if the camera images are first undistorted using our method and then are used for estimating the linear camera parameters. The last row lists the results when applying both the traditional and our method for camera calibration.

As can be seen from the standard deviations in the table, the accuracy of the estimation of the intrinsic linear camera parameters is substantially improved if the images are undistorted before with the presented method, while decreasing the reprojection error. Since different camera parameters have different stability in the reference method, only the most stable results are used here for comparison: It can be calculated, that the accuracy of the focal lengths fs_x and fs_y is improved by 20% and 7.5%, respectively[5], whereas the stability of the estimation of the principal point (h_x, h_y) is yet improved by 37.5% and 40%, respectively. When trying to estimate any distortion parameters on the undistorted images, the reprojection error is slightly improved but the accuracy of the camera parameters drops.

It has to be pointed out that these are results for a *normal* lens, i.e. no wide-angled or fisheye lens. This is especially remarkable because the lens is of high industrial quality, which normally means it was manufactured with high precision.

4 Conclusions

This article introduces a non-iterative camera calibration procedure. The method uses a flexible calibration pattern displayed by a TFT monitor. By capturing a high number of different calibration patterns from the monitor, a correspondence list between pixels on the camera target and points on the calibration patterns is calculated. Using this correspondence list, subsequent images may be undistorted precisely by projecting them onto a virtual target. The intrinsic linear camera parameters of this virtual target are then determined using a traditional calibration method. These can be adjusted

[5] Since the pixel width and height need not be equal the focal lengths may be different.

	Undistortion method	Intrinsic parameters $\sigma[fs_x]$	$\sigma[fs_y]$	$\sigma[h_x]$	$\sigma[h_y]$	Reprojection error $\sigma[e_x]$	$\sigma[e_y]$
	no undistortion	1.6964	1.7701	0.9062	1.1979	0.9673	0.7906
reference	undistortion using κ_1	0.5728	0.5877	0.4156	0.4982	0.2534	0.3144
reference	undistortion using κ_1, κ_2	0.4846	0.4980	0.3547	0.4228	0.1905	0.2812
reference	undistortion using κ_1, κ_2, t_1	0.4700	0.4836	0.3430	0.6514	0.1899	0.2684
reference	undistortion using κ_1, κ_2, t_1, t_2	0.4697	0.4833	0.8893	0.6513	0.1898	0.2681
reference	undistortion using κ_1-κ_3, t_1, t_2	0.4833	0.4976	0.8898	0.6509	0.1896	0.2681
new	undistortion using virtual camera	0.3749	0.4479	0.2144	0.2555	0.1599	0.2316
new	undistortion using virtual camera and additionally κ_1-κ_3, t_1, t_2	0.3785	0.4335	0.7552	0.5151	0.1433	0.2236

Table 1: Impact of the virtual camera on camera calibration. The standard deviations of the parameters are given in pixel.

to rotate the optical axis of the virtual camera onto that of the real one.

Examination of the accuracy of the undistortion method shows a very low reprojection error. Undistorting camera images with our method greatly improves the accuracy of traditional calibration methods by 7.5% to 39.5%. Since the undistortion process is independent from the actual camera calibration process, it is very suitable to be used as a preprocessing method.

In particular, multi camera calibration methods, which are very complicated for distorted camera images, should greatly benefit by using this method.

References

[1] I. N. Bronstein and K. A. Semendjajew. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, Thun und Frankfurt/Main, 6th edition, 2005.

[2] K. Hanke and P. Grussenmeyer. Architectural photogrammetry: Basic theory, procedures, tools. *ISPRS Commission 5 tutorial*, September 2002.

[3] F. J. Harris. On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83, January 1978.

[4] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2003.

[5] J. Heikkilä. Geometric camera calibration using circular control points. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(10):1066–1077, 2000.

[6] K. Hirakawa and T. W. Parks. Adaptive homogeneity-directed demosaicing algorithm. *IEEE Transactions on Image Processing*, 14(3), March 2005.

[7] R. Sagawa, M. Takatsuji, T. Echigo, and Y. Yagi. Calibration of lens distortion by structured-light scanning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 832–837, 2005.

[8] S. N. Sinha, M. Pollefeys, and L. McMillan. Camera network calibration from dynamic silhouettes. *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1:195–202, July 2004.

[9] P. Sturm and B. Triggs. A factorization based algorithm for multi-image projective structure and motion. In *ECCV (2)*, pages 709–720, 1996.

[10] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3, No.4, August 1987.

[11] G. Wolberg. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1994.

[12] Z. Zhang. A flexible new technique for camera calibration, August 2002.

[13] H. Zollner and R. Sablatnig. A method for determining geometric distortion of off-the-shelf wide-angle cameras. In *27th DAGM Symposium on Pattern Recognition, Vienna*, 2005.

Accurate Computation of Geodesic Distance Fields for Polygonal Curves on Triangle Meshes

David Bommes, Leif Kobbelt

Computer Graphics Group, RWTH Aachen

Email: {bommes,kobbelt}@cs.rwth-aachen.de

Abstract

We present an algorithm for the efficient and accurate computation of geodesic distance fields on triangle meshes. We generalize the algorithm originally proposed by Surazhsky et al. [1]. While the original algorithm is able to compute geodesic distances to isolated points on the mesh only, our generalization can handle arbitrary, possibly open, polygons on the mesh to define the zero set of the distance field. Our extensions integrate naturally into the base algorithm and consequently maintain all its nice properties.

For most geometry processing algorithms, the exact geodesic distance information is sampled at the mesh vertices and the resulting piecewise linear interpolant is used as an approximation to the true distance field. The quality of this approximation strongly depends on the structure of the mesh and the location of the medial axis of the distance field. Hence our second contribution is a simple adaptive refinement scheme, which inserts new vertices at critical locations on the mesh such that the final piecewise linear interpolant is guaranteed to be a faithful approximation to the true geodesic distance field.

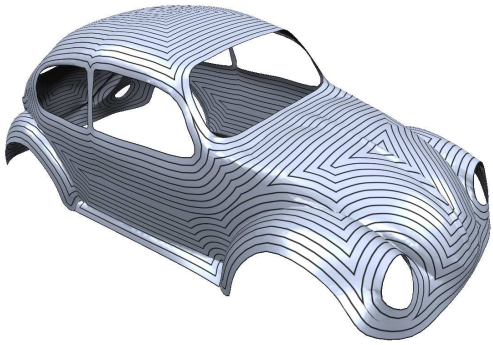


Figure 1: The isolines of the geodesic distance field with respect to the boundaries of the car model are visualized.

1 Introduction

The computation of geodesic distances on a triangle mesh has many applications in geometry processing, ranging from segmentation and low distortion parametrization to motion planning and tool path optimization. In most cases the true geodesic distance field is approximated by some fast marching method which leads to acceptable results on nicely structured meshes and away from singularities of the distance function. However, such simple propagation schemes tend to become numerically unstable on not-so-nice meshes as they often occur in practical applications. Moreover, since they use the same mesh as a representation for the input geometry as well as the distance field, the precision is limited by the mesh resolution. Surazhsky et al. [1] present a practical implementation of the geodesic distance algorithm of Mitchell et al. [2]. This was the first time that an exact geodesic distance computation has become applicable to arbitrary input meshes of practically relevant complexities. However, in this algorithm, the distance computation is initialized by one or more isolated points on the mesh and the distance is propagated from them (in Section 3 we present a summary of this algorithm). Unfortunately, for many practical applications this is too restricted. In general one would like to be able to compute the geodesic distance with respect to a curve on the surface, i.e., a polygon on the mesh since this allows us to take arbitrary boundary conditions into account. See Fig.1 for an example.

2 Previous work

In this paper we address two elementary mesh operations, geodesic distance computation and adaptive refinement.

Dijkstra's algorithm for computing shortest paths along edges can be used as an approximation for the geodesic field. Lanthier et al. [3] improved the initial poor results by adding many extra edges to the mesh.
Kimmel and Sethian [4] adapted the fast marching method to compute closer approximations of geodesic distances. On well-shaped input meshes this method performs very good, but in the case of obtuse angles or needle triangles even improved update rules and special handling as proposed before [5, 6, 7] can lead to large errors. Fast marching algorithms are able to approximate the geodesic distance field induced by polygonal sources, but the quality strongly depends on the mesh structure near the medial axis, which is typically not suited to represent the geodesic field, as we show in this paper.

The most famous exact algorithm was developed by Mitchell et al. [2] in 1987 and the first practical implementation was proposed eighteen years later by Surazhsky et al. [1]. They showed that the worst case complexity of $O(n^2 \log n)$ is quite pessimistic and in practice the average is close to $O(n^{1.5})$ which makes the algorithm practical for common model complexities. Details of this algorithm are presented in the next section. They also introduce a merging operation to design an approximation algorithm with guaranteed error bounds. Liu et al. [8] studied a robust implementation strategy which handles all degenerate cases that occur in real world data. In this paper we will generalize this exact and approximate algorithm from point sources to arbitrary source polygons.

In the context of adaptive mesh refinement two common techniques, namely red-green-triangulations [10] [11] and $\sqrt{3}$-subdivision, lead to regular structures and preserve the triangle quality. $\sqrt{3}$-subdivision is composed of one-to-three triangle splits and edge flips which changes not only the tessellation but also the geometry (and hence the geodesic distance field) of a triangle mesh and is thus not suited for our application.

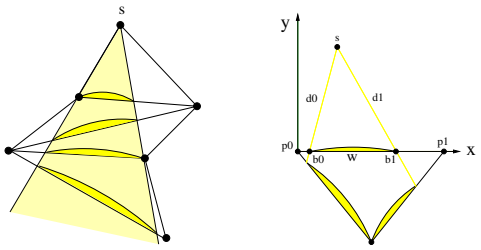


Figure 2: (a) Starting on the point source $\mathbf{s}$ a (shaded) pencil of rays is propagated through three unfolded triangles along of straight lines. Each window is highlighted by an arc which is always on the edge side pointing to the source. (b) An edge aligned two dimensional coordinate system is used to compute new windows which are induced from window w.

At the core of red-green-triangulations is the one-to-four triangle split. Red and green tags are used to preserve consistency. The refinement conserves the original geometry and could be integrated into our framework. However for our application a simpler and more local refinement is possible which we propose in section 6.

3 The exact geodesic algorithm

Since our algorithm is an extension of [1] we briefly explain the basic principles and the resulting base algorithm.

In the plane, the geodesic distance coincides with the Euclidean distance. Hence, with respect to an isolated point, it is the square root of a quadratic function. On a triangle mesh, i.e. on a piecewise planar surface, the geodesic distance with respect to a point turns out to be a piecewise function where in each segment the distance is given by the square root of a quadratic function plus an optional constant offset. This offset has to be introduced to properly handle saddle points on the surface.

The central idea of the algorithm [1] is to propagate exact distance information from one triangle to its neighbors with a Dijkstra-type algorithm. The key observation is that it is sufficient to store the piecewise distance function on the edges of the tri-

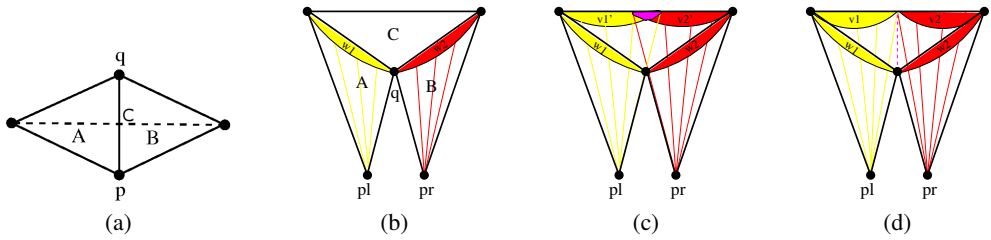


Figure 3: (a) The geodesic distance field w.r.t point $\mathbf{p}$ is computed on a cap consisting of triangles A,B, and C. (b) Cutting along the edge $\overline{pq}$ unfolds the cap isometrically and enables the distance propagation in the plane through windows w_1 and w_2. (c) The temporarily propagated windows v_1' and v_2' overlap in the middle region. (d) The final windows v_1 and v_2 are properly cut to represent the piecewise geodesic distance along the edge.

angle mesh since this is sufficient for the propagation and also for the exact evaluation of distances everywhere on the surface.

For each edge of the mesh the algorithm maintains a list of segments, so-called *windows*. Each window defines the geodesic distance field within a pencil of rays covering both neighboring triangles (see Figure 2). When distance information is propagated across a triangle, the (incoming) windows have to be mapped to the opposite side. The propagation includes the proper intersection of windows, because unlike the planar case on a surface propagated windows can overlap. Since the distance function is continuous, the intersection requires to find the point where the distance function values in both windows are identical.

We illustrate the procedure with a simple example. The cap of Figure 3 (a) consists of three isosceles triangles A,B and C. Now we want to compute the geodesic distance field w.r.t the point $\mathbf{p}$. Since $\mathbf{p}$ is coplanar with the triangles A and B they are covered with a pencil of rays emanating from $\mathbf{p}$ through single *windows* w_1 and w_2. To propagate the distance information through w_1 and w_2 we cut the cap along the edge connecting $\mathbf{p}$ and $\mathbf{q}$ and unfold the triangles isometrically into the plane, i.e. all edge lengths and angles of the triangles are preserved (see 3 (b)). In this setting $\mathbf{p}$ is doubled into $\mathbf{p}_l$ and $\mathbf{p}_r$. Now we are ready to propagate the pencil of rays defined by w_1 and w_2 across triangle C and create new temporarily overlapping windows v_1' and v_2' depicted in Figure 3 (c). Evaluating the distances induced by both pencils of rays the windows can be intersected and properly cut to final windows v_1 and v_2 (see Figure 3 (d)) which correctly represent the continuous piecewise geodesic distance function along the edge.

A nice feature of this *window* formulation is that all computations can be formulated in local two dimensional coordinates, i.e. only the mesh topology and scalar edge lengths are required. The necessary condition for this edge based algorithm is that geodesic paths can only pass through vertices with a total angle greater or equal than 2π, i.e. saddles and flat points. This result was proven by Mitchell et al. in [2]. Saddle points and concave boundary points act as pseudo sources which generate additional new windows covering the geometric shadow of the locally expanded surface.

3.1 Base algorithm

At first all source windows in the immediate vicinity of source points are created and pushed into a priority queue preferring shorter distances, because we want to compute the minimal geodesic distance. Notice that in general the result is independent of the propagation order but the priority queue ensures that windows are propagated as a wavefront which gives a strong speedup and makes the algorithm practical. Working off the queue the current window is always propagated into the next unfolded triangle, where new windows are created (see Figure 2). When the front reaches saddle or boundary vertices new source windows are added. All new windows can overlap with already existing windows and must be intersected accordingly. The algorithm terminates when all edges are parti-

tioned by the minimal geodesic distance windows, i.e. when the queue is empty. The pseudocode algorithm is presented below and all necessary computations are explained in more detail in the next sections.

Algorithm 1 Exact Geodesic Field

```
sourceWs = createSourceWindows()
PQueue.add( sourceWs )
repeat
  curW   = PQueue.popFront()
  newWs  = propagate( curW )
  newWs += saddleAndBoundaryWs( curW )
  newWs  = intersect( newWs, oldWs )
  PQueue.add( newWs )
until queue.empty()
```

3.2 Circular window propagation

In the next section we will define a second type of windows, so from now on windows originating from point sources are called *circular windows*. The starting point for a propagation is depicted in Figure 2 (b). Given a window w the corresponding edge $\overline{p_0 p_1}$ is aligned to the x-axis of the local coordinate system with the origin in $\mathbf{p}_0$. Each window is described by a six tuple $(b_0, b_1, d_0, d_1, \sigma, \tau)$ with σ representing the optional constant offset between a pseudo source and a real source. The binary flag τ determines on which side of the x-axis the unfolded pseudo source $\mathbf{s}$ lies (symbolized in pictures by the arc). The window extents are encoded in b_0 and b_1 which are in the range $[0..|\overline{p_0 p_1}|]$. Due to the fact that the distances d_0 and d_1 of the window endpoints from the pseudo source are known the unfolded position $\mathbf{s}$ can be reconstructed via circle intersection.

$$\begin{aligned} s_x &= \frac{1}{2}(b_0 + b_1 + \frac{d_0^2 - d_1^2}{b_1 - b_0}) \\ s_y &= -1^{\tau}\sqrt{d_0^2 - (c_x - b_0)^2} \end{aligned}$$

Using the local coordinates of $\mathbf{p}_3$ which are computed analogously to $\mathbf{s}$ the new windows are found by 2D ray intersection. There are different constellations which can lead to one, two or three (on saddle points) new windows.

3.3 Circular window intersection

If two windows overlap and one provides a smaller distance everywhere the other is simply clipped against it. If both are minimal in part of the overlapping interval, both ranges are clipped to the point where both distance functions are equal. Notice that clipped windows have to be reinserted into the queue because their priority can change. Using the unfolded pseudo source $\mathbf{s}$ from the previous section the distance function d_c of an arbitrary point $(p_x, 0)$ in the interval of a window can easily be formulated. Due to the fact that $\mathbf{s}$ is not necessarily a real source (e.g. saddles induce pseudo sources) the distance σ from all traversed pseudo sources to the real source must be added.

$$d_c(p_x) = \sqrt{(p_x - s_x)^2 + s_y^2} + \sigma \qquad (1)$$

Trying to find the intersection of two such distance functions, namely $d_{c1}(p_x) = d_{c2}(p_x)$ the computation ends up as the solution of a quadratic equation $Ap_x^2 + Bp_x + C = 0$. In this case there is exactly one solution in the overlapping interval and the coefficients of the polynomial are

$$\begin{aligned} A &= \alpha^2 - \beta^2 \\ B &= \gamma\alpha + 2s_{1x}\beta^2 \\ C &= \frac{1}{4}\gamma^2 - |s_1|^2\beta^2 \\ \alpha &= s_{1x} - s_{0x} \\ \beta &= \sigma_1 - \sigma_0 \\ \gamma &= |s_0|^2 - |s_1|^2 - \beta^2 \end{aligned}$$

4 Generalization to arbitrary sources

Our goal is to generalize the original geodesic distance computation algorithm from isolated points to polygonal curves on the surface. In a planar configuration the Euclidean distance function to a polygonal curve falls into several segments. In some segments the distance function is, again, the square root of a quadratic function. Those segments correspond to the vertices of the polygon. In other segments, the distance function is just linear. These segments correspond to the edges of the polygon. See Figure 1 for an example.

Going from the plane to a piecewise planar triangle mesh, we can still propagate the distance function from one triangle to its neighbors by storing

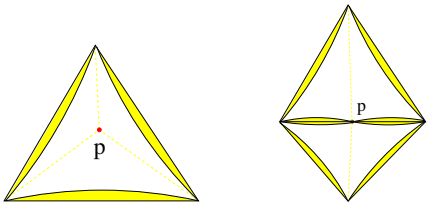


Figure 4: (a) An arbitrary point source p on the surface induces three windows in the corresponding triangle. (b) The six windows of a point source on an edge.

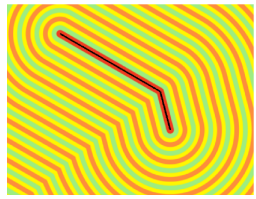


Figure 5: The geodesic distance field w.r.t the black polygon. Linear waves emanate orthogonal to line segments and circular waves emanate from each endpoint of a line segment.

windows of the piecewise distance function on each edge. The only difference regarding the last section is that now we need to handle two different types of windows: the ones where the distance function is of the form (1) and the ones where the distance function is linear. The Dijkstra-type propagation algorithm then has to handle all kinds of window intersections: circular-circular, circular-linear, and linear-linear. In the following we will give the explicit formulae for the corresponding intersection points where the two distance functions coincide. Additionally we need the ability to create circular source windows induced by arbitrary points on the surface which will be discussed first.

4.1 Arbitrary points

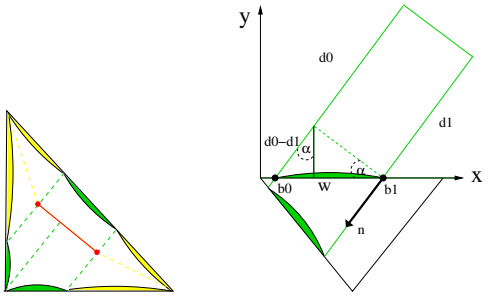
The original algorithm [1] was proposed to allow point sources only at vertex positions. However it is straightforward to overcome this limitation. Given an arbitrary point p on the surface the three edges of its containing triangle are initialized with windows emanating from this point as depicted in Figure 4. The new created windows are intersected with all other windows on an edge to handle multiple sources. Special care is needed for points lying exactly on an edge. In this case the edges of both triangles must be initialized.

4.2 Polygons on the mesh

As seen in Figure 5 straight line segments induce linear and circular waves from its endpoints. Consequently we create linear and circular windows for each segment of a piecewise linear polygon. Exploiting the window intersection algorithms already necessary for the window propagation the overall initialization becomes very simple, because overlaps are handled consistently.

As a preprocess we subdivide the piecewise linear input polygon such that every segment lies entirely in one triangle. This can easily be done by inserting vertices on all intersections between triangle and polygon edges. Using this decomposition it is possible to handle each polygon segment independently. We illustrate the procedure with one line segment in a single triangle as depicted in Figure 6 (a). At first we add linear windows (green) whose extents are computed by intersecting orthogonal rays starting from the endpoints of the line segment with all triangle edges. Additionally, both endpoints induce circular windows (yellow) which are computed as described in Section 4.1. All new windows are again intersected with windows already registered to an edge. Notice that due to the exact equal distance intersection the result is again independent of the order in which the windows are added.

To complete the algorithm we next describe the propagation and intersection of linear windows. Now each window is expressed as a seven tuple $(id, b_0, b_1, d_0, d_1, \sigma, \tau)$ in which the added type id is either *circular* or *linear*. In the case of a *circular* window we proceed exactly as described in Section 3. For *linear* windows the tuple components have analogous meanings. The key difference is that the emanating boundary rays of a window starting at $(b_i, 0)$ in local coordinates are computed in a different way. They do not intersect at a pseudosource center but are always parallel (see Figure 6 (b)). The distance function over a linear window is a simple linear function fully determined by b_i and d_i.



(a) (b)

Figure 6: (a) A line source within a triangle induces a set of linear (green) and circular (yellow) windows. (b) Computation of the propagation direction in local coordinates.

4.2.1 Linear window propagation

The starting point is depicted in Figure 6 (b). Similar to section 3 the window w covers the segment between b_0 and b_1 on the edge e. The x-axis is aligned to e and the y-axis lies in the plane of the triangle where the window should be propagated through. Using elementary geometric calculations the propagation direction $\mathbf{n} = (n_x, n_y)$ can be computed in terms of the local coordinate system. The differences $|d_1 - d_0|$ and $b_1 - b_0$ define the angle between the linear front and the mesh edge:

$$\sin\alpha = \frac{-n_x}{|d_0 - d_1|} = \frac{|d_0 - d_1|}{b_1 - b_0}$$

Solving the previous equation for n_x, n_y can be computed by the theorem of Pythagoras:

$$\begin{aligned} n_x &= -\frac{(d_0 - d_1)^2}{b_1 - b_0} \\ n_y &= -\sqrt{(d_0 - d_1)^2 - n_x^2} \end{aligned}$$

Using these ray direction instead of the ray directions induced by the unfolded pseudo source the remaining part of the window propagation is identical to Section 3. Here overlaps of propagated windows can happen as well. For this reason the next paragraph describes all possible cases, namely linear-linear and circular-linear window intersections. Both reduce to the solution of a quadratic equation.

4.2.2 Linear window intersection

Again there are two different cases for window intersections. The trivial one occurs when the distance function of one window is larger in the whole overlapping interval. In this case it is easily clipped against the other window.
The more interesting case happens when the minimal distance function in the overlapping interval is composed of both windows. In this case there must be a point $(p_x, 0)$ where both distance functions are equal.

The distance function of a linear window along an edge is a simple linear function (cp. Figure 6 (b)) which can be formulated in terms of $\mathbf{n}$ or directly using the window components. It fulfills the interpolation condition $d_l(b_i) = d_i$.

$$d_l(p_x) = p_x \underbrace{\frac{d_1 - d_0}{b_1 - b_0}}_{m} + \underbrace{\frac{b_1 d_0 - b_0 d_1}{b_1 - b_0} + \sigma}_{n}$$

Now we are ready to compute intersections of linear windows with linear and circular windows to find the separation point p_x on the corresponding edge:

1. linear-linear intersection

$$\begin{aligned} & d_{p1}(p_x) &=& \; d_{p2}(p_x) \\ \Leftrightarrow \quad & p_x m_1 + n_1 &=& \; p_x m_2 + n_2 \\ \Leftrightarrow \quad & p_x &=& \; \tfrac{n_2 - n_1}{m_1 - m_2} \end{aligned}$$

2. circular-linear intersection

$$\begin{aligned} & d_c(p_x) &=& \; d_l(p_x) \\ \Leftrightarrow \quad & \sqrt{(p_x - s_x)^2 + s_y^2} + \sigma &=& \; p_x m + n \end{aligned}$$

Squaring the previous equation leads to a quadratic equation $Ap_x^2 + Bp_x + C = 0$ with coefficients

$$\begin{aligned} A &= 1 - m^2 \\ B &= -2(s_x + m(n - \sigma)) \\ C &= s_x^2 + s_y^2 - (n - \sigma)^2 \end{aligned}$$

Notice that unlike the previous intersections here exist possibly two valid solutions which can lead to a trisection of the overlapping interval. In this case the cut circular window lies in the middle of two disconnected parts of the linear window.

5 Approximation algorithm

The propagation of distance information across many triangles leads to an increasing number of windows per edge because windows split up at vertices. A large number of windows increases the time as well as the space complexity of the algorithm. So the idea for the ε-Approximation-Algorithm in [1] is to merge neighboring windows on an edge whenever the induced relative error is acceptable. Allowing for example a relative error of $\varepsilon = 0.1\%$ leads to visually indistinguishable results but enables the processing of huge models with several millions of faces which are far too complex for the exact algorithm. Again the proposed linear windows fit naturally in the original framework and share all properties necessary for window merging. Before we describe the merging of linear windows we shortly review the basic principles and the case of circular windows. For details see [1].

To guarantee consistency of the geodesic field some conditions must be checked before merging two neighboring windows.

1. **Directionality:** Both windows propagate into the same direction.
2. **Visibility:** The pencil of rays of the merged window must at least cover all rays of the original windows so that no gaps arise.
3. **Continuity:** The distance at the endpoints bounding the merged window must be preserved to conserve distance field continuity.
4. **Type:** Both windows must be of the same type, e.g. planar or circular.

Additionally the user can prescribe a relative error bound ε_U so that only those merges are perfomed where the relative difference between the distance function of the new window $d'(p_x)$ and the original piecewise distance function $d_{lr}(p_x) = d_l(p_x) \cup d_r(p_x)$ is smaller than ε_U, i.e.

$$\frac{|d_{lr}(p_x) - d'(p_x)|}{d_{lr}(p_x)} \leq \varepsilon_U$$

5.1 Merging of circular windows

Taking two neighboring circular windows

$$\begin{aligned} w_l &= (id, b_{0l}, b_{1l}, d_{0l}, d_{1l}, \sigma_l, \tau_l) \\ w_r &= (id, b_{0r}, b_{1r}, d_{0r}, d_{1r}, \sigma_r, \tau_r) \end{aligned}$$

which meet at the common point $(b_{1l}, 0) = (b_{0r}, 0)$ the merged window w' is already determined up to σ' due to the necessary conditions:

$$\begin{aligned} id' &= id \\ b'_0 &= b_{0l} \\ b'_1 &= b_{1r} \\ d'_0 &= d_{0l} + \sigma_l - \sigma' \\ d'_1 &= d_{1r} + \sigma_r - \sigma' \\ \tau' &= \tau_l = \tau_r \end{aligned}$$

The continuity constrain restricts w''s pseudosource $s' = (s'_x, s'_y)$ to lie on a conic curve $s'^2_y(s_x)$. Because of the positivity of the d'_i and the visibility constraint the valid domain of this conic curve is further restricted. If it is the empty set, the merge is disallowed and in all other cases the smallest possible σ' is chosen (see [1] for details and how to evaluate the approximation error).

5.2 Merging of linear windows

The distance values d_i of a linear window can always be transformed so that the corresponding pseudosource distance σ vanishes. So w.l.o.g. two neighboring linear windows

$$\begin{aligned} w_l &= (id, b_{0l}, b_{1l}, d_{0l}, d_{1l}, 0, \tau_l) \\ w_r &= (id, b_{0r}, b_{1r}, d_{0r}, d_{1r}, 0, \tau_r) \end{aligned}$$

which join at the common point $(b_{1l}, 0) = (b_{0r}, 0)$ can be merged into a linear window

$$w' = (id, b_{0l}, b_{1r}, d_{0l}, d_{1r}, 0, \tau_l = \tau_r)$$

which satisfies all necessary constraints and is fully determined by the original windows. Notice that the visibility constraint is always fulfilled because diverging linear windows can only occur in combination with an additional point source or a saddle. The maximum approximation error is obtained at the joining point and can be computed as

$$\varepsilon = |1 - \frac{d_{1r}(b_{1l} - b_{0l}) + d_{0l}(b_{1r} - b_{1l}))}{d_{1l}(b_{1r} - b_{0l})}|$$

6 Adaptive refinement

The algorithm presented in the last section is able to compute the exact geodesic distance field on a triangle mesh with respect to an arbitrary polygon

embedded on the mesh. However, the distance information is not given explicitly but rather through a set of windows defined on the edges of the mesh. For most geometry processing algorithms this implicit information has to be made explicit. The standard approach to do this is to simply sample the distance function at the mesh vertices and then use a linear interpolant on each face as an approximation of the original distance field. In order to have some guarantee about the approximation tolerance, we have to refine the mesh in regions where this tolerance is violated. Usually this happens in the vicinity of the geodesic medial axis. To decide where to refine we compare the exact geodesic distance on edges with the linear interpolant and check if a user-defined threshold is exceeded. In this case we split the edge and insert a new sample point.

The geodesic distance field is smooth with constant gradient magnitude everywhere except for the geodesic medial axis. By properly placing the newly inserted vertices on the medial axis (i.e. at the maximum distance value) we can avoid excessive local refinement. This feature sensitive placement leads to optimal convergence and is in the spirit of [9].

Since edge splits in arbitrary order lead to poor triangles we employ a strategy similar to adaptive red-green triangulations. An important feature is that our refinement does not change the underlying geometry and can be seen as a pure upsampling of the original geodesic distance field. Due to this fact no recomputation of the geodesic field is necessary. The geodesic distance has to be updated only for those edges that are newly inserted. The edge-based refinement and the evaluation procedure are described in more detail in the next sections.

6.1 Edge-based refinement

In each refinement step we evaluate for each edge the maximal deviation between the exact distance function given as a pieceweise function along the edge and the linear function interpolating the exact distance only at the edge endpoints. If this maximal deviation exceeds a user-defined threshold the edge is tagged for refinement and the corresponding point p_{max} is cached as the optimal splitting position. Simply splitting all tagged edges would result in poor triangle quality. We aim at applying a one-to-four split (see Figure 7) of triangles lying entirely in the refined region. The one-to-four split operator

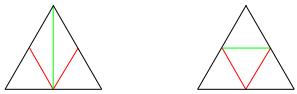


Figure 7: Implementation of a one-to-four split of a triangle using only edge split and edge flip operators. (a) Each edge of the black triangle is first split in arbitrary order. (b) The green edge, characterized by two adjacent triangles with only one original edge segment, is flipped to complete the one-to-four refinement.

can be composed of edge split and edge flip operations. For one triangle this requires the splitting of all edges (in arbitrary order) and the flipping of one specific edge (see Figure 7). To increase the number of regular one-to-four splits we iteratively tag all edges which are adjacent to triangles with already two tagged edges. This edges will be splitted on their midpoint. Subsequently all tagged edges are split at their cached split positions and all necessary flips are done. Identifying which edges should be flipped is easy if we mark all new created edges as red during the splitting process. If both triangles of a red edge are bounded by exactly two (the edge itself plus one additional) red edges the edge must be flipped.

6.2 Evaluation of interpolation error

The Geodesic Distance Function along a mesh edge e is defined piecewisely and consists of linear and circular segments corresponding to linear and circular windows. To compute the maximum deviation between this exact function and the linear interpolant defined by the exact distances on the edge vertices it is possible to first evaluate the maximal deviation for each segment individually and then take the overall maximum.
In the case of a linear segment the evaluation is simple. The difference between two linear functions is again a linear function and so the maximum is always on the boundary of the corresponding linear window.
In the case of a circular window the maximum can be computed analytically. The difference of both distance functions along the edge

$$E(p_x) = \sqrt{(p_x - s_x)^2 + s_y^2} + \sigma - (ax + b)$$

has a single extremum at

$$q_x = s_x - a\sqrt{\frac{s_y^2}{a^2-1}}$$

If q_x is not in the valid interval $[b_0..b_1]$ of the window the maximal deviation is on the boundary of the window as in the linear case.
The optimal position for a new sample point is exactly the position p_{max} where the deviation is maximal. However allowing split points to lie arbitrarily close to the edge endpoints leads to degenerate triangles. In practice we clamp the splitting position to be in the range of $25 - 75\%$ of the edge length. Additionally if the optimal position lies between $12.5 - 25\%$ or $75 - 87.5\%$ we adjust the new vertex so that the optimal position lies exactly on the midpoint of the new created edge because this leads to better triangulations. Given the optimal sample position $t \in [0..1]$ the update is as follows:

$$t \mapsto \begin{cases} 0.25 & 0 \leq t < 0.125 \\ 2t & 0.125 \leq t < 0.25 \\ t & 0.25 \leq t \leq 0.75 \\ 2t-1 & 0.75 < t < 0.875 \\ 0.75 & 0.875 \leq t \leq 1 \end{cases}$$

7 Results

We demonstrate the results of our algorithm on models of different complexity. Table 1 shows the corresponding timings for the computation of the exact and approximated geodesic fields which were generated on an AMD 64 3500+ system with 2GB of RAM. Additionally the average number of windows per edge (WPE) is listed. On the David and the Fandisk model we computed the geodesic field w.r.t. the red polygonal curves on the surface (see figure 9). The visualization uses a 1D texture to transfer the linear interpolant of the geodesic field into a color. For the car model depicted in Figure 1 we computed the geodesic field for the boundary and applied the adaptive refinement to get an satisfactory visualization. The refined mesh is showed in Figure 9. Obviously most of the mesh refinement occurs in a thin local neighborhood near the medial axis of the geodesic field. The plane model in Figure 8 illustrates the quality gain of our adaptive refinement in more detail. The upper row shows the original mesh with the corresponding linear interpolant of the geodesic field. Even though the mesh structure looks nice the result is very noisy near the medial axis and shows large errors. Applying the presented adaptive refinement we gain a high quality explicit representation of the geodesic field showed in the lower row together with the generated mesh structure. The approximation error reduced by a factor of 100 while the number of faces increased by a factor of 4.

Table 1: Timings

Model	#Faces	Time exact	WPE exact	Time 0.1%	WPE 0.1%
Plane	422	4ms	2.40	2ms	1.2
Fandisk	12k	1.90 s	9.06	0.12s	1.6
Car	34k	3.03 s	7.06	0.91s	3.4
David	8M	-	-	165s	1.3

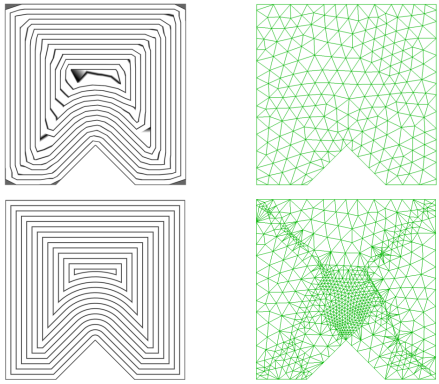


Figure 8: plane (422 faces)

8 Conclusion and future work

We have presented a generalization of the exact and approximate geodesic algorithm of [1] which allows to use arbitrary polygons as the boundary condition for the geodesic field. Our extensions integrate very naturally into the original algorithm and can easily be added to existing implementations. To increase the quality of the vertex based piecewise linear representation required for many applications using geodesics we included a well suited adaptive refinement technique which increases the quality of the piecewise linear representation to a user prescribed quality. The adaptive refinement strategy is very local and yields satisfactory mesh quality.

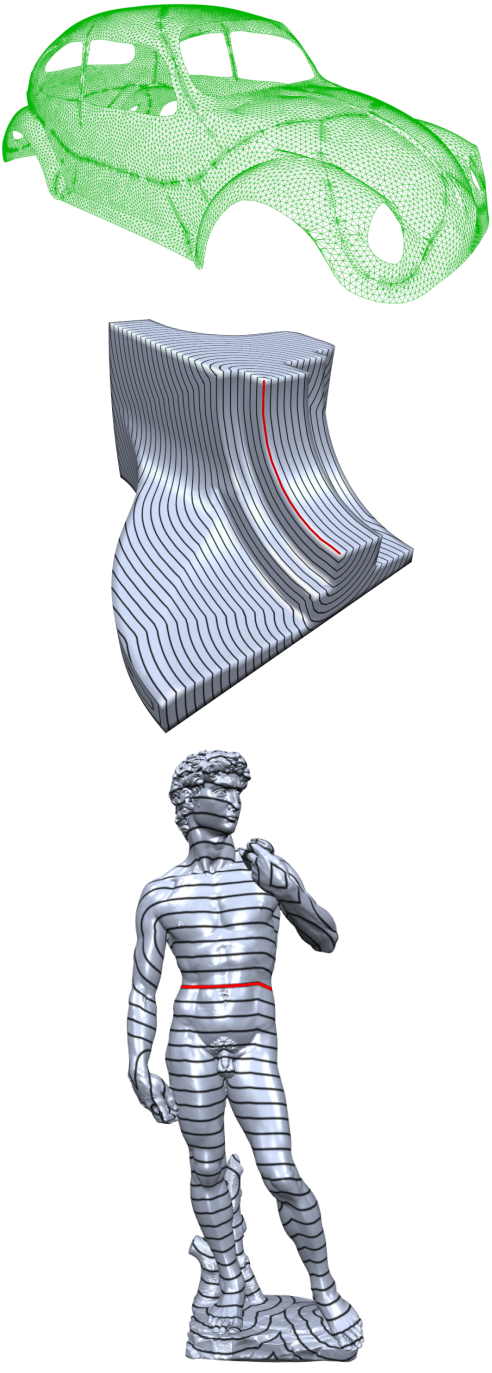


Figure 9: Car (34k faces), Fandisk (12k faces) and David (8M faces)

References

[1] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. Gortler and H. Hoppe. Fast exact and approximate geodesics on meshes. Recognizing Surfaces using Three-Dimensional Textures. In *ACM SIGGRAPH Proc.,* 553–560, 2005.

[2] J.S.B. Mitchell, D.M. Mount and C.H. Papadimitriou. The discrete geodesic problem *SIAM Journal of Computing,* 16(4):647–668, 1987.

[3] M. Lanthier, A. Maheshwari, and J.-R. Sack. Approximating weighted shortest paths on polyhedral surfaces. In *Proc. 13th Annu. ACM Symp. Comput. Geom.*, 274–283, 1997.

[4] R. Kimmel, and J.A. Sethian. Minimal Discrete Curves and Surfaces. In *Proc. of National Academy of Sci.95(15)*, 8431–8435, 1998.

[5] M. Novotni and R. Klein. Computing geodesic distances on triangular meshes. In *Proc. of WSCG*, 341–347, 2002.

[6] D. Kirsanov. Minimal discrete curves and surfaces. PhD thesis, Applied Math., Harvard University.

[7] M. Reimers. Minimal discrete curves and surfaces. PhD thesis, Math., Univ. of Oslo.

[8] Y.-J. Liu, Q.-Y. Zhou and S-M. Hu. Handling Degenerate Cases in Exact Geodesic Computation on Triangle Meshes. Computer Graphics International, 2007, to appear.

[9] L. P. Kobbelt, M. Botsch, U. Schwanecke and H.-P. Seidel. Feature-Sensitive Surface Extraction From Volume Data. In *ACM SIGGRAPH Proc.,* 57–66, 2001.

[10] R.E. Bank, A. H. Sherman and A. Weiser. Refinement Algorithms and Data Structures for Regular Local Mesh Refinement. In *Sci. Computing*,IMACS/North Holland, Amsterdam, 3–17, 1983.

[11] M. Vasilescu and D. Terzopoulus. Irregular triangulation, discontinuities and hierarchical subdivision. In *Proc. of Computer Vision and Pattern Recognition Conference*, 829–832, 1992.

[12] L. Kobbelt. $\sqrt{3}$ Subdivision. In *Proc. of ACM SIGGRAPH* 103–112, 2000.

Distance Calculation between a Point and a Subdivision Surface

Torsten Ullrich, Volker Settgast, Ulrich Krispel, Christoph Fünfzig, and Dieter W. Fellner

Institute of Computer Graphics and Knowledge Visualization (CGV), TU Graz, Austria
Graphisch-Interaktive Systeme (GRIS), TU Darmstadt, Germany
PRISM Lab, Arizona State University, Tempe, USA
Contact Email: t.ullrich@cgv.tugraz.at

Abstract

This article focuses on algorithms for fast computation of the Euclidean distance between a query point and a subdivision surface. The analyzed algorithms include uniform tessellation approaches, an adaptive evalution technique, and an algorithm using Bézier conversions. These methods are combined with a grid hashing structure for space partitioning to speed up their runtime.

The results show that a pretessellated surface is sufficient for small models. Considering the runtime, accuracy and memory usage an adaptive on-the-fly evaluation of the surface turns out to be the best choice.

1 Introduction

The problem to determine the Euclidean distance between an arbitrary point in 3D and a free-form subdivision surface is fundamental in many different communities including computer-aided geometric design, robotics, computer graphics, and computational geometry.

A lot of algorithms in the context of physical simulation, path planning, etc. have to determine this distance: an exemplary algorithm is the shape fitting approach by TORSTEN ULLRICH. It evaluates distances between a point cloud and some subdivision surfaces in order to fit a procedural model [1]. As query time is always an issue, the goal is to choose the best combination for the application at hand.

Subdivision surfaces are based on polygonal meshes, and they can be subdivided into triangle meshes. So, is it suitable to preprocess the object into a triangle mesh and compute distances to the object just by searching the closest triangle? Should the search be performed on the subdivision surface patches? This article discusses accuracy, runtime and memory usage of various approaches for searching strategy, surface primitives used, and calculation of the primitive's minimum distance.

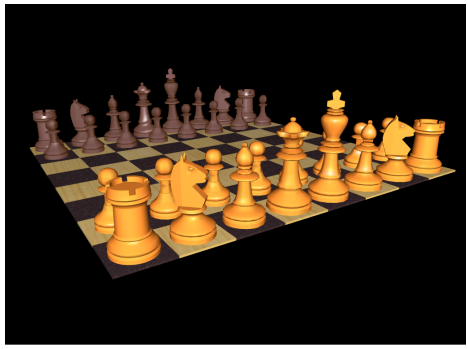


Figure 1: The test objects are chess figures modeled with subdivision surfaces. Each initial mesh has between 70 ("pawn") and 1 454 ("rook") polygons. The Figure shows all test pieces in their initial chess position.

2 Related Work

Subdivision surfaces are part and parcel of this article. They define an object through recursive subdivision starting from an initial control mesh. A variety of schemes with different subdivision rules exist for geometric design and graphics applications. An overview on subdivision surface modeling in the context of computer-aided design has been presented, e.g., by WEIYIN MA [2].

2.1 Subdivision Surfaces

A subdivision surface is defined by an infinite subdivision process. In contrast to parametric surfaces which provide a finite evaluation algorithm, a subdivision surface does not come with a direct evaluation method at arbitrary parameter values. Currently, it can be evaluated via

- **Uniform subdivision:** If the subdivision rules are applied sufficiently often, the resulting mesh will be a tight approximation of the limit surface [3]. For non-interpolating subdivision schemes, e.g., Catmull-Clark, the resulting mesh points will not lie on the limit surface in general. In order to decrease the deviation, limit point rules can calculate the point on the limit surface for a subdivision mesh point [4].
- **Adaptive subdivision:** Due to the exponential need of memory it is a good strategy to subdivide the mesh adaptively. This results in a subdivision process with varying subdivision depth but constant overall accuracy [5]. The use of limit point rules is essential for the connection of mesh parts with different subdivision depths.
- **Exact evaluation & conversion:** Stationary subdivision schemes, e.g., Catmull-Clark, allow an exact evaluation at arbitrary parameter values [6]. JOS STAM makes use of the property that regular patches (control mesh faces with all vertices of valence 4) can be evaluated as uniform, bicubic b-spline patches. The region around irregular points (non-valence 4) shrinks successively when subdividing the irregular patches, and the eigen-structure of the subdivision matrix is used to determine the limit there. Two alternative parameterizations for irregular patches were proposed in [7], which ensure non-degenerate derivatives of the parameterization. For Catmull-Clark subdivision, a regular quad patch can even be represented as a single bicubic Bézier patch.

2.2 Distance Calculations

Distance fields are a representation, where at each point within the field, the distance from that point to the closest point on a fixed object within the domain is known. In addition to distance, other properties (direction to the surface, etc.) may be derived from the distance field. A survey of methods for the precomputation and representation of distance fields can be found in “3D Distance Fields: A Survey of Techniques and Applications” [8]. To speed up the search in a domain, space-partitioning data structures allow to access object parts by spatial proximity and other properties. Data structures used for this, like tree structures (e.g., kd-tree), grid structures (e.g., 3D regular grid) and cell structures (e.g., Voronoi diagram) are described in “Geometric Data Structures for Computer Graphics” [9].

For applications where many distance queries are performed and the object is fixed within the domain, the distance field can be precomputed and represented in a scalar data structure for the domain, like a regular grid or a compressed regular grid. With this *distance field data structure*, the distance from a domain point to the closest object point can be retrieved from the data structure. In the case, where the object is deforming, it is necessary to update this derived, scalar field. In any case, where object information is stored about the location within the domain, this has to be updated. So for deforming and changing objects it is beneficial to keep this amount as small as possible, at best without any domain data structure. This setting without preprocessing is called *online distance evaluation*.

For the problem of distance computation to subdivision surfaces, we propose the following classification of approaches:

- **Approaches based on the distance field:** A separate scalar data structure reconstructs the (signed) distance to the closest point on the object [8]. We also consider in this group GPU approaches like [10], which compute and evolve the distance field in a small narrow band around the object.
- **Searching of surface primitives of the original object representation:** The curved surface patches, which correspond to a face of the control mesh, are organized in a spatial data structure for the domain based on their bounding volume. Only this data structure has to be updated after model deformations. The spatial data structure is then traversed in increasing minimum distance to the query point, and the primitive’s minimum distance is computed as a subproblem. A termination condition is necessary to stop the search with the correct distance value.

- **Searching of surface primitives derived from the original object representation:** Instead of using the surface primitives of the original object representation immediately, one derives a small set of simpler primitives from the original surface primitives. The reason could be that they offer a simpler minimum distance algorithm. In the case of subdivision surfaces, the surface's triangulation is often available also from other tasks.

3 Exposition of Methods

In this work, we chose three kinds of algorithms to determine the distance between a query point and a subdivision surface. The first group consists of three algorithms which use the triangulation of a subdivision surface. The next approach evaluates the subdivision surface on-the-fly. And the last algorithm converts it into Bézier patches. In this case distance queries are answered by a numerical minimization routine.

3.1 Uniform Triangulation

The most simple approach uses an uniform tesselation of the subdivision surface at a fixed depth to create a triangle mesh. For a tight approximation of the limit surface, the limit points of the control vertices have been used. For each query point the distance to each triangle is calculated [11], and the minimum is selected. This approach does naive search without any spatial data structure.

pros The calculation is robust and its correctness can be verified easily.

cons As runtime and memory footprint of a single distance query are linear in the number of triangles and exponential in the subdivision depth, this algorithm is not useful for real world applications. The implementation has been used to verify the results of the following algorithms, but it is not considered to be a practical solution.

3.2 Hashed Triangulation

A significant speed-up can be achieved if the triangulation is stored in a space-partitioning data structure. The hashed triangulation approach is a space-efficient implementation of a 3D regular grid by using spatial hashing [12]. In this way, the storage requirements can be restricted arbitrarily, e.g., linear in the number of model triangles.

For a given query point, the hashed triangulation approach determines which grid cells may potentially contain the nearest triangle. Within the grid cells in question, the registered triangles are checked. According to our classification, it is based on *searching of surface primitives derived from the original object representation*.

pros The technique is easy to implement, and a well chosen grid cell size gives good query times.

cons The memory footprint is exponential in the subdivision depth which disqualifies it for many applications. Another problem is the algorithm's dependency on the choice of the grid cell size. A reasonable size takes into account the model's bounding volume as well as its face distribution within the domain. This problem is discussed in detail in Section 4.2.

3.3 Hashed Triangulation – First Hit

A further speed-up is possible, if only the distance value (not the corresponding perpendicular point) is needed, and if a small error is acceptable. In this case, only the nearest non-empty cell is checked. If no other cell is checked, the returned value may have an error up to the length of the cell's diagonal.

pros Same as 3.2 Hashed Triangulation.

cons Same as 3.2 Hashed Triangulation. The returned distance value is only a rough approximation.

