Gradient Domain Manipulation Techniques in Vision and Graphics

Amit Agrawal and Ramesh Raskar

Mitsubishi Electric Research Labs (MERL)
Cambridge, MA, USA

Course WebPage :
http://www.cfar.umd.edu/~aagrawal/ICCV2007Course/

Course: Gradient Domain Techniques

Course Web Page

Google "ICCV 2007 Gradient Course"

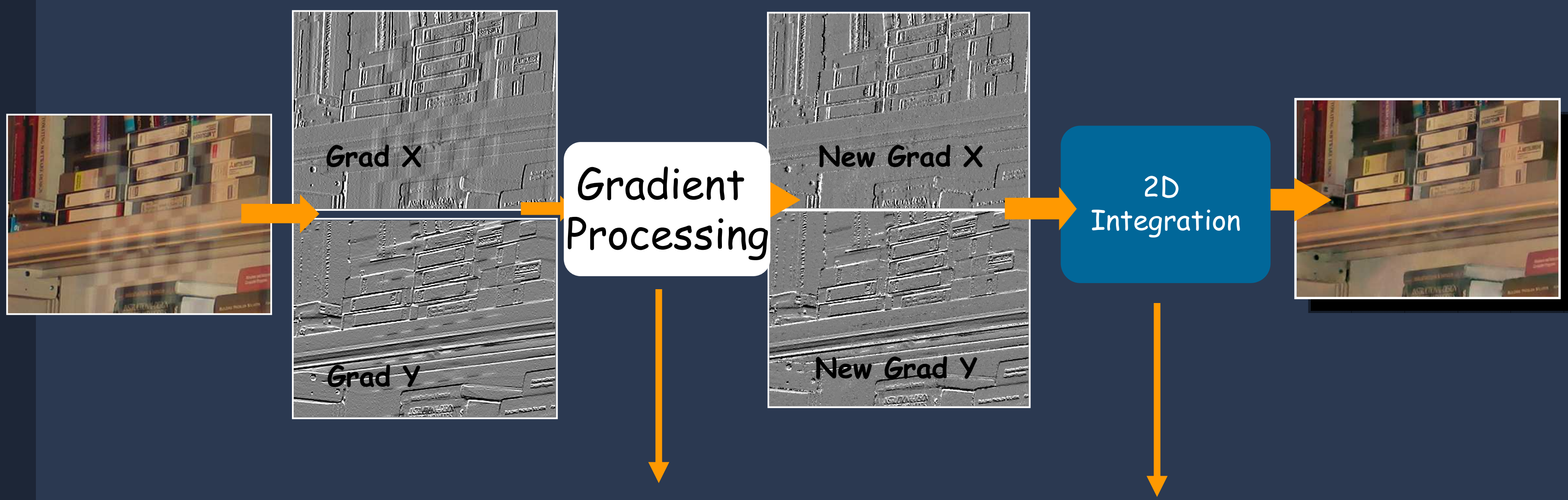
Schedule

Introduction	(30 min, Agrawal)
Gradient Domain Manipulations	(1 hr, Raskar)
Break	(30 min)
Reconstruction Techniques	(1 hr, Agrawal)
Advanced Topics	(30 min, Raskar)
Discussion	

Course Web Page : Google "ICCV 2007 Gradient Course"

Intensity Gradient Manipulation

A Common Pipeline



Last Section

This Section

Overview: The Reconstruction problem

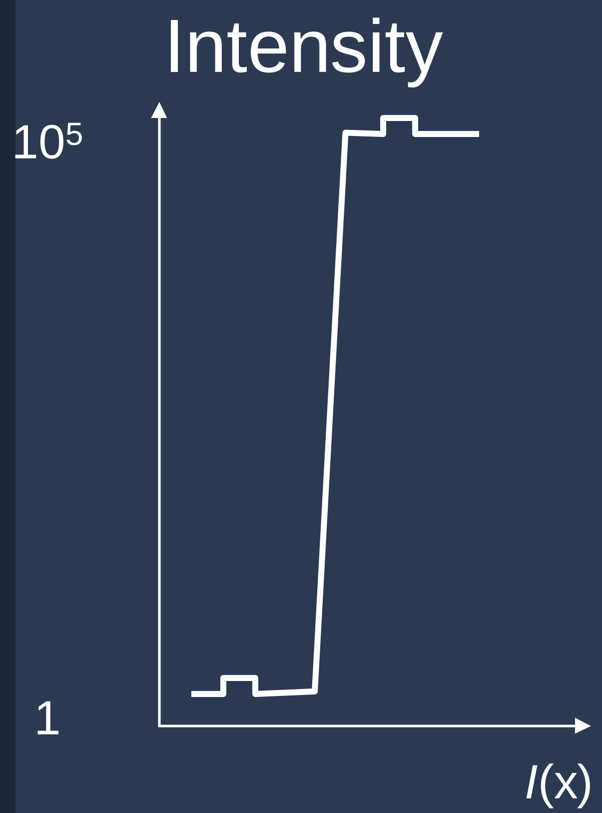
Gradients **Images**

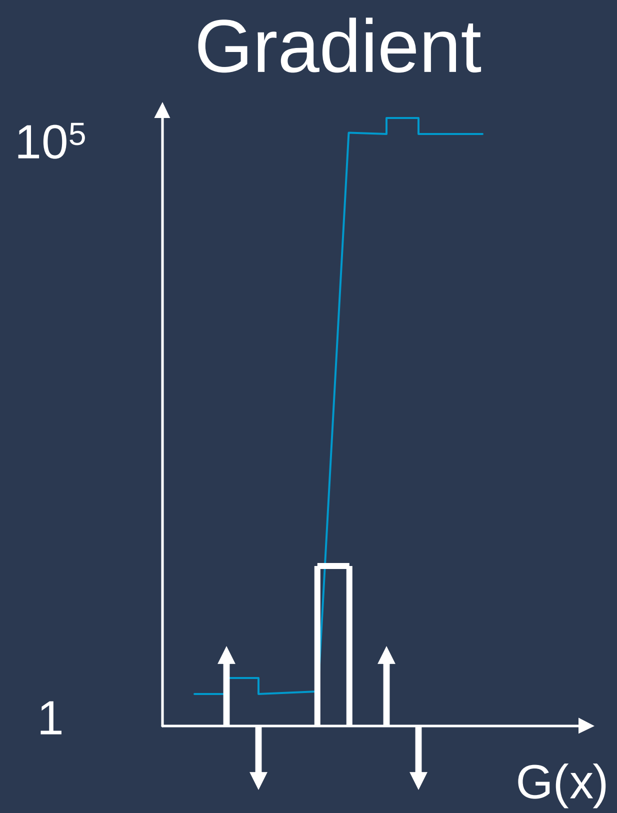
- Algorithms
  - Poisson solver: Least Squares
  - Other approaches?
    - Projection on basis, Robust reconstruction, Gradient Transformations
- Numerical Methods
  - Direct solutions, Multigrid, Preconditioned congujate gradients, Hierarchical basis, Approximate Solutions
  - Tradeoffs

Reconstruction from Gradients

- Intensity gradient manipulation
  - Reconstruction to obtain desired image
- Spatio-Temporal gradients
  - Reconstruction to obtain desired video
- Surface gradients
  - Photometric Stereo, Shape from Shading
  - Obtain shape
- Mesh manipulation
  - 3D

