PART III
Efficient Numerics

Contents

M I A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

Successive Overrelaxation

- *Known:* So far we have used the Jacobi method or its improved variant the Gauß-Seidel method to solve the linear system of equation $A\mathbf{x} = \mathbf{b}$.
- *Jacobi Method:* The iteration step for the **Jacobi method** is given by

$$\mathbf{x}^{k+1} = D^{-1}(\,\mathbf{b} + (L+U)\,\mathbf{x}^k\,) \quad \Leftrightarrow \quad x_i^{k+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij} x_j^k - \sum_{j>i} a_{ij} x_j^k\right).$$

- *Gauß-Seidel Method:* The iteration step for the **Gauß-Seidel method** reads

$$\mathbf{x}^{k+1} = (D-L)^{-1}(\mathbf{b} + U\,\mathbf{x}^k) \quad \Leftrightarrow \quad x_i^{k+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij} x_j^{k+1} - \sum_{j>i} a_{ij} x_j^k\right).$$

  Here, D is the diagonal part, L is the lower triangular part and U ist the upper triangular part of the system matrix A.

M I A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

How Can We Accelerate the Gauß-Seidel Method?

- *Idea:* **Pointwise extrapolate** the result of the Gauß-Seidel method
- *Successive Overrelaxation Method (SOR):* Given the Gauß-Seidel result $\bar{x}_i^{k+1}$ at pixel i and iteration $k+1$, the **SOR method** proceeds (*Young 1971*)

$$x_i^{k+1} = (1-\omega)\,x_i^k + \omega\,\bar{x}_i^{k+1} = (1-\omega)\,x_i^k + \omega\,\underbrace{\frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij} x_j^{k+1} - \sum_{j>i} a_{ij} x_j^k\right)}_{\text{Gauß-Seidel result}}$$

  with **overrelaxation parameter** $\omega \in (0,2)$. In matrix notation this reads

$$\mathbf{x}^{k+1} = (D + \omega L)^{-1}(\omega\mathbf{b} - (\omega U + (1-\omega)D)\,\mathbf{x}^k)) \neq (1-\omega)\,\mathbf{x}^k + \omega\bar{\mathbf{x}}^k\,.$$

- *Properties:* For symmetric and positive definite system matrices the SOR method
  - converges if the overrelaxation parameter ω is chosen in the interval $(0,2)$
  - is 1-2 orders of magnitude more efficient than the Gauß-Seidel method

M I A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

How Does the SOR Method looks like for the Horn and Schunck Method?

- *Equation System:* For the method of Horn and Schunck the associated linear system of equations is given by (cf. PART I)

$$0 = [J_{11}]_{i,j}\,u_{i,j} + [J_{12}]_{i,j}\,v_{i,j} + [J_{13}]_{i,j} - \alpha \sum_{l\in x,y} \sum_{(\tilde{i},\tilde{j})\in\mathcal{N}_l(i,j)} \frac{u_{\tilde{i},\tilde{j}} - u_{i,j}}{h_l^2}$$
$$0 = [J_{12}]_{i,j}\,u_{i,j} + [J_{22}]_{i,j}\,v_{i,j} + [J_{23}]_{i,j} - \alpha \sum_{l\in x,y} \sum_{(\tilde{i},\tilde{j})\in\mathcal{N}_l(i,j)} \frac{v_{\tilde{i},\tilde{j}} - v_{i,j}}{h_l^2}$$

  for $i = 1, ..., N$ and $j = 1, ..., M$.

- *Example:* The corresponding SOR iteration step then reads

$$u_{i,j}^{k+1} = \boxed{(1-\omega)}\,u_{i,j}^k + \boxed{\omega}\,\frac{\left(-[J_{13}]_{i,j} - \left([J_{12}]_{i,j}\,v_{i,j}^k - \alpha \sum\limits_{l\in x,y} \sum\limits_{\mathcal{N}_l^-(i,j)} \frac{1}{h_l^2}\,u_{\tilde{i},\tilde{j}}^{k+1} - \alpha \sum\limits_{l\in x,y} \sum\limits_{\mathcal{N}_l^+(i,j)} \frac{1}{h_l^2}\,u_{\tilde{i},\tilde{j}}^k\right)\right)}{[J_{11}]_{i,j} + \alpha \sum\limits_{l\in x,y} \sum\limits_{(\tilde{i},\tilde{j})\in\mathcal{N}_l(i,j)} \frac{1}{h_l^2}},$$

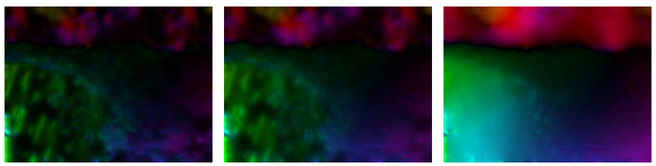
$$v_{i,j}^{k+1} = \boxed{(1-\omega)}\,v_{i,j}^k + \boxed{\omega}\,\frac{\left(-[J_{23}]_{i,j} - \left([J_{12}]_{i,j}\,u_{i,j}^{k+1} - \alpha \sum\limits_{l\in x,y} \sum\limits_{\mathcal{N}_l^-(i,j)} \frac{1}{h_l^2}\,v_{\tilde{i},\tilde{j}}^{k+1} - \alpha \sum\limits_{l\in x,y} \sum\limits_{\mathcal{N}_l^+(i,j)} \frac{1}{h_l^2}\,v_{\tilde{i},\tilde{j}}^k\right)\right)}{[J_{22}]_{i,j} + \alpha \sum\limits_{l\in x,y} \sum\limits_{(\tilde{i},\tilde{j})\in\mathcal{N}_l(i,j)} \frac{1}{h_l^2}}.$$

