This article has appeared in:
Proc. 30th Picture Coding Symposium (PCS 2013, Dec. 8-11, 2013, San Jose, CA). IEEE, Piscataway, 2013.
©IEEE. DOI: 10.1109/PCS.2013.6737726. Link to the paper

PROGRESSIVE MODES IN PDE-BASED IMAGE COMPRESSION

Christian Schmaltz, Nicolas Mach, Markus Mainberger, Joachim Weickert
Mathematical Image Analysis Group
Faculty of Mathematics and Computer Science, Campus, E1.7
Saarland University, 66041 Saarbrücken, Germany
Email: {schmaltz, mach, mainberger, weickert}@mia.uni-saarland.de

***Abstract*—Algorithms based on partial differential equations (PDEs) constitute a relatively novel class of lossy image compression methods. In this paper we introduce a practically relevant extension: We demonstrate how to incorporate progressive modes into these codecs. Since the data in PDE-based codecs is only available at irregular locations, this is a challenging task. We propose two progressive modes: The first one changes the order in which the grey values are stored, while the second additionally distributes the stored information more evenly over the file. Our experiments show that the novel codecs can outperform JPEG and even JPEG 2000 for high compression ratios.**

***Index Terms*—PDE-based image compression, progressive mode**

I. Introduction

The main task in lossy image compression is to reduce the file size of an image while degrading the image quality as little as possible. However, there are several additional requirements on an image compression algorithm in practise. In this paper, we focus on one of the most useful and widely employed additional requirements, namely the so-called *progressive mode*. This mode, which is also referred to as *embedded bitstreams* or *signal-to-noise ratio scalability*, allows to generate a coarse preview using only the beginning of a complete data stream. This is especially useful for applications in which bandwidth is scarce, e.g. when browsing a database of large images.

Progressive modes are readily available in most standard image compression codecs. Lossless image compression algorithms such as GIF and PNG use *interleaving* or *interlacing*, which only changes the order in which pixels are stored. In the lossy mode of the JPEG standard, there are three progressive modes that can be activated: (1) Low-frequent coefficients can be transmitted first, (2) one can store all upper bits of the coefficients first before the lower bits are stored, and (3) there is a hierarchical mode which stores a downsampled version of the image, which is used as a predictor for the next resolution. However, this can increase the total file size up to one third [1].

The JPEG 2000 algorithm [2] always stores images in progressive mode. This is achieved with the *Embedded Block Coding with Optimal Truncation* (*EBCOT*) scheme, which encodes bit-planes in three passes.

In contrast to these popular transformation-based approaches, there are also recent approaches that store only a subset of all image points, and build upon partial differential equations (*PDE*s) to reconstruct the remainder of the image. So far, PDE-based image compression methods do not include progressive modes, though. Note that standard progressive modes of existing image compression algorithms are not directly applicable to PDE-based image compression, as data is only available at irregular locations. Thus, the goal of this paper is to introduce progressive modes into PDE-based image compression algorithms.

While there is a long history on research on feature-based image representations where missing information is filled in by homogeneous diffusion (see [3]–[5], among others), most of the early works do not consider applications in image compression. Only recently, it has been shown that edge-based homogeneous diffusion approaches can beat JPEG 2000 for cartoon-like images [6], as well as for depth maps [7].

Apart from these linear, feature-based approaches, there are some attempts to use nonlinear PDEs for image compression. Chan and Zhou [8] propose a variational approach with total variation regularisation to minimise oscillations in wavelet decompositions. Work by Solé *et al.* [9] evaluates different PDEs for compression of digital elevation maps, and Liu *et al* [10] integrate inpainting into existing approaches.