Intensity Gradient in 1D



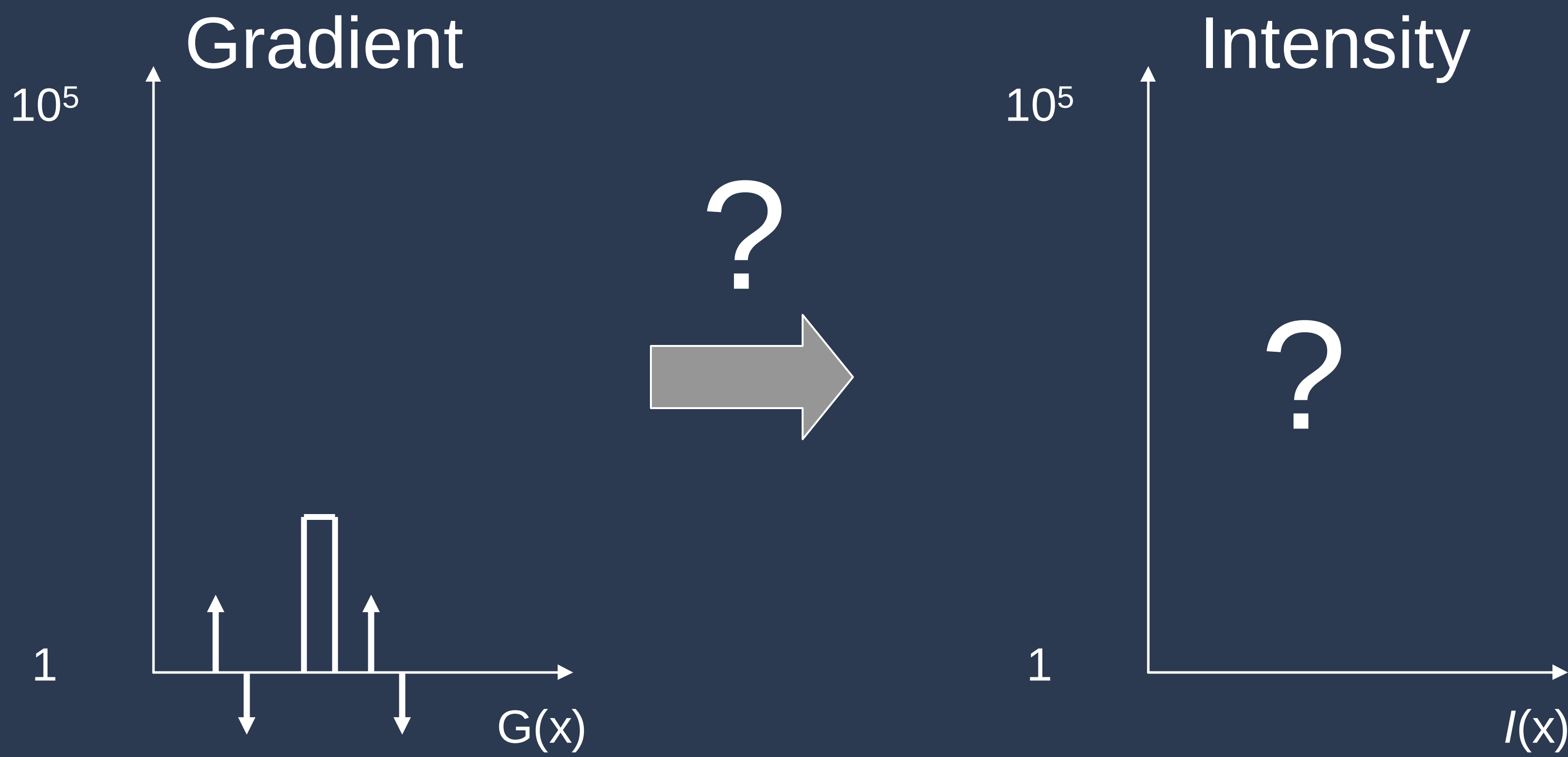


Gradient at x,

$$G(x) = I(x+1) - I(x)$$

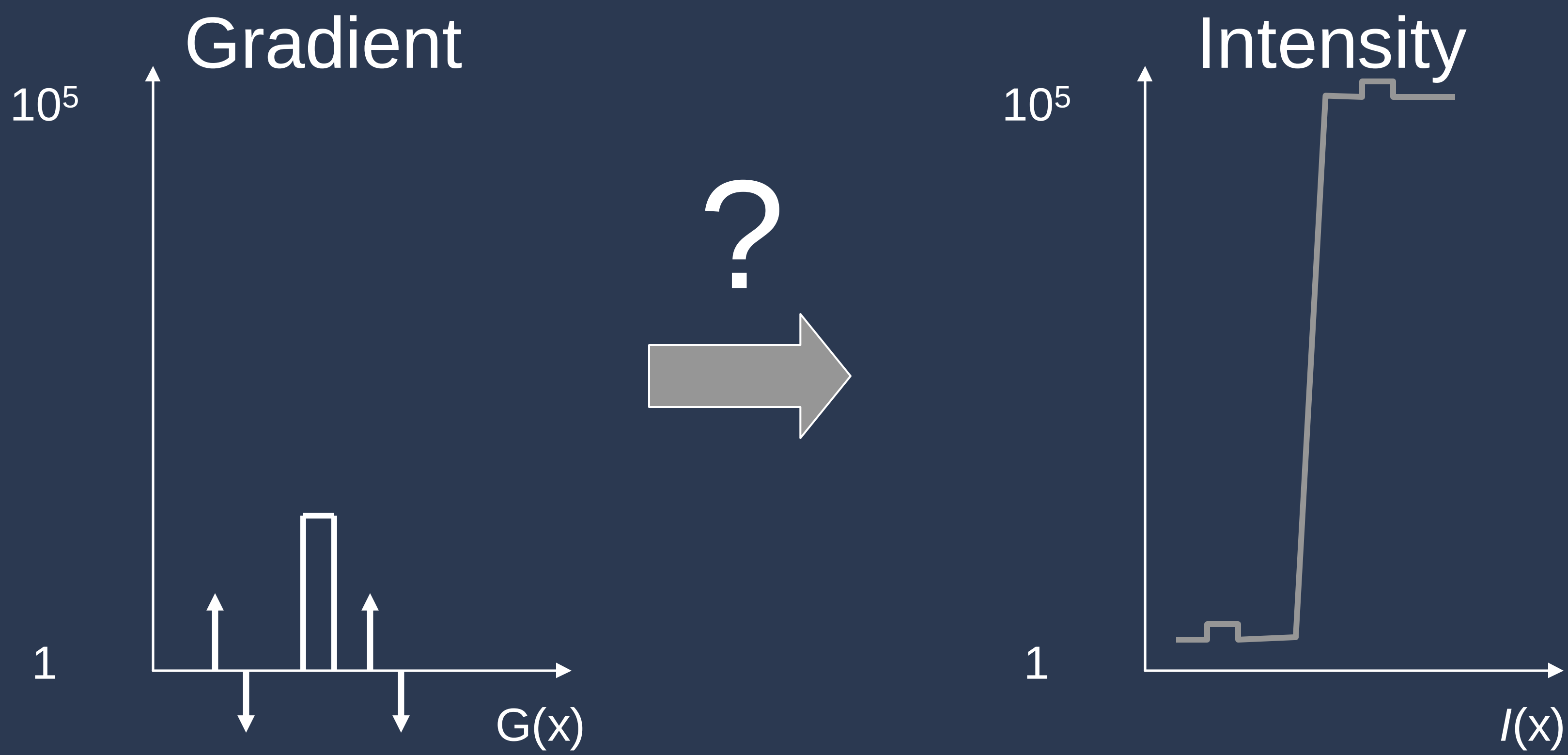
Forward Difference

Reconstruction from Gradients



For *n* intensity values, *n-1* gradients

Reconstruction from Gradients

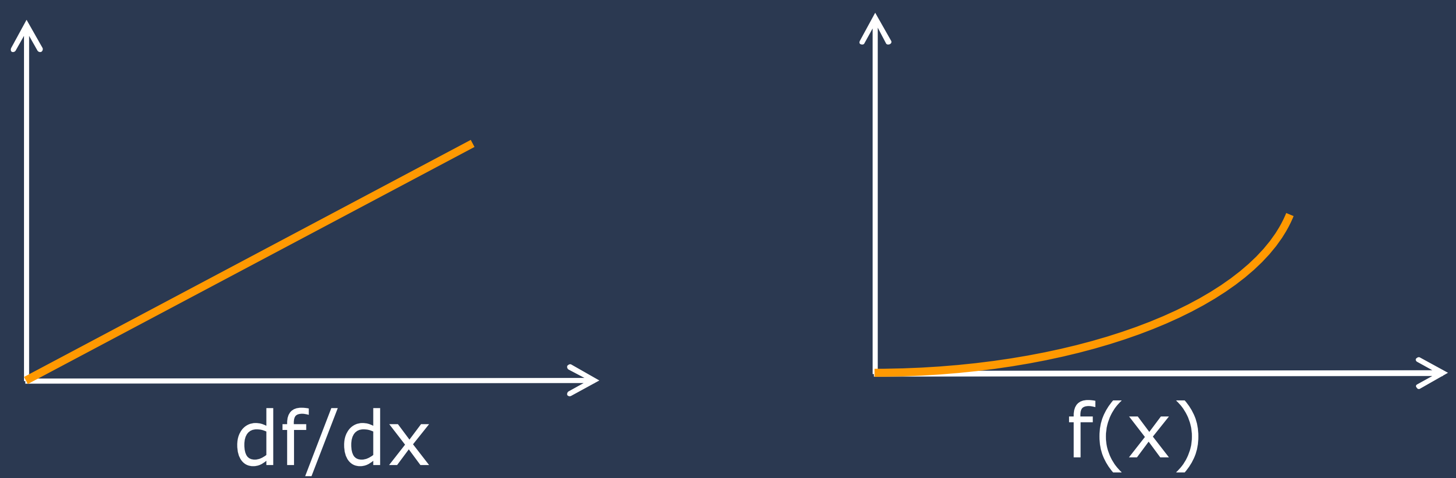


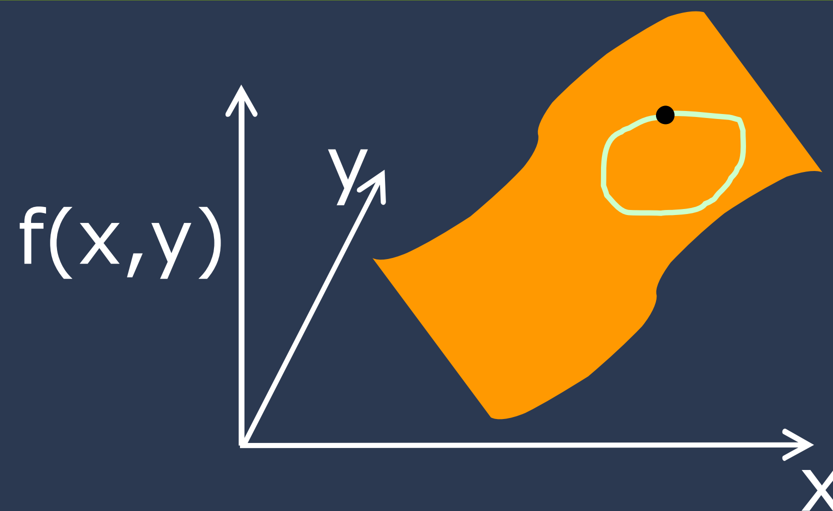
1D Integration

$$I(x) = I(x-1) + G(x)$$

Cumulative sum

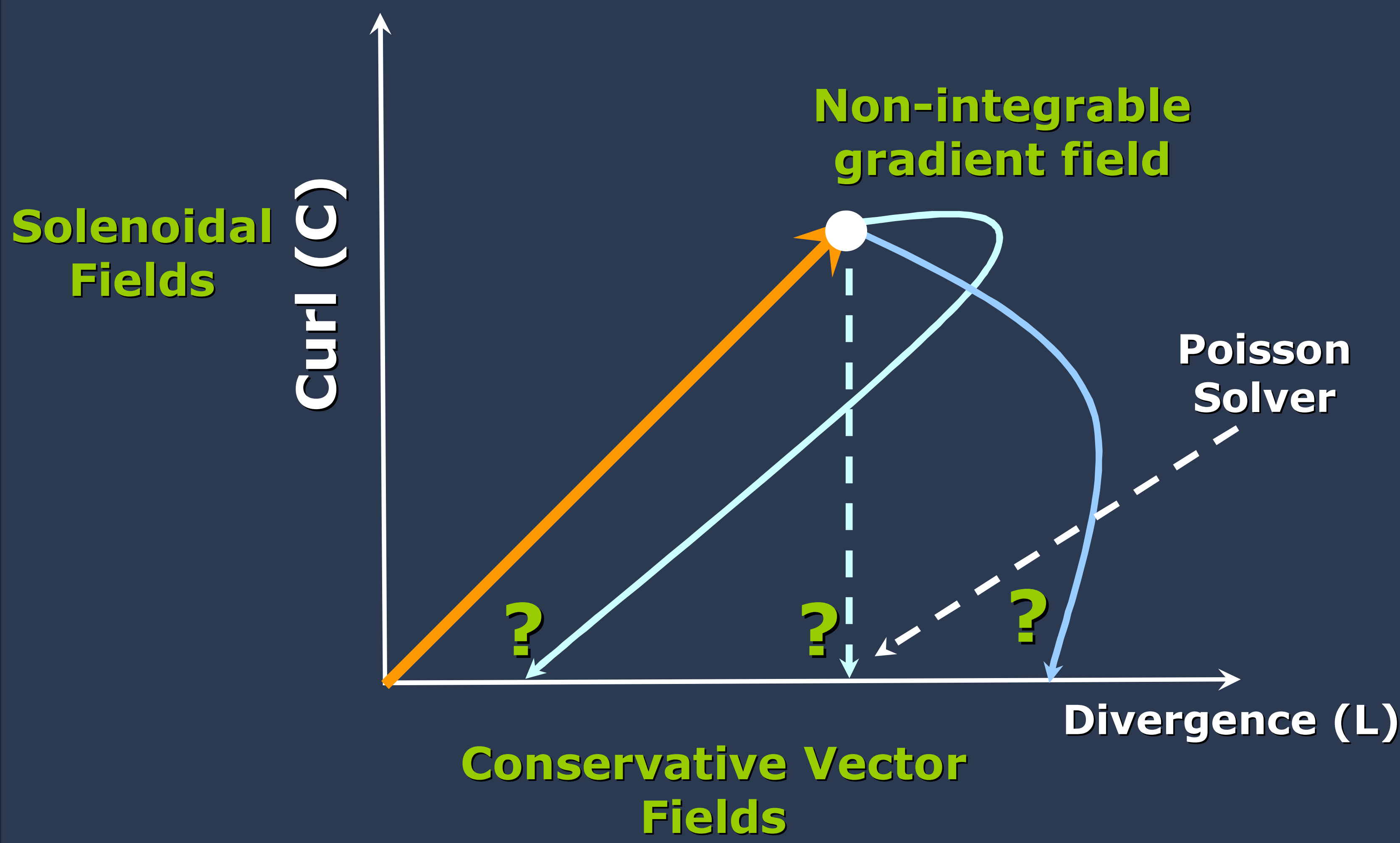
2D Integration is non-trivial





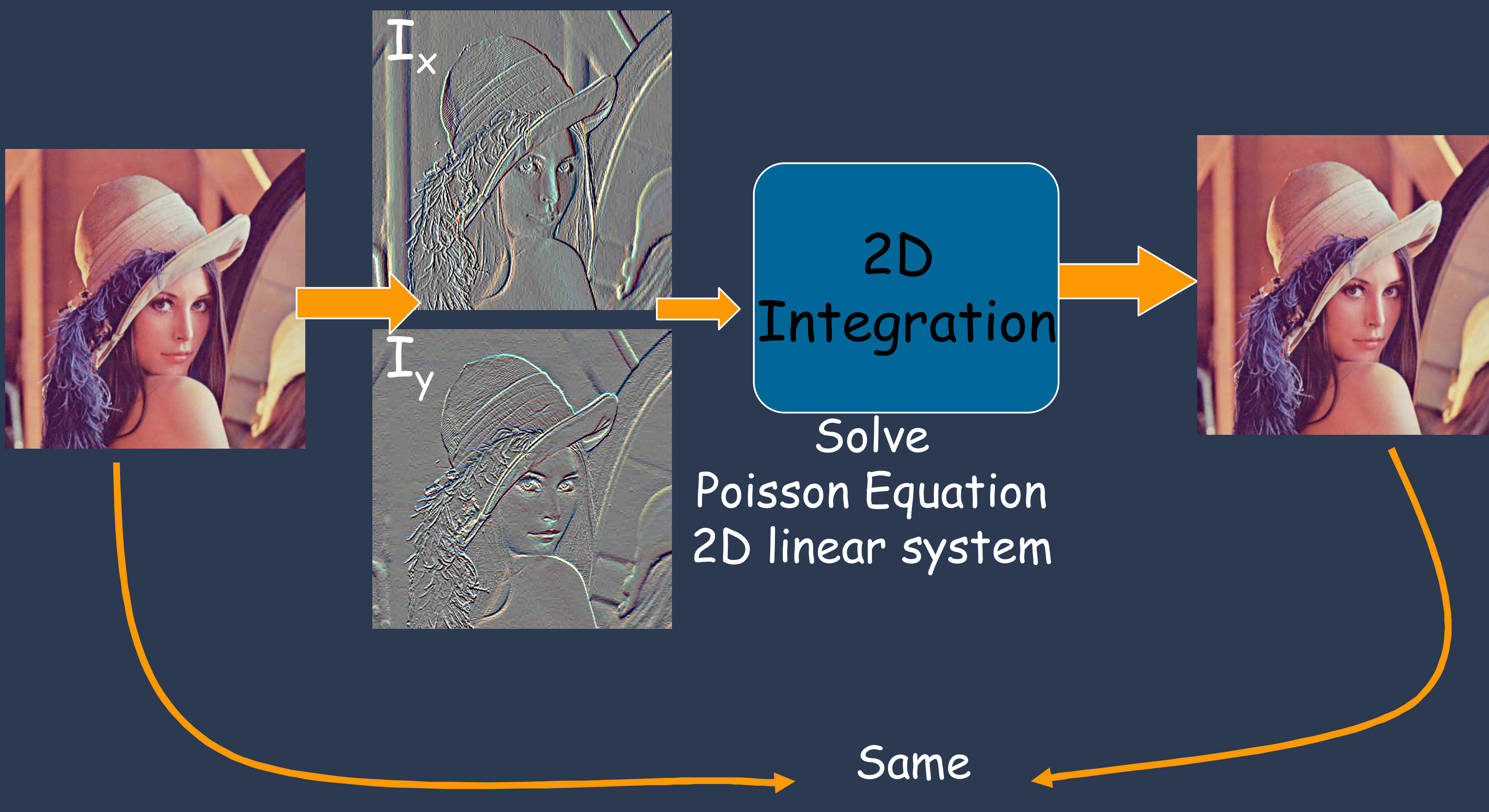
Reconstruction depends on chosen path

The Reconstruction Problem



Reconstruction from Gradients

Sanity Check: Recovering Original Image

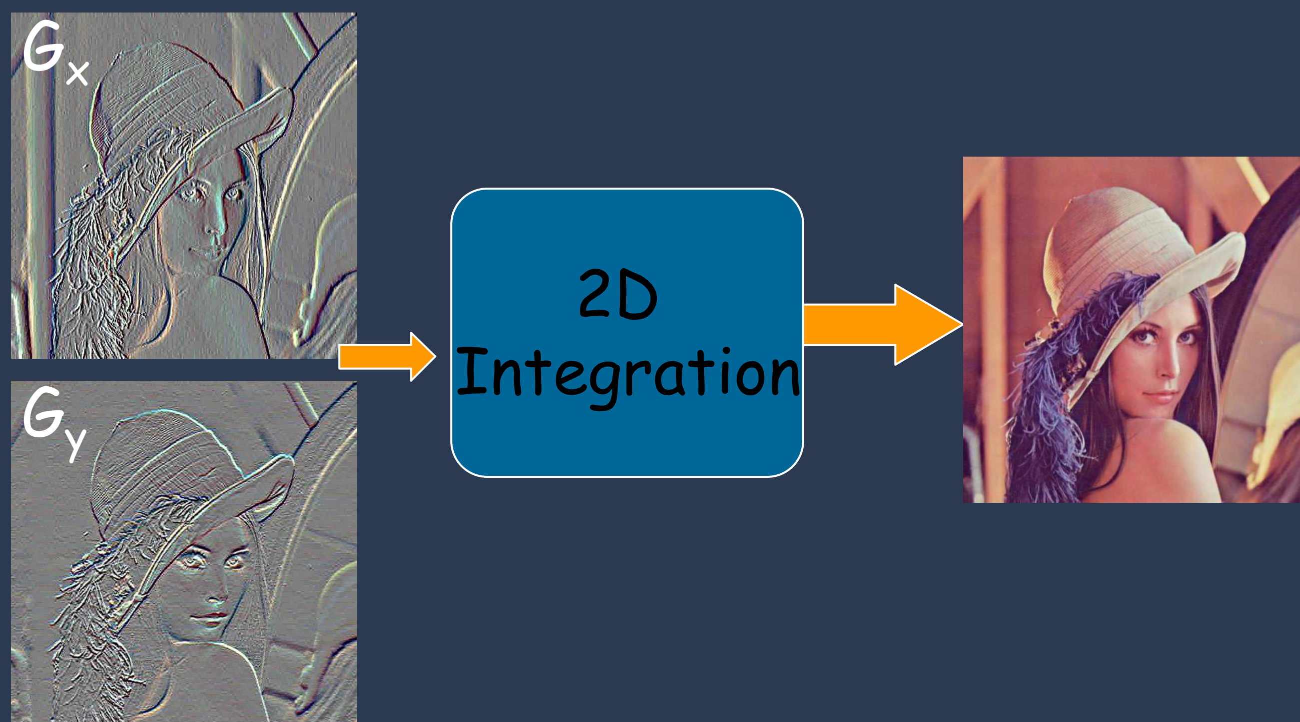


Reconstruction from Gradients

Given $G(x,y) = (G_x , G_y)$

How to compute $I(x,y)$ for the image ?

For n^2 image pixels, $2n^2$ gradients !



Reconstruction from Gradient Field G

- Look for image I with gradient closest to G in the least squares sense.
- I minimizes the integral: $\iint F(\nabla I, G)\, dx\, dy$

$$F(\nabla I, G) = \|\nabla I - G\|^2 = \left(\frac{\partial I}{\partial x} - G_x\right)^2 + \left(\frac{\partial I}{\partial y} - G_y\right)^2$$

Euler-Lagrange Equation

- I must satisfy: $$\frac{\partial F}{\partial I} - \frac{d}{dx}\frac{\partial F}{\partial I_x} - \frac{d}{dy}\frac{\partial F}{\partial I_y} = 0$$
- Substituting F we get:

$$2\left(\frac{\partial^2 I}{\partial x^2} - \frac{\partial G_x}{\partial x}\right) + 2\left(\frac{\partial^2 I}{\partial y^2} - \frac{\partial G_y}{\partial y}\right) = 0$$

$$\nabla^2 I = \operatorname{div} G$$

Poisson Equation

$$\nabla^2 I = div(G_x, G_y) = \frac{\partial G_x}{\partial x} + \frac{\partial G_y}{\partial x}$$

Second order PDE

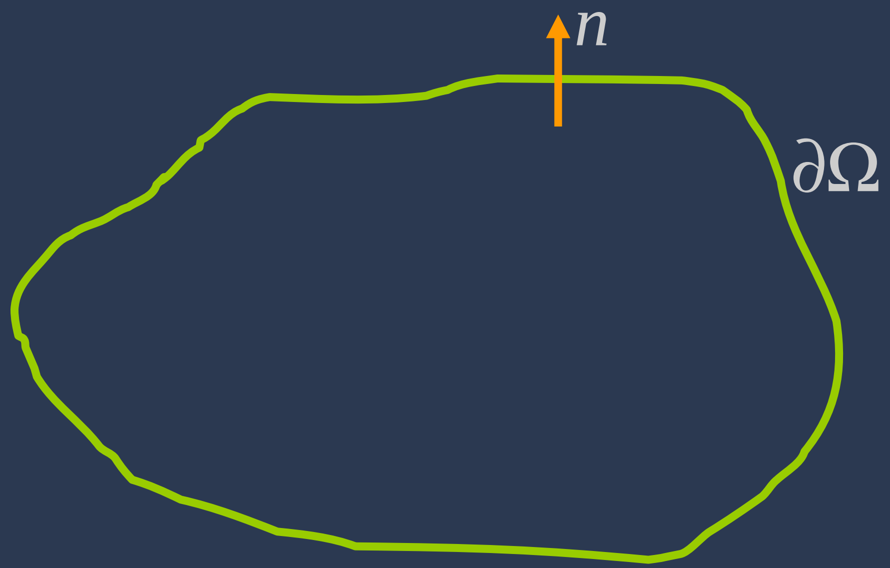
Boundary Conditions

- Dirichlet: Function values at boundary are known

$$I(x,y) = I_0(x,y) \forall (x,y) \in \partial\Omega$$

- Neumann: Derivative normal to boundary = 0

$$\nabla I(x,y) \bullet n(x,y) = 0, \forall (x,y) \in \partial\Omega$$



Numerical Solution

- Discretize Laplacian

$$\nabla^2 \longrightarrow \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

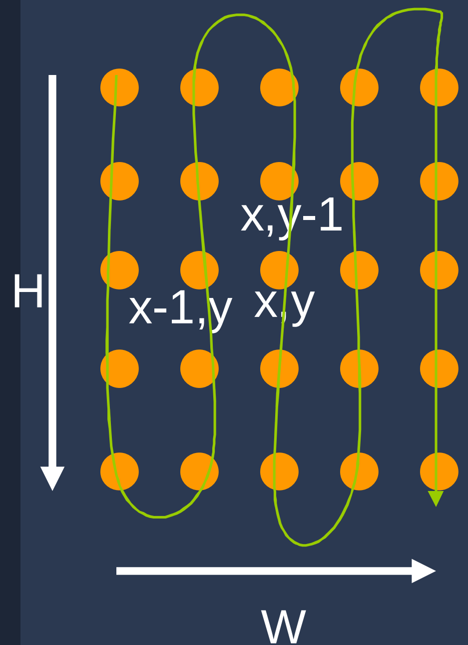
$$\nabla^2 I = div(G_x, G_y) = u(x, y)$$

$$-4I(x,y) + I(x,y+1) + I(x,y-1) + I(x+1,y) + I(x-1,y) = h^2 u(x,y)$$

h = grid size

Linear System

$$-4I(x,y)+I(x,y+1)+I(x,y-1)+I(x+1,y)+I(x-1,y)=u(x,y)$$

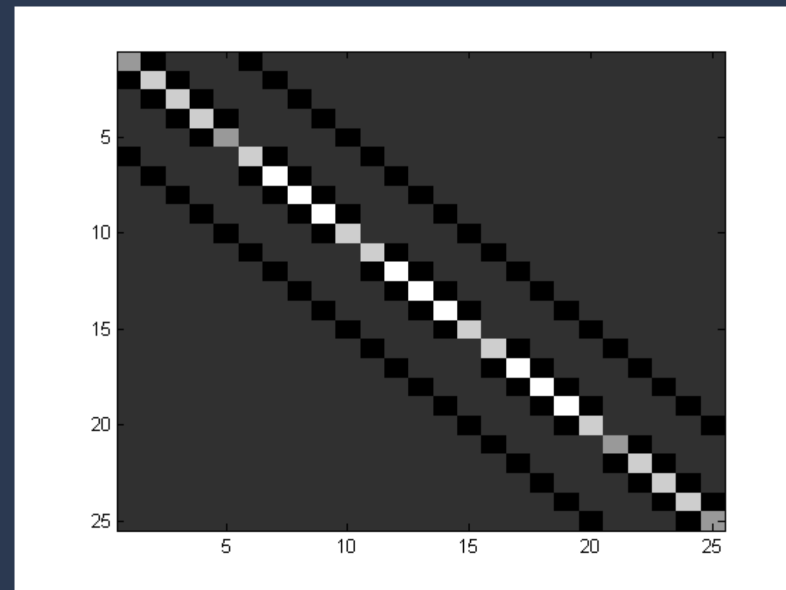


$$\underbrace{[.\;\; 1 \;\; . \;\; . \;\; . \;\; 1}_{H} \;\; -4 \;\; 1 \;\; . \;\; . \;\; . \;\; 1 \;\; .] \begin{bmatrix} . \\ . \\ I(x-1,y) \\ . \\ . \\ . \\ I(x,y-1) \\ I(x,y) \\ I(x,y+1) \\ . \\ . \\ . \\ I(x+1,y) \\ . \\ . \end{bmatrix} = \begin{bmatrix} . \\ . \\ . \\ . \\ . \\ . \\ u(x,y) \\ . \\ . \\ . \\ . \\ . \\ . \\ . \end{bmatrix}$$

A x b

Sparse Linear system

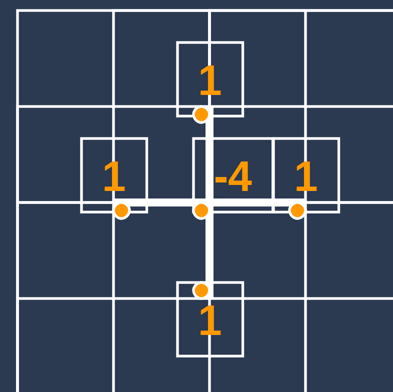
$$
\begin{bmatrix}
 & & 1 & -4 & 1 & & & & 1 & & & & \\
 & & & 1 & -4 & 1 & & & & 1 & & & \\
1 & & & & 1 & -4 & 1 & & & & 1 & & \\
 & 1 & & & & 1 & -4 & 1 & & & & 1 & \\
 & & 1 & & & & 1 & -4 & 1 & & & & 1 \\
 & & & 1 & & & & 1 & -4 & 1 & & & \\
 & & & & 1 & & & & 1 & -4 & 1 & &
\end{bmatrix}
$$



A matrix

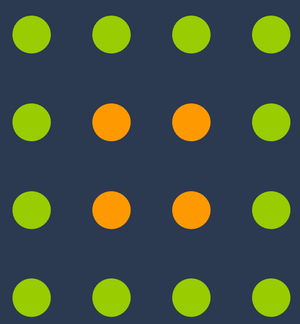
$$A = \left(\begin{array}{ccc|ccc|ccc}
-4 & -1 & & 1 & & & & & \\
1 & -4 & -1 & & 1 & & & & \\
& 1 & -4 & & & 1 & & & \\
\hline
1 & & & -4 & 1 & & 1 & & \\
& 1 & & 1 & -4 & 1 & & 1 & \\
& & 1 & & 1 & -4 & & & 1 \\
\hline
& & & 1 & & & -4 & 1 & \\
& & & & 1 & & 1 & -4 & 1 \\
& & & & & 1 & & 1 & -4
\end{array}\right)$$

Graph and "stencil"



Effect of Boundary Conditions

- Toy case: image 4*4 pixels



Dirichlet Boundary Condition

→ known pixel

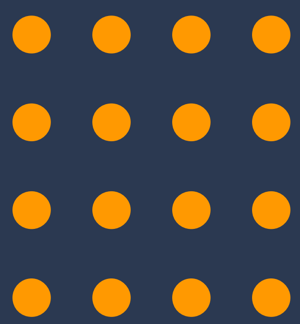
→ Unknown pixel

A matrix is 4 by 4

$$\begin{matrix} -4 & 1 & 1 & 0 \\ 1 & -4 & 0 & 1 \\ 1 & 0 & -4 & 1 \\ 0 & 1 & 1 & -4 \end{matrix}$$

Effect of Boundary Conditions

- Toy case: image 4*4 pixels



Neumann Boundary Condition

● → known pixel

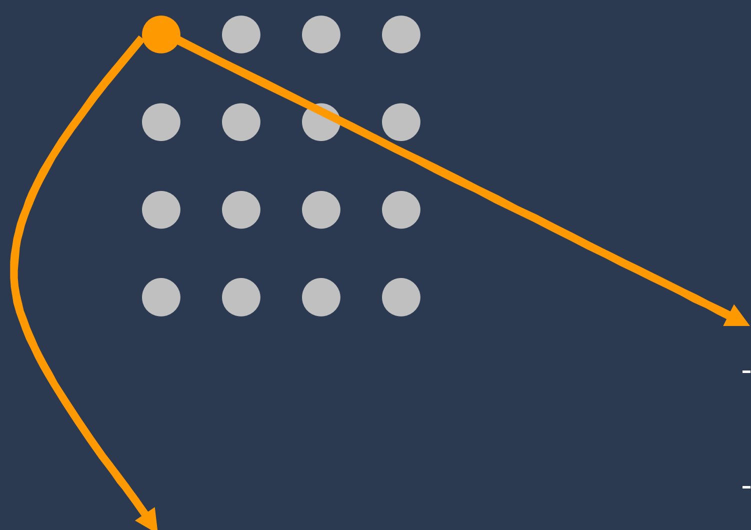
● → Unknown pixel

A matrix is 16 by 16

2	-1	0	0	-1	0	0	0	0	0	0	0	0	0	0	0
-1	3	-1	0	0	-1	0	0	0	0	0	0	0	0	0	0
0	-1	3	-1	0	0	-1	0	0	0	0	0	0	0	0	0
0	0	-1	2	0	0	0	-1	0	0	0	0	0	0	0	0
-1	0	0	0	3	-1	0	0	-1	0	0	0	0	0	0	0
0	-1	0	0	-1	4	-1	0	0	-1	0	0	0	0	0	0
0	0	-1	0	0	-1	4	-1	0	0	-1	0	0	0	0	0
0	0	0	-1	0	0	-1	3	0	0	0	-1	0	0	0	0
0	0	0	0	-1	0	0	0	3	-1	0	0	-1	0	0	0
0	0	0	0	0	-1	0	0	-1	4	-1	0	0	-1	0	0
0	0	0	0	0	0	-1	0	0	-1	4	-1	0	0	-1	0
0	0	0	0	0	0	0	-1	0	0	-1	3	0	0	0	-1
0	0	0	0	0	0	0	0	-1	0	0	0	2	-1	0	0
0	0	0	0	0	0	0	0	0	-1	0	0	-1	3	-1	0
0	0	0	0	0	0	0	0	0	0	-1	0	0	-1	3	-1
0	0	0	0	0	0	0	0	0	0	0	-1	0	0	-1	2

