Image Processing and Computer Graphics

# Image Processing

Class 4

Energy Minimization

# Energy minimization concept

1. Formalize your model assumptions and cast the task as an **optimization problem**

$$E(x) = A_1(x) +, ..., + A_n(x)$$

2. Solve the optimization problem

$$x^* = \operatorname{argmin}_x E(x)$$

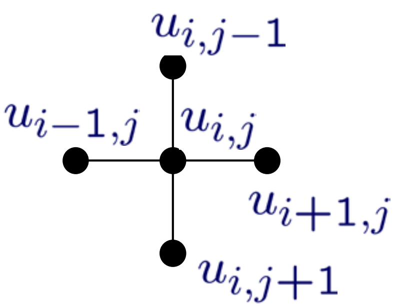
- Objective function $E(x)$ often called **energy** (motivated from physics)
- In machine learning it is often called **loss function**

# Example: image denoising

- First step: formulate the model assumptions
  - The outcome should be similar to the input image
  - The result should be smooth

- Second step: formalize these assumptions
  - Similarity to the input data (data term):

$$E_D(u_{i,j}) := \sum_{i,j} (u_{i,j} - I_{i,j})^2 \rightarrow \min$$




  - Similarity to neighboring values (smoothness term):

$$E_S(u_{i,j}) := \sum_{i,j} (u_{i+1,j} - u_{i,j})^2 + (u_{i,j+1} - u_{i,j})^2 \rightarrow \min$$

  - Yields an energy minimization problem including a weighting parameter $\alpha$

$$u^*_{i,j} = \operatorname{argmin}_{u_{i,j}} \left( E_D(u_{i,j}) + \alpha E_S(u_{i,j}) \right)$$

- Third step: solve this optimization problem

# Advantages

- All model assumptions are clearly stated
  → transparency
- Global approach: all variables are optimized in a joint manner; interdependencies are not lost by intermediate decisions → optimality
- Theoretical aspects can be analyzed:
  - Existence and uniqueness of solutions
  - Stability of solutions with respect of the input data
  - Difficulty of the problem class
- Usually fewer parameters than in heuristic multi-step methods
- Combination of energy minimization approaches is straightforward

# Problematic issues

- Formalizing the assumptions is rather ad-hoc: Why minimizing the sum of squared differences and not some other distance?

$$E_D(u_{i,j}) := \sum_{i,j} (u_{i,j} - I_{i,j})^2 \rightarrow \text{min}$$

→ Probabilistic interpretations can usually answer this question.

- Choosing the weight parameter(s) is not easy and often depends on the data.

→ (Fixed) parameters can be learned from a training dataset.

- Global optimization is often hard.
Heuristics obliterate the initial transparency of the model.

# Energy minimization in our denoising example

- Here is our energy from the denoising example

$$E(u_{i,j}) := \sum_{i,j} \left( (u_{i,j} - I_{i,j})^2 + \alpha \left( (u_{i+1,j} - u_{i,j})^2 + (u_{i,j+1} - u_{i,j})^2 \right) \right)$$

- How do we find the minimum?

- **Necessary condition** for a minimum: the first derivatives must be zero

$$\frac{dE}{du} = 0 \quad \Leftrightarrow \quad \frac{\partial E}{\partial u_{i,j}} = 0 \quad \forall i,j$$

- Here we go:

$$\begin{aligned} \frac{\partial E}{\partial u_{i,j}} = \; & 2(u_{i,j} - I_{i,j}) \\ & + 2\alpha(u_{i,j} - u_{i-1,j}) - 2\alpha(u_{i+1,j} - u_{i,j}) \\ & + 2\alpha(u_{i,j} - u_{i,j-1}) - 2\alpha(u_{i,j+1} - u_{i,j}) = 0 \end{aligned}$$

- At boundary pixels some terms are missing due to missing neighbors

# Linear system of equations

- Necessary conditions…

$$\begin{aligned}\frac{\partial E}{\partial u_{i,j}} = \; & (u_{i,j} - I_{i,j}) \\ & +\alpha(u_{i,j} - u_{i-1,j}) - \alpha(u_{i+1,j} - u_{i,j}) \\ & +\alpha(u_{i,j} - u_{i,j-1}) - \alpha(u_{i,j+1} - u_{i,j}) = 0\end{aligned}$$