Convergence Comparisons for Different Iterative Methods

- *Qualitative Comparison:* Horn and Schunck method after 100 Iterations

Solver	AAE
Jacobi method	29.58°
Gauß-Seidel method	22.63°
Successive Overrelaxation ($\omega = 1.96$)	7.18°

- *Visual Comparison:* Horn and Schunck method after 100 Iterations



Results for the Yosemite Sequence with clouds (L. Quam) using 100 solver iterations. **(a) Left:** Jacobi method. **(b) Center:** Gauß-Seidel method. **(c) Right:** Successive Overrelaxation method ($\omega = 1.96$).

Nested Fixed Point Iteration

How to Apply Linear Solvers to Nonlinear Systems?

- *Idea:* Derive a **Quasi-Newton scheme** for $A(\mathbf{x}) = \mathbf{b}$ by approximating the Jacobian $B(\mathbf{x})$ of the nonlinear operator $A(\mathbf{x})$ via the linear decomposition

$$A(\mathbf{x}) = B(\mathbf{x})\,\mathbf{x} + \mathbf{c}(\mathbf{x})$$

with matrix $B(\mathbf{x})$ being symmetric and positive definite for any values of $\mathbf{x}$.

- *Consequence:* Original nonlinear problem solved by determining the fixed point of the **series of linear problems** (*Fučik et al. 1973, Vogel/Oman 1996, Axelson 1997*)

$$\underbrace{B(\mathbf{x}^k)}_{A^k}\mathbf{x}^{k+1} = \underbrace{\mathbf{b} - \mathbf{c}(\mathbf{x}^k)}_{\mathbf{b}^k} \quad \Leftrightarrow \quad \mathbf{x}^{k+1} = \mathbf{x}^k - B^{-1}(\mathbf{x}^k)\big(A(\mathbf{x}^k) - \mathbf{b}\big)$$

Terms involving nonlinear expressions, i.e. A^k and $\mathbf{b}^k$ are computed using the solution $\mathbf{x}^k$ from the old iteration k → **lagged nonlinearity method**.

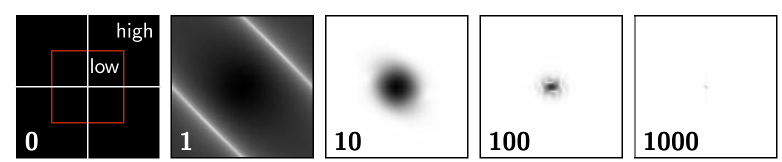
Nested Iterations

- *Nonlinear Case:* Each linear problem requires a solver → 2 nested iterations
  - **outer loop:** lagged nonlinearity FP iteration (remove nonlinearity)
  - **solver loop:** solver FP iteration (solve linear system)
- *Nonlinear Case with Warping:* Multiple nonlinear problems → 3 nested iterations
  - **outer loop:** warping FP iteration (remove implicit nonlinearity)
  - **inner loop:** lagged nonlinearity FP iteration (remove nonlinearity)
  - **solver loop:** solver FP iteration (solve linear system)
- *Attention:* Very inexact solution of solver loop sufficient (*Vogel/Oman 1996*)
  - fast update of nonlinear expressions essential for performance
  - applying only a few solver iterations yields fastest convergence

Basic Linear Multigrid

How Can We Solve Linear Systems of Equations Even More Efficiently?

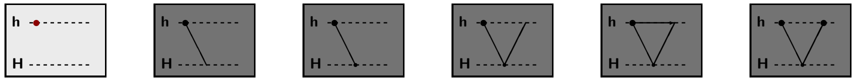
- *Observation:* Slow convergence of iterative solvers (Jacobi, Gauß Seidel, SOR) already after a few iterations. What is the reason of this behavior?
  - logarithmic error spectrum reveals slow decrease of lower frequency parts (→ only efficient damping of **higher error frequency parts**)



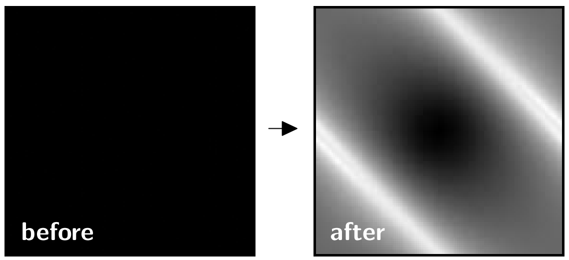
- *Sophisticated Idea*: Transfer and compute **error(!)** on coarser grids *(Brand 1977, Hackbusch 1985)*
  - low frequencies reappear as higher frequencies (→ also efficient damping of **lower error frequency parts**)

M I A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

Basic Linear Multigrid – The Two–Grid Cycle

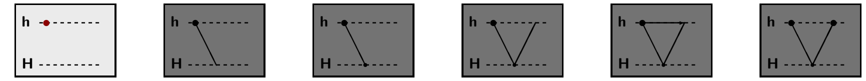


- *Step 1:* Presmoothing Relaxation
  - smoothing of higher error frequencies
    $\rightarrow$ application of n_1 solver iterations to $A^h \mathbf{x}^h = \mathbf{b}^h$
  - logarithmic error spectrum shows decrease of higher frequency parts



