Deblurring & Poisson

Fredo Durand
MIT EECS 6.8370/1

Final problem set

- You can use whatever programming language you want.
- We mostly grade based on the PDF report.

Deblurring

Can we remove blur?



original

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



Blurred



Blurred

?



original

Images and linear algebra



- We want to use powerful tools in linear algebra
  - System Mx=y
- Our unknown *x* is an image

Images and linear algebra



- We want to use powerful tools in linear algebra
  - System Mx=y
- Our unknown *x* is an image
- We could solve deconvolution by stacking all our pixels into a big 1D vector x, build a big matrix M, solve for Mx=b (e.g. with Eigen), then copy the pixels back into the image. M would be big!
- Instead, we'll keep images as Image, but we'll use abstract derivations in terms of M and x to know what to do
- The implementation will use simple image operations such as convolution, multiplication

Least squares

- **Turn equation into a least square**

$$\min \|Mx-y\|^2$$

  - deals with over or ill-conditioned systems
  - enables derivation of iterative methods in terms of gradient descent
  - enables the addition of other energy terms to regularize the system

Turn it into a least square

- min $\|Mx-y\|^2$
- min $(Mx-y)^T (Mx-y)$
- min $x^TM^TMx -2x^TM^Ty+y^Ty$

- derive wrt x, set to 0
- $2M^TMx-2M^Ty$

- Normal equation:
- $M^TMx-M^Ty=0$

Keep in mind

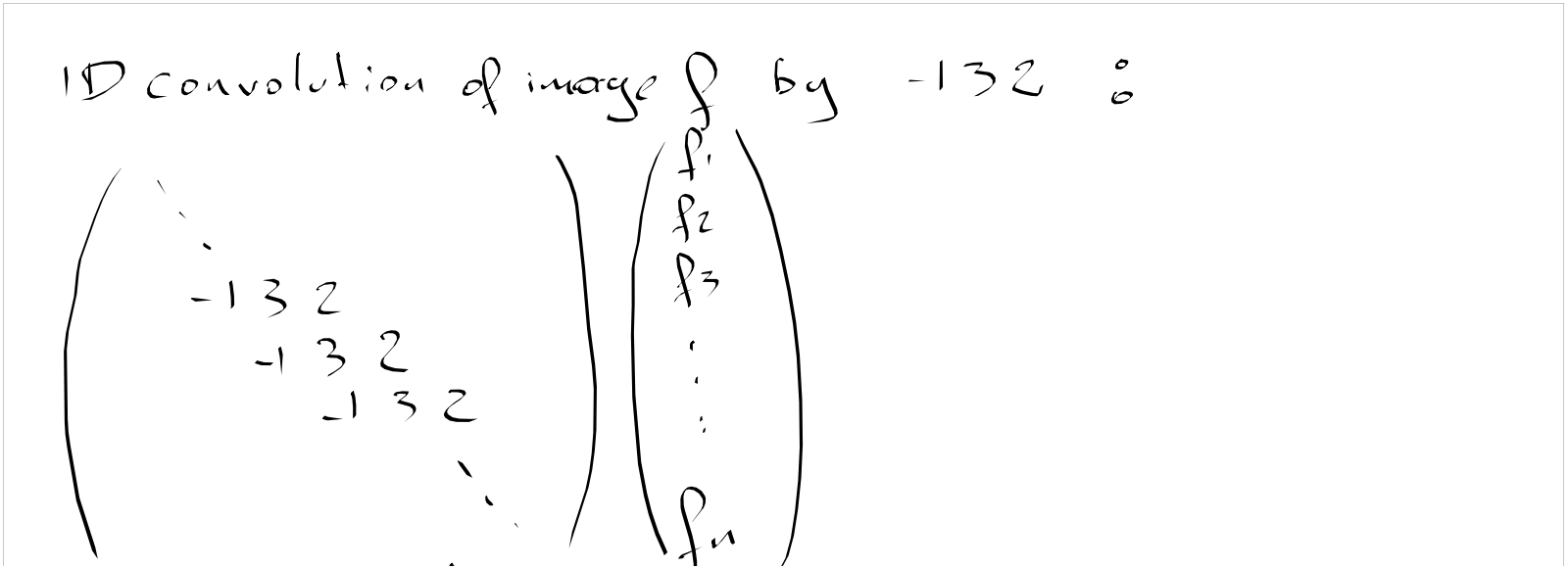
- $\min \|Mx-y\|^2$
- $\min x^T M^T M x - 2x^T M^T y + y^T y$
- $M^T M x - M^T y = 0$

- In our case:
  - x is the unknown sharp image
  - y is the blurry input
  - M is the convolution by the known blurring kernel
    - try it for 1D convolution

1D convolution: transpose & flipping

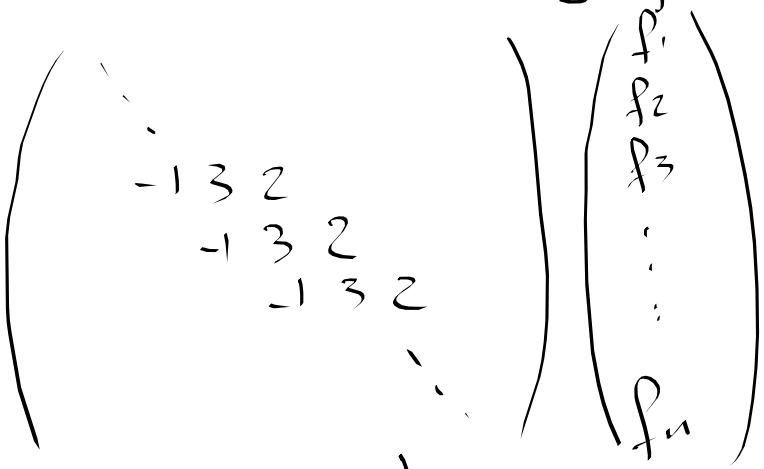
1D convolution of image f by $-1\ 3\ 2$:

1D convolution: transpose & flipping

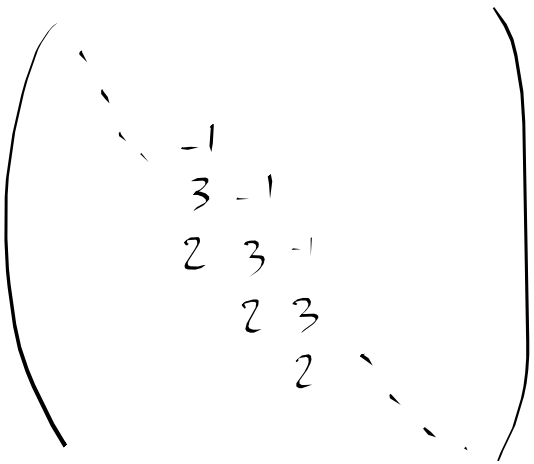


1D convolution: transpose & flipping

1D convolution of image f by $-1\ 3\ 2$:



transpose matrix :



= convolution by $2\ 3\ -1$

aka the flipped kernel

Flipping in 2D

- Flip both the x and y axis of the kernel

Notation

- min $x^T M^T M x - 2x^T M^T y + y^T y$
- $M^T M x - M^T y = 0$

- call $A = M^T M$; $b = M^T y$; $c = 1/2\ y^T y$
  add a 1/2 factor to get a cleaner derivative:

- minimize $$f(x) = \frac{1}{2} x^T A x - b^T x + c$$

- derivative: $$f'(x) = Ax - b$$

Reference

- **Strang 4.3 p206**
  - http://www.amazon.com/Introduction-Linear-Algebra-Fourth-Gilbert/dp/0980232716/ref=la_B001H6L6OK_1_1?s=books&ie=UTF8&qid=1383058561&sr=1-1
- **Take 18.C06 or 18.06fgdtcrhdbryrhyy**
- **http://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/**
-

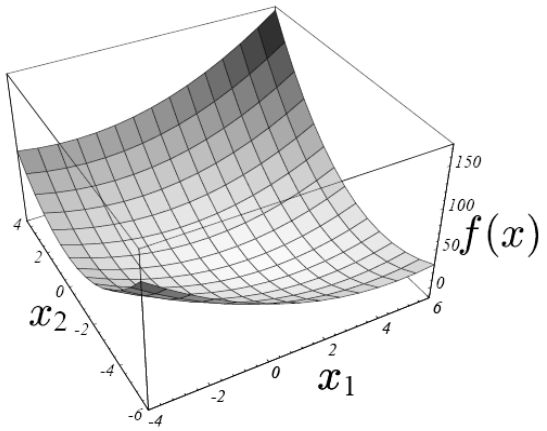
Gradient descent

with many illustrations from Jonathan Schewchuck

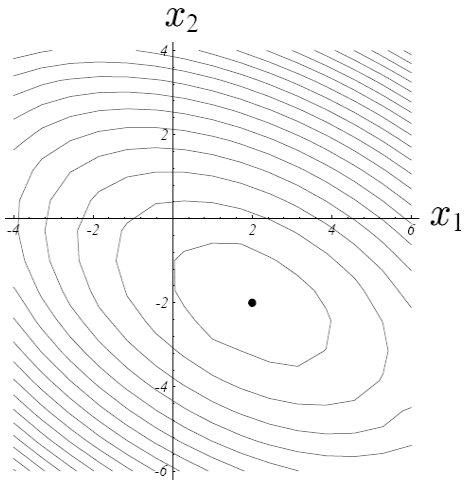
Linear least squares

- **Minimize** $$f(x) = \frac{1}{2}x^T Ax - b^T x + c$$
  - $A=M^TM$ is nicely positive semi definite which means we have a nice parabola with a minimum (not a maximum)

Simple 2-variable example: $$\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} x = \begin{bmatrix} 2 \\ -8 \end{bmatrix}$$



Graph of f(x). Our solution is the minimum



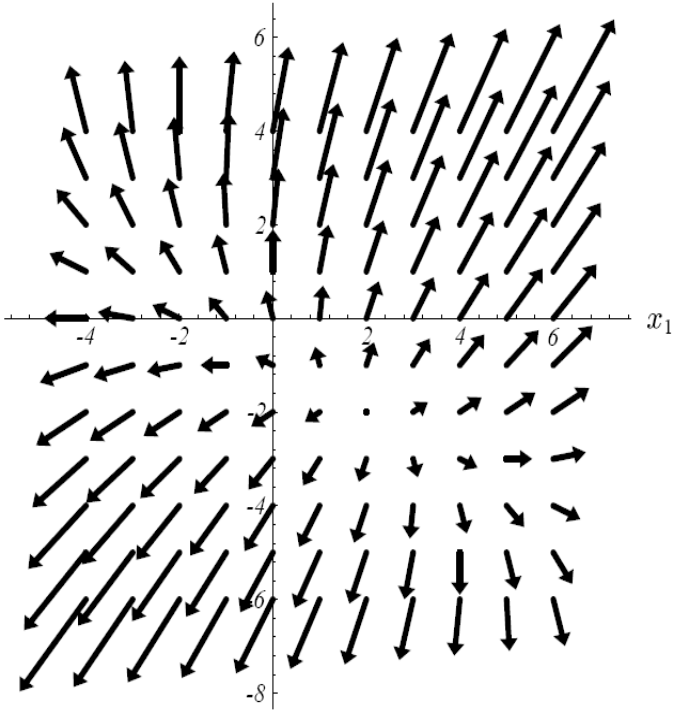
Isocontours of f(x). Our solution is the dot