- …can be written as a large linear system of equations (schematic view)

$$\begin{pmatrix} 1+2\alpha & -\alpha & & -\alpha & & & \\ -\alpha & 1+3\alpha & -\alpha & & -\alpha & & \\ & -\alpha & 1+3\alpha & -\alpha & & -\alpha & \\ -\alpha & & -\alpha & 1+4\alpha & -\alpha & & -\alpha \\ & -\alpha & & -\alpha & 1+3\alpha & -\alpha & \\ & & -\alpha & & -\alpha & 1+3\alpha & -\alpha \\ & & & -\alpha & & -\alpha & 1+2\alpha \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ . \\ . \\ . \\ u_{N-1} \\ u_N \end{pmatrix} = \begin{pmatrix} I_1 \\ I_2 \\ . \\ . \\ . \\ I_{N-1} \\ I_N \end{pmatrix}$$

- $N \times N$ system matrix (for $N$ pixels) is symmetric and positive definite

- It contains one main diagonal (central pixels) and four off-diagonals (for each of the four neighbors of a pixel)

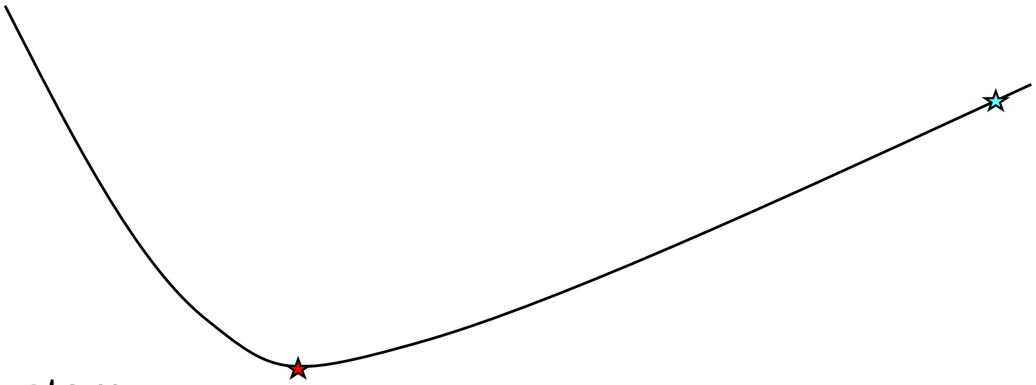
# Now what?

$$\begin{pmatrix} 1+2\alpha & -\alpha & & -\alpha & & & \\ -\alpha & 1+3\alpha & -\alpha & & -\alpha & & \\ & -\alpha & 1+3\alpha & -\alpha & & -\alpha & \\ -\alpha & & -\alpha & 1+4\alpha & -\alpha & & -\alpha \\ & -\alpha & & -\alpha & 1+3\alpha & -\alpha & \\ & & -\alpha & & -\alpha & 1+3\alpha & -\alpha \\ & & & -\alpha & & -\alpha & 1+2\alpha \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ . \\ . \\ . \\ u_{N-1} \\ u_N \end{pmatrix} = \begin{pmatrix} I_1 \\ I_2 \\ . \\ . \\ . \\ I_{N-1} \\ I_N \end{pmatrix}$$

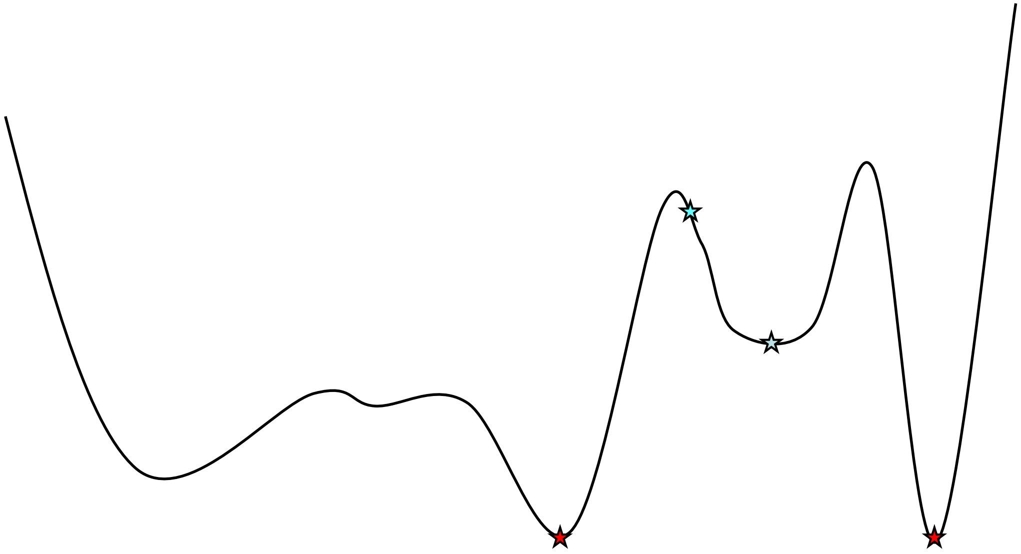
- The system matrix $A$ is **sparse** (almost all entries are 0)
- Positive definite → the inverse $A^{-1}$ exists and we can solve for $\mathbf{u}$
- Questions:
  - When do we get a linear/nonlinear system with a unique solution?
  - How can this system be solved (efficiently)?