Effect of Boundary Conditions

- Toy case: image 4*4 pixels



Has 2 neighbors

Neumann Boundary Condition

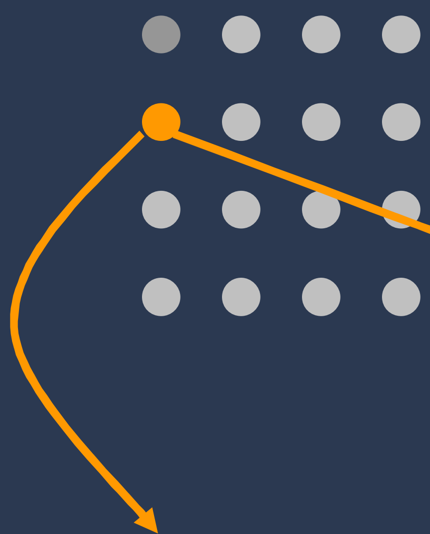
known pixel

Unknown pixel

2	-1	0	0	-1	0	0	0	0	0	0	0	0	0	0	0
-1	3	-1	0	0	-1	0	0	0	0	0	0	0	0	0	0
0	-1	3	-1	0	0	-1	0	0	0	0	0	0	0	0	0
0	0	-1	2	0	0	0	-1	0	0	0	0	0	0	0	0
-1	0	0	0	3	-1	0	0	-1	0	0	0	0	0	0	0
0	-1	0	0	-1	4	-1	0	0	-1	0	0	0	0	0	0
0	0	-1	0	0	-1	4	-1	0	0	-1	0	0	0	0	0
0	0	0	-1	0	0	-1	3	0	0	0	-1	0	0	0	0
0	0	0	0	-1	0	0	0	3	-1	0	0	-1	0	0	0
0	0	0	0	0	-1	0	0	-1	4	-1	0	0	-1	0	0
0	0	0	0	0	0	-1	0	0	-1	4	-1	0	0	-1	0
0	0	0	0	0	0	0	-1	0	0	-1	3	0	0	0	-1
0	0	0	0	0	0	0	0	-1	0	0	0	2	-1	0	0
0	0	0	0	0	0	0	0	0	-1	0	0	-1	3	-1	0
0	0	0	0	0	0	0	0	0	0	-1	0	0	-1	3	-1
0	0	0	0	0	0	0	0	0	0	0	-1	0	0	-1	2

Effect of Boundary Conditions

- Toy case: image 4*4 pixels



Has 3 neighbors

Neumann Boundary Condition

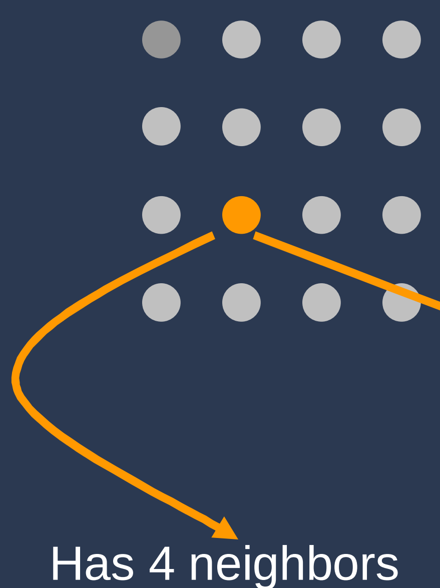
known pixel

Unknown pixel

2	-1	0	0	-1	0	0	0	0	0	0	0	0	0	0	0
-1	3	-1	0	0	-1	0	0	0	0	0	0	0	0	0	0
0	-1	3	-1	0	0	-1	0	0	0	0	0	0	0	0	0
0	0	-1	2	0	0	0	-1	0	0	0	0	0	0	0	0
-1	0	0	0	3	-1	0	0	-1	0	0	0	0	0	0	0
0	-1	0	0	-1	4	-1	0	0	-1	0	0	0	0	0	0
0	0	-1	0	0	-1	4	-1	0	0	-1	0	0	0	0	0
0	0	0	-1	0	0	-1	3	0	0	0	-1	0	0	0	0
0	0	0	0	-1	0	0	0	3	-1	0	0	-1	0	0	0
0	0	0	0	0	-1	0	0	-1	4	-1	0	0	-1	0	0
0	0	0	0	0	0	-1	0	0	-1	4	-1	0	0	-1	0
0	0	0	0	0	0	0	-1	0	0	-1	3	0	0	0	-1
0	0	0	0	0	0	0	0	-1	0	0	0	2	-1	0	0
0	0	0	0	0	0	0	0	0	-1	0	0	-1	3	-1	0
0	0	0	0	0	0	0	0	0	0	-1	0	0	-1	3	-1
0	0	0	0	0	0	0	0	0	0	0	-1	0	0	-1	2

Effect of Boundary Conditions

- Toy case: image 4*4 pixels



Neumann Boundary Condition

● → known pixel

● → Unknown pixel

2	-1	0	0	-1	0	0	0	0	0	0	0	0	0	0	0
-1	3	-1	0	0	-1	0	0	0	0	0	0	0	0	0	0
0	-1	3	-1	0	0	-1	0	0	0	0	0	0	0	0	0
0	0	-1	2	0	0	0	-1	0	0	0	0	0	0	0	0
-1	0	0	0	3	-1	0	0	-1	0	0	0	0	0	0	0
0	-1	0	0	-1	4	-1	0	0	-1	0	0	0	0	0	0
0	0	-1	0	0	-1	4	-1	0	0	-1	0	0	0	0	0
0	0	0	-1	0	0	-1	3	0	0	0	-1	0	0	0	0
0	0	0	0	-1	0	0	0	3	-1	0	0	-1	0	0	0
0	0	0	0	0	-1	0	0	-1	4	-1	0	0	-1	0	0
0	0	0	0	0	0	-1	0	0	-1	4	-1	0	0	-1	0
0	0	0	0	0	0	0	-1	0	0	-1	3	0	0	0	-1
0	0	0	0	0	0	0	0	-1	0	0	0	2	-1	0	0
0	0	0	0	0	0	0	0	0	-1	0	0	-1	3	-1	0
0	0	0	0	0	0	0	0	0	0	-1	0	0	-1	3	-1
0	0	0	0	0	0	0	0	0	0	0	-1	0	0	-1	2

Effect of Boundary Conditions

- Dirichlet
- Size of A = (N-2)*(N-2)

-4	1	1	0
1	-4	0	1
1	0	-4	1
0	1	1	-4
.	.	.	.

- Neumann
- Size of A = N*N
- Unknown additive constant

2	-1	0	0	-1	0	0	0	0	0	0	0	0	0	0	0
-1	3	-1	0	0	-1	0	0	0	0	0	0	0	0	0	0
0	-1	3	-1	0	0	-1	0	0	0	0	0	0	0	0	0
0	0	-1	2	0	0	0	-1	0	0	0	0	0	0	0	0
-1	0	0	0	3	-1	0	0	-1	0	0	0	0	0	0	0
0	-1	0	0	-1	4	-1	0	0	-1	0	0	0	0	0	0
0	0	-1	0	0	-1	4	-1	0	0	-1	0	0	0	0	0
0	0	0	-1	0	0	-1	3	0	0	0	-1	0	0	0	0
0	0	0	0	-1	0	0	0	3	-1	0	0	-1	0	0	0
0	0	0	0	0	-1	0	0	-1	4	-1	0	0	-1	0	0
0	0	0	0	0	0	-1	0	0	-1	4	-1	0	0	-1	0
0	0	0	0	0	0	0	-1	0	0	-1	3	0	0	0	-1
0	0	0	0	0	0	0	0	-1	0	0	0	2	-1	0	0
0	0	0	0	0	0	0	0	0	-1	0	0	-1	3	-1	0
0	0	0	0	0	0	0	0	0	0	-1	0	0	-1	3	-1
0	0	0	0	0	0	0	0	0	0	0	-1	0	0	-1	2

Solving Linear System

- Image size N*N
- Size of A ~ N^2 by N^2
- Impractical to form and store A

- Direct Solvers
- Basis Functions
- Multigrid
- Conjugate Gradients

Direct Solvers

- Extremely fast
- Single iteration
- No convergence issues
- No magic numbers
- Best approach for solving Poisson equation *if*
  - Rectangular boundary & no other constraints
- <3 sec, 1M image, Matlab

- Basic Idea
  - Decompose $A = PDP^T$
  - For Dirichlet, sine functions diagonalize A
  - For Neumann, cosine functions diagonalize A

Direct Solvers

- $A = PDP^T$
- Eigen Values of A (Neumann)

$$\lambda(x, y) = 2\cos\left(\frac{\pi x}{W}\right) + 2\cos\left(\frac{\pi y}{H}\right) - 4$$

- Eigen Values of A (Dirichlet)

$$\lambda(x, y) = 2\cos\left(\frac{\pi x}{W - 1}\right) + 2\cos\left(\frac{\pi y}{H - 1}\right) - 4$$

Eigen value only depends on image size!!

Solve $\nabla^2 I = div(G_x, G_y) = u(x, y)$

- Dirichlet
- Compute 2D sine transform of u(x,y)
- Divide by eigen values
- Compute 2D inverse sine transform

$$I(x, y) = \mathrm{idst2}\left(\frac{\mathrm{dst2}(u(x, y))}{\lambda(x, y)}\right)$$

- Neumann
- Compute cosine transform of u(x,y)
- Divide by eigen values (except DC)
- Compute inverse cosine transform

$$I(x, y) = \mathrm{idct2}\left(\frac{\mathrm{dct2}(u(x, y))}{\lambda(x, y)}\right)$$

C and Matlab Code available at http://www.merl.com/people/agrawal

Extension to higher dimensions

- Simple
- For 3D, eigen values of A are

$$\lambda(x, y, t) = 2\cos\left(\frac{\pi x}{W}\right) + 2\cos\left(\frac{\pi y}{H}\right) + 2\cos\left(\frac{\pi t}{T}\right) - 6$$

Solve $\nabla^2 I = div(G_x, G_y, G_t) = u(x, y, t)$

$$I(x, y, t) = \mathrm{idct3}\left(\frac{\mathrm{dct3}(u(x, y, t))}{\lambda(x, y, t)}\right)$$

Multigrid Methods

- Direct solvers
  - Special elliptic PDE's, O(Nlog(N))
- Multigrid O(N)
  - More general
  - Non-constant coefficients
  - Non-linear systems
- Fedorenko, 1961
- Brandt, 1963
- 1970's Hackbusch

Iterative Solvers: Smoothers

- Solve Ax = b

$$x^{i+1} = x^i + M^{-1}(b - Ax^i)$$

- A = L + D + U

Lower Triangular

Diagonal

Upper Triangular

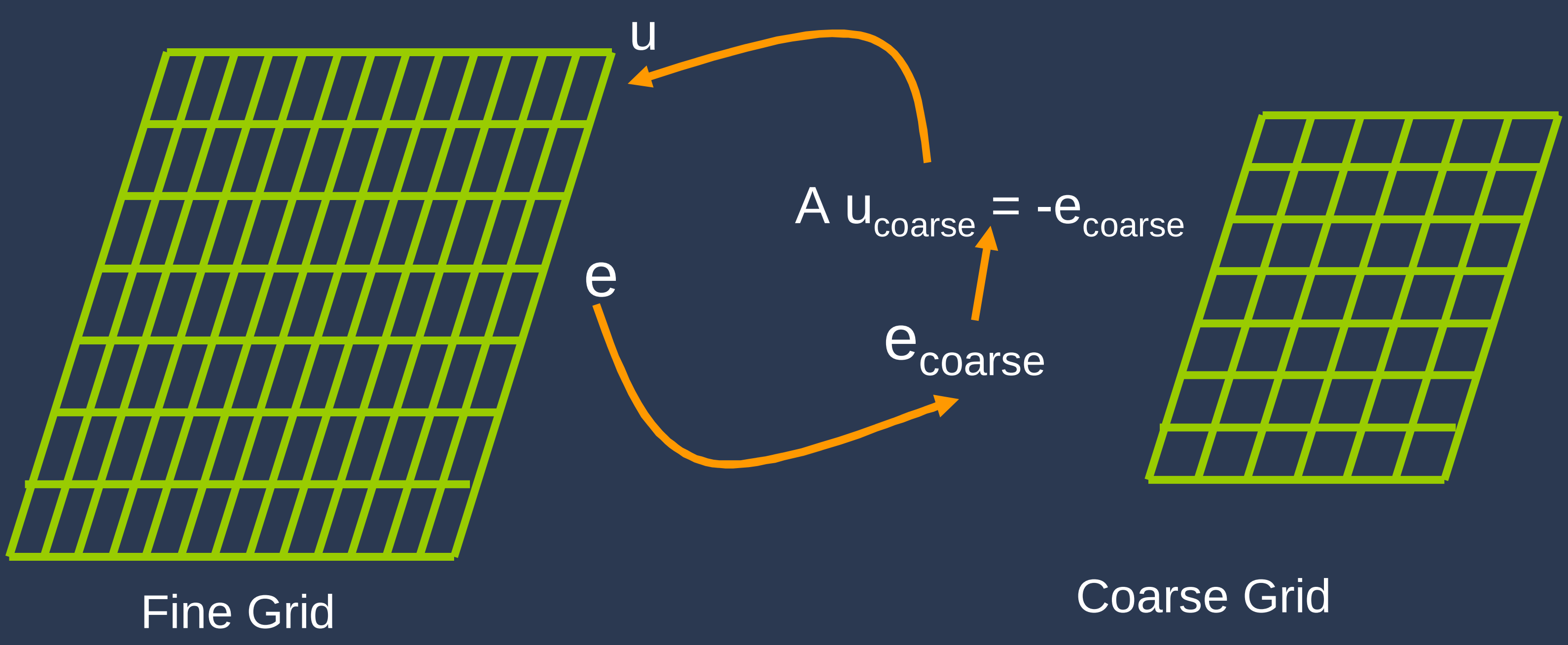
- M = D (Jacobi)
- M = D + L (Gauss-Siedel)
- Slow convergence, low frequency error are reduced slowly

Multigrid: Key idea

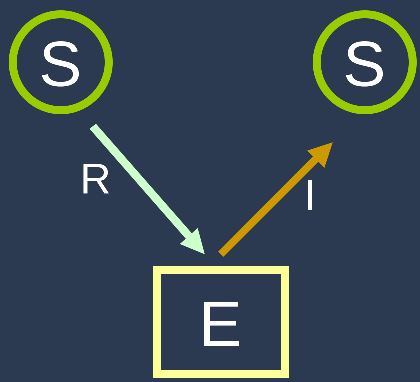
- Faster Convergence
- Replace problem on fine grid by an approximation on a coarser grid
- Solve the coarse grid problem and use as initial solution
- Recursive
- Low freuqency errors are reduced by coarse grid correction

Two grid solution

- Solve $Ax = b$, x_0 initial solution
- Compute error $e = b - Ax_0$ at fine grid
- Restrict on coarse grid
- Solve for correction u_{coarse}
- Interpolate correction to fine grid
- $x_1 = x_0 + u$



Combine smoothing and Coarse grid correction



S: Smoothing

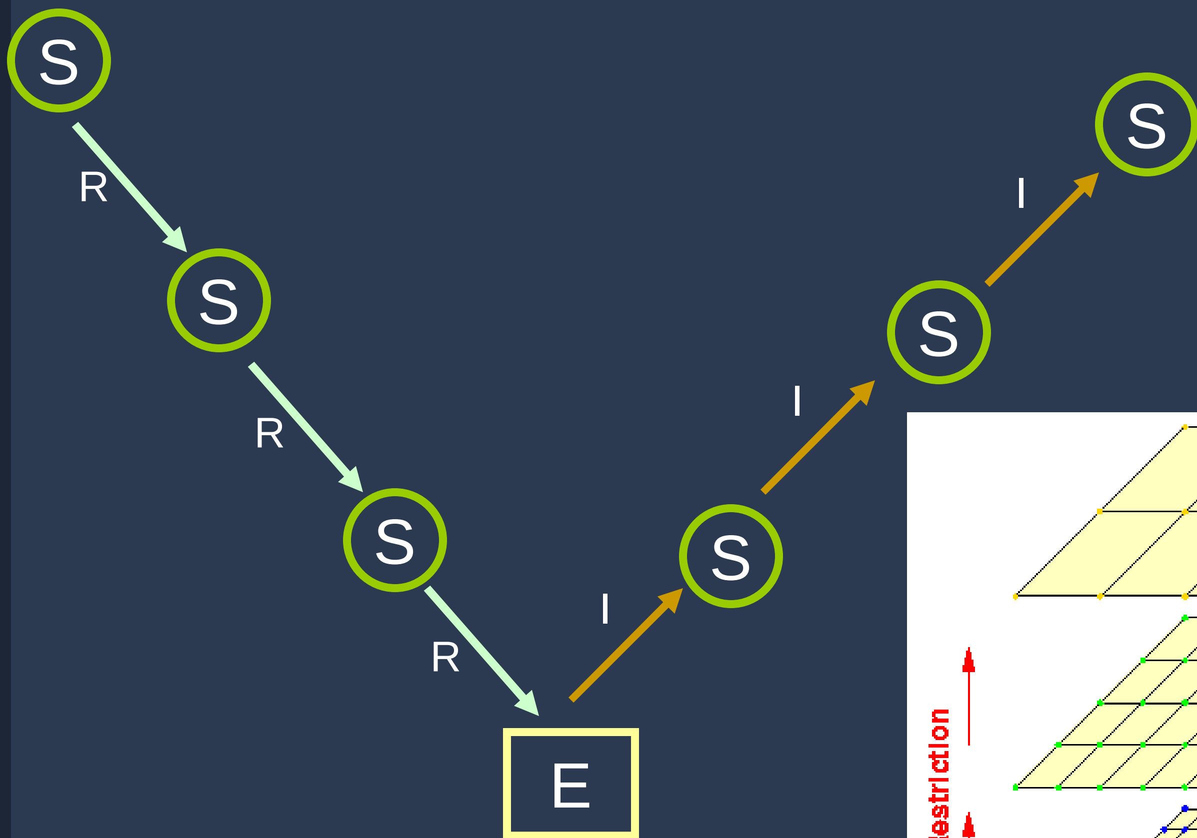
E: Exact Solution

R: Restrict

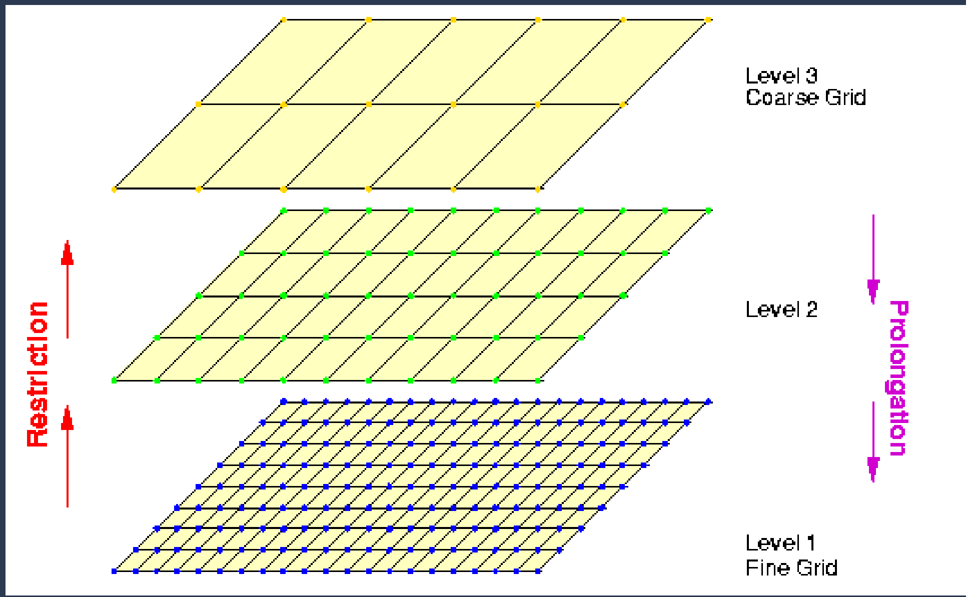
I: Interpolate

Multigrid

- Use coarser grid recursively



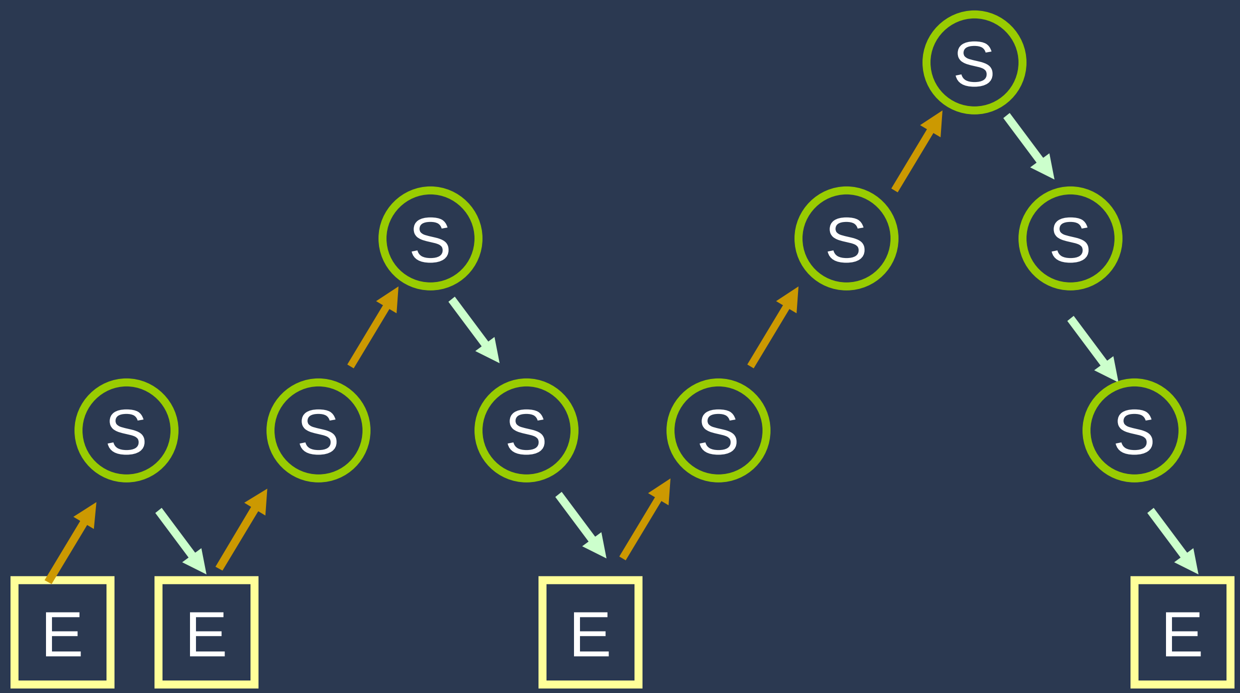
V Cycle



Full Multigrid

- Multigrid
  - Start at finest grid, approximate solution
  - Needs b (Ax=b) only at finest level
- Full Multigrid
  - Find b at all levels
  - Start at coarsest level and move up
  - Faster convergence