Gradient of the quadratic form

$$f'(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \frac{\partial}{\partial x_2} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{bmatrix}$$

– Multidimensional gradient (as many dim as rows in x)

Careful:
not an image gradient
but gradient of
quadratic



Gradient $f'(x)$ of the quadratic form. For every x, the gradient points in the direction of steepest increase of $f(x)$, and is orthogonal to the contour lines.

We saw that

$$f'(x) = Ax - b$$

Gradient of the quadratic form

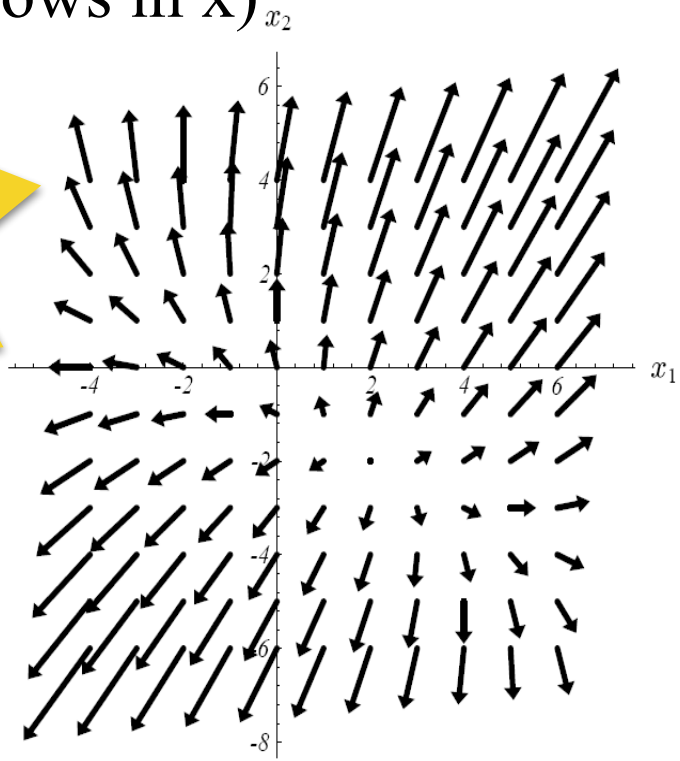
$$f'(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \frac{\partial}{\partial x_2} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{bmatrix}$$

– Multidimensional gradient (as many dim as rows in x)

We saw that

$$f'(x) = Ax - b$$

Careful: not an image gradient but gradient of quadratic



Gradient $f'(x)$ of the quadratic form. For every x, the gradient points in the direction of steepest increase of $f(x)$, and is orthogonal to the contour lines.



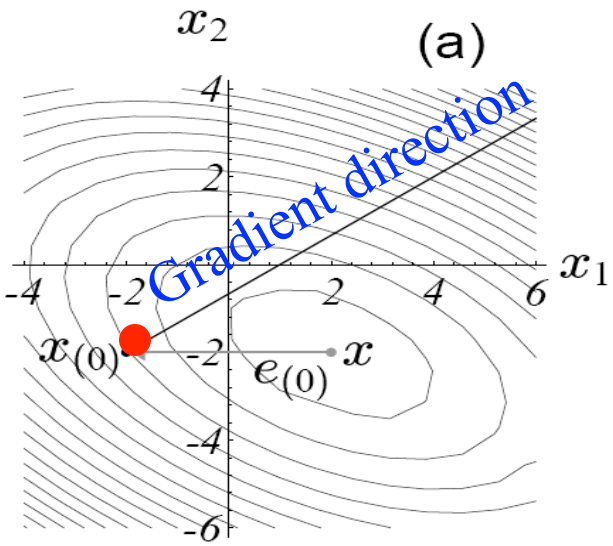
xi -f'(xi)

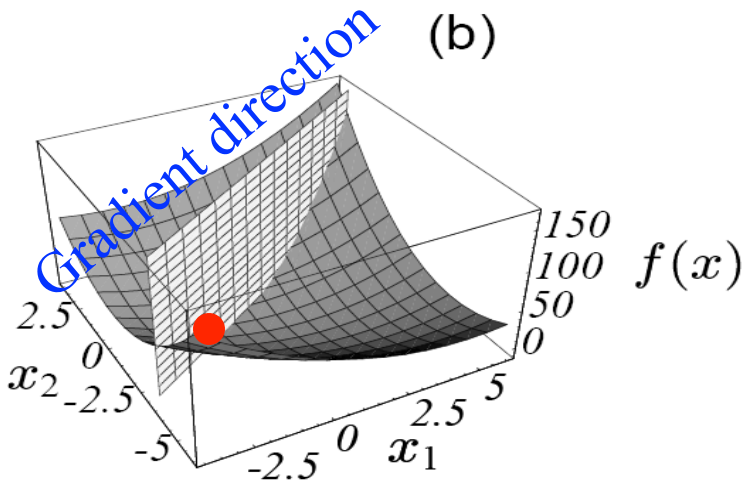
New term: Residual

- **How different is the value of an equation from the desired value**
- **At iteration i, we are at a point $x_{(i)}$**
- **Residual of normal form $r_{(i)}=b-Ax_{(i)}$**
- **Cool property of quadratic form: residual = - gradient**