# Convexity

- Convex functions:
  - Positive curvature
  - No local minima
  - Global minimum is unique
  - Minimization by setting the derivative to 0 and solving the emerging linear or nonlinear system



- Non-convex functions:
  - Usually many local minima (and maxima)
  - Global minimum may not be unique
  - Global minimization is usually impossible, only heuristics exist



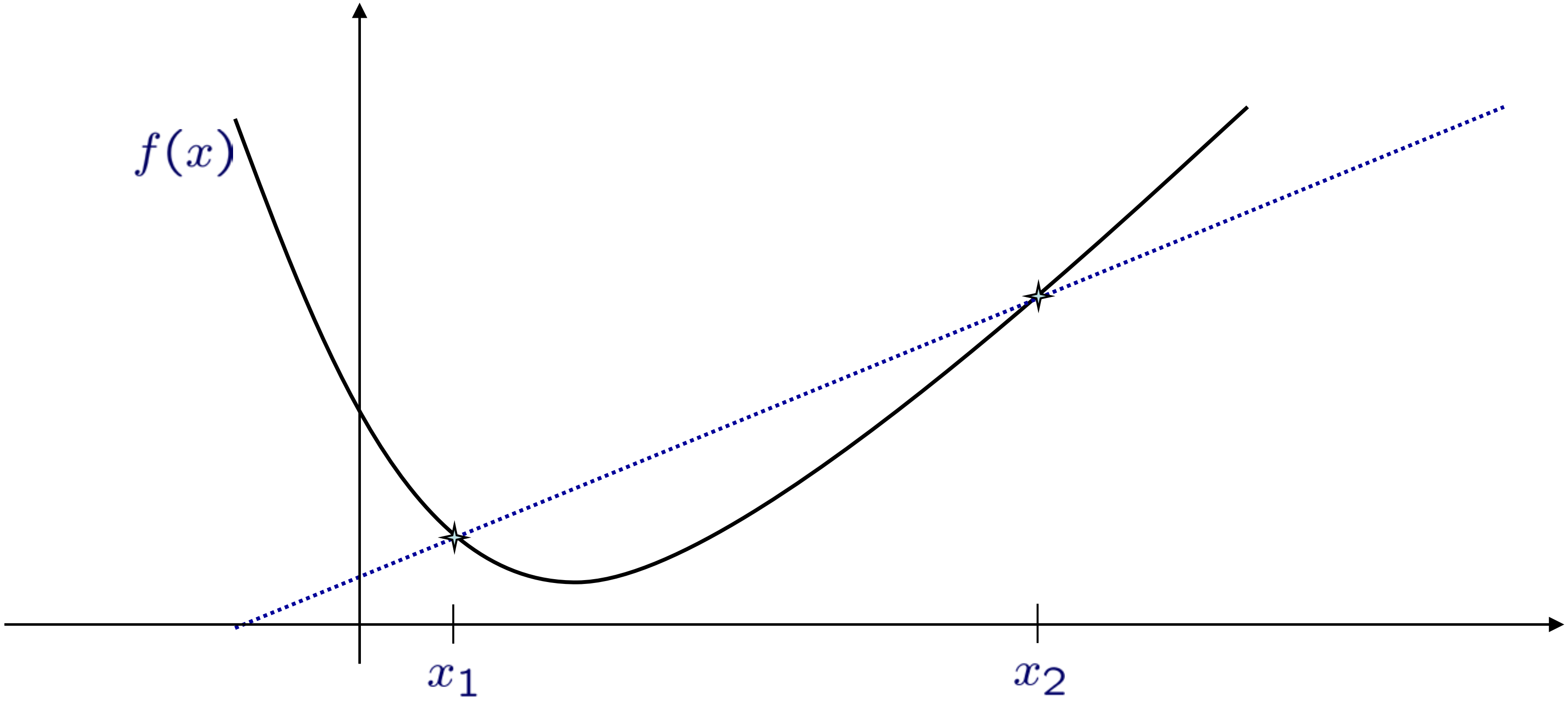
# Convexity

- A function is **convex** if

$$f((1-\alpha)x_1+\alpha x_2) \leq (1-\alpha)f(x_1)+\alpha f(x_2) \quad \forall x_1, x_2, \forall \alpha \in (0,1)$$

- A function is **strictly convex** if

$$f((1-\alpha)x_1+\alpha x_2) < (1-\alpha)f(x_1)+\alpha f(x_2) \quad \forall x_1, x_2, \forall \alpha \in (0,1)$$



# Convexity

- Theorem: every convex combination of (strictly) convex functions is again (strictly) convex

- Proof:

$$h(x) := \gamma f(x) + \delta g(x) \quad \gamma, \delta \in \mathbb{R}, \quad \gamma, \delta \geq 0$$

$$\begin{aligned} h((1-\alpha)x_1 + \alpha x_2) &= \gamma f((1-\alpha)x_1 + \alpha x_2) + \delta g((1-\alpha)x_1 + \alpha x_2) \\ &\leq \gamma(1-\alpha)f(x_1) + \gamma\alpha f(x_2) + \delta(1-\alpha)g(x_1) + \delta\alpha g(x_2) \\ &= (1-\alpha)(\gamma f(x_1) + \delta g(x_1)) + \alpha(\gamma f(x_2) + \delta g(x_2)) \\ &= (1-\alpha)h(x_1) + \alpha h(x_2) \end{aligned}$$

- Our energy function

$$E(u_{i,j}) := \sum_{i,j} \left( (u_{i,j} - I_{i,j})^2 + \alpha \left( (u_{i+1,j} - u_{i,j})^2 + (u_{i,j+1} - u_{i,j})^2 \right) \right)$$

is a convex combination of strictly convex (quadratic) functions
→ It is strictly convex as well
→ It has a unique global minimum