Full Multigrid



Algorithms for solving Poisson Equation

Algorithm	2D ($n = N^2$)
• Explicit Inv.	n^2
• Jacobi/GS	n^2
• Conj.Grad.	$n^{3/2}$
• FFT	$n*\log n$
• Multigrid	n
• Lower bound	n

Multigrid is much more general than FFT approach (many elliptic PDE)

Preconditioning

- Solve Ax = b
- Improve efficiency and robustness

- Multiply by some 'pre-conditioner' M
  - $M^{-1}Ax = M^{-1}b$
  - Might be easier to solve
  - Better condition number than original linear system
  - Good choice depends on the problem

- Recap
  - For Direct solvers, M is sine (cosine) functions
  - Exactly diagonalize A. Perfect Preconditioner

Hierarchical Basis Preconditioning

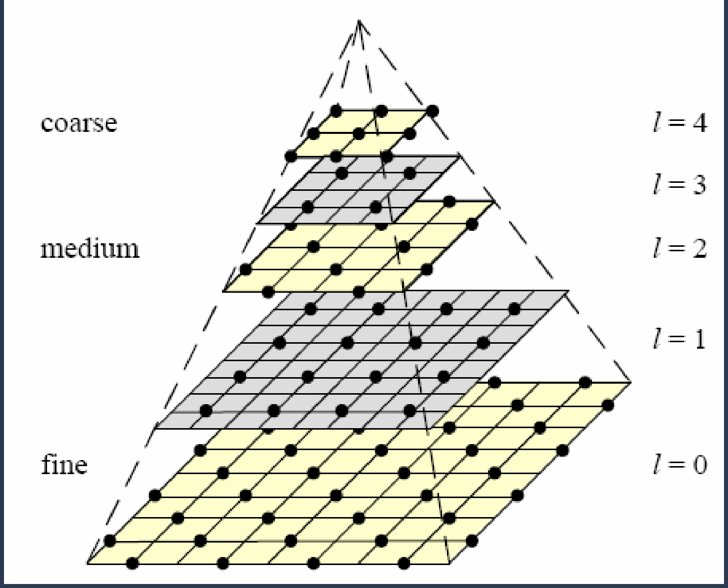
- Use hierarchical basis as preconditioners
  - Surface Interpolation
    - Szeliski PAMI'90, Triangular functions
    - Yaou and Chang PAMI'94, Wavelets
    - Pentland 94, Wavelets
  - Geometric Modeling
    - Gortler and Cohen 95
  - Shape from Shading
    - Szeliski, CVGIP'91

- Key idea
  - Larger support of basis functions
  - Updates can be propagated fast

Hierarchical Basis Preconditioning

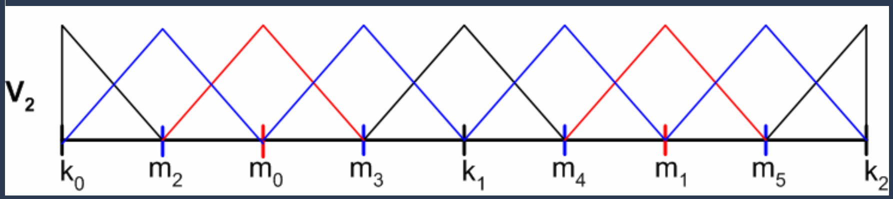
- Solve Ax = b
- Substitute x = Sy
  - Columns of S are basis functions
  - Solve $S^TAS\ y = S^Tb$

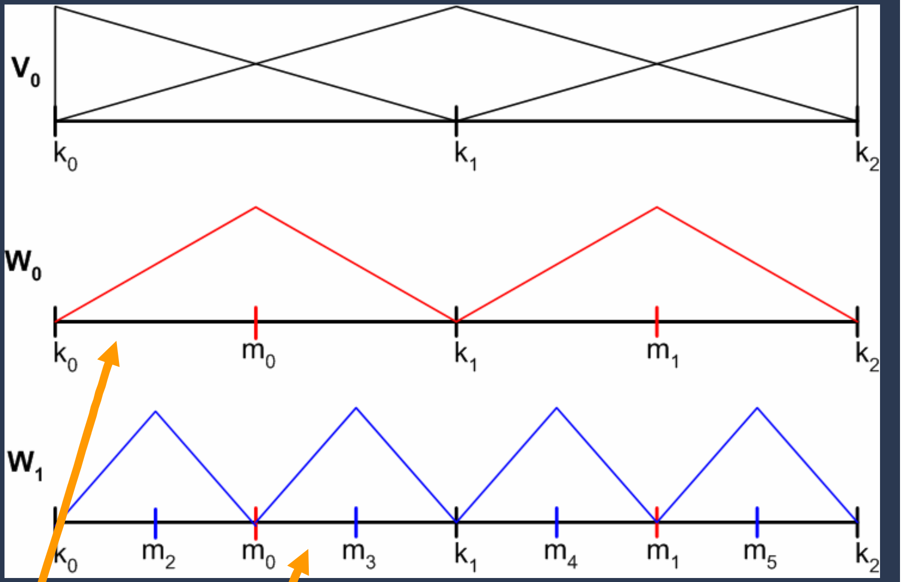
Better condition number



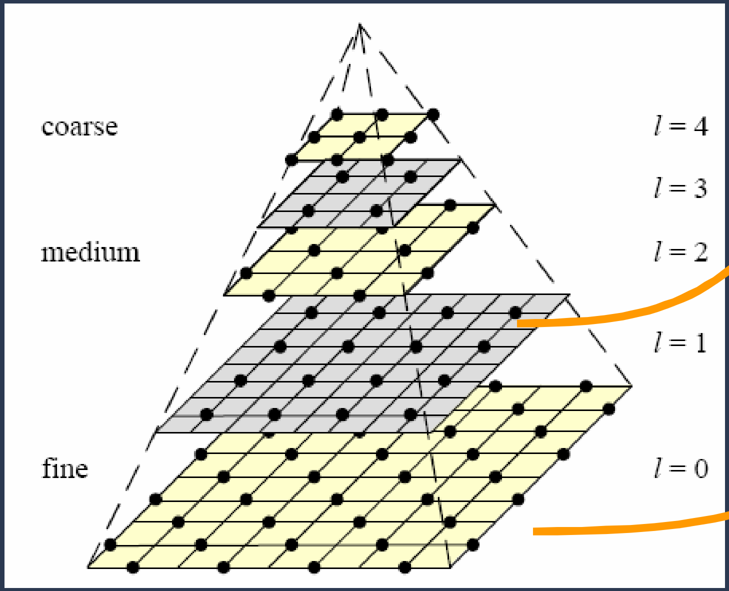
Szeliski 1990

Nodal Basis





Hierarchical Basis



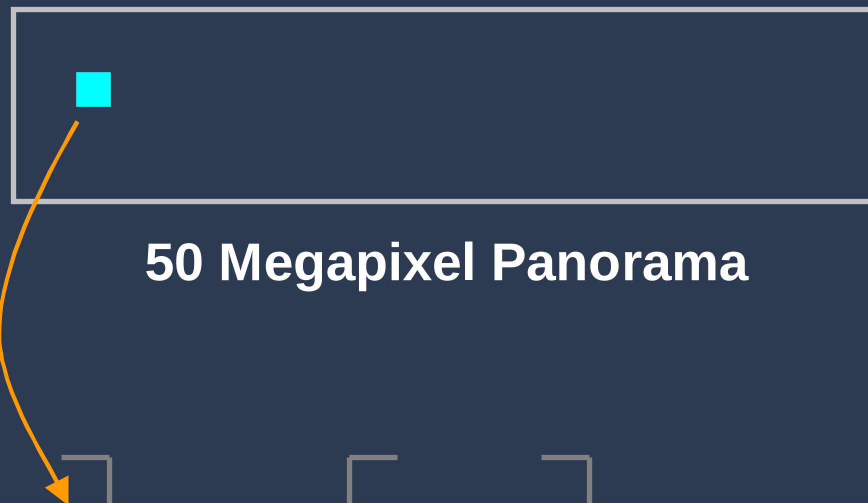
Finest level

Approximate Solution for Large Scale Problems

- Resolution is increasing in digital cameras
- Stitching, Alignment requires solving large linear system

Scalability problem

10 X 10 MP X 50% overlap =



50 Megapixel Panorama

$$\begin{bmatrix} & A & \end{bmatrix} \begin{bmatrix} x \end{bmatrix} = \begin{bmatrix} b \end{bmatrix}$$

Scalability problem

$$\begin{bmatrix} \cancel{A} \end{bmatrix} \begin{bmatrix} x \end{bmatrix} = \begin{bmatrix} b \end{bmatrix}$$

50 million element vectors!

Approximate Solution

- Reduce size of linear system
- Handle high resolution images
- Part of Photoshop CS3

Aseem Agarwala. "Efficient gradient-domain compositing using quadtrees," ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)

The key insight

Desired solution x

—

Initial Solution x_0

=

Difference x_δ

$$Ax = b$$

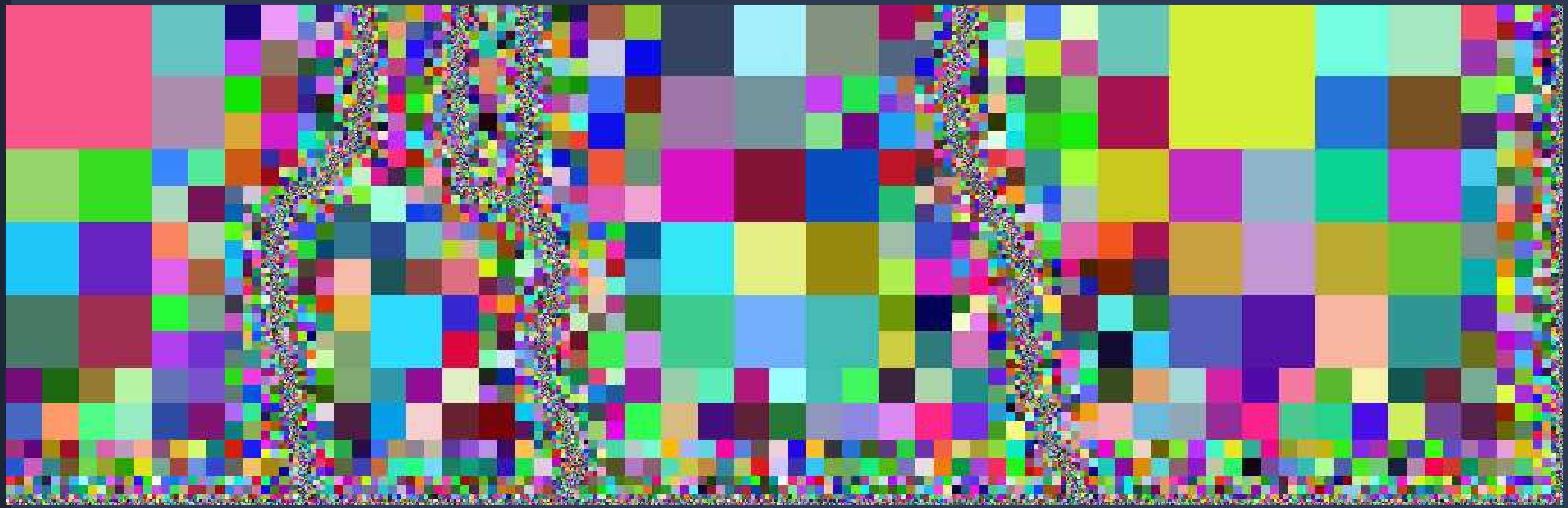
$$A(x_0 + x_\delta) = b$$

$$Ax_\delta = b - Ax_0$$

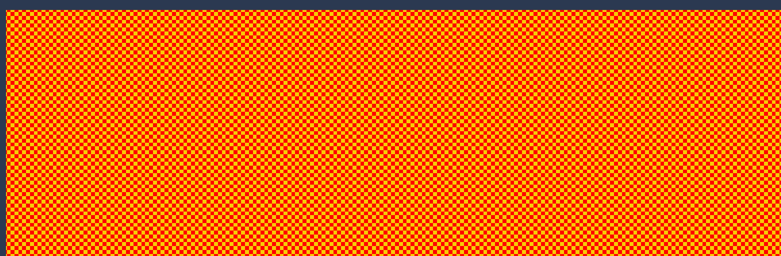
$$A^T A x_\delta = A^T(\boxed{b - Ax_0})$$