Steepest descent/ascent

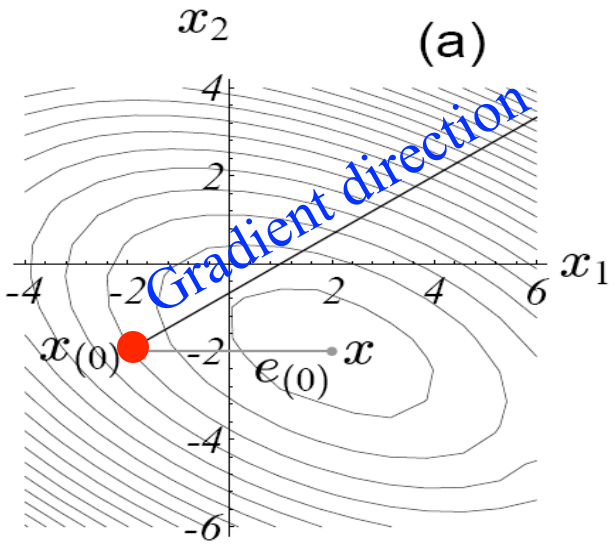
- **Pick residual (negative gradient) direction**
  - $Ax_{(i)}$-b

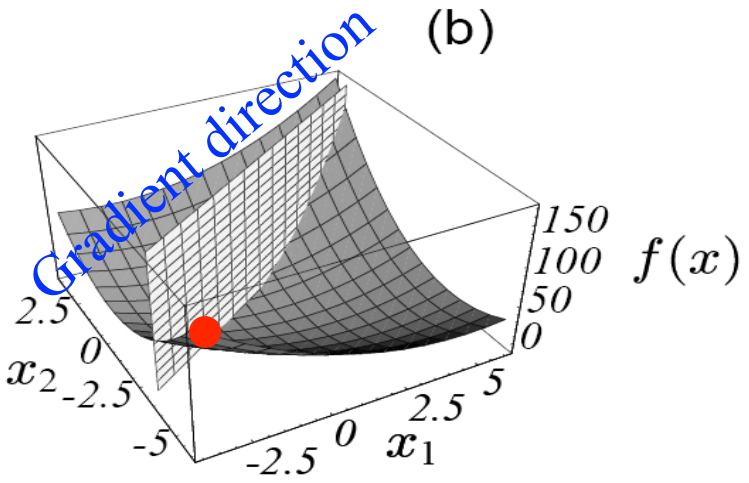


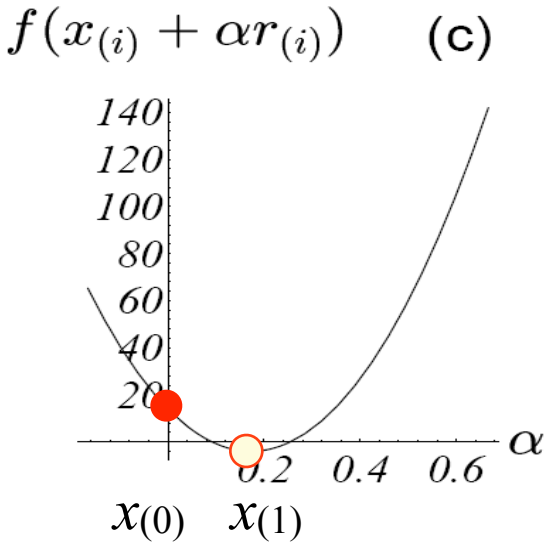


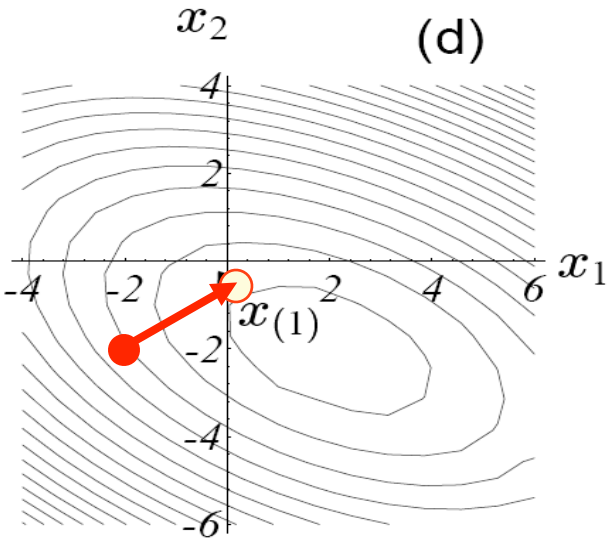
Steepest descent/ascent

- **Pick residual (negative gradient) direction**
  - $Ax_{(i)}$-b
- **Find optimum in this direction**









Energy along the gradient direction

Optimum along gradient direction

- **minimize $f(x_1)$**
- **$x_1 = x_0 + \alpha r_0$**

unknown?