# Solving the linear system

$$\begin{pmatrix} 1+2\alpha & -\alpha & & -\alpha & & & \\ -\alpha & 1+3\alpha & -\alpha & & -\alpha & & \\ & -\alpha & 1+3\alpha & -\alpha & & -\alpha & \\ -\alpha & & -\alpha & 1+4\alpha & -\alpha & & -\alpha \\ & -\alpha & & -\alpha & 1+3\alpha & -\alpha & \\ & & -\alpha & & -\alpha & 1+3\alpha & -\alpha \\ & & & -\alpha & & -\alpha & 1+2\alpha \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ . \\ . \\ . \\ u_{N-2} \\ u_{N-1} \end{pmatrix} = \begin{pmatrix} I_1 \\ I_2 \\ . \\ . \\ . \\ I_{N-2} \\ I_{N-1} \end{pmatrix}$$

- The two additional off-diagonals (together with the size of matrix) rule out Gauß-elimination (in 1D, however, Gauß-elimination is very efficient).
- An iterative solver is needed to preserve the sparsity of the matrix
- Simplest iterative solver: **Jacobi method**
- Converges if the matrix is **strictly diagonal dominant**

$$|a_{ii}| > \sum_{i \neq j} |a_{ij}| \quad \forall i$$

# Jacobi method

- Decompose the matrix into its diagonal part $D$ and its off-diagonal part $M$

  $$A = D + M$$

- For the linear system this means

  $$Ax = b \quad \Leftrightarrow \quad (D+M)x = b \quad \Leftrightarrow \quad Dx = b - Mx$$

- $D^{-1}$ can be computed very easily: just replace the diagonal elements by their inverse.

- Now we can compute the solution $x$ iteratively. Starting with any initialization $x^0$, iterate

  $$x^{k+1} = D^{-1}(b - Mx^k)$$

- Iterate until the norm of the **residual** $r^k := Ax^k - b$ is smaller than a threshold or the change in the solution $(x^{k+1} - x^k)^2$ becomes small. When the change is 0, the iterate has **converged**.