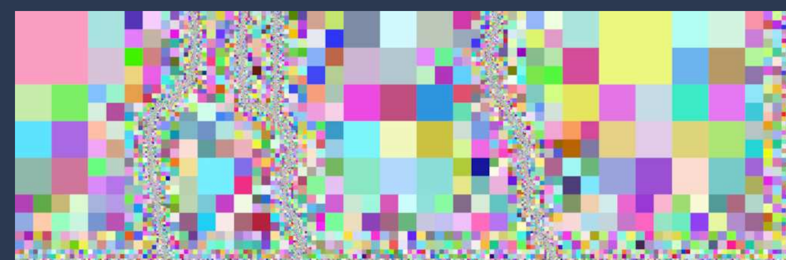
Away from seams, $A^T A x_\delta = 0$

Quadtree decomposition



Reduced space





x

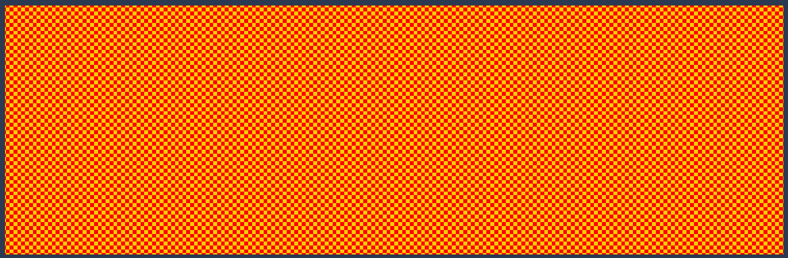
n variables

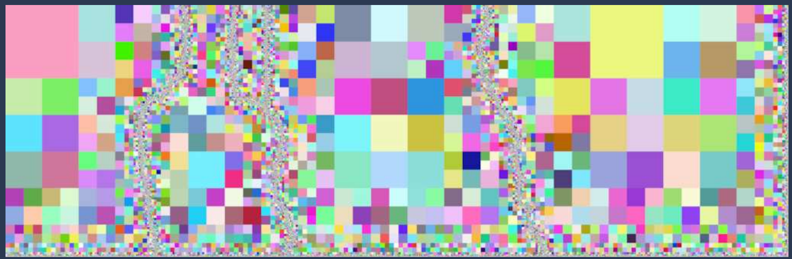
y

m variables

$$m << n$$

Reduced space

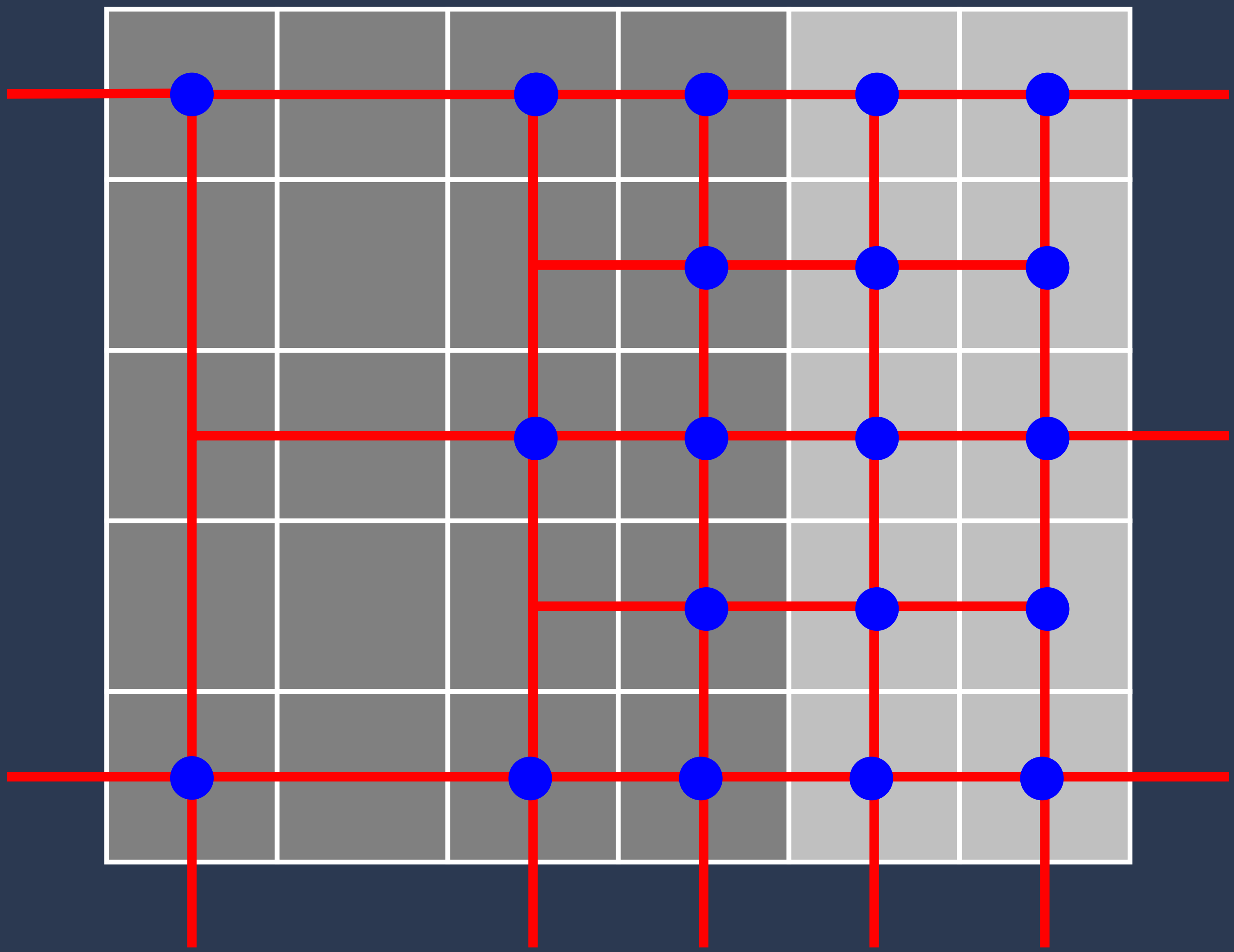




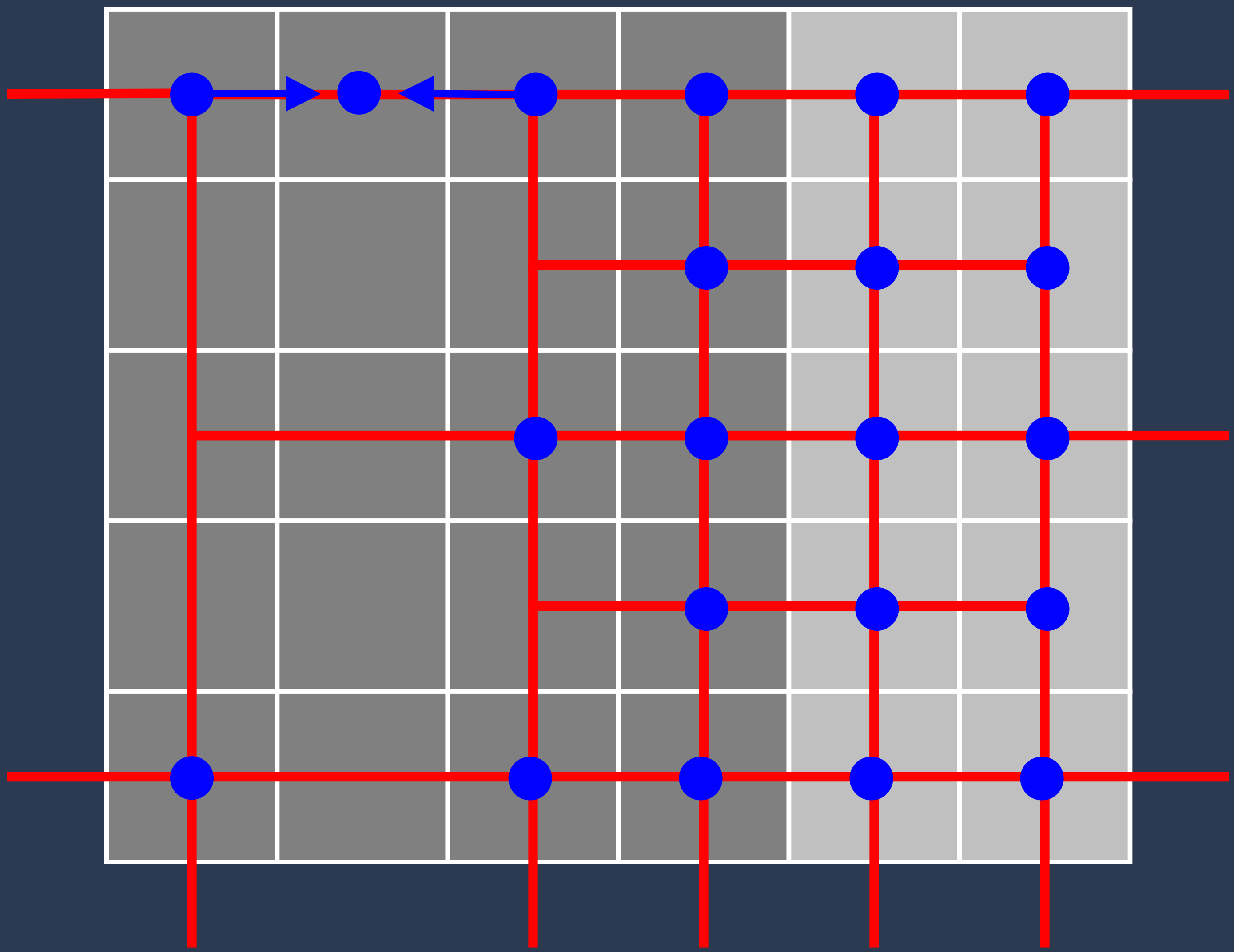
x
n variables

y
m variables

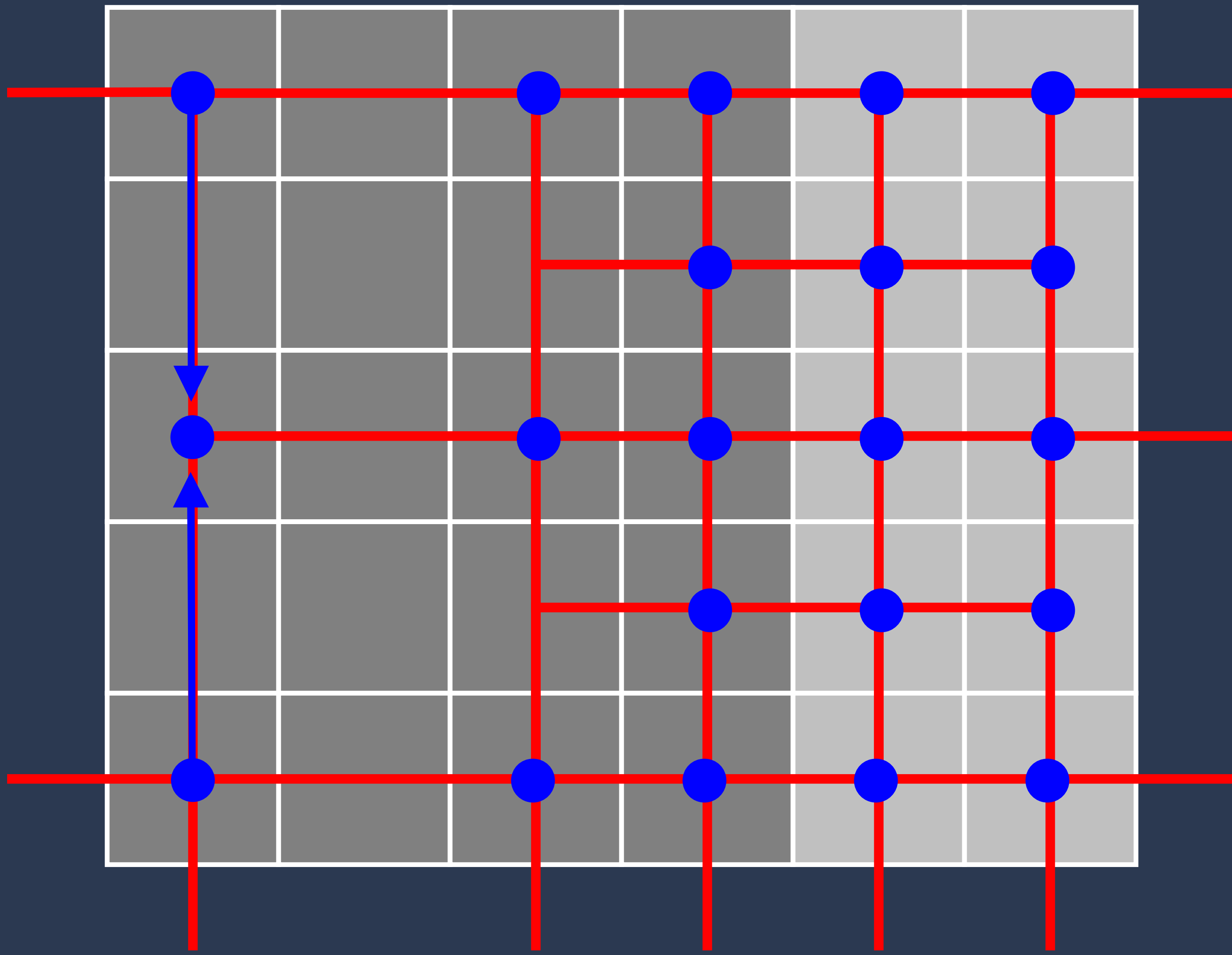
$$x = Sy$$



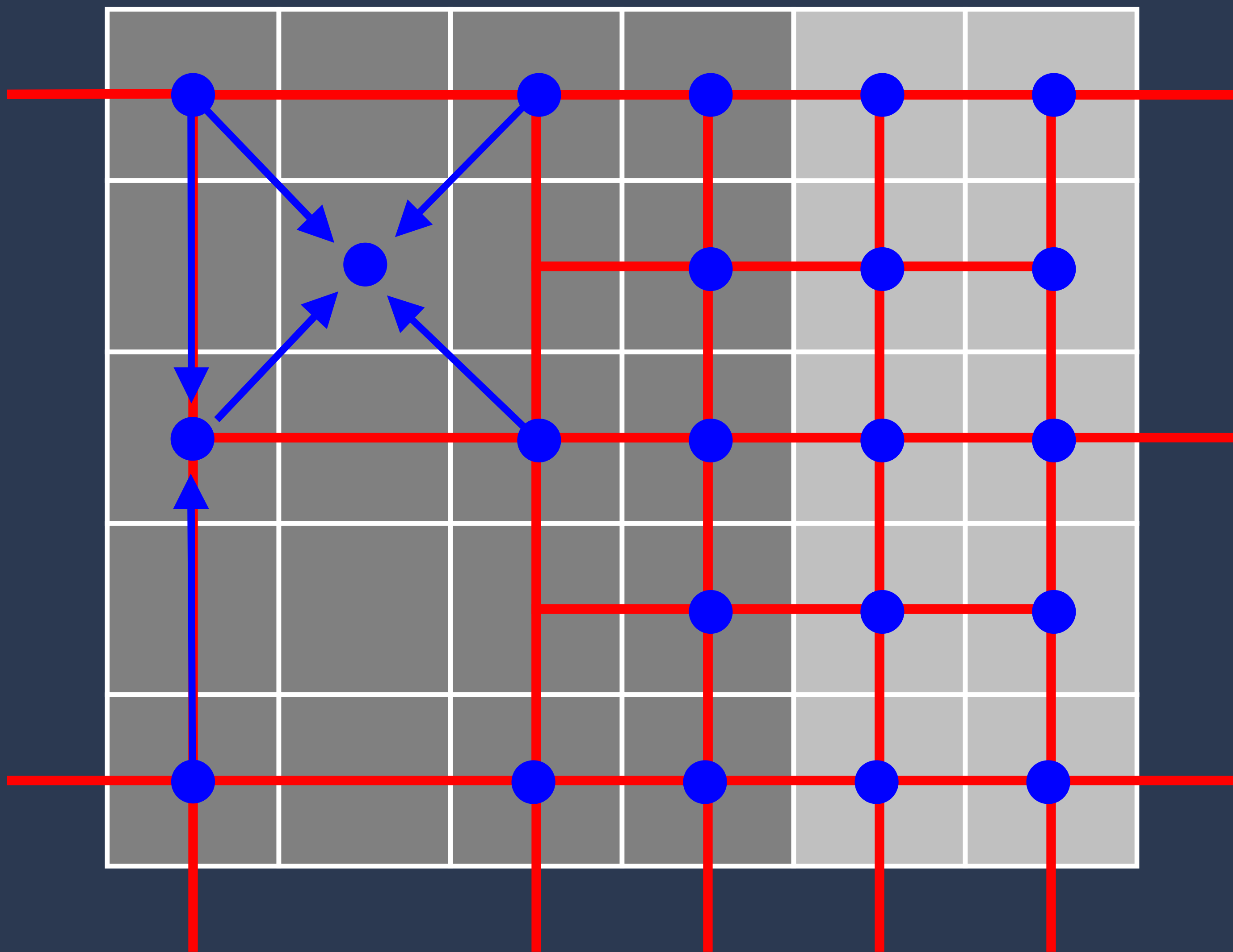
$$x = Sy$$



$$x = Sy$$

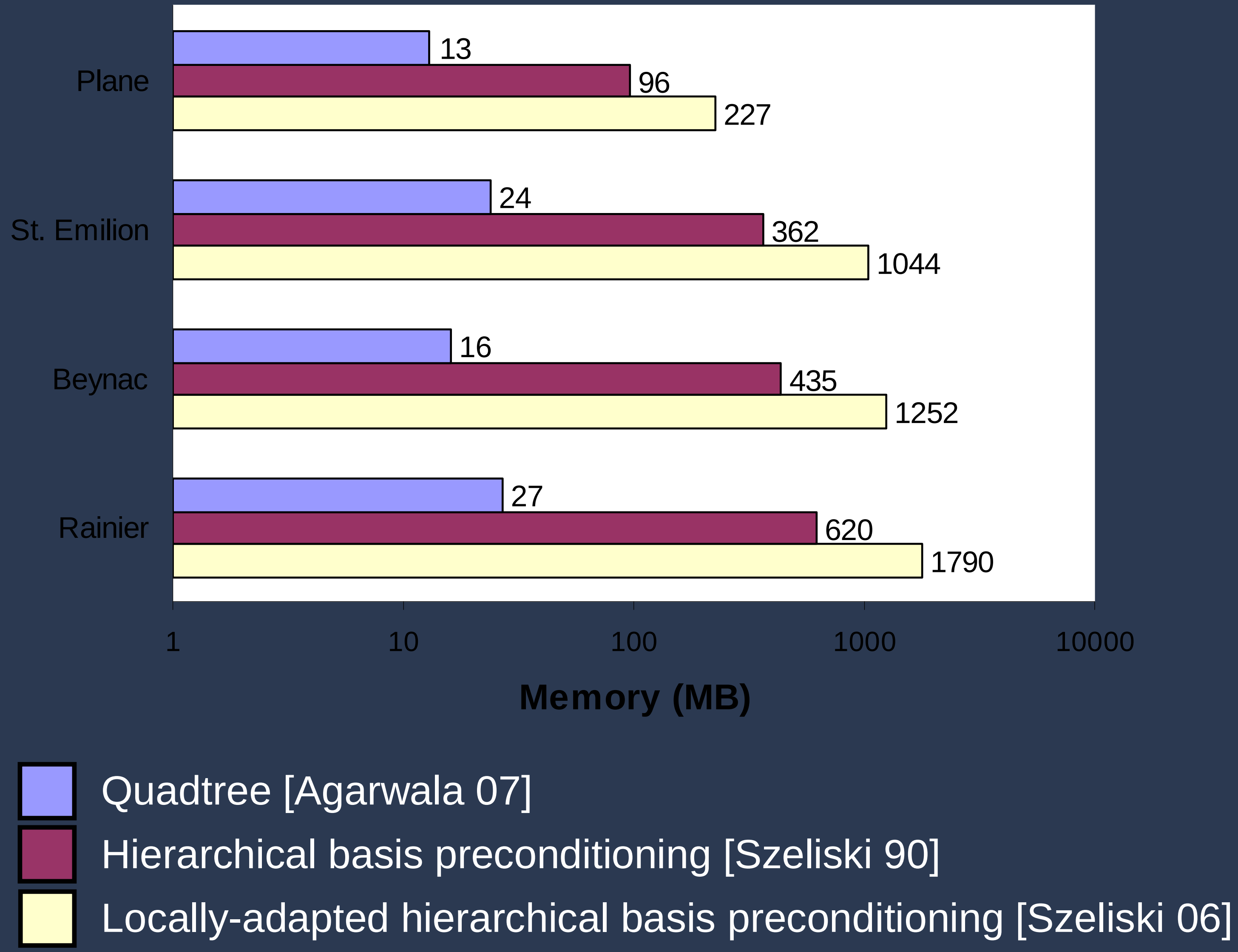


$$x = Sy$$



$$x = Sy$$

Performance



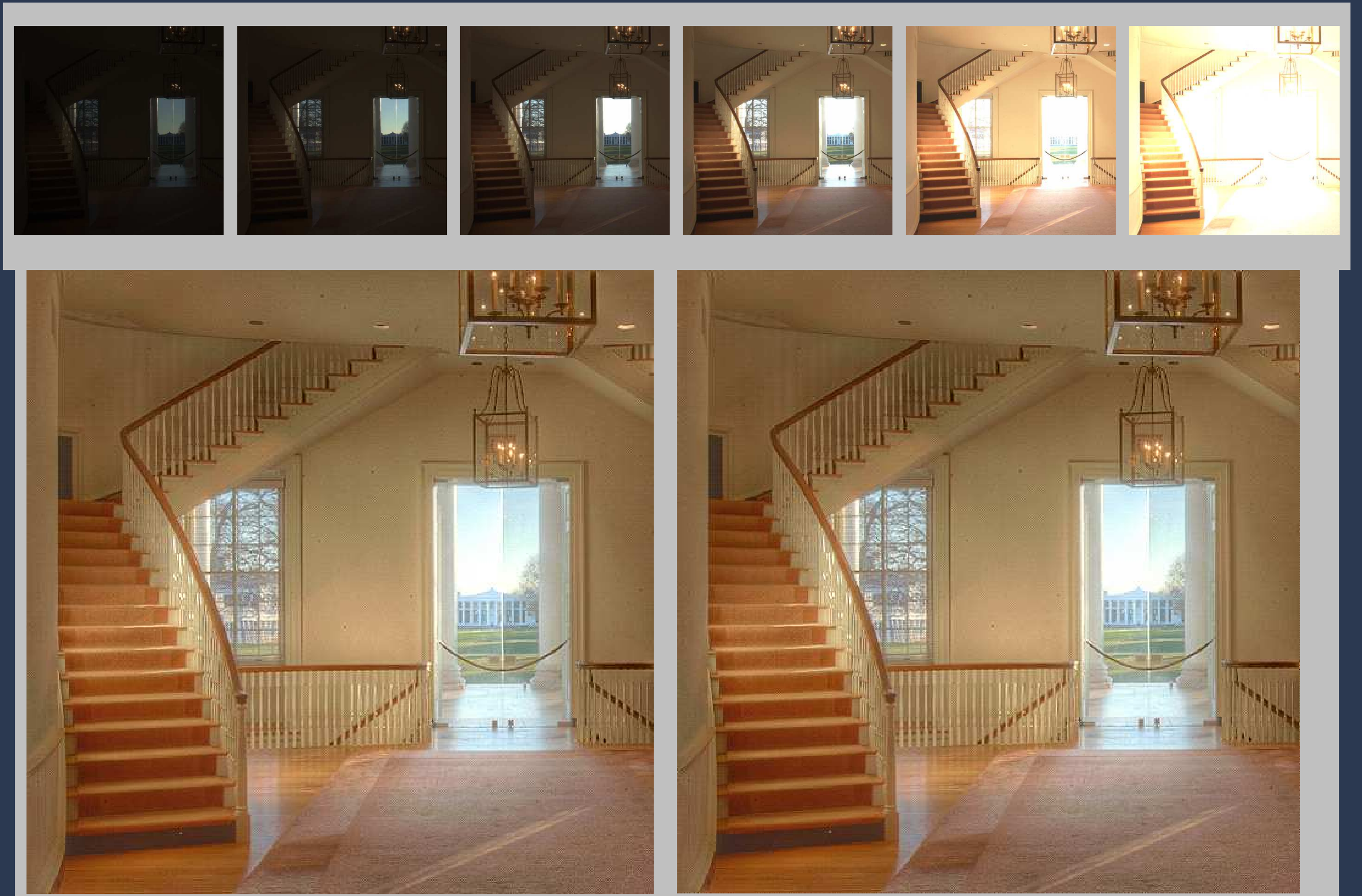
Cut-and-paste

Cut-and-paste

GPU implementations

- J. Bolz, I. Farmer, E. Grinspun, P. Schroder, “Sparse matrix solvers on the GPU: Conjugate gradients and Multigrid”. TOG 22 (2003), 917–924
- N. Goodnight, C. Woolley, G. Lewin, D. Luebke, G. Humphreys, “A multigrid solver for boundary value problems using programmable graphics hardware”. In Graphics Hardware (2003), 102–111.

Nolan Goodnight, Cliff Woolley, Gregory Lewin, David Luebke, and Greg Humphreys, A Multigrid Solver for Boundary Value Problems Using Programmable Graphics Hardware, Graphics Hardware 2003



CPU

GPU

Summary

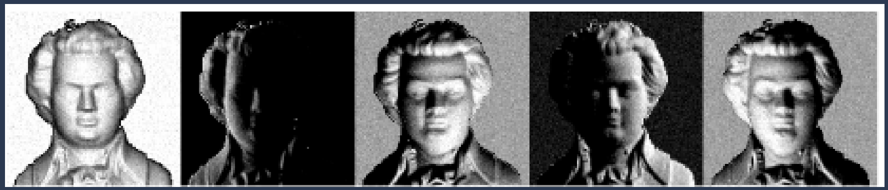
- Poisson solver requires solving large sparse linear system
- Direct Solvers
  - Specific, rectangular domain, fast, single iteration O(Nlog(N))
- Multigrid
  - O(N), general purpose, may need fine tuning
- Conjugate Gradients
  - General, (A should be positive definite)
  - Preconditioning can improve performance
- Preconditioning
  - Incomplete LU factorization etc., general but slow
  - Hierarchical Basis, Wavelets, works well for vision problems
  - Locally adaptive hierarchical basis, general, improves
- Approximate Solution
  - Quadtree, O(sqrt(n)), but only for special cases (image stitching)

Understanding Poisson Solver

- We only talked about solving as a least square problem
  - Minimizing L2 norm
- Are there other solutions?
- How do we get other meaningful solutions?