3.4 Adaptive Subdivision

The triangulation-based distance calculations described before have large memory requirements in common. If the subdivision control mesh has to remain in memory, for any reason, the triangulation-based methods are not suitable due to their large memory requirements. An approach which does the refinement of the subdivision mesh on-the-fly has always smaller memory requirements. Our implementation of the adaptive subdivision algorithm uses a hashed 3D regular grid structure to identify relevant subdivision patches. These patches are subdivided using slates [13] as needed. According to our classification, it uses *searching of surface primitives of the original object representation.*

pros The memory footprint is only linear in the size of the subdivision mesh due to the 3D hash table. The additional overhead during a patch evaluation is of small, fixed size and can be neglected.

Only a small preprocessing is needed. In contrast to triangulation-based approaches, this allows to modify the maximum subdivision depth and therefore adapt the accuracy of the distance calculation as needed.

cons The algorithm requires a substantial implementation.

3.5 Bézier Representation & Numerical Optimization

Some subdivision schemes, e.g. Catmull-Clark subdivision [6], allow direct evaluation at arbitrary parameter values. This property can be used to formulate a distance calculation algorithm. Having identified relevant subdivision patches, the algorithm converts them into Bézier patches. For regular patches this can be done exactly. Irregular patches have to be approximated. Using a parameterization as a Bézier patch, the distance calculation can be formulated as a minimization problem in parameter space [14–16]. For the resulting nonlinear minimization problem, Newton-type techniques [17], [18] can be used with suitable start values in parameter space.

pros The memory requirements are comparable to the adaptive subdivision algorithm. As the distance calculation is reduced to a standard problem of numerical optimization, highly-optimized numerical libraries can be used.

cons The Bézier approximation has some additional runtime overhead, but can be cached with the subdivision mesh. The following distance minimization requires considerable tuning of the step sizes. The choice of the start parameter of the Newton-like iteration has more influence on the runtime than the size of the model.

Furthermore the conversion of Catmull-Clark subdivision to bicubic Bézier patches is patent-registered ("Approximation of Catmull-Clark subdivision surfaces by Bézier patches", United States Patent No. 6950099).

4 Implementation

In order to allow a thorough comparison of the chosen algorithms some implementation issues are discussed in detail.

4.1 Evaluation Errors

The triangulation-based methods use a fixed, uniform subdivision depth of three subdivisions. Note that the use of limit points improves the approximation error, which can be bounded by a factor times the maximum of the triangle's side lengths, where the factor depends on the model. The limit points lie in the convex hull of the Bézier control mesh instead of the convex hull of the corresponding face's 1-ring in the Catmull-Clark mesh. This error has been used to set the termination condition of the adaptive subdivision algorithm. Therefore, the adaptive version has a maximum subdivision depth of three, but it is allowed to terminate earlier, if the resulting maximum error is of same size.

The Bézier surface patches resulting from the conversion have a deviation from the Catmull-Clark surface patches only in irregular patches. But the subsequent parameter search, which works with the Bézier representation, produces an error by itself. With the termination condition in parameter space it is difficult to control the distance error because the threshold in parameter space depends on the curvature near a minimum point's parameter. In our experiments we used only a fixed threshold.

The accuracy of the First-Hit algorithm is determined by the triangulation error plus $\sqrt{3}$ times the grid cell size.

4.2 Grid Size Problems

The grid cell size is not only responsible for the algorithm's accuracy. The choice of a reasonable value affects the algorithm's performance significantly. Unfortunately, the value depends on the distribution of the cached geometric primitives (triangles, Bézier patches, etc.) within space. Without additional knowledge only some heuristics are at hand. Let d be the bounding volume's diagonal length, and p be the number of geometric primitives to hash. If all objects are distributed uniformly in their bounding volume, a grid cell size of $d/\sqrt[3]{p}$ is a reasonable choice. If the surface of a geometric object is not distributed uniformly in space, the

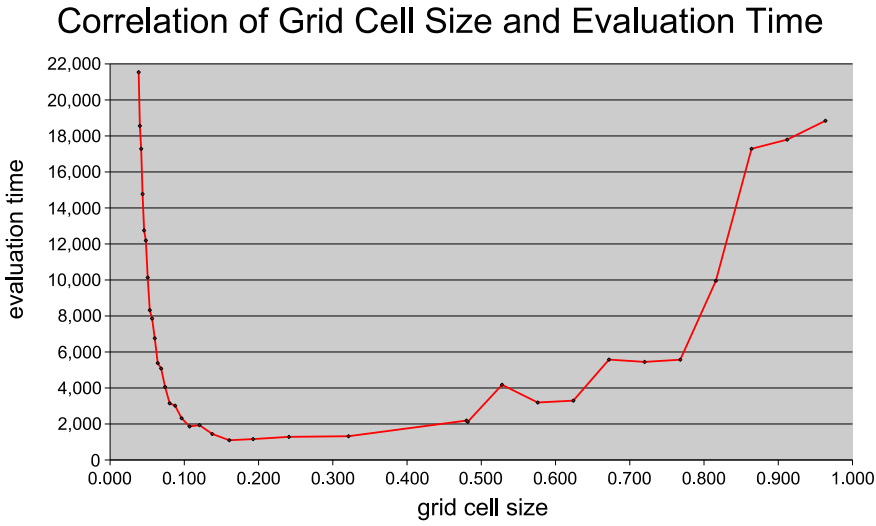


Figure 2: This Figure demonstrates the correlation between grid cell size and runtimes of hashing-based algorithms. The used test object "pawn" has been triangulated (8 862 triangles). All triangles reside inside the axis-aligned bounding box whose diagonal has a length of 3.94. According to the heuristics in Equation 1 the cell size should be between $3.94/\sqrt[3]{8862} \approx 0.19$ and $3.94/\sqrt[5]{8862} \approx 0.65$. The needed time in milliseconds to calculate the distance of 10 000 arbitrary points to the triangle mesh using the First-Hit algorithm is plotted against the used grid cell size.

grid should be coarsened. In our implementation the grid cell size had been chosen to

$$s := \frac{d}{\sqrt[n]{p}} \tag{1}$$

with $3 \leq n \leq 5$, which has led to feasible runtimes. An illustrative example in Figure 2 shows the correlation of cell grid size and evaluation time for a test object.

4.3 Hashing

All presented algorithms use grid-based hashing. We used the hashing function presented by MATTHIAS TESCHNER [12]. It takes the indices (x, y, z) of a grid cell and returns the hash value

$$hash(x, y, z) = (x\, p_1 \textbf{ xor } y\, p_2 \textbf{ xor } z\, p_3) \textbf{ mod } n \tag{2}$$

using the prime numbers $p_1 = 73\ 856\ 093$, $p_2 = 19\ 349\ 669$, $p_3 = 83\ 492\ 791$. The function can be evaluated very efficiently and produces a comparatively small number of hash collisions for small hash tables of size n. The traversal within the grid structure is illustrated in Figure 3.

4.4 Slates for Subdivision Surfaces

The adaptive subdivision algorithm does not modify the base mesh. Instead a separate data structure is used consisting of two so-called *slates*.

A slate is composed of a two-dimensional array table of size

$$(2^{sd} + 3)^2$$

and four one-dimensional corner arrays of size

$$(val - 4) \cdot 2,$$

where sd is the maximum subdivision depth and val the maximum valence. For performance reasons, the slates are allocated statically as they can be reused for each face to be tessellated.

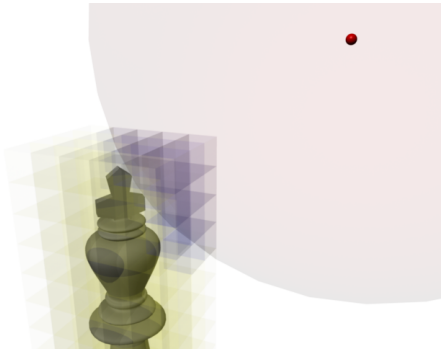


Figure 3: The storage of a model in a regular grid allows a fast preselection of relevant patches/triangles, which are near the query point (red). In combination with a good hash function the memory footprint is proportional to the number of model primitives.

The subdivision process firstly collects the 1-neighborhood of the considered face f and stores it in the first slate. The vertices of f and the vertices of its edge neighbor faces are stored in the table. If one of the vertices of f has valence greater than four, the remaining vertices are stored in the dedicated corner arrays. Figure 4 illustrates this storage scheme for a quad. Other configurations and further details on slates can be found in "Adaptive Tessellation of Subdivision Surfaces" [13].

The subdivision algorithm processes the vertices row by row and stores the result of one subdivision step in the second slate.

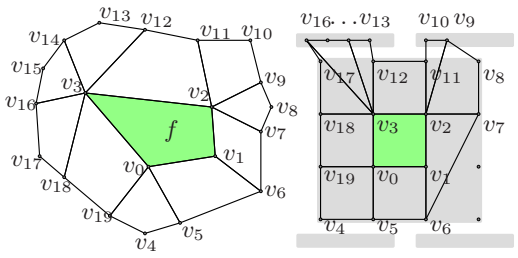


Figure 4: The adaptive subdivision algorithm stores the collected 1-neighborhood of a face f from the base mesh (left) in a data structure called slate (right).

For the next step, source and destination slates are swapped. After two subdivision steps, the algorithm starts calculating distances from the corresponding limit points of the 25 (5×5) vertices to the query point. For the following subdivision steps, only a subpart of 9 (3×3) vertices of the table array is used, see Figure 5. The subpart is chosen depending on the results of the distance calculations. The process is repeated until the difference of the minimal distance for the current and the last iteration is below a user defined threshold.

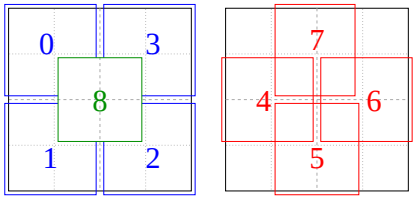


Figure 5: After the second subdivision step each face of the control mesh consists of 5×5 vertices. During the distance calculation only relevant subparts out of nine possibilities are processed further on. Five possible sectors are illustrated on the left, four on the right.

5 Benchmarks

The test scenario is made of six subdivision surface models.

1. **Pawn** This object consists of 70 patches. Its triangulation at subdivision level 3 has 8 862 triangles.
2. **Rook** Within the test scenario this object is the most complex one. It is composed of 1 454 subdivision surface patches, respectively 185 328 triangles.
3. **Knight** The control mesh of this model has 78 faces. Triangulated after three successive subdivisions it consists of 9 356 triangles.
4. **Bishop** The bishop is modeled using 130 patches. In this case the triangulation-based algorithm have to handle 16 542 triangles.
5. **Queen** This model has 387 subdivision surface patches which results in a triangulation with 49 508 elements.
6. **King** The king consists of a subdivision mesh with 175 faces. Its tessellation with 19 560 triangles ranges in the midfield of the test scenario.

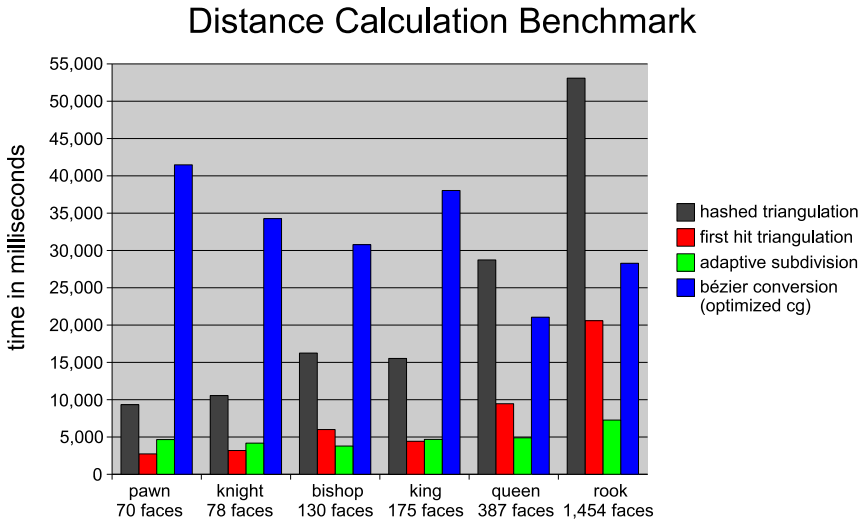


Figure 6: The test objects are Catmull-Clark surfaces. The smallest object – the Pawn – has a subdivision control mesh which consists of 70 faces. The most complex model is the Rook with 1 454 patches. Its triangulation at subdivision level 3 has 185 328 triangles. The triangulation-based algorithms as well as the adaptive subdivision one correlates with the model's complexity in contrast to the approach using Bézier conversion and numerical optimization. This algorithm is rather determined by internal parameters (initial parameter values, step size, etc.) than by model complexity.

During each test an algorithm has to calculate the distance between the test object and 10 000 arbitrary query points. The query points are uniformly distributed within a box whose volume is twice as large as the test object's axis-aligned bounding box (AABB) volume. Each test model has a closed 2-manifold boundary and the query points may be located inside and outside of it; whereas the returned distance has no sign and does not distinguish between interior and exterior.

The runtimes of these tests are shown in Figure 6. The results indicate some interesting facts. Both the adaptive subdivision technique and the Bézier conversion approach use the same 3D hashed grid structure to identify relevant patches with a grid cell size of $d/\sqrt[3]{p}$, whereas d denotes the AABB diagonal and p the number of patches in the base mesh. The adaptive subdivision depends on the number of relevant patches which correlates with the model's complexity. But the Bézier conversion is rather determined by internal parameters (start values for numerical iterations, etc.) than by model complexity. This calculation overhead is almost independent from the input data and surmounts the time needed by the adaptive subdivision approach several times.

Another interesting point which can be seen in the diagram is the speed-up factor of the first-hit algorithm. Compared with the variant which checks additional grid cells in order to return the exact distance instead of an approximation the first-hit version is three times faster ($\phi \approx 3.09$). Of course, both algorithms use the same grid size. The number of grid cells is proportional to the number of triangles in the tessellation.

While it is normally not recommended to triangulate a subdivision surface ahead of time, the first hit version has similar timings as the adaptive evaluation technique, at least for small- and medium-sized models.

6 Conclusion

According to the benchmarks presented above, the distance between an arbitrary point and a subdivision surface should be determined using an efficient space partitioning technique such as hashed, regular 3D grid and an on-the-fly subdivision surface evaluation algorithm. The result is a distance calculation which

- needs considerably less memory than triangulation based approaches, and
- is the fastest method in most cases.

The only negative point of the adaptive subdivision method is its complex implementation. The conversion method may use numerical libraries and the triangulation methods can use wide-spread, standard techniques, whereas an efficient, on-the-fly evaluation of subdivision surfaces must be implemented efficiently for the mesh structure used.

Therefore, the triangulation-based approach with the first-hit termination might be considered for small model sizes, if the perpendicular point is not needed and if an approximation of the distance is enough. In all other cases the adaptive subdivision technique is the best choice.

7 Acknowledgment

The authors would like to thank RENÉ BERNDT. He created and provided the chess models which have been used for benchmarking.

References

[1] T. Ullrich and D. W. Fellner, "Robust shape fitting and semantic enrichment," *Proceedings of the International Symposium of the International Committee for Architectural Photogrammetry (CIPA) 2007*, vol. 21, p. to appear, 2007.

[2] W. Ma, "Subdivision surfaces for CAD - an overview," *Computer-Aided Design*, vol. 37, no. 7, pp. 693–709, 2005.

[3] E. Catmull and J. Clark, "Recursively generated B-spline surfaces on arbitrary topological meshes," *Computer-Aided Design*, vol. 10, pp. 350–355, 1978.

[4] D. Zorin, P. Schröder, T. DeRose, L. Kobbelt, A. Levin, and W. Sweldens, "Subdivision for Modeling and Animation," *SIGGRAPH 2000 Course Notes*, vol. 1, pp. 1–116, 2000.

[5] K. Müller and S. Havemann, "Subdivision Surface Tesselation on the Fly using a versatile Mesh Data Structure," *Computer Graphics Forum*, vol. 19, no. 3, pp. 151–159, 2000.

[6] J. Stam, "Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values," *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, vol. 1, pp. 395 – 404, 1998.

[7] I. Boier-Martin and D. Zorin, "Differentiable Parameterization of Catmull-Clark Subdivision Surfaces," *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, vol. 71, pp. 155 – 164, 2004.

[8] M. W. Jones, A. J. Baerentzen, and M. Sramek, "3D Distance Fields: A Survey of Techniques and Applications," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 4, pp. 581– 599, 2006.

[9] G. Zachmann and E. Langetepe, "Geometric Data Structures for Computer Graphics," Course notes, EG tutorial, 2002.

[10] M. Botsch, D. Bommes, C. Vogel, and L. Kobbelt, "Gpu-based tolerance volumes for mesh processing," *Proceedings of 12th Pacific Conference on Computer Graphics and Applications*, vol. 12, pp. 237– 243, 2004.

[11] M. W. Jones, "3D Distance from a Point to a Triangle," *Technical Report, Department of Computer Science, University of Wales Swansea*, vol. 5, pp. 1–5, 1995.

[12] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross, "Optimized Spatial Hashing for Collision Detection of Deformable Objects," *Proceedings of Vision, Modeling, and Visualization*, vol. 1, pp. 47–54, 2003.

[13] V. Settgast, K. Müller, C. Fünfzig, and D. W. Fellner, "Adaptive Tesselation of Subdivision Surfaces," *Computers and Graphics*, vol. 28, no. 1, pp. 73 – 78, 2004.

[14] Y. L. Ma and W. T. Hewitt, "Point inversion and projection for nurbs curve and surface: control polygon approach," *Computer Aided Geometric Design*, vol. 20, no. 2, pp. 79–99, 2003.

[15] M. Marinov and L. Kobbelt, "Optimization methods for scattered data approximation with subdivision surfaces," *Graphical Models*, vol. 67, no. 5, pp. 452 – 473, 2005.

[16] I. Selimovic, "Improved algorithms for the projection of points on nurbs curves and surfaces," *Computer Aided Geometric Design*, vol. 23, no. 5, pp. 439–445, 2006.

[17] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *The Computer Journal*, vol. 7, pp. 149–154, 1964.

[18] C. Kelley, *Iterative Methods for Optimization*, ser. Frontier in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM), 1999, vol. 18.

Narrow Band Methods for PDEs on Very Large Implicit Surfaces

Oliver Nemitz[1], Michael Bang Nielsen[2], Martin Rumpf[3], Ross Whitaker[4]

[1,3]University of Bonn, Germany, [2]University of Århus, Denmark, [4]University of Utah, U.S.A.
Email: [1,3]{oliver.nemitz,martin.rumpf}@ins.uni-bonn.de,
[2]bang@daimi.au.dk, [4]whitaker@cs.utah.edu

Abstract

Physical simulation on surfaces and various applications in geometry processing are based on partial differential equations on surfaces. The implicit representation of these eventually evolving surfaces in terms of level set methods leads to effective and flexible numerical tools. This paper addresses the computational problem of how to solve partial differential equations on level sets with an underlying very high-resolution discrete grid. These high-resolution grids are represented in a very efficient format, which stores only grid points in a thin narrow band. Reaction diffusion equations on a fixed surface and the evolution of a surface under curvature motion are considered as model problems. The proposed methods are based on a semi implicit finite element discretization directly on these thin narrow bands and allow for large time steps. To ensure this, suitable transparent boundary conditions are introduced on the boundary of the narrow band and the time discretization is based on a nested iteration scheme. Methods are provided to assemble finite element matrices and to apply matrix vector operators in a manner that do not incur additional overhead and give fast, cache-coherent access to very large data sets.

1 Introduction

This paper addresses the computational problem of how to solve partial differential equations (PDEs) on the level sets of smooth scalar functions that are approximated by very high-resolution discrete grids. The context for this work is the growing interest in computing PDEs on surfaces that are represented implicitly as the level sets of a smooth scalar function ϕ. Starting with the pioneering paper by Osher and Sethian [31] this way has become increasing important in a variety of fields such as computational physics [3,4,6,7,20], scientific visualization [23], image analysis [5, 8], and computer graphics [27, 30]. Most of these applications rely on the efficient computation of partial differential equations on curves or surfaces implicitly represented by a level set function ϕ resolved on a discrete, usually structured, grid. The attraction of solving problems with discretely sampled implicit surfaces is the relatively large number of degrees of freedom provided by the grid and the freedom of not having to choose an explicit surface parameterization, which often limits shape and topology.

There are in particular two scenarios in which such surface-based PDEs are interesting. The first is when the implicit surface serves as the domain and one would like to solve a PDE for a function u intrinsic on the surface. Projections of the derivatives in the ambient space onto the surface provide a mechanism for computing differential operators that live on the surface [5]. The other scenario is when the surface itself evolves according to a geometric PDE that depends on the shape. The most prominent example is motion by mean curvature [16]. For the discretization in space either finite difference [31, 33] or finite element schemes [10] are considered. Semi-implicit time discretizations are suitable due to their stability properties also for large time steps, compared to explicit time discretization for diffusion type problems which require time steps of the grid size squared. This is particularly important when one is considering higher order PDEs [14, 20].

Perhaps the greatest promise of level-set methods, for both moving interfaces and PDEs defined on static surfaces (codimension one), is their ability to deal with a wide variety of complicated shapes in an elegant manner within a single computational framework. However, the computation and memory requirements on the discrete grid that represents ϕ become prohibitive as the grid resolution increases.

The complexity of the surface increases (roughly) as the grid resolution squared, but the overall grid size increases with the cube of the resolution.

Several technical advances have addressed different aspects of the problem associated with storing and computing level-set equations at high resolutions. The introduction of methods that solve PDEs on a small subset of grid points, that constitute a narrow band around the surface [2,32,37] provided significant advantages in computation time. As grid sizes become progressively larger the number of computations in the narrow band is not the limiting factor on performance. Rather, the performance of computations is limited by the very small fraction of the grid values that can fit into cache or random-access memory. To address this several authors have proposed memory-efficient data structures for storing the narrow bands associated with level sets that are represented with large grids.

The use of such narrow bands, which can encode many millions of degrees of freedom, gives the level-set approach to surface representation a *distinct computational advantage* relative to parametric representations, such as triangle meshes. The reason is that with careful attention to how grid points are stored and accessed, the grid-based, implicit method for processing surfaces provides regular, predictable access to memory in a way that allows for cache coherency (on conventional processors) and data streaming on more advanced architectures.

This very narrow computational domain presents a challenge for numerical schemes, however, because one must introduce a solution for the PDE along the boundary domain, whose shape can be quite irregular. As the resolution increases the boundaries of the computational domain become progressively closer to the level set of interest, and the so called *natural* boundary conditions allow artifacts from the grid (whose faces are aligned with the cardinal directions) to propagate into the PDE on the surface. Furthermore, when solving free boundary problems with finite differences, the time steps must be limited so that at each iteration the moving interface (level set of interest) is neither impeded by the boundary conditions nor allowed to pass outside of the computational domain (at which point its shape is lost).

This paper addresses these issues by introducing numerical schemes for finite element solutions to PDEs on implicit surfaces that are appropriate for the Dynamic Tubular Grid (DT-grid) data structure [28] which is storage efficient and fast in practice. In contrast to the finite difference schemes already implemented in this context we consider here finite element methods with semi-implicit schemes in time and introduce the required suitable DT-grid based linear algebra operations on finite element matrices. These numerical schemes introduce transparent boundary conditions together with nested iterations in the time discretization that do not allow the irregularity of the narrow band to impact the solution. In the case of moving interfaces, this allows semi-implicit updates with larger times steps that do not restrict the updated solution to the computational domain from the previous time step. As applications we consider texture synthesis by systems of reaction diffusion equations and the evolution of surfaces by mean curvature motion on very large data sets that are appropriate for state-of-the-art applications in surface processing.

2 Related Work

There are several bodies of related work with respect to narrow band methods and corresponding sparse storage schemes. Narrow band methods for moving interface simulation were first proposed in [2] and proved their efficiency in various applications [18, 27, 38]. Narrow band techniques have been combined with boundary element methods [17] and with multiscale resolution techniques [39]. In [19] the reinitialization of the level set function on a narrow band is discussed and instabilities at the narrow band boundary are avoided by smoothing kernels applied to the level set function. Surface evolution based on the evolution of distance functions on narrow band domains is investigated theoretically in [11]. A heap sorted queue is applied for the adminstration of narrow band data in an active contour method [29]. Already in [12] Marc Droske proposed a finite element method for Willmore flow based on a iterative update scheme on narrow bands. We pick up and refine this type of iterative update scheme here on very thin narrow bands and high resolution background grids.

The work presented also builds on the research in computer science on efficient data structures for storing sparse computational domains associated with level sets. In recent years quadtrees (2D)

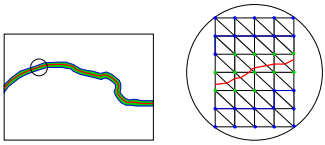


Figure 1: A sketch of a narrow band domain Ω_n corresponding to a level set (plotted in red) is shown. In the zoom in on the right interior nodes are indicated by green dots, whereas the boundary $\partial\Omega_n$ is represented by the blue lines.

and octrees (3D) [9] have been applied to level sets in numerous papers [13, 15, 25, 26, 34]. The quadtree and octree data structures reduce the storage requirements of level sets to $O((d+1)n)$, but also introduce an $O(d)$ access time, where d is the depth of the quadtree or octree and n is the number of grid points in the narrow band. The Dynamic Tubular Grid [28] employs a hierarchical encoding of the topology of the narrow band, inspired by the storage-format of sparse matrices. Subsequent works employ a run-length encoding, and focus either on flexibility [21] or are tailored for a specific application in fluid simulation [22]. All of these data structures require $O(n)$ storage and have $O(1)$ access time to grid points in a local stencil during the sequential access typically required by level set methods. Also they perform faster in practice than recent narrow band and octree approaches due both to the lower memory footprint and the more cache coherent memory layout and access patterns [21]. In this work we utilize the DT-Grid since it has been shown to perform slightly faster than the run-length encoding alternatives.

3 Finite Elements on Narrow Bands

3.1 Review of Level Set Finite Elements

Let us consider the finite-element formulation for reaction-diffusion equations on a fixed surface and a discretization of curvature motion, respectively. We deal with both as model problems for the cases of static and moving surfaces, respectively.

A reaction-diffusion model on level sets. We consider the following scalar initial value problem: The solution is a function $u : \mathbb{R}^+ \times \mathcal{M} \to \mathbb{R}$, such that $\partial_t u - \Delta_{\mathcal{M}} u = f(u)$ with initial condition $u(0) = u^0$, where u^0 is some initial value function on the surface $\mathcal{M}$ and $\Delta_{\mathcal{M}}$ is the Laplace-Beltrami operator on $\mathcal{M}$. We represent the surface $\mathcal{M}$ as the zero set of a smooth scalar function $\phi : \Omega \to \mathbb{R}$, so that $\mathcal{M} = \{x | \phi(x) = 0\}$, where Ω is a box domain enclosing $\mathcal{M}$. The Laplace Beltrami operator of u is expressed in terms of derivatives of ϕ, which gives $\Delta_{\mathcal{M}} u = |\nabla\phi|^{-1} \mathrm{div}(|\nabla\phi| P[\phi] \nabla u)$, where $P[\phi] = \mathbb{I} - \frac{\nabla\phi}{|\nabla\phi|} \otimes \frac{\nabla\phi}{|\nabla\phi|}$ is the projection onto the tangent space $T_x\mathcal{M}$ of the surface $\mathcal{M}$. Now, we first discretize in time and introduce a time derivative $\frac{u^{k+1}-u^k}{\tau}$ for time step functions u^k. Testing the equation with a smooth function θ and applying integration by parts we derive the following time discrete weak formulation:

$$\int_\Omega |\nabla\phi| \frac{u^{k+1}-u^k}{\tau}\theta + |\nabla\phi| P[\phi] \nabla u^{k+1} \cdot \nabla\theta \, \mathrm{d}x = \int_\Omega |\nabla\phi| f(u^k)\theta \, \mathrm{d}x \tag{1}$$

for all test functions $\theta \in C^1$. Here the nonlinear right hand side f is evaluated on the old time step (forward differences). The operator $P[\phi]$ ensures a decoupling of the reaction-diffusion process on different level sets $[\phi = c]$, which reflects the geometric nature of the problem. Thus, to identify the solution on $\mathcal{M}$ it suffices to consider the weak formulation restricted to a small band arround $\mathcal{M}$. Next, we discretize in space based on a finite element approximation. We denote discrete quantities with upper case letters to distinguish them from continuous quantities in lower case letters. The domain Ω is supposed to be covered by a regular hexahedral grid and we denote the corresponding space of continuous, piecewise tri-linear functions by $\mathcal{V}^h$, where h indicates the grid size. Let $\{\Phi_i\}_{i\in\mathbf{I}}$ be the canonical nodal basis of this finite element space for an index set $\mathbf{I}$ corresponding to all grid nodes. A discrete function U is represented as a nodal vector $\bar{U}$ corresponding to nodes of the spatial grid. We achieve the vector $\bar{U} = (U_i)_{i\in\mathbf{I}}$, where $U = \sum_{i\in\mathbf{I}} U_i \Phi_i$ is the corresponding function. Given an approximation $\Phi \in \mathcal{V}_h$ of the level set function ϕ, we obtain an approximation $\mathcal{M}^h := [\Phi = 0]$ of the continuous surface $\mathcal{M}$ as one particular discrete level set represented by the function Φ. Concerning the reaction-diffusion model, we replace all continuous quantities in (1) by their discrete counterparts and introduce mass lumping. Thus, we define the weighted lumped mass and stiffness matrix

$$\mathbf{M}[\Phi] = \left(\int_\Omega \mathcal{I}_h^0(|\nabla\Phi|) \mathcal{I}_h^1(\Phi_i\Phi_j) \, \mathrm{d}x \right)_{i,j\in\mathbf{I}},$$

$$\mathbf{L}[\Phi] = \left(\int_\Omega |\nabla\Phi| P[\Phi] \nabla\Phi_i \cdot \nabla\Phi_j \, \mathrm{d}x \right)_{i,j\in\mathbf{I}},$$

where $\mathcal{I}_h^0$, $\mathcal{I}_h^1$ denote the piecewise constant and the piecewise multilinear Lagrangian projection, respectively. Furthermore, we introduce the right hand side vector $\bar{F}[U] = (f(U_i))_{i\in\mathbf{I}}$ and end up with the system of linear equations

$$(\mathbf{M}[\Phi] + \tau\mathbf{L}[\Phi])\,\bar{U}^{k+1} = \mathbf{M}[\Phi]\left(\tau\bar{F}[U^k] + \bar{U}^k\right).$$

Solving these systems we iteratively compute $(U^k)_{k\geq 1}$ for a given approximation U^0 of u^0.

Curvature Motion of Level Sets. The second application considered in this paper is the evolution of surfaces under mean curvature motion. Given an initial surface $\mathcal{M}_0$ we ask for a family of surfaces $\mathcal{M}(t)$ generated from the motion of points $x(t)$ under the evolution $\dot{x}(t) = -h(t)n(t)$ with initial condition $x(0) = x_0$ with $x_0 \in \mathcal{M}_0$. Here $n(t)$ is the normal and $h(t)$ the mean curvature on $\mathcal{M}(t)$. The corresponding level set equation is given by $\partial_t\phi - |\nabla\phi|\mathrm{div}\left(\frac{\nabla\phi}{|\nabla\phi|}\right) = 0$ on $\mathbb{R}^+ \times \Omega$ with initial data ϕ_0. Again discretizing in time and applying integration by parts we obtain the weak formulation

$$\int_\Omega \frac{\phi^{k+1} - \phi^k}{\tau|\nabla\phi^k|_\epsilon}\theta + \frac{\nabla\phi^{k+1}}{|\nabla\phi^k|_\epsilon} \cdot \nabla\theta \, \mathrm{d}x = 0 \qquad (2)$$

for test functions $\theta \in C^1$. Here, we take into account the old time step solution for the weight $|\nabla\phi|^{-1}$ and the usual regularization $|x|_\epsilon = \sqrt{\epsilon^2 + |x|^2}$. Now, as in the case of the reaction-diffusion equation on a fixed surface we discretize in space and again end up with a sequence of linear systems of equations

$$\left(\mathbf{M}[\Phi^k] + \tau\mathbf{L}[\Phi^k]\right)\bar{\Phi}^{k+1} = \mathbf{M}[\Phi^k]\bar{\Phi}^k$$

for the nodal vector $\bar{\Phi}^{k+1}$ of the discrete level set function at time $t_{k+1} = \tau(k+1)$. Here, the involved lumped mass and stiffness matrices are given by

$$\mathbf{M}[\Phi] = \left(\int_\Omega \mathcal{I}_h^0(|\nabla\Phi|_\epsilon^{-1})\mathcal{I}_h^1(\Phi_i\Phi_j) \, \mathrm{d}x \right)_{i,j\in\mathbf{I}},$$

$$\mathbf{L}[\Phi] = \left(\int_\Omega |\nabla\Phi|_\epsilon^{-1} \nabla\Phi_i \cdot \nabla\Phi_j \, \mathrm{d}x \right)_{i,j\in\mathbf{I}}.$$

Solving these systems we iteratively compute $(\Phi^k)_{k\geq 1}$ for a given approximation Φ^0 of ϕ^0 and obtain a sequence of discrete surfaces $\mathcal{M}_h^k = [\Phi^k = 0]$.

3.2 Transparent Neumann Boundary

The continuous formulation operates on the solution of each level-set separately, and thus, solutions from different level sets do not interact and we can truncate the computational domain to a narrow band around the zero set without affecting the solution. However, the discrete formulation introduces a coupling of nearby level sets through the finite extent of the test/basis functions and the natural boundary conditions induced by the weak formulation interfere strongly with the solution on $\mathcal{M}_h$ in case of a thin narrow band. This interaction undermines the numerical convergence of the scheme on finer grids. In this section we will describe boundary conditions which avoid this interference.

Given a level set surface $\mathcal{M}_h$ for a level set function $\Phi \in \mathcal{V}^h$ we define a discrete narrow band as a union of supports of discrete basis functions. Hence, we consider a corresponding index set $\mathbf{I}_{\text{int}} := \{i \in \mathbf{I} \,|\, \operatorname{supp}\Phi_i \cap \mathcal{M}^h \neq \emptyset\}$ and the resulting narrow band domain $\Omega_{\text{n}} = \bigcup_{i\in\mathbf{I}_{\text{int}}} \operatorname{supp}\Phi_i$. This is the smallest possible band to resolve the discrete surface $\mathcal{M}_h$. Let us denote by $\mathcal{V}_{\text{int}}^h = \operatorname{span}\{\Phi_i \,|\, i \in \mathbf{I}_{\text{int}}\}$ the space of discrete functions on Ω_{n} which vanish on $\partial\Omega$ and by $\mathcal{V}_{\text{bd}} = \operatorname{span}\{\Phi_i \,|\, i \in \mathbf{I}\setminus\mathbf{I}_{\text{int}},\ \operatorname{supp}\Phi_i \cap \Omega_{\text{n}} \neq \emptyset\}$ the discrete function space corresponding to boundary values on $\partial\Omega_{\text{n}}$. Hence, the direct sum $\mathcal{V}_{\text{n}}^h = \mathcal{V}_{\text{int}}^h \oplus \mathcal{V}_{\text{bd}}^h$ represents the discrete finite element space corresponding to the narrow band domain Ω_{n}. Now, we replace the domain of integration in the weak formulations by the narrow band domain.

For the **reaction diffusion model** integration by parts leads to the boundary integral $\int_{\partial\Omega_{\text{n}}} |\nabla\phi| P[\phi]\nabla u^{k+1} \cdot \nu\theta \, \mathrm{d}a$ on $\partial\Omega_{\text{n}}$, which gives rise to the Neumann boundary condition $P[\phi]\nabla u^{k+1} \cdot \nu = 0$. This condition is meaningless and for $P[\phi]\nu \neq 0$ artificially couples the gradient of the solution u^{k+1} with the faceted, grid-aligned (jaggy) boundary of the narrow band domain. Let us suppose that a good estimate u_{approx}^{k+1} of the solution u^{k+1} is given. Then, we could compensate for this defect adding the above boundary with the unknown u^{k+1} replaced by the known approximation u_{approx}^{k+1} on the right hand side of the weak formula-

tion (cf. [12]). For the finite element discretization we obtain the corresponding correction vector

$$\bar{\Gamma}[U_{\text{approx}}] = \left(\int_{\partial\Omega_{\text{n}}} |\nabla\Phi| P[\Phi] \nabla U_{\text{approx}}^{k+1} \cdot \nu \Phi_j \, \mathrm{d}a \right)_{j \in \mathbf{I}}$$

for a given approximation U_{approx}^{k+1} of U^{k+1} on the right hand side of the modified system of linear equations

$$(\mathbf{M}[\Phi] + \tau \mathbf{L}[\Phi])\, \bar{U}^{k+1} = \mathbf{M}[\Phi] \left(\tau \bar{F}[U^k] + \bar{U}^k \right) + \tau \bar{\Gamma}[U_{\text{approx}}^{k+1}] \,.$$

Here $\bar{F}$ is the nodal vector in $\mathbb{R}^{\mathbf{I}_{\text{ext}}}$ corresponding to the right hand side f. A first possible choice for the approximation is given by the last time step, i. e. we may set $U_{\text{approx}}^{k+1} = U^k$.

For the discrete **mean curvature motion** we can proceed similarly. The natural boundary condition implied by the weak formulation is $|\nabla\Phi^k|_\epsilon^{-1} \nabla\phi^{k+1} \cdot \nu\theta = 0$. Hence, given an approximation $\phi_{\text{approx}}^{k+1}$ of the unknown ϕ^{k+1}, we again compensate for this defect adding the boundary integral $\int_{\partial\Omega_{\text{n}}} |\nabla\phi^k|_\epsilon^{-1} \phi_{\text{approx}}^{k+1} \cdot \nu\theta \, \mathrm{d}a$ on the right hand side of the weak equation. For the finite element discretization we correspondingly consider the correction vector

$$\bar{\Gamma}[\Phi_{\text{approx}}^{k+1}, \Omega_{\text{n}}] = \left(\int_{\partial\Omega_{\text{n}}} |\nabla\Phi^k|_\epsilon^{-1} \nabla\Phi_{\text{approx}}^{k+1} \cdot \nu\Phi_j \, \mathrm{d}a \right)_{j \in \mathbf{I}}$$

and obtain the modified system to be solved:

$$(\mathbf{M}[\Phi] + \tau \mathbf{L}[\Phi])\, \bar{\Phi}^{k+1} = \mathbf{M}[\Phi] \bar{U}^k + \tau \bar{\Gamma}[\Phi_{\text{approx}}^{k+1}, \Omega_{\text{n}}] \tag{3}$$

Again the old time step solution may serve as a first approximation of the unknown Φ^{k+1}.

3.3 Transparent Dirichlet Boundary

So far, we have focused on Neumann type boundary conditions. In particular in case of mean curvature motion one might alternatively consider Dirichlet conditions. We propose boundary data which is coherent with a suitable, smooth extension of the unknown level set function ϕ^{k+1} in time step $k+1$. Here a signed distance function from the current discrete surface is a good approximation. To present the discrete scheme in matrix vector notation, we exploit the introduced splitting of the finite element space $\mathcal{V}_{\text{n}}^h = \mathcal{V}_{\text{int}}^h \oplus \mathcal{V}_{\text{bd}}^h$. Thus, reordering degrees of freedom we obtain a splitting $\bar{U}_{\text{n}} = (\bar{U}_{\text{int}}, \bar{U}_{\text{bd}})$, where $U_{\text{int}} \in \mathcal{V}_{\text{int}}^h$ and $U_{\text{bd}} \in \mathcal{V}_{\text{bd}}^h$. Correspondingly, we obtain a splitting of the stiffness with respect to V_{int} and V_{bd} by $\mathbf{L} = \begin{pmatrix} \mathbf{L}_{\text{int,int}} & \mathbf{L}_{\text{bd,int}} \\ \mathbf{L}_{\text{int,bd}} & \mathbf{L}_{\text{bd,bd}} \end{pmatrix}$. Here $\mathbf{L}_{\text{int,int}}$ is the actual stiffness matrix on V_{int}. Furthermore, let us introduce a trivial extension operator $\mathbf{E} : \mathbb{R}^{\mathbf{I}_{\text{int}}} \to \mathbb{R}^{\mathbf{I}_{\text{n}}}$; $\bar{U}_{\text{n}} \mapsto (\bar{U}_{\text{int}}, 0)$ and the corresponding restriction operator $\mathbf{R} : \mathbb{R}^{\mathbf{I}} \to \mathbb{R}^{\mathbf{I}_{\text{n}}}$; $\bar{U} \mapsto \bar{U}_{\text{int}}$. Based on this notation we can rewrite $L_{\text{int,int}} = RLE$ and hence the linear system to be solved in case of Dirichlet data $\Phi_{\text{approx}}^{k+1}$ is $\mathbf{R}(M[\Phi^k] + \tau \mathbf{L}[\Phi^k]) \mathbf{E} \bar{\Phi}_{\text{int}}^{k+1} = \mathbf{R}(M\bar{\Phi}^k - \tau \mathbf{L}[\Phi^k] \bar{\Phi}_{\text{approx}}^{k+1})$. The practical consequence is that we always work with the full matrix $\mathbf{L}$ and do not explicitly extract $\mathbf{L}_{nn}$ from it. Boundary data and solution vector are stored in one vector in $\mathbb{R}^{\mathbf{I}_{\text{n}}}$.

3.4 Solver Based on Nested Iterations

Here, we discuss the scheme for mean curvature motion, which requires special care because of the iterative update of the computational domain. The corresponding scheme for the reaction diffusion model on a fixed narrow band is a special case.

The inner iterations must modify the boundary conditions *and* the computational domain, as the solution moves toward the edge of the narrow band. Thus, the inner iterations compute the new Φ^{k+1} relative to the old Φ^k using Eq. 3. After each inner iteration we compute a new distance map to the zero set of the new Φ^{k+1}. This redistancing rebuilds the DT-grid (to a specified width, as described in [28]). If the narrow band changes, we i) extend the old solution onto the new band using the signed distance transform (Eikonal equation) from the previous domain and ii) repeat the inner iteration with Dirichlet conditions using the distance field to Φ^{k+1} on the boundary. That is, the domain extension $\mathcal{E}[\Omega_{\text{n}}, \tilde{\Omega}_{\text{n}}]\bar{\Phi}$ from a narrow band Ω_{n} onto a new band $\tilde{\Omega}_{\text{n}}$ is the discrete solution of the Eikonal equation $|\nabla\phi| = 1$ with boundary data $\bar{\Phi}$ on the inner nodes of the band Ω_{n}. If the computational domain does not change, we simply redistance, update the boundary conditions, and solve. These inner iterations repeat until the change from one iteration to the next falls below a threshold.

This scheme allows for large time steps and discrete surfaces propagating significantly outside the initial narrow band in this time step. In each time step we compute intermediate solutions $\Phi^{k+1,j}$ and intermediate narrow band domains $\Omega_{\text{n}}^{k+1,j}$. The ex-

tension operator $\mathcal{E}[\Omega_{\mathrm{n}}^{k+1,j}, \Omega_{\mathrm{n}}^{k+1,j+1}]$ ensures that the previous time step solution Φ^k as well as the successively updated solution $\Phi^{k+1,j}$ itself is extended onto the new band. In pseudo code notation the scheme looks as follows:

MeanCurvatureMotion($\bar{\Phi}^0$) {
initialize Ω_{n}^0;
for ($k = 1; k \leq K; k++$) {
  $\bar{\Phi}^{k+1,0} = \Phi^k; j = 0; \Omega_{\mathrm{n}}^{k+1,0} = \Omega_{\mathrm{n}}^k$;
  do {
    compute $\bar{\Phi}^{k+1,j+1}$ on $\Omega_{\mathrm{n}}^{k+1,j}$ solving
      $\mathbf{R}(M[\Phi^k] + \tau\mathbf{L}[\Phi^k])\mathbf{E}\bar{\Phi}_{\mathrm{int}}^{k+1,j+1}$
        $= \mathbf{R}(M[\Phi^k] - \tau\mathbf{L}[\Phi^k]\bar{\Phi}^{k+1,j})$;
    Define new band $\Omega_{\mathrm{n}}^{k+1,j+1}$ for $\bar{\Phi}^{k+1,j+1}$;
    Apply $\mathcal{E}[\Omega_{\mathrm{n}}^{k+1,j}, \Omega_{\mathrm{n}}^{k+1,j+1}]$ to $\bar{\Phi}^{k+1,j+1}, \Phi^k$;
    $j = j + 1$;
  } while($|\bar{\Phi}^{k+1,j} - \bar{\Phi}^{k+1,j-1}| \geq \delta$
      or $\Omega_{\mathrm{n}}^{k+1,j} \neq \Omega_{\mathrm{n}}^{k+1,j-1}$)
  $\bar{\Phi}^{k+1} = \Phi^{k+1,j}; \Omega_{\mathrm{n}}^{k+1} = \Omega_{\mathrm{n}}^{k+1,j}$;
} }

The procedure in the case of Neumann conditions imposed on the boundary of the narrow band is completely analogues. We just exchange the linear system to be solved. The Dirichlet boundary conditions ensure that the new interface $[\Phi^{k+1,j+1} = 0]$ is a subset of the current narrow band $\Omega_{\mathrm{n}}^{k+1,j}$. This is no longer true in case of Neumann boundary conditions. Indeed, the discrete interface may cross $\partial\Omega_{\mathrm{n}}^{k+1,j}$. Hence, before we are able to define the new band, we have to extend $\Phi^{k+1,j+1}$ until we resolve the zero level set.

4 Narrow Band on the DT-Grid

In this section we describe how the proposed narrow band algorithms can be implemented on the Dynamic Tubular Grid (DT-Grid) data structure [28] in order to obtain a framework that is efficient both with regard to memory and time utilization. The DT-Grid is a data structure and set of algorithms designed for storing data of a subset of nodes or elements defined on a regular grid. Constant time access and cache performance for neighborhood operations are achieved through the careful use of *iterators*, which are used to build, store, and manipulate the solution and all of the associated matrices/vectors. We begin with a brief overview of the DT-Grid terminology required to comprehend the exposition of the implementation issues, and next we describe how to implement the proposed narrow band algorithms in the DT-Grid framework.

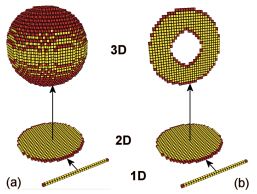


Figure 2: a) The 1D, 2D and 3D components of the DT-Grid encoding of a sphere. b) A slice of the narrow band of a DT-Grid encoding of a sphere.

4.1 Dynamic Tubular Grid Terminology

The nodes in the narrow band are stored in the DT-Grid in (x, y, z) lexicographic order which allows for a number of specific algorithmic constructs. In order to represent the topology of the narrow band, a 3D DT-Grid consists of 1D, 2D and 3D grid components as shown in Figure 2a. The 3D grid component consists of the nodes in the narrow band, the 2D grid component is the projection of the narrow band onto the XY-plane, and the 1D grid component is the projection of the 2D grid component onto the X-axis. For a full explanation of the DT-Grid we refer the reader to [28]. Here it is sufficient to say that each grid component has two constituents: $data$ and $coord$. The $coord$ constituent in the nD grid component stores the nth coordinate of the first and last node in each topologically connected component of grid points in a column of the nD grid component. These are colored red in Figure 2. As also depicted in Figure 2b, the $data_{1D}$ and $data_{2D}$ constituents link the 1D, 2D and 3D grid components together by storing indices that point to the first coordinate in a column in the $coord$ constituent of the 2D and 3D grid components respectively.

We denote the $coord_{1D}$, $coord_{2D}$, $coord_{3D}$, $data_{1D}$ and $data_{2D}$ constituents of the *topology* since they specify the topology of the narrow band. The $data_{3D}$ constituent contains the actual data values, e.g., a level set function, and is stored separately from the topology in a flat data vector of length equal to the number of nodes in the narrow band. Since a total (lexicographic) ordering, starting from zero, is imposed on the nodes in the narrow band, entry i in the data vector corresponds uniquely to node i in the narrow band. In fact, traversing the entries in the data vector sequentially from start to end corresponds to accessing the data of all nodes in the narrow band in lexicographic or-

der. This also means that storing multiple data items at each node in the narrow band can be done by allocating multiple separate data vectors. Entry i in each of these data vectors then identify the data stored at node i in the narrow band.

The DT-Grid utilizes the concept of *stencil iterators* to sequentially access each individual node of the narrow band and provide constant time access to the node's neighbors as defined by a stencil suited for some computational task. In particular a specific stencil iterator consists of M individual iterators, where M is the number of nodes in the stencil, and an iterator is simply a construct that sequentially visits all grid points of the narrow band in lexicographic order. Each iterator provides constant time access to the appropriate data items. Details are given in [28].