# Jacobi method

- Advantages:
  - Simple
  - Can be implemented in parallel

- Disadvantages:
  - Slow
  - Convergence only for $k \rightarrow \infty$
  - Computation not in-place

- Alternatives:
  - Gauß-Seidel, Successive Over-relaxation (faster, in-place)
  - Conjugate gradient (convergence after finite number of iterations)
  - Multigrid methods (sometimes much faster)

# Gauß-Seidel method

- Split the off-diagonal part $M$ into the lower triangle $L$ and the upper triangle $U$

$$Ax = b \quad \Leftrightarrow \quad Dx = b - Lx - Ux$$

- During iteration, traverse the vector $x$ from top to bottom and use already the new values for multiplication with the lower triangle

$$x^{k+1} = D^{-1}(b - Lx^{k+1} - Ux^k)$$

- In our denoising example this reads

$$u_i^{k+1} = \frac{I_i + \alpha \sum_{j \in \mathcal{N}^-(i)} u_j^{k+1} + \alpha \sum_{j \in \mathcal{N}^+(i)} u_j^k}{1 + \sum_{j \in \mathcal{N}(i)} \alpha}$$

- Converges if $A$ is positive or negative definite
- For already updated neighbors take the new value → in-place computation
- Recursive propagation of information → faster

# Successive over-relaxation (SOR)

- Emphasize the Gauß-Seidel idea by over-relaxing the new solution

$$x^{k+1} = (1-\omega)x^k + \omega D^{-1}(b - Lx^{k+1} - Ux^k)$$

- For $\omega = 1$ this is the Gauß-Seidel method
- Converges for positive- or negative-definite matrices (all eigenvalues positive or negative, respectively), if $\omega \in (0,2)$
- Over-relaxation for $\omega > 1$: faster convergence
- Under-relaxation for $\omega < 1$ : can help establish convergence in case of divergent iterative processes
- Optimal $\omega$ must be determined empirically

# Conjugate gradient (CG)

- Two non-zero vectors $u$ and $v$ are **conjugate** with respect to $A$ if the inner product $\langle u, v \rangle_A := u^\top A v = 0$ . This means the two vectors are orthogonal with respect to this special scalar product.

- A set of $n$ conjugate vectors $\{p_k\}$ forms a basis of $\mathbb{R}^n$ , so the solution $x^*$ of $Ax = b$ can be expanded as $x^* = \alpha_1 p_1 +, ..., +\alpha_n p_n$

- The coefficients $\alpha_k$ are derived as follows:

$$Ax^* = \alpha_1 A p_1 +, ..., +\alpha_n A p_n = b$$

$$p_k^\top A x^* = p_k^\top \alpha_1 A p_1 +, ..., +p_k^\top \alpha_n A p_n = p_k^\top b$$ (expansion with $p_k$ )

$$\alpha_k = \frac{p_k^\top b}{p_k^\top A p_k}$$

- After $n$ computations we obtain the exact solution $x^*$

- Good choice of $\{p_k\}$ → few coefficients approximate the solution well

# Conjugate gradient: iteratively assembling the basis

- Start with some initial point $x^0$
- Let $p_0$ be the residual $r_0 = b - Ax^0$. This is the gradient of

  $$E(x) = \frac{1}{2}x^\top Ax - b^\top x$$

  the minimizer of which is $x^*$. Therefore the name conjugate gradient.
- Iteratively compute:

  $$\alpha_k = \frac{r_k^\top r_k}{p_k^\top A p_k} \qquad x^{k+1} = x^k + \alpha_k p_k \qquad r_{k+1} = r_k - \alpha_k A p_k$$

  $$\beta_k = \frac{r_{k+1}^\top r_{k+1}}{r_k^\top r_k} \qquad p_{k+1} = r_{k+1} + \beta_k p_k$$
- Stop when residual is small. Guaranteed solution after $n$ iterations.

# Conjugate gradient: convergence and preconditioning

- Matrix $A$ must be symmetric and positive definite

- Usually in image processing, computing the exact solution is not an option since $n$ is the number of pixels

- Number of iterations needed to get a good approximate solution depends on the **condition number** of $A$ (largest vs. smallest eigenvalue). The same holds for the other iterative methods (Gauß-Seidel, etc.)