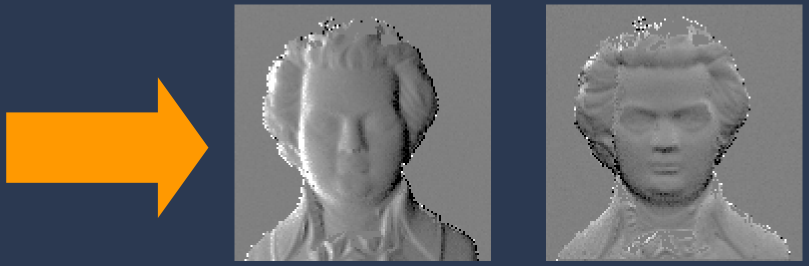
Example Application: Photometric Stereo

- Multiple images, varying illumination
- Obtain surface gradient field from images
  - Lambertian reflectance model

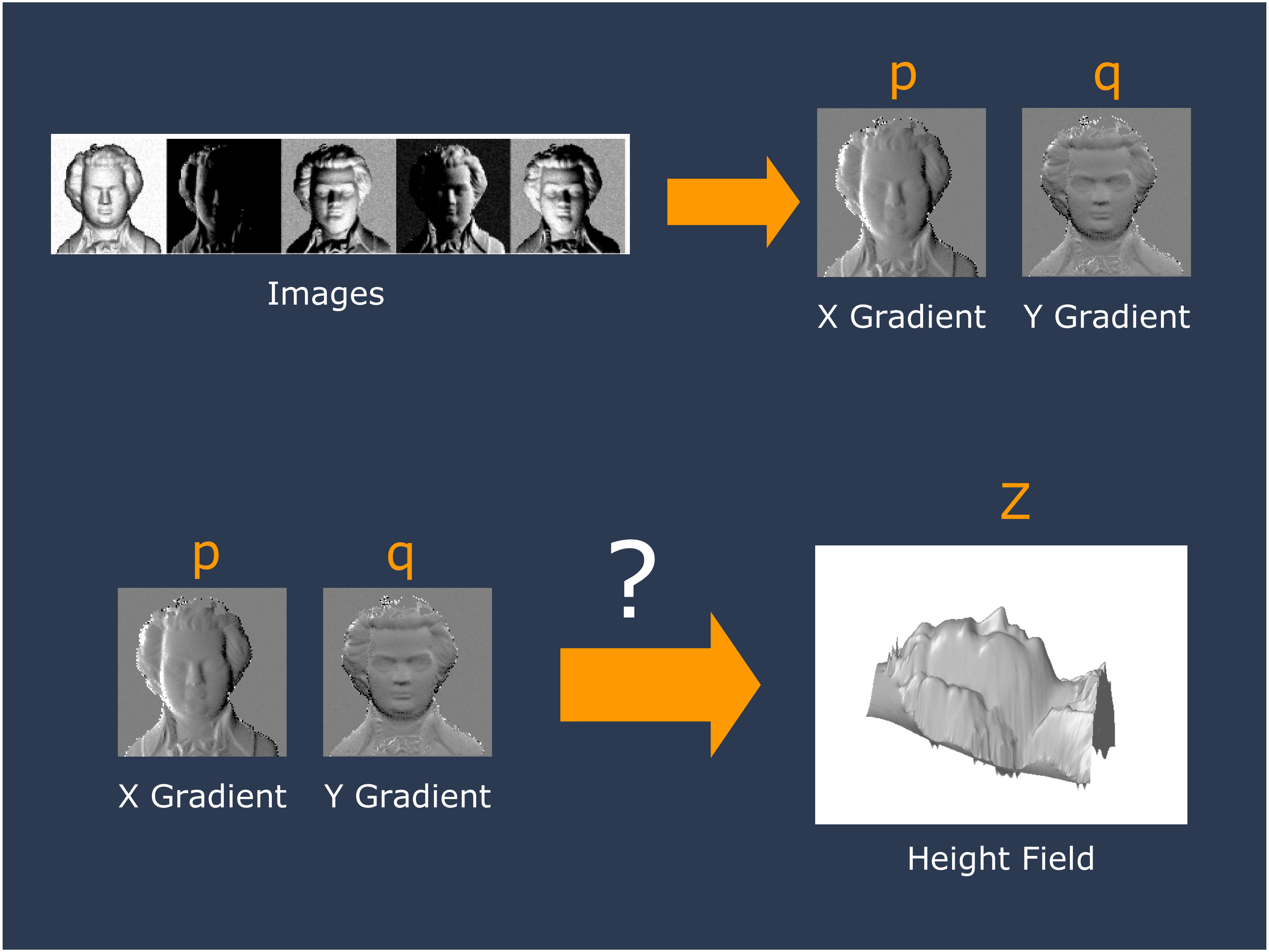


Images

p q

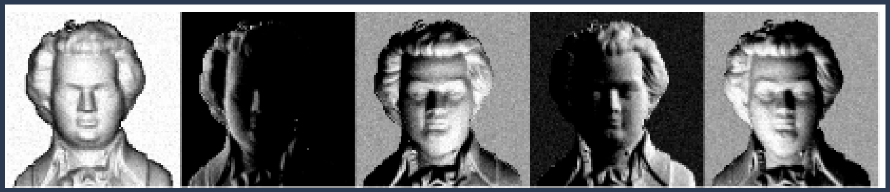


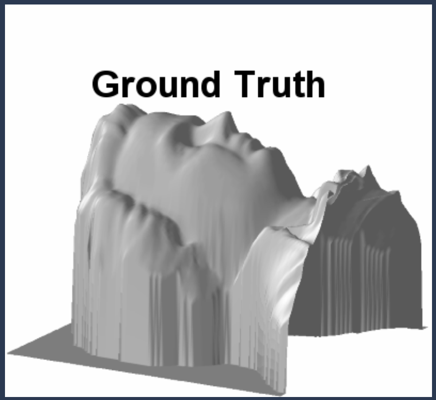
X Gradient Y Gradient

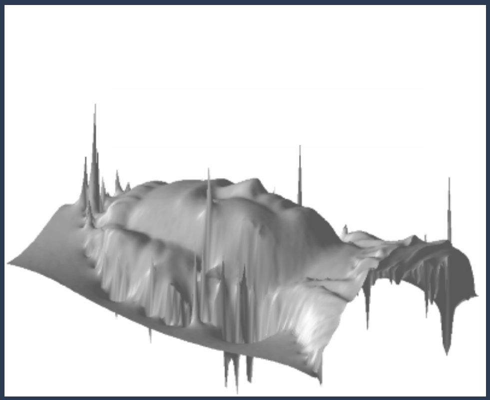

Images
p
q
X Gradient
Y Gradient
p
q
?
Z
X Gradient
Y Gradient
Height Field

Motivation

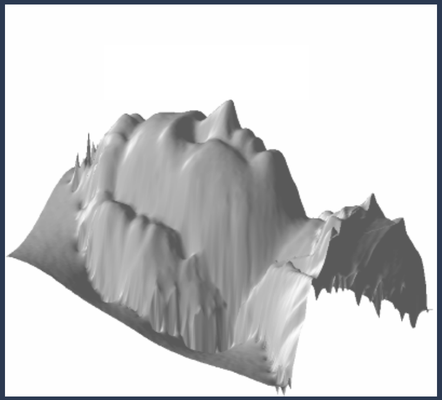
- Reconstruction
  - Feature preserving rather than smooth solution
  - Handle outliers





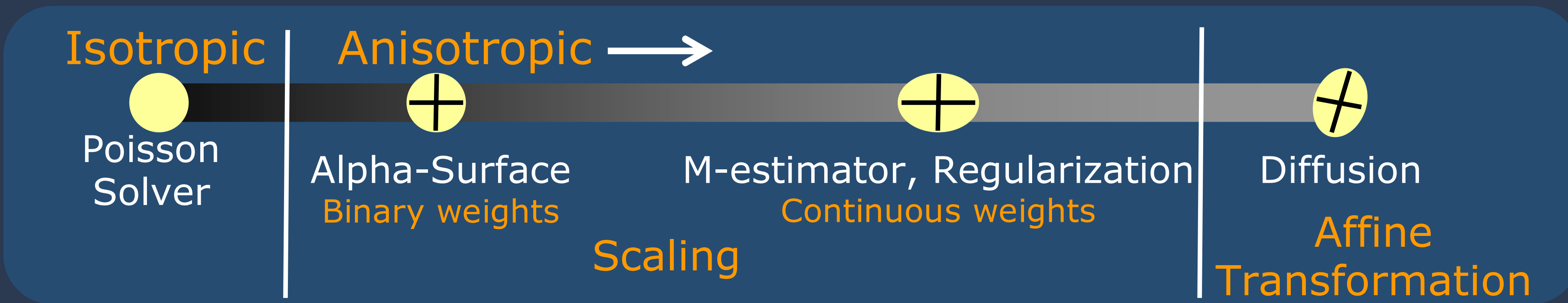


MSE=2339.2
Least Square approach

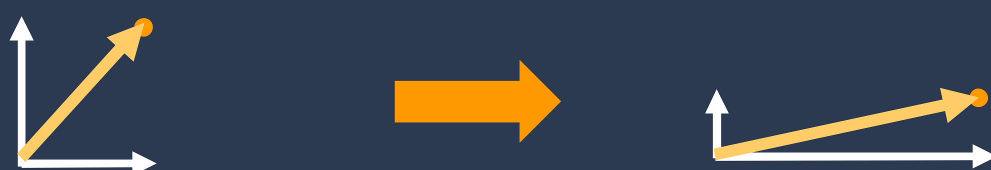


MSE=373.72
Our approach

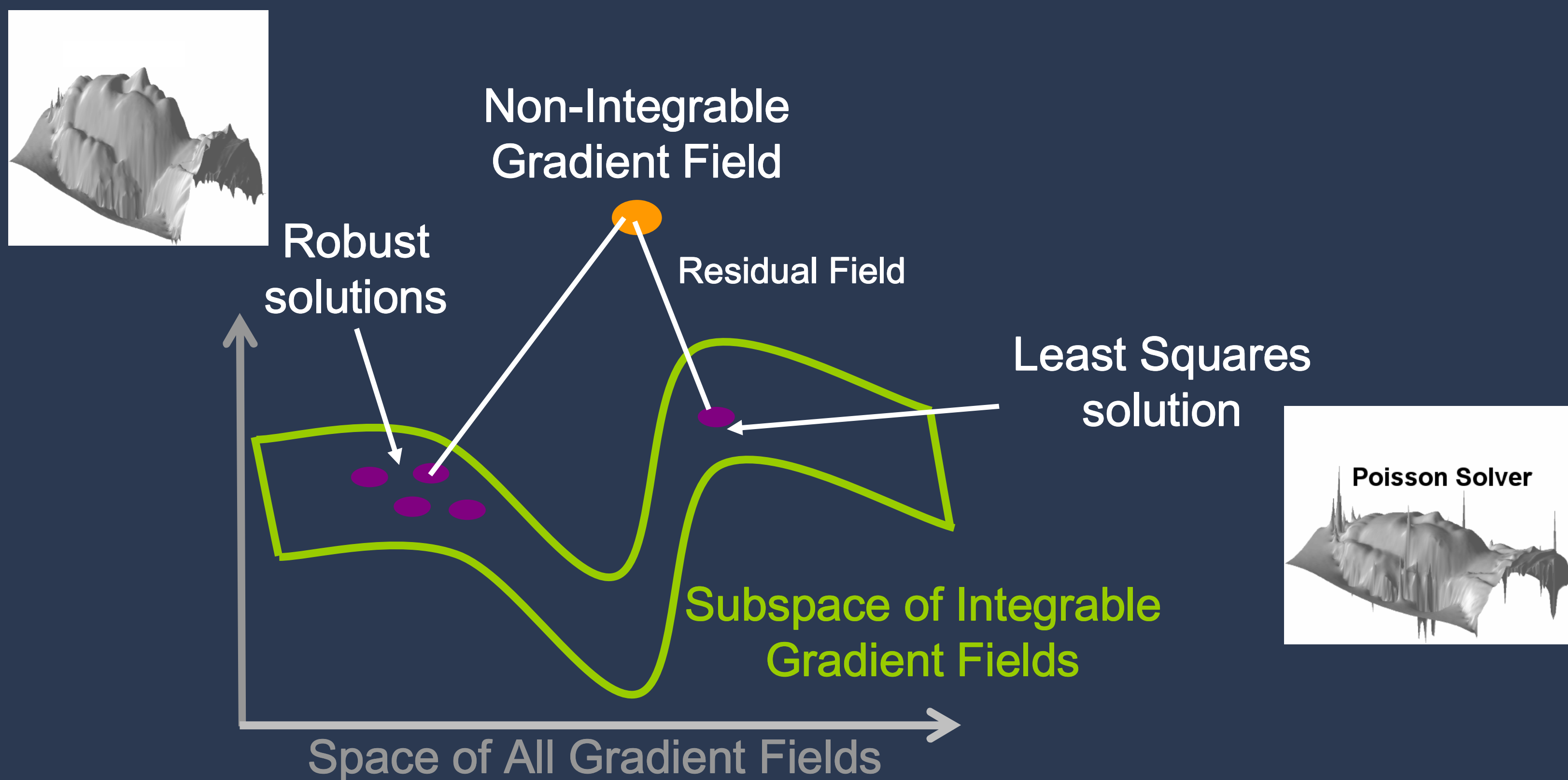
A Range of Reconstructions



By linear transformation of gradients



Space of Solutions



Common approaches

- Least Squares
  - Horn et al. (IJCV'90), Simchony et al. (PAMI'90)
  - Minimize least square error between
    - Estimated gradients (p,q) and
    - gradients of Z
- Solution: Poisson equation

$$\nabla^2 Z = \mathtt{div}(p, q)$$

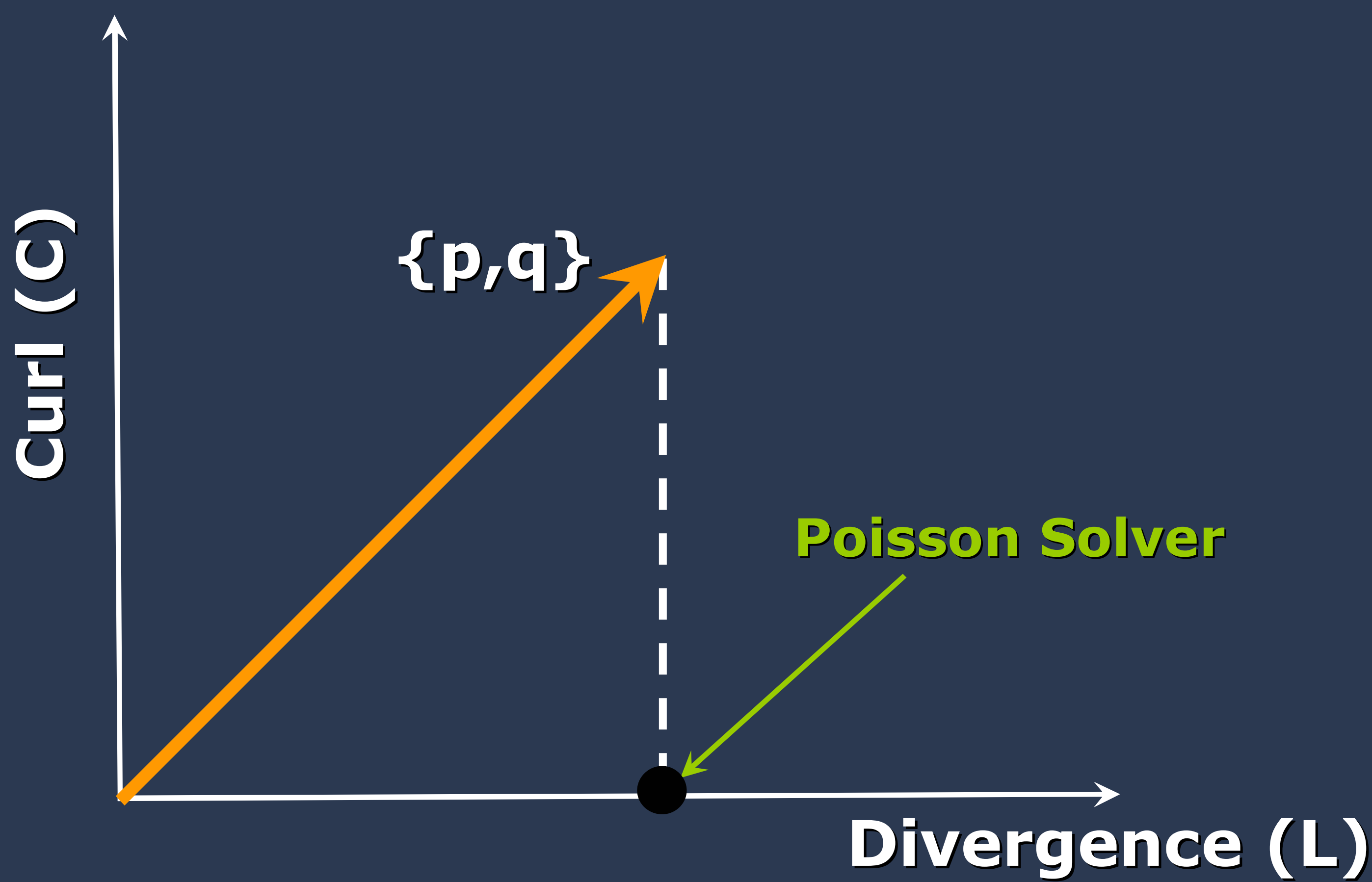
$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

Laplacian

$$\mathtt{div}(p, q) = \frac{\partial p}{\partial x} + \frac{\partial q}{\partial y}$$

Divergence

Curl-Divergence Space



Correction gradient field

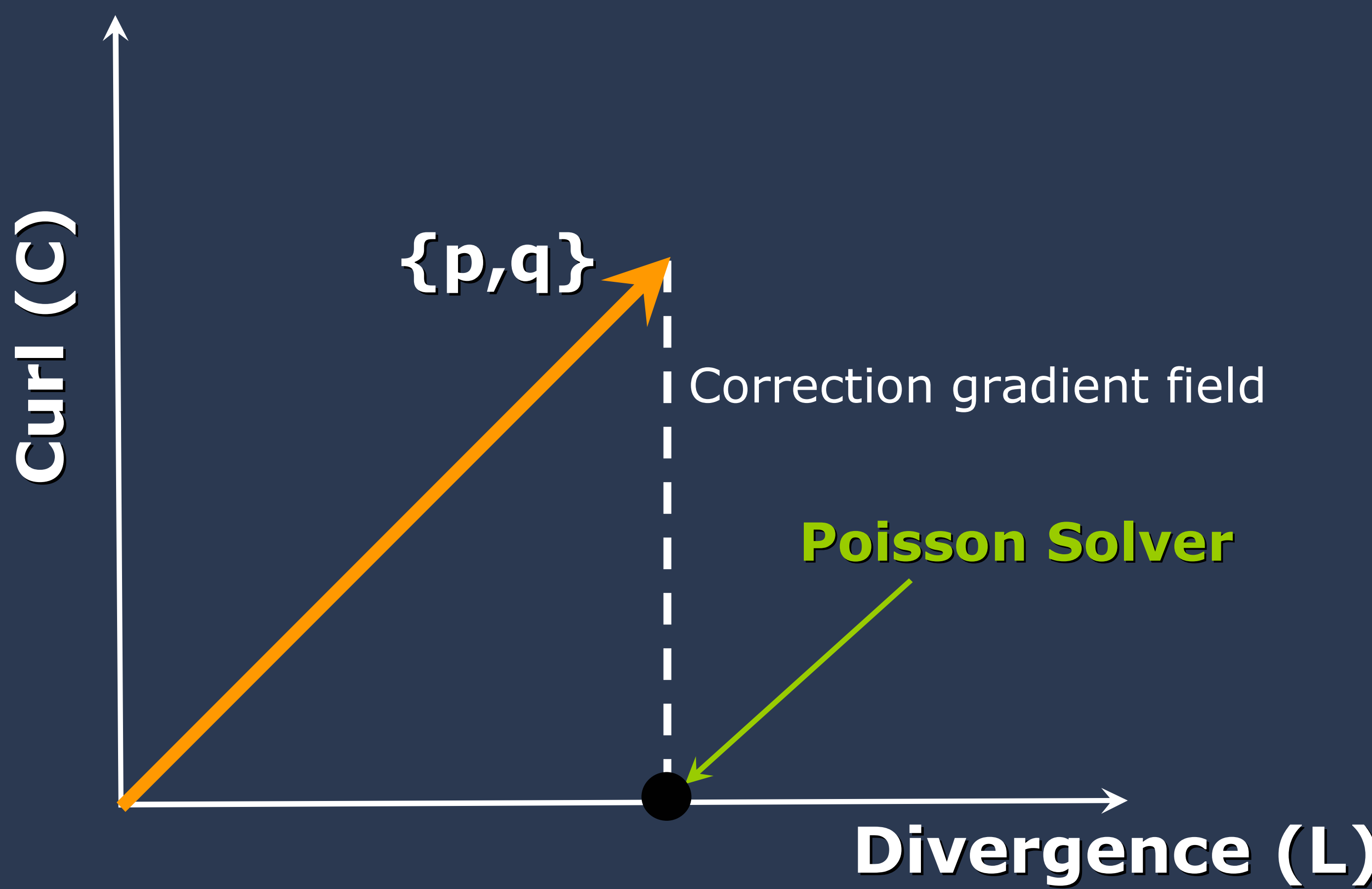
- Gradient field added to estimated non-integrable gradient field to make it integrable

$$\{Z_x, Z_y\} = \{p, q\} + \{\epsilon_x, \epsilon_y\}$$

Correction gradient field

$$J(Z) = \int\int \left((Z_x - p)^2 + (Z_y - q)^2\right) dxdy = \int\int (\epsilon_x^2 + \epsilon_y^2) dxdy \ .$$

Curl-Divergence Space



Reconstruction using basis functions

- Frankot-Chellappa algorithm
  - PAMI'88
  - Project the non-integrable gradients on to Fourier basis functions
- Other basis functions
  - Cosine: Georghiades (PAMI'01)
- Redundant basis: Shapelets Kovesi (ICCV'05)