M I A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

Basic Linear Multigrid – The Two–Grid Cycle



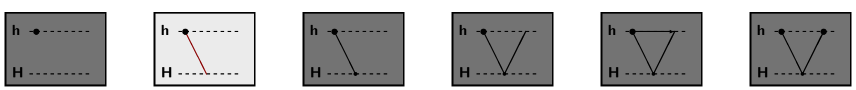
- *Question:* How shall we proceed?
  - error $\mathbf{e}^h = \mathbf{x}^h - \tilde{\mathbf{x}}^h$ **cannot** be computed directly
  - residual $\mathbf{r}^h = \mathbf{b}^h - A^h \tilde{\mathbf{x}}^h$ **can** be computed directly
  - linearity of matrix A^h yields the **residual equation**

$$\begin{aligned} A^h \mathbf{e}^h &= A^h(\mathbf{x}^h - \tilde{\mathbf{x}}^h) \\ &= A^h \mathbf{x}^h - A^h \tilde{\mathbf{x}}^h \\ &= \mathbf{b}^h - A^h \tilde{\mathbf{x}}^h \qquad = \mathbf{r}^h \, . \end{aligned}$$

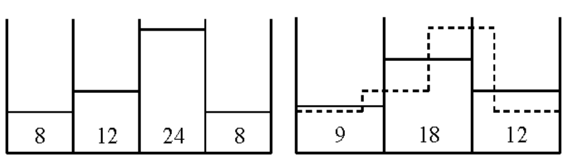
  - solving this **linear** system of equations $A^h \mathbf{e}^h = \mathbf{r}^h$ allows the desired correction of the approximate solution $\tilde{\mathbf{x}}^h$ by its error $\mathbf{e}^h$

M I A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

Basic Linear Multigrid – The Two–Grid Cycle



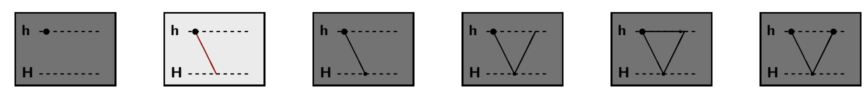
- *Step 2:* Restriction
  - transfer residual equation $A^h \mathbf{e}^h = \mathbf{r}^h$ to coarser grid $\rightarrow A^H \mathbf{x}^H = \mathbf{b}^H$
  - decision 1: choice of **coarse cell/grid size** H
    e.g. halving the pixel number yields doubling of the cell/grid size
  - decision 2: choice of **restriction operator** $R^{h \rightarrow H}$
    e.g. area-based averaging over $h \times h$ pixels



  - coarse grid RHS is then obtained by restriction : $\mathbf{b}^H = R^{h \rightarrow H} \mathbf{r}^h$

M I A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

Basic Linear Multigrid – The Two–Grid Cycle



- *Step 2:* Restriction (continued)
  - transfer residual equation $A^h \mathbf{e}^h = \mathbf{r}^h$ to coarser grid $\rightarrow A^H \mathbf{x}^H = \mathbf{b}^H$
  - decision 3: choice of **coarse grid matrix** A^H of operator A^h
    e.g. by Discretisation Coarse Grid Approximation (DCA)
- *Discretisation Coarse Grid Approximation (DCA)*
  - rediscretisation of Euler–Lagrange equations (restriction of motion tensors)

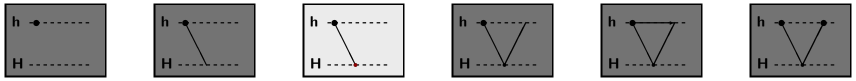
$$[J_{nm}]^H = R^{h \rightarrow H} [J_{nm}]^h \quad \text{for} \quad n, m \in \{1, 2, 3\}$$

  - substitution of fine grid size h by coarse grid size H (smoothness term)

$$\sum_{l \in x,y} \sum_{(\tilde{i},\tilde{j}) \in \mathcal{N}_l(i,j)} \frac{u^h_{\tilde{i},\tilde{j}} - u^h_{i,j}}{h_l^2} \quad \rightarrow \quad \sum_{l \in x,y} \sum_{(\tilde{i},\tilde{j}) \in \mathcal{N}_l(i,j)} \frac{{u^H}_{\tilde{i},\tilde{j}} - {u^H}_{i,j}}{H_l^2}$$

Basic Linear Multigrid (6)

Basic Linear Multigrid – The Two–Grid Cycle



- *Step 3:* Coarse Grid Computation
  - solve the restricted linear system of equations $A^H \mathbf{x}^H = \mathbf{b}^H$ given by

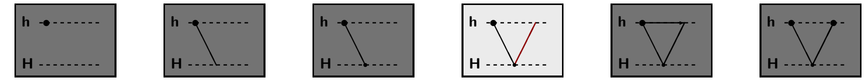
$$0 \;=\; [J_{11}]^H{}_{i,j}\, u^H{}_{i,j} + [J_{12}]^H{}_{i,j}\, v^H{}_{i,j} + [b_1]^H_{i,j} - \alpha \sum_{l \in x,y} \sum_{(\tilde{i},\tilde{j}) \in \mathcal{N}_l(i,j)} \frac{u^H{}_{\tilde{i},\tilde{j}} - u^H{}_{i,j}}{H_l^2}$$

$$0 \;=\; [J_{12}]^H{}_{i,j}\, u^H{}_{i,j} + [J_{22}]^H{}_{i,j}\, v^H{}_{i,j} + [b_2]^H_{i,j} - \alpha \sum_{l \in x,y} \sum_{(\tilde{i},\tilde{j}) \in \mathcal{N}_l(i,j)} \frac{v^H{}_{\tilde{i},\tilde{j}} - v^H{}_{i,j}}{H_l^2}$$