Optimum along gradient direction

- **minimize $f(x_1)$**
- **$x_1 = x_0 + \alpha r_0$**

unknown: **α**

- **make derivative of f wrt α along direction zero:**
- $$\frac{d}{d\alpha} f(x_{(1)}) = \underbrace{f'(x_{(1)})}_{b - Ax_{(1)}} \underbrace{\frac{dx_{(1)}}{d\alpha}}_{r_{(0)}}$$

$$(b - A(x_{(0)} + \alpha r_{(0)}))^T r_{(0)} = 0$$

$$\alpha = \frac{r_{(0)}^T r_{(0)}}{r_{(0)}^T A r_{(0)}}$$

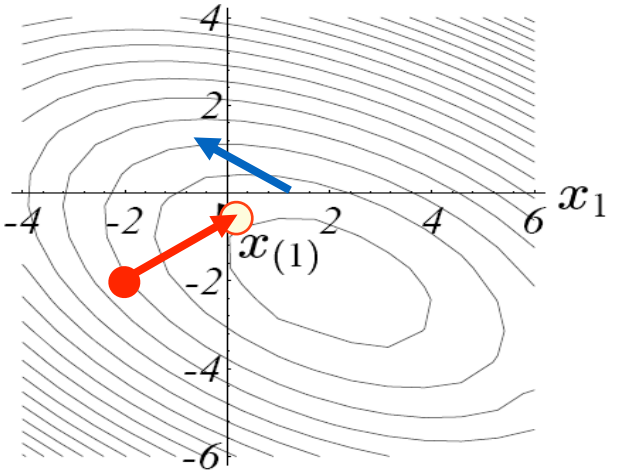
Geometric interpretation

$$\frac{d}{d\alpha} f(x_{(1)}) = \underbrace{f'(x_{(1)})}_{b - Ax_{(1)}} \underbrace{\frac{dx_{(1)}}{d\alpha}}_{r_{(0)}}$$

$$(b - A(x_{(0)} + \alpha r_{(0)}))^T r_{(0)} = 0$$

$$\alpha = \frac{r_{(0)}^T r_{(0)}}{r_{(0)}^T A r_{(0)}}$$

- **Line search:**
  - **make the gradient orthogonal to the line**



Recap: Gradient Descent

- Residual = - gradient : $r_{(i)}=b-Ax_{(i)}$
- Iteratively walk along residual: $x_{(i+1)}=x_{(i)}+a\ r_{(i)}$
- Find optimal along residual direction:

$$\alpha = \frac{r_{(i)}^T r_{(i)}}{r_{(i)}^T A r_{(i)}}$$

- note that we only need to apply matrix A to vectors
  - Not too expensive because A is sparse
- Not a very efficient method but we'll improve it

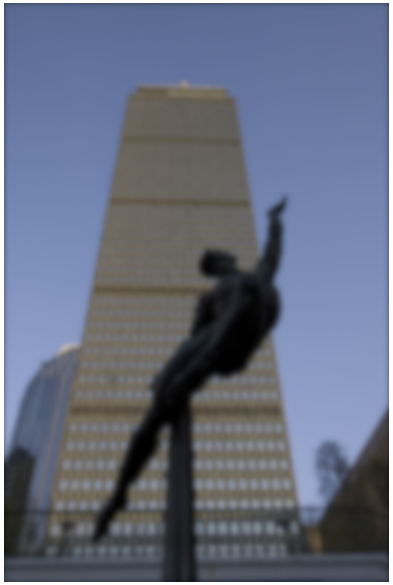
Conjugate gradient and other tricks

- **Smarter gradient descent**
  - Conjugate gradient
    - Smarter iteration as a function of history to avoid zigzags
  - Preconditioning
    - Change of basis (extra matrix) so that problem is easier to solve

Applying it to images

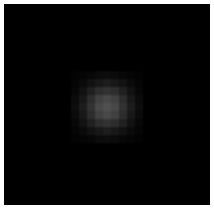
Inputs:

- input: blurry image y, known kernel

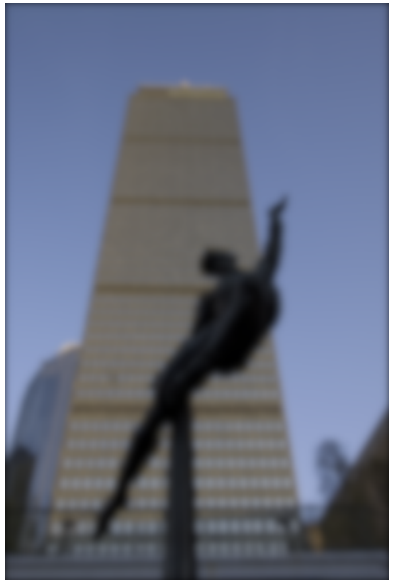


y

M: Convolution by



(careful, M is not the kernel itself but the application of the kernel)



$b=M^T y$

Recall gradient descent

$$r_i = b - Ax_i$$

$$\alpha = \frac{r_i \cdot r_i}{r_i \cdot Ar_i}$$

$$x_{i+1} = x_i + \alpha r_i$$

Recall gradient descent

$$r_i = b - Ax_i$$

$$\alpha = \frac{r_i \cdot r_i}{r_i \cdot Ar_i}$$

$$x_{i+1} = x_i + \alpha r_i$$

- **x is an unknown image.**
  - x_i is the current guess. x_{i+1} is the next guess
- **$A=M^TM$ where M is the blur operator**
  - Multiplying by M just means computing a convolution
  - Mx is the convolved image
  - M^T blurs by the flipped kernel (flipped along x & y)
- **$b=M^Ty$ is the input blurry image, blurred by the flipped kernel**
- **r_i is also an image**