Frankot-Chellappa Algorithm

- Fourier Basis Functions

$$\phi = \exp(j(x\omega_x + y\omega_y))$$

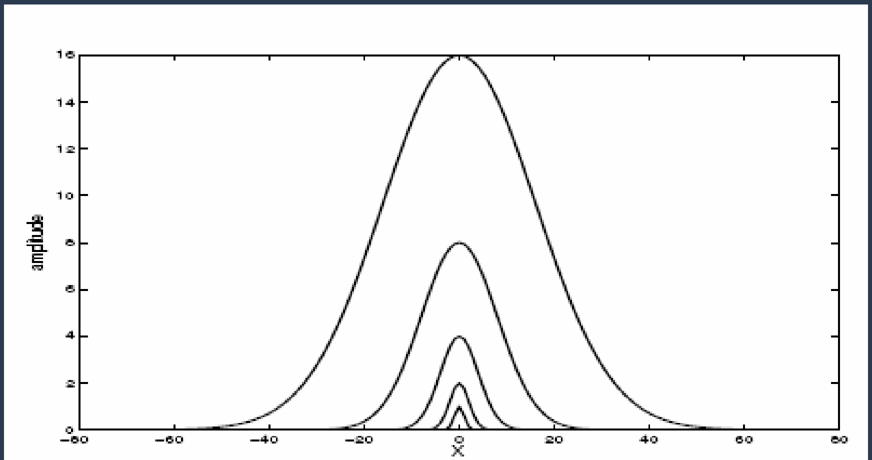
$$Z = F^{-1}\left(-j\,\frac{\omega_x F(p) + \omega_y F(q)}{\omega_x^2 + \omega_y^2}\right)$$

F denote Fourier Transform

R. T. Frankot and R. Chellappa, "A method for enforcing integrability in shape from shading algorithms", PAMI, Volume 10, Issue 4, Jul 1988 Page(s):439 - 451

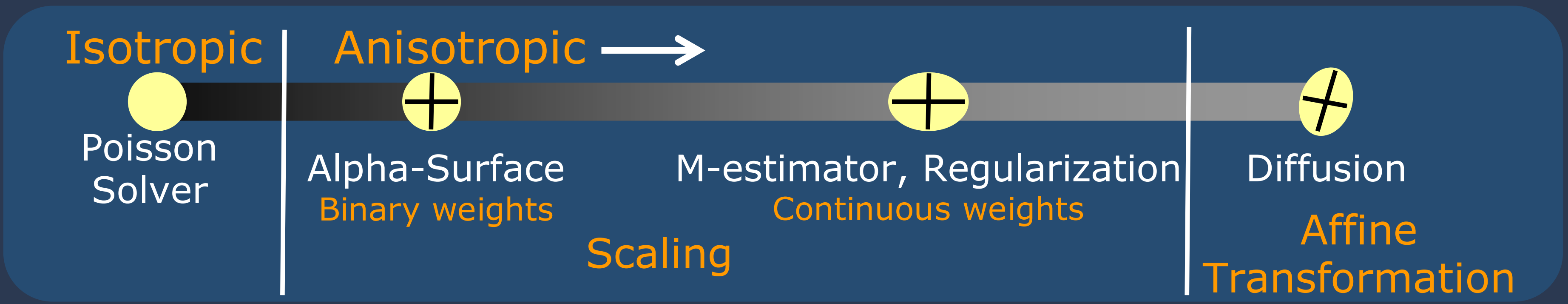
Shapelets

- Non-orthogonal redundant basis functions
- Gaussian functions
- Formulation in terms of slant and tilt



Peter Kovesi, "Shapelets Correlated with Surface Normals Produce Surfaces“, ICCV 2005

A Range of Reconstructions



By linear transformation of gradients

Key Ideas

- All gradients are not required for integration
- Replace gradients by functions of gradients

Approach

- Transforming input and output gradients
- Poisson equation

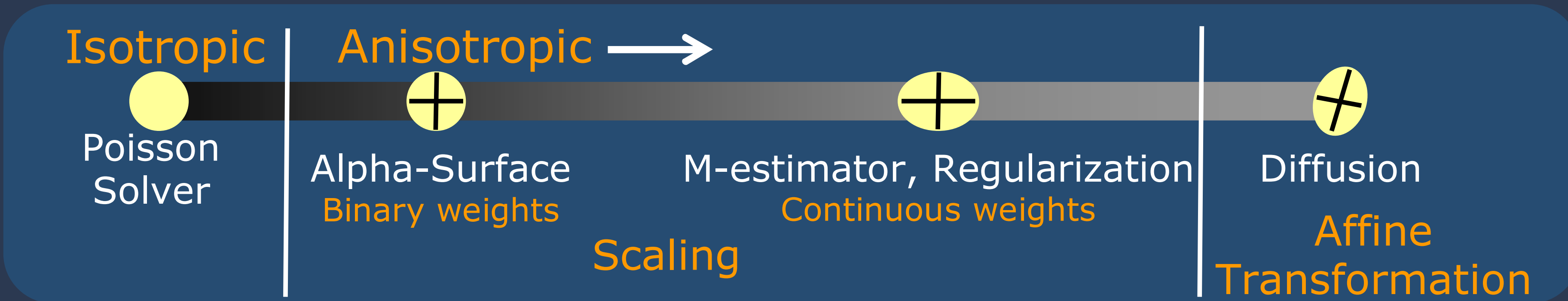
$$\nabla^2 Z = div(Z_x, Z_y) = div(p, q)$$

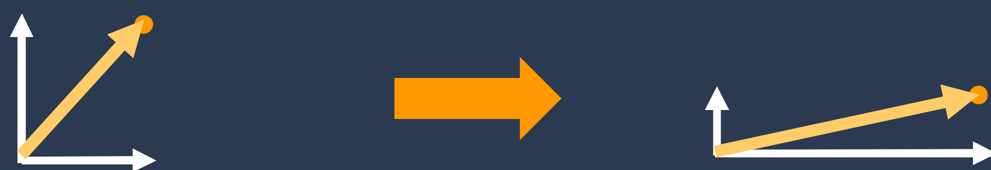
- Change to Generalized Equation

$$\mathtt{div}(f_1(Z_x, Z_y), f_2(Z_x, Z_y)) = \mathtt{div}(f_3(p, q), f_4(p, q))$$

- Using functions f_1, f_2, f_3, f_4

A Range of Solutions by Transforming Gradients





	$f_1(Z_x,Z_y)$	$f_2(Z_x,Z_y)$	$f_3(p,q)$	$f_4(p,q)$
Poisson solver	Z_x	Z_y	p	q
1.α-surface	b_xZ_x	b_yZ_y	b_xp	b_yq
2.M-estimators	w_xZ_x	w_yZ_y	w_xp	w_yq
3.Regularization	w_xZ_x	w_yZ_y	p	q
4.Diffusion	$d_{11}Z_x+d_{12}Z_y$	$d_{12}Z_x+d_{22}Z_y$	$d_{11}p+d_{12}q$	$d_{12}p+d_{22}q$

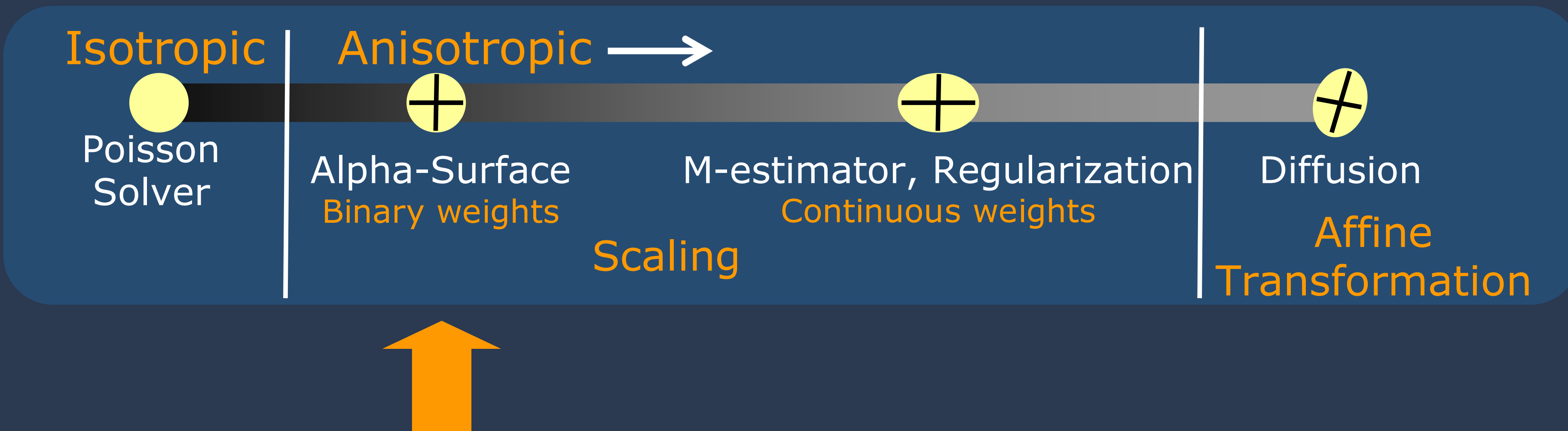
- Poisson solution = simplest case
  - $f_1(Z_x,Z_y) = Z_x$
  - $f_2(Z_x,Z_y) = Z_y$
  - $f_3(p,q) = p$
  - $f_4(p,q) = q$

$$\mathtt{div}(f_1(Z_x,Z_y), f_2(Z_x,Z_y)) = \mathtt{div}(f_3(p,q), f_4(p,q))$$

$$\Downarrow$$

$$\mathtt{div}(Z_x,Z_y) = \mathtt{div}(p,q)$$

A Range of Solutions

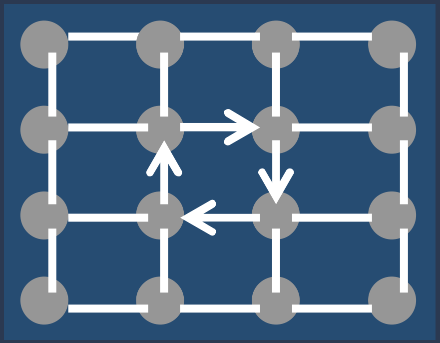


1. Robust Estimation by ignoring outliers in gradients

α-Surface: Binary Weights

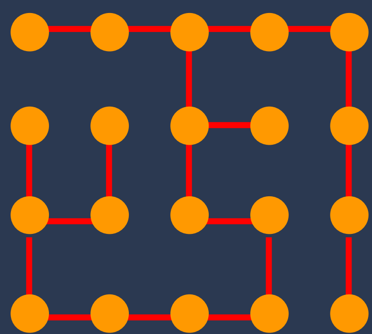
- Classifying gradients as inliers/outliers
  - Based on tolerance α

- Graph Analogy
  - 2D grid as a planar graph
  - Nodes correspond to height values
  - Edges correspond to gradient values



All Gradients are not required

- Minimal set is the spanning tree of the graph
- All nodes can be reached via spanning tree

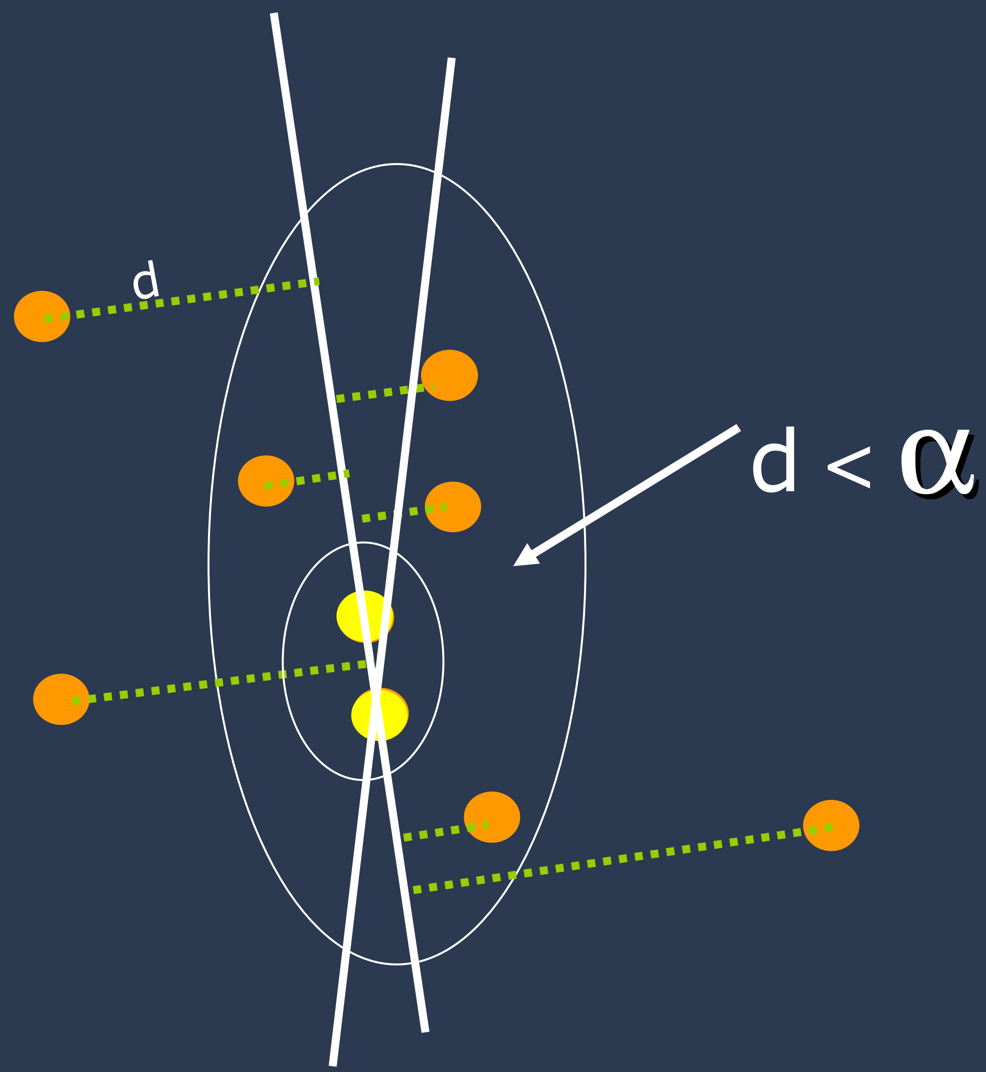


- Dimensionality of gradient field ~= $2N^2$
- N^2 -1 edges in spanning tree for N^2 nodes
- Dimensionality of solution space = $N^2 - 1$

α -Surface

- Start with spanning tree
  - Minimal set of (N^2-1) edges
- Integrate
- Iterate
  - Find inliers using given tolerance α
  - Reconstruct using new set of inliers

Visualization

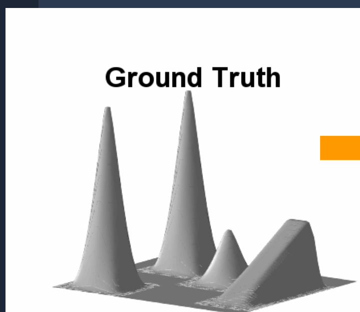


Choice of α

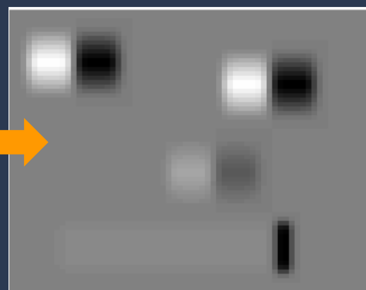
$\alpha = 0$
Minimum Spanning Tree
Unique Solution
Robust to outliers

$\alpha >>$
Overconstrained: All Gradients
Least Squares Solution
Smooth

Toy Example

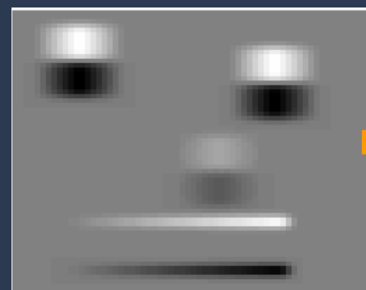


X Gradient

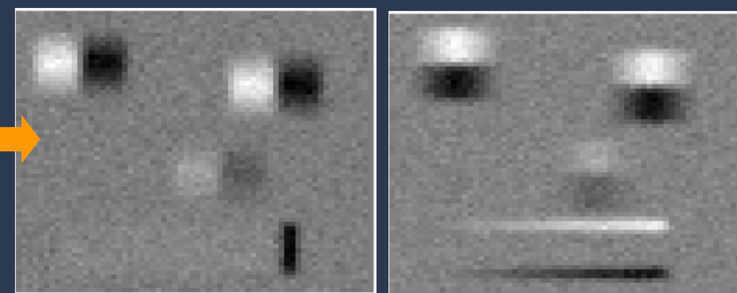


p

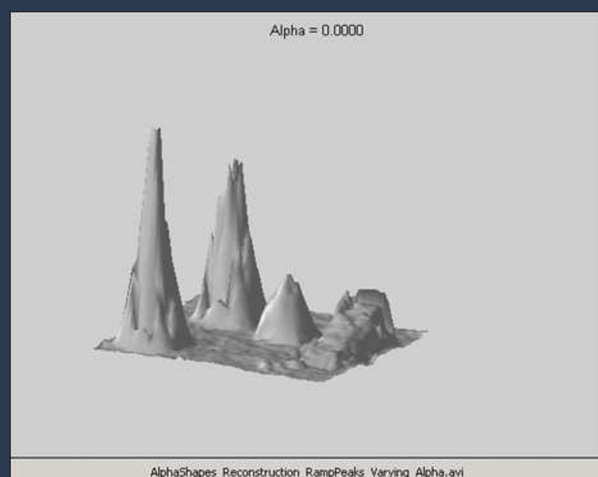
Y Gradient



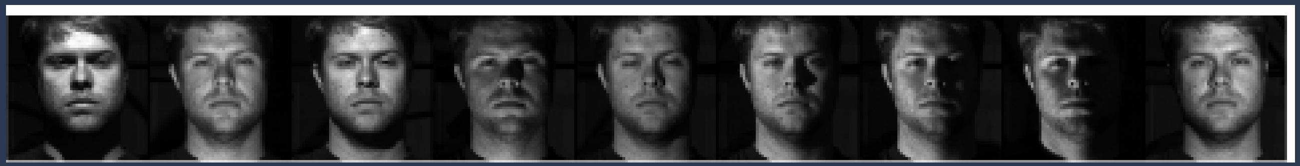
q



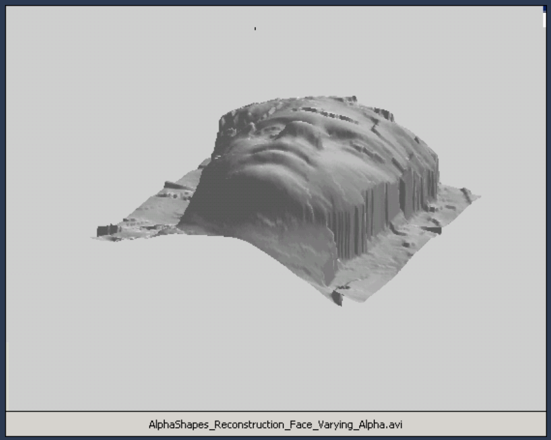
Add noise and outliers



Face

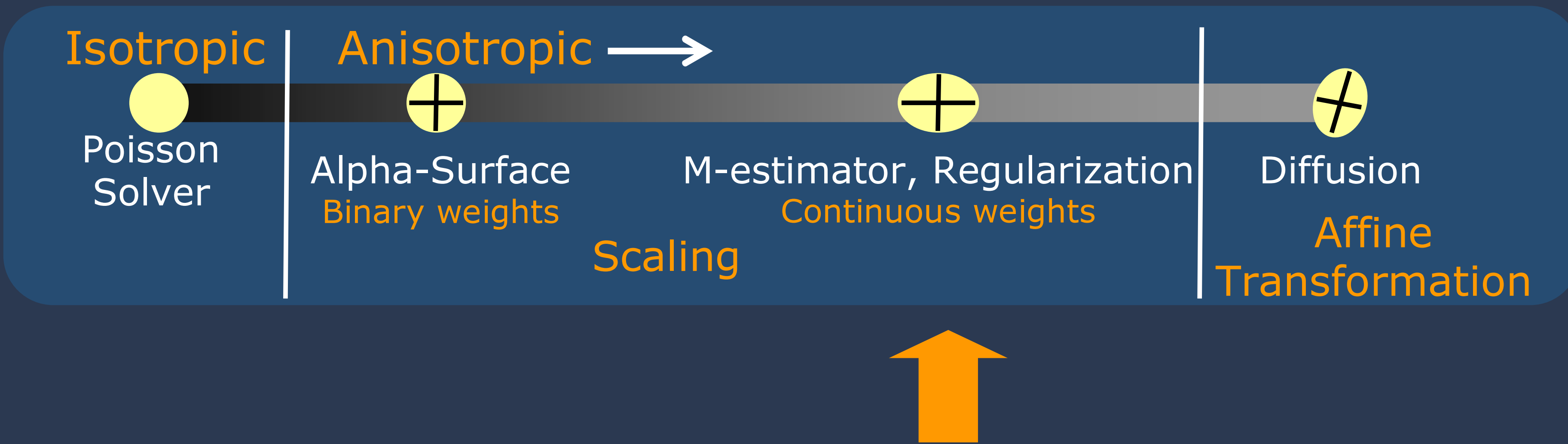


Input Images



Estimated Heights with α-tolerance

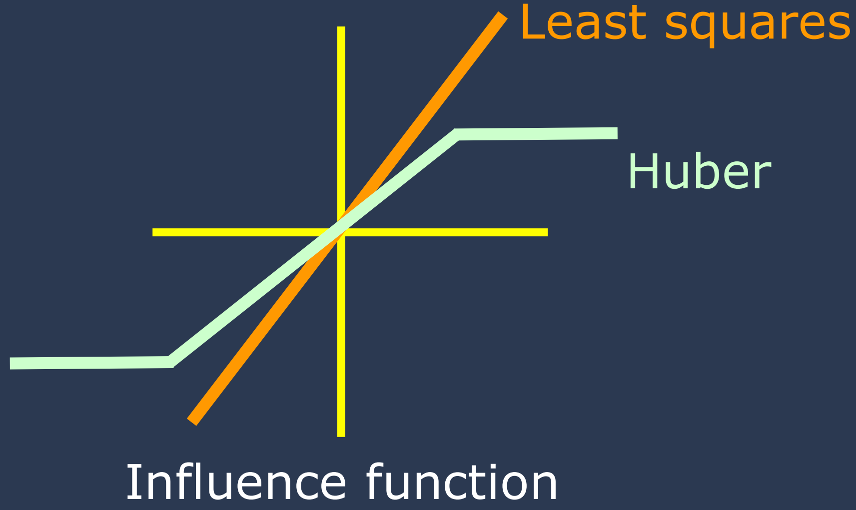
A Range of Solutions



2. Robust Estimation
by weighting gradients

Continuous weights solution

- M-estimators



Continuous weights solution

$$J = \int\int w(\epsilon_x^{k-1})(Z_x - p)^2 + w(\epsilon_y^{k-1})(Z_y - q)^2 dxdy$$