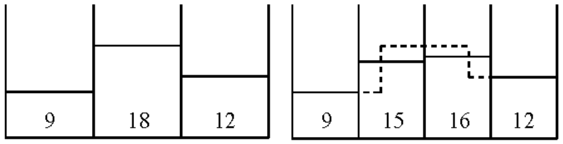
    for $i = 1, ..., N^H$ and $j = 1, ..., M^H$ on the coarse grid.
  - if pixel number is small, direct computation via Gaussian elimination
  - else iterative computation, e.g. by using the SOR method

Basic Linear Multigrid (7)

Basic Linear Multigrid – The Two–Grid Cycle



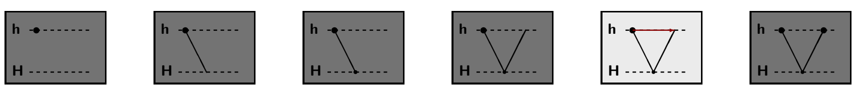
- *Step 4:* Prolongation
  - decision 4: choice of **prolongation operator** $P^{H \to h}$
    e.g. area-based interpolation over $h \times h$ pixels



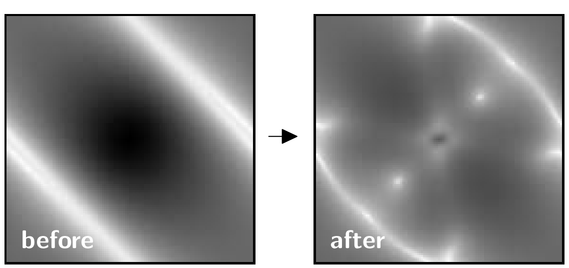
  - transfer result from coarse grid to fine grid : $\mathbf{e}^h = P^{H \to h} \mathbf{x}^H$

Basic Linear Multigrid (8)

Basic Linear Multigrid – The Two–Grid Cycle

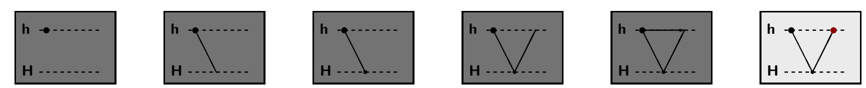


- *Step 5:* Correction From Coarse Grid
  - correction of approximation from presmoothing relaxation : $\tilde{\mathbf{x}}^h_{\mathbf{new}} = \tilde{\mathbf{x}}^h + \mathbf{e}^h$
  - logarithmic error spectrum shows decrease of lower frequency parts
  - however, prolongation of error introduces new high frequency parts

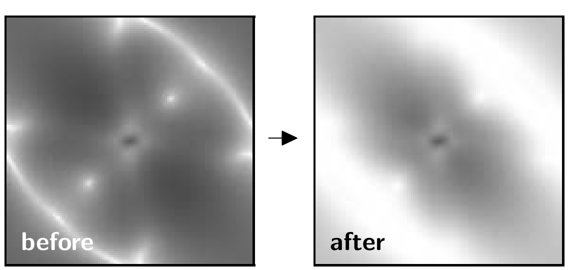


Basic Linear Multigrid (9)

Basic Linear Multigrid – The Two–Grid Cycle

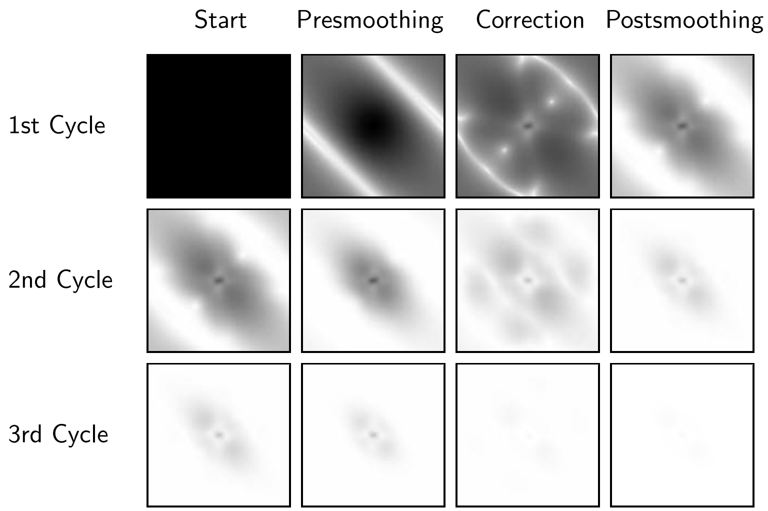


- *Step 6:* Postsmoothing Relaxation
  - smoothing of higher error frequencies introduced by interpolation
    $\rightarrow$ Application of n_2 solver iterations to $A^h \mathbf{x}^h = \mathbf{b}^h$
  - logarithmic error spectrum shows decrease of higher frequency parts



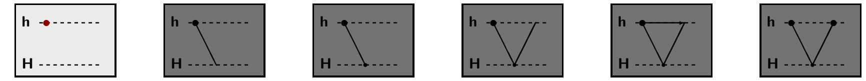
Basic Linear Multigrid (10)