Image dot product

$$r_i = b - Ax_i$$

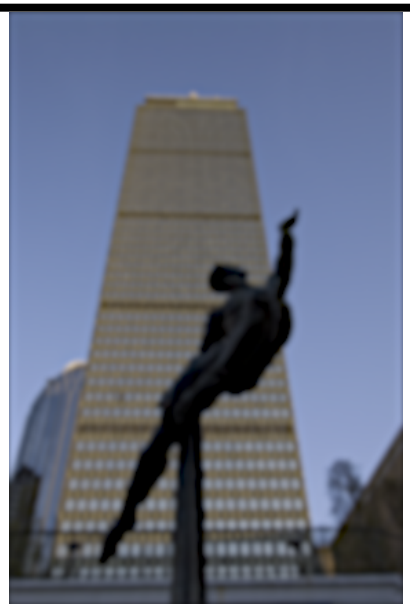
$$\alpha = \frac{r_i \cdot r_i}{r_i \cdot Ar_i}$$

$$x_{i+1} = x_i + \alpha r_i$$

- **What does r_i. r_i or r_i.Ar_i mean?**
- **Dot product: sum of the pairwise product of all the coordinates; returns a scalar**
  - For images: take the product of all pixel values for all channels and add them
- **Easy with numpy operations**

Image terms

xi



ri



$$r_i = b - Ax_i$$

$$\alpha = \frac{r_i \cdot r_i}{r_i \cdot Ar_i}$$

$$x_{i+1} = x_i + \alpha r_i$$

Results



Not perfect but sharper.

Bottomline

- We solve Ax=b
- But we never form or store matrix A
- We just do repeated convolutions, a few image dot products, scalar multiplication and addition

$$r_i = b - Ax_i$$

convolution, subtraction

$$\alpha = \frac{r_i \cdot r_i}{r_i \cdot Ar_i}$$

Dot product, convolution, dot product, division of two scalars

$$x_{i+1} = x_i + \alpha r_i$$

multiplication by scalar, addition

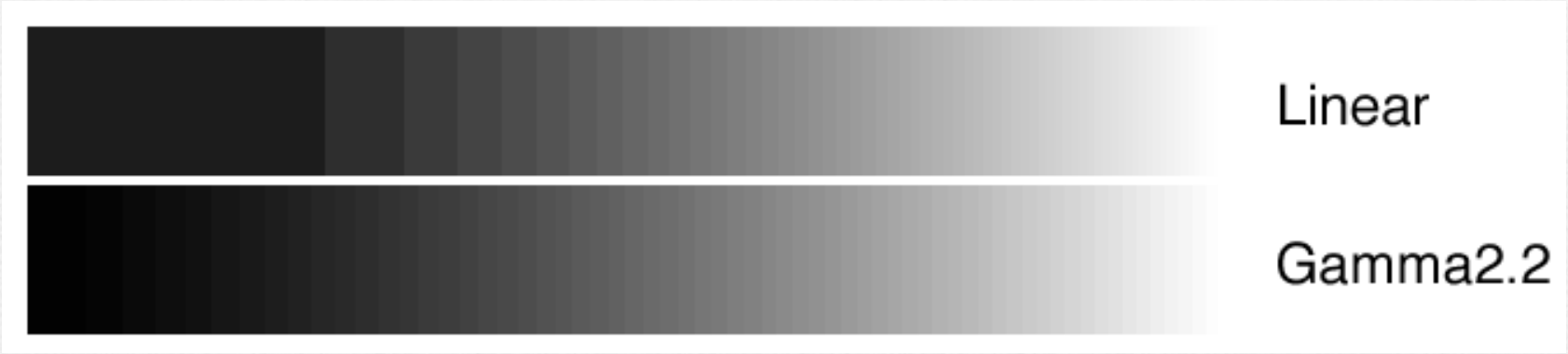
Very general idea

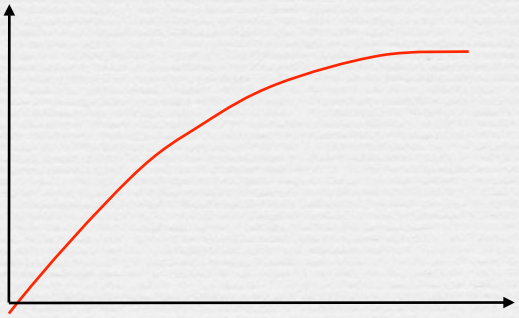
- Whenever you have a linear process
- You can use tools from linear algebra
- You don’t necessarily to form matrices and vectors

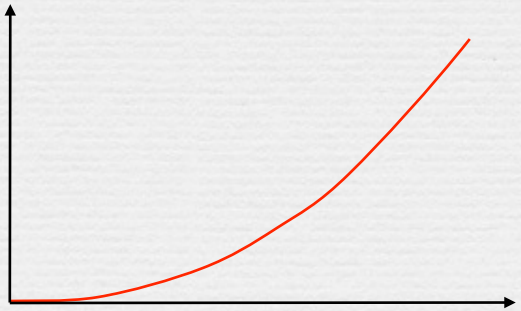
Testing the limits

Recall gamma

- Digital images are usually gamma encoded
- Don't forget to decode before performing linear processing