- Sometimes so-called **preconditioners** $P^{-1}$ are used to have a small condition number for $P^{-1}A$

$$Ax = b \quad \Leftrightarrow \quad P^{-1}Ax = P^{-1}b$$

- Simplest preconditioner: Jacobi preconditioner

$$P = D \quad \Leftrightarrow \quad P^{-1} = D^{-1}$$

$$Ax = b \quad \Leftrightarrow \quad D^{-1}Ax = D^{-1}b$$

# Multigrid methods

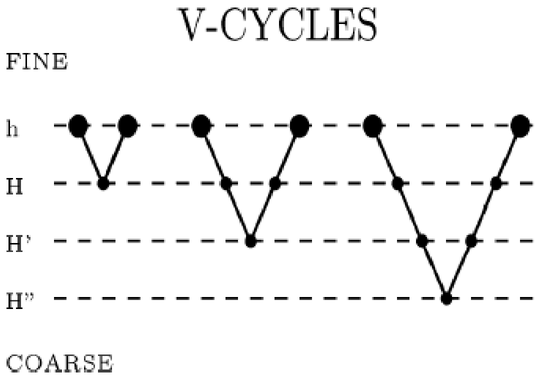
- All previous linear solvers have the drawback that they only act locally.
- This is due to the sparsity of the matrix: one iteration only distributes information at a pixel to its four neighbors (the recursive nature of Gauß-Seidel relaxes this statement a bit).
- Idea of multigrid solvers: shorten distances by regarding the system from a coarser point of view
- Additional effect: coarse versions of the system have fewer entries
  → iterations are faster at coarse levels
- Different types of multigrid solvers
  - Unidirectional (cascadic) multigrid
  - Bidirectional (correcting) multigrid
  - Full multigrid (a combination of both)

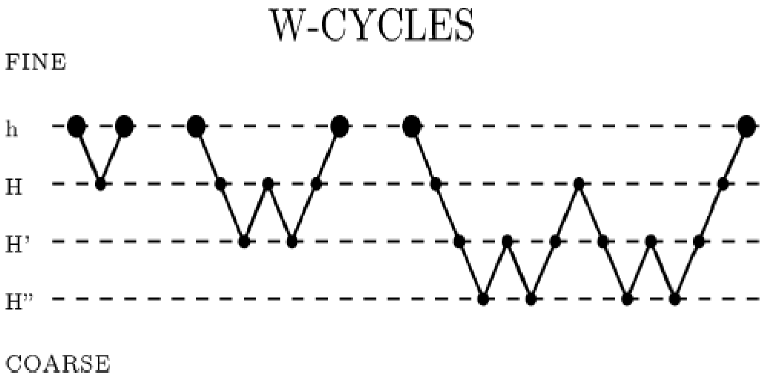
# Unidirectional (cascadic) multigrid

- Create downsampled versions of the linear system (usually by downsampling the image and deriving the linear system from this)
- Compute first approximate solution at the coarse grid (e.g. with SOR)
- Take upsampled result as initial guess for the next finer grid
- Refine result there (again with SOR)
- Advantages:
  - Iterations at the coarse level are very fast
  - Simple implementation
- Disadvantage:
  - Coarse level systems often do not approximate the original system well
    → Iterations at coarse levels do not really help

# Correcting (bidirectional) multi-grid

- Basic idea: do not downsample the image <u>but the error</u>
- Compute first solution at fine grid
- Correct error at coarse grid
- Refine result at finer grid