Efficient Error Reduction Through Three Cycles

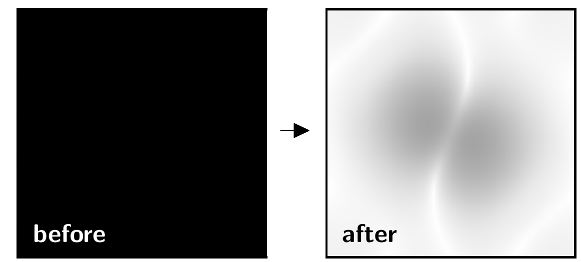


Basic Nonlinear Multigrid (1)

Basic Nonlinear Multigrid – The Two–Grid Cycle

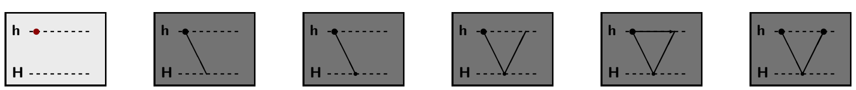


- *Step 1:* Presmoothing Relaxation
  - smoothing of higher error frequencies
    $\rightarrow$ application of n_1 **nonlinear** solver iterations to $A^h(\mathbf{x}^h) = \mathbf{b}^h$
  - logarithmic error spectrum shows decrease of higher frequency parts



Basic Nonlinear Multigrid (2)

Basic Nonlinear Multigrid – The Two–Grid Cycle



- *Question:* How shall we proceed?
  - error $\mathbf{e}^h = \mathbf{x}^h - \tilde{\mathbf{x}}^h$ **cannot** be computed directly
  - residual $\mathbf{r}^h = \mathbf{b}^h - A^h(\tilde{\mathbf{x}}^h)$ **can** be computed directly
  - nonlinearity of A^h does not yield residual equation

$$A^h(\mathbf{e}^h) = A^h(\mathbf{x}^h - \tilde{\mathbf{x}}^h) \neq A^h(\mathbf{x}^h) - A^h(\tilde{\mathbf{x}}^h) = \mathbf{b}^h - A^h(\tilde{\mathbf{x}}^h) = \mathbf{r}^h$$

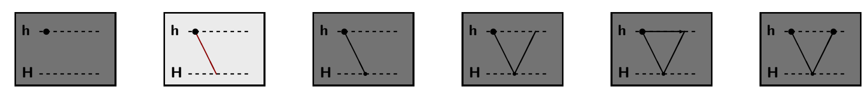
- *New Idea:* Full approximation scheme (FAS)
  - but implicit relation is given by

$$A^h(\mathbf{x}^h) - A^h(\tilde{\mathbf{x}}^h) = \boxed{A^h(\tilde{\mathbf{x}}^h + \mathbf{e}^h) - A^h(\tilde{\mathbf{x}}^h)} = \mathbf{b}^h - A^h(\tilde{\mathbf{x}}^h) = \boxed{\mathbf{r}^h}$$

  - this yields the **full approximation** : $A^h(\mathbf{e}^h + \tilde{\mathbf{x}}^h) = \mathbf{r}^h + A^h(\tilde{\mathbf{x}}^h)$
  - solving this **nonlinear** equation system allows desired correction of $\tilde{\mathbf{x}}^h$ by $\mathbf{e}^h$

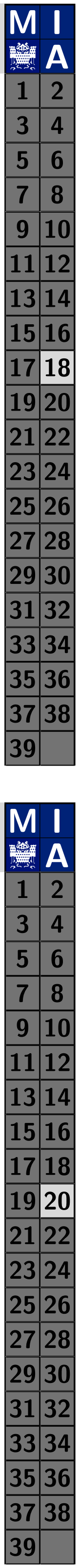
Basic Nonlinear Multigrid (3)

Basic Nonlinear Multigrid – The Two–Grid Cycle

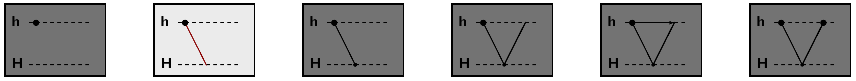


- *Step 2:* Restriction
  - transfer $A^h(\mathbf{e}^h + \tilde{\mathbf{x}}^h) = \mathbf{r}^h + A^h(\tilde{\mathbf{x}}^h)$ to coarser grid $\rightarrow A^H(\mathbf{x}^H) = \mathbf{b}^H$
  - decision 1: choice of **coarse cell/grid size** H
    e.g. halving the pixel number yields doubling of the cell/grid size
  - decision 2: choice of **restriction operator** $R^{h \rightarrow H}$
    e.g. area-based averaging over $h \times h$ pixels
  - decision 3: choice of **coarse grid version** A^H of operator A^h
    e.g. by Discretisation Coarse Grid Approximation (DCA)

How is the actual coarse grid right hand side obtained ?



Basic Nonlinear Multigrid – The Two–Grid Cycle



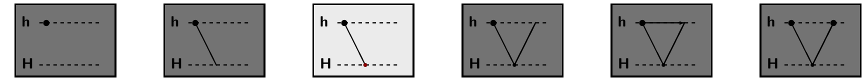
- *Step 2:* Restriction (continued)
  - transfer $A^h(\mathbf{e}^h + \tilde{\mathbf{x}}^h) = \mathbf{r}^h + A^h(\tilde{\mathbf{x}}^h)$ to coarser grid $\to$ $A^H(\mathbf{x}^H) = \mathbf{b}^H$

$$\downarrow \qquad \searrow$$

$$\begin{aligned} \mathbf{b}^H &= R^{h\to H}\mathbf{r}^h + A^H(R^{h\to H}\tilde{\mathbf{x}}^h) \\ &= R^{h\to H}\big(\mathbf{b}^h - A^h(\tilde{\mathbf{x}}^h)\big) + A^H(R^{h\to H}\tilde{\mathbf{x}}^h) \\ &= R^{h\to H}\mathbf{b}^h - \underbrace{\Big(R^{h\to H}A^h(\tilde{\mathbf{x}}^h) - A^H(R^{h\to H}\tilde{\mathbf{x}}^h)\Big)}_{\text{modification of right hand side } \tilde{\mathbf{b}}^H} . \end{aligned}$$

  - thus one actually solves a coarse grid variant of the fine grid problem with **modified right hand side**: $A^h(\mathbf{x}^h) = \mathbf{b}^h \to A^H(\mathbf{x}^H) = R^{h\to H}\mathbf{b}^h - \tilde{\mathbf{b}}^H$.