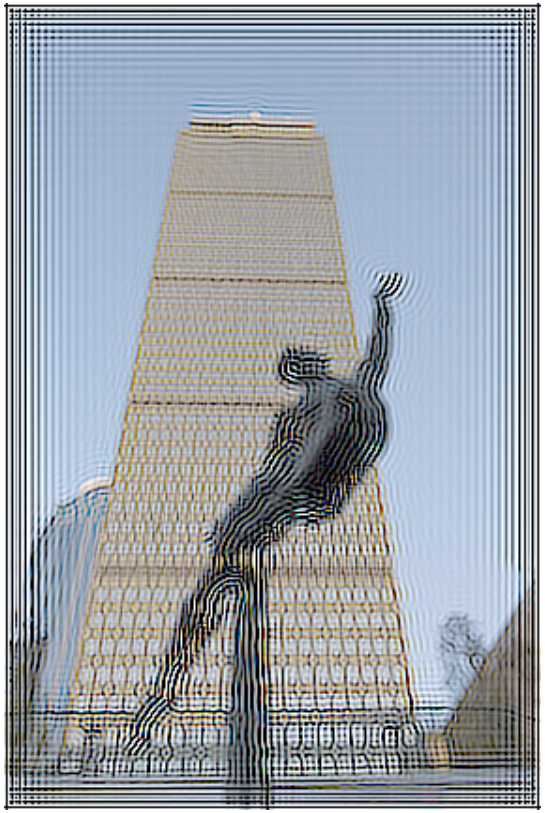






Non-linear encoding

- Images are usually gamma-encoded
- What if you forget to linearize them?
- Yikes!



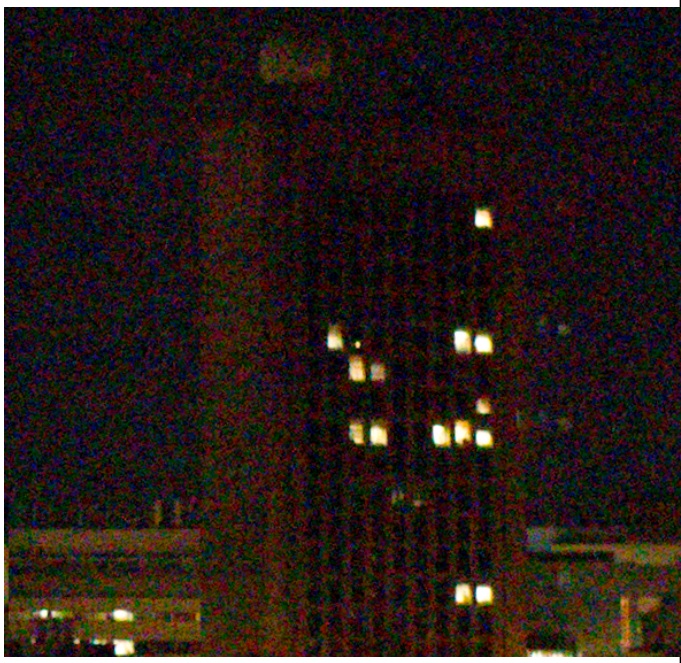
Sensor noise

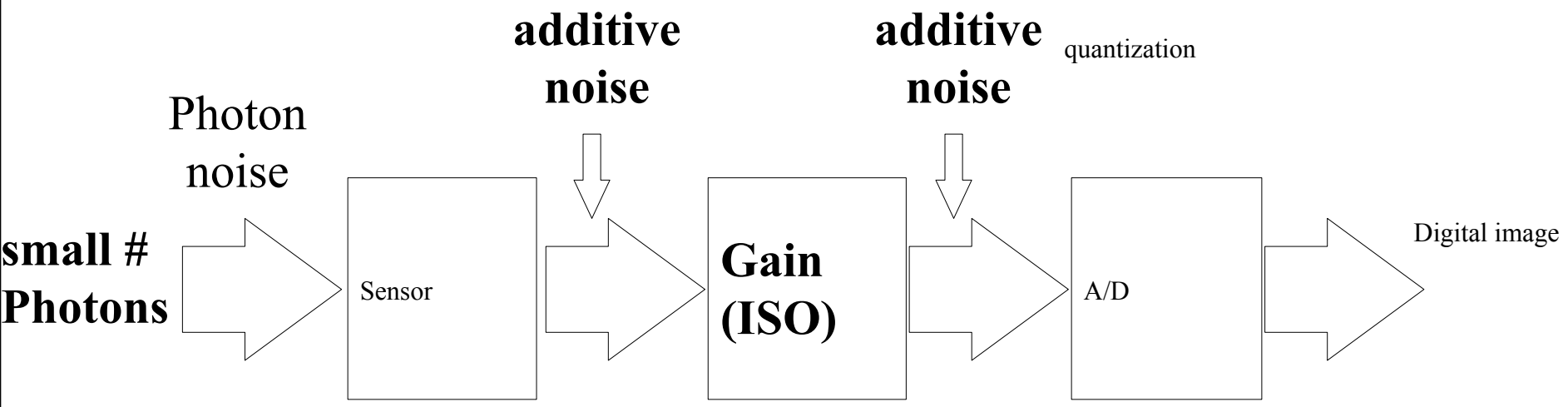
- We usually don't observe

$$y = x \otimes k$$

- Noise n gets added:

$$y = x \otimes k + n$$

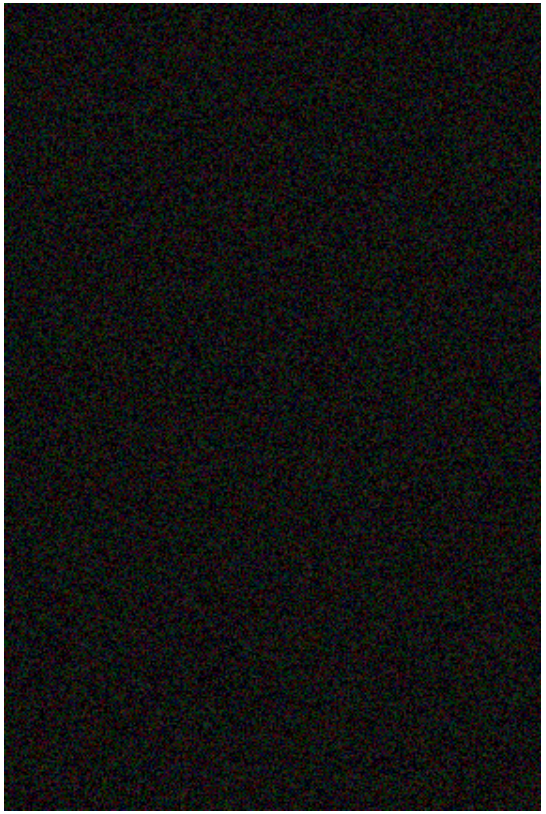




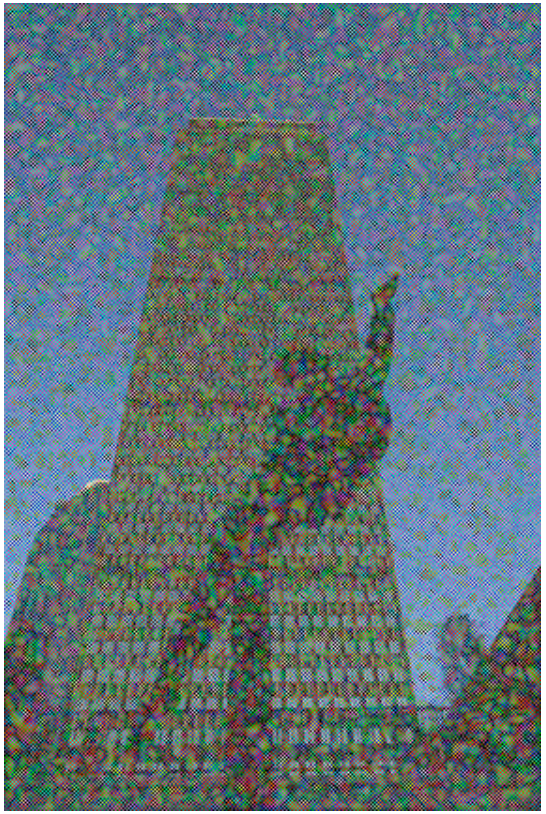
Effect of noise



Blurry image
Gaussian, sigma=1



Noise*20



Deblurred

Effect of noise

$$y = Mx + n$$

Effect of noise

$$y = Mx + n$$

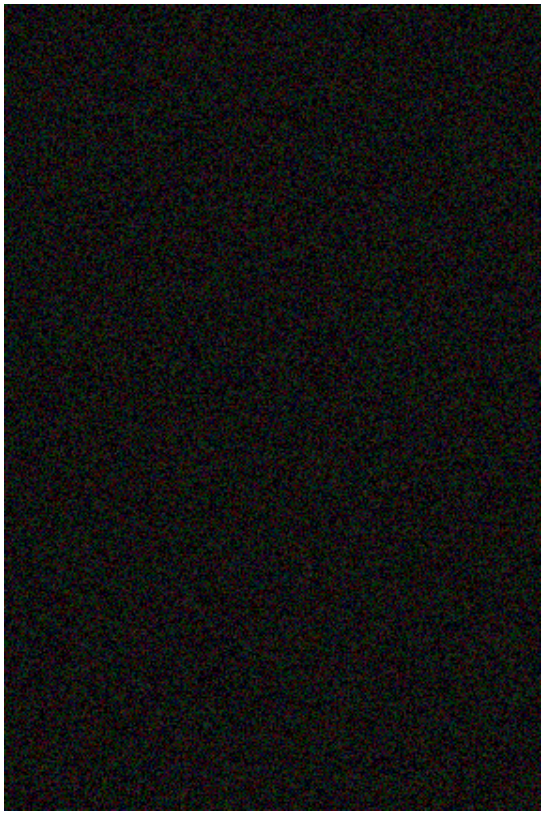
$$\hat{x} = M^{-1}y$$

$$\hat{x} = M^{-1}(Mx + n)$$

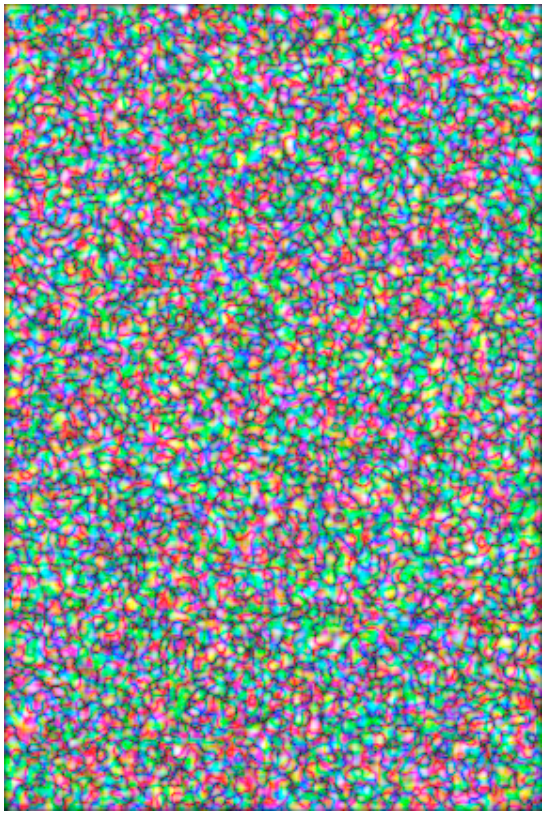
$$\hat{x} = x + M^{-1}n$$

- **Deconvolved noise gets added!**

Deconvolving the noise



Noise*20