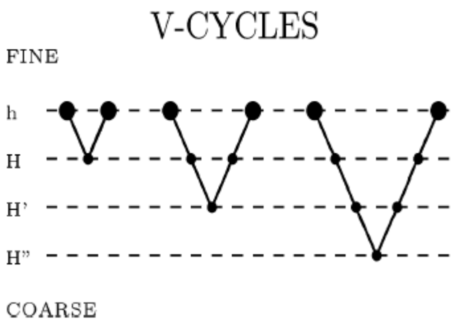



Author: Andrés Bruhn

# Correcting multigrid in detail



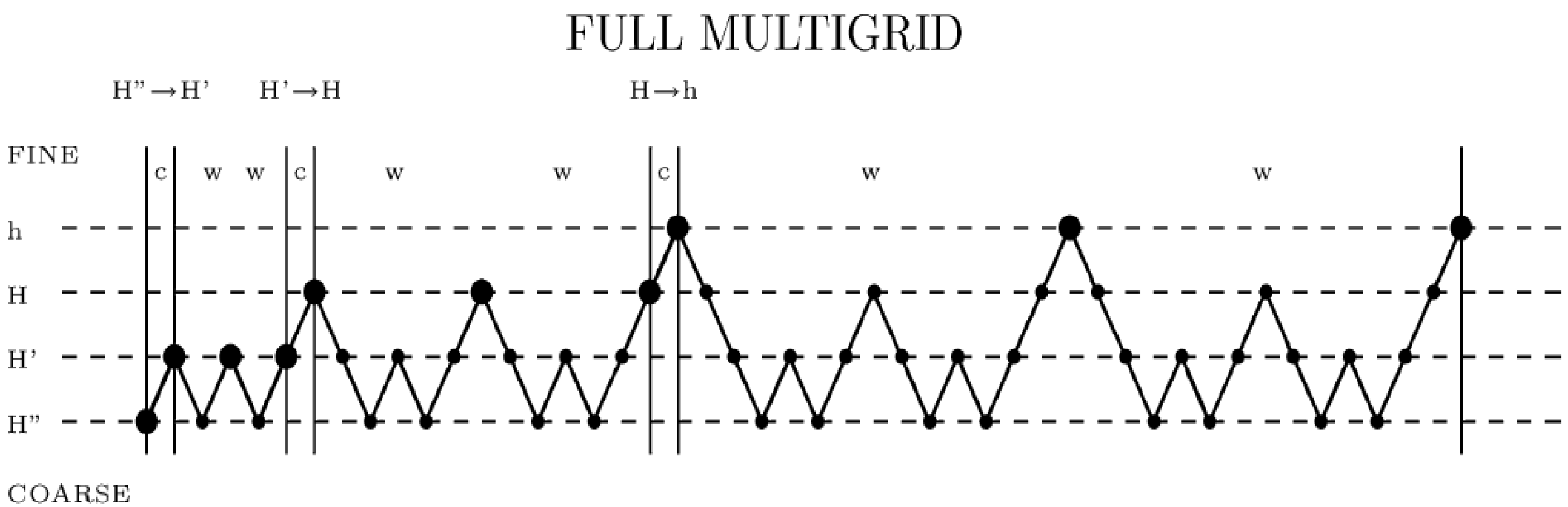

1. Presmoothing relaxation step $A^h x^h = b^h$
   - Run some iterations at the fine grid
   - Yields approximate solution $\tilde{x}^h$
   - Remaining error: $e^h = x^h - \tilde{x}^h$

2. Correction step:
   - Goal: compute error $A^h e^h = r^h$ $\qquad r^h = b^h - A^h \tilde{x}^h$
   - Local part of error already removed
     → Solve this system at coarser grid $A^H e^H = r^H$
   - Transfer error to fine grid and correct the solution $\tilde{\tilde{x}}^h = \tilde{x}^h + \tilde{e}^h$

3. Postsmoothing relaxation step
   - Apply some further iterations at fine grid to remove local errors introduced by $\tilde{e}^h$

# Full multigrid

- Combination of cascadic and correcting multigrid
- Start at coarse grid with downsampled image
- At each finer level apply a W-cycle




Author: Andrés Bruhn

# Integer problems

- Here we were considering problems with continuous variables (each vector component of the solution is a real number)
- Segmentation and matching typically leads to integer problems
- These are combinatorial problems, only few of them being solvable in polynomial time
- Some typical problems arising in computer vision are:
  - Linear programs (LP)
  - Integer quadratic programs (IQP) including special cases like min-cut
  - Second order cone programs (SOCP)
- More in the Computer Vision course

# Summary

- The energy minimization framework is a sound way to model and solve image processing problems
- All model assumptions are clearly stated, no hidden assumptions
- Global optimization is “easy” if the energy function is convex
- The necessary condition for a minimum is that the gradient is zero
- Leads to a large, but sparse, linear or nonlinear system of equations
- There are several methods to solve sparse linear systems iteratively, some are easier to implement, others are faster

# Literature


- D. Young: Iterative Solution of Large Linear Systems, Academic Press, 1971. Reprint by Dover 2003.
- J. R. Shewchuk: An introduction to the Conjugate Gradient method without the agonizing pain. CMU Technical Report, 1994.