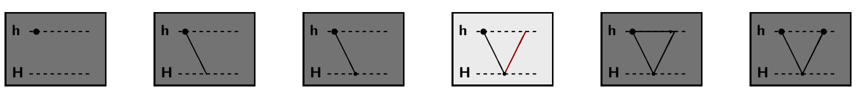
Basic Nonlinear Multigrid – The Two–Grid Cycle



- *Step 3:* Coarse Grid Computation
  - solve the restricted **nonlinear system** of equations $A^H(\mathbf{x}^H) = \mathbf{b}^H$
  - due to $A^H(\mathbf{x}^H) = A^H(\mathbf{e}^H + \tilde{\mathbf{x}}^H) = \mathbf{b}^H$ extract the **coarse grid error**

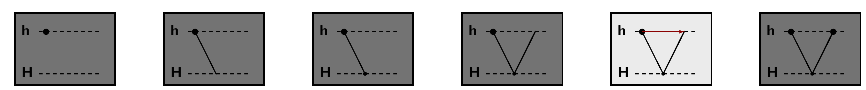
$$\begin{aligned} \mathbf{e}^H &= \mathbf{x}^H - \tilde{\mathbf{x}}^H \\ &= \mathbf{x}^H - R^{h\to H}\tilde{\mathbf{x}}^h . \end{aligned}$$

Basic Nonlinear Multigrid – The Two–Grid Cycle

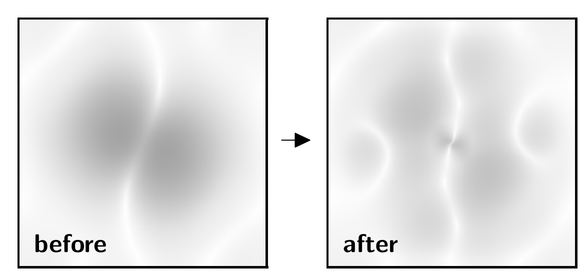


- *Step 4:* Prologation
  - decision 4: choice of **prolongation operator** $P^{H\to h}$
    e.g. area-based interpolation over $h \times h$ pixels
  - transfer result from coarse grid to fine grid : $\mathbf{e}^h = P^{H\to h}\mathbf{e}^H$

Basic Nonlinear Multigrid – The Two–Grid Cycle

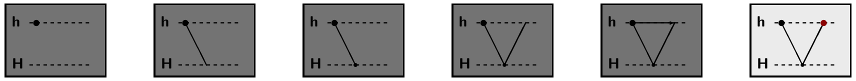


- *Step 5:* Correction From Coarse Grid
  - correction of approximation from presmoothing relaxation : $\tilde{\mathbf{x}}^h_{\text{new}} = \tilde{\mathbf{x}}^h + \mathbf{e}^h$
  - logarithmic error spectrum shows decrease of lower frequency parts
  - however, prolongation of error introduces new high frequency parts

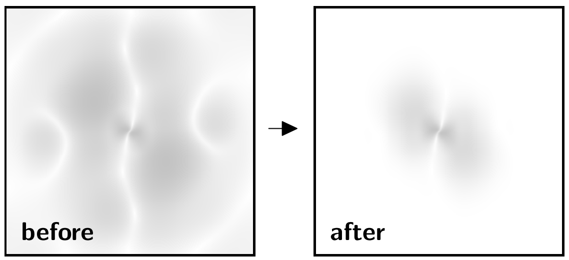


Basic Nonlinear Multigrid (8)

Basic Nonlinear Multigrid – The Two–Grid Cycle

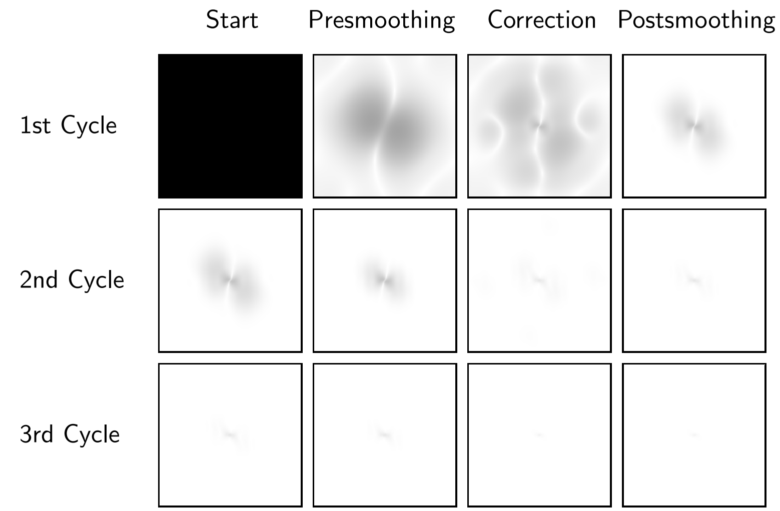


- *Step 6:* Postsmoothing Relaxation
  - smoothing of higher error frequencies introduced by interpolation → application of n_2 **nonlinear** solver iterations to $A^h(\mathbf{x}^h) = \mathbf{b}^h$
  - logarithmic error spectrum shows decrease of higher frequency parts