4.2 DT-Grid Implementation

The applications of the narrow band framework proposed in this paper require the definition of a narrow band level set function as well as a number of vectors and matrices defined over this narrow band. The vectors contain an entry for each node in the narrow band, and the matrices are defined over the cardinal product of the narrow band with itself. However, the matrices are sparse and banded due to the limited support of the nodal basis functions employed in the finite element method.

The narrow band, vectors and sparse matrices are represented as a single instance of a DT-Grid topology and a number of flat data vectors. We create a number of customized stencil iterators that compute boundary face integration on the narrow band, matrix-vector multiplication, and mass and stiffness matrix assembly with the stencil iterator framework.

The narrow band mesh used in our proposed framework is defined in terms of finite elements. Assembling the mass and stiffness matrices require an iteration over these elements, whereas the stencil iterators of the DT-Grid visit all the nodes. However, an iterator that sequentially visits all elements of the narrow band can be phrased as a stencil iterator with a stencil of eight nodes that form a finite element cell. The iterator of the lexicographically smallest node in the stencil (corner) dictates the movement of the stencil, and the stencil iterator skips a node whenever at least one of the seven remaining nodes in the stencil are outside the narrow band. Similarly an iterator that sequentially visits all boundary faces of the narrow band can be phrased as a stencil iterator.

5 Applications

We begin with the generation of a texture. Therefore we solve the initial value problem for two functions $a, b : \mathbb{R}^+ \times \mathcal{M} \rightarrow \mathbb{R}$ where $\mathcal{M}$ is the 0-isosurface of a level set function ϕ. Reaction-diffusion equations describe a variety of biological and chemical phenomenon, but have been used in 3D graphics for the generation of interesting, natural-looking textures on surface [35, 36]. The form we use in this paper, as way of demonstrating the proposed numerical scheme, are the equations proposed by Turing:

$$\begin{aligned} \partial_t a &= c_s(\alpha - ab) + c_a \Delta_{\mathcal{M}} a \\ \partial_t b &= c_s(ab - b - \beta) + c_b \Delta_{\mathcal{M}} b \end{aligned}$$

Generally, c_s, c_a, c_b, and α are parameters that determine the shapes, frequencies, sizes, etc. of the resulting texture (steady state) and $\beta : \mathcal{M} \rightarrow \mathbb{R}$ is a stochastic function (e.g. generated through a pseudo-random number generator), that creates a degree of randomness in the texture.

We use the weak formulation combined with a forward difference scheme for the reaction terms and an implicit scheme for the diffusion.

Figure 3 (top) shows the solution of a reaction-diffusion equation on a 3D model of a dragon [1], which has been scan-converted to a DT-Grid, using the method in [21], with a volumetric representation of $982 \times 695 \times 442$ grid points. With the reaction-diffusion quantities a and b and the mass and stiffness matrices, the full volumetric problem would not be solvable at this resolution on a conventional computer. The parameters, given in the caption, have been choosen to produce spots. The renderings at different levels of resolution demonstrate the difference in scale between the full model and the underlying grid.

Several authors have proposed anisotropic reaction-diffusion methods for anisotropic textures. For that purpose we replace the isotropic diffusion operator $\Delta_{\mathcal{M}} u$ by an anisotropic version, defined by $\tilde{\Delta}_{\mathcal{M}} u = |\nabla\phi|^{-1} \mathrm{div}(|\nabla\phi| D P[\phi] \nabla u) = |\nabla\phi|^{-1} \mathrm{div}(|\nabla\phi| \big(\tilde{\alpha}^2 P[v] + \tilde{\beta}^2 P[v^{\perp}]\big) P[\phi] \nabla u)$. Figure 3 (bottom) shows solutions to the anisotropic

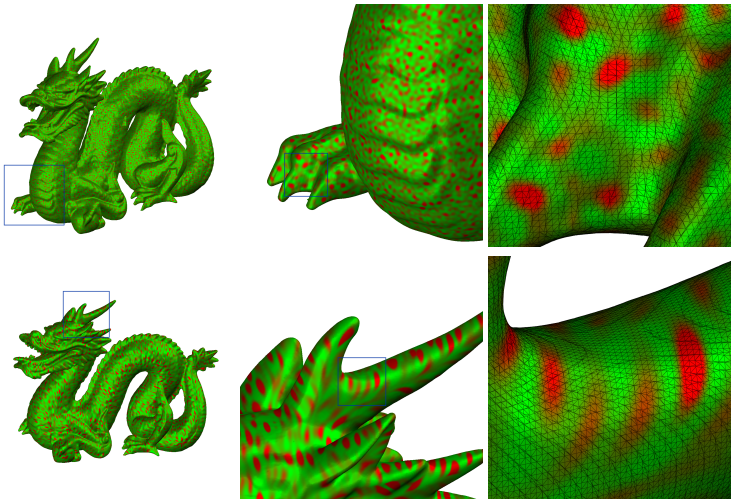


Figure 3: Results of the reaction-diffusion. (Top) Isotropic reaction-diffusion that generates a pattern with round spots after 150 iterations. (Left) A picture of the whole dragon of resolution $982 \times 695 \times 442$. Parameters are $c_s = 0.05$, $c_a = 2.5e-07$, $c_b = 6.3e-08$, $\alpha = 16$, and $\beta = 12 \pm 0.4$, with grid spacing $h = 0.00102$. (Middle) A zoom to the feet region is shown, and (right) an even closer zoom to the foot overlaid with the mesh shows the fineness of the resolution. Bottom row: The same dragon surface with two zooms as result of an anisotropic reaction diffusion after 400 iterations with $v = (0, 0, 1)^T$, and diffusion in the orthogonal direction at one-fourth the main direction. (and otherwise the same parameters as the isotropic case).

reaction-diffusion equation with dominant diffusion in the z direction.

Another demonstration of the framework is the use of mean curvature motion on surfaces, which has been proposed for denoising (fairing) models that are derived from measured surface data. Various extensions of the approach, for both parametric and implicit surfaces, have been proposed to preserve high-curvature features. Figure 4 shows results for mean curvature motion for a scan converted model (DT-grid of size $2471 \times 1439 \times 827$) of the *Lucy Statue* [1]. The full 3D grid of level-set data, stored as floats, would require almost 11 gigabytes of data, whereas the DT-Grid representation of the model requires roughly 159MB. These results demonstrate different levels of blurring rendered from different distances, in order to demonstrate the smoothing of small-scale features on very large models.

Because of the very large sizes of these models, the run times in the current version are still significant. The reaction-diffusion results required approximately 10 minutes (11.5 minutes in the anisotropic version) per timestep on an Intel Pentium 3.6 GHz processor. The mean-curvature results required roughly 8 minutes per timestep for the Lucy Statue [1] in Figure 4 and about 5.5 minutes per timestep for the Asian Dragon [1] in Figure 5. However, these are very large models, which are not solvable without the very narrow band offered by the DT-grid and associated numerical schemes.

Acknowledgements. The narrow band approach for higher order geometric evolution problems in the PhD thesis of Marc Droske inspired this work. We thank Marc Droske and Martin Burger for many discussion and fruitful remarks on narrow band methods. We also thank Ken Museth and Ola Nilsson for permission to use their software. This work was partially supported by Deutsche Forschungsgemeinschaft through the Collaborative Research Center 611 *Singular phenomena and scaling in mathematical models*, the Hausdorff Center for Mathematics at Bonn University and the Danish Agency for Science, Technology and Innovation.

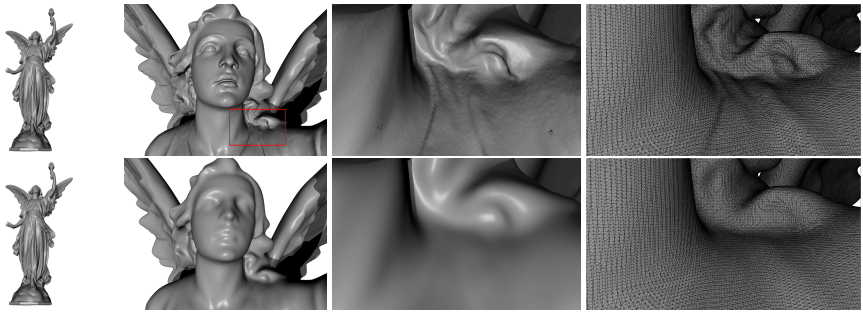


Figure 4: Results of the mean curvature motion on the lucy statue (scan converted to a DT-grid of size $2471 \times 1439 \times 827$). Initial surface (Top) and after 19 timesteps with $\tau = h$ (Bottom). The first image shows the whole statue, the second a first zoom into the head region, the third depicts an even closer zoom to the neck and the last image shows the belonging mesh generated using the marching cubes algorithm [24].

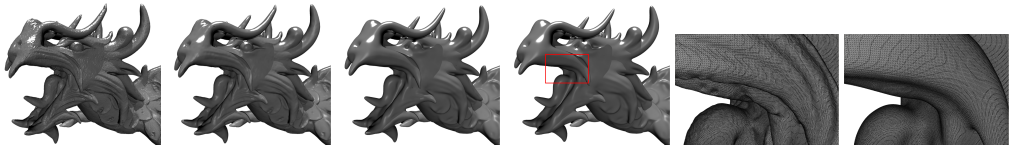


Figure 5: Evolution of the mean curvature motion on the asian dragon surface ($1986 \times 1323 \times 1104$). Here a closeup to the head with timesteps 0,6,30,44 ($\tau = h^2$) is shown. The last two images show the mesh, generated using the marching cubes algorithm [24], of an even closer zoom to the tongue at timesteps 0 and 44.

References

[1] Stanford scanning repository. http://graphics.stanford.edu/data/3Dscanrep/.

[2] D. Adalsteinsson and J. A. Sethian. A fast level set method for propagating interfaces. *Journal of Computational Physics*, 118(2):269–277, 1995.

[3] D. Adalsteinsson and J. A. Sethian. A level set approach to a unified model for etching, deposition, and lithography III: Re-deposition, re-emission, surface diffusion, and complex simulations. *Journal of Computational Physics*, Volume 138, Issue 1:193–223, 1997.

[4] H. Ben Ameur, M. Burger, and B. Hackl. Level set methods for geometric inverse problems in linear elasticity. *Inverse Problems*, 20(3):673–696, 2004.

[5] M. Bertalmío, F. Mémili, L. T. Cheng, G. Sapiro, and S. Osher. *Geometric level set methods in imaging, vision, and graphics*, chapter Variational Problems and Partial Differential Equations on Implicit Surfaces: Bye Bye Triangulated Sufaces?, pages 381–397. Springer, New York, 2003.

[6] M. Burger. A framework for the construction of level set methods for shape optimization and reconstruction. *Interfaces and Free Boundaries*, 5:301–329, 2003.

[7] S. Chen, B. Merriman, M. Kang, R. E. Caflisch, C. Ratsch, C. L.-T., M. Gyure, R. P. Fedkiw, and S. Osher. A level set method for thin film epitaxial growth. *Journal of Computational Physics*, 167:475 – 500, January 2001.

[8] D. Cremers, M. Rousson, and R. Deriche. A review of statistical approaches to level set segmentation: Integrating color, texture, motion and shape. *International Journal of Computer Vision*, 72(2):195–215, 2007.

[9] M. de Berg. *Computational Geometry*. Springer, January 2000.

[10] K. Deckelnick, G. Dziuk, and C. M. Elliott. Computation of geometric partial differential equations and mean curvature flow. *Acta Numerica*, 14:139–232, 2005.

[11] M. Delfour and J. Zolésio. Oriented distance funtion and its evolution equation for initial sets with boundary. *SIAM Journal on Control and Optimization*, 42, No. 6:2286 – 2304, 2004.

[12] M. Droske. *On Variational Problems and Gradient Flows in Image Processing*. Dissertation, University Duisburg, 2005.

[13] M. Droske, B. Meyer, M. Rumpf, and C. Schaller. An adaptive level set method for medical image

segmentation. *Lecture Notes in Computer Science*, pages 412–422, 2001.

[14] M. Droske and M. Rumpf. A level set formulation for willmore flow. *Interfaces and Free Boundaries*, 6(3):361–378, 2004.

[15] D. Enright, F. Losasso, and R. Fedkiw. A fast and accurate semi-lagrangian particle level set method. *Computers and Structures*, 83:479–490, Feb. 2005.

[16] L. Evans and J. Spruck. Motion of level sets by mean curvature I. *Journal of Differential Geometry*, 33(3):635–681, 1991.

[17] M. Garzon, D. Adalsteinsson, L. Gray, and J. A. Sethian. A coupled level set-boundary integral method for moving boundary simulations. *Interfaces and free boundaries*, 7 (3):277–302, 2005.

[18] R. Goldenberg, R. Kimmel, E. Rivlin, and M. Rudzsky. Fast geodesic active contours. *IEEE Transactions on Image Processing*, 10:1467–75, October 2001.

[19] P. Gomez, J. Hernandez, and J. Lopez. On the reinitialization procedure in a narrow-band locally refined level set method for interfacial flows. *International Journal for Numerical Methods in Engineering*, 63 (10):1478–1512, 2005.

[20] J. B. Greer, A. L. Bertozzi, and G. Sapiro. Fourth order partial differential equations on general geometries. Ucla computational and applied mathematics reports, University of California Los Angeles, 2005. Journal of Computational Physics, Volume 216 , Issue 1, Pages: 216 - 246, Year of Publication: 2006.

[21] B. Houston, M. Nielsen, C. Batty, O. Nilsson, and K. Museth. Hierarchical RLE Level Set: A Compact and Versatile Deformable Surface Representation. *ACM Transactions on Graphics*, 25(1):1–24, 2006.

[22] G. Irving, E. Guendelman, F. Losasso, and R. Fedkiw. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. *ACM Transactions on Graphics (SIGGRAPH)*, Aug. 2006.

[23] A. E. Lefohn, J. M. Kniss, C. D. Hansen, and R. T. Whitaker. A streaming narrow-band algorithm: Interactive computation and vusualization of level sets. *IEEE Transactions on Visualization and Computer Graphics, Juli-August 2004*, 10 (4):422–433, 2004.

[24] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA, 1987. ACM Press.

[25] F. Losasso, R. Fedkiw, and S. Osher. Spatially adaptive techniques for level set methods and incompressible flow. *Computers and Fluids*, 35:995–1010, 2006.

[26] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics*, 23(3), Aug. 2004.

[27] K. Museth, D. Breen, R. Whitaker, S. Mauch, and D. Johnson. Algorithms for interactive editing of level set models. *Computer Graphics Forum*, 24 (4):821–841, 2005.

[28] M. B. Nielsen and K. Museth. Dynamic Tubular Grid: An efficient data structure and algorithms for high resolution level sets. *Journal of Scientific Computing*, 26(3):261–299, 2006. (submitted November, 2004; accepted January, 2005).

[29] B. Nilsson and A. Heyden. A fast algorithm for level set-like active contours. *Pattern Recognition Letters*, 24(9-10):1331–1337, 2003.

[30] S. J. Osher and R. P. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, 2002.

[31] S. J. Osher and J. A. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on Hamilton–Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[32] D. Peng, B. Merriman, S. Osher, H. Yhao, and M. Kang. A pde-based fast local level set method. *Journal of Computational Physics*, 155(2):410–438, 1999.

[33] P. Smereka. Semi-implicit level set methods for curvature and surface diffusion motion. *Journal of Scientific Computing*, 19 (1 – 3):439 – 456, 2003.

[34] J. Strain. Tree methods for moving interfaces. *Journal of Computational Physics*, 151(2):616–648, 1999.

[35] A. M. Turing. The chemical basis of morphogenesis. *Phil. Trans. Roy. Soc. London*, B 237:37–72, 1952.

[36] G. Turk. Re-tiling polygonal surfaces. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 25, No. 4, 1991.

[37] R. T. Whitaker. A level-set approach to 3D reconstruction from range data. *International Journal of Computer Vision*, 29(3):203–231, 1998.

[38] M. H. Xu, P. M. Thompson, and A. W. Toga. An adaptive level set segmentation on a triangulated mesh. *IEEE Transactions on Medical imaging*, 23 (2):191–201, 2004.

[39] G. Zheng, J. Feng, X. Jin, and Q. Peng. Adaptive level set method for mesh evolution. *Proceedings Lecture Notes in Computer Science*, 3942:1094–1097, 2006.

ACUT: Out-Of-Core Delaunay Triangulation of Large Terrain Data Sets

Josef Kohout, Ivana Kolingerová

Centre of Computer Graphics and Data Visualization, University of West Bohemia, Plzeň, Czech Republic
Email: {besoft, kolinger}@kiv.zcu.cz

Abstract

In the last couple of years, very detailed high-resolution terrain data sets have become available thanks to new acquisition techniques, e.g., the airborne laser scanning. Such data sets contain, typically, several millions of points and, therefore, several gigabytes are required just to store them, which disallows their loading into the memory of a common computer. In this paper, we propose a novel out-of-core technique for construction of the Delaunay triangulation of such large data sets. It is based on the method of incremental insertion with flipping that is simple, robust and can be easily extended for weights of points, constraints, etc. The proposed technique was tested on various data sets with sizes up to 128M points on a commodity hardware (P4 3.2GHz, 2GB RAM, 250GB SATA disk). The largest data set was processed in about 2.5 hours.

1 Introduction

Given a point set S, the Delaunay triangulation [16] in E^2 is a triangulation which satisfies the Delaunay criterion for each triangle: the circum-circle of the triangle does not contain any input point $p \in S$ in its interior. One of the most important properties of the Delaunay triangulation is that it maximizes the minimal angle and, therefore, it contains the most equiangular triangles of all triangulations (i.e., it limits the number of too narrow triangles that may cause problems in further processing). Due to its properties [21], the Delaunay triangulation is used in many research areas, e.g., in terrain modeling (GIS) [22], scientific data visualization and interpolation [17], [5], [36], pattern recognition [38], meshing for finite element methods (FEM) [15], [8], [34], natural sciences [33], [2], computer graphics and multimedia [17], [35], etc.

In the past, many researchers focused on the problem how to compute the Delaunay triangulation in a reasonable time. The pressure was put on the exploitation of sophisticated data structures to speed up the computation, e.g., Directed Acyclic Graph [9], hierarchical tree [19], multi level uniform grid [40], skip lists [39], etc. Many parallel solutions have been also developed; most of them, e.g., [13], [30] were designed for specialized parallel architectures, often with hundreds of processors. Parallel algorithms suitable for low-cost clusters of workstations also exist, e.g., [14], [29], [25].

Modern computer architectures allow us to compute the Delaunay triangulation of data sets with thousands of points by any of existing sequential algorithms in a reasonable time. The problem is that current data sets are, due to the progress in acquisition techniques, much larger, e.g., typical data sets produced by airborne laser scanning contain several millions of uniformly distributed points, and they cannot be, therefore, loaded into the physical memory of a common computer - either because the computer is a 32-bits architecture that is able to address up to 4GB of memory only, or simply because it is not equipped by the requested amount of memory. Majority of popular sequential or parallel algorithms thus fail to process these data sets.

Hardwick [25] describes a parallel algorithm based on the divide-and-conquer strategy. The input points are subdivided recursively into two groups according to the median of their x or y-coordinate. Points are projected onto the perpendicular plane going through the median, the lower convex hull of projected points is found using quickhull algorithm and the back projection of the convex hull gives a set of Delaunay edges that separates the input region with the input points into two non-convex subregions. Both sub-regions are simultaneously triangulated by Dwyer's algorithm [20] and the Delaunay triangulation is obtained by a union of both local triangulations. Similar approach was also proposed by Lee et al. [30]. As both approaches need to keep all points in the memory for the computation

of Delaunay separators, they are not suitable for the processing of large data sets. This drawback, however, can be easily removed.

Another parallel approach was proposed by Chrisochoides et al. [14]. It starts by a construction of a coarse triangulation of a subset of points. The created triangles are partitioned into k continuous regions and distributed over k processors. After that, the processors insert points lying in their regions using the Watson's approach [37]. The triangles constructed on boundaries are redistributed heuristically in order to balance the load of the processors. The algorithm is again not suitable for large data sets because every processor needs all points but its extension seems to be possible.

Recently, Blandford et al. [10] proposed, in our opinion, a very complex parallel algorithm based also on the Watson's approach. It exploits a special data structure that maintains the triangulation in a compressed format in the memory and dynamically decompresses a small region of this triangulation whenever a point has to be inserted into this region. The authors were able to process a data set with billions of points on parallel architecture with 64 processors (each processor had 4GB RAM).

Chen et al. [13] proposed a parallel algorithm that distributes points according to their coordinates over k processors and let processors to triangulate their points. After that, each processor computes an "interface" using an incremental construction approach, i.e., a set of triangles that crosses the area boundaries. For this operation, the processor must have available all points lying in the areas adjacent to the area assigned to this processor. In the final step, interfaces are merged together in order to obtain the resulting triangulation. As the processor does not need the whole input set, the algorithm is able to triangulate huge data sets.

Kohout at al. [28] proposed an application independent software layer that supports processing of large data sets by the simulation of the shared memory on a cluster of workstations. The layer provides universal routines for the manipulation with data, no matter whether the data is stored locally or remotely. It was used for the construction of Delaunay triangulation by a parallel approach described in [27]. Due to an intensive network communication required to fetch the remote simplices, this solution is not, however, effective.

Parallel processing is not the only way how to handle large data sets. Recent research has been also focused on external memory algorithms (acknowledged also as out-of-core algorithms) that use disks for temporary storage of data structures that are too large to fit in the memory and load them into the memory when necessary. Many theoretical papers discussing the optimal strategy to minimize data movement have been published, e.g., [1], [4], etc. A good survey of these approaches can be found in [6]. Surprisingly only a few practical papers dealing with the construction of Delaunay triangulation of large data sets exist.

Agarwal et al. [3] designed an out-of-core algorithm for the construction of Constrained Delaunay triangulation based on the divide-and-conquer approach. It sorts all points in such a way that they lie on a space-filling Hilbert curve and splits them into subsets that are successively triangulated. Whenever the triangulation of a subset is completed, the algorithm checks each of the remaining points if it does not violate the Delaunay criterion and if the outcome is positive, the triangulation is altered by an appropriate way. Although the authors claim that their algorithm is practical, we think that it is quite difficult to be understood and implemented. Nevertheless, the proposed algorithm is quite efficient. According to the results, it was able to compute the Delaunay triangulation of a data set with several millions of points in a couple of minutes.

A data streaming approach described by Isenburg et al. [26] constructs the Delaunay triangulation in two steps. In the first step, the input stream with points is read (several times) and points are partitioned into buckets written into an output stream. In the second pass, this output stream is read and the Delaunay triangulation is successively constructed using the Watson's approach. When all points from one bucket are processed, the constructed triangles that will not change in the future are removed from the memory (and their points as well) and written into the output stream. The algorithm is easy to be understood but its implementation may be more difficult as many singular cases must be handled. Nevertheless, it is very efficient: a data set with billion points can be processed in several hours on a common computer.

In this paper, we propose an out-of-core algorithm called ACUT (Area CUTting) that is suitable for the construction of Delaunay triangulation of large E^2 data sets. It is very simple to be understood

as well as implemented. It is based on the method of incremental insertion with local transformations (i.e., edge flipping) [23], thus it offers better robustness than algorithms based on other approaches. As far as we know, it is the first algorithm based on this approach. It does not need to sort points in preprocessing (but it performs a spatial reorganization of points), is insensitive on distribution of points and can be generalized to incorporate constraints given in the form of prescribed edges, to use non-Euclidian metrics or weights of points, etc.

The paper is structured as follow. An overview of our approach (called ACUT) is given in Section 2; Sections 3 and 4 describe its steps in detail. Section 5 brings results of the performed experiments and a comparison with existing approaches. Section 6 concludes the paper.

2 Overview of ACUT

The proposed approach for the construction of Delaunay triangulation of large data sets is based on the obvious idea to split the input data into several smaller sets, compute local triangulation for these sets and merge these triangulations together.

As many existing solutions, our approach has two main steps. In the first one, the points from the input file are reorganized into cells of a uniform grid and stored into a temporary grid file. In the second step, these cells (and their points, indeed) are partitioned into the requested number of regions and an appropriate star shaped domain with Delaunay edges on its boundary is computed for every region - see Figure 1. Domains are processed successively: their points, i.e., points lying inside the domain (or on the boundary) are triangulated and the triangulation is stored into the output file. As domain boundaries are formed by Delaunay edges, no complex merge phase is required, all that is needed is to update the connectivity between triangles of adjacent domains, if necessary. The schematic view of the proposed approach is shown in Figure 2.

The most complex part is the construction of domains. Let us describe it in more detail. It starts with a construction of the convex hull of input points. The computed convex hull is the domain appropriate to the initial region that represents the whole grid of points. This region (and its domain) is recursively subdivided by horizontal and vertical cuts using an approach by Mueller [32]. More information about it is given in the following section. Whenever a cut for the region is found, the technique proposed by Blelloch et al. [11] is used to construct a poly-line of Delaunay edges along the cutting edge (details are described in the Section 3). This poly-line is combined together with separators forming the boundary of the appropriate domain to create two new domains. The recursion stops when the requested number of domains is reached.

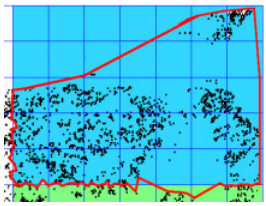


Figure 1: The region (cyan) and its corresponding star-shaped domain (thick red poly-line).

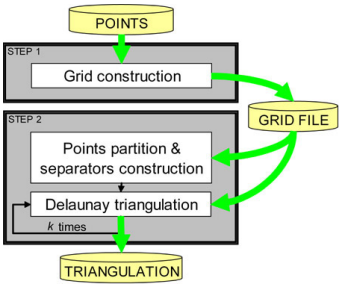


Figure 2: The schema of the proposed approach.

3 Details of ACUT

In this section, we describe both steps of our approach (see the previous section) in detail.

3.1 Grid Construction

In this first step of our approach, the input points are subdivided into cells of a uniform grid that covers the min-max box of points and has $\sqrt{c \cdot \sqrt{N} \cdot \frac{H}{W}} \times \sqrt{c \cdot \sqrt{N} \cdot \frac{W}{H}}$, cells, where N is the number of input points, H and W are height and width of the

min-max box of the input points and c is a constant (we use 0.5). For each cell of the grid, a small point buffer is created in the memory. Its capacity depends on the number of points to be processed, number of cells in the grid and on the amount of memory available for this first step. The input points are successively read and inserted into the corresponding buffer according to their coordinates. If the buffer is full, its points are written at the end of a temporary grid file and the position of the written block in the file is enlisted in a list of fragments stored in the cell structure. When the entire input is read, all buffers are flushed. At the end, the grid structure, i.e., the matrix of lists of fragments is also written into the grid file.

The problem is that very often we do not know the min-max box of points in advance and sometimes even the precise number of points is also unknown. To avoid multiple reading of input file, we propose a data-streaming algorithm that works as follows. At the beginning, a chunk of M ($M < N$) points is loaded into the memory, the overall number of points in the file is estimated from the size of input file and the size of currently loaded number of points, the min-max box of the loaded points is computed, an initial uniform grid is created using the formula written above and points are subdivided into the grid. After that another chunk of points is loaded, their min-max box is computed, the current grid is enlarged by adding some rows and/or columns, if necessary, and points are again subdivided into cells. If the grid dimension is larger than some given threshold, pairs of cells are merged together. Let us point out that this merge stage only concatenates lists of fragments, no point is moved. So it goes until all points are processed.

An advantage of this partitioning is that it subdivides input points into cells in one pass. On the other hand, points lying in one cell are very often fragmented into blocks (especially, if cell buffers are too small), which slows down the processing due to an inefficient use of spatial coherence.

3.2 Cells Partition

The points have to be subdivided into k subsets in such a manner that not only the almost equal number of points in each subset is ensured but also the bounding boxes of these subsets have minimal intersection and the total length of boundaries is minimal. To achieve this, we use the strategy proposed by Mueller [32] that is based on a recursive subdivision of the summed-area table using horizontal and vertical cuts. The summed-area table is constructed from a matrix of values corresponding to numbers of points lying in appropriate cells of a uniform grid covering the bounding box of all points - see Figure 3. Let us note that we already created this grid in the first step of our approach. An advantage of the Mueller approach is that the summed-area table can be efficiently found in $O(R)$, where R is the total number of cells and split into k regions in $O(k \cdot log(R))$ using a binary search algorithm. Detailed description is out of the scope of this paper.

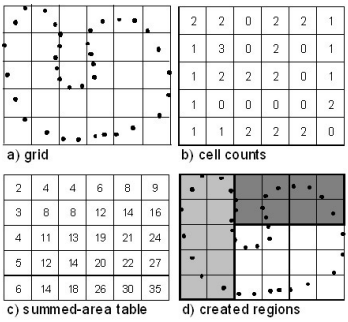


Figure 3: The subdivision of cells into three regions.

3.3 Construction of Delaunay Separators

When, in the Mueller algorithm (see the previous section), a cut for the current region is computed, we construct the convex hull of points transformed by the following formulas: $P(P_x, P_y) \rightarrow P'(P_y - C_y, \|P - C\|^2)$, if the cut is the vertical one and $P(P_x, P_y) \rightarrow P'(P_x - C_x, \|P - C\|^2)$ otherwise, where C is the centre of the cut. Unlike Hardwick et al., not all points are processed but only points lying in cells covered by the domain appropriate to the region to be cut. The lower part of the constructed convex hull is taken and its edges give the Delaunay separators between corresponding untransformed points - see Figure 4.

All that remains is to combine the constructed poly-line of Delaunay separators with the domain separators and to create two new domains. Starting from the first point from the poly-line, we search in the chain of vertices of the domain polygon to find this point. If the corresponding point is not found,

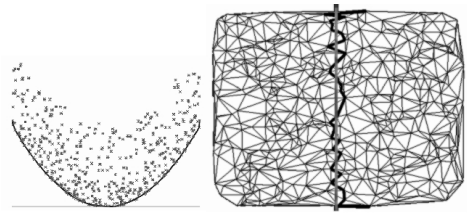


Figure 4: The lower convex hull of transformed points and the Delaunay triangulation with Delaunay separators along the cutting line (gray line). Images were adopted from [25].

the next point is taken and the search restarts. So it goes until the match is found. The other end of the poly-line is also processed, the domain polygon is split into two poly-lines at the positions of matches and they are connected to the appropriate part of the constructed poly-line to form two new domains. The complexity of this brute-force combination is $O(\sqrt{N})$ in the worst-case. The performance could be improved by using a hash table - in our current implementation, we do not use it. Figure 5 shows the domains constructed for a real data set.

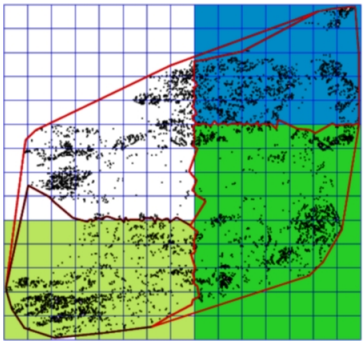


Figure 5: The constructed domains.

3.4 Construction of Convex Hull

As there is not enough memory to load every point into the memory, we use an incremental construction algorithm that works as follows. Starting with an initial convex hull, points are successively tested whether they lie outside the current convex hull. If the result of the test is positive, the convex hull has to be updated in such a manner that the point lying outside belongs now to the new convex hull.

In our implementation, the location is speeded up by red-black trees [7]; one is used for the lower part of the convex hull, another one for the upper part. An internal node stores a pair of key x and an associated value p, where p is a point on the convex hull and x is its x-coordinate. An external node represents an edge or an empty half-space. For each given point q, we search its x-coordinate in both trees to find either vertex v or an edge e and we test whether the given point lies above or below the found primitive - see Figure 6a. If the point lies outside the current convex hull, it is inserted into the appropriate red-black tree and points that no longer belong to the convex hull are removed from this tree - see Figure 6b. Let us note that the convex hull can be computed by this algorithm in $O(N \cdot log(M))$ expected time, where M is the number of points on the convex hull (it is usually much less than N).

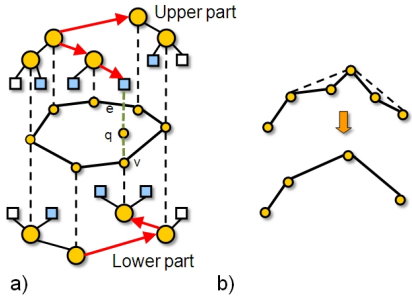


Figure 6: The location of mutual position of the point q and the convex hull (a) and the update of convex hull (b).

As the orientation test used to determine whether a point lies below or above some edge and to determine whether a point should be removed from the convex hull is exactly the same test that is used in the computation of the Delaunay triangulation itself and red-black trees are nowadays a common part of libraries, the implementation of this part of our approach is pretty simple.

3.5 Domain Triangulation

Domain points are triangulated using the method of incremental insertion with flipping. We decided to

use this method because of its simplicity and robustness: in the case of an incorrect or inconsistent Delaunay criterion evaluation caused by numerical inaccuracy, a triangulation with two or more non-Delaunay triangles is obtained, but it is still a valid triangulation. The method can be also simply modified to incorporate constraints given in the form of prescribed edges, to use non-Euclidian metrics or weights of points, etc.

Starting with an auxiliary triangle containing all domain points in its interior, points are inserted successively into the triangulation as follows. The triangle containing the point to be inserted is located (we use the remembering stochastic walking [18]) and subdivided. Afterwards, the empty circumsphere criterion is tested recursively on all triangles adjacent to the new ones, and if necessary, their edges are flipped, i.e., local transformation are applied. Figure 7 shows an example of the insertion.

Figure 7: The insertion of point into the DT.

In our approach, points on the boundary of the domain are inserted first and the remaining domain points (they must lie in some cell of the corresponding rectangular region [11]) are afterwards inserted in a pseudorandom order as follows. While the insertion order of points from one cell is randomized, cells are ordered according to the space-filling Hilbert curve [12] and all points from one cell must be processed before the algorithm may advance to another cell - see Figure 8. This pseudorandom order of insertion has two important features. First, it exploits the spatial coherence in data and, therefore, the walking needs *O(1)* in the expected case to locate the triangle to be subdivided. Next, the randomness lowers the sensitivity of the approach to numerical errors.

In our implementation, an array is used as a compact and efficient data structure to hold the triangulation. Let us note that as planar triangulations of *N* points have at most *2·N* triangles, the array can be allocated at the beginning to hold this amount.

When all points are processed, every outer triangle, i.e., the triangle that lies outside the domain, has to be removed and the remaining triangles must be stored into the output file. The removal starts with searching for any triangle that has a vertex of the auxiliary initial triangle. Usually, only a few triangles have to be checked (the worst-case we have experienced was 1% of triangles). The reference on this triangle is pushed into a stack and the following steps are repeated until the stack is empty. The triangle is popped from the stack and every adjacent triangle not sharing edge that belongs to the Delaunay separators is pushed into the stack. A hash table is used to speed-up this test. The triangle is marked then as processed and disconnected from the mesh.

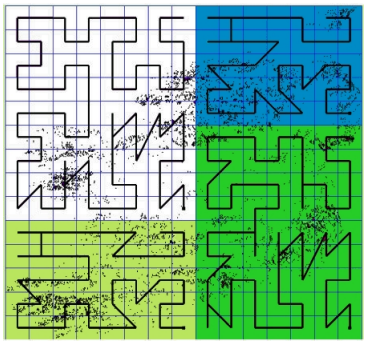


Figure 8: The insertion order of cells (black path).

4 Singular Cases

The proposed approach, as it was described in previous sections, may not work perfectly for all data sets. In this section, we discuss singular cases that must be handled in order to have a robust solution.

First problem roots in the ambiguity of Delaunay triangulation for data sets containing four points lying on a common circle - see Figure 9. As this quadrilateral, say p_a, p_b, p_c, p_d, may be triangulated by two different ways, it may happen that while the diagonal p_b, p_d is chosen during the construction of Delaunay separators, the triangulator creates the other diagonal, i.e., the edge p_a, p_c. Due to this inconsistency, we can walk from an outer triangle to an inner triangle without crossing any Delaunay separator, which leads to that no triangle is stored in the extraction stage.

Fortunatelly, this problem can be easily solved by adding an additional test to the extraction algorithm.

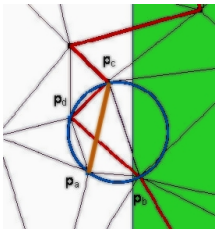


Figure 9: The ambiguity in the DT.

When the triangle is popped, we check whether all of its vertices belong to a set of vertices forming the Delaunay separators (this can be done quickly using another hash table). If the outcome of this test is negative, i.e., the triangle contains some inner domain point, the appropriate adjacent triangle forming the quadrilateral is found and their common edge is swapped.

The second problem is caused by a numerical inaccuracy during the construction of convex hull. It may happen that a point that should be on the convex hull is due to round-off errors located inside the convex hull, which leads to the construction of incorrect Delaunay separators. The result is that this point lies outside the domain into which it is assigned - see Figure 10. Therefore, its incident triangles are removed during the extraction, which means that this point will not be triangulated in the final Delaunay triangulation.

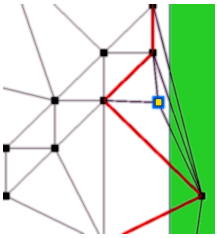


Figure 10: The problem of a numerical inaccuracy: the highlighted point lies outside its domain.

This problem is automatically detected when the algorithm handles the problem of ambiguity and is unable to find an appropriate adjacent triangle to perform the swap edge. Unfortunately, its solving is extremely difficult (if not impossible). Indeed, we could redistribute these points assigned to incorrect domain, but their detection would harm the performance significantly. As we detected, in our experiments, up to 5 such cases per one domain (35 in total for a data set with 64M points) when single float precision was used, we cease to correct this problem, thus allowing a small amount of unconnected points in the resulting mesh. Let us note that we have not experienced this problem when we switched to double precision.

5 Experiments & Results

The proposed approach was implemented in C++ using Microsoft Visual Studio.NET 2005. Loading/storing is written generally (e.g., it can load data into various structures from various file formats), which has a significant negative impact on the performance. Also some profiling is included in the code, e.g., counting time spent in important routines. It, indeed, slow downs the computation. Single precision arithmetics was used in the code.

We tested the implemented solution on various data sets (generated and real) in a binary format on Dell Optiplex GX620: Intel Pentium Processor 3.2 GHz with Hyper-Threading Technology and EM64, 2GB Dual Channel DDR2 RAM (but we used only 1GB), 250GB SATA Disk, MS Windows XP. For each data size several different data sets were tested and the presented times were calculated as average of measured total times. Let us note that in the total time everything is included, i.e., the time needed for loading of points, for the grid construction, convex hull computation, triangulation and for the storing of the output mesh.

Figure 11 shows dependency of the total time on the number of points for generated data sets with various point distributions: uniform, gauss, clusters and for real data sets (mostly GTOPO30 [24]). As it can be seen, the proposed approach is insensitive to point distributions. Moreover, it can process data sets in almost linear time.

A runtime profiling for tested uniform data sets is given in Figure 12. If we consider larger data sets only, i.e., 16M+, more than 50% of the overall time is consumed by I/O operations (grid construction - 10–20%, points loading - up to 10%, domains construction 30%). The time required for the domains construction, i.e., for the cells partition and the construction of Delaunay separators, grows proportionally to the number of points. For a 128M data

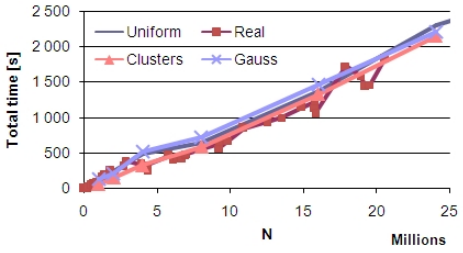


Figure 11: The runtime for data sets with various point distributions.

set, when 32 domains were used, ACUT needed as much time for this operation as for the triangulation itself. Let us note that the domains construction also performs some I/O operations.

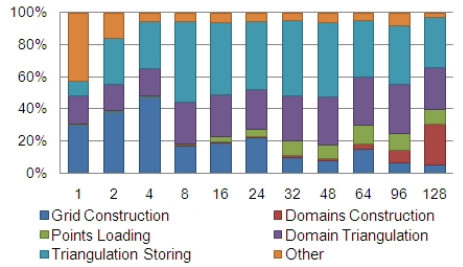


Figure 12: The profiling for uniform data sets.

We compared our results with the results achieved by Agarwal et al. [3]. According to published graphs, their approach is about three times faster. However, this comparison is skewed because Agarwal et al. do not include the time consumed by the required sorting of input points into the published total time needed for the Delaunay triangulation. They also used a more powerful computer for their experiments: Intel Pentium XEON 2.4 GHz with Hyper-Threading Technology, 1GB RAM (but only 128 MB was used), 4x72 GB SCSI (10000 rpm) disks in RAID-0 configuration running Linux with kernel 2.4.5-smp. Therefore, we daresay that both approaches are more or less competitive in the performance, however, our approach is, in our opinion, much easier to be implemented.

We also compared the results with the results by Isenburg et al. [26]. Although authors claim that their approach is 12 times faster than the approach by Agarwal et al, the experiments that we performed with their software on our data sets show that for large uniform data sets our approach outpaces theirs - see Figure 13. For real data sets, however, Isenburg et al. achieved much better performance (about 4 times for 20M) - see Figure 14. The reason for this behavior is that points in tested real data sets are already ordered according to their coordinates, while points for uniform data sets are unordered. Let us note that this comparison is, however, imprecise because of two following reasons. First, unlike our approach, Isenburg et al. do not store the connectivity between triangles, which is, indeed, unacceptable for many applications. According to our additional experiments, the triangulation extraction (in ACUT) runs much faster if we do not store the connectivity (about 12 times for a 16M uniform data set, which speed ups the overall process 2 times). On the other hand, for experiments with the software by Isenburg et al., the input was given in text format, which is, indeed, quite inefficient format (our approach requires about 70% more time to process text files than to process binary files, e.g., 74.2% for a 16M uniform data set). Let us, therefore, conclude the comparison by the claiming that both approaches are more or less competitive, each has its pros and cons.

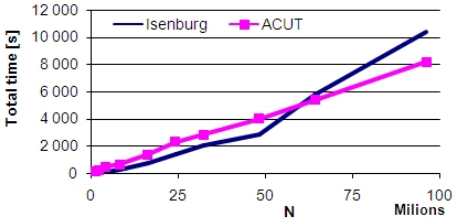


Figure 13: The runtime comparison of our and Isenburg's approach [26] for uniform data sets.

6 Conclusion

In this paper, we have proposed an out-of-core approach for the construction of Delaunay triangulation in E^2 based on the incremental insertion with local transformations, which, as far as we know, has not been used for the processing of large data sets yet. The approach is easy to implement and robust.

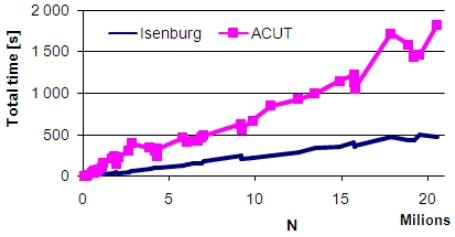


Figure 14: The runtime comparison of our and Isenburg's approach [26] for real data sets.

It can be also generalized to incorporate constraints given by a set of prescribed edges into the triangulation or to use weights of points. All that is needed is to modify the transformation of points (see Section 3.4) using the idea proposed by Maur et al. [31]. It transforms constrained edges (or non Delaunay edges introduced by given weights of points) onto a lower convex hull and to modify the Delaunay criterion test in such a way that constrained edges are always considered valid, i.e., they are never flipped.

The proposed approach was implemented in C++ and tested on various data sets with up to 128 millions of points. According to the our experiments, our solution processes data sets in an almost linear time (a 128M uniform data set was processed in 2.5 hours on a common hardware) and, due to its simplicity, insensivity to point distribution and generalization possibilities,it is, in our opinion, an interesting alternative to existing more efficient approaches by Agarwal et al. [3] and Isenburg et al. [26].

Acknowledgment

This work was supported by the Ministry of Education of The Czech Republic - project LC 06008. We would also like to thank Prof. V.Skala from the University of West Bohemia for providing conditions in which this work has been possible.

References

[1] J. Abello, A.L. Buchsbaum, J.R. Westbrook. A functional approach to external graph algorithms. *Algorithmica,* 32(3):437–458, 2002.

[2] L. Adamian, R. Jackups, A.T. Binkowski, J. Liang. Higher-order interhelical spatial interactions in membrane proteins. *Journal of Molecular Biology,* 327(1):251–272, 2003.

[3] P.K. Agarwal, L. Arge, K. Yi. I/O efficient construction of constrained Delaunay triangulations. In *13th European Symposium on Algorithms,* 355–366, 2005.

[4] L. Arge. External memory data structures. In *European Symposium on Computational Geometry,* 1–29, 2003.

[5] D. Attali, O.J. Lachaud. Delaunay conforming iso-surface, skeleton extraction and noise removal. *Computational Geometry,* 19(2–3):175–189, 2001.

[6] J.M. Abello, J.S. Vitter.*External memory algorithms.* American Math. Society, 2000.

[7] http://ww3.algorithmdesign.net/ handouts/ IncrementalHull.pdf

[8] E. Béchet, J.C. Cuilliere, F. Trochu. Generation of a finite element MESH from stereolithography (STL) files. *Computer-Aided Design,* 34(1):1–17, 2002.

[9] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf. *Computational Geometry. Algorithms and applications.* Springer-Verlag, Berlin Heidelberg, 1997.

[10] D.K. Blandford, G.E. Blelloch, C. Kadow. Engineering a compact parallel Delaunay algorithm in 3D. In *SCG'06,* 292–300, 2006.

[11] G.E. Blelloch, G.L. Miller, D. Talmor. Developing a Practical Projection-Based Parallel Delaunay Algorithm. In *12th Annual Symposium on Computational Geometry*, 1996.

[12] K. Buchin. Incremental construction along space-filling curve. In *European Workshop on Computational Geometry*, 17–20, 2005.

[13] M.B. Chen, T.R. Chuang, J.J Wu. Efficient parallel implementations of 2D Delaunay triangulation with High Performance Fortran. In *10th SIAM Conference on Parallel Processing for Scientific Computing,* 2001.

[14] N. Chrisochoides, D. Nave. Parallel Delaunay mesh generation kernel. *International Journal for Numerical Methods in Engineering,* 58:161–176, 2003.

[15] S.W. Chung, S.J. Kim. A remeshing algorithm based on bubble packing method and its application to large deformation problems. *Finite Elements in Analysis and Design,* 39(4):301–324, 2003.

[16] B. Delaunay. Sur la sphere vide. Izv. Akad.

Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk, 793–800, 1934.

[17] L. Demaret, N. Dyn, M. S. Floater, A. Iske. Adaptive thinning for terrain modelling and image compression. In *Advances in Multiresolution for Geom. Modelling,* 321-340, 2004.

[18] O. Devillers, S. Pion, M. Teillaud. Walking in triangulation. In *17th Annual Symposium on Computational Geometry,* 106–114, 2001.

[19] O. Devillers. Improved incremental randomized Delaunay triangulation. In *14th Annual Symposium on Computational Geometry*, 106–115, 1998.

[20] R.A. Dwyer. A Simple Divide-and-conquer algorithm for constructing Delaunay triangulation in O(n log log n) expected time. In *2nd Annual Symposium on Computational Geometry,* 276–284, 1986.

[21] N. A. Golias, R. W. Dutton. Delaunay triangulation and 3D adaptive mesh generation.*Finite Elements in Analysis and Design,* 25:331–341, 1997.

[22] G. Gonalves, P. Julien, S. Riazanoff, B. Cervelle. Preserving cartographic quality in DTM interpolation from contour lines. *ISPRS Journal of Photogrammetry and Remote Sensing,* 56(3):210–220, 2002.

[23] L.J. Guibas, D.E. Knuth, M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica,* 7:381–413, 1992.

[24] GTOPO30 - Global topographic data. http:// edc.usgs.gov/ products/ elevation/ gtopo30/ gtopo30.html

[25] J.C. Hardwick. Implementation and evaluation of an efficient parallel Delaunay triangulation algorithm. In *Symposium on Parallel Algorithms and Architectures*, 22–25, 1997.

[26] M. Isenburg, Y. Liu, J. Shewchuck, J. Snoyeink. Streaming computation of Delaunay triangulation. In *ACM SIGGRAPH 2006,* 1049–1056, 2006.

[27] J. Kohout, I. Kolingerová. Parallel Delaunay triangulation in E3: Make it simple. *The Visual Computer,* 19(7–8): 532–548, 2003.

[28] J. Kohout, M. Varnuška, I. Kolingerová. Surface reconstruction from large point clouds using virtual shared memory manager. In *Computational Science and Its Applications - ICCSA 2006,* 71–80, 2006.

[29] I. Kolingerová, J. Kohout. Optimistic parallel Delaunay triangulation. *The Visual Computer,* 18(8):511–529, 2002.

[30] S. Lee, C.I. Park, C.M. Park. An improved parallel algorithm for Delaunay triangulation on distributed memory parallel computers. *Parallel Processing Letters*, 11(2–3):341–352, 2001.

[31] P. Maur, I. Kolingerová. The employment of regular triangulation for constrained Delaunay triangulation. In *Computational Science and Its Applications - ICCSA,* 198–206, 2004.

[32] C. Mueller. Hierarchical graphics databases in sort-first. In *IEEE Symposium on Parallel Rendering,* 49–57, 1997.

[33] K.F. Mulchrone. Application of Delaunay triangulation to the nearest neighbour method of strain analysis. *Journal of Structural Geology,* 25(5):689–702, 2003.

[34] T. Nishioka, H. Tokudome, M. Kinoshita. Dynamic fracture-path prediction in impact fracture phenomena using moving finite element method based on Delaunay automatic mesh generation. *International Journal of Solids and Structures,* 38(30-31):5273–5301, 2001.

[35] L. Prasad, A.N. Skourikhine. Vectorized image segmentation via trixel agglomeration. *Pattern Recognition,* 39:501–514, 2006.

[36] N.J. Walkington, J.F. Antaki, G.E. Blelloch, O. Ghattas, I. Melcevic, G.L. Miller. A parallel dynamic-mesh Lagrangian method for simulation of flows with dynamic interfaces. In *ACM/IEEE conf. on Supercomputing,* 2000.

[37] D.F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *Computer Journal,* 24(2):167–172, 1981.

[38] Y. Xiao, H. Yan. Text region extraction in a document image based on the Delaunay tessellation. *Pattern Recognition,* 36(3):799–809, 2003.

[39] M. Zadravec, B. Žalik. An almost distribution-independent incremental Delaunay triangulation algorithm. *Visual Computer,* 21(6):384–396, 2005.

[40] B. Žalik, I. Kolingerová. An incremental construction algorithm for Delaunay triangulation using the nearest-point paradigm. *The International Journal of Geographical Information Science,* 17(2):119–138, 2003.

Invited Talk

Less is More: Coded Computational Photography

Ramesh Raskar

Mitsubishi Electric Research Laboratories (MERL)
Cambridge, MA, USA

Abstract

Computational photography combines plentiful computing, digital sensors, modern optics, actuators, and smart lights to escape the limitations of traditional cameras, enables novel imaging applications and simplifies many computer vision tasks. Unbounded dynamic range, variable focus, resolution, and depth of field, hints about shape, reflectance, and lighting, and new interactive forms of photos that are partly snapshots and partly videos are just some of the new applications found in Computational Photography. I will discuss Coded Photography which involves encoding of the photographic signal and post-capture decoding for improved scene analysis.With film-like photography, the captured image is a 2D projection of the scene. Due to limited capabilities of the camera, the recorded image is a partial representation of the view. Nevertheless, the captured image is ready for human consumption: what you see is what you almost get in the photo. In Coded Photography, the goal is to achieve a potentially richer representation of the scene during the encoding process. In some cases, Computational Photography reduces to 'Epsilon Photography', where the scene is recorded via multiple images, each captured by epsilon variation of the camera parameters. For example, successive images (or neighboring pixels) may have a different exposure, focus, aperture, view, illumination, or instant of capture. Each setting allows recording of partial information about the scene and the final image is reconstructed from these multiple observations. In Coded Computational Photography, the recorded image may appear distorted or random to a human observer. But the corresponding decoding recovers valuable information about the scene.

'Less is more' in Coded Photography. By blocking light over time or space, we can preserve more details about the scene in the recorded single photograph.

1. Coded Exposure: By blocking light in time, by fluttering the shutter open and closed in a carefully chosen binary sequence, we can preserve high spatial frequencies of fast moving objects to support high quality motion deblurring.
2. Coded Aperture Optical Heterodyning: By blocking light near the sensor with a sinusoidal grating mask, we can record 4D light field on a 2D sensor. And by blocking light with a mask at the aperture, we can extend the depth of field and achieve full resolution digital refocussing.
3. Coded Illumination: By observing blocked light at silhouettes, a multi-flash camera can locate depth discontinuities in challenging scenes without depth recovery.
4. Coded Sensing: By sensing intensities with lateral inhibition, a gradient sensing camera can record large as well as subtle changes in intensity to recover a high-dynamic range image.

I will show several applications of coding exposure, aperture, illumination and sensing and describe emerging techniques to recover scene parameters from coded photographs.

A Method to Detect and Mark False Branches of a Vessel Graph

J. Bruijns, F.J. Peters, R.P.M. Berretty, B. Barenbrug

Philips Research
High Tech Campus 36, 5656 AE EINDHOVEN, The Netherlands
Email: Jan.Bruijns@philips.com

Abstract

Volume representations of blood vessels acquired by 3D rotational angiography are very suitable for diagnosing a stenosis or an aneurysm. For optimal treatment, physicians need to know the shape of the diseased vessel parts. Therefore, we previously developed a method for fully-automatic extraction of this shape from such a volume representation. In some cases, neighbor vessels are erroneously connected via false branches. In this paper, we describe a method to detect and mark false branches of a vessel graph.

CR Descriptors:
3D Rotational Angiography, volume visualization, Computer Assisted Diagnosis, Shape Extraction.

1 Introduction

Volume representations of blood vessels acquired by 3D rotational angiography after injection with a contrast agent [14] have a clear distinction in gray values between tissue and vessel voxels. Therefore, these volume representations are very suitable for diagnosing a stenosis, a local narrowing of a vessel caused for example by cholesterol (see Figure 11), or an aneurysm, a local widening of a vessel caused by a weak vessel wall (see Figure 14).

For optimal treatment of a stenosis or an aneurysm, physicians need to know the cross-sectional shape parameters in the neighborhood of the diseased vessel parts. We developed a method for semi-automatic extraction of these parameters from a surface model of the vessel boundaries [4]. However, if two vessel branches are close together, it is possible that vertices of the neighbor vessel branch are included in the set of selected vertices which are used to estimate the local shape parameters of the vessel branch investigated. To exclude surface vertices of neighbor vessel branches, we developed a method for fully-automatic branch labelling [5] to give the vessel voxels (and from these the surface vertices) a unique label per vessel branch. This method results also in a set of directed graphs with nodes and edges (called *"skeleton branches"*) which facilitates fully-automatic vessel tracing of a minimum path along the skeleton branches of the same graph.

Because of the finite resolution, it is possible that neighbor vessels are erroneously connected at a number of places. Examples of erroneous vessel junctions are shown in Figure 13 (the erroneous vessel junctions are more clearly visible when the surface model is rendered from a varying viewpoint). These erroneous vessel junctions result either in *false branches* or in *merged branches*. A false branch is an erroneous skeleton branch between two in reality separated vessel branches (see left picture of Figure 1). A merged branch is a single common skeleton branch for two in reality separated vessel branches (see right picture of Figure 1). A false branch is generated if the erroneous vessel junction is relatively short. A merged branch is generated if the erroneous vessel junction is relatively long.

In case of false branches, it is possible that some of the cross-sectional shape parameters are extracted from neighbor vessel branches. Therefore, we developed a method to detect and mark such false branches so that the minimum path generation can avoid these false branches.

Remark:

Merged branches are not yet detected and repaired!

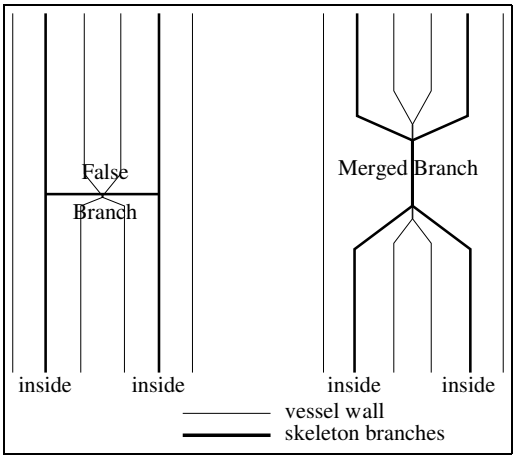


Figure 1: False versus merged branches

2 Related Work on Branch Labelling

The graph structure which represents the topology of the vessels, can be generated by various skeletonization algorithms. Skeletonization algorithms based on topological thinning are presented in [1, 7, 13]. Skeletonization algorithms based on morphological thinning are presented in [8, 12]. Skeletonization algorithms based on distance transformations are described in [6, 2]. Skeletonization algorithms based on thinning and distance maps are presented in [17, 3]. Skeletonization algorithms based on propagation are described in [10, 16, 18, 19]. A skeletonization algorithm based on path tracking is presented in [9]. A skeletonization algorithm based on ridge extraction is described in [11].

Algorithms for correction of the generated graph structure are presented in [18, 17, 3].

3 Method

3.1 Preamble

The input of our method is the set of directed graphs (one for each component of the voxel vessel structures) with nodes (one for each vessel junction and one for each vessel extremity) and skeleton branches (one for each vessel branch). A skeleton branch consists of a set of face connected vessel voxels, called *"skeleton voxels"*. The skeleton voxels, located close to the center line of the vessel branch, have a unique label per vessel branch. An example is shown in Figure 12. The skeleton voxels are displayed in a color according to their label but skeleton voxels with different labels can have the same color because of the limited number of colors used for display. The nodes are equipped with geometry (see Figure 15). A center sphere together with the center planes give the position, the size and the delineation of the center region (i.e. the junction). The branch spheres together with the branch planes give the size and the direction of the branch regions adjacent to the center region.

We use a number of thresholds (length factors, radius factors, angles etcetera). These thresholds are empirically determined: a much smaller and/or a much larger threshold (factor) gives wrong outcomes at least for our clinical volume datasets.

3.2 Typification of a False Branch

A skeleton branch is marked as a false branch if it fulfills the following qualitative conditions which express that the configuration of a false branch with two continuing main branches is topologically equivalent to the H-shape shown in the left picture of Figure 1:

1. The branch is relatively short.
2. The branch is not connected to another branch so that these two form a continuing branch (explained in Section 3.4).
3. The branch is located between two continuing main branches (explained in Section 3.5).

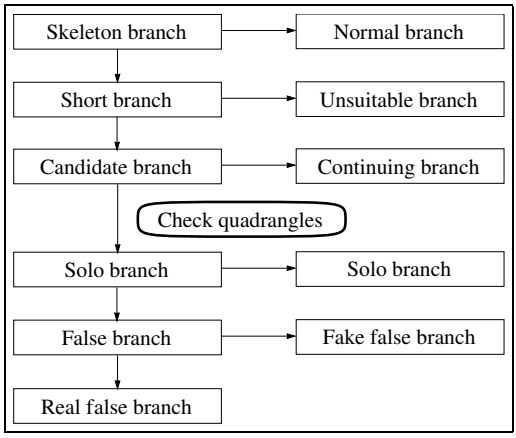


Figure 2: Flowchart of branch classification. "Check quadrangles" is explained in Section 3.7

3.3 Detection of a False Brach

The false branches are detected by generating subsets of the set of all skeleton branches (see Figure 2). First, the set of skeleton branches is subdivided in the set of *normal branches* and the set of *short branches* (first qualitative condition) by applying the following tests in the given order:

1. A skeleton branch is classified as a normal branch if the number of skeleton voxels of the branch (i.e. the length of the branch) is greater than 2 times the Manhattan distance between the two nodes of the branch.
   The Manhattan distance is used because the length of a branch (a set of face connected vessel voxels) is a kind of cumulative Manhattan distance.
   This test is needed to prevent that a skeleton branch with two very close nodes (as tested in the following two conditions) but with a long detour is classified as a short branch.
2. A skeleton branch is classified as a short branch if the two center spheres overlap.
3. A skeleton branch is classified as a short branch if the centers of both branch spheres are located inside one of the two center spheres.
   An example is shown in Figure 3.

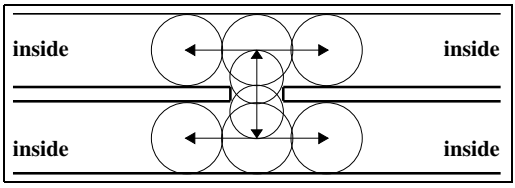


Figure 3: A short branch which is also a candidate branch. The thick lines are the detected vessel boundaries. The two big central circles are the center spheres. The four big flank circles and the two small circles are the branch spheres. For clarity the candidate branch is extended. Normally, the branch spheres of the candidate branch will almost completely overlap. The two horizontal arrow line segments indicate the points used to compute the local plane normals (explained in Section 3.4) for the normal branches. The two vertical arrow line segments indicate the points used to compute the local plane normals for the candidate branch.

4. A skeleton branch is classified as a normal branch when the previous two tests fail.

Next, the set of short branches is subdivided in the set of *unsuitable branches* and the set of *candidate branches*. Only candidate branches may be marked as a false branch. An unsuitable branch is a short branch for which it is either immediately clear that it will never be marked as a false branch, or for which it is too difficult to decide whether it can be correctly marked as a false branch:

1. A short branch is classified as an unsuitable branch if one or both nodes of the short branch has less than three skeleton branches.
   The third qualitative condition can be fulfilled only if at both nodes of the short branch at least two additional skeleton branches exist to form the required continuing main branches.
2. A short branch is classified as an unsuitable branch if there is yet another skeleton branch between the two nodes of the short branch.
   More than one skeleton branch between the two nodes makes checking not only more complex because this configuration is not topologically equivalent to the H-shape shown in the left picture of Figure 1 but also more unreliable because the other branch may disturb the fitting branch pair relation (this relation will be given further detail in Section 3.4) for the branch tested.
3. A short branch is classified as a candidate branch when the previous tests fail.

The topological configuration of a candidate branch with two additional skeleton branches is shown in Figure 4.

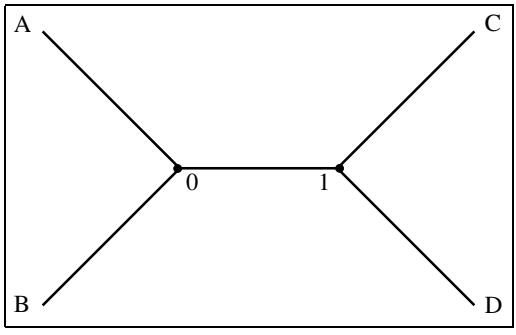


Figure 4: A candidate branch. The digits “0” and “1” indicate the nodes at the two ends of the candidate branch. The characters “A”, “B”, “C” and “D” indicate the nodes at the other end of the additional branches.

A candidate branch will never be marked as a false branch if the candidate branch is connected to another branch so that these two form a continuing branch (second qualitative condition). So, we have to check whether one of the additional skeleton branches forms a continuing branch pair with the candidate branch. This check is given further detail in Section 3.4. The outcome of this check is used to subdivide the set of candidate branches in the set of *continuing branches* and the set of *solo branches*. This check is also applied to the set of unsuitable branches to split off the unsuitable branches which are continuing branches because this difference in branch type is used in the detection of a continuing main branch (see Section 3.5).

Finally, the set of false branches is split off from the set of solo branches. A solo branch is marked as a false branch if at both nodes, two of the additional skeleton branches form a continuing main branch. This check is given further detail in Section 3.5.

Our algorithm may result in *unwanted false branches*. How these unwanted false branches are detected and repaired is described in Section 3.6.

In some cases, four candidate branches form a quadrangle. Such a quadrangle requires a special approach. This approach is given further detail in Section 3.7.

3.4 Detection of a Continuing Candidate Branch

Local Geometry:

We use the local geometry around both nodes of a candidate branch to detect whether a candidate branch forms a *continuing branch pair* with another branch. The local geometry of a node consists of a set of local planes, one for each branch of the node. The position of each local plane is equal to the position of the node. The normal of each local plane is the normalized direction from the position of this plane to a position depending on the type of the branch. If the branch is a normal branch, the position of the branch plane is used, else, the position of the node at the other end of the branch is used. The normal of each local plane gives the local direction of the corresponding branch away from the node. An example of a local geometry without the local planes is shown in Figure 3.

Fitting Brach Pairs:

Given the local geometry of a node (i.e. the inner products between the normals of the local planes) we could check whether a candidate branch b forms a continuing branch pair with another branch c at that node using an absolute threshold for the angle between the normals of the local planes. To avoid the choice for such an absolute threshold, we use a relative relation between the branches at a node. This relative relation is used to check whether the other branch c is aligned with the candidate branch b as good as the other branches at that node. Therefore, the local geometry of a node is used to define a number of fitting relations for a branch b at that node:

1. The *best fitting branch* of a branch b, indicated by $bfb(b)$, is the branch for which the inner product $ip(b, bfb(b))$ between the normals of their local planes is minimal (normals of local planes point away from the center position of the node!).
2. A *fitting branch* of a branch b, indicated by $fb(b)$, is a branch for which the inner product $ip(b, fb(b))$ between the normals of their local planes is equal (given the numerical errors) to the inner product of the best fitting branch: $ip(b, fb(b)) \leq ip(b, bfb(b)) + 10^{-10}$ So, a best fitting branch is also a fitting branch.
3. A branch b forms a *fitting branch pair* with a fitting branch $fb(b)$ if and only if branch b is a fitting branch of the fitting branch $fb(b)$.

The fitting branch pair relation is not symmetric: branch b forms a fitting branch pair with branch c does not imply that branch c forms a fitting branch pair with branch b.

Two examples of fitting branch pairs:

If in the example of Figure 5 the angle between branch BA and branch BC is almost equal to the angle between branch BA and branch BD, there are four fitting branch pairs:

Figure 5: Four fitting branch pairs between the nodes A, B, C and D.

Branch BC forms a fitting branch pair with branch BA, branch BD forms a fitting branch pair

with branch BA, and branch BA forms a fitting branch pair with branch BC and with branch BD.

In the example of Figure 6 the angle between branch BA and branch BC is much smaller than the angle between branch BA and branch BD. So, there are two fitting branch pairs:

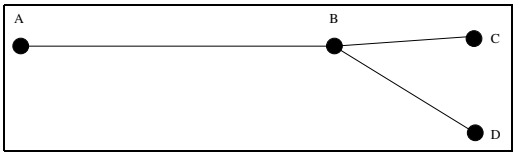


Figure 6: Two fitting branch pairs between the nodes A, B, C and D.

Branch BC forms a fitting branch pair with branch BA and branch BA forms a fitting branch pair with branch BC.

Continuing Branch Pairs:

Now we can check whether a candidate branch b forms a *continuing branch pair* with a branch c at the node bc between these two branches by applying the following tests in the given order:

1. A branch pair is not classified as a continuing branch pair if the candidate branch b does not form a fitting branch pair with the other branch c of the branch pair.
2. A branch pair is not classified as a continuing branch pair if the ratio of the average branch radii is less than 0.5 or greater than 2.0.
   The diameter of a continuing branch pair changes only slightly.
3. A branch pair is classified as a continuing branch pair if there exists a *valid run-up path* to the node bc.
   A *run-up path* to the node bc is a minimum path along the skeleton voxels of the skeleton branches of the generated graph between an arbitrary node as start node and the node bc as finish node. A run-up path to the node bc is a valid run-up path if the following conditions are fulfilled:
   (a) The radius along the run-up path is greater than 0.75 times the radius of the candidate branch b.
   This condition excludes run-up paths along vessels which are relatively thin compared to the candidate branch b.
   (b) The last part of the run-up path consist of the skeleton voxels of the other branch c.
   So, the length of the run-up path should be at least equal to the length of branch c.
   (c) The last part of the run-up path is inside a special local truncated cylinder. The normal of the local plane of the candidate branch b gives the central axis of this cylinder. The radius of this cylinder is given by the radius of the candidate branch b. The length of this cylinder is equal to two times the center radius of the node bc plus the number of skeleton voxels of the candidate branch b.
   Since the cross-section of a "normal" vessel branch is approximately a circle, this cylinder is a faithful extrapolation of the candidate branch b in the direction of the other branch c.
   An example of the last part of a valid run-up path is shown in Figure 7

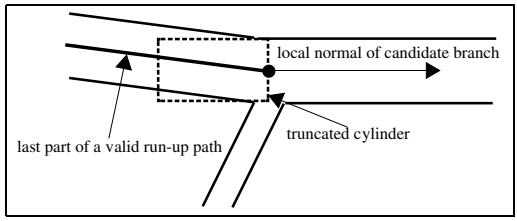


Figure 7: A valid run-up path

   (d) The Manhattan distance between the begin and the end of the run-up path is greater than or equal to the length of the local truncated cylinder.
   The Manhattan distance is used because the length of the run-up path (a set of face connected vessel voxels) is a kind of cumulative Manhattan distance.
   This condition excludes hairpin paths.

   Since the last part of the run-up path consists of the skeleton voxels of the other branch c, these conditions indicate that the other branch c and the candidate branch b form a continuing vessel part.

If a candidate branch forms a continuing branch pair with at least one of the other branches at that node, the candidate branch is classified as a continuing branch.

As already mentioned in Section 3, if a candidate branch does not form a continuing branch pair with

at least one of the other branches at that node, the candidate branch is classified as a solo branch.

3.5 Detection of a Continuing Main Branch

As already mentioned in Section 3, a solo branch is a false branch if at both nodes two of the additional skeleton branches form a *continuing main branch*. We check whether two of the additional branches form a continuing main branch by applying the following tests in the given order:

1. The two additional branches are not classified as a continuing main branch if their radii are less than 0.5 times the radius of the solo branch.
   A continuing main branch should not be a relatively thin vessel compared to the solo branch. As stated in the introduction, a false branch is generated if the erroneous vessel junction is relatively short. A merged branch is generated if the erroneous vessel junction is relatively long. If the radius of the solo branch is much greater than the radii of the continuing main branch, the vessel junction is relatively long. In that case, a false branch is unlikely.
2. The two additional branches are not classified as a continuing main branch if the angle between the two additional branches is less than 90 degrees (the two branches kink).
3. The two additional branches are not classified as a continuing main branch if these branches do not form a fitting branch pair.
   The same test with the same local geometry as described in Section 3.4 is applied.
4. The two additional branches are classified as a continuing main branch if these branches are either a normal branch or a continuing branch.
5. The two additional branches are classified as a continuing main branch if there exist a valid run-up path along the additional branches which are neither a normal branch nor a continuing branch.
   The valid run-up path for an additional branch is similar to a valid run-up path for a candidate branch (see Section 3.4).

3.6 Repair of Unwanted False Branches

The algorithm, described in the previous sections, may result in *unwanted false branches*. These unwanted false branches are detected and repaired as follows. First, *false end nodes* are detected and repaired: a false end node is a node with more than two skeleton branches from which all branches except one are marked as false branches. Since a path for vessel tracing should not continue along a false branch, such a path would end at such node (hence its name). A possible configuration of a false end node is shown in Figure 8. If the branches B3 and B4 are both candidate branches, and if branch B2 is a normal branch and part of a continuing main branch, and if the branch pairs (B1,B3) and (B1,B4) are both continuing main branches, the branches B3 and B4 are both marked as false branches. In this case N1 is a false end node.

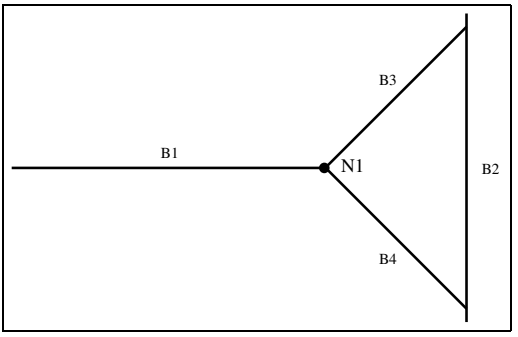


Figure 8: The local graph structure of the false end node N1. B1 and B2 are normal branches. B3 and B4 are marked as false branches.

A false end node is detected easily by counting the number of false branches. A false end node is repaired by marking the false branch which is the best fitting branch (see Section 3.4) of the only non-false branch, as a *fake false branch*. False branches which have almost the same inner product with this non-false branch as this best fitting branch, are also marked as fake false branches. In the example of Figure 8 the false branches B3 and B4 will be both marked as fake false branches.

After that, *erroneous false branches* are detected and repaired. A false branch is an erroneous false branch if this branch is the only connection between two completely separated subparts of the graph structure. Since a path for vessel tracing should not continue along a false branch, an erroneous false branch would prevent vessel tracing along a path from one subpart to the other subpart. Erroneous false branches are detected by generating the min-

imum paths from the first node of the graph to all other nodes. Since we assign to the false branches a very high branch distance, a minimum path along a false branch results in a very high path distance. So, an erroneous false branch is a false branch with a very high path distance for one node of this branch and a normal path distance for the other node of this branch. A very simple schematic example is shown in Figure 9. The node labelled S is the first node of the graph. The nodes labelled L are connected to the first node of the graph via non-false branches. So, these nodes have a low path distance. The nodes labelled H are connected to the first node of the graph via a false branch. So, these nodes have a high path distance. Since the false branch connects a node with a low path distance with a node with a high path distance, this false branch is an erroneous false branch. Erroneous false branches are repaired by marking them as fake false branches.

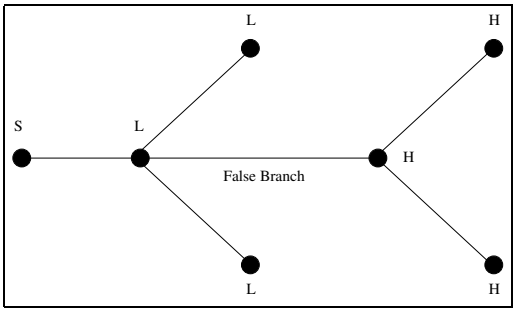


Figure 9: An erroneous false branch with first node S, three low path distance nodes, labelled L, and three high path distance nodes, labelled H.

3.7 Detect and Repair False Short Quadrangles

In some cases, four candidate branches form a quadrangle without diagonals. The general topological configuration is shown in Figure 10. Such a quadrangle needs a special approach, if one of the following three path configurations is true:

1. There are two *horizontal real continuing vessel paths* (with regard to Figure 10, not with regard to the patient), namely $ABCD$ and $EFGH$. In this case the two branches BF and CG are false branches.
2. There are two *vertical real continuing vessel paths*, namely $ABFE$ and $DCGH$. In this case the two branches BC and FG are false branches.
3. There are two *diagonal real continuing vessel paths*, namely $ABGH$ and $DCFE$. In this case the four branches BF, CG, BC and FG are false branches. Besides, we need two new skeleton branches, namely BG and CF. Note that these diagonal continuing vessel paths are not connected!

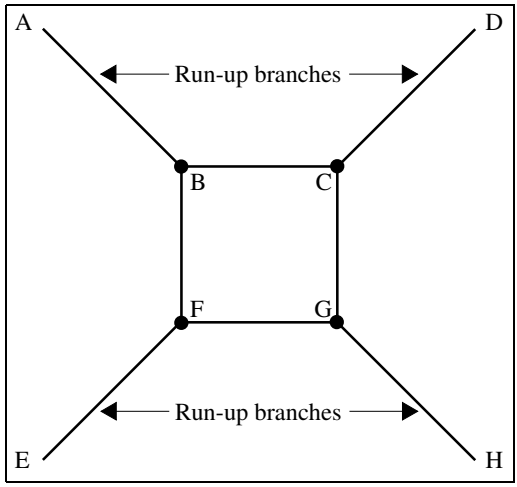


Figure 10: A quadrangle of candidate branches between the nodes B, C, F and G. A, D, E, and H are the nodes connected to the quadrangle nodes via the run-up branches.

The procedure to detect and repair *false short quadrangles* is started after the set of short branches is subdivided in the set of candidate branches and the set of unsuitable branches.

A quadrangle without diagonals is classified as a *short quadrangle* if the following conditions are fulfilled:

1. Each pair of nodes of the quadrangle is connected by at most one skeleton branch.
   More than one skeleton branch between the two nodes makes checking the possible paths not only more complex but also more unreliable.
2. Each of the four nodes of the quadrangle has exactly three skeleton branches.
   If one of the nodes has less than three skeleton branches, that node has no run-up branch.
   If one of the nodes has more than three skele-

ton branches, that node has more than one run-up branch. More than one run-up branch makes checking the possible paths not only more complex but also more unreliable because one run-up branch may disturb the fitting branch pair relation for the other run-up branch. However, in practice, most nodes (quadrangle or not) have no more than three skeleton branches.

3. The skeleton branch between two nodes is either a candidate branch or a real false branch. A real false branch may occur if two short quadrangles are connected.

For each node of a short quadrangle the local geometry is computed (see Section 3.4). The local geometry is used to compute the inner products between the directions of the run-up branches. The branch radii of these branches are used to compute the ratio's between the minimum and maximum radius of two run-up branches.

After that, we check the three path configurations in the given order. There are two horizontal real continuing vessel paths if the following conditions are fulfilled:

1. $0.5 \leq ratio(AB, CD) \leq 2.0$
2. $0.5 \leq ratio(EF, GH) \leq 2.0$
3. $ip(AB, CD) \leq cos(120^\circ)$
4. $ip(EF, GH) \leq cos(120^\circ)$
5. $ip(AB, CD) \leq ip(AB, EF)$
6. $ip(AB, CD) \leq ip(CD, GH)$
7. $ip(EF, GH) \leq ip(AB, EF)$
8. $ip(EF, GH) \leq ip(CD, GH)$

The last four conditions express that the run-up branches of the horizontal paths are aligned as least as good as the run-up branches of the vertical paths.

If the previous tests fail, there are two vertical real continuing vessel paths if similar conditions for the two vertical vessel paths are fulfilled.

If the previous tests fail, there are two diagonal real continuing vessel paths if the following conditions are fulfilled:

1. $0.5 \leq ratio(AB, GH) \leq 2.0$
2. $0.5 \leq ratio(CD, EF) \leq 2.0$
3. $ip(AB, GH) \leq cos(120^\circ)$
4. $ip(CD, EF) \leq cos(120^\circ)$
5. $ip(AB, GH) \leq ip(AB, CD)$
6. $ip(AB, GH) \leq ip(EF, GH)$
7. $ip(AB, GH) \leq ip(AB, EF)$
8. $ip(AB, GH) \leq ip(CD, GH)$
9. $ip(CD, EF) \leq ip(AB, CD)$
10. $ip(CD, EF) \leq ip(EF, GH)$
11. $ip(CD, EF) \leq ip(AB, EF)$
12. $ip(CD, EF) \leq ip(CD, GH)$

The last eight conditions express that the run-up branches of the diagonal paths are aligned as least as good as the run-up branches of the horizontal and vertical paths.

The branch radius of the false branches of a short quadrangle are set to zero, to prevent that these branches are reset to fake false branches (see Section 3.6).

4 Results and Conclusions

We have applied the method to detect and mark false branches of a vessel graph to 53 clinical volume datasets (14 of them with a resolution of 256x256x256 voxels, the rest 128x128x128 voxels), acquired with the 3D Integris system [15]. The voxel size varies between 0.2 and 1.2 millimeter.

An example of the surface model of a vessel structure with false branches is shown in Figure 13. The corresponding graph structure is shown in Figure 16.

The total number of small quadrangles found in all the test sets is 93, the total number of small quadrangles repaired is 43, the total number of real false branches of small quadrangles is 98. The total number of branches is 7798, the total number of candidate branches is 1780 (22.8% of total number of branches), the total number of solo branches is 977 (12.5% of total number of branches) and the total number of real false branches is 276 (3.5% of total number of branches).

The time to detect and mark false branches is a small fraction ($\leq$ 1 %) of the time for fully-automatic branch labelling. The average elapsed time for fully-automatic branch labelling of an 128x128x128 volume is 1.2 seconds on a Linux PC (2.8GHz Pentium 4). The average elapsed time for an 256x256x256 volume is 15.7 seconds.

Visual inspection of each real false branch on top of the surface visualization revealed that not a single real false branch was an erroneous false branch (i.e. a branch which should not be marked as a real false branch). Visual inspection revealed also that not all false branches are detected and marked as a real false branch.

Therefore, our method to detect and mark false branches makes fully-automatic extraction of the

cross-sectional shape parameters in the neighborhood of the diseased vessel parts more reliable. A clinical validation still has to be done.

References

[1] G. Bertrand and G. Malandain. A new characterization of three-dimensional simple points. *Pattern Recognition Letters*, 2(15):169–175, February 1994.

[2] I Bitter, A. E. Kaufman, and M. Sato. Penalized-distance volumetric skeleton algorithm. *IEEE Trans. Visual. Comput. Graphics*, 7(3):195–206, July 2001.

[3] T. Boskamp, D. Rinck, F. Link, B. Kuemmerlen, G. Stamm, and P. Mildenberger. New vessel analysis tool for morphometric quantification and visualization of vessels in CT and MR imaging data sets. *Radiographics*, 24(1):287–297, January 2004.

[4] J. Bruijns. Semi-automatic shape extraction from tube-like geometry. In *Proc. VMV*, pages 347–355, Saarbruecken, Germany, November 2000.

[5] J. Bruijns. Fully-automatic branch labelling of voxel vessel structures. In *Proc. VMV*, pages 341–350, Stuttgart, Germany, November 2001.

[6] D. Chen, B. Li, Z. Liang, M. Wan, A.E. Kaufman, and M. Wax. A tree-branch searching, multiresolution approach to skeletonization for virtual endoscopy. In *Proc. SPIE: Medical Imaging, volume 3979*, pages 726–734, San Diego, CA, USA, February 2000.

[7] P. Dokldal. *Grey-Scale Image Segmentation: A Topological Approach*. PhD thesis, University Marne La Vallee, France, January 2000.

[8] S. Eiho and Y. Qian. Detection of coronary artery tree using morphological operator. In *Proc. IEEE Computers in Cardiology, Vol. 24*, pages 525–528, Lund, Sweden, September 1997.

[9] A. Kanitsar, D. Fleischmann, R. Wegenkittl, P. Felkel, and E. Groeller. Advanced curved planar reformation: Flattening of vascular structures. In *Proc. IEEE Visualization*, pages 43–50, Seattle, WA, USA, October 2003.

[10] G. Langs, P. Radeva, D. Rotger, and F. Carreras. Explorative building of 3D vessel tree models. In *Proc. of 28th annual workshop of the Austrian Association for Pattern Recognition (OAGM/AAPR): Digital Imaging in Media and Education*, pages 117–124, Hagenberg, Austria, June 2004.

[11] S. Lobregt, P.W. Verbeek, and F.C.A. Groen. Three-dimensional skeletonization: Principle and algorithm. *IEEE Trans. Pattern Anal. Machine Intell.*, 2(1):75–77, January 1980.

[12] N. Maglaveras, K. Haris, S.N. Efstratiadis, J. Gourassas, and G. Louridas. Artery skeleton extraction using topographic and connected component labeling. In *Proc. IEEE Computers in Cardiology, Vol. 28*, pages 265–268, Rotterdam, The Netherlands, September 2001.

[13] V. Marion-Poty and S. Miguet. A new 2-d and 3-d thinning algorithm based on successive border generations. In *Proc. 4th Conference on Discrete Geometry in Computer Imagery*, pages 195–206, Grenoble, France, September 1994.

[14] J. Moret, R. Kemkers, J. Op de Beek, R. Koppe, E. Klotz, and M. Grass. 3D rotational angiography: Clinical value in endovascular treatment. *Medicamundi*, 42(3):8–14, November 1998.

[15] Philips-Medical-Systems-Nederland. INTEGRIS 3D-RA. instructions for use. release 2.2. Technical Report 9896 001 32943, Philips Medical Systems Nederland, Best, The Netherlands, January 2001.

[16] F. K. H. Quek and C. Kirbas. Vessel extraction in medical images by wave-propagation and traceback. *IEEE Trans. Med. Imag.*, 20(2):117–131, February 2001.

[17] D. Selle, B. Preim, A. Schenk, and H.O. Peitgen. Analysis of vasculature for liver surgery planning. *IEEE Trans. Med. Imag.*, 21(11):1344–1357, November 2002.

[18] P.J. Yim, P.L. Choyke, and R.M. Summers. Gray-scale skeletonization of small vessels in magnetic resonance angiography. *IEEE Trans. Med. Imag.*, 19(6):568–576, June 2000.

[19] C. Zahlten, H. Juergens, and H.O. Peitgen. *Reconstruction of Branching Blood Vessels From CT-Data*, pages 41–52. Springer-Verlag, Wien, 1995. In Goebel, M., Mueller, H., Urban, B. (Hrsg): Visualization in Scientific Computing.

A Pictures

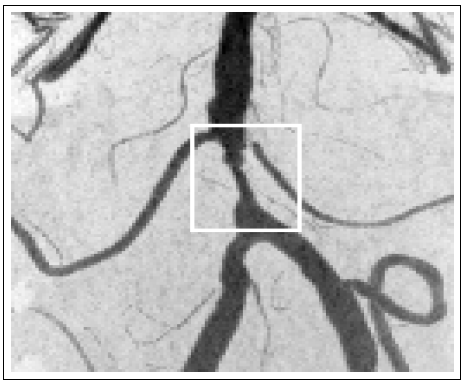


Figure 11: A stenosis inside the white rectangle

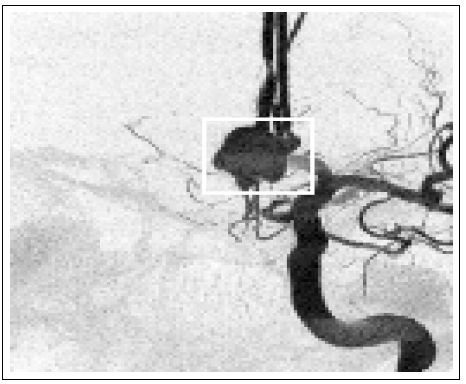


Figure 14: An aneurysm inside the white rectangle

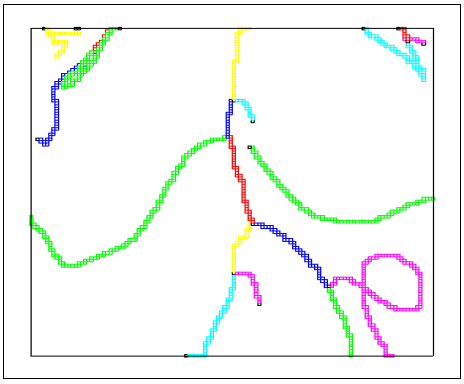


Figure 12: The skeleton voxels of the graphs

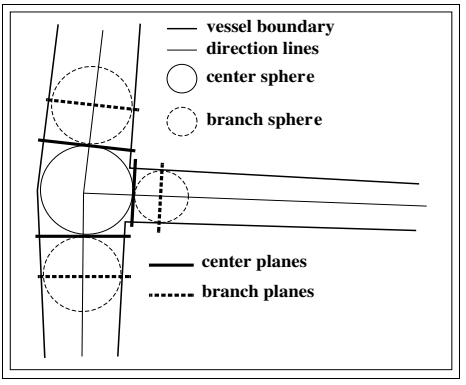


Figure 15: Node geometry

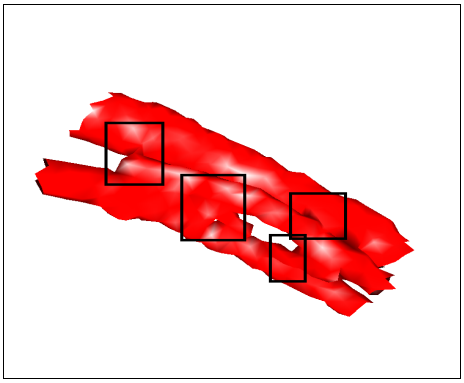


Figure 13: A surface with erroneous junctions in the black rectangles

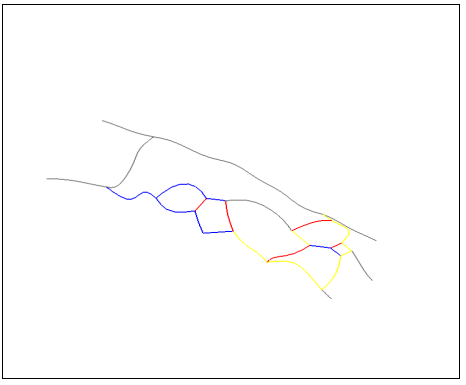


Figure 16: A graph with erroneous junctions. The normal branches are gray, the continuing branches are blue, the solo branches are yellow and the false branches are red.

Internal Labels as Shape Cues for Medical Illustration

Timo Ropinski, Jörg-Stefan Praßni, Jan Roters, Klaus Hinrichs

Visualization and Computer Graphics Working Group (VisCG),
Westfälische Wilhelms-Universität Münster
Einsteinstraße 62, 48149 Münster, Germany
Email: {ropinski, j_prass01, j_rote01, khh}@math.uni-muenster.de

Abstract

In this paper we describe an interactive labeling algorithm, which allows to integrate internal 3D labels into medical visualizations generated from volumetric data sets. The proposed algorithm is motivated by internal labeling techniques found in anatomical atlases, and in contrast to existing algorithms it provides additional shape cues by fitting internal labels to the depth structure of their associated objects. In this paper we discuss guidelines for the layout of internal 3D labels and describe our labeling algorithm, which extends 2D shape fitting and introduces 3D shape fitting. Furthermore we propose related interaction concepts and provide application examples.

1 Introduction

Illustration is an important technique widely used in medical text books to communicate anatomical structures. In recent years computer-assisted interactive generation of medical illustrations from volume data sets acquired by medical scanners has advanced significantly [3, 4, 11]. Textual annotations are essential in order to assign descriptive labels to the objects of interest and thus ease identification of different parts of an illustration. Further on annotating medical data sets is an everyday task performed by radiologists during medical diagnosis. Since for intervention planing and other application scenarios it is important that visualizations can be enriched by labels in a meaningful way, several algorithms have been proposed. Most of these algorithms are inspired by existing illustrations and mimic the layout of the labels found for instance in medical text books like Gray's anatomy atlas [6]. They can be classified into three groups: internal, external and hybrid label layout algorithms. The latter incorporate both internal and external labels. While internal labels are spatially bound to the object of interest, external labels are displayed besides the object and associated with the object of interest using a connection line. Internal labels have the advantage that they allow an easy visual association with the object of interest, since they are placed on top of it and no connection lines are required. In contrast to external labels they occlude parts of the object of interest. Especially for objects with a varying depth structure, placing an internal label on top of it can make the spatial comprehension cognitively more challenging. In medical text books 3D labels are used to deal with this problem. These 3D labels do not only match the shape of the projection of the object of interest, but also match its 3D shape, i.e., the depth structure. As it can be seen in Figure 1, the shape of the objects of interest is emphasized by the distortion of the textual annotation associated with them. For instance is the font of the label *Thoracic aorta* at the ‚lower middle of the image, distorted in a way that propagates that the vessel has a cylindrical shape. In this paper we propose an interactive algorithm, which allows to incorporate such 3D labels into medical illustrations generated from volumetric data sets. Compared to the usage of flat 2D labels, applying our technique allows to provide additional shape cues introduced by these internal 3D labels. Based on the proposed labeling algorithm we will also introduce interaction metaphors supporting an easy manual integration of labels into medical visualizations.

The remainder of this paper is structured as follows. The next section discusses related work about labeling, especially in the context of medical illustration. In Section 3 we will present some guidelines for the layout of internal 3D labels, which we have derived from illustrations found in medical text books. The proposed algorithm for generating

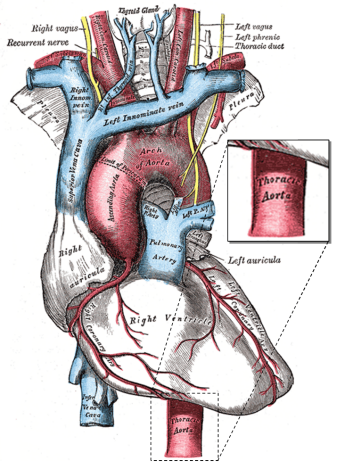


Figure 1: Usage of internal 3D labels as found in the Gray's anatomy atlas [6]. The shape of the structures is emphasized through the text distortion. For instance, the distortion of *Thoracic Aorta* shows that the vessel has a cylindrical shape.

internal 3D labels is described in Section 4, while the label rendering is described in Section 5. Section 6 briefly describes some interaction concepts that can be used to manually integrate labels and edit an existing label layout. Application examples of our technique are discussed in Section 7 before the paper concludes in Section 8.

2 Related Work

Bell et al. [2] were the first who investigated how meaningful annotations can be integrated into virtual environments. They have already identified the main problems of interactive labeling algorithms, which are caused due to changing viewing parameters and possibly occluding objects. Hartmann et al. [7] were the first to address the occlusion problem. While Bell et al. exclusively use internal labels drawn centered on top of the objects of interest, Hartmann et al. introduce floating labels. Floating labels are external labels which change the position when the object of interest is moved or the viewing parameters have been changed.

In the following years Hartmann et al. have further improved their label placing algorithms and have combined internal and external labels [1, 13]. This combination has revealed the need for aesthetic label layout algorithms which produce a visually pleasing placement as found in manually created illustrations [8, 14].

In 2006 Götzelmann et al. [12] have presented an agent-based approach used to deform the text path of internal labels in a manner that the 2D shape of a label fits the 2D shape of the object of interest. This approach has been incorporated into our extension for 2D shape fitting, which is described in Subsection 4.1.

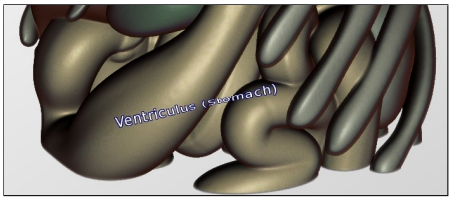
Based on the basic work done for investigating how to achieve interactive annotations, labels have been integrated into various applications, e.g., for medical illustration [4] or in general to combine text and images [5]. To our knowledge the only work towards providing 3D shape cues by means of labels is the application of labels to 3D city models done by Maass and Döllner [10]. They project viewpoint-depending labels onto the bounding boxes of building objects. When the viewpoint changes, the labels are repositioned, and the authors state that this label movement provides shape cues, i.e., the label floating over the invisible bounding box confines the borders of a building.

3 Guidelines for Internal 3D Label Layout

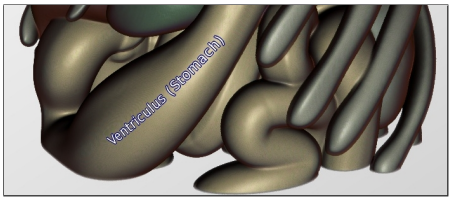
As mentioned above different hybrid label layouts have been described by Götzelmann et al. as well as Hartmann et al. [8, 14]. In this paper we introduce design rules for the layout of internal labels, whereas we do not exclusively focus on the positioning of internal labels but also on their 3D orientation. The presented guidelines are motivated by the layout of internal labels as found in medical illustrations.

As already proposed in [12] for internal 2D labels, internal 3D labels should also fit the 2D shape of the object of interest. A possible way to achieve this goal is to orient a label along the medial axis of its object of interest as it is defined in image space. This criterion ensures a 2D association between an object and its label solely by this 2D path. However, when extending the labels to 3D, additional layout guidelines should be taken into account in order to achieve a satisfying label placement.

First of all, when labels are distorted based on the underlying surface, it has to be ensured that the



(a)



(b)

Figure 2: Internal 3D labels crossing contours may lead to unwanted distortion and have a disturbing effect (a). By avoiding contour crossing this distortion can be prevented (b).

labels still remain readable. Therefore a tradeoff between label distortion and 3D shape fitting, i.e., alignment along the associated surface, has to be taken into account. In order to ensure readability a very bumpy surface should not be labeled by simply projecting a label onto it. Moreover the characteristic surface structures should be considered by omitting the bumpy arbitrary surface shifts. A possible realization would be using Bezier surfaces to approximate the surface of the object of interest. By using Bezier patches having a high polynomial degree, the surface is approximated very closely and surface details will be reflected. In contrast, when minimizing the polynomial degree of the approximating Bezier surface, it acts as a smoothing filter only reflecting the larger scale surface structure. In Subsection 4.2 we will describe how we will take this approximation into account.

Besides bumpy surfaces, an internal 3D label may also be distorted when crossing a contour (see Figure 2). Therefore internal 3D labels should be positioned in a way that they do not cross contours of the objects of interest.

Furthermore the perspective distortion of an internal label should be minimized. Therefore the layout should try to maximize the number of labels which are aligned almost perpendicular to the viewing direction in order to improve readability.

In the following section we will explain how to follow these guidelines, by among others exploiting object normals as well as contour detection.

4 Generating 3D Labels

By considering the guidelines proposed in the previous section, we have developed a labeling algorithm for automatically enriching a medical volume visualization with internal 3D labels. The proposed algorithm combines a 2D and a 3D shape fitting approach. With the 2D shape fitting we ensure that the path of the label matches the shape of the projection of the current object of interest given in image space. After this path has been determined we analyze the depth structure at the appropriate positions and generate Bezier patches to fit the 3D shape. Afterwards the label itself is rendered by texturing the generated patches. In the next two subsections we describe the necessary steps for performing the 2D and the 3D shape fitting. The workflow of the algorithm is illustrated in Figure 3.

4.1 2D Shape Fitting

To perform the 2D shape fitting, we use a modified version of the agent-based approach proposed in [12]. After the rendering parameters (RP), e.g., the transfer function, have been specified, we render a segmented volumetric data set into an *ID map* as well as a *depth map*. In the ID map each pixel has a unique color which has been associated with the segment it belongs to. Thus the region covered by each segment in image space can be determined, and its medial axis can be calculated. To comply with the guidelines of the previous section and to ensure that the internal labels do not cross any contour edges, we apply an image-based contour detection before calculating the medial axis. This contour detection is performed by applying a 3×3 filter kernel to the depth map in order to identify discontinuities of the depth values. Pixels having a high depth discontinuity belong to contour edges and are drawn as black pixels onto the ID map, resulting in the *edged ID map*. Based on the edged ID map we can perform the medial axis calculation, which is done similar to the technique described in [1]. Since we have initially super-imposed the contour edges, the resulting *distance map* contains no medial axis crossing any contour edges. However, using our

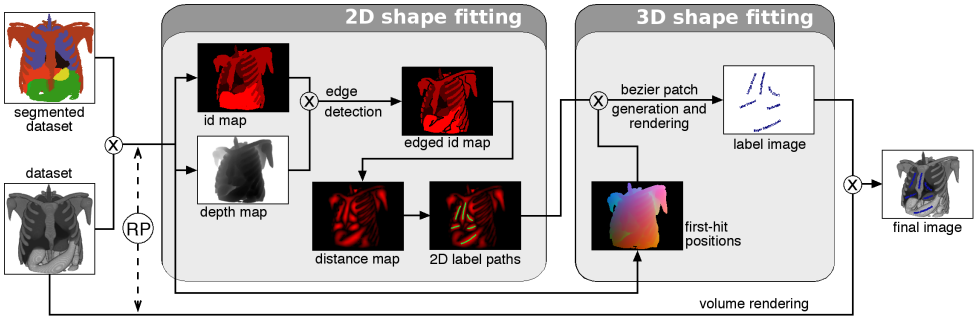


Figure 3: **Workflow** of our algorithm, with all steps used to render internal 3D labels. After the rendering parameters (RP) have been set, a 2D shape fitting is performed to find a label path matching the shape of the object of interest, before 3D shape fitting is used to match its depth structure.

distance map we can apply the algorithm as proposed in [12] to determine the 2D label paths. The goal of this technique is to find an approximation of the medial axis implicitly defined by the distance map. The used approach first determines for each segment the pixel with the maximum distance to the segment's border. In the next step, in contrast to the previous approach we do not consider all directions on a circle around the current position, but look only at those directions with maximal distance to the border to find the 2D path. This can be done by using our intermediate maps, which are generated when calculating the final distance maps. Since in the intermediate maps we have the distances separated for the three main directions, i.e., horizontal, vertical and diagonal. Thus by exploiting the current position and the determined direction, an arc with a certain angle and a given radius defines the pixels that are checked, in order to determine the two pixels among them having the maximum distance to the segments border. This process is repeated iteratively until the desired number of control points is found or the border of the segment is reached. In addition to the technique described in [12], we also consider the surface normals in order to ensure that labels do not appear on surfaces being almost parallel to the viewing direction, since their perspective distortion would be too high. Finally, the 2D label path is calculated by using a 2D fitting curve defined by the estimated control points.

With the thus computed 2D label paths we enter the next stage of our algorithm to perform the 3D shape fitting.

4.2 3D Shape Fitting

Now that the 2D label paths are available we take into account the underlying depth structure in order to generate Bezier patches approximating a segment's surface. To determine the control points for the used Bezier patches, we consider the volume coordinates for each pixel. When using GPU-based ray-casting [9] as in our system, the volume coordinates for a surface can be easily obtained by writing the first-hit positions into a 2D texture. Thus we can fetch for each point lying on the 2D path its corresponding volume coordinate and get the resulting 3D path. This 3D path is approximated by a 3D fitting curve defined by the control points later referred to as s_i and can be thought of as an approximation of the medial axis of the Bezier surface used to project the labels onto. Based on this medial axis we construct the Bezier patches as illustrated in Figure 4. As it is shown the s_i serve as control points for the Bezier patches. Additional control points are generated by considering the main normalized path direction of the current segment l as well as the normalized gradient g at each s_i. The distance of the newly computed control points to the s_i is given by the volume space offset h. Thus we can construct the outermost control points for the Bezier patches by calculating $m = h \cdot (l \times g)$ as well as $-m = -h \cdot (l \times g)$ and projecting these points back on the object's surface as shown in Figure 4. Since these are the outermost control points, h directly determines the label height. Depending on the degree of the Bezier patches we may add additional control points along $l \times g$ or between two successive

s_i. While a higher horizontal degree, i.e., along the main path direction, may be reasonable a high vertical degree, i.e., along $l \times g$, is usually not necessary. Instead a higher vertical degree may result in decreased readability through high perspective distortion.

5 Label Rendering

Based on the Bezier patch control points derived as explained in the previous section, the Bezier patches are generated to serve as a surface for the labels. The actual label rendering is done by exploiting 2D texturing functionality. Therefore we simply use the Bezier patch parameters s and t as texture coordinates in order to apply the labels, which we have rendered into a bitmap initially. During this bitmap generation we render the text with twice the desired font size. This ensures that we get smoother looking labels, since we can exploit the filtering capabilities of the texturing hardware during the minification process.

To increase contrast and further enhance readability we additionally render a halo around each letter. This is also done in the preprocessing rendering pass where the label textures are generated. Each label is rendered four times using the halo color before rendering it once in the current font color. During each halo rendering pass the label is shifted by one pixel in the four main directions. This ensures that the text rendered on top of this halo is surrounded by the halo color in all directions. The effect of the halo color becomes clearly visible in the Figures shown in Section 7.

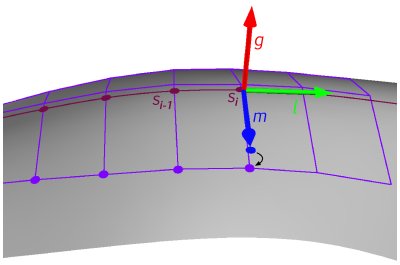


Figure 4: The control points for the Bezier patches are generated by adding points lying on the surface vertically perpendicular to the respective gradient.

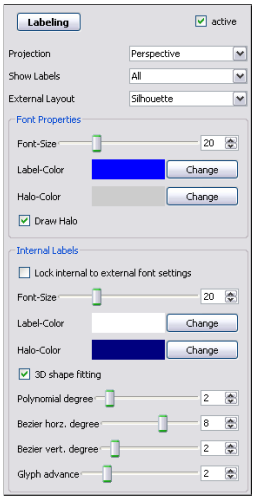


Figure 5: The graphical user interface used for interactive label manipulation. The user can change type, text and position of each label as well as change the rendering parameters.

During rendering we possibly might run into occlusion issues. Since the control points of the used Bezier patch lie on the object's surface and the patch lies in the convex hull of its control points, it may possibly intersect the surface. In these intersection regions the text would be occluded by the objects of interest. Therefore we ensure that the depth test is disabled when rendering the labels.

6 Interactive Label Manipulation

Although the automatically generated label layout is sufficient for most illustration cases, sometimes it is desirable to apply modifications manually or even integrate labels manually as done by radiologists during medical diagnosis. Therefore we have integrated simple though effective interaction concepts into our user interface which allow the user to perform the label positioning manually (see Figure 6) and to modify the most important label properties (see Figure 5).

The automatic layout proposed by our system is a hybrid layout, where internal 3D labels are used for those objects of interest which provide sufficient screen space. All other objects are annotated by using external labels. The user can reposition internal as well as external labels via drag-and-

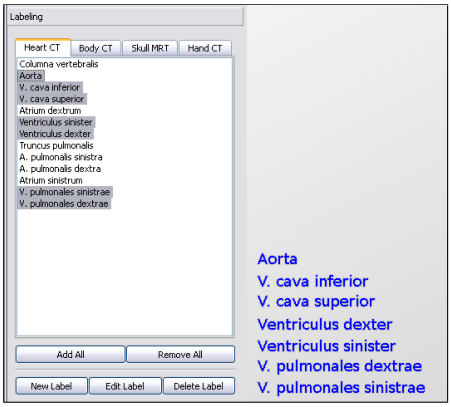


Figure 6: When dealing with non-segmented data sets, the user may select a series and add labels from that series manually to the image. Furthermore presets for frequent series can be modified by adding or removing labels.

drop, whereas the 3D shape of the internal labels is adapted to their current position automatically. In cases where the user drags an internal label outside the screen space occupied by its associated object of interest, the label turns into an external label, i.e., a connection line is added. By dragging the label back to *its* object of interest it can be reverted into an internal 3D label. Thus the user can change type and position of all labels.

The automatic layout algorithm works only in those cases where a segmentation for the current data set is available. Otherwise it is not possible to generate an ID map, which is required to calculate the label positions (see Subsection 4.1). However, often in medical diagnosis when annotation is required, no segmentation is present. For these everyday tasks we have integrated a simple though effective manual annotation technique. As it is shown in Figure 6 the user interface provides predefined label sets for different cases of diagnosis, e.g., heart CT or head MRI. These sets contain the labels which the physician expects to use during a diagnosis based on the given data set. While it is possible to add and remove label sets, they can also be modified by adding, deleting or editing labels in order to serve the needs of the physician. Once the required label sets are defined, a physician may add the labels contained in a set to the current image. This can be either done by pressing the *Add All* button or by applying a possibly multiple selection in the label set. Once the labels to be used have been specified, they appear in the lower right corner of the 3D canvas and can be dragged to the desired position. As described in Subsection 4.2, the labels also match the surface structure of the underlying objects. After the labels have been positioned, their coordinates may be saved in order to reproduce the renderings. Furthermore rendering parameters, e.g., the transfer function or the thresholding, can be changed during this task in order to be able to associate labels with internal structures.

Further on the user may alter the most important label properties. As it can be seen in Figure 5 the user may select which subset of labels is to be shown: all, internal only or external only. Additionally she may select the type of external label layout - currently silhouette layout as well as side layout is supported. Silhouette layout aligns the external labels along the objects silhouette, side layout renders the labels on the left and right side of the image. To adjust the look of the individual labels, font size, label color and halo color can be adapted. Furthermore the halo can be switched on and off. Changing the font as well as halo color is especially necessary for internal labels in order to increase the contrast between the label and its background color given by its object of interest. Therefore these options can be changed separately for internal labels. To adjust the shape of the internal labels, 3D shape fitting can be switched on and off and can be modified by adapting the degrees of the used fitting curves.

The text of a label can also be changed easily by performing a double click on top of it. All changes made are saved within an XML document which is associated with the segmented data set.

7 Application Examples

With the techniques proposed in the previous sections we are now able to integrate internal 3D labels into existing medical applications. As noted above, this approach is motivated by existing medical illustrations where text distortion provides an additional shape cue. In Figure 7 a comparison of 2D and 3D internal labels is shown. In both cases the text path is determined by exploiting the shape of the object of interest given in image space. In contrast to the 2D labels the 3D labels allow a better shape perception. This becomes clear when inspecting the region

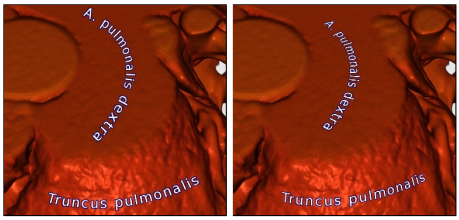


Figure 7: Comparison of internal 2D labels (left) and the introduced 3D labels (right). Although shading provides shape cues in both cases, the *floating* labels in the left column make estimation of distances cognitively more challenging.

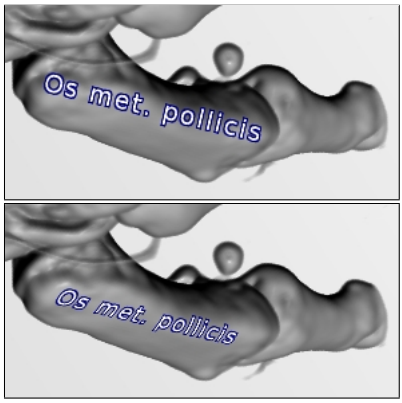


Figure 8: Comparison of internal 2D labels (top) and the introduced 3D labels (bottom). The internal 3D labels smoothly approximate the surface structure.

labeled as *A. pulmonalis dextra*. In the left image it is rather difficult to estimate the orientation of the cutting plane, the label is associated with. When using internal 3D labels the text distortion gives an additional cue for comprehending this orientation. Figure 8 shows a similar effect when visualizing curved surfaces. With 3D labels the curvature of the surface comes out more clearly, since the text is distorted accordingly. For instance when inspecting the distortion of the *ll* in *pollicis*. All these effects can be enhanced by using an appropriate label placement with the interaction techniques discussed in the previous section.

In Figure 9 the application of our technique to the NCAT phantom data set is shown. Since the *Columna vertebralis* does not provide enough

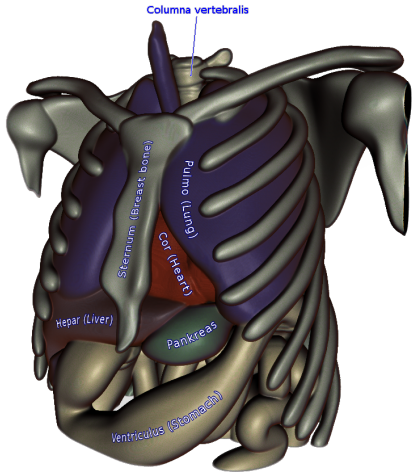


Figure 9: Application of our internal labeling technique to the NCAT phantom data set.

screen space for positioning an internal label in the current view, an external label has been used automatically. Again the effects of surface structure emphasize becomes visible. One may argue that undistorted 2D labels would result in better readability. However, it must be considered that there is a tradeoff between the readability of the text and the perception, or readability, of the whole illustration. Obviously the best label readability would be achieved, when not even using 2D shape fitting and just integrating horizontal labels. When looking at medical text books, it can be seen that the reduced readability is accepted and 2D as well as 3D shape fitting is used. This approach has been evolved over the years and ensures besides improved perception of the illustration the characteristic style of medical illustrations. In addition to the usage of text distortion in classical medical illustrations, our approach also allows the illustrator to easily tune the parameters in order to get the most convincing representation. For instance, the degree of the fitting curves can be changed manually in order to emphasize or deemphasize certain spatial structures.

Other examples of hybrid label layouts are shown in Figure 10 and Figure 11. In contrast to the other figures, where we exploit Phong shading, a quantized toon shader has been used in Figure 10. To further enhance the illustrative effect the silhouette has been highlighted. In addition to the improved

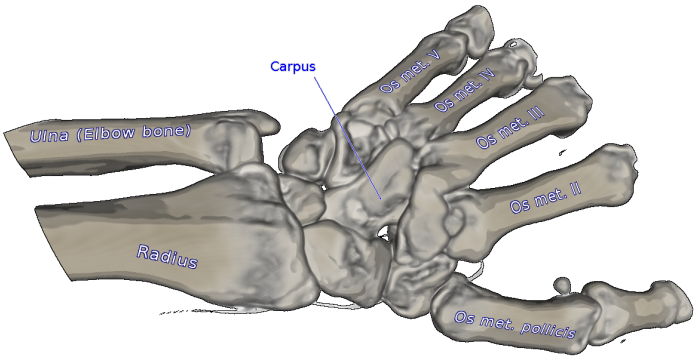


Figure 10: An automatically annotated illustration of a CT scan of a human hand. The internal labels are positioned appropriately and fit smoothly to the surface of the metacarpus bones.

shape perception through internal 3D labels, Figure 11 shows also that the association between a label and its object of interest is cognitively less demanding when using internal labels in comparison to external labels.

We have shown the images of Figure 9-11 as well as the corresponding versions using internal 2D labels to our medical partners. We got very positive comments and the physicians especially liked the internal 3D label approach to be used in anatomical illustrations containing parts of the skeleton as in Figure 10.

8 Conclusion

In this paper we have proposed an interactive algorithm for the generation of internal 3D labels. The technique has been motivated by labeling techniques found in medical illustrations and is also applicable during the interactive annotation process as used in medical diagnosis. In contrast to commonly used internal 2D labels, 3D labels have the benefit that they may provide additional shape cues. Since the distortion of the text being visualized on top of the object of interest is influenced directly by its 3D shape, spatial comprehension is expected to be improved. We have presented guidelines for the layout of internal 3D labels, have described our interactive rendering technique and presented some application examples.

Since text distortion may also influence readability, a detailed evaluation is necessary in order to find an optimal tradeoff between readability and spatial comprehension. However, it should be noted again that optimal readability can only be achieved when using horizontal labels, without applying any shape fitting. Since this would make it more difficult to perceive the association between the objects and the labels as well as the object's shape, we believe that shape fitting is a good way to enrich visualizations having internal labels. Although we already got positive comments about the interactive annotation mechanism from radiologists, we plan to evaluate this technique in order to further improve its usability for medical diagnosis.

While in this paper we focussed on volumetric visualization, it should be stated that the proposed technique can also be applied to polygonal rendering. Moreover it can be used in any visualization setup, as long as a depth image and eventually a segmentation is available.

Acknowledgements

This work was partly supported by grants from the Deutsche Forschungsgemeinschaft (DFG), SFB 656 MoBil Münster, Germany (project Z1).

References

[1] Kamran Ali, Knut Hartmann, and Thomas Strothotte. Label Layout for Interactive 3D Illustrations. In *Proceedings of the 13th In-*

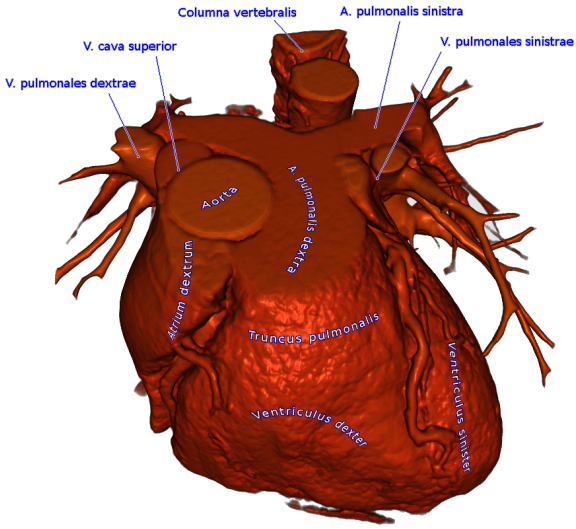


Figure 11: Manually annotated visualization of a CT scan of the human heart. A hybrid label layout is used, while the internal labels are represented as 3D labels providing additional shape cues.

ternational Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG05), pages 1–8. UNION Agency - Science Press, 2005.

[2] Blaine Bell, Steven Feiner, and Tobias Höllerer. View management for virtual and augmented reality. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 101–110, New York, NY, USA, 2001. ACM Press.

[3] Stefan Bruckner, Sören Grimm, Armin Kanitsar, and Meister Eduard Gröller. Illustrative context-preserving exploration of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1559–1569, 11 2006.

[4] Stefan Bruckner and Meister Eduard Gröller. Volumeshop: An interactive system for direct volume illustration. In *Proceedings of IEEE Visualization 2005*, pages 671–678, 2005.

[5] T. Götzelmann, M. Götze, K. Ali, K. Hartmann, and Th. Strothotte. Annotating Images through Adaptation: An Integrated Text Authoring and Illustration Framework. *Journal of the WSCG*, 15(1), 2007. (in print).

[6] Gray. *Gray's anatomy atlas*.

[7] K. Hartmann, K. Ali, and Th. Strothotte. Floating labels: Applying dynamic potential fields for label layout. In A. Krüger P. Olivier A. Butz, B. Fisher, editor, *4th International Symposium on Smart Graphics 2004, LNCS 4073*. Springer Verlag, 2004.

[8] K. Ali K. Hartmann, T. Götzelmann and Th. Strothotte. Metrics for functional and aesthetic label layouts. In *5th International Symposium on Smart Graphics, LNCS 3638*, pages 115–126. Springer Verlag, 2005.

[9] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proceedings IEEE Visualization 2003*, 2003.

[10] Stefan Maass and Jürgen Döllner. Dynamic Annotation of Interactive Environments using Object-Integrated Billboards. In *Proceedings of the 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG06)*, pages 327–334. UNION Agency - Science Press, 2006.

[11] Felix Ritter, Henry Sonnet, Knut Hartmann, and Thomas Strothotte. Illustrative shadows: integrating 3d and 2d information displays. In *IUI '03: Proceedings of the 8th inter-*

national conference on Intelligent user interfaces, pages 166–173. ACM Press, 2003.

[12] K. Hartmann T. Götzelmann and Th. Strothotte. Agent-Based Annotation of Interactive 3D Visualizations. In A. Krüger P. Olivier A. Butz, B. Fisher, editor, *6th International Symposium on Smart Graphics 2006, LNCS 4073*, pages 24–35. Springer Verlag, 2006.

[13] K. Hartmann T. Götzelmann, K. Ali and Th. Strothotte. Adaptive labeling for illustrations. In C. Gotsman E. Wu, D. Manocha, editor, *Proceedings of Pacific Graphics*, pages 64–66, 196. The 13th Pacific Conference of Computer Graphics and Applications, 2005.

[14] K. Hartmann T. Götzelmann, K. Ali and Th. Strothotte. Form follows function: Aesthetic interactive labels. In B. Gooch W. Purgathofer L. Neumann, M. Sbert, editor, *EuroGraphics Workshop on Computational Aesthetics in Graphics, Visualization and Imaging*, pages 193–200. Eurographics Association, 2005.

Interactive Model-based Image Registration

Thomas Schiwietz[1,2] and Joachim Georgii[1] and Rüdiger Westermann[1]

[1] Technische Universität München
[2] Siemens Corporate Research
Email: {schiwiet,georgii,westermann}@in.tum.de

Abstract

We present an interactive technique for the registration of captured images of elastic and rigid body parts in which the user is given flexible control over material specific deformation properties. Our method can effectively handle arbitrary stiffness distributions and it achieves an accurate matching without sacrificing the physical correctness of the simulated deformations. The algorithm consists of three steps, which are performed iteratively until an optimal spatial mapping is determined: First, the optical flow is used to predict an initial image transformation. Second, a priori knowledge of the deformation model is used to refine the predicted field. A physics-based filter operation generates a transformation that is consistent with the model of linear elasticity. Third, the process is repeated using the displaced template image. To achieve accurate image deformations we employ implicit multigrid solvers using finite differences (optical flow) and finite elements (linear elasticity). The robustness and accuracy of our method is validated using synthetic and real clinical data composed of heterogeneous materials exhibiting different stiffness characteristics.

1 Introduction

Especially in medical applications there is an ever growing interest in techniques that can accurately relate information in multiple data sets resulting from different measurements. Image registration techniques are designed for that purpose as they try to find the correspondence between images of the same anatomical structure taken under different conditions (e.g. different relative camera-patient position, different methods of acquisition, etc.). Such techniques compute a spatial mapping of structures into a common coordinate system or aim at compensating individual structural characteristics and differences due to movements over time.

One of the main challenges in image registration is to accurately model and simulate the deformation behavior of the structures contained in the data. Scanned body parts are usually composed of highly heterogeneous tissue types and in particular an elastic modulus with a dynamic range of several orders of magnitude is not unusual to be found in a single scan. Registration techniques capable of dealing with such data require to adjust the stiffness of the transformation to permit realistic displacements over all parts in the data. Physics-based deformation models then have to be considered to realistically simulate the structural changes.

In this work, we present an algorithm for image registration that addresses the aforementioned requirements by using a predictor-corrector approach to iteratively compute an optimal spatial mapping. The predicted deformation field can be computed from image pairs using any standard evaluation method. The corrector step implements a finite-element solver for the Lagrangian equation of motion simulating linear elasticity. It takes as input the predicted vector field and modifies it according to the underlying physical model. As all parts of the algorithm run interactively, intuitive control mechanisms to flexibly adjust the deformation properties of specific regions in the data are integrated.

In particular, we utilize a GPU-based random walker [14] for assigning specific stiffness values to parts of an image. While it is generally possible to assign these properties using a painting tool on a per-pixel basis this is a tedious and time-consuming task not feasible in clinical practice. Instead, we use the segmentation algorithm to assign stiffness on a per-object basis. The segmentation algorithm extracts parts from the image automatically by single mouse clicks or strokes on top of the object, and it then fills the object's interior with a user-defined stiffness.

1.1 Related Work

With respect to the assumed image transformation, image registration techniques can be classified into parametric and non-parametric approaches. Comprehensive surveys of these approaches along with a number of application areas can be found in [20, 28, 21]. Parametric approaches impose specific restrictions on the transformation, e.g. requiring the transformation to be rigid, polynomial or affine. Non-parametric, i.e. non-linear, approaches, on the other hand, are far more flexible in the type of transformation they compute. Such transformations rely on additional constraints like the regularization of the displacement field, which can be enforced by explicit smoothing of the deformation field [1, 22, 25]. Over the last years special emphasis has been put on the development of variational PDE approaches to obtain approximate solutions to the non-linear registration problem [7]. In this case an additional smoothing term is considered in the global energy function to be minimized. In particular, physics-based regularizers based on linear elastic and viscous fluid models have been shown to be effective in penalizing unrealistic local and global displacements [6, 27, 26, 4]. While finite difference methods are frequently applied to discretize the linear elasticity equations [21], finite element methods are so far rarely applied to the best of our knowledge. This is due to the fact that they are computationally more complex, and thus result in poor performance rates [15]. However, as has be shown by Georgii and Westermann [12], multigrid algorithms can efficiently be applied to speed up the solution process of finite element techniques.

Curvature-based constraints for non-linear registration problem have been discussed in [11, 16]. While effective in computing deformations that comply with the incorporated regularization models, only a few approaches have attempt to also take heterogeneous materials exhibiting varying stiffness into account [9, 19, 18].

2 Method

We propose a model-based approach for the computation of a non-linear transformation of a template image T onto a reference image R. A priori knowledge about the physical deformation model is exploited to make the transformation consistent with this model. Our algorithm proceeds in multiple steps, each of which is performed interactively to accommodate steering of model-based parameters by the user. The major parts of our algorithm are illustrated in Figure 1.

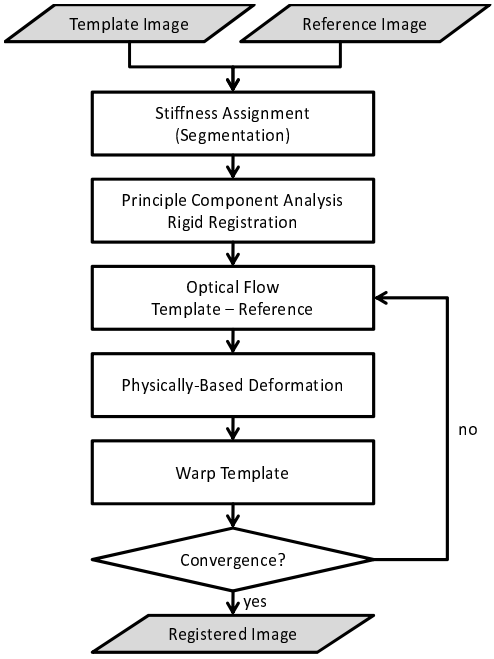


Figure 1: Overview of major parts of the registration system. A template and a reference image are registered using our algorithm: After stiffness assignment the images are rigidly registered by PCA. Then, our iterative loop estimates a vector field using the optical flow method. This field is corrected by physically based image deformation. The loop stops if the convergence criterium is met.

We utilize the Random Walker segmentation algorithm [14] to automatically extract objects from the image, for which we then specify the respective stiffness value. For our purposes, non-binary segmentation algorithms like the Random Walker provide attractive properties since the segmentation results yield a smooth transition between the segmented object and the background. A smooth stiffness distribution is required by the finite element method, since material stiffness is only approximated C^0-continuous, and thus large jumps can introduce instabilities. The smooth transition is highly desirable for stiffness assignment because

the segmentation result can be mapped directly to stiffness using a transfer function. In this way, abrupt stiffness changes in the simulation grid can be avoided at no extra cost, enabling realistic deformations at boundary transitions.

In a pre-process a rigid PCA (principle component analysis) registration is performed to obtain an initial mapping between T and R. Therefore, the covariance matrix of both images is computed by weighting pixel-positions with the intensity values. An eigenvalue decomposition of that matrix reveals the relative rotation and scaling of the two images. The relative translation is the distance between the mean positions of both images.

2.1 Gridding

From the segmented and pre-registered template a regular simulation grid consisting of quadrangular elements is constructed by placing exactly one grid vertex at each pixel center. Pixel colors and associated stiffness values are respectively assigned as vertex and element attributes. Given such a grid in the reference configuration $x \in \Omega$, a deformed grid is modelled using a displacement function $u(x), u : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ yielding the deformed configuration $x + u(x)$. The transformed template image is then generated by rendering the deformed grid onto a regular pixel grid. This image is used in the next iteration to calculate the optical flow.

2.2 Deformation Estimate

An initial deformation estimate is computed using classical optical flow as introduced by [17]. The main idea behind this algorithm is to minimize a global cost function representing the rate of change of image brightness from one image to the other. As the optical flow estimates the apparent motion of brightness patterns in two images it is clear that this particular kind of deformation estimate cannot be applied in general for multi-modal registration. However, the optical flow works fine for single modalities, where all images are generated with the same scanning conditions. In the multi-modal case, deformation estimates based on common similarity measures like mutual information have to be favored [21].

Since techniques based on the optical flow are considered to be rather slow due to the numerical complexity of the employed solvers, in a number of research projects considerable effort has been put in the development of advanced numerical techniques like multigrid schemes. In this work, we make use of the implementation by Bruhn et al. [4] to significantly speedup the prediction of an initial deformation field.

2.3 Model-based Correction

Rather than penalizing the optical flow with a model-based regularizer as proposed in [23], the optical flow field is used as an external force field driving the deformation of structures contained in the template image. The deformation is based on a linear elasticity model, where the dynamic behavior is governed by the Lagrangian equation of motion. An implicit multigrid solver presented in [13] is extended to efficiently compute the resulting displacements.

In contrast to previous registration approaches using finite differences, it is worth noting that our method employs a finite-element discretization. In this way, improved physical accuracy is achieved, but we have to pay for that by an increasing numerical complexity. As our timings show, however, the multigrid solver we have developed performs favorable to respective finite difference techniques.

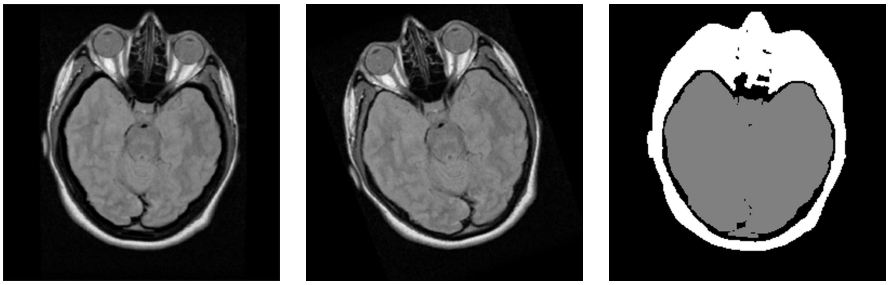
By using a model-based approach as described it is clear, that the simulated displacement field enforced by the elasticity equation differs from the optical flow field. Physically speaking, the optical flow field is corrected towards a displacement field that complies with the underlying model.

2.3.1 Linear Elasticity Model

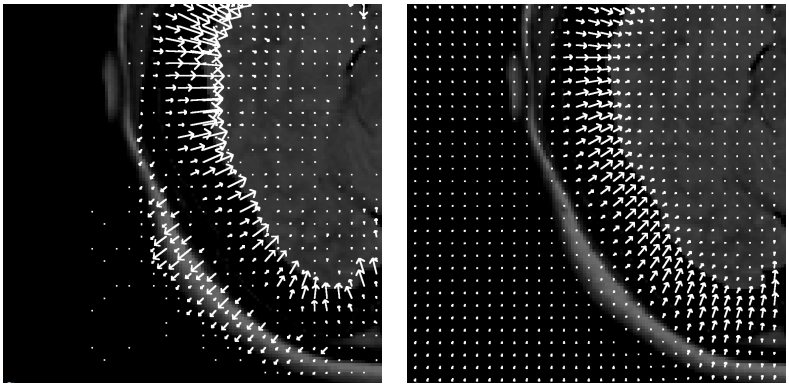
If the object to be simulated obeys to the model of linear elasticity the dynamic behavior is governed by the Lagrangian equation of motion

$$M\ddot{u} + C\dot{u} + Ku = f \tag{1}$$

where M, C, and K are respectively known as the mass, damping and stiffness matrices. u consists of the linearized displacement vectors of all vertices and f are the linearized force vectors applied to each vertex. It is worth noting that in linear elasticity we only consider material that has a linear relationship between how hard it is squeezed or torn (stress) and how much it deforms (strain). This relationship is expressed by the material law, which is accounted for by the stiffness matrix K.



(a) Reference (b) Template (c) PCA/Seg.



(d) Prediction (e) Correction

Figure 2: The template image (b) is registered to the reference image (a) by first applying a PCA rigid registration (c). Then, stiffness values are assigned to segmented parts (c). The iterative loop predicts a vector field (d) using the optical flow method and corrects it by our physical deformation model (e) until convergence is reached.

Equipped with any suitable discretization finite element methods typically built these matrices by assembling all element matrices to yield a sparse system of linear equations. For the details on the discretization process as well as the numerical schemes used to solve the resulting system let us refer the reader to [2, 3].

2.3.2 Finite Element Method

To improve simulation accuracy we have integrated quadrangular elements with bilinear nodal basis functions into our approach. Quadrangular elements consist of 4 supporting nodes $\underline{v}_k$, thus interpolating the deformation in the interior as

$$u(x) = \sum_{k=1}^{4} N_k(x)\underline{u}_k$$

where

$$N_k(x) = c_0^k + c_1^k x_1 + c_2^k x_2 + c_3^k x_1 x_2$$

and $\underline{u}_k$ is the displacement of the k-th node. The coefficients c_i^k can be easily derived from $N_k(\underline{v}_k) = 1$ and $N_k(\underline{v}_i) = 0$ if $k \neq i$. The shape functions $N_k(x)$ and its derivatives are needed to build the final element matrices, which are then assembled into the global stiffness matrix,

By using quadrangular elements the overall number of elements is reduced. This allows for a slightly faster assembly process of the global system matrix. Furthermore, the semi-quadratic interpolation scheme increases the simulation accuracy and thus improves physical correctness.

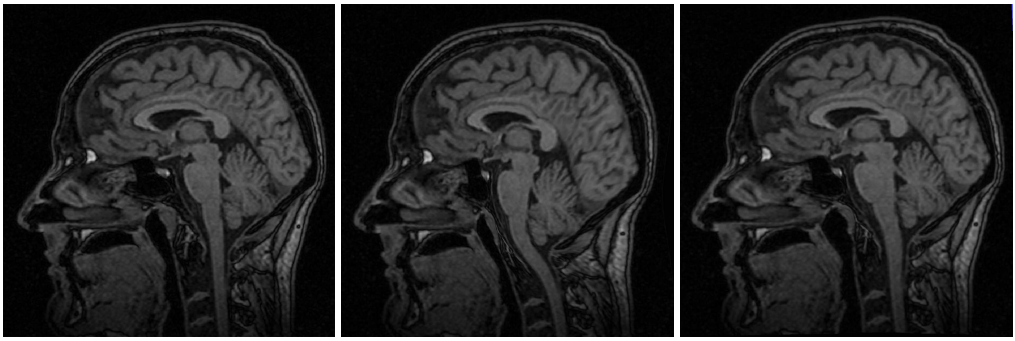


Figure 3: The first image show the reference configuration. The second image was generated by a physically deformation using the finite element method. The third image shows the result of our registration algorithm.

3 System Assembly

In the following, we will show how we interconnect the different parts of our algorithm into one system. An illustration of our algorithm is depicted in Figure 2.

Once the user has selected two images to register, stiffness values are assigned to all pixels of the template image using a painting tool, a segmentation algorithm (Random Walker), or other automatic assignment methods. Once the stiffness assignment is done, the images are pre-registered using PCA.

All the upcoming steps are then performed automatically, including mesh generation, deformation estimate, model-based correction and image warping. These steps are performed iteratively until the template image matches the reference image with respect to any suitable metric, i.e. the mean square deviation of the current per-vertex deformations to the previous ones.

The iterative registration loop starts by computing the optical flow field between the current warped template image and the reference image. Next, optical flow vectors are applied as external forces to the finite element vertices. Bilinear interpolation of the four closest optical flow vectors is used to obtain the force for one finite element vertex in the deformed grid. Note that the external forces are accumulate in each step while the previous accumulated force is damped by a small percentage. The average force vector of all finite element vertices is subtracted from all vertices in order to keep the mesh in place. Since we cannot compute the exact force field required for convergence in one single step, we choose an iterative approach using a dynamic simulation. Thus, in every time step we only do a small step into the direction predicted by the optical flow force field; this loop is iterated until convergence is finally achieved.

To correct the predicted optical flow field via the linear elasticity module we simulate the deformation over one time step until the multigrid solver converges.

Resulting displacements are send to the graphics card where the simulation mesh is updated accordingly. This mesh is then rendered onto a regular pixel grid using hardware supported interpolation to generate the warped template image used in the next iteration. As the update step is entirely performed on graphics hardware, and because sending displacement values from the CPU to the graphics card does not introduce any time constraints, its contribution to the overall runtime is negligible. Note, that both the optical flow and the physical correction are computed on the CPU.

4 Results

4.1 Performance Measurement

For our experiments we used a computer with an Intel Core 2 6600 2,4 GHz CPU, equipped with 2 GB RAM and an NVIDIA GeForce 8800 GTX graphics card. The following table shows the performance (one iteration) of the most important steps of our algorithm on various grid sizes.

As can be seen in Table 1, one single iteration loop runs at interactive speed. Therefore, param-

Grid size	128^2	256^2	512^2
OF	18.3 ms	79 ms	342 ms
Defo	28.8 ms	116 ms	478 ms
Update	5.4 ms	17 ms	52 ms
Total	52.5 ms	212 ms	872 ms

Table 1: Timing statistics for one iteration of the proposed algorithm. In the first row, the time required by the optical flow computations (OF) are listed. Then, timings for the elastic deformation engine (Defo) and for the update of the vertices, forces and template image (Update) are shown. Due to the multigrid approach timings scale linearly with grid size.

eters like force field scaling, stiffness values and stiffness distribution can be changed interactively within the iteration loop. Our examples shown in the next section demonstrate that five to thirty iterative steps are required to achieve the final result. Note, that it is in general possible to use smaller grids for deformation and optical flow calculations. In all examples, the mesh resolution was set to 128^2.

4.2 Examples

On the last page we show three results that have been generated using our approach. Figure 5 shows a synthetic data set demonstrating varying stiffness distribution. Next, we created a semi-synthetic data set from a MR scan of a brain by manually shrinking the grey matter in the reference image (see Figure 6). Finally, we evaluate our method on a real data set shown in Figure 7.

Our results demonstrated the accuracy of realistic material deformation at interactive rates and the effective application to image registration problems. The registration process converges if the accumulated force field does not change anymore. This is the case if all vectors produced by the optical flow estimate are below a certain threshold, e.g. 0.25 pixel width. Then, the internal forces of the deformed material are in balance with the external forces of the predictor.

4.3 Validation

Schnabel et. al. have developed a general framework for validation of non-rigid registration algorithms [24]. Based on this framework, we have validated our approach. We generated various image pairs with a physically accurate finite element method and registered them by means of our algorithm. Figure 3 shows, that our algorithm achieves highly accurate results even for large deformations. The mean intensity difference between theses images is below 10^{-2}.

4.4 Discussion

In comparison to Modersitzki [21], our method distinguishes in several parts. First, we use a finite element discretization rather than a finite difference method. This allows a much better approximation of the partial derivatives and thus improves both stability and accuracy. Second, the external forces applied to the deformation engine are computed using an optical flow model rather than a gradient based approach. Therefore, the elastic deformation is driven more precisely to its final state, and thus a significant smaller number of iteration steps is required until convergence. This effect is illustrated in Figure 4. As a sidenote let us mention that both the optical flow method [8] and the finite element method [24] are used for validation of non-rigid registration techniques. By incorporating these techniques into our registration algorithm, high accuracy can be achieved.

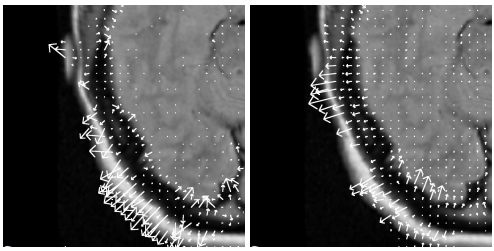


Figure 4: Comparison of image gradient (left) and optical flow (right) as deformation estimator. As can be seen, the optical flow yields generally much more accurate vector fields.

Another benefit of our system is that both the prediction and the correction stage are based on standard methods that can be used as black boxes. As a consequence, the predictor as well as the corrector can be easily adjusted to different methods that are either faster or specificly adapted to the underlying

image modalities. Especially, the deformation engine used has several advantages: It can efficiently handle different kinds of material parameters (heterogeneous, soft and stiff material) and strain formulations (linear Cauchy strain, corotated Cauchy strain). Due to its implicit nature, stability can be guaranteed during the whole convergence process.

Especially for large deformations one might wish to address the problem that the linear approximation of the strain tensor lacks rotational invariance, thus resulting in unrealistic deformations once elements get rotated out of their reference configuration. Fortunately, as has been shown in [13], co-rotated elements [10] could be easily integrated into the registration process by adapting the deformation engine to the co-rotated strain measurement.

Note, that the method is especially applicable to detect regions of the image where the optical flow model and the elastic model contradict each other. For example, this might be the case when registrating images before and after a bone fracture if the bone parts are not healed at exactly the same cut surface. In those cases, the registration process can be either manually controlled, or a-priori knowledge can be incorporated into the whole process. In this example, one would allow the bone to deform such that both images can match exactly.

Furthermore, due to this system design, our method can be easily adapted to 3D registration problems. Prediction of a 3D vector field can be achieved via 3D optical flow or similar methods and physical deformation engines based on volumetric elements are already fast [13].

5 Conclusion

In this work we have described a physics-based technique for image registration that enables flexible control over the kind of deformations to perform. In extending previous methods we have proposed a sophisticated predictor of an initial deformation field, and we have shown that highly accurate and stable finite-element methods can be integrated into interactive scenarios. In this way, image registrations can be achieved at high performance while material specific deformation properties as they arise in reality are simulated.

In the future, we will investigate more advanced optical flow algorithms, e.g. the one proposed by Bruhn et al. [5]. Furthermore, we will try to achieve faster convergence of our registration algorithm; one possibility is to optimize the way the optical flow field is applied as external forces to the finite element vertices. Additionally, we plan to improve the algorithm by integrating mixed boundary conditions (forces and displacements) into the dynamic elasticity problem efficiently. This allows us to enforce displacements at specific points, e.g. bones.

The extension of our registration algorithm to 3D is straightforward enabling accurate registrations of volumetric bodies. To deal with multi-modal registration problems, we can replace our deformation estimator by an appropriate method like mutual information. Note, that due to the design of our system the deformation engine is in no way affected by this change.

References

[1] Luis Alvarez, Joachim Weickert, and Javier Sanchez. Reliable estimation of dense optical flow fields with large displacements. *International Journal of Computer Vision*, 39(1):41–56, 2000.

[2] Klaus-Jurgen Bathe. *Finite Element Procedures*. Prentice Hall, 2002.

[3] Morten Bro-Nielsen and Stephane Cotin. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *Computer Graphics Forum*, 15(3):57–66, 1996.

[4] Andrés Bruhn, Joachim Weickert, Christian Feddern, Timo Kohlberger, and Christoph Schnörr. Variational optical flow computation in real-time. *IEEE Transactions on Image Processing*, 14:608–615, 2005.

[5] Andrés Bruhn, Joachim Weickert, Timo Kohlberger, and Christoph Schnörr. A multigrid platform for real-time motion computation with discontinuity-preserving variational methods. *Int. J. Comput. Vision*, 70(3):257–277, 2006.

[6] Gary E. Christensen, Richard D. Rabbitt, and Michael I. Miller. Deformable templates using large deformation kinematics. *IEEE Transactions on Image Processing*, 5(10):1435–1447, 1996.

[7] Ulrich Clarenz, Marc Droske, and Martin Rumpf. Towards fast non-rigid registration. *DFG 1114*, December 2002.

[8] James Cooper. Optical flow for validating medical image registration. In *IASTED International Conference on Signal and Image Processing*, pages 502–506, 2003.

[9] Valerie Duay, Pierre-François D'Haese, Rui Li, and Benoit M. Dawant. Non-rigid registration algorithm with spatially varying stiffness properties. In *ISBI*, pages 408–411, 2004.

[10] Olaf Etzmuß, Michael Keckeisen, and Wolfgang Straßer. A fast finite element solution for cloth modelling. In *Pacific Conference on Computer Graphics and Applications*, pages 244–251, 2003.

[11] Bernd Fischer and Jan Modersitzki. Curvature based image registration. *Journal of Mathematical Imaging and Vision (JMIV)*, 18(1):81–85, 2003.

[12] Joachim Georgii and Rüdiger Westermann. Interactive simulation and rendering of heterogeneous deformable bodies. In *Vision, Modeling and Visualization 2005*, pages 381–390, 2005.

[13] Joachim Georgii and Rüdiger Westermann. A multigrid framework for real-time simulation of deformable volumes. In *Proceedings of VRIPHYS Workshop*, pages 50–57, 2005.

[14] Leo Grady and Gareth Funka-Lea. Multi-label image segmentation for medical applications based on graph-theoretic electrical potentials. In *ECCV 2004 Workshops CVAMIA and MMBIA*, number LNCS3117 in Lecture Notes in Computer Science, pages 230–245. Springer, May 2004.

[15] Alexander Hagemann, Karl Rohr, H. Siegfried Stiehl, Uwe Spetzger, and Joachim M. Gilsbach. A biomechanical model of the human head for elasitc registration of MR-images. In *Bildverarbeitung fur die Medizin*, pages 44–48, 1999.

[16] Stefan Henn. A full curvature based algorithm for image registration. *Journal of Mathematical Imaging and Vision (JMIV)*, 24(2):195–208, 2006.

[17] Berthold Horn and Brian Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[18] Florian Jäger, Jingfeng Han, Joachim Hornegger, and Torsten Kuwert. A variational approach to spatially dependent non-rigid registration. In *Proceedings of the SPIE, Medical Image Processing*, volume 6144, pages 860–869, 2006.

[19] Sven Kabus, Astrid Franz, and Bernd Fischer. On elastic image registration with varying material parameters. In *Bildverarbeitung für die Medizin (BVM)*, pages 330–334. Springer Verlag, 2005.

[20] J.B.Antoine Maintz and Max A. Viergever. A survey of medical image registration. *Medical Image Analysis*, 2(1):1–36, 1998.

[21] Jan Modersitzki. *Numerical Methods for Image Registration*. Oxford university press, New York, 2004.

[22] Esther Radmoser, Otmar Scherzer, and Joachim Weickert. Scale-space properties of regularization methods. In *Scale-Space Theories in Computer Vision*, pages 211–222, 1999.

[23] Paul Ruhnau and Christoph Schnörr. Optical stokes flow estimation: an imaging-based control approach. *Experiments in Fluids*, 42(1):61–78, 2007.

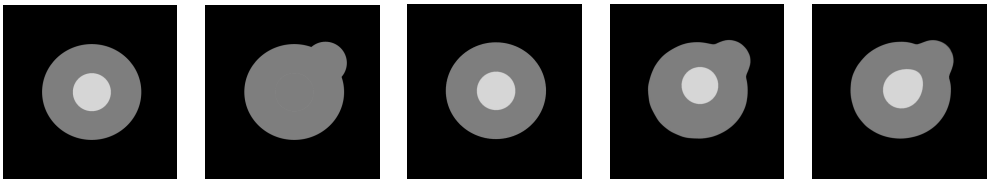
[24] J. A. Schnabel, C. Tanner, A. D. Castellano-Smith, A. Degenhard, M. O. Leach, D. R. Hose, D. L. G. Hill, and D. J. Hawkes. Validation of non-rigid image registration using finite element methods: Application to breast mr images. *IEEE Transactions on Medical Imaging*, 22:238–247, 2003.

[25] Marius Staring, Stefan Klein, and Josien P.W. Pluim. Nonrigid registration with adaptive, content-based filtering of the deformation field. In *Proceedings of SPIE Medical Imaging: Image Processing*, volume 5747, pages 212 – 221, February 2005.

[26] Yongmei Wang and Lawrence H. Staib. Integrated approaches to non rigid registration in medical images. In *Proceedings of WACV'98*, pages 102–108, 1998.

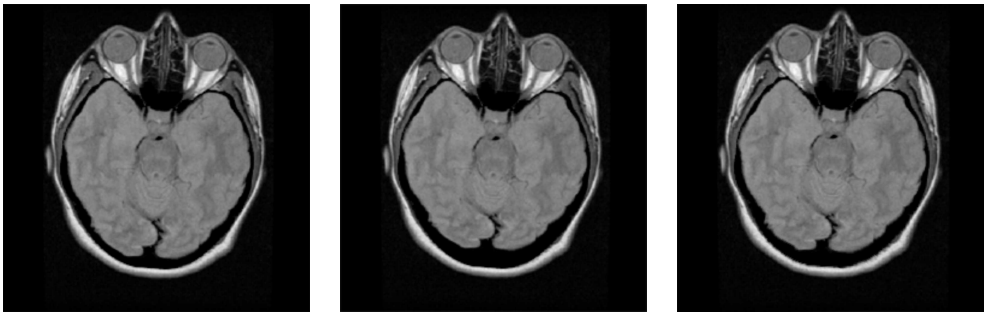
[27] Joachim Weickert and Christoph Schnörr. A theoretical framework for convex regularizers in pde-based computation of image motion. *International Journal of Computer Vision*, 45(3):245–264, 2001.

[28] Barbara Zitova and Jan Flusser. Image registration methods: a survey. *Image and Vision Computing*, 21(11):977–1000, October 2003.



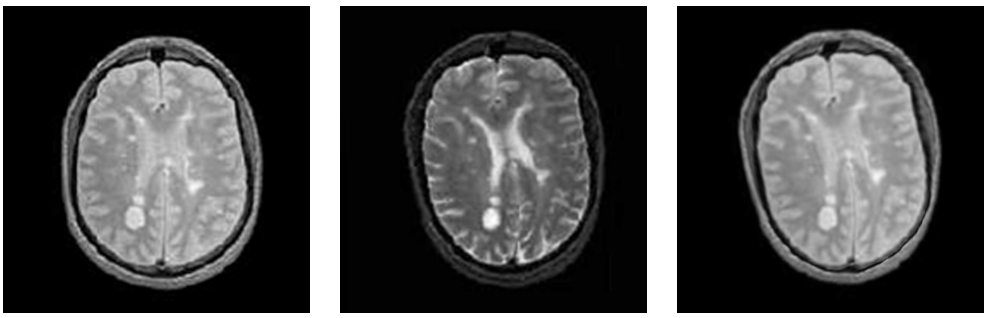
(a) Template (b) Reference (c) All hard (d) Mixed (e) All soft

Figure 5: This synthetic image simulates an object with two different tissue types. The template image (a) is registered to the reference image (b) using different tissue stiffness. Image (c) shows a uniformly hard stiffness distribution over the object so that no deformation is possible. Image (d) shows the result of assigning soft stiffness to the outer ring of the object. Now, the bump in the outer ring is matched while the hard white core remains undeformed. Image (e) shows results from a uniformly soft tissue throughout the object. Note, how the white core deforms.



(a) Template (b) Reference (c) Registered

Figure 6: This Figure shows a semi-synthetic data set of a manually shrunken brain. The template image (a) is registered to the reference image (b). Image (c) shows the registered result using a 256×256 grid. The stiffness was set to 10^8 for the skull, 10^6 for the grey matter and 10^4 for dark area in between. Note, how the brain compacts, the soft tissue expands and the bones remain stiff. The images register in 5 iterations (approximately 1 second) with an average per pixel error of 10^{-2}.



(a) Template (b) Reference (c) Registered

Figure 7: Note the different image contrasts of the template image (a) and the reference image (b). Our algorithm registers the two images accurately (c) in 0.5 seconds using a grid resolution of 128×128.

Nonlinear Diffusion vs. Wavelet Based Noise Reduction in CT Using Correlation Analysis

M. Mayer, A. Borsdorf, H. Köstler, J. Hornegger, and U. Rüde

Lehrstuhl für Mustererkennung & Lehrstuhl für Systemsimulation
Friedrich-Alexander Universität Erlangen-Nürnberg
Correspondence Email: harald.koestler@informatik.uni-erlangen.de

Abstract

We propose a new PDE based method for noise reduction in computed tomography (CT) using correlation analysis and compare it to a previously introduced wavelet based method. The arising nonlinear (an)isotropic PDEs are solved by an efficient multigrid solver.

Both approaches are based on the assumption that the data can be decomposed into information and temporally uncorrelated noise. In CT two spatially identical images can be generated by reconstructions from disjoint subsets of projection, e.g., by taking every other projection respectively. Our experimental results in 2D and 3D show that a noise reduction up to 66% can be achieved without noticable loss of image resolution. Additionally, a radiologist compared the visual quality of both methods with respect to noise and visibility of structures for real clinical data.

1 Introduction

Noise reduction in CT images gains more and more attention. It provides a possibility to increase the signal-to-noise ratio (SNR), hence giving more space for a further reduction of radiation dose.

Several methods for the reduction of noise in CT have been proposed [1, 2, 3], in most cases reducing noise in the projections before reconstruction. In contrast to this we focus on the reduction of noise in the reconstructed 2D slices or 3D volumes. Noise reduction in reconstructed CT datasets is not an easy task due to the difficult noise properties: after reconstruction the distribution of noise is unknown. Furthermore, noise is non-stationary and directed noise due to high attenuation along certain directions may be present.

In [4] we proposed a wavelet based denoising method for CT images. By separately reconstructing the odd and even numbered projections of a CT scan two sets of slices are obtained which include the same information but noise between the data is uncorrelated. By using correlation analysis in the wavelet domain combined with an orientation and position dependent noise estimation [5] only those wavelet coefficients containing image structure are kept for reconstruction of a noise suppressed image. In [6], the idea of this approach, e.g., using two data sets with uncorrelated noise, is picked up and transfered to the spatial domain, where we apply nonlinear isotropic diffusion filtering [7, 8]. Now, we extend it to the anisotropic case, compare it to the wavelets, and present results on clinical data.

An overview of the methodology is presented in Section 2. Section 3 summarizes the wavelet based approach. Then, we propose in Section 4 a new noise reduction method based on nonlinear (an)isotropic diffusion that tries to estimate the real image structure out of the correlations of two input datasets affected with uncorrelated noise. In Section 5 we compare the two methods with respect to noise reduction and edge preservation for phantom measurements in 2D and 3D. Furthermore, we discuss the visual results of real clinical data. Finally Section 6 concludes our work.

2 Method Overview

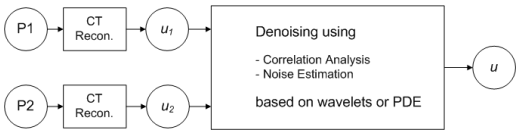


Figure 1: Overview of the noise reduction method

Figure 1 shows a brief overview of the denoising methodology: first, two CT datasets u_1 and u_2 are generated, which only differ with respect to noise. Note that if more than two sets are used, the SNR in the separately reconstructed images decreases what leads to a worse edge detection.

In CT, this can be achieved by separate reconstructions from disjoint subsets of projections $\mathrm{P1} \subset \mathrm{P}$ and $\mathrm{P2} \subset \mathrm{P}$, with $\mathrm{P1} \cap \mathrm{P2} = \emptyset$. Assuming that the sampling rate is high enough for both single sets of projections (see [9, 10]), then $R\{\mathrm{P}\} = 0.5\,(R\{\mathrm{P1}\} + R\{\mathrm{P2}\})$, with R defining an arbitrary reconstruction operator. Specifically, we reconstruct one dataset from the even and the other from the odd numbered projections using the weighted filtered back projection (WFBP) [11]. Instead of just averaging u_1 and u_2 we use these two datasets as input to our denoising algorithm, which can be a wavelet based or PDE based method.

Both methods have in common that they try to separate structure and noise by taking into account the local correlation of the two input datasets. Additionally, the local standard deviation of noise can be estimated from the difference between u_1 and u_2. After denoising, we obtain u, which corresponds to the reconstruction from the complete set of projections, but with improved signal-to-noise ratio.

3 Wavelet Based Denoising

The discrete, dyadic wavelet transformation (DWT) of a signal is a linear operation that maps the discrete d-dimensional input signal with N sample points onto the set of N wavelet coefficients: $a(\mathbf{x})$ defining the approximation and $w^D(\mathbf{x})$ the detail coefficients at position $\mathbf{x} = (x_1, ..., x_d)$ and orientation D. For the 2D case, e.g., we have $D \in \{LH, HL, HH\}$ altogether resulting in four blocks of coefficients: the lowpass filtered approximation and three detail images which include high frequency structures in the horizontal (LH), vertical (HL) and diagonal (HH) directions, respectively, together with noise in the corresponding frequency band.

Multidimensional signals are usually decomposed by applying a 1D transformation successively to all dimensions, whereas the 1D transformation can be described by a filter bank [12]: The signal is filtered with a high-pass filter $\tilde{g}$ and a corresponding lowpass filter $\tilde{h}$ followed by a dyadic downsampling step respectively. This decomposition can be repeated for the lowpass filtered approximation coefficients until the maximum decomposition level $l_{\max} \leq \log_2 N$ (assumed N is a power of two) is reached leading to a multiresolution decomposition. For perfect reconstruction of the signal, the dual filters g and h are applied to the coefficients at decomposition level l after upsampling. The two resulting parts are summed up leading to the approximation coefficients at level $l-1$.

The two separately reconstructed datasets u_1 and u_2 are both decomposed into multiple frequency bands and orientations by a discrete dyadic wavelet transformation. Because of the linearity of the wavelet transformation the average of u_1 and u_2 can directly be computed in the wavelet domain. The denoising is performed by applying adapted weights to the averaged high frequency detail coefficients. These weights depend on the local and frequency dependent similarity of the input datasets. Different methods for detecting correlated structures between the two datasets have been proposed in [4]. Here, we use the correlation between the approximation coefficients for the detection of structures. At each decomposition level l for each position $\mathbf{x}$, the empirical correlation coefficients between pixel regions taken from the approximation at level $l-1$ are computed. The pixel regions in the approximation are chosen within a local neighborhood $\Omega_{\mathbf{x}}$ around the corresponding position $\mathbf{x}^{l-1}$ of $\mathbf{x}^l$:

$$r^l(\mathbf{x}) = \frac{\mathrm{Cov}(a_1^{l-1}, a_2^{l-1})}{\sqrt{\mathrm{Var}(a_1^{l-1})\mathrm{Var}(a_2^{l-1})}} \tag{1}$$

with

$$\mathrm{Var}(s) = \sum_{\tilde{\mathbf{x}} \in \Omega_{\mathbf{x}}} \left(s(\tilde{\mathbf{x}}) - \bar{s}\right)^2 \tag{2}$$

and

$$\mathrm{Cov}(s_1, s_2) = \sum_{\tilde{\mathbf{x}} \in \Omega_{\mathbf{x}}} \left(s_1(\tilde{\mathbf{x}}) - \bar{s}_1\right)\left(s_2(\tilde{\mathbf{x}}) - \bar{s}_2\right), \tag{3}$$

where a_1^{l-1} and a_2^{l-1} define the approximation coefficients of u_1 and u_2, respectively, $\bar{a}_1^{l-1}$ and $\bar{a}_2^{l-1}$ denote the corresponding average values within $\Omega_{\mathbf{x}}$. The neighborhood $\Omega_{\mathbf{x}}$ is chosen in dependence on the used wavelet in order to assure that all pixels that influenced the detail coefficient at position $\mathbf{x}$ during decomposition and all those coefficients that are influenced by the reconstruction from this

coefficient are included into the correlation analysis. Therefore, the neighborhood is defined by the length of the used analysis ($\tilde{g}$, $\tilde{h}$) and synthesis (g and h) filters. If we assume that the length $\mathbf{m} = (m_1, \ldots, m_d)$ of all these four filters is the same and even, we define

$$\Omega_{\mathbf{x}} = \left\{ \tilde{\mathbf{x}}^{l-1} \;\middle|\; |\tilde{x}^{l-1} - x^{l-1}| \leq \mathbf{m} \right\}. \quad (4)$$

The final noise suppressed result u is computed by an inverse wavelet transformation from the averaged and weighted coefficients:

$$\begin{aligned} a^{l_{\max}}(\mathbf{x}) &= \frac{1}{2}\left(a_1^{l_{\max}}(\mathbf{x}) + a_2^{l_{\max}}(\mathbf{x})\right), \quad (5)\\ w^{D,l}(\mathbf{x}) &= \frac{1}{2}\left(w_1^{D,l}(\mathbf{x}) + w_2^{D,l}(\mathbf{x})\right) f^l(\mathbf{x}), \\ & \quad l \in \{1, \ldots, l_{\max}\}, \end{aligned}$$

with weighting function

$$f^l(\mathbf{x}) = \frac{1}{2}\left(r^l(\mathbf{x}) + 1\right). \quad (6)$$

4 PDE Based Denoising

4.1 PDE model

In contrast to the wavelet based denoising the PDE based denoising works in the spatial domain. Noise is removed using a nonlinear diffusion process described by the nonlinear PDE

$$u - u^0 = \tau \text{div}(g(\|\nabla u\|)\nabla u) \quad (7)$$

with Neumann boundary conditions. This is equivalent to solving a Perona and Malik nonlinear isotropic diffusion equation [7] for a fixed artificial time step τ. The initial image u^0 is set to the average of the two input images u_1^0 and u_2^0. The edge-stopping function $g(\|\nabla u\|)$ regulates the diffusion process. We use the Tukey edge-stopping function

$$g_\sigma(x) = \begin{cases} \left(1 - (\frac{x}{\sigma})^2\right)^2, & |x| \leq \sigma, \\ 0, & |x| > \sigma. \end{cases} \quad (8)$$

introduced in [13] because of its good edge preserving properties. Usually, the parameter σ is set to the standard deviation when dealing with white noise.

For denoising CT images we have to modify g_σ resulting in $\tilde{g}_V$ described next (cf. (15)) to achieve adequate results and we simultaneously denoise the two input CT images u_1^0 and u_2^0 and its average u^0. This means we have to solve the nonlinear system of partial differential equations

$$\begin{aligned} u - u^0 &= \tau \text{div}(D'_{u_1,u_2}\nabla u) \quad (9a)\\ u_1 - u_1^0 &= \tau \text{div}(D'_{u_1,u_2}\nabla u_1) \quad (9b)\\ u_2 - u_2^0 &= \tau \text{div}(D'_{u_1,u_2}\nabla u_2) \quad (9c) \end{aligned}$$

with Neumann boundary conditions. Here, D'_{u_1,u_2} is a nonlinear diffusion tensor with

$$D'_{u_1,u_2} = \tilde{g}_V \cdot E \quad (10)$$

in the isotropic case and

$$D'_{u_1,u_2} = \tilde{g}_V(D) \quad (11)$$

with $D = 0.25\left((\nabla u_1 + \nabla u_2)(\nabla u_1 + \nabla u_2)^T\right)$ in the anisotropic case. E denotes the identity matrix and $\tilde{g}_V$ is applied, e.g., in the 2D case, to D by applying $\tilde{g}_V$ to the eigenvalues λ_1, λ_2 of D and leaving the eigenvectors v_1, v_2 of D unchanged, i.e., $D'_{u_1,u_2} = \tilde{g}_V(\lambda_1)v_1v_1^T + \tilde{g}_V(\lambda_2)v_2v_2^T$.

Two ways of exploiting the availability of two input images with uncorrelated noise are to compute the correlation between both and to estimate noise variance.

Because of the spatially varying noise properties in CT images the analysis is done locally in a neighborhood $\Omega_{\mathbf{x}}$ around a pixel $\mathbf{x}$ analog to wavelet based denoising. Additionally, the neighboring pixels $\tilde{\mathbf{x}}$ are weighted with Gaussian weights

$$\alpha(\tilde{\mathbf{x}}, \mathbf{x}) = \frac{1}{\sigma\sqrt{2\pi}}\left(-\frac{1}{2}e^{\frac{\|\tilde{\mathbf{x}}-\mathbf{x}\|^2}{\sigma^2}}\right) \quad (12)$$

depending on the distance between pixel $\tilde{\mathbf{x}}$ and $\mathbf{x}$. A local estimate for the correlation of two image regions can be computed by the weighted correlation coefficient

$$c_\alpha(\mathbf{x}) = \frac{\text{Cov}_\alpha(u_1, u_2)}{\sqrt{\text{Var}_\alpha(u_1)\text{Var}_\alpha(u_2)}}.$$

using the weighted covariance

$$\text{Cov}_\alpha(s_1, s_2) = \sum_{\tilde{\mathbf{x}}\in\Omega_{\mathbf{x}}} \prod_{j=1}^{2} (s_j(\tilde{\mathbf{x}}) - \bar{s}_j)\alpha(\tilde{\mathbf{x}}, \mathbf{x})$$

and weighted variance

$$\text{Var}_\alpha(s) = \sum_{\tilde{\mathbf{x}}\in\Omega_{\mathbf{x}}} (s(\tilde{\mathbf{x}}) - \bar{s})^2\alpha(\tilde{\mathbf{x}}, \mathbf{x}),$$

where $\bar{s}$ is the local gray value average. Because in our case only the amount of similarity between image regions is interesting, the values below 0 of c_α, denoting anti-correlation, are set to 0, yielding a local similarity measure

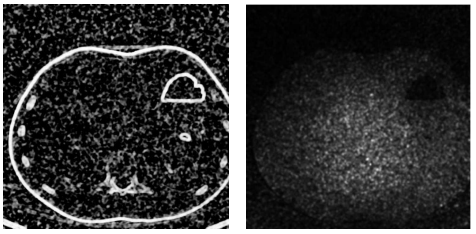
$$C_\alpha(\mathbf{x}) \quad = \quad \begin{cases} 1 & c_\alpha(\mathbf{x}) > 0 \\ 0 & c_\alpha(\mathbf{x}) \leq 0 \end{cases}. \qquad (13)$$

A visualization of C_α of the input images of a liver CT scan is shown in Fig. 2(a).

Two input images give us the possibility to estimate the local noise variance of the average of the input images by [14]

$$V(\mathbf{x}) = \frac{\sum_{\tilde{\mathbf{x}} \in \Omega_\mathbf{x}} \alpha(\tilde{\mathbf{x}}, \mathbf{x}) (u_1(\tilde{\mathbf{x}}) - u_2(\tilde{\mathbf{x}}))^2}{4 \sum_{\tilde{\mathbf{x}} \in \Omega_\mathbf{x}} \alpha(\tilde{\mathbf{x}}, \mathbf{x})}. \qquad (14)$$

A plot of the estimated local variance V in Fig. 2(b) shows the spatially changing behavior of the noise variance.



(a) Correlation (b) Variance

Figure 2: Plot of the local correlation C_α and local variance estimate V of a liver CT scan.

Based on the Tukey edge-stopping function we now design a new function taking into account V and C_α. The fixed parameter for the noise standard deviation of the Tukey edge-stopping function is replaced by $V(\mathbf{x})$ resulting in

$$\tilde{g}_V(x) = \begin{cases} \left(1 - \frac{x^2}{V(\mathbf{x})}\right)^2, & |x| \leq \beta\sqrt{V(\mathbf{x})}, \\ 0, & |x| > \beta\sqrt{V(\mathbf{x})}. \end{cases} \qquad (15)$$

with an additional fixed weighting factor $\beta \in \mathbb{R}^+$. If nothing else is stated, we set $\beta = 1$.

The square root of the product of the gradients on the input images

$$\|\nabla u_{1,2}\| = \sqrt{\|\nabla u_1\| \cdot \|\nabla u_2\|} \qquad (16)$$

is taken as input for the edge-stopping function $\tilde{g}_V$. It is further linearly scaled by the local similarity measure, i.e., we use $\tilde{g}_V(s), s \in \mathbb{R}$ with

$$s = \begin{cases} \|\nabla u_{1,2}\| \cdot W(\mathbf{x}) & \text{if } W(\mathbf{x}) > 0, \\ 0 & \text{else} \end{cases} \qquad (17)$$

and

$$W(\mathbf{x}) = 1 + \gamma(2C_\alpha(\mathbf{x}) - 1) \qquad (18)$$

with $\gamma \in \mathbb{R}^+$. This has the effect to damp high gradients in image regions with small similarity, e.g., in homogeneous regions, and to enlarge the gradient where similarity is high, i.e., when image structure is present.

4.2 PDE solver

We discretize Eq. (9) in 2D and 3D by finite volumes on a cell-based grid Ω^h, the gradients required for the computation of $\tilde{g}_V$ are approximated by finite differences. The resulting nonlinear system of equations

$$A^h(u^h) = f^h \qquad (19)$$

is solved by a cell-based nonlinear multigrid solver on a hierarchy of grids [15, 16, 17, 18] based on the full approximation scheme (FAS) [19] to obtain the denoised discrete image u^h. In the following explanations, we restrict ourselves to two grids for simplicity, a fine grid Ω^h and a coarse grid Ω^H.

To deal with the nonlinearity we apply an inexact lagged diffusivity [20, 21]. The idea is to keep the diffusivity function $\tilde{g}_V(s)$ constant during the iteration step $k+1$ and to evaluate it at the old iteration step k. That means we successively solve

$$u^{k+1} - u_0 = \tau \text{div}(\left(D'_{u_1,u_2}\right)^k \nabla u^{k+1}) \qquad (20a)$$

$$u_1^{k+1} - u_1^0 = \tau \text{div}(\left(D'_{u_1,u_2}\right)^k \nabla u_1^{k+1}) \qquad (20b)$$

$$u_2^{k+1} - u_2^0 = \tau \text{div}(\left(D'_{u_1,u_2}\right)^k \nabla u_2^{k+1}) \qquad (20c)$$

Note that now the three solution components u, u_1, and u_2 are decoupled and can be updated independently from each other.

Within the FAS multigrid iteration the Gauss-Seidel method denoted by the operator S^ν, where ν are the number of Gauss-Seidel iterations, serves as pre- and post-smoother. After each Gauss-Seidel iteration on each level, we update the values of the diffusivity function $\tilde{g}_V(s)$. Furthermore, we apply simple restriction and interpolation operators [16].

The restriction operator in 2D can be described by the stencil

$$I_h^H = \frac{1}{4}\begin{bmatrix} 1 & & 1 \\ & \cdot & \\ 1 & & 1 \end{bmatrix}, \tag{21}$$

where the dot in the stencil denotes the position on the coarse grid at which the restriction is applied, and the constant interpolation in 2D by the stencil

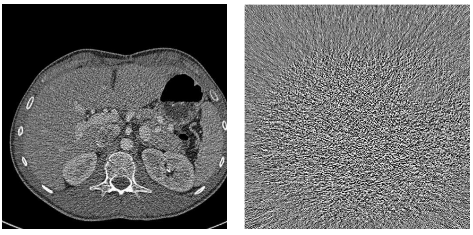
$$I_H^h = \begin{bmatrix} 1 & & 1 \\ & \cdot & \\ 1 & & 1 \end{bmatrix}. \tag{22}$$

Extension to 3D is straightforward. We denote one FAS iteration, e.g., by FAS(2,2), i.e., we perform 2 pre- and 2 post-smoothing steps. For the results in the next section we typically apply 3-4 FAS(2,2) iterations.

The FAS scheme can be extended to full multigrid (FMG) by constructing an image pyramid and starting to compute a denoised image at the lowest resolution or level. Then, the solution is interpolated to the next finer level and used as an initial guess there. On each level one or more FAS iterations are computed. One recursive FAS iterations to solve the nonlinear system $A^h u^h = f^h$ is shown in Algorithm 1. The initial guess $u_h^{(0)} = 0$.

5 Comparison Results

For a thin reconstructed example slice (0.8 mm) of an abdomen CT scan (see Fig.3), Fig. 4 shows the results for the proposed method in comparison to one standard nonlinear diffusion method and the wavelet based approach.



(a) Original (b) $(u_1 - u_2)/2$

Figure 3: Original average image and its difference images (one slice is of size 512×512), displayed with $c = 0$ and $w = 200$.

In 2D we used the redundant SWT, in 3D the non-redundant DWT, both in combination with the Haar wavelet and three decomposition levels. The noise reduced images are compared to the average of the input images, which corresponds to the result of a reconstruction from all projections and is in the following referred to as the original image. In Fig. 5 the corresponding difference images to the original image are shown, providing an impression of the denoising behavior of the different approaches. Fig. 4(a) clearly demonstrates that a standard nonlinear diffusion method fails to denoise a CT image with spatially varying noise power in an adequate manner. While noise in the center of the image is nearly unchanged, the outer regions are already blurred. Using the proposed PDE approach or the wavelet based method (Fig. 4(c)- 4(f)) shows that both methods are capable of adapting themselves to the local noise variance, thus removing noise more uniformly. For both approaches in 2D a noise reduction of about 45% is achieved throughout the image. To get a proper estimate of the correlation of the input images a 5×5 neighborhood was used in the wavelet approach and a 8×8 neighbourhood with gaussian weights of standard deviation 2 in case of the PDEs. Because image structures like edges have influence on the correlation value of distant pixels in their neighborhood, unfortunately noise remains around high contrast edges if the window for correlation analysis is chosen too large. However, if the window is chosen too small the correlation analysis gets unreliable. Using 3D data reduces this problem, because pixels for estimating the noise variance and correlation can be taken from the neighborhood in all three dimensions. Fig. 4(d) and 4(f) show the results in 3D using a window of $5 \times 5 \times 5$ pixels (with gaussian weights of standart derivation 1 in case of the PDEs). It can be seen clearly that a strong noise suppression of about 60% is achieved while image structures are well preserved.

These images were also used for clinical tests, where a radiologist judged the images with respect to noise and visibility of structures in two consecutive tests. Within these tests unlabeled image pairs were shown to the radiologist in randomized order. For all image pairs, the radiologist decided if there was one preferred image with respect to the current evaluation criterion. Switching the position of the two images, allowed to notice even small dif-

Algorithm 1 FAS iteration (V-cycle): Compute $u_h^{(k+1)} = M_h^{FAS}\left(u_h^{(k)}, A^h, f^h, \nu_1, \nu_2\right)$

1: $\bar{u}_h^{(k)} = S_h^{\nu_1}\left(u_h^{(k)}, A^h, f^h\right)$ {pre-smoothing}
2: $r^h = f^h - A^h\left(\bar{u}_h^{(k)}\right)$ {compute residual}
3: $r^H = \mathcal{I}_h^H r^h$ {restrict residual}
4: $\bar{u}^H = \mathcal{I}_h^H \bar{u}_h^{(k)}$ {restrict solution}
5: $f^H = r^H + A^H\left(\bar{u}^H\right)$
6: **if** number of coarse grid points $< \epsilon_{min}$ **then**
7: Solve nonlinear problem $A^H(w^H) = f^H$ by a suitable nonlinear solver or sufficiently many Gauss-Seidel iterations
8: **else**
9: $w^H = M_H^{FAS}\left(\bar{u}^H, A^H, f^H, \nu_1, \nu_2\right)$
10: **end if**
11: $e^H = w^H - \bar{u}^H$
12: $e^h = \mathcal{I}_H^h e^H$ {interpolate error}
13: $\tilde{u}_h^{(k)} = \bar{u}_h^{(k)} + e^h$ {coarse grid correction}
14: $u_h^{(k+1)} = S_h^{\nu_2}\left(\tilde{u}_h^{(k)}, A^h, f^h\right)$ {post-smoothing}

ferences between the images. Within the tests all denoised images were compared to the original image. Furthermore, the 2D and 3D results of the proposed PDE approach were compared to the results of the wavelet based approach. Summarizing, the test showed that all denoised images were judged superior in comparison to the original for both evaluation criteria.

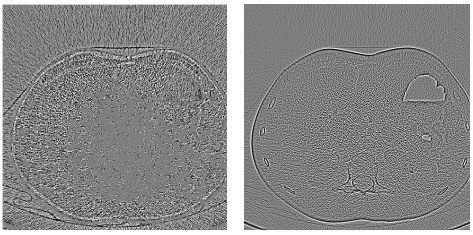
For the comparison between PDE and wavelets the results of the tests were not that clear. With respect to noise the wavelet based methods were preferred. Regarding the visibility of structures the PDE approach gives better results.

In addition to the visual inspection, quantitative tests were performed evaluating noise reduction and edge preservation in phantom images. For generating the simulated CT-scans the *DRASIM* software package provided by Karl Stierstorfer [22] was used. The phantom consists of a water cylinder with an inlaid quartered cylinder of defined density. In order to test the preservation of edges at different contrast-to-noise levels the density of the object was varied leading to edge-contrasts between 20 to 100 Hounsfield units (HU). The Hounsfield scale is a quantitative measure for radiodensity, i.e., it describes the relative transparency of a material, if X-rays pass through it.

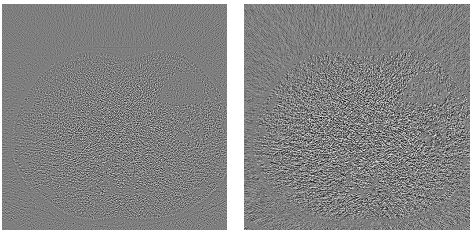
In addition to noisy phantoms, ideal CT-scans were simulated leading to noise-free ground-truth data. In CT a standard measurement for resolution is the modulation transfer function (MTF) (see e.g. [10, 23]), indicating how many line pairs per cm (lp/cm) can be distinguished. It is possible to determine the local MTF directly from the edge in an image. For this purpose, we manually selected a fixed region of 20×125 pixels around an edge (with a slope of approx. 4 degrees). The slight tilt of the edge allows a higher sampling of the edge profile, which is additionally averaged along the edge. The derivation of the edge profile leads to the line-spread function (LSF). The Fourier transformation of the LSF results in the MTF, which is additionally normalized so that $\mathrm{MTF}(0) = 1$. Reliable measurements of the MTF from this *edge technique* can only be achieved if the contrast of the edge is much higher than the pixel noise in the images [24].

In case of the wavelet based approach this can be easily circumvented by applying the computed weights at each decomposition level to the wavelet coefficients of the ideal noise-free image and computing the inverse transformation. This has the effect of making the influence of the weighting to the real signal directly visible. The MTF can then be computed at the edge in the processed noise-free image.

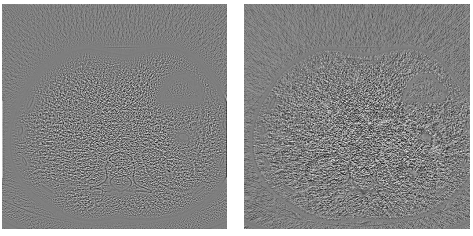
In case of the diffusion method this is not possible and, therefore, the average of 200 denoised slices was used for computing the MTF. The MTFs



(a) Tukey - Original (b) 2D Anisotropic - Original



(c) 2D Isotropic - Original (d) 3D Isotropic - Original



(e) 2D Wavelet - Original (f) 3D Wavelet - Original

Figure 5: Difference images, displayed with $c = 0$ and $w = 200$.

measured at edges with different contrasts are plotted in Fig. 6 and compared to the MTF measured at the original edge of 100 HU. All approaches used a 5×5 neighborhood, the PDE with Gaussian weights of standard derivation 1.

It can be clearly seen that the preservation of an edge very much depends on the contrast-to-noise level. The lower the contrast at the edge the stronger the MTF falls below the original curve, indicating that the edge was blurred. Fig. 6(a) shows that the phantom edges with a contrast of more than 40 HU are nearly fully preserved by the 2D PDE method. Additionally, due to the large time parameter needed to obtain a noise reduction of 45% the edge is enhanced by the diffusion process. The 3D PDE approach leads to better results for low-contrasty edges and the amount of sharpening applied is lower, as it can be seen in Fig. 6(c). Suprisingly, if anisotropic diffusion is used the edges are blurred as Fig. 6(b) shows. It seems that this approach is not capable of denoising images with fine structures, because even minor errors in the estimation of the edge gradient lead to a blurring over it.

For comparing the PDE approach to the wavelet based method the ρ_{50}-values of the MTFs were plotted against the contrast of the edge in Fig. 7. The ρ_{50}-value is defined as $\mathrm{MTF}(\rho_{50}) = 0.5$. Additionally, the values of the original noise-free images are plotted for comparison. It can be seen that the edge preservation of the PDE method outperforms the wavelet based method.

6 Conclusions

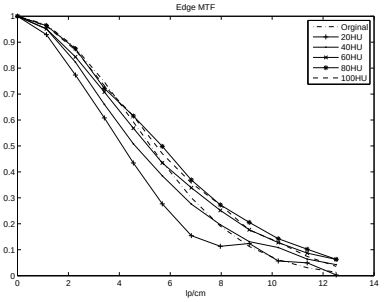
Isotropic and Anisotropic diffusion is adapted to be able to deal with the special noise characteristics of CT data. The diffusion depends on local noise variations and an estimation of the real image structure by calculating correlations between two input images with uncorrelated noise. The approach is compared to a similar wavelet based denoising method.

To enable the use of the presented algorithms in practical applications, it is necessary to improve their performance. Currently it takes for both approaches on a Laptop (Pentium M 2.0 GHz and 1 GB RAM) and typical parameter settings about 3–7 seconds to denoise a 2D slice of size 512×512 and about 80–120 seconds for a 3D volume of size $512 \times 512 \times 16$. Although the wavelet method is implemented within Matlab, it is slightly faster. The PDE based approach is implemented in C++. Applying standard optimization techniques [25, 26, 27] to the unoptimized multigrid solver could lead to a performance gain of factor 2–5.

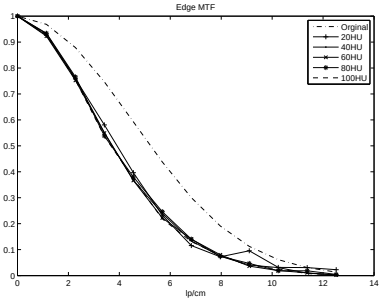
References

[1] J. Hsieh, "Adaptive streak artifact reduction in computed tomography resulting from excessive x-ray photon noise," *Medical Physics*, vol. 25, no. 11, pp. 2139–2147, November 1998.

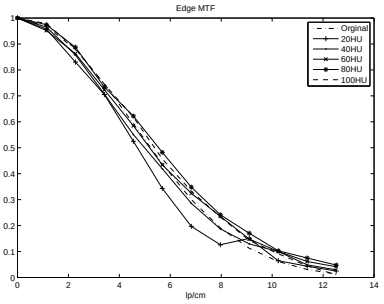
[2] M. Kachelrieß, O. Watzke, and W. A. Kalender, "Generalized multi-dimensional adaptive filtering for conventional and spiral single-slice, multi-slice, and cone-beam CT," *Medical Physics*, vol. 28, no. 4, pp. 475–490, 2001.



(a) 2D Isotropic

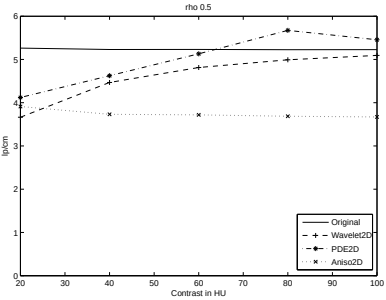


(b) 2D Anisotropic

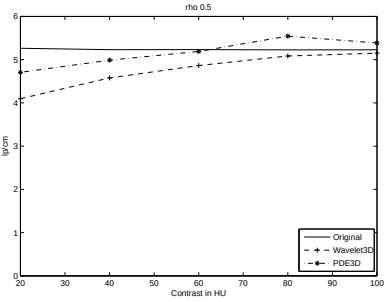


(c) 3D PDE

Figure 6: MTFs achieved with the PDE approach in 2D and 3D for different contrasts at the edge.



(a) 2D PDE vs. Wavelet



(b) 3D PDE vs. Wavelet

Figure 7: Comparison of ρ_{50}-values achieved for PDE or wavelet based method plotted against the contrast of the edge.

[3] O. Demirkaya, "Reduction of noise and image artifacts in computed tomography by nonlinear filtration of projection images," in *Proc. SPIE, Medical Imaging 2001: Image Processing*, M. Sonka and K. M. Hanson, Eds., vol. 4322, 2001, pp. 917–923.

[4] A. Borsdorf, R. Raupach, and J. Hornegger, "Wavelet based Noise Reduction by Identification of Correlation," in *Pattern Recognition (DAGM 2006), Lecture Notes in Computer Science*, K. Franke, K. Müller, B. Nickolay, and R. Schäfer, Eds., vol. 4174. Berlin: Springer, 2006, pp. 21–30.

[5] A. Borsdorf, R. Raupach, and J. Hornegger, "Separate CT-Reconstruction for Orientation and Position Adaptive Wavelet Denoising," in *Bildverarbeitung für die Medizin 2007*, A. Horsch, T. Deserno, H. Handels, H. Meinzer, and T. Tolxdoff, Eds. Berlin: Springer, 2007, pp. 232–236.

[6] M. Mayer, A. Borsdorf, H. Köstler, J. Hornegger, and U. Rüde, "Nonlinear Diffusion Noise Reduction in CT Using Correlation Analysis," in *3rd Russian-Bavarian Conference on Biomedical Engineering*, J. Hornegger, E. Mayr, S. Schookin, H. Feußner, N. Navab, Y. Gulyaev, K. Höller, and V. Ganzha, Eds., vol. 1. Erlangen, Germany: Union aktuell,

2007, pp. 155–159.

[7] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 7, pp. 629–639, 1990.

[8] J. Weickert, *Anisotropic Diffusion in Image Processing*. Teubner Verlag, Stuttgart, Germany, 1998.

[9] A. Kak and M. Slanely, *Principles of Computerized Tomographic Imaging*. Society for Industrial and Applied Mathematics, Juli 2001, http://www.slaney.org/pct/pct-toc.html.

[10] T. Buzug, *Einführung in die Computertomographie*. Berlin Heidelberg: Springer-Verlag, 2004.

[11] K. Stierstorfer, A. Rauscher, J. Boese, H. Bruder, S. Schaller, and T. Flohr, "Weighted FBP - a simple approximate 3DFBP algorithm for multislice spiral CT with good dose usage for arbitrary pitch," *Physics in Medicine and Biology*, vol. 49, no. 11, pp. 2209–2218, 2004.

[12] G. Strang and T. Nguyen, *Wavelets and Filter Banks*. Wellesley- Cambridge Press, 1996.

[13] M. J. Black, G. Sapiro, D. Marimont, and D. Heeger, "Robust anisotropic diffusion," *IEEE Trans. on Image Processing*, vol. 7, no. 3, pp. 421–432, 1998.

[14] A. Borsdorf, R. Raupach, and J. Hornegger, "Separate CT-Reconstruction for Orientation and Position Adaptive Wavelet Denoising," in *Bildverarbeitung für die Medizin 2007*, A. Horsch, T. Deserno, H. Handels, H. Meinzer, and T. Tolxdoff, Eds. Springer-Verlag, Berlin, Heidelberg, New York, 2007, pp. 232–236.

[15] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*. Academic Press, San Diego, CA, USA, 2001.

[16] P. Wesseling, *Multigrid Methods*. Edwards, Philadelphia, PA, USA, 2004.

[17] E. Kalmoun, H. Köstler and U. Rüde, "3D optical flow computation using a parallel variational multigrid scheme with application to cardiac C-arm CT motion," *Image and Vision Computing*, vol. 25, no. 9, pp. 1482–1494, 2007.

[18] A. Bruhn, "Variational optic flow computation: Accurate modeling and efficient numerics," Ph.D. dissertation, Department of Mathematics, Saarland University, Saarbrücken, Germany, 2006.

[19] A. Brandt, "Multi-Level Adaptive Solutions to Boundary-Value Problems," *Mathematics of Computation*, vol. 31, no. 138, pp. 333–390, 1977.

[20] T. Chan and P. Mulet, "On the convergence of the lagged diffusivity fixed point method in total variation image restoration," *SIAM Journal on Numerical Analysis*, vol. 36, no. 2, pp. 354–367, 1999.

[21] C. Vogel, *Computational Methods for Inverse Problems*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA, 2002.

[22] K. Stierstorfer, T. Flohr, and H. Bruder, "Segmented multiple plane reconstruction: A novel approximate reconstruction scheme for multi-slice spiral CT," *Physics in Medicine and Biology*, vol. 47, no. 4, pp. 2571–2581, July 2002.

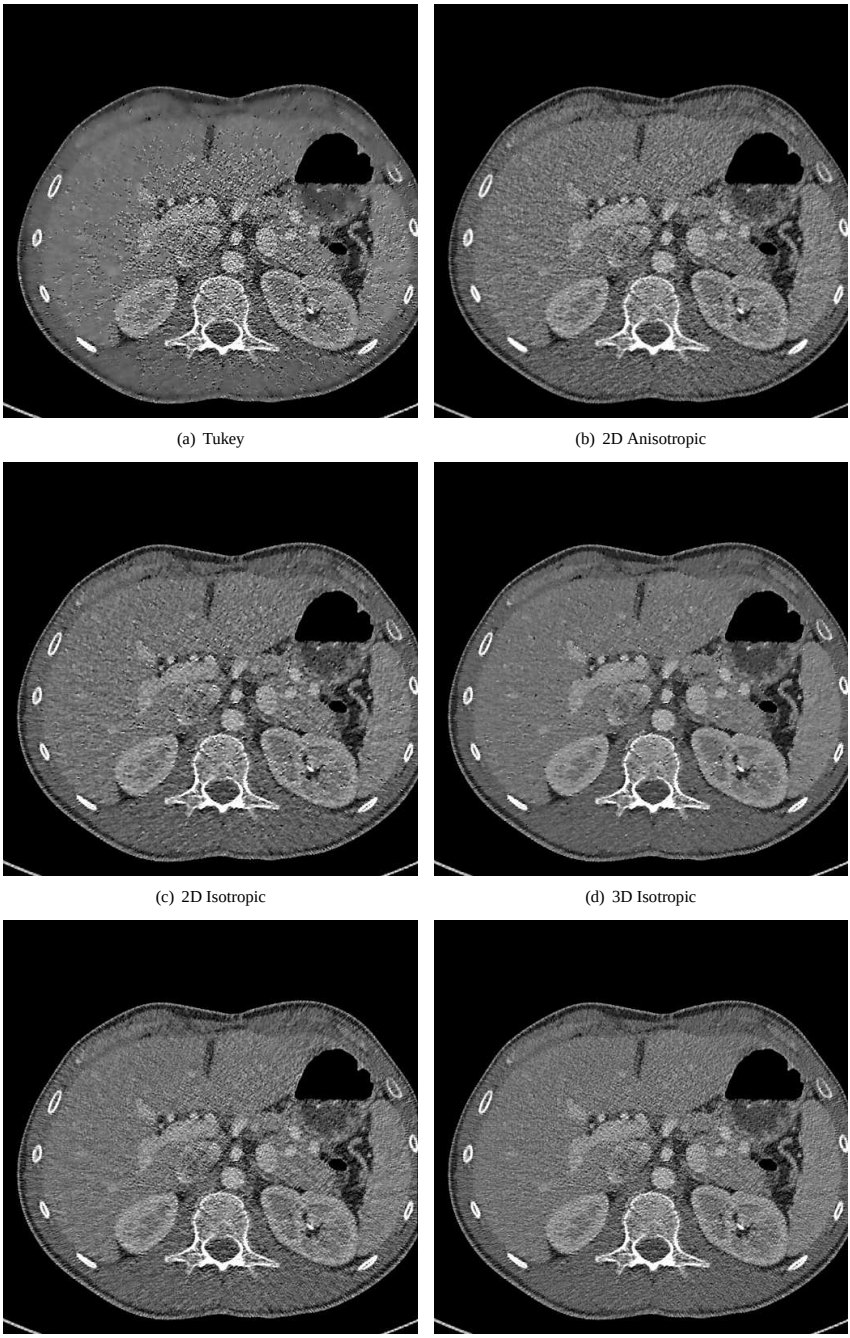
[23] O. Dössel, *Bildgebende Verfahren in der Medizin: Von der Technik zur medizinischen Anwendung*. Springer-Verlag, Berlin, Heidelberg, New York, 1999.

[24] I. A. Cunningham and B. K. Reid, "Signal and noise in modulation transfer function determinations using the slit, wire, and edge techniques," *Medical Physics*, vol. 19, no. 4, pp. 1037–1044, July 1992.

[25] C. Douglas, J. Hu, M. Kowarschik, U. Rüde, and C. Weiß, "Cache Optimization for Structured and Unstructured Grid Multigrid," *Electronic Transactions on Numerical Analysis (ETNA)*, vol. 10, pp. 21–40, 2000.

[26] H. Köstler, M. Stürmer, and U. Rüde, "A fast full multigrid solver for applications in image processing," Department of Computer Science 10 (System Simulation), Friedrich-Alexander-University of Erlangen-Nuremberg, Germany, Tech. Rep. 07-6, 2007, submitted to Numerical Linear Algebra with Applications.

[27] C. Weiß, "Data Locality Optimizations for Multigrid Methods on Structured Grids," Ph.D. dissertation, Lehrstuhl für Rechnertechnik und Rechnerorganisation, Institut für Informatik, Technische Universität München, Germany, 2001.



(a) Tukey (b) 2D Anisotropic

(c) 2D Isotropic (d) 3D Isotropic

(e) 2D Wavelet (f) 3D Wavelet

Figure 4: Denoising results for a CTA of a liver, displayed with $c = 200$ and $w = 700$.

Estimating Natural Activity by Fitting 3D Models via Learned Objective Functions

Matthias Wimmer*, Christoph Mayer, Freek Stulp and Bernd Radig

Faculty of Science and Engineering, Waseda University, Tokyo 169-8555, Japan
Institut für Informatik, Technische Universität München, 85748 Garching, Germany
Kognitive Neuroinformatik, Universität Bremen, 28359 Bremen, Germany
Email: {wimmerm,mayerc,stulp,radig}@cs.tum.edu

Abstract

Model-based image interpretation has proven to robustly extract high-level scene descriptors from raw image data. Furthermore, geometric texture models represent a fundamental component for visualizing real-world scenarios. However, the motion of the model and the real-world object must be similar in order to portray natural activity. Again, this information can be determined by inspecting images via model-based image interpretation.

This paper sketches the challenge of fitting models to images, describes the shortcomings of current approaches and proposes a technique based on machine learning techniques. We identify the objective function as a crucial component for fitting models to images. Furthermore, we state preferable properties of these functions and we propose to learn such a function from manually annotated example images.

1 Introduction

Model-based image interpretation is appropriate to extract high-level information from single images and from image sequences. Models induce a priori knowledge about the object of interest and reduce the large amount of image data to a small number of model parameters. The model parameters $\boldsymbol{p}$ represent its configuration, including position, rotation, scaling, and deformation. These parameters are usually mapped to the surface of an image, via a set of feature points, a contour, or a textured region.

Model fitting is the computational challenge of finding the model parameters that describe the content of a given image best [6]. This task consists of two components: the fitting algorithm and the objective function. The *objective function* yields a comparable value that determines how accurately a parameterized model fits to an image. Without losing generality, we consider lower values to represent a better model fitness. Depending on context, they are also known as the likelihood, similarity, energy, cost, goodness or quality functions. The *fitting algorithm* searches for the model parameters $\boldsymbol{p}$ that optimize the objective function, i.e. it searches for the global minimum or maximum, depending on the definition of the objective function. Since the described methods are independent of the used fitting algorithm, this paper shall not elaborate on them but we refer to [6] for a recent overview and categorization.

For interpreting the content of image sequences, model tracking fits the model to the individual images of the sequence. Each step benefits from the pose estimate derived from the previous image within the sequence. However, determining the pose estimate for the first image of the sequence has not been sufficiently solved yet. This so-called initial pose estimation is identical to fitting models to single images.

This paper evaluates our approach considering a rigid 3D model of a human face. Since faces highly vary in shape and texture in contrast to artificial objects, such as cars, fitting face models to images represents a particular difficulty.

Many researchers engage in fitting 3D models. Lepetit *et al.* [8] treat this issue as a classification problem. They integrate decision trees to solve this challenge.

Blanz *et al.* use a morphable three-dimensional model of human faces that describes not only the pose but also the appearance of the face [2]. Drummond and Cipolla [13] track rigid object by per-

*This research is partly funded by a JSPS Postdoctoral Fellowship for North American and European Researchers (FY2007).

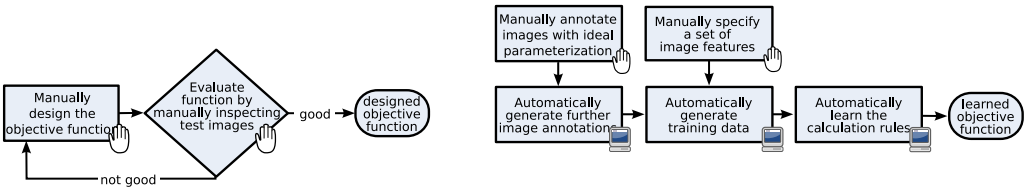


Figure 1: The procedures for designing (left) and learning (right) objective functions.

forming a fitting step on every image. The model pose is changed that way that the distances between edges of the projected model and edges in the image are minimized. Plagemann *et al.* [3] use different views of a model that are generated off-line to estimate the pose of the object during the on-line application.

Problem Statement. Although the accuracy of model fitting heavily depends on the objective function, it is often designed by hand using the designer's intuition about a reasonable measure of fitness. Afterwards, its appropriateness is subjectively determined by inspecting the objective function on example images and example model parameters. If the result is not satisfactory the function is tuned or redesigned from scratch [11, 5], see Figure 1 (left). Therefore, building the objective function manually is very time consuming and the function does not guarantee to yield accurate results. Furthermore domain-dependent knowledge is needed in this approach. First, knowledge of the modeled object and the application is needed to select the image features and afterwards mathematical knowledge is needed to obtain the objective function. Therefore, in summary, the traditional approach of designing objective functions manually shows a bad temporal behavior, requires a specialist to perform the work and holds no guarantee concerning the quality of the resulting objective function.

Solution Outline. Our novel approach focuses on the root problem of model fitting: We improve the objective function rather than the fitting algorithm. As a solution to this challenge, we propose to conduct a five-step methodology that learns robust local objective functions from annotated example images. The obtained functions consider specific issues of 3D models, such as out-of-plane rotations and self-occlusion. Here, we compute the features not in the 2D image plane but in the space of the 3D model. Therefore, we connect the individual features directly to the model. No domain-dependent knowledge is needed in this approach and the time requirement is easily determinable.

Contributions. The resulting objective functions work very accurately in real-world scenarios and they are able to solve the challenge of initial pose estimation that is required by model tracking. This easy-to-use approach is applicable to various image interpretation scenarios and requires the designer just to annotate example images with the correct model parameters. Since no further computer vision expertise is necessary, this approach has great potential for commercialization.

The paper proceeds as follows. In Section 2, we sketch the challenge of model-based image interpretation. In Section 3, we propose our methodology to learn accurate local objective functions from annotated training images with particular focus on 3D models. Section 4 conducts experimental evaluations that verify this approach. Section 5 summarizes our approach and shows future work.

2 Model-based Image Interpretation

A rigid 3D model represents the geometric properties of a real-world object. A six-dimensional parameter vector $\boldsymbol{p}=(t_x, t_y, t_z, \alpha, \beta, \gamma)^T$ describes its position and orientation. The model consists of $1\leq n\leq N$ three-dimensional model points specified by $\boldsymbol{c}_n(\boldsymbol{p})$ which are mapped on the image plane using perspective projection for feature extraction or visualization. Figure 2 depicts our face model with $N=214$ model points.

Fitting 3D models to images requires two essential components: The *fitting algorithm* searches for the model parameters that match the content of the image best. For this task, it searches for the minimum of the *objective function* $f(I, \boldsymbol{p})$, which determines how well a model $\boldsymbol{p}$ matches an image I. As in Equation 1, this function is often subdivided into

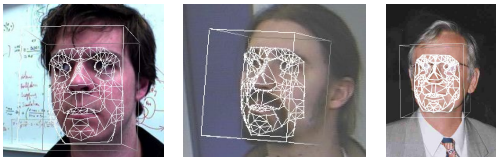


Figure 2: Our 3D model of a human face correctly fitted to images.

N local components $f_n(I, \boldsymbol{x})$, one for each model point [8, 1, 9].

$$f(I, \boldsymbol{p}) = \sum_{n=1}^{N} f_n(I, \boldsymbol{c}_n(\boldsymbol{p})) \qquad (1)$$

These local functions determine how well the n^{th} model point at a three-dimensional position x fits to the content of the image I. The advantage of this partitioning is that designing the local functions is more straightforward than designing the global function, because only the image content in the vicinity of one perceptively projected model point needs to be taken into consideration. The disadvantage is that dependencies and interactions between local errors cannot be combined.

2.1 Ideal Objective Functions

This section explicitly formulates two properties that local objective functions should have in the best case. First, they should have a global minimum that corresponds to the best model fit with model parameters $\boldsymbol{p}_I^\star$. Otherwise, we cannot be certain that determining the minimum of the local objective function yields the intended result. Second, local objective functions should have no other local minima. This implies that any minimum found corresponds to the global minimum, which facilitates search.

P1: Correctness property: The global minimum corresponds to the best model fit.

$$\forall \boldsymbol{x}(\boldsymbol{c}_n(\boldsymbol{p}_I^\star) \neq \boldsymbol{x}) \quad \Rightarrow \quad f_n(I, \boldsymbol{c}_n(\boldsymbol{p}_I^\star)) < f_n(I, \boldsymbol{x})$$

P2: Uni-modality property: The objective function has no local extrema.

$$\exists \boldsymbol{m} \forall \boldsymbol{x}\, (\boldsymbol{m} \neq \boldsymbol{x}) \quad \Rightarrow \quad f_n(I, \boldsymbol{m}) < f_n(I, \boldsymbol{x}) \quad \wedge \quad \nabla f_n(I, \boldsymbol{x}) \neq \boldsymbol{0}$$

Property P1 relates to the *correctness* of the objective function. Fitting algorithms search for the global minimum of the objective function and P1 ensures that the result of a successful search corresponds to the best fit of the model. Property P2 guarantees that any minimum that is found is the global minimum. This facilitates search, as fitting algorithms can not get stuck in a local minimum. Local optimization strategies, which are easier to design than global ones, then suffice to find the global minimum. Note that the global minimum $\boldsymbol{m}$ does not need to correspond with the best fit; this is only required by the independent property P1. We call functions that have both properties *ideal*. A concrete example of an ideal local objective function is shown in Equation 2. As described in the next section, our approach approximates an ideal function by learning it from examples created by an ideal objective function.

$$f_n^\star(I, \boldsymbol{x}) = |\boldsymbol{x} - \boldsymbol{c}_n(\boldsymbol{p}_I^\star)| \qquad (2)$$

2.2 Characteristic Directions of Local Objective Functions

Fitting 2D contour models to images usually searches along the perpendicular to the contour for the minimum of the local objective function [4]. Our approach sticks to this procedure, and therefore we create several local objective functions that are specific to these characteristic directions.

In the case of 3D-models, we connect these three-dimensional characteristic directions very tightly to the structure of the model. Transforming the pose of

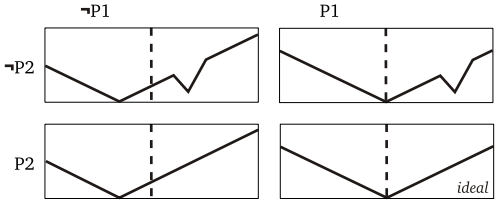


Figure 3: These four graphs display typical example functions that do or do not have properties P1 and P2, which influence the behaviour of an objective function. The dashed line indicates the preferred position of the contour point. If both P1 and P2 hold, the objective function is considered to be ideal.

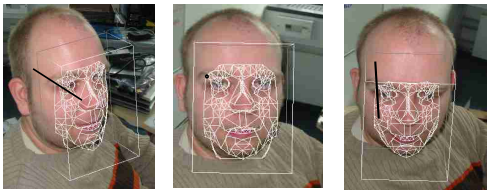


Figure 4: Due to transformation of the face model the characteristic direction will not remain parallel to the image plane, most often.

the model will transform these directions as well. Note, that in the case of 2D contour models these characteristic directions are defined in image space whereas they are defined in three-dimensional space in the context of three-dimensional models. Again, search for the minimum of the local objective function is conducted along these characteristic directions. As a novelty the input features positions are also represented three-dimensionally, but their value is calculated by projecting them to the image.

An image is most descriptive for a characteristic direction if the characteristic direction is parallel to the image plane. Unfortunately, transforming the model will usually yield characteristic directions that are not parallel to the image plane. Therefore, we propose to consider not only one but $1 \leq l \leq L$ characteristic directions that are differently oriented. This yields a specific objective function $f_{n,l}(I, \boldsymbol{x})$ for each characteristic direction l and each model point n. For every point we propose to consider only the $f_{n,l}$ whose characteristic direction is most parallel to the image. The local objective function f_n is computed as in Equation 3. The indicator $g_n(\boldsymbol{p})$ computes the index of the characteristic direction that is most significant for the current pose $\boldsymbol{p}$ of the model and a model point n. Even if these directions are arbitrary, we define them to be pair wise orthogonal.

$$f_n(I, \boldsymbol{x}) = f_{n,g_n(p)}(I, \boldsymbol{x}) \tag{3}$$

3 Learning Objective Functions from Image Annotations

Unfortunately, the ideal objective function $f_n^\star$ cannot be applied to previously unseen images, for which the best model parameters $\boldsymbol{p}_I^\star$ are not known. Nevertheless, we apply this ideal objective function to annotated training images and obtain ideal training data for learning a further local objective function $f_{n,l}^\ell$. The key idea behind our approach is that since the training data is generated by an ideal local objective function, the function learned from this data will also be approximately ideal. The subsequent sections will explain the five steps of the proposed procedure of learning local objective functions, see also Figure 1 (right).

3.1 Step 1: Annotating Example Images with the Preferred Model Parameters

We gather a database of $1 \leq k \leq K$ images I_k and each image is manually annotated with the preferred model parameters $\boldsymbol{p}_{I_k}^\star$. These parameters are necessary to compute the ideal local objective functions $f_n^\star$. This annotation is the only laborious step in the entire procedure. It takes less then 30 seconds for an experienced person to annotate one image. Figure 2 illustrates three example images for which these preferred model parameters have been specified.

3.2 Step 2: Generating Further Image Annotations

Because $\boldsymbol{x}$ is set to $\boldsymbol{c}_n(\boldsymbol{p}_I^\star)$, the ideal objective function returns the minimum $f_n^\star(I, \boldsymbol{x}) = 0$ for all manual image annotations $\boldsymbol{x} = \boldsymbol{c}_n(\boldsymbol{p}_{I_k}^\star)$. Therefore, these annotations are not sufficient to learn the characteristics of $f_n^\star$ and we will acquire image annotations $\boldsymbol{x} \neq \boldsymbol{c}_n(\boldsymbol{p}_{I_k}^\star)$, for which $f_n^\star(I, \boldsymbol{x}) \neq 0$. In general, any position in space may represent one of these annotations. However, it is more practicable to restrict this displacement in terms of distance and direction, see Figure 5. Therefore, we only take positions along the most important characteristic direction. The maximum displacement Δ is termed *learning radius*

In this paper, we use $L = 3$ characteristic directions, because the model points of our model are defined within the Euclidean 3D space. The characteristic direction with the largest angle to the normal of the image plane is chosen to be the most important one and thus selected by $g_n(\boldsymbol{p})$. That direction is easily estimated, because it is also the longest when projected in the image plane. Taking those relocations facilitates the later learning step and improves the accuracy of the resulting calculation rules. The first characteristic direction is chosen to be tangent

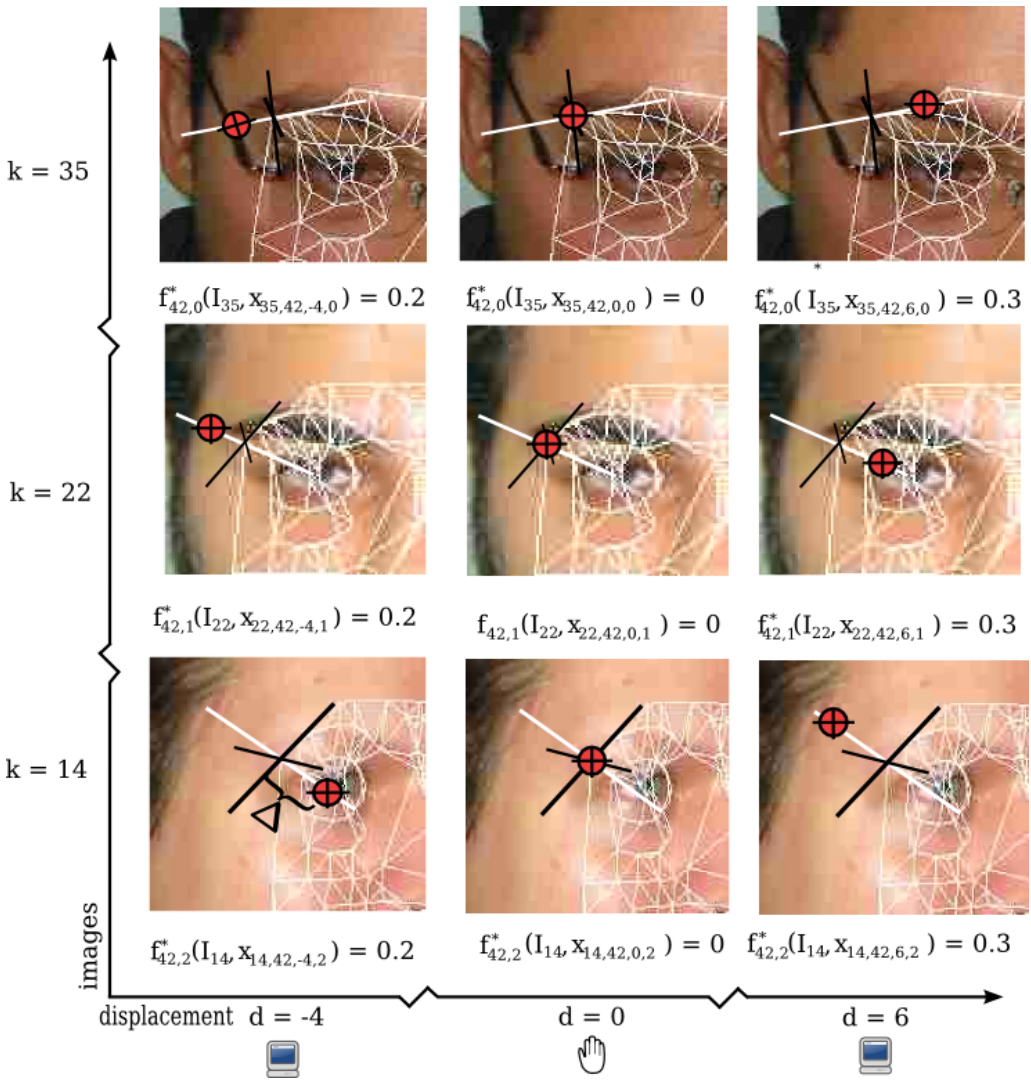


Figure 5: Further annotations are generated by moving along the line that is longest when projected in the image. That line is coloured white here. The directions illustrated in black are not used.

on the model's surface. It is estimated with the help of the nearest model point. The second characteristic direction is perpendicular to the first characteristic direction and minimizes the distance to the center of gravity of the model. The third characteristic direction is obtained from the cross product.

3.3 Step 3: Specifying Image Features

The point $\boldsymbol{x}_{k,n,d,l}$ describes the annotation of the n^{th} model point in the image I_k, which is displaced by d along the characteristic direction l. Our approach learns the calculation rules of a function $f^{\ell}_{n,l}(I_k, \boldsymbol{x}_{k,n,d,l})$ that maps the values of an image I_k and an annotation $\boldsymbol{x}_{k,n,d,l}$ to the value that is computed by the ideal objective func-

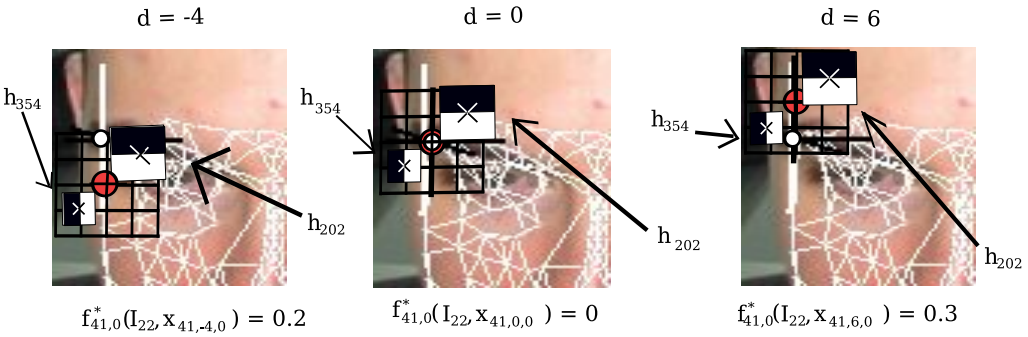


Figure 6: The grid of image features moves along with the displacement.

tion $f_n^\star(I_k, \boldsymbol{x}_{k,n,d,l})$. Since $f_{n,l}^\ell$ has no knowledge of $\boldsymbol{p}_I^\star$, it must compute its value from the content of the image.

Instead of learning a direct mapping from the pixel values of I in the vicinity of the projection of $\boldsymbol{x}$ to $f_n^\star(I, \boldsymbol{x})$, we use a feature-extracting method [6], which extracts features from the image around the projection of $\boldsymbol{x}$. Our idea is to provide a multitude of image features, and let the training algorithm choose which of them are relevant to the computation rules of the objective function and which are not. Each feature $h_a(I, \boldsymbol{x})$ with $1 \leq a \leq A$ is calculated from an image I and a particular location $\boldsymbol{x}$ and delivers a scalar value. The approach presented in this paper relies on Haar-like features. Note again, that $\boldsymbol{x}$ does not represent a 2D pixel position in the image I but a 3D position in Euclidean space. However, its corresponding position in I is obtained by applying the perspective projection.

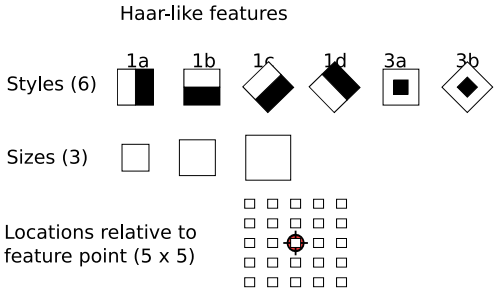


Figure 7: This comprehensive set of image features is provided for learning local objective functions. In our experiments, we use a total number of $A = 6 \cdot 3 \cdot 5 \cdot 5 = 450$ features.

Each Haar-like feature defines two regions of pixels, depicted in black and white in Figure 7. Its value is calculated by subtracting the sum of pixel intensities within the black region from the sum of pixel intensities within the white region. Figure 7 lists the styles and sizes of each Haar-like feature used in this paper. Furthermore, these features are not only computed at the location of the model point itself, but also at positions located on a grid within its vicinity, as shown in Figure 6 and Figure 7. This variety of styles, sizes, and locations delivers a set of A=450 different image features as we use it in our experiments in Section 4. This multitude of features enables the learned objective function to exploit the texture of the image at the projected model point and in its surrounding area. When moving the model point in space, the image features move along with it in the image, leading their values to change, as can be seen in Figure 6.

3.4 Step 4: Generating Training Data

The result of the manual annotation step (Step 1) and the automated annotation step (Step 2) is a list of correspondences between positions in space and the corresponding value of $f_n^\star$. Since K images, N model points, and $2D + 1$ displacements are landmarked these correspondences amount to $K \cdot N \cdot (2D+1)$. From every image for every point a line with Equation 4 illustrates the list of these correspondences. Applying the list of manually selected features to the list of correspondences yields the list of training data in Equation 5. This step simplifies matters greatly. Since each feature returns a single value, we hereby reduce the problem of mapping the vast amount of image data

$$\left[\quad I_k, \quad \boldsymbol{x}_{k,n,d,l}, \quad f_n^\star(I_k,\, \boldsymbol{x}_{k,n,d,l}) \right] \tag{4}$$

$$\left[\quad \boldsymbol{h}_1(I_k,\, \boldsymbol{x}_{k,n,d,l}), \ldots, \boldsymbol{h}_A(I_k,\, \boldsymbol{x}_{k,n,d,l}), \quad f_n^\star(I_k,\, \boldsymbol{x}_{k,n,d,l}) \right] \tag{5}$$

$$\text{with } 1 \leq k \leq K,\ 1 \leq n \leq N, l = g(p_{I_k}),\ -D \leq d \leq D$$

and the related pixel locations to the corresponding target value, to mapping a list of feature values to the target value. Note that the size of the training data amounts to $K(2D+1)$ records for each of the N model points.

3.5 Step 5: Learning the Calculation Rules

Given the training data from Equation 5, the goal is to now learn the function $f_{n,l}^\ell(I, \boldsymbol{x})$ that approximates $f_n^\star(I, \boldsymbol{x})$. Note that we are not simply relearning the already known function $f_n^\star$ in Equation 2. The difference is that $f_{n,l}^\ell$ does not require knowledge of $\boldsymbol{p}_I^\star$, and can therefore be applied to previously unseen images as well. We obtain this function by training a model tree [10, 14] with the comprehensive training data from Equation 5. The $N{\cdot}L$ local objective functions $f_{n,l}^\ell$ have to be learned separately. However, only those lines of Equation 5 are necessary to generate one local objective function where n and l of the function $f_{n,l}^\ell$ that is learned and $\boldsymbol{x}_{k,n,d,l}$ match.

Model trees are a generalization of regression trees and, in term, decision trees. Whereas decision trees have nominal values at their leaf nodes, model trees have line segments, allowing them to also map features to a continuous value, such as the value returned by the ideal objective function. One of the reasons for deciding for model trees is that they tend to select only features that are relevant to predict the target value. Therefore, they pick a small number of M_n Haar-like features from the provided set of $A \gg M_n$ features.

After executing these five steps, we obtain a local objective function $f_{n,l}^\ell$ for each model point n and each direction l. It can now be called with an arbitrary location $\boldsymbol{x}$ and an arbitrary direction l on an arbitrary image I. The learned model tree calculates the values of the specified features at this location from the content of the image and executes its calculation rules.

4 Experimental Evaluation

In this section, two experiments show the capability of fitting algorithms equipped with a learned objective function to fit a face model to previously unseen images. The objective functions were learned using a database of 240 images. The evaluation is performed on 80 test images with a non-overlapping set of individuals. Furthermore, the images differ in face pose, illumination, and background.

Our evaluations randomly displace the face models according to the manually specified pose. For every model point, the fitting process exhaustively searches along the most important characteristic direction in order to determine the global minimum of the local objective function. Then, the model parameters $\boldsymbol{p}$ are approximated. The figures below illustrate the average point-to-point error between the obtained model $\boldsymbol{p}$ and the manual annotation $\boldsymbol{p}_I^\star$.

Our first evaluation investigates the impact of iteratively executing the fitting process. Figure 8 illustrates that each iteration improves the model parameters. However, there is a lower bound to the quality of the obtained model fit, because more

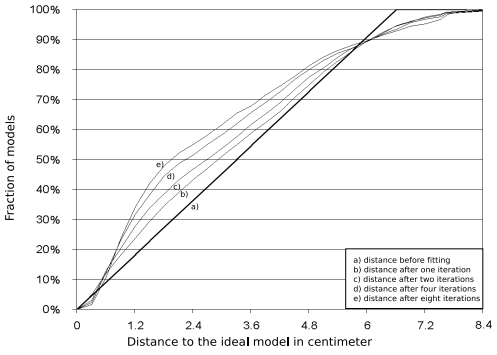


Figure 8: If the initial distance to the ideal pose is smaller than the learning radius this distance is improved within every iteration. Otherwise, the result of the fitting step is unpredictable and the models are spread further with every iteration.

than 10 iterations do not improve the fraction of well-fitted models significantly. In contrast, models with a high distance from the correct fit become even more with every iteration, because the objective function's value is arbitrary for these distances. Therefore, these model parameters are affected by a Gaussian distribution.

In our second experiment, we conduct model fitting by subsequently applying the fitting process with two different local objective functions f^A and f^B learned with decreasing learning radii Δ. f^A with a large Δ is able to handle large initial displacements in translation and rotation. However, the obtained fitting result gets less accurate, see Figure 9. The opposite holds true for f^B. The idea is to apply a local objective function learned with large Δ first and then gradually apply objective functions learned with smaller values for Δ. As opposed to the previous experiment, where we iteratively executed the same objective function, this iteration scheme executes different objective functions, which compensates the weakness of one function by the strength of another.

The advantage of concatenating algorithms with objective functions with decreasing learning radii compared to iterating one algorithm several times is illustrated by Figure 9. Sequentially applying f^A and f^B is significantly better than both of the other algorithms. Note that we execute each experiment with ten iterations, because we don't expect any improvement in quality with a higher number of iterations, see our first experiment. Therefore, the obtained accuracy from the sequential execution does not base of the fact that some additional iterations are applied.

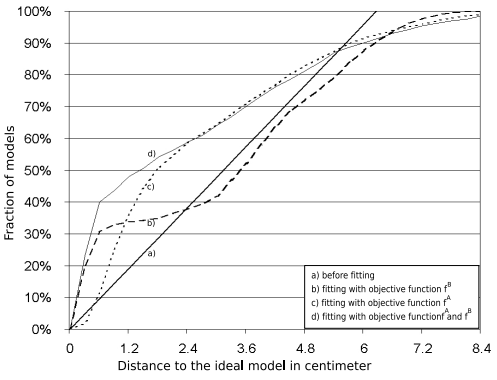


Figure 9: By combining fitting algorithms using objective functions with different learning radius we obtain result that show the strengths of both objective functions. The sequential approach shows the tolerance to errors of f^A and the accuracy of f^B

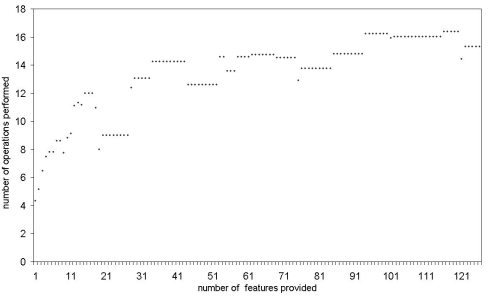


Figure 10: The number of rejected features increases with the number of features offered to the learning algorithm in total.

Our last experiment evaluates the timing characteristics of our approach by inspecting the number of operations performed when the objective function is executed with respect to the number of features provided. We consider basic arithmetic operations to be atomic as well as comparisons. The model tree generated by the learning algorithm selects the most important features provided. Therefore, usually only a small number of features $M \ll A$ is selected to build the model tree. As shown in Figure 10 When the number of features provided to the learning algorithm is small all of them are used. However, as the number of provided features is increased the algorithm rejects less relevant features. The more features are provided the larger the fraction of rejected features gets and the number of selected features converges to a fixed number.

5 Summary and Outlook

In this paper, we present a five-step methodology for learning local objective functions. This approach automates many critical decisions and the remaining manual steps require little domain-dependent knowledge. Furthermore, it contains no time-consuming loops within the design process. These features enable non-expert users to customize the fitting application to their specific domain.

The resulting objective function is not only able to process objects that look similarly, such as in [8, 7] but objects that differ significantly in shape and texture, such as human faces. Being trained with a limited number of annotated images as described in Section 3 the resulting objective function is able to fit faces that are not part of the training data as well. However, the database of annotated faces must be representative enough. If there were no bearded men in the training data the algorithm would have problem in fitting the model to an image of such a man. The disadvantage of our approach is the laborious annotation step. Gathering and annotating hundreds of images requires several days or even weeks.

In our ongoing research, we are applying our methods to tracking systems. Since tracking algorithms can exploit knowledge about the illumination, background, camera settings, and current pose of the model to bias search in the next image in the sequence, they perform faster and more accurately than algorithms for fitting a model to single images [11]. Given the accuracy and speed with which fitting algorithms can optimize learned objective functions, we expect to achieve excellent tracking results.

References

[1] N. Allezard, M. Dhome, and F. Jurie. Recognition of 3D textured objects by mixing view-based and model-based representations. In *Int. Conf. on Pattern Recognition*, pp 960–963, Barcelona, Spain, September 2000.

[2] V. Blanz and T. Vetter. Face recognition based on fitting a 3D morphable model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(9):1063–1074, 2003.

[3] W. Burgard C. Plageman, T. Müller. Vision-based 3d object localization using probabilistic models of appearance, 2005.

[4] T. F. Cootes, C. J. Taylor, A. Lanitis, D. H. Cooper, and J. Graham. Building and using flexible models incorporating grey-level information. In *International Fourth Conference of Computer Vision*, pp 242–246, 1993.

[5] D. Cristinacce and T. F. Cootes. Facial feature detection and tracking with automatic template selection. In *7th IEEE International Conference on Automatic Face and Gesture Recognition, Southampton, England*, pp 429–434, April 2006.

[6] R. Hanek. *Fitting Parametric Curve Models to Images Using Local Self-adapting Seperation Criteria*. PhD thesis, Department of Informatics, Technische Universität München, 2004.

[7] V. Lepetit, P. Lagger, and P. Fua. Randomized trees for real-time keypoint recognRealAudio Streamition. In *CVPR 2005*, vol 2, pp 775–781, Computer Vision Laboratory, EPFL, 1015 Lousanne, Switzerland, 2005.

[8] V. Lepetit, J. Pilet, and P. Fua. Point matching as a classification problem for fast and robust object pose estimation. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004)*, vol 2, pp 244–250, 2004.

[9] E. Marchand, P. Bouthemy, F. Chaumette, and V. Moreau. Robust real-time visual tracking using a 2D-3D model-based approach. In *International Conference on Computer Vision*, pp 262–268, Gorfu, Greece, 1999.

[10] R. Quinlan. Learning with continuous classes. In A. Adams and L. Sterling, editors, *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, pp 343–348, 1992.

[11] S. Romdhani. *Face Image Analysis using a Multiple Feature Fitting Strategy*. PhD thesis, University of Basel, Computer Science Department, Basel, CH, 2005.

[12] D. Simon, M. Hebert, and T. Kanade. Real-time 3-D pose estimation using a high-speed range sensor. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA '94)*, vol 3, pp 2235–2241, 1994.

[13] R. Cipolla and T. Drummond. 3d pose estimation of the face from video. In *Transaction on Pattern Analysis and Machine Inteligence*, Interval Research Corporation, 1801 Page Mill Road, Bldg. C, Palo Alto, CA 94304, 2002.

[14] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2005.

Qualitative Portrait Classification

Georgia Albuquerque, Timo Stich, Marcus Magnor

Computer Graphics Lab, TU Braunschweig
Mühlenpfordtstr. 23, 38106 Braunschweig, Germany
Email: {georgia,stich,magnor}@cg.tu-bs.de

Abstract

Due to recent advances in high-quality digital photography, taking a large series of images is very inexpensive. Especially in portrait situations, this results in a possible advantage because subjects often feel uncomfortable during acquisition. Selecting from a larger set of images increases the chance of a more satisfying outcome. However, the selection process is not easy and time consuming as only a small number of images is typically considered as aesthetically pleasing. In this work, we propose a machine learning approach to mimic the selection process of a human subject. After a short training period, a large set of images can be classified instantly into two categories, *good* or *bad*. With the proposed automatic pre-selection, the advantage of digital photography for portrait images is brought to a new level.

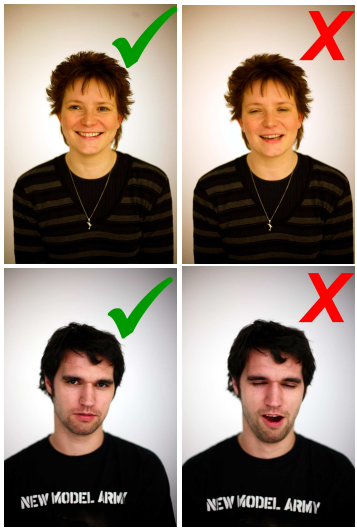


Figure 1: Samples of *good* and *bad* portraits.

1 Introduction

In the early days of photography, it was common to have people waiting for several minutes in front of the camera until the image could impress sufficiently the photographic film. It was common to retouch some areas of the photo, such as eyes, mouth and hands, which could hardly be kept immobile for long time. Later, the technology of fast films allowed to overcome this problem, but yet at a high cost of photographic material, such as special pellicles and a complex film development process.

Recent advances in high-quality digital photography allow the acquisition of large series of images in a very inexpensive way. Modern digital cameras are capable of taking and storing hundreds of high-resolution pictures within seconds. Some of them allow capturing video streams which are converted later to a sequence of individual photos. These improvements turn out in a low cost per picture for the photographer and more comfort for the photographed subject. Especially in portrait situations, such as studio or interview photos, the subject may be entertained or involved in a conversation while a fast sequence of pictures is captured. Later, the best photos from the sequence may be extracted, and the others may be discarded or digitally processed. In general, photos obtained in that way look much more natural and expressive because the subject is not worried about positioning himself into an appropriate pose.

Selecting from a large set of photos increases the chance for a better outcome. However, the selection process is very time consuming and usually not easy. While the price for acquiring and storing the images drops, the amount of produced pictures makes it tedious to run the selection phase manually. In other words, a non-automated classification

of several hundred photos quickly becomes an overwhelming task. The main reason is that selection should be applied involving complex criteria over a large set, where only some few elements are considered aesthetically pleasing. Moreover, the group of chosen photos may be highly dependent on the person running the selection, even if the acceptance rules are clearly stated.

For these reasons an automated solution for classifying portraits becomes very interesting. A semi-automated portrait classifier should be able to get rid of portraits that are obviously bad according to some criteria. The remaining portraits may be then finely classified by a human. A completely automated classifier system should be able to indicate securely the photos in the set which would most fit the predilection of the system user. For both cases, it is necessary to mimic the selection process of a human user. This is a non-trivial task, because the rules for selection of pleasant and aesthetic portraits are very problematic to be expressed in objective ways.

1.1 Related Work

Image classification is an important part of remote sensing, image analysis, and pattern recognition. It has a wide range of applications in many different areas like, for example, classification of satellite images and air photos [8], image content based search engines in the Web[12], and Biometric recognition[4][6]. Although these works present particularities dependent on the modality, all of them classify images into classes defined by previously known subjects.

The main challenges in this work are likewise how to identify facial elements, extract features and the final classification. Moreover the system should be able to learn the user's taste, based on user input.

Considering the actual research work on recognizing and classifying subject portraits, the *Image Intelligence* [2] technology from *Fujifilm* brings some approaches for face picture classification based on face recognition. On the World Wide Web, a good example of portrait a recognition-classification system is the Riya System [12]. It is a new kind of visual search engine specialized in facial recognition cataloguing. It offers the users the possibility to find similar faces and objects on many images across the web.

Most research made with facial picture classification considers face recognition. In other words, the main objective is to recognise *who* is the subject in the picture. That is the goal, for example, in biometrics authentication area. Although we use some common techniques, we are mainly interested in classifying pictures into qualitative categories such as good or bad.

In this point of view, some similar work can be found in the area of affective computing, an application of pattern recognition introduced by Picard [11]. The recognition of facial expressions, brings many contributions to the face classification area. Pantic and Rothkrantz[9] provide an overview over the area of facial expression analysis.

Some contributions for the qualitative classification of portraits were introduced in [13] with their concept of identifying neutral faces. The main motivation for this work is augmenting the accuracy of an arbitrary authentication algorithm by feeding it with a neutral face. Moreover, an impressive method for portrait images processing is presented in [7]. In contrast to a pattern recognition approach, the method automatically increases the predicted attractiveness rating of the face image. However, their method actually distorts the face and the beautification comes at the cost of identity loss.

All the cited work have some indirect relation to our topic, but as far as we know, no work was developed in order to separate portraits in adequation qualitative classes, as *good* or *bad* (Figure 1).

2 Portrait Classification

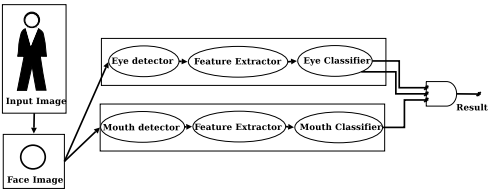


Figure 2: First the face is selected, then eyes and mouth regions are separated and used to feed the feature extraction module. Finally, the features are classified. If both eyes and mouth are classified as *good* the picture is considered good, else it is considered *bad*.

The main goal of this work is, using machine

learn techniques, mimic the selection process of a human subject allowing later an automatic selection of good portrait photos from a sequence of natural poses. This section presents an overview of our system with the main stages needed to evaluate a portrait image and classify it as a *good* or *bad* shot (Figure 2).

As input, our method receives pictures containing a single person photographed in a frontal pose. It inspects them to determine whether the presented picture is a good shot or not. However, the perception of how "good" a picture is very subjective. For example, some persons may prefer portraits where the photographed subject appears smiling, while others may consider the smile a sign of frivolity or distrust. This personal aspect restrains the acquisition of an enough general training set for the application. We propose two distinct approaches to overcome this problem. The difference between these approaches depends only on the origin of the training set.

Since mouth and eyes play extraordinary roles in facial expressions, we limit the classification to those features. In this way, we can focus on smaller, more significant regions of the picture.

The first approach is a more general system, which tries to learn the common sense of pleasantness of an image. The idea is to identify and discard face images with undesired details as, for example, closed eyes or mouths in movement (often present when someone speaks). For this mode of operation, we trained our portrait classifier with samples of eyes and mouth images in many different configurations. The samples were generated from images of a face database and were labelled as *good* or *bad*, according to simple criteria as discussed previously.

The second approach requires additionally two input image sets selected by the user, one containing good shots and the other disliked pictures. Our system then tries to learn the person's taste for the following classifications.

2.1 Detecting Face Regions

At any classification procedure, the feature extraction process plays a very important role. Largely applied in the field of machine learning and pattern recognition, feature extraction is an intelligent way to reduce the dimensionality of a large set of data, in our case, face images. The reduction of the dimensionality minimizes the amount of resources required to describe an image, and thus the resource demand for the training algorithms. We have chosen Adaptative Boosting (AdaBoost)[3] combined with Principal Component Analyses (PCA)[10] in order to define our feature set. The AdaBoost module detects the eyes and mouth regions, while the PCA module extracts the main components of these regions.

Adaptative Boosting, also called AdaBoost, was introduced in 1995 in [3]. It is a special case of Boosting, which is a general way to increase the accuracy of any given learning algorithm. We employ for our face detection module a very interesting AdaBoost implementation proposed by Viola and Jones in [16] to select the eyes and mouth regions of a portrait.

For our approach we trained two different AdaBoost modules, one for eye detection and another for mouth detection. Both classifiers are trained by two image sets. A positive set, containing images from the object of interest and a negative set, containing background images i.e. any possible image that does not correspond to the object.

To train the cascade of classifiers, the AdaBoost algorithm tries to meet an adequate trade-off between the quantity of features chosen for the classifier and the time necessary to compute the classifier. Each stage in the cascade reduces the false positive rate and decreases the detection rate as well. Each classifier stage is trained by adding features until a desired level of detection and false positive rates for the respective stage is reached. Similarly, stages are added until the overall desired levels of detection and false positive rates are met (see Figure 3 and 4).

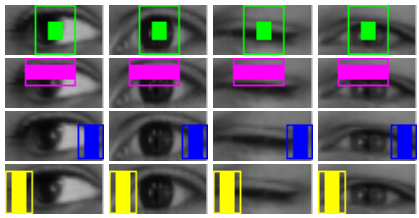


Figure 3: Haar-like features for the first stage of the AdaBoost algorithm trained to detect eyes. The first up to fourth features are marked in green, magenta, blue, and yellow, respectively.

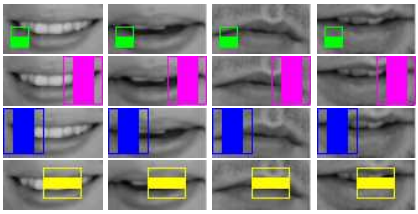


Figure 4: Haar-like features for the first stage of the AdaBoost algorithm trained to detect mouths. First up to fourth features are marked in green, magenta, blue, and yellow, respectively.

2.2 Feature Extraction

After detecting the regions of interest (eyes and mouth), the next step is to extract specific features of these regions. These are usually more robust than working directly on pixel values and can additionally encode ad-hoc domain knowledge which is otherwise difficult to learn using a finite quantity of training data. Using Principal Component Analysis (PCA) [14] is a common approach to find such robust feature descriptions in images. We follow this approach, by using the set of eyes and mouths previously selected by the detection module to calculate an eigenobject basis. Another possibility for a training set, used for the general classification mode, are the images used to train the detector module. This approach has the advantage that the images are better controlled, giving better results.

Figure 5 shows the ten most significant eigenobjects calculated for the eye and mouth spaces respectively. Interesting aspects from the eigenobject pictures can be seen. For example, the first eigenobject picture of the eye space represents eyes that look straight very well, while the second an third eigenobjects represent eyes looking to the right and left, respectively. In the case of the mouth space, the first eigenobject represents a laughing mouth, while the second represents a closed mouth. The appearance of the eigenobjects supports the hypothesis that our training set is very well represented in terms of the extracted eigenobjects.

2.3 Classification

The last step of our method is the final classification which partitions the input into a number of categories or classes[5]. In the case of recognizing whether a face image is a good shot or not, two final classes are defined: one class for the ”good” and another for the ”bad” pictures.

Figure 5: 5 first calculated eigenobjects for the eye and mouth space, respectively. Sorted from the more significant to the less significant.

The classification approach adopted in this work follows a geometric approach based on decision boundaries. We chose to use a Support Vector Machine[15] (SVM) classifier in order to categorize the final selected features. The introduction of support vector classifiers by Vapnik[15] is one of the most interesting advances in classifier design. The SVM maximizes the margin between the classes by selecting a minimum number of support vectors. The SVM algorithm is now-a-days one of the most commonly used classifiers and has many advantages, e.g., it can generate nonlinear classifiers with a very good generalization performance, even when using a small training set. Furthermore, when a large training set is used, the SVM classifier is able to select the minimal set of support vectors. This minimizes the computing requirements when testing new samples and avoids overfitting.

For our approach we trained two distinct classification modules, one for eye and another for mouth classification. If both eyes and mouth are classified as *good* the picture is considered good, else it is considered *bad*. The training sets used for the classification modules are explained in detail in the sections 3.3.1 and 3.3.2.

3 Results

This section describes some results obtained using our approach to classify portraits, including details about the structure and training of the different modules presented in Section 2.

In order to demonstrate the efficiency of our approach for portrait classification, we performed experiments with a pre-stored sequence of face images from different subjects in different poses. The used image database contains both training and test sets, but the composition of such sets variate depending

on the experiment.

3.1 Face Database

The first step in the implementation of the system was the creation of a face database, that is a database of face portraits. The major incentive to create a new database comes from the need of a large quantity of data and the idea of training and testing the system with real, i.e. not pre-processed images. The images were taken in a semi-controlled environment: Only frontal face pictures avoiding head rotations of more than 15 degrees and each picture including only one person. Moreover, the image acquisition was made at different days, and though under different conditions of lighting.

The camera utilized during the acquisition of the database images was a Canon EOS 5D, with shutter speed of $1/6$ Sec., aperture of F1.6, Lens 50mm and Focal Length 50.0mm. Furthermore, the shots were taken in a *Continuous Shot Mode* of one picture per second and stored in a Canon proprietary raw format to avoid compression artefacts.

The database contains a total of 1262 images from 12 distinct subjects at different facial poses, as for example: Open and closed eyes, looking toward different directions, laughing, speaking, yawning, etc.

3.2 Object Selection

We implemented two object selection modules: One for eye detection and another for mouth detection of face images. Both modules are similar. They differ only in the image training set and some simple restrictions discussed in section 3.2.1

For the eye detection module, we trained the system using 834 positive samples in different configurations and 2168 negative samples containing pieces of the background of some face images. After the training process, a classifier with 16 stages was achieved.

The mouth detection module was trained using 472 positive samples and 1884 negative samples. Figures 6 and 7 show some examples of positive and negative samples used to train the detector for eyes and mouths, respectively.

The training time for the AdaBoost algorithm is rather long. It takes several hours to accomplish a satisfying object detector. However, once the detector is adequately trained, the features that compose the classifier for each AdaBoost stage can be stored. After the training, the classifiers detect objects in real-time.

Figure 6: Example of images from the eye and mouth databases, to train the eye and mouth detection modules, respectively.

Figure 7: Examples of negative samples to train the eye and mouth detection modules, respectively.

3.2.1 Detection Results

The face selection module extracts the face from an input image and feeds it into the selection modules for eyes and mouths. It was implemented using a stage classifier with 24 stages, pre-defined in the face detection application from OpenCV[1]. The detector returns any possible occurrence of a face in a given image. Occasionally, the stage classifier misclassifies one region and extracts some non-face images. We call such regions a false positive detection. In general, the face selector module achieved a high detection rate. About 99% of the faces were detected successfully in a test set containing 645 pictures from 10 different subjects. False positive detections were extracted in 12% of the pictures additionally to the correct face. However, the false positive detections are not meaningful for the face detection module. Even if the program extracts some other non-face regions, the application searches in each of the candidate regions for at least two eyes and one mouth occurrences. Because the eye and mouth detection modules are unlikely to find target objects in a non-face region, this kind of region is naturally discarded.

In contrast to the face detection case, false positive detections arising during the eye and mouth selection may be problematic. That is because no filtering is made after the regions are selected.

Thus, we apply some simple location constraints for the eye and mouth detector in order to reduce the false positive detection. Our eye detection module achieved a detection rate of 90% over 332 previously selected faces, i.e. 664 eyes. Additional false positive detections were found in 7% of the images. Such set of portraits considered 5 distinct subjects, where three of them did not contributed to the training set of the eye detector. Considering the 10% of non-detected eyes, about 75% of them consisted of closed eyes. In this case, it is not a problem to reject the picture, since a portrait with closed eyes is not considered a *good* portrait. Table 1 shows the individual results per subject. The second column indicates the rate of eyes correctly extracted from the quantity in column 4. The false positive indicator corresponds to the quota of portraits where missdetected regions were additionally found. Similarly for the mouth selection, the system achieved 97% of successful detection over the same set of portraits. Table 2 shows the individual results per subject.

Figure 8 shows an example of the complete object detection process, including the face and face components selection.

Subject	Detection Rate	False Positives	Eye Quantity
Person 1	91%	13%	150
Person 2	80%	5%	110
Person 3	93%	2%	110
Person 4	97%	9%	154
Person 5	86%	24%	140

Table 1: Eye detection, individual results. The pictures of the last three persons in the table did not contribute to the training set for the eye detection module.

Subject	Detection Rate	False Positives	Mouth Quantity
Person 1	97%	19%	70
Person 2	96%	36%	55
Person 3	96%	35%	55
Person 4	97%	66%	77
Person 5	97%	47%	70

Table 2: Mouth detection, individual results. The pictures of the last three persons in the table did not contribute to the training set for the mouth detection module.

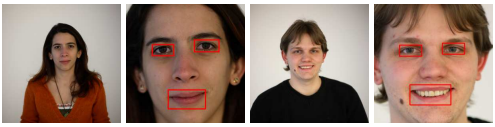


Figure 8: complete object detection process, including the face and face components selection.

3.3 Classification

This section shows some results accomplished with both classification approaches: The general approach, where the classifier is trained with previously defined sample sets of eyes and mouths labelled as *good* or *bad*. And the personal approach, where the user supplies a training face image set, representing his personal taste, labelled in the same way.

3.3.1 General Classifier

In order to build the eyes and mouth training set for the general classifier, we selected eyes and mouths from the detection module positive sample set. The labelling criteria for eyes and mouths follows a simple rule: An eye is labelled as *bad* when it is closed or looks to the right or left. Otherwise, it is labelled as *good*. A mouth is labelled as *bad* when it is open, when it makes some movement to speak, or when the smile is too large. Figures 9 and 10 show, respectively, examples of our training sets for eyes and mouths.

(a) Eyes labeled as good.

(b) Eyes labeled as bad.

Figure 9: Example of images (eyes) from the training set used for the general classification mode. Manually labeled as *good* or *bad*.

We tested the general classifier approach with 7 different subjects. Of the 7 persons used in this test, only two of them have taken part in the training set for the classification. This image test set is distinct

(a) Mouths labeled as good.

(b) Mouths labeled as bad.

Figure 10: Example of images (mouth) from the training set used for the general classification mode. Manually labeled as *good* or *bad*.

from image training set considered for the detection module and for the classifier. Table 3, shows some results for the general classification test. The column *Hit Rate* lists the percentage of face images which were correctly classified. It variates in range from 67% to 96%.

Most classification errors occurred when the images were to be classified as *bad* but were classified as *good*. Similarly to the terminology adopted in the detection phase, we call such misclassified elements a false positive classification. In the same way, we call it a false negative classification if the element was classified as *bad*, but was actually to be classified as *good*.

One can notice the strong incidence of false positive classification when classifying mouth elements. Analyzing the images where this kind of error occurs, we perceived that a considerable part of them have a mouth pose that should be considered *bad*. For such cases, our mouth classifier was not able to generalize as well as the eye classifier for dealing with counter-examples. That is explained by the fact that the mouth negative training set was not large enough. It implies that some new and undesired mouth configurations were not covered adequately by the training set. For the classification of eye elements the test rejected all occurrences of closed eyes and most of occurrences of eyes looking to right of left.

The variation in the classification hit rate for the different subjects lies mainly in the fact that between the bad images, some have more pictures with bad eyes while others with bad mouth poses. For example, most of the misclassified bad pictures for *person 5* were cases in which the person slightly pressed the lips, while the eyes were considered good. In this case, additional *bad* mouth poses are required for the training of the classifier.

	Hit Rate	**Miss rate**		**Samples**	
		FP	FN	Good	Bad
1	88%	3%	9%	20	13
2	79%	0%	21%	37	29
3	96%	0%	4%	13	14
4	79%	0%	21%	27	29
5	67%	0%	33%	32	22
6	80%	5%	15%	57	17
7	93%	7%	0%	13	16

Table 3: Classification results for the general mode, where the classifier is trained with previously defined sample sets of eyes and mouths (FP means False Positive and FN, False Negative). The persons 2 and 4 have taken participation in the training of the classifier.

3.3.2 Personal Classifier

In order to validate the personal classification mode, we executed a leave-one-out cross-validation with the same subjects from the general classification approach. For each subject, training sets are supplied, one containing face images labelled as *good* and another labelled as *bad*.

For each iteration of the test, the eyes and mouth are extracted from the pictures that compose the training set. If the picture is labelled as *good*, the extracted regions are also considered good, else the extracted regions are considered bad. After all the eye and mouth elements are extracted and labelled, the software extracts the features of the regions and uses them to train the eyes and mouth classifier, respectively. When the classifier is trained, the test picture can be classified. In a similar process to the used in the training phase, the eye and mouth elements from the test image are detected and its features are then evaluated by corresponding classifiers. If the two detected eyes and the mouth are classified as *good*, the picture is, as well, indicated to be good. Otherwise it is considered a bad picture.

The results of the leave-one-out cross-validation are shown in table 4. The column *Hit rate* indicates the percentage of the pictures that were classified accordingly with the original labels. For misclassified portraits, we indicate also the rate of false positive and false negative classification. The last two columns depict the amount of pictures for each subject and its original distribution in good and bad sets. More classification results can be seen in the Figure 12.

	Hit rate	Miss rate		Samples	
		FP	FN	Good	Bad
1	78%	19%	3%	20	12
2	70%	12%	18%	31	35
3	89%	7%	4%	13	14
4	89%	5,5%	5,5%	27	29
5	76%	7%	17%	32	22
6	72%	20%	8%	56	18
7	74%	16%	0%	13	17

Table 4: Results of the leave-one-out cross-validation for the personal classifier (FP means False Positive and FN, False Negative). Invidual results for 7 subjects

4 Conclusion & Future Work

In this work, we proposed a procedure to efficiently carry out a qualitative classification of portraits in a machine learning environment. To the best of our knowledge, we are the first to establish a workflow that entails such task. Our workflow brings together established techniques used in the area of feature extraction and machine learning. We are able to classify portraits within qualitative categories, such as good/pleasant and bad/unpleasant shots, with an accuracy rate up to 96 percent.

Our system is also capable of learning the user's preferences or taste. The user must not know details about the underlying classification system, but he indicates his taste by selecting example images for good and bad photos. We consider these aspects fundamental for the application of an qualitative image classification in realistic applications. Although our approach deals only with eyes and mouth regions of a portrait when inferring its quality the performance of our classifier was very good in general. To further increase the performance, the system may be extended to take in account other portrait elements, such as eyebrows, ears or the nose. Since the single detection units are independent, such additional features are easily added.

Another very interesting extension of our application would be to select the best shots from video sequences. The overhead to implement this extension is minimal because the set of video frames corresponds directly to a set of portraits in the face database.

References

[1] Intel Open Source Computer Vision Library. Website. http://www.intel.com/research/mrl/research/opencv/.

[2] FUJIFILM Corporation. Image intelligence. Website. http://www.fujifilm.com/image_intelligence/.

[3] Yoav Freunf and Robert E. Schapire. A decision-theoretic generalization of on-line learning and application to boosting. *In Computational Learning Theory: Eurocolt' 95*, pages 23–37, 1995.

[4] Anil K. Jain, Ruud Bolle, and Sharath Pankanti. *Biometrics: Personal Identification in Networked Society*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[5] Anil K. Jain, Robert P. W. Duin, and Jianchang Mao. Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

[6] S. Y. Kung, M. W. Mak, and S. H. Lin. *Biometric Authentication: A Machine Learning Approach*. Prentice Hall, 2004.

[7] Tommer Leyvand, Daniel Cohen-Or, Gideon Dror, and Dani Lischinski. Digital face beautification. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, page 169, New York, NY, USA, 2006. ACM Press.

[8] Duda T.; Canty M. Unsupervised classification of satellite imagery: choosing a good algorithm. *International Journal of Remote Sensing*, 23:2193–2212, 2002.

[9] Maja Pantic and Leon J. M. Rothkrantz. Automatic analysis of facial expressions: The state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1424–1445, 2000.

[10] K. Pearson. On lines and planes of closest fit to systems of points in space. *London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. Sixth Series.

[11] Rosalind W. Picard. *Affective Computing*. MIT Press, 1997.

[12] Inc. Riya. Rija - visual search. Website. http://www.riya.com/.

[13] Yingli Tian and Rudolf M. Bulle. Automatic detecting neutral face for face authentication. In *AAAI-03 Spring Symposium on Intelligent Multimedia Knowledge Management*, 2003.

[14] Matthew Turk and Alex Paul Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[15] V. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, New York, 1998.

[16] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *In IEEE Conference on Computer Vision and Pattern Recognition*, pages 609–615, 2001.

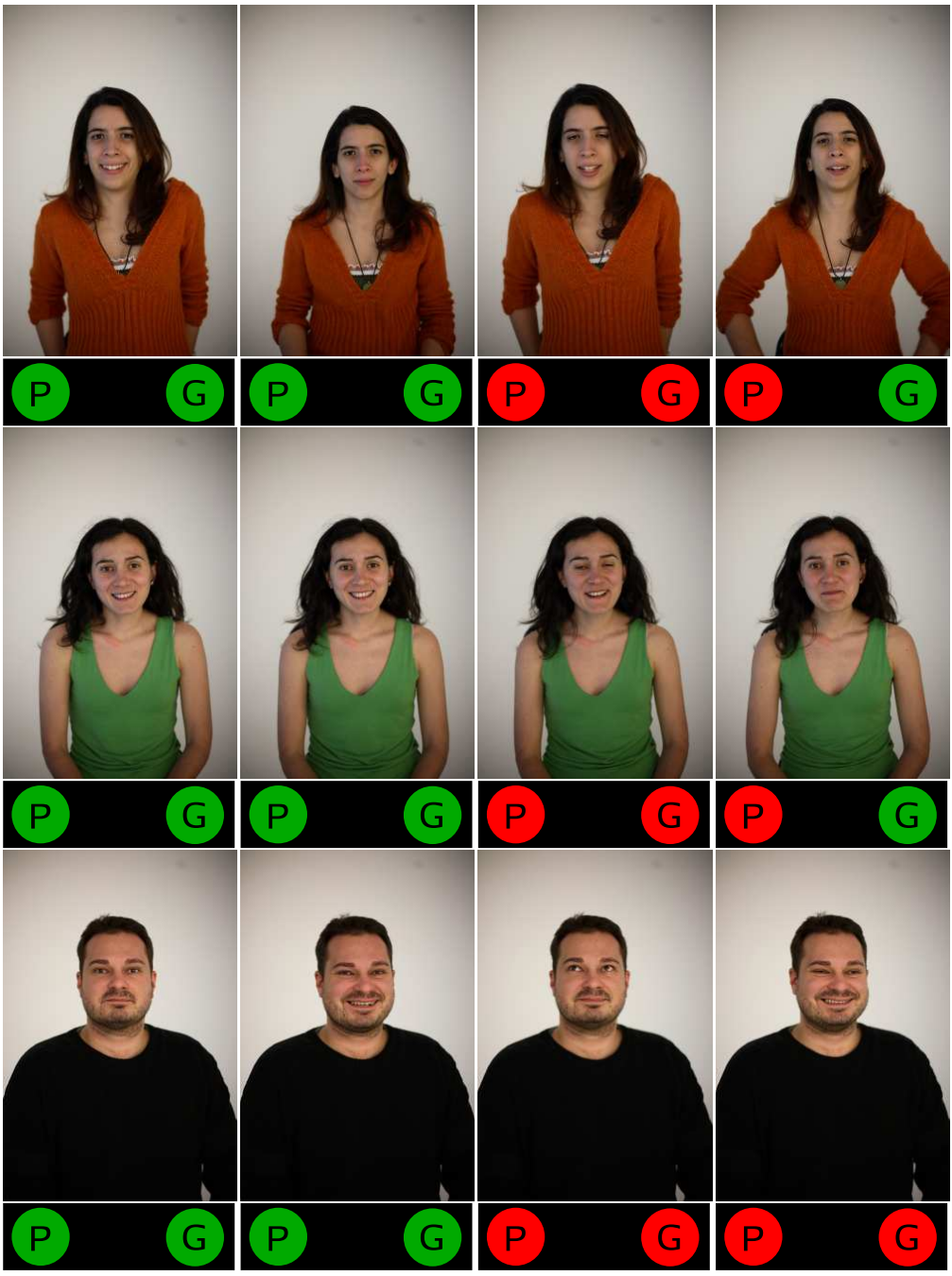


Figure 11: Example classfication results. The first circle is the vote of the Personal Classifier trained with the taste of one of the authors. The second shows the vote of the General Classifier.(green for good, red for bad)

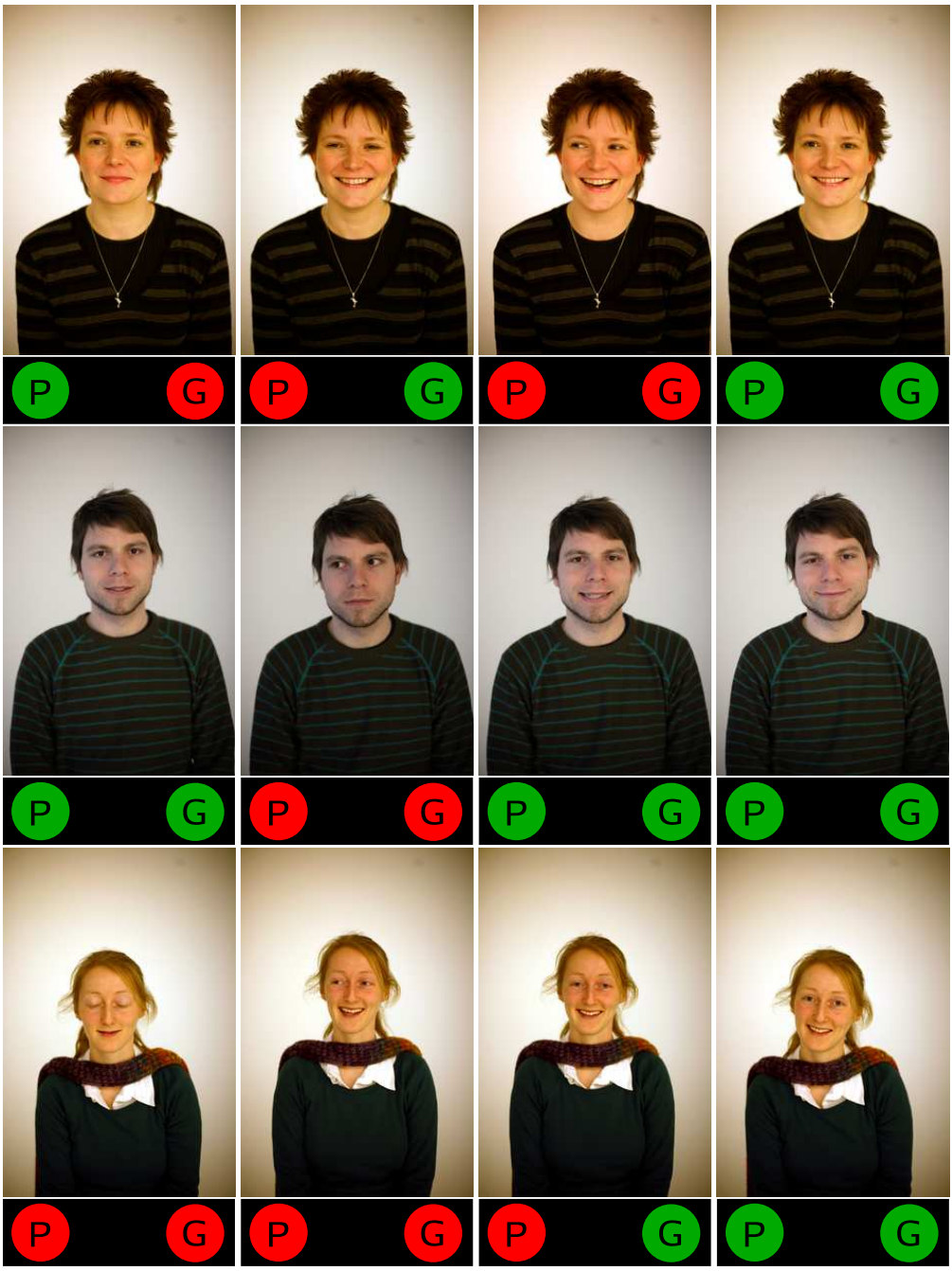


Figure 12: Example classfication results. The first is the vote of the Personal Classifier trained with the taste of one of the authors. The second shows the vote of the General Classifier. (green for good, red for bad)

Exemplar-based Parametric Hidden Markov Models for Recognition and Synthesis of Movements

Dennis Herzog, Volker Krüger, Daniel Grest

Computer Vision and Machine Intelligence Lab
CIT, Aalborg University Copenhagen
Lautrupvang 15
2750 Ballerup, Denmark
Email: {deh,vok,dag}@cvmi.aau.dk

Abstract

A common problem in movement recognition is the recognition of movements of a particular type. E.g. pointing movements are of a particular type but differ in terms of the pointing direction. Arm movements with the goal of reaching out and grasping an object are of a particular type but differ with the location of the involved object. In this paper, we present an exemplar-based parametric hidden Markov model (PHMM) that is able to recognize and synthesize movements of a particular type. The PHMM is based on exemplar movements that have to be "demonstrated" to the system. Recognition and synthesis are carried out through locally linear interpolation of the exemplar movements. Experiments are performed with pointing and grasping movements. Synthesis is done based on the object position as parameterization. In case of the recognition, the coordinates of the grasped or pointed at object are recovered. Our experiments show the flexibility of our exemplar-based PHMMs in terms of the amount of training data and its robustness in terms of noisy observation data.

1 Introduction

One of the major problems in action and movement[1] recognition is to recognize actions that are of the same type but can have very different appearances depending on the situation they appear in. In addition, for some actions these differences are of major importance in order to convey their meaning. Consider for example the movement of a human pointing at an object, "This object there...", with the finger pointing at a particular object. Clearly, for such an action, the action itself needs to be recognized but also the spot in 3D space at which the human is pointing. Only together do these two pieces of information convey the full semantics of the movement. Another common problem is the synthesis of action: This concerns two major problem areas: In robotics, one is interested in teaching robots through simple demonstrations (imitation learning) [3, 13, 2]. In 3D human body tracking, one is interested in using motion models in order to constrain the parameter space (e.g. [11] for simple cyclic motions). In both cases, one is interested in teaching the system in an easy and efficient manner a particular movement so that afterwards, the system is able to synthesize movements of the same type, however, with a different parameterization. Here, we consider grasping movements as an example where a human is reaching out for an object to grasp it[2]. One may perform as demonstration a set of grasping movements. All grasping movements depend on the location of the object to be grasped. In case of a humanoid robot, the synthesis should then allow the robot to perform the learned grasping movements with new parameterizations, e.g., grasping objects at different positions. In case of the 3D body tracking, the synthesis would allow a better prediction of the next pose.

Most current approaches model movements with a set of movement *prototypes*, and identify a movement by identifying the prototype which explains the observed movement best. This approach, how-

[1] we use the terms *action* and *movement* interchangingly. Actions usually denote movements that involve objects.

[2] The precise choice of a hand grasp depends on the type of object, from where it is being grasped, etc. In our discussion, we omit the issue of the different hand grasps and focus only on the arm movements.

ever, has its limits concerning efficiency when the space of possible parameterizations is large.

A pioneering work in this context was done by Wilson and Bobick [15]. Wilson and Bobick presented a parametric HMM approach that is able to learn an HMM based on a set of demonstrations. Their training and recognition approach is based on the EM algorithm, where the parameters of the movements are taken as latent variables. For recognition, they recover the parameter set that explains best the observation.

In this paper, we develop a different parametric hidden Markov model approach. Contrary to Wilson and Bobick, our aim is recognition as well as synthesis. Also, we would like to provide a simpler and more efficient training strategy by being able to simply provide exemplars based on which the generation of novel HMMs can be done.

In the following section, we give a short overview of the related work. In Sect. 3 and 4 we introduce our exemplar-based parametric HMMs. Extensive experimental results are presented in Sect. 5. Conclusions in Sect. 6 complete our paper.

2 Related Work

Most approaches for movement representation that are of interest in our problem context are trajectory based: Training trajectories, e.g., sequences of human body poses, are encoded in a suitable manner. Newly incoming trajectories are then compared with the previously trained ones. A recent review can be found in [9].

Some of the most common approaches to represent movement trajectories use hidden Markov models (HMMs) [12, 5]. HMMs offer a statistical framework for representing and recognition of movements. One major advantage of HMMs is their ability to compensate for some uncertainty in time. However, due to their nature, HMMs are only able to model specific movement trajectories, but they are not able to generalize over a class of movements that vary accordingly to a specific set of parameters.

One possibility to recognize an entire class of movements is to use a set of hidden Markov models (HMMs) in a mixture-of-experts approach, as first proposed in [7]. In order to deal with a large parameter space one ends up, however, with a lot of experts and a large amount of training becomes necessary.

Another extension of the classical HMMs into parametric HMMs was presented in [15], as mentioned above. A more recent approach was presented by [2]. In this work, the interpolation is carried out in spline space where the trajectory of the end-effector is modeled. Apart from the fact that the authors have not yet performed an evaluation of their system, their approach does not seem suitable for controlling the entire arm movements for movement synthesis and recognition.

In addition to HMMs, there are also other movement representations that are interesting in our context, e.g., [14, 8]. However, these approaches share the same problems as the HMM based approaches.

3 Preliminaries

A hidden Markov model is a probabilistic finite state machine, which is generally defined as a triple $\boldsymbol{\lambda} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$, where the transition matrix $\boldsymbol{A}$ defines the transition probability between the hidden states $q = 1, \ldots, N$, $\boldsymbol{B}$ defines the output probabilities of each state, and $\boldsymbol{\pi}$ defines the probabilities of each state of being the initial state of a hidden state sequence.

In this approach continues HMMs are used, whose output probabilities are modeled by single Gaussian distribution. For each state i the output distribution $b_i(\boldsymbol{x}) = \mathrm{P}(\boldsymbol{x}|q = i) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ is just defined by the parameters $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$. In this simple case of single Gaussian distributions it is more intuitive to use the notation $\mathcal{N}_i(\boldsymbol{x})$ instead of $b_i(\boldsymbol{x})$.

To facilitate synthesis of actions we chose to use left-right HMMs (as used in [4]) to model the actions. Here, a state sequence always has to start at the same state, therefore $\boldsymbol{\pi}$ is set $\boldsymbol{\pi} = (1, 0, \ldots, 0)$. In addition, state transition are restricted to the state itself or to the next state (other transition probabilities are set to zero). Such an HMM is depicted in Fig. 1.

3.1 Recognition using HMMs

For recognition or classification HMMs are generally used as follows: For each specific class k of sequences an HMM $\boldsymbol{\lambda}^k$ is trained by a representative training set $\mathcal{X}^k$ for that class. The training of an HMM $\boldsymbol{\lambda}$ is done by adjusting the model parameters to values, which are maximizing the likelihood

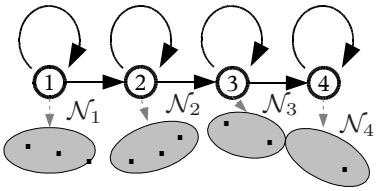


Figure 1: Left-to-Right HMM. For each state i the output normal distribution $\mathcal{N}_i$ is implied by an ellipsoid. An output sequence $\boldsymbol{X} = \boldsymbol{x}_1 \dots \boldsymbol{x}_9$ going from left to right is implied by small black dots. This sequence is likely to be generated by that HMM, where $\mathrm{P}(\boldsymbol{X}|\boldsymbol{\lambda}) \gg 0$.

function $\mathrm{P}(\mathcal{X}|\boldsymbol{\lambda})$. For this maximization, we apply the Baum/Welch algorithm [6].

The classification of an specific output sequence $\boldsymbol{X} = \boldsymbol{x}_1 \dots \boldsymbol{x}_T$ is done by selecting that class k, for which the likelihood $\mathrm{P}(\boldsymbol{X}|\boldsymbol{\lambda}^k) = \max_i \mathrm{P}(\boldsymbol{X}|\boldsymbol{\lambda}^i)$ is maximal.

One obvious approach for handling whole classes of parameterized actions for the purpose of action recognition and parameter estimation is a mixture-of-experts approach [7] and to sample the parameter space by training for each sample a prototype HMM. The HMM maximizing the likelihood of an given action sequence identifies class membership and the parameterization of the action. However, this approach is not appropriate, because too many repetitions of the action are needed to train the prototype HMMs of all samples.

4 Parametric HMMs

The main idea of our approach for handling whole classes of parameterized actions is a supervised learning approach where we generate an HMM for novel action parameters by locally linear interpolation of exemplar HMMs that were previously trained on exemplar movements with known parameters. The generation of a newly parameterized HMM can be done online or offline.

The interpolation of HMMs is carried out statewise, as we will explain in Sect. 4.1. As we will see, an interpolation is only possible, if the states of the exemplar HMMs are aligned in time. This alignment is discussed in Sect. 4.2. In Sect. 4.3 and 4.4, we explain the approaches for recognition and synthesis of the movements.

4.1 Interpolation of HMM

For simplicity the approach is explained in the case of a class of actions parameterized by a single parameter u, e. g., sequences of trajectories of a person's wrist leading to different positions on a table, where the positions are on a straight line leading from left to right on the table. This idea is depicted in Fig. 2.

It is assumed that two HMMs $\boldsymbol{\lambda}^{\mathrm{l}}$ and $\boldsymbol{\lambda}^{\mathrm{r}}$ belonging to motions of different parameterization u are given, where $u = 0$ and $u = 1$ respectively. Additionally, it is assumed that the Gaussians $\mathcal{N}_i^{\mathrm{l}}$ and $\mathcal{N}_i^{\mathrm{r}}$ of $\boldsymbol{\lambda}^{\mathrm{l}}$ and $\boldsymbol{\lambda}^{\mathrm{r}}$ are arranged as depicted in Fig. 2. Under this assumptions the Gaussians $\mathcal{N}_i^u$ of an HMM, that shall model motions leading to some parameterized location u, can be approximated by statewise interpolation of corresponding Gaussians. The interpolation of the Gaussians is directly applied to the means and sigmas:

$$\mathcal{N}_i^u(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_i^u, \boldsymbol{\Sigma}_i^u), \tag{1}$$

where

$$\begin{aligned} \boldsymbol{\mu}_i^u &= (1-u)\boldsymbol{\mu}_i^{\mathrm{l}} + u\boldsymbol{\mu}_i^{\mathrm{r}} \\ \boldsymbol{\Sigma}_i^u &= (1-u)\boldsymbol{\Sigma}_i^{\mathrm{l}} + u\boldsymbol{\Sigma}_i^{\mathrm{r}} \end{aligned} \tag{2}$$

This is very intuitive in the constellation of Fig. 2, e. g., for $u = 0.5$, the $\mathcal{N}_i^u$ would define horizontal-oriented ellipsoids laying between the $\mathcal{N}_i^{\mathrm{l}}$ and $\mathcal{N}_i^{\mathrm{r}}$. Clearly, the interpolation is not meaningful if the two Gaussians of the same state of different HMMs do not belong to the same part of different motions, e. g., if the third state of one HMM belongs to a part of a forward motion of a forward-and-backward motion and the third state of the other belongs to the backward part of that motion.

Therefore, it is crucial for a meaningful interpolation that learned HMMs are synchronized or statewise aligned.

4.2 Aligned or Synchronized Setup of several HMMs

Starting point are movements that are performed into different directions. Even if they are aligned through linear warping the alignment with in the training sequences is not sufficiently good. Dynamic time warping algorithms [10] are existing just for pairwise alignment of sequences. And time

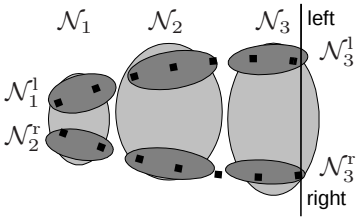


Figure 2: The upper three dark ellipsoids are depicting the output normal densities $\mathcal{N}_1^{\mathrm{l}}, \ldots, \mathcal{N}_3^{\mathrm{l}}$ of the states $1, \ldots, 3$ of an HMM $\boldsymbol{\lambda}^{\mathrm{l}}$ that is trained by sequences beginning on the left side of the picture and are leading onto the left of the vertical line, where the parameter of the parameterization of these sequences is supposed to be $u = 0$. The dots passing this ellipsoids imply one training sequence. Accordingly, the lower three ellipsoids due to an HMM $\boldsymbol{\lambda}^{\mathrm{r}}$ trained by sequences leading to the right of the line, where the parameter u is supposed to be $u = 1$. Additionally, Gaussians $\mathcal{N}_i$ of an $\boldsymbol{\lambda}$ trained by sequences of different parameterizations $u \in [0, 1]$ are indicated in light gray.

warping does not solve the problem of setting up the aligned HMMs. Our aim now is to train HMMs for non-aligned training data such that the HMM states correspond, so that Eq. (2) becomes senseful.

For easier explanation, we will use again the example setup from Sect. 4.1: For both HMMs $\boldsymbol{\lambda}^{\mathrm{l}}$, $\boldsymbol{\lambda}^{\mathrm{r}}$ training sets $\mathcal{X}^{\mathrm{l}}$ and $\mathcal{X}^{\mathrm{r}}$ are given. In order to find the two aligned HMMs $\boldsymbol{\lambda}^{\mathrm{r}}$ and $\boldsymbol{\lambda}^{\mathrm{l}}$, we first train a more general HMM $\boldsymbol{\lambda}$ for the whole training data $\mathcal{X}^{\mathrm{l}} \cup \mathcal{X}^{\mathrm{r}}$. In Fig. 2, we have depicted the general HMM $\boldsymbol{\lambda}$ with light gray ellipses. Now, for each sample $\boldsymbol{x}$ of each sequence $\boldsymbol{X}$ in $\mathcal{X}^{\mathrm{l}}$ and $\mathcal{X}^{\mathrm{r}}$, the closest Gaussian (or State) of the HMM $\boldsymbol{\lambda}$ can be easily assigned. HMM $\boldsymbol{\lambda}$ is then used as the starting point for the training of the specific HMMs $\boldsymbol{\lambda}^{\mathrm{l}}$ and $\boldsymbol{\lambda}^{\mathrm{r}}$ using this assignment. In that sense, HMM $\boldsymbol{\lambda}$ can be interpreted as a pre-alignment of the training data.

In the precise notion of the Baum/Welch EM-algorithm—an iterative algorithm, that repeatedly executes EM steps—that aligned setup is performed, e. g., by starting with the HMM $\boldsymbol{\lambda}^{\mathrm{l}} := \boldsymbol{\lambda}$ trained by the whole training set (likewise: $\boldsymbol{\lambda}^{\mathrm{r}} := \boldsymbol{\lambda}$). Then, one EM step is performed for $\boldsymbol{\lambda}^{\mathrm{l}}$ just using the data $\mathcal{X}^{\mathrm{l}}$ for training. The same procedure is applied to setup all HMMs for the needed key points in parameter space (like the HMM $\boldsymbol{\lambda}^{\mathrm{r}}$ using $\mathcal{X}^{\mathrm{r}}$ in our example). The single final EM step adapts the Gaussians $\mathcal{N}_i$ to its specific exemplar movements, but preserves the alignment.

Now, let's look at the precise EM steps, and let's consider $\boldsymbol{\lambda}^{\mathrm{l}}$. Let $\mathcal{X}^{\mathrm{l}} = \{\boldsymbol{X}^1, \ldots, \boldsymbol{X}^M\}$ be the training set of $\boldsymbol{\lambda}^{\mathrm{l}}$, which repetitions are denoted by $\boldsymbol{X}^k = \boldsymbol{x}_1^k \ldots \boldsymbol{x}_T^k$. In the E step the posterior probabilities $\gamma_t^k(i) = \mathrm{P}(q_t = i | \boldsymbol{X}^k, \boldsymbol{\lambda})$ of being in state i at time t (where $\boldsymbol{x}_t^k$ is emitted) are computed for each $\boldsymbol{X}^k$. Thus, $\gamma_t^k(i)$ defines the responsibility of state i for generating $\boldsymbol{x}_t^k$—or vice versa the membership of $\boldsymbol{x}_t^k$ to state i. In the M step the output probability density functions are re-estimated. Therefore, the means and covariance matrices are re-estimated for each Gaussian $\mathcal{N}_i$ of state i based on the responsibilities $\gamma_t^k(i)$. In the case of a single output sequence $\boldsymbol{X} = \boldsymbol{x}_1 \ldots \boldsymbol{x}_T$ the mean re-estimation

$$\boldsymbol{\mu}_i = \frac{\sum_t \gamma_t(i) \boldsymbol{x}_t}{\sum_t \gamma_t(i)} \tag{3}$$

can be regarded as the weighted mean of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$ by using the responsibilities as weights. The covariance re-estimation

$$\boldsymbol{\Sigma}_i = \frac{\sum_t \gamma_t(i) (\boldsymbol{x}_t - \boldsymbol{\mu}_t)^\top (\boldsymbol{x}_t - \boldsymbol{\mu}_t)}{\sum_t \gamma_t(i)} \tag{4}$$

can also be regarded as $\gamma_t(i)$-weighted covariance, where $\boldsymbol{\mu}_i$ denotes the previous re-estimation. (In the case of multiple observation $\boldsymbol{X}^k$ the nominators and denominators of (3,4) have to be extended to marginalize $\gamma_t^k(\cdot)$ and $\boldsymbol{x}_t^k$ over k.) As the responsibilities $\gamma_t(i)$ are chosen due to the general HMM $\boldsymbol{\lambda}$ each state of the HMM trained with a subset of the training data remains aligned to those of the other HMMs.

4.3 Recognition of Parameterized Sequences

The recognition of parameterized sequences is straightforward compared to the simple class classification. Given a sequence $\boldsymbol{X}$. For each class k and corresponding parameterized HMM $\boldsymbol{\lambda}_k^{\boldsymbol{\phi}}$ that class k and parameter $\boldsymbol{\phi} \in \mathbb{R}^n$ are taken, which yields the maximal likelihood $\max_{k,\boldsymbol{\phi}} \mathrm{P}(\boldsymbol{X} | \boldsymbol{\lambda}_k^{\boldsymbol{\phi}})$.

In the more general case $n > 1$ with $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_n)$ this is done by using gradient descent methods for the maximization of $P(\boldsymbol{X} | \boldsymbol{\lambda}_k^{\boldsymbol{\phi}})$

for each k separately. Therefore, for a fixed k the parameters of ϕ are iteratively adapted in the direction of the gradient $\frac{\partial}{\partial \phi} P(\boldsymbol{X}|\boldsymbol{\lambda}_k^{\phi})$, into which the likelihood function $f(\phi) = P(\boldsymbol{X}|\boldsymbol{\lambda}_k^{\phi})$ increases. Therefore, the likelihood function f has to be evaluated several times in the iteration process. The computation is done by the Forward/Backward Algorithm [6], which is the standard algorithm for this task. It is worth to be mentioned that the computationally cost of the evaluation of f is very small $\mathcal{O}(NT)$ in case of left-right HMMs. Here, N is the number of states, and T is the length of the output sequence.

In the case of several available exemplar HMMs, one needs to find those four closest exemplar HMMs that interpolate a newly observed movement most accurately. We do this by recursively comparing pairwise sets of HMMs. Consider the case of 2×3 aligned HMMs $\boldsymbol{\lambda}^{ij}$ with associated grasping positions $\boldsymbol{p}^{ij}$, that are forming a grid on the table such as

$$\begin{bmatrix} \boldsymbol{p}^{01} & \boldsymbol{p}^{11} & \boldsymbol{p}^{21} \\ \boldsymbol{p}^{00} & \boldsymbol{p}^{10} & \boldsymbol{p}^{20} \end{bmatrix} .$$

First, the bilinear interpolation parameters u, v of

$$\boldsymbol{\lambda}^{uv} = \begin{pmatrix} 1-u, u \end{pmatrix} \begin{bmatrix} \boldsymbol{\lambda}^{01} & \boldsymbol{\lambda}^{21} \\ \boldsymbol{\lambda}^{00} & \boldsymbol{\lambda}^{20} \end{bmatrix} \begin{pmatrix} 1-v \\ v \end{pmatrix}$$

maximizing $\mathrm{P}(\boldsymbol{X}|\boldsymbol{\lambda}^{uv})$ are determined. Now, if the point

$$\boldsymbol{p}^{uv} = \begin{pmatrix} 1-u, u \end{pmatrix} \begin{bmatrix} \boldsymbol{p}^{01} & \boldsymbol{p}^{21} \\ \boldsymbol{p}^{00} & \boldsymbol{p}^{20} \end{bmatrix} \begin{pmatrix} 1-v \\ v \end{pmatrix}$$

is lying on the left side of the line defined by $\boldsymbol{p}^{11}$ and $\boldsymbol{p}^{10}$ the procedure is repeated using the four left most HMMs $\boldsymbol{\lambda}^{01}, \boldsymbol{\lambda}^{00}, \boldsymbol{\lambda}^{11}, \boldsymbol{\lambda}^{10}$, otherwise using the four right most.

4.4 Action Synthesis

Suppose a grasp position $\boldsymbol{p}$ on the table-top is given. Then synthesis can be done as following:

1. The three HMMs $\boldsymbol{\lambda}^{i}{}_{,i=1,2,3}$ with closest associated grasp positions $\boldsymbol{p}^{i}$ are chosen under the constraint that the $\boldsymbol{p}^{i}$ are not collinear.
2. Then the interpolation parameters u, v are estimated as given by the point equation

$$\boldsymbol{p} - \boldsymbol{p}^1 = \left[\boldsymbol{p}^2 - \boldsymbol{p}^1 \middle| \boldsymbol{p}^3 - \boldsymbol{p}^1 \right] \begin{pmatrix} u \\ v \end{pmatrix} .$$

   Here, least-squares approximation has to be used in that case that $\boldsymbol{p}$ is not exactly in the plane of $\boldsymbol{p}^{i}$.
3. As basis for synthesis the interpolated HMM $\boldsymbol{\lambda}^{uv} = \boldsymbol{\lambda}^1 + u(\boldsymbol{\lambda}^2 - \boldsymbol{\lambda}^1) + v(\boldsymbol{\lambda}^3 - \boldsymbol{\lambda}^1)$ is used. The synthesized sequence is the sequence $\boldsymbol{\mu}_1 \ldots \boldsymbol{\mu}_N$ of the means of the HMM $\boldsymbol{\lambda}^{uv}$. As the number of states N is small compared to the original recorded sequences, the sequence is linearly interpolated.

5 Experiments

In our experiments we focus our considerations on pointing and grasping actions, which are in common action scenarios the most important movements. The pointing movements were movements such as “This object there...”. Our grasping movements are reaching towards a particular object in order to grasp it.

In our experiments we do not use visual tracking data but limit our considerations on data that is acquired using a motion capture system. This way, we exclude the vision problem and are able to focus only on the representational issues for movement representation.

The motion capture data is acquired with the electro-magnetic motion capture system *Motion Star* by *Ascension*[1]. For the capturing seven markers are placed on the person (see Fig. 3): one

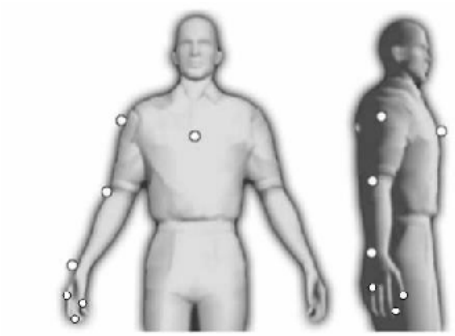


Figure 3: The dots mark the positions of the electromagnetic sensors in our dataset.

at the neck and one at the shoulder, the wrist, in the middle of the hand, at the index finger and the thumb of the right elbow. The captured sequences consist of the 3D point positions for each of the

seven markers recorded with 25Hz. The person or actor sits in front of a table (see Fig. 4). The positions on the table, on which the actions are performed, are covering a region of 30*cm* (in deeps) by 80*cm* (in width). For recognition the 3D point positions of the markers of right arm are directly used for training the HMMs. We have recorded six different exemplar movements (i.e. (action/grasping) movements to six different distinct positions on the table), each one with 9 repetitions. Two table positions are at the left and two are at the right side of the $30cm \times 80cm$ region. The other two exemplars are in the middle. For validation of the approach, additional movements to 10 different random positions on the table were recorded with 4 repetitions, each. All training and test sequences are normalized to 50 samples in length (2 sec.). No additional temporal alignment between the sequences was done.

Figure 4: The image shows the setup of the recording session of our dataset.

5.1 Synchronized Setup of HMMs

The synchronized setup of the special/prototype HMMs for the 6 exemplar positions is done as described in Sect. 4.2. As one of the aims is to provide an approach with fast and simple training, we will focus our investigation of synthesis and recognition performance vs. number of HMM states N, number of exemplar movements and number of repetitions R. In our experiments, we have trained HMMs with $N = 16, 28, 40$ states, with 4 or 6 exemplar movements and with $R = 3, 6, 9$ repetitions[3]. Fig. 5 shows example images of different states of the synchronized HMMs (yellow) and the general HMM (red). Here, one can verify the synchronization visually. In the recognition and synthesis experiments, below, we will implicitly verify the quality of the alignment.

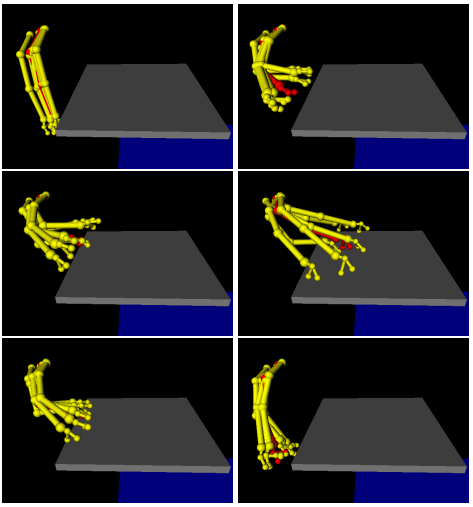


Figure 5: The 6 images (row-wise) are showing the states 1,7,9,14,25 and 30 of six synchronized HMMs with 40 states trained with grasping sequences of the six exemplar positions on the table (yellow arms) and the same states of the general HMM (red), that was used to align the states of the six HMMs.

5.2 Synthesis

In this section we summarize our results for the synthesis experiments. Here, we have tested the quality of the synthesis of movements, given a specific parameterization. The parameters were given as coordinates on the table. As test parameters we used those of the 10 random test movements. The quality of our synthesis approach has been tested based on these 10 random test movements, which serve here as ground truth references with known parameters (coordinates). The synthesis tests have been done for pointing and for grasping movements.

As explained above, we have trained different exemplar HMMs with different number of states and different number of repetitions. To estimate the synthesis quality we have tested based on

[3]The relatively high number of states is used to assure a reasonable good precision of the synthesis. Arguably, we could have chosen HMMs with less states.

1. different number of states $N = 16, 28, 40$,
2. different number of repetitions $R = 3, 6, 9$,
3. 4 and 6 exemplar movements for interpolation.

Tab. 1 and Tab. 2 summarize the experimental results for different number of HMM states, number of repetitions for each exemplar movement and number of exemplars used for the synthesis. As the intuitively best configuration, we chose a setting of 16 states, 6 repetitions for each exemplar and 6 exemplars movements. This setting is highlighted bold for grasping and pointing (top of each table).

The errors in the table were computed as follows: A movement is synthesized for each of the random table positions. Each synthesized movement is compared to the corresponding reference movement by computing the average Euclidean distance between the synthesized movement and the reference movement. The mean and standard deviation of this error are listed in the tables as *Synthesis Error* and is given in *cm*.

There is a natural variance in the human movements. As a reference, we have computed the mean error and the variance for the 40 different reference movements (10 movements with 4 repetitions each). These values are denoted in the tables as *Intrinsic Error* and are again given in *cm*.

Table 1: Synthesis of Grasping. For different numbers of States, Repetitions of exemplars and different number of Exemplars: the average Synthesis Error for 10 randomly chosen positions and the Standard Deviation in regard to the positions are listed in *cm*. Additionally, the Intrinsic Error Means and Deviations are listed.

			Synth. Err.		Intr. Err.	
St.	Rp.	Ex.	Mean	σ	Mean	σ
16	**6**	**6**	**2.7**	**0.7**	**1.4**	**0.4**
16	6	**4**	3.2	0.7	1.4	0.4
28	6	6	3.0	0.6	1.4	0.4
40	6	6	3.1	0.6	1.4	0.4
16	**3**	4	2.7	0.7	1.4	0.4
16	**9**	4	2.8	0.7	1.4	0.4

One can see that the synthesis error is rather small $\approx 3cm$, where the intrinsic error is above $1cm$. Furthermore, it is interesting to see that the influence of different number of states or repetitions on the synthesis quality is surprisingly small. Also interesting is that the experiment with only three repetitions and only four exemplars gave one of the best results.

Table 2: Synthesis of Grasping. For different numbers of States, Repetitions of exemplars and number of Exemplars: the Synthesis and Intrinsic Errors for 10 positions are listed in *cm*.

			Synth. Err.		Intr. Err.	
St.	Rp.	Ex.	Mean	σ	Mean	σ
16	**6**	**6**	**2.5**	**0.5**	**1.5**	**0.3**
16	6	**4**	2.6	0.5	1.5	0.3
28	6	6	2.6	0.5	1.5	0.3
40	6	6	2.6	0.5	1.5	0.3
16	**3**	4	2.5	0.4	1.5	0.3
16	**9**	4	2.4	0.5	1.5	0.3

5.3 Recognition

In the experiments for recognition, we do not test the ability to recognize the type of movement (pointing/grasping), but how good our approach is able to recover the parameterization of a newly observed movement. Once the parameters are known, the recovery of the movement type is trivially solved with ML. We have used for testing the 80 random movements (10 movements with 4 repetitions each for pointing and grasping) with known ground truth. For each of these movements, the parameters, i.e., the coordinates on the table, were recovered and compared to the ground truth values.

As above, we have run our experiment with different settings (number of states, number of repetitions used for the prototype exemplars and number of prototypes). The recognized positions $\boldsymbol{p} = \boldsymbol{p}(u, v)$ are calculated by using the interpolation parameters u, v, which maximize the likelihood function $f(u, v) = \mathrm{P}(\boldsymbol{X}|\boldsymbol{\lambda}^{uv})$ of the bilinear interpolation between the 4 HMMs of the nearest exemplars. As the grasping/pointing positions of the 6 exemplars are labeled, the recognized position $\boldsymbol{p}$ is calculated through bilinear interpolation.

In Tab. 3 and Tab. 4 the recognized position mean error and standard deviation with respect to the 80 test movements are summarized. It is interesting to note that since the random pointing actions were supposed to be performed in a "natural" way, the in-

Table 3: Recognition of Grasping Positions. The Recognition Error and Intrinsic Error are listed for each setup.

			Recog. Err.		Intr. Err.	
St.	Rp.	Ex.	Mean	σ	Mean	σ
16	**6**	**6**	**3.2**	**1.1**	**0.6**	**0.3**
16	6	**4**	5.0	1.5	0.6	0.3
28	6	6	2.7	1.1	0.6	0.3
40	6	6	2.7	1.2	0.6	0.3
16	**3**	4	3.0	1.0	0.6	0.3
16	**9**	4	3.2	1.1	0.6	0.3

Table 4: Recognition of Pointing Positions. Here, the recognition of the meant Pointing Positions on the table (upper block of table) and Fingertip Positions (lower block) are listed, separately. (The Fingertip doesn't touch the table!) The Recognition Error and Intrinsic Error are listed for each setup. However, as the meant pointing positions on the table are given ahead, no intrinsic error is meaningful.

			Recog. Err.		Intr. Err.	
St.	Rp.	Ex.	Mean	σ	Mean	σ
16	**6**	**6**	**2.5**	**1.0**		
16	6	**4**	3.7	0.8		
28	6	6	2.5	0.9		
40	6	6	2.4	0.9		
16	**3**	4	2.7	1.2		
16	**9**	4	2.3	0.9		
16	**6**	**6**	**4.2**	**1.9**	**2.0**	**0.9**
16	6	**4**	4.7	1.4	2.0	0.9
28	6	6	4.2	1.7	2.0	0.9
40	6	6	4.0	1.7	2.0	0.9
16	**3**	4	4.7	2.1	2.0	0.9
16	**9**	4	4.1	1.9	2.0	0.9

dex finger did not touch the table but rather stopped some *cm* above! Therefore, we summarize in the Tab. 4 two different errors: the error of the recognized pointing position on the table, which was meant by the person as elongation of the pointing direction, and the index finger tip position itself.

As before, we compute the intrinsic mean errors and standard deviations of the ground-truth coordinates of the 80 test movements. However, as the meant pointing position on the table is predefined and supposed to be correct, no intrinsic errors are listed. On the other hand, the grasping positions and the fingertip positions of the pointing actions are given by the motion sequences.

The entire experiment has been repeated with the random test movements disturbed by noise. The added noise was Gaussian with $\sigma = 5, 10, 15cm$. Tab. 5 and Tab. 6 summarize the results for $\sigma = 15cm$. For $\sigma = 5, 10cm$, no or only a very minor error increase was observable.

The errors of the recognized positions are acceptable ($\approx 3cm$) and as presumed slightly higher for the meant pointing positions on the table. In contrast to the results of synthesis fewer prototype exemplars (4 instead of 6) give significantly less accurate results (errors $\approx 4 - 5cm$).

Table 5: Recognition of Grasping Positions with Noise. The noise is normal distributed with sigma equal 15*cm*.

			Recog. Err.		Intr. Err.	
St.	Rp.	Ex.	Mean	σ	Mean	σ
16	**6**	**6**	**4.1**	**1.5**	**0.6**	**0.3**
16	6	**4**	5.7	1.3	0.6	0.3
28	6	6	3.5	1.3	0.6	0.3
40	6	6	3.0	1.3	0.6	0.3
16	**3**	4	4.5	1.7	0.6	0.3
16	**9**	4	3.0	1.2	0.6	0.3

6 Conclusion

In this paper we have presented an exemplar-based parametric hidden Markov model which allows to represent entire classes of movements. We have focused our considerations on human arm movements, but we believe that the approach can also be used in other contexts such as surveillance scenarios.

We have limited our considerations on movement data captured with an electro-magnetic motion capture system. We view the vision and the movement representation issues as two distinct problems and have focused our considerations on the movement representation alone. Presently, we are in the process of combining our movement representation with our 3D human body tracker.

Table 6: Recognition of Pointing Positions with Noise. The noise is normal distributed with sigma equal 15*cm*. Again, the recognition of the meant Pointing Positions on the table (upper block) and Fingertip Positions (lower block) are listed, separately.

			Recog. Err.		Intr. Err.	
St.	Rp.	Ex.	Mean	σ	Mean	σ
16	**6**	**6**	**4.0**	**0.8**		
16	6	**4**	4.3	1.0		
28	6	6	3.2	1.2		
40	6	6	3.2	1.2		
16	**3**	4	5.0	2.0		
16	**9**	4	4.5	2.2		
16	**6**	**6**	**5.1**	**1.2**	**2.0**	**0.9**
16	6	**4**	5.1	1.8	2.0	0.9
28	6	6	4.9	1.6	2.0	0.9
40	6	6	4.9	1.6	2.0	0.9
16	**3**	4	6.2	2.1	2.0	0.9
16	**9**	4	5.3	3.0	2.0	0.9

Acknowledgment.
This work was partially supported by EU through grant PACO-PLUS, FP6-2004-IST-4-27657.

References

[1] *Motion Star Real-Time Motion Capture*, http://www.ascension-tech.com/products/motionstar_10_04.pdf edition.

[2] T. Asfour, K. Welke, A. Ude, P. Azad, J. Hoeft, and R. Dillmann. Perceiving objects and movemetns to generate actions on a humanoid robot. In *Proc. Workshop: From features to actions – Unifying perspectives in compnational and robot vision, ICRA*, Rome, Italy, April 2007.

[3] B. Dariush. Human Motion Analysis for Biomechanics and Biomedicine. *Machine Vision and Applications*, 14:202–205, 2003.

[4] Simon Gunter and Horst Bunke. Optimizing the number of states, training iterations and gaussians in an hmm-based handwritten word recognizer. *icdar*, 01:472, 2003.

[5] X.D. Huang, Y. Ariki, and M.A. Jack. *Hidden Markov Models for Speech Recognition*. Edinburgh University Press, 1990.

[6] Xuedong Huang, Yasuo Ariki, and Mervyn Jack. *Hidden Markov Models for Speech Recognition*. Columbia University Press, New York, NY, USA, 1990.

[7] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

[8] C. Lu and N. Ferrier. Repetitive Motion Analysis: Segmentation and Event Classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):258–263, 2004.

[9] T. Moeslund, A. Hilton, and V. Krueger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104(2-3):90–127, 2006.

[10] M. Mozerov, I. Rius, X. Roca, and J. Gonzàlez. 3d human motion sequences synchronization using dense matching algorithm. In *In 28th Annual Symposium of the German Association for Pattern Recognition (DAGM'2006)*, volume LNCS-4174, pages 475–489, Berlin, Germany, September 2006.

[11] D. Ormoneit, H. Sidenbladh, M.J. Black, and T. Hastie. Learning and Tracking Cyclic Human Motion. In *Workshop on Human Modeling, Analysis and Synthesis at CVPR*, Hilton Head Island, South Carolina, June 13-15 2000.

[12] L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, pages 4–15, January 1986.

[13] S. Schaal. Is Imitation Learning the Route to Humanoid Robots? *Trends in Cognitive Sciences*, 3(6):233–242, 1999.

[14] D.D. Vecchio, R.M. Murray, and P. Perona. Decomposition of Human Motion into Dynamics-based Primitives with Application to Drawing Tasks. *Automatica*, 39(12):2085–2098, 2003.

[15] Andrew D. Wilson and Aaron F. Bobick. Parametric hidden markov models for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9):884–900, 1999.