- Formulated as iterative re-weighted least squares
- w_x, w_y are weights applied to gradients

$$\mathtt{div}(w_x Z_x, w_y Z_y) = \mathtt{div}(w_x p, w_y q)$$

Regularization

- Add edge-preserving smoothness term using function Φ

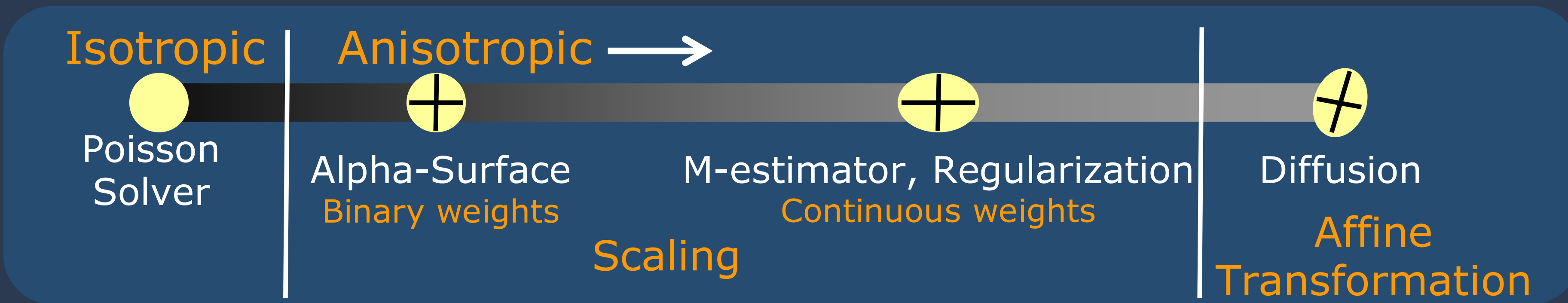
$$J(Z) = \int\int ((Z_x - p)^2 + (Z_y - q)^2) + \lambda(\phi(Z_x) + \phi(Z_y))dxdy$$

- Solved iteratively
  - Estimate weights w_x, w_y using Z
  - Update Z using weights

- $Z^0 \equiv 0$. $k \leftarrow 1$. Repeat until convergence
- $w_x^k = \phi'(Z_x^{k-1})/(2Z_x^{k-1})$, $w_y^k = \phi'(Z_y^{k-1})/(2Z_y^{k-1})$
- Solve for Z^k: $\nabla^2 Z^k + \lambda \mathtt{div}(w_x^k Z_x^k, w_y^k Z_y^k) = \mathtt{div}(p, q)$

$$\phi(s) = \sqrt{1 + s^2}$$

A Range of Solutions



3. Robust Estimation
by affine transformation
of gradients

Affine transformation of Gradient Field

- Transform gradients using matrix $\mathbf{D}_{2x2}$

$$\texttt{div}(D \begin{bmatrix} Z_x \\ Z_y \end{bmatrix}) = \texttt{div}(D \begin{bmatrix} p \\ q \end{bmatrix})$$

$$D = \begin{bmatrix} d_{11} & d_{12} \\ d_{12} & d_{22} \end{bmatrix}$$

- D is a field of tensors
  - Estimated using the given gradient field (p,q)
  - Similar to edge-preserving diffusion tensor
  - Image Restoration: Weickert'96

Affine transformation of Gradient Field

- Image restoration

Heat Equation: $I_t = \mathtt{div}(\nabla I)$ → $D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Perona-Malik $I_t = \mathtt{div}(c(\nabla I)\nabla I)$ → $D = \begin{bmatrix} c(\nabla I) & 0 \\ 0 & c(\nabla I) \end{bmatrix}$

Weickert $I_t = \mathtt{div}(D\nabla I),$ → $D(y,x) = \begin{bmatrix} d_{11}(y,x) & d_{12}(y,x) \\ d_{21}(y,x) & d_{22}(y,x) \end{bmatrix}$

Affine transformation of Gradient Field

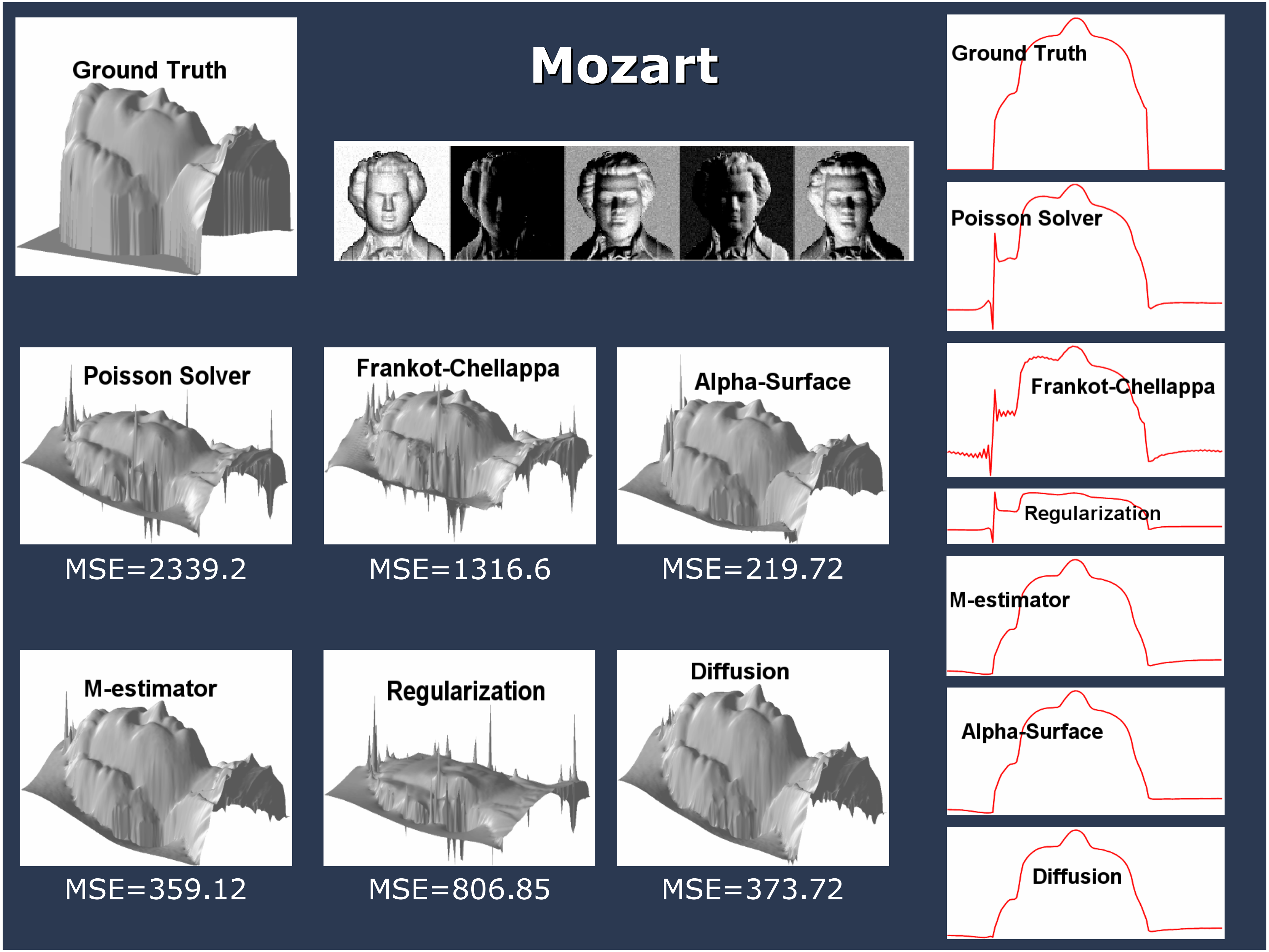
$$\mathtt{div}(D\begin{bmatrix} Z_x \\ Z_y \end{bmatrix}) = \mathtt{div}(D\begin{bmatrix} p \\ q \end{bmatrix})$$

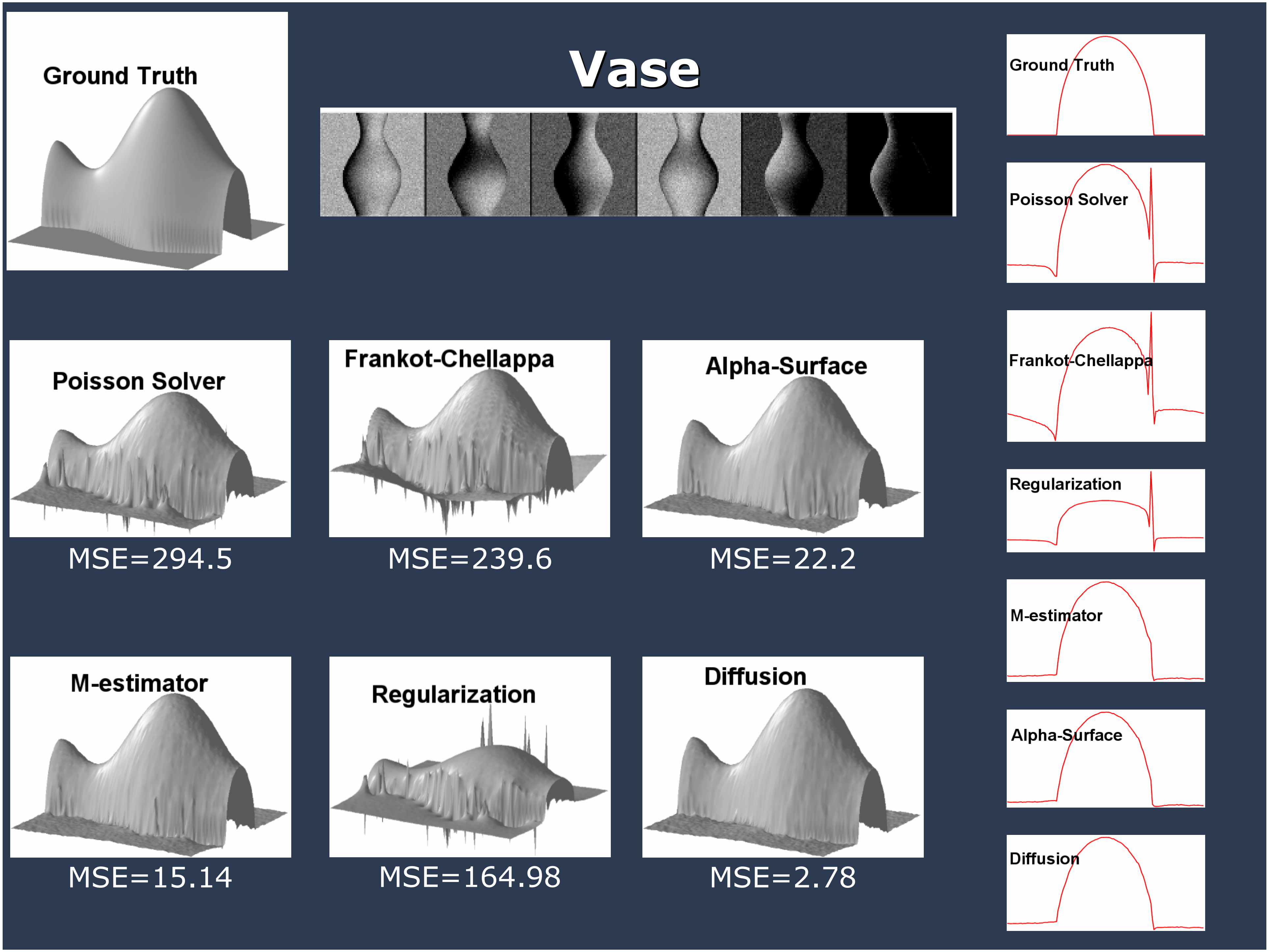
$$D = \begin{bmatrix} d_{11} & d_{12} \\ d_{12} & d_{22} \end{bmatrix}$$

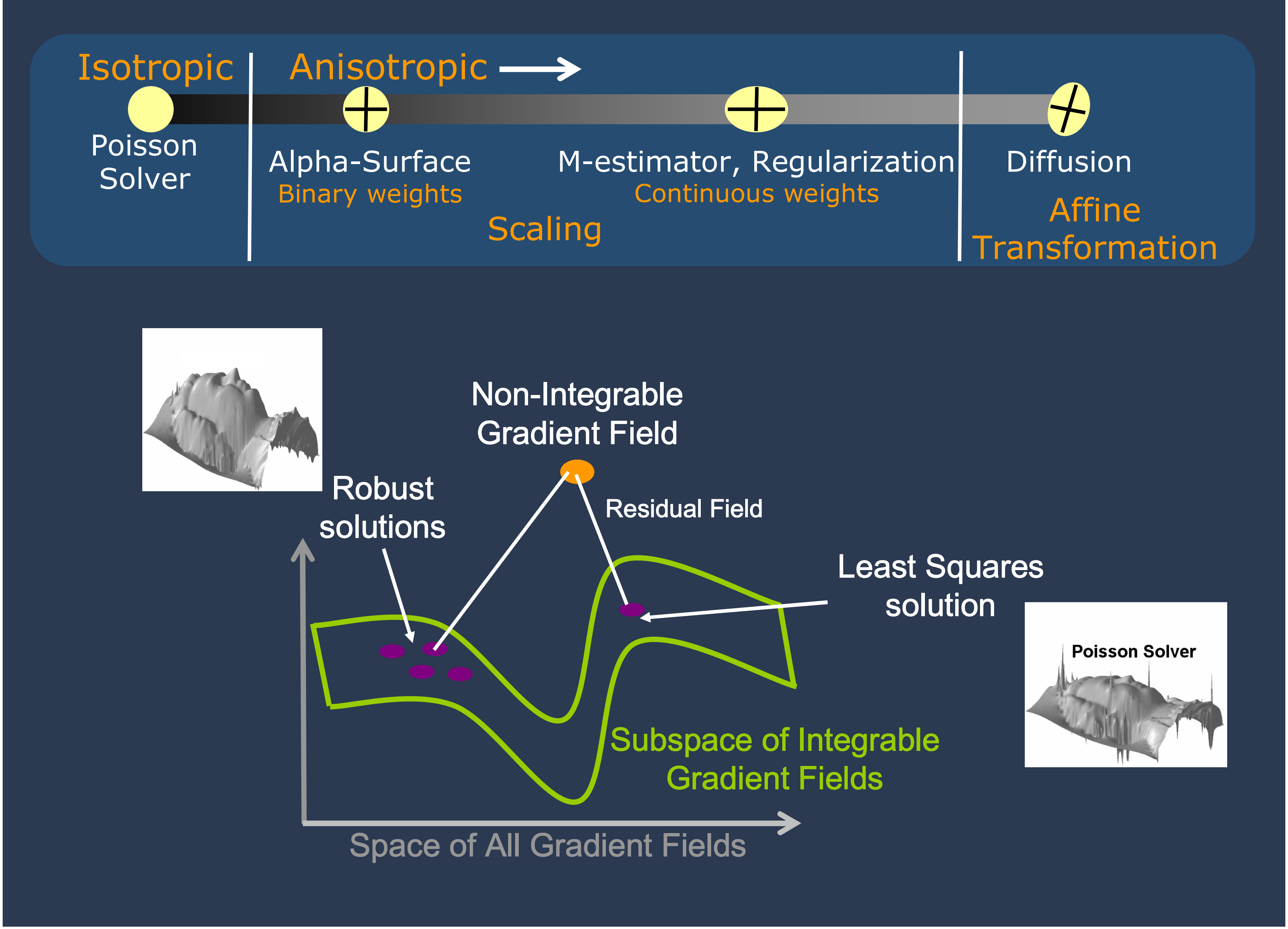
- Minimize error functional

$$J(Z) = \int\int d_{11}(Z_x - p)^2 + (d_{12} + d_{21})(Z_x - p)(Z_y - q) + d_{22}(Z_y - q)^2 dxdy$$

- More importance to low gradients
- Less importance to high gradients


Mozart
Ground Truth
Poisson Solver
MSE=2339.2
Frankot-Chellappa
MSE=1316.6
Alpha-Surface
MSE=219.72
M-estimator
MSE=359.12
Regularization
MSE=806.85
Diffusion
MSE=373.72
Ground Truth
Poisson Solver
Frankot-Chellappa
Regularization
M-estimator
Alpha-Surface
Diffusion


Vase
Ground Truth
Poisson Solver
MSE=294.5
Frankot-Chellappa
MSE=239.6
Alpha-Surface
MSE=22.2
M-estimator
MSE=15.14
Regularization
MSE=164.98
Diffusion
MSE=2.78
Ground Truth
Poisson Solver
Frankot-Chellappa
Regularization
M-estimator
Alpha-Surface
Diffusion


Isotropic
Anisotropic
Poisson Solver
Alpha-Surface
Binary weights
M-estimator, Regularization
Continuous weights
Scaling
Diffusion
Affine Transformation
Non-Integrable Gradient Field
Robust solutions
Residual Field
Least Squares solution
Poisson Solver
Subspace of Integrable Gradient Fields
Space of All Gradient Fields

Summary

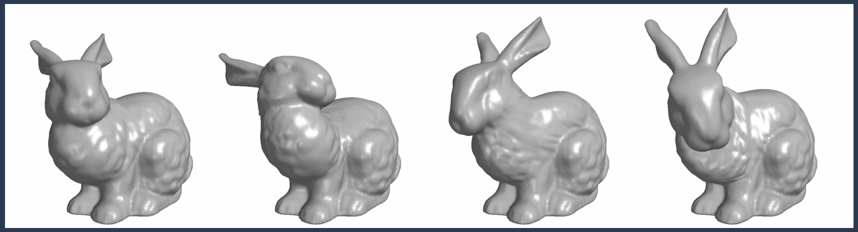
- Numerical Methods
  - Direct Solvers, Multigrid, Preconditioning
- Poisson Solver == Least Squares
  - Favor smoothness
  - Fails in presence of outliers
- Other approaches
  - Projection on basis functions
- Feature preserving reconstructions
  - Gradient Transformations
    - Robust to outliers

Acknowledgements

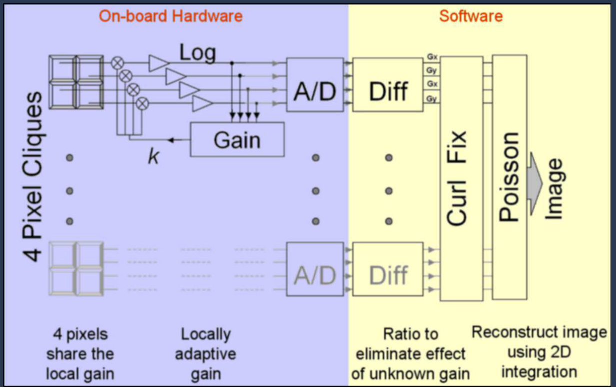
- Slides Credits
  - Rick Szeliski, Microsoft
  - Aseem Agarwala, Adobe

Next Section

- Video Manipulations
  - Space-time gradients
- Mesh Deformations
  - 3D gradients



- Color2Gray
  - Large Neighborhood differences
- Gradient Camera
  - High dynamic range imaging
  - Gradient operations at sensor level



Schedule

Introduction	(30 min, Agrawal)
Gradient Domain Manipulations	(1 hr, Raskar)
Break	(30 min)
Reconstruction Techniques	(1 hr, Agrawal)
Advanced Topics	(30 min, Raskar)
Discussion	

Course WebPage : http://www.cfar.umd.edu/~aagrawal/ICCV2007Course