Basic Nonlinear Multigrid (9)

Efficient Error Reduction Through Three Cycles

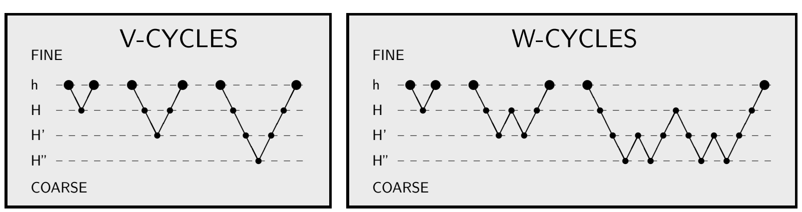


Advanced Multigrid Strategies (1)

Advanced Multigrid Strategies

How Can We Improve the Convergence of Multigrid Methods Even Further?

- *Idea 1:* Hierarchical application of the two–grid correction cycle
- *Example:* **one** or **two** recursive calls per level (→ **V–cycle**, **W–cycle**)

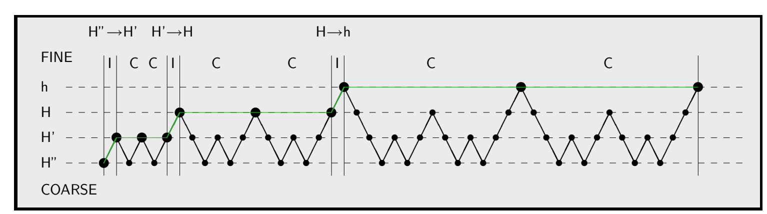


- *Idea 2:* Additionally start with better initialisation
- *Example:* embed V–/W–cycles in hierarchical initialisation (→ **Full Multigrid**)

Advanced Multigrid Strategies (2)

The Full Multigrid Strategy

- *Hierarchical Initialisation:* **Coarse-to-Fine Approach**
  - start with coarse version of original problem
  - refine problem step by step
  - use coarse solution as initial guess on next finer grid
- *At Each Level:* **Correcting Multigrid Solver**
  - coarse grid corrections → error ex-/implicitly computable via V-/W-cycles



Further Multigrid Developments in the Variational Optical Flow Literature

- *Coarse Grid Representation*
  - problem approximation (*Terzopoulos TPAMI 1986*) $\longrightarrow$ cheap but inaccurate
  - Galerkin approximation (*El Kalmoun/Rüde VMV 2003)* $\longrightarrow$ optimal but expensive
  - graph-based approximation (*Ghosal/Vaněk TPAMI 1996*) $\longrightarrow$ algebraic multigrid
- *Intergrid Transfer Operators*
  - matrix-dependent operators (*Köstler et al. TR 2005*) $\longrightarrow$ discont. coefficients
- *Basic Iterative Solvers*
  - point-/line-coupled solvers (*Bruhn et al. SSVM 2005*) $\longrightarrow$ anisotropic problems
  - incomplete LU factorization (*Köstler et al. TR 2005*) $\longrightarrow$ broad applicability
  - re-iterant recombination (*Köstler et al. TR 2005*) $\longrightarrow$ improved convergence
- *Extensions to 3-D Multigrid*
  - cardiac motion analysis (*Zini et al. TIP 1997, El Kamoun et al. IMAVIS 2007*)

Comparison of Numerical Solvers

- *Testbed for Various Prototypes*
  - standard desktop PC with 3.06 GHz Pentium4 CPU
  - C/C++ implementation
  - image size 160 $\times$ 120
  - stopping criterion : relative error $e_{\mathbf{rel}} := ||\tilde{x} - x||_2/||x||_2$ of 10^{-2}
- *Linear Case:* Horn and Schunck (*Bruhn et al. SSVM 2005*)

Solver	Iterations	Time [s]	FPS [s^{-1}]	Speedup
Mod. Explicit Scheme	4425	3.509	0.285	1
Gauß-Seidel (CPR)	2193	1.152	0.868	3
SOR	82	0.052	19.233	67
Full Multigrid	**1**	**0.016**	**62.790**	**220**

M I A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

Comparison of Numerical Solvers

- *Linear Case:* Nagel and Enkelmann (directional smoothness) (*Bruhn et al. SSVM 2005*)

Solver	Iterations	Time [s]	FPS [s^{-1}]	Speedup
Mod. Explicit Scheme	36433	47.087	0.021	1
Gauß-Seidel (ALR)	607	3.608	0.277	13
SOR	202	0.212	4.417	224
Full Multigrid	**1**	**0.171**	**5.882**	**275**

- *Nonlinear Case:* Nonquadratic Smoothness Term (TV norm) (*Bruhn et al. SSVM 2005*)

Solver	Iterations	Time [s]	FPS [s^{-1}]	Speedup
Mod. Explicit Scheme	10633	30.492	0.033	1
Gauß-Seidel (CPR)	2679	6.911	0.145	4
SOR	17/5	0.174	5.748	174
Full Multigrid	**1**	**0.082**	**12.172**	**372**

M I A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

Multigrid Speedups

- *Overview For Different Model Prototypes* (*Bruhn et al. IJCV 2006*)

Two to three orders of magnitude for different smoothness terms

Type	Solver	FPS	Speedup
Horn and Schunck	Full Multigrid	62.7	220
Nagel and Enkelmann	Full Multigrid	5.8	275
Nonquadratic (TV norm)	FAS Full Multigrid	12.1	372

Three to four orders of magnitude for high accuracy methods

Type	Solver	FPS	Speedup
Nonquadratic (L_1 + TV)	FAS Full Multigrid	11.5	**2836**
Brox et al. ECCV 2004	FAS Full Multigrid	9.9	**10588**
Bruhn/Weickert ICCV 2005	Warp FAS Full Multigrid	2.9	**5454**

Are further accelerations possible?