The image compression algorithm of Galić *et al.* [11] uses points located on an adaptive triangulation, which is stored as binary tree. The remainder of the image is then reconstructed using edge-enhancing anisotropic diffusion (EED) [12]. The performance of this compression approach only lies between that of JPEG and JPEG 2000 for medium to high compression ratios. The R-EED approach from [13] builds upon these ideas. By changing a number of concepts, e.g. using rectangular subdivisions instead of triangular ones, the obtained image compression codec yields much better results. These results can even surpass those of JPEG 2000 for most compression ratios in images with little texture. In [14] it was demonstrated how to further improve the compression quality. Due to the promising results reported for the R-EED algorithm, we use it as the basis of our proposed progressive modes.

Our paper is organised as follows: First, we briefly explain the R-EED codec in Section II. In Section III, we illustrate two ways how to incorporate progressive modes into this codec. After evaluating these approaches in Section IV, we conclude the paper with a summary in Section IV.

II. The Baseline Compression Codec

The R-EED codec [14] only stores the grey values at a small subset of all pixels and interpolates the missing values when decoding. Here, we first describe this interpolation step.

Let $f : \Omega \to \mathbb{R}$ be the original image to be reconstructed from the grey values in a given subset $K \subset \Omega$. The set K is called the *interpolation mask*. The idea is to compute a reconstruction u such that u is equal to f in K, and such that u is smooth or piecewise smooth in $\Omega \setminus K$. Both conditions are met when solving the *partial differential equation* (*PDE*)

$$(1 - c_K)\, Lu - c_K\, (u - f) = 0 \tag{1}$$

for u with reflecting (i.e. homogeneous Neumann) boundary conditions. Thereby, L denotes a differential operator that ensures the requested smoothness properties, and c_K is the *characteristic function* of K, i.e. a function that is 1 at the specified data set K, and 0 elsewhere. Since this PDE always keeps the given points fixed, it can also be written as the simplified PDE $Lu = 0$ on $\Omega \setminus K$, with Dirichlet boundary conditions on K, and homogeneous Neumann boundary conditions on the boundary of Ω. This equation can be solved by introducing an artificial time parameter t and computing the steady state of the evolution equation $\partial_t u = Lu$ with a finite difference approximation. As proposed in [11], [14], we use *edge-enhancing anisotropic diffusion* (*EED*) [12] as interpolation operator:

$$Lu = \operatorname{div}(\boldsymbol{D}(\boldsymbol{\nabla} u_\sigma)\boldsymbol{\nabla} u). \tag{2}$$

Thereby, u_σ denotes the image u after smoothing with a 2-D Gaussian K_σ with standard deviation σ. The diffusion tensor $\boldsymbol{D}(\boldsymbol{\nabla} u_\sigma)$ is a symmetric matrix with eigenvectors $\boldsymbol{\nabla} u_\sigma$ and $\boldsymbol{\nabla} u_\sigma^\perp$ that are oriented across image edges, and along them, respectively. The corresponding eigenvalues are given by

$$\mu_1 = \frac{1}{\sqrt{1 + \frac{|\boldsymbol{\nabla} u_\sigma|^2}{\lambda^2}}}, \quad \mu_2 = 1. \tag{3}$$

This combines smooth interpolation along edges with discontinuity-preserving interpolation across edges. Finally, λ is a contrast parameter. R-EED stores the (quantised) value of λ for which the best results are obtained using a single byte.

To decide which pixels are kept in K, R-EED simulates the decompression step by reconstructing the image from five points, namely the four corners of the image and its centre. Then, original and reconstruction are compared by computing the mean squared error (MSE). If it exceeds a threshold T, the reconstruction is considered to be insufficiently close to the original. In this case, the image is split along the middle of the x or y direction, whichever is larger, and both subimages are recursively processed in the same way. Thereby, the threshold $T = a\ell^d$, depends on the free parameters a and ℓ, as well as on the recursion depth d.

Once these splitting steps are completed, it is known which points will be saved, namely the five points indicated by each rectangle. Instead of storing the position of each pixel individually, it is sufficient to store the binary tree containing the splitting decisions. This is done by first storing the depth until which all sub-images are split, as well as the depth at which no sub-image is split anymore. In between, one bit for each node of the tree needs to be saved. The obtained points form the inpainting mask K used for reconstructing the image.

The amount of data necessary to store the tree depends on its maximal and minimal depth. Thus, it pays off to specify a maximal depth up to which all rectangles are split independent of T, and a minimal depth at which no rectangle is split no matter what T is. This way, more points can be saved without increasing the file size, as less space is required for the tree.

The grey values of all points to be kept are then quantised and saved using a standard entropy coding scheme. Here, we use the simple Huffman coding algorithm to allow a fair comparison against JPEG and JPEG 2000. The Huffman codes used depend on the image to be compressed, and are thus stored in the file header.

It is possible to store brightness values which differ from the actual values in the input image. While this introduces an additional error at the encoded pixels, it can improve the overall reconstruction quality. We use a simple approach that optimises one grey value after the other, but more advanced optimisation schemes are possible, see [6] for details.

Due to quantisation of the stored grey values as well as the optimisations described above, the reconstructed values might be more accurate than the stored brightness values. Thus, after inpainting is done, all points close to the points in K are marked as unknown, and another inpainting step is done to reconstruct them from the remaining points. Details about all these steps can be found in [14].

III. R-EED in Progressive Mode

The original R-EED algorithm stores the brightness values row-wise. This is not only the simplest way to traverse all mask points, but also ensures that adjacent grey values in the resulting file are often close together in the image. As nearby grey values have a slightly higher probability of being similar, sophisticated entropy coders can make use of this fact to improve the compression ratio. However, when only a part of the image was transmitted, grey values are known only in some image areas. While the image is reconstructed well in those regions, the remainder of the image is basically unknown, see the bottom left image in Figure 1.

At this point, the basic idea how to include a progressive mode into R-EED becomes clear: Instead of storing grey values row-wise, they should be stored such that the points are distributed over the complete image domain no matter how much of the compressed image has been transmitted. How to chose a good ordering is less obvious, though.

A tempting idea would be to always chose the point from the inpainting mask that improves the final reconstruction quality most. Even though this would yield very good results, the overhead necessary to store the ordering of the points is much too expensive. Even if only 720 grey values are stored, which corresponds to a compression ratio of $150 : 1$ for the image "trui" (see first image in Figure 1), storing an ordering requires $\log_2(720!) \cdot \frac{1}{8} > 725$ bytes. However, the size of the compressed image is only 437 bytes. With more points, the overhead is even worse.

Thus, we focus on methods which do not require additional information. Instead, a good ordering is estimated from the inpainting mask and the used tree structure, as this data is stored anyway. To achieve this goal, we remember that the inpainting mask is created in a specific way: Each part of the image should (approximately) contain the fraction of points necessary to obtain a comparably good reconstruction. Thus, it appears natural to preserve these fractions as good as possible if only a part of the compressed file is available.

Therefore, we traverse the tree containing the splitting decisions once for each point to be saved. At every node, we continue with the child corresponding to the most under-represented image part, i.e. to the image part for which the smallest fraction of the points to be stored has been saved so far. Finally, we arrive at a node which has no children, or only children with completely saved mask points. Then, we store an unsaved mask point in the image part corresponding to this tree node. When loading an image, the values stored in the file are put into the positions indicated by the inpainting mask in the same order. Points for which no grey value is available yet are assumed to be unknown, i.e. these points are removed from the inpainting mask before inpainting. In our experiments, we denote this progressive mode as "R-EED-P1".

We have tried several other approaches, e.g. a pseudo-random permutation of the mask points, or a level-wise storage of the points depending only on their depth in the tree node. Due to space restrictions, and to avoid possible confusions, we refrain from explaining those inferior approaches in detail.

The performance of the first approach is still sub-optimal if only a very small part of the file is known. This is due to the overhead necessary to store file header and Huffman tree, and the fact that inpainting a complex image from a very small number of points typically yields very bad results (see fourth image of the first row in Figure 1).

To diminish these problems, we propose a second progressive mode which builds upon the first. In this mode, which we call "R-EED-P2", the grey values of the points in the inpainting mask are split in two parts. The first part consists of the grey values f_i divided by $s := [\sqrt{q}]$ using integer division, where $[\cdot]$ denotes rounding to the closest integer and q is the number of quantisation levels used in an image. In the second part, the remainder $r := f_i \mod s$ is stored.

Saving grey values in two parts has two advantages: First of all, the resulting two Huffman trees are smaller than the single Huffman tree used before, which reduces the overhead of the file header. Thus, less information is necessary to obtain the first grey values and a first rough approximation, as visible in the graph shown in Figure 2. The smaller Huffman trees often result in an overall slightly less efficient entropy coding, but the total file size can also be smaller, as for the image "walter" and a compression ratio of $78 : 1$. The second Huffman tree is not stored in the file header but directly before it is used for the first time. The second advantage is that less bits are initially necessary for each point. Thus, information at more points is available when inpainting after a small part of the file is known. Even though these grey values are quantised coarser, they still allow a much better reconstruction than a significantly smaller number of more accurate points. This is shown in the last two images in the first row in Figure 1.

After a certain number of coarse grey values are known, the shapes appearing in the image can already be reconstructed rather accurately. However, the specific grey values are still quite imprecise due to the strong quantisation performed in the first step. At this point, the reconstruction benefits more from more accurate grey values than from additional inaccurate points. Thus, it is not a good idea to store *all* points inaccurately before saving the second part of each grey value. Due to this reason, our second progressive mode actually performs three passes: In the first pass, a certain percentage of the brightness values is stored inaccurately, as described above. Afterwards, the additional information necessary to obtain the actual grey values for the points stored in the first pass is saved. In the final pass, the grey values missing so far are stored as accurate as without progressive modes. Storing the actual grey values directly would require a novel Huffman tree, resulting in an additional overhead. Instead, we store each grey value in two parts using the same Huffman trees as before.

IV. Experiments

To judge the performance of the proposed progressive modes, we evaluate the results obtained with different amount of transmitted data. For this, we employ the *peak signal to noise ratio* (PSNR). The underlying test images are "trui" and "walter"; see Figure 1. For our tests, we used compression ratios ranging from $19 : 1$ to $80 : 1$ and partial files ranging from the smallest possible fraction for which a reconstruction is possible up to the complete files. The images for JPEG and JPEG 2000 have been created using the Linux tool "convert" (version 6.8.3-6 2013-03-04 Q16) with default parameters. The resulting PSNRs for different parts of the files are shown in Figure 2. Note that the JPEG and JPEG 2000 files are slightly larger than those created with our algorithm, and that the number of bytes that must be transmitted before the first estimation is possible differs due to different header sizes. JPEG 2000 has a disadvantage with this respect, as its header is comparably large.

For very small fractions of a stored image compressed with a small compression ratio, the progressive mode of JPEG and JPEG 2000 typically yield slightly better reconstructions than

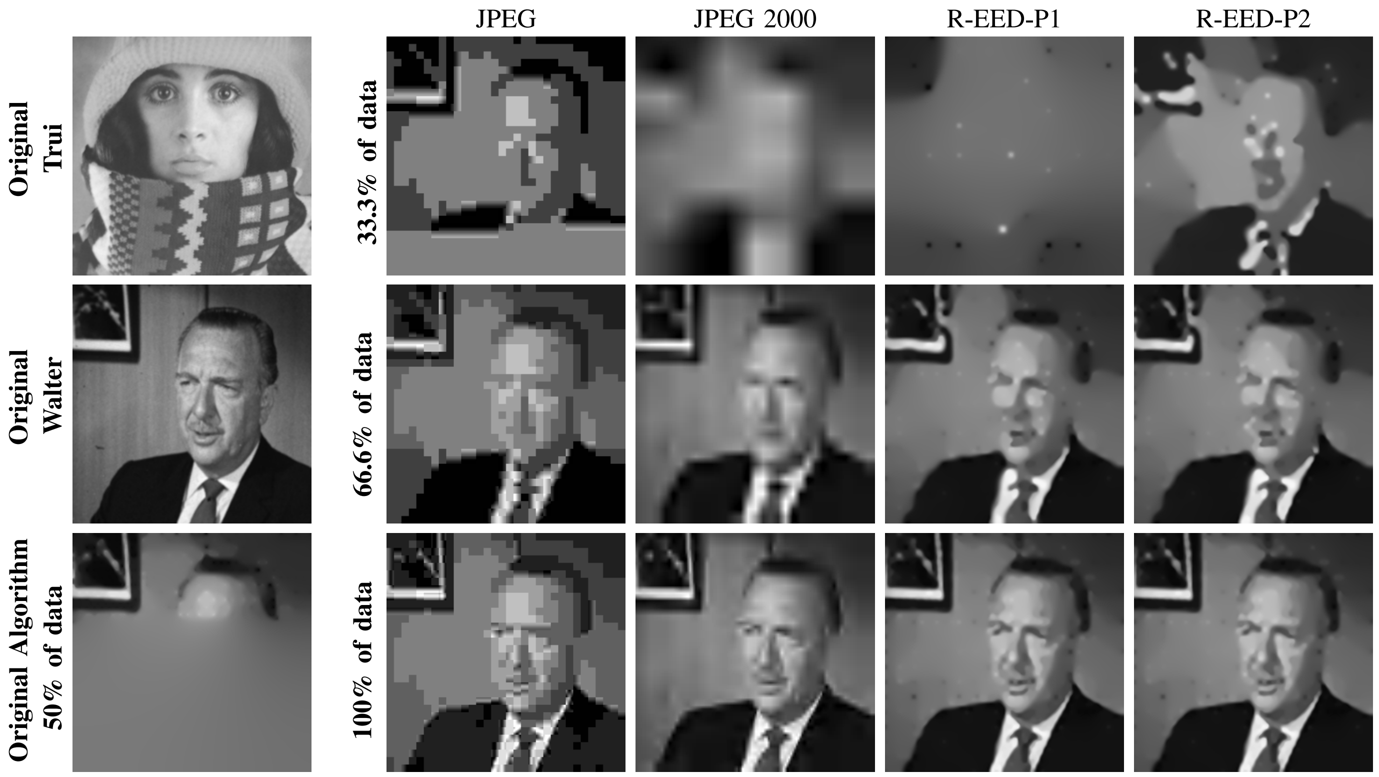

Fig. 1. **First column**: Input images “trui” and “walter” used in the experiments, and image reconstructed from half the data without using the progressive mode proposed in this paper. **Remaining columns**: Images reconstructed from partial files containing the image “walter”, compressed using JPEG, JPEG 2000, and the proposed methods, respectively. All compressed images have a compression ratio of $\mathbf{80:1}$.

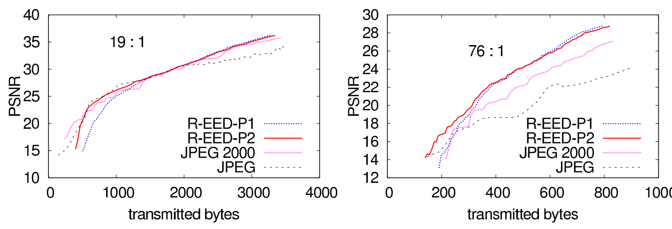

Fig. 2. These graphs show the performance of JPEG, JPEG 2000, and of our progressive modes for the image “trui” (size: 256×256, see Figure 1(a)) and compression ratios around $\mathbf{19:1}$ (left) and $\mathbf{76:1}$ (right).

our approaches. For medium and high compression ratios, however, our progressive modes clearly surpass that of JPEG and JPEG 2000. For a visual evaluation, we show example results for the image “walter” in Figure 1.

Conclusion

In this paper, we have explained how to incorporate progressive modes into PDE-based image compression algorithms. More precisely, we proposed two progressive modes, and demonstrated that they go head to head with existing modes for JPEG and JPEG 2000. This is an important next step towards making lossy PDE-based image compression algorithms more attractive for real-world applications.

Our future work includes investigating more advanced entropy coding schemes for our progressive modes. Another line of research is to transfer our results to other PDE-based compression approaches for 2-D and 3-D data.

References

[1] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still Image Data Compression Standard.* New York: Springer, 1992.

[2] D. S. Taubman and M. W. Marcellin, Eds., *JPEG 2000: Image Compression Fundamentals, Standards and Practice.* Boston: Kluwer, 2002.

[3] S. Carlsson, “Sketch based coding of grey level images,” *Signal Processing*, vol. 15, pp. 57–83, 1988.

[4] J. H. Elder, “Are edges incomplete?” *International Journal of Computer Vision*, vol. 34, no. 2/3, pp. 97–122, 1999.

[5] M. Lillholm, M. Nielsen, and L. D. Griffin, “Feature-based image analysis,” *International Journal of Computer Vision*, vol. 52, no. 2/3, pp. 73–95, 2003.

[6] M. Mainberger, S. Hoffmann, J. Weickert, C. H. Tang, D. Johannsen, F. Neumann, and B. Doerr, “Optimising spatial and tonal data for homogeneous diffusion inpainting.” in *Scale Space and Variational Methods in Computer Vision*, ser. Lecture Notes in Computer Science, A. Bruckstein, B. ter Haar Romeny, A. Bronstein, and M. Bronstein, Eds. Berlin: Springer, Jun. 2011, vol. 6667, pp. 26–37.

[7] J. Gautier, O. L. Meur, and C. Guillemot, “Efficient depth map compression based on lossless edge coding and diffusion,” in *Picture Coding Symposium*, Kraków, Poland, May 2012, pp. 81–84.

[8] T. F. Chan and H. M. Zhou, “Total variation improved wavelet thresholding in image compression,” in *Proc. Seventh International Conference on Image Processing*, vol. II, Vancouver, Canada, Sep. 2000, pp. 391–394.

[9] A. Solé, V. Caselles, G. Sapiro, and F. Arandiga, “Morse description and geometric encoding of digital elevation maps,” *IEEE Transactions on Image Processing*, vol. 13, no. 9, pp. 1245–1262, Sep. 2004.

[10] D. Liu, X. Sun, and F. Wu, “Edge-based inpainting and texture synthesis for image compression,” in *Proc. 2007 International Conference on Multimedia and Expo*, Beijing, China, Jul. 2007, pp. 1443–1446.

[11] I. Galić, J. Weickert, M. Welk, A. Bruhn, A. Belyaev, and H.-P. Seidel, "Image compression with anisotropic diffusion," *Journal of Mathematical Imaging and Vision*, vol. 31, no. 2–3, pp. 255–269, Jul. 2008.

[12] J. Weickert, "Theoretical foundations of anisotropic diffusion in image processing," *Computing Supplement*, vol. 11, pp. 221–236, 1996.

[13] C. Schmaltz, J. Weickert, and A. Bruhn, "Beating the quality of JPEG 2000 with anisotropic diffusion," in *Pattern Recognition*, ser. Lecture Notes in Computer Science, J. Denzler, G. Notni, and H. Süße, Eds., vol. 5748. Berlin: Springer, 2009, pp. 452–461.

[14] C. Schmaltz, P. Peter, M. Mainberger, F. Ebel, J. Weickert, and A. Bruhn, "Understanding, optimising, and extending data compression with anisotropic diffusion," Department of Mathematics, Saarland University, Saarbrücken, Germany, Tech. Rep. 329, 2013.