M I A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

Advanced Multigrid Strategies (7)

Real-Time Live Demo

- *Live Computation with Webcam (160 × 120)*

Start
Stop

Flow fields are computed with a 1.7 GHz PentiumM CPU

M I A
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

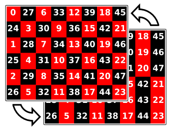
Implementations on Parallel Hardware (1)

Implementations on Parallel Hardware

Moderate Parallel Systems

- *Example: Cell Processor - Sony Playstation 3*
  - two types of parallelism
  - coarse grain: 6 SPUs on ringbus interface
  - fine grain: SIMD with 4 instructions per SPU

- *Parallelization Variant 1: Solver Parallelization*
  (*Gwosdek et al. VMV 2008, Gwosdek et al. JRTIP 2009*)
  - iterative solvers without local data dependency:
    red-black GS/SOR with data reshuffling
  - difficult to combine with multigrid ideas
  - performance loss due to synchronisation, communication and reordering

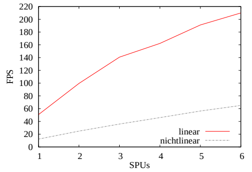


M I A
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

Implementations on Parallel Hardware (2)

Moderately Parallel Systems

- *Parallelization Variant 2: Frame Scheduling*
  (*Gwosdek et al. VMV 2008, Gwosdek et al. JRTIP 2009*)
  - distribution of different frames on different SPUs
  - minimal synchronisation and communication costs
  - high frame rate but latency of sequential algorithm
  - Cell processor, implementation based on CPL



- **linear case:** up to 210 FPS at 316 × 252 (16.7M pix/s)
- **nonlinear case:** up to 65 FPS at 316 × 252 (5.2M pix/s)

M I A
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

Implementations on Parallel Hardware (3)

Massively Parallel Systems

- *Example: Graphics Cards (GPGPU) - Intel, NVIDIA*
  - two types of parallelism
  - coarse grain: up to 30 multiprocessors
  - fine grain: up to 8 cores per processor

- *Parallelization Variant 1: Solver Parallelization*
  (*Grossauer/Thoman ICVS 2008*)
  - implementation of the Bruhn/Weickert ICCV 2005 multigrid method
  - iterative solvers without local data dependency:
    point-coupled damped Jacobi solver (does not require reshuffling)
  - performance loss at coarser levels (no FMG, replace W- by V-cycles)
  - NVIDIA GeForce 8800 GTX, implementation based on shader programs
    nonlinear case with warping: up to 17 FPS at 511 × 511 (4.5M pix/s)

M I A
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

Massively Parallel Systems

- *Parallelization Variant 2: Dual Algorithms* (*Zach et al. DAGM 2007, Steinbrücker et al. ICCV 2009, Wedel et al. ICCV 2009*)
  - decoupling of data and smoothness term via auxiliary variables

$$E(\mathbf{w}_1, \mathbf{w}_2) = \int_\Omega \underbrace{|I_x u_1 + I_y v_1 + I_t|}_{\text{data term}} + \frac{1}{2\theta} \underbrace{\left((u_1 - u_2)^2 + (v_1 - v_2)^2\right)}_{\text{coupling term}} + \alpha \underbrace{\left(|\nabla u_2| + |\nabla v_2|\right)}_{\text{smoothness term}} \, dx \, dy \, .$$

  - alternating optimization steps for data and smoothness term
  - **data term**: pointwise thresholding
  - **smoothness term:** Chambolle's algorithm (*Chambolle et al. JMIV 2004*)
  - NVIDIA GeForce GTX 280, implementation based on CUDA

    nonlinear case with warping: up to 32 FPS at 512 $\times$ 512 (8.4M pix/s)

Main problem of parallel systems: Severe limitation of fast memory!

Summary

- There exists a variety of efficient solvers
  - SOR as extrapolation variant of Gauß-Seidel (simple to implement)
  - Nested fixed point iterations allow to handle nonlinear problems
  - Multigrid is optimal solver but requires problem-specific adaptations
  - Parallel hardware allows further speedups (Cell, GPUs)
  - High accuracy and real-time performance are not contradictive

General Summary for the Tutorial

- **PART I:** Insights into the concept behind variational methods
- **PART II:** Problem-specific modelling for high accuracy results
- **PART III:** Efficient numerical strategies for minimisation

Acknowledgements

- Joachim Weickert
- Christoph Schnörr
- Daniel Cremers
- Jitendra Malik
- Christoph Bregler
- Bodo Rosenhahn
- Hans-Peter Seidel
- Henning Zimmer
- Levi Valgaerts
- Nils Papenberg
- Timo Kohlberger
- Yana Mileva
- Stephan Didas
- Agustin Salgado

Thank you very much!

further information: www.mia.uni-saarland.de
www.eecs.berkeley.edu/Research/Projects/CS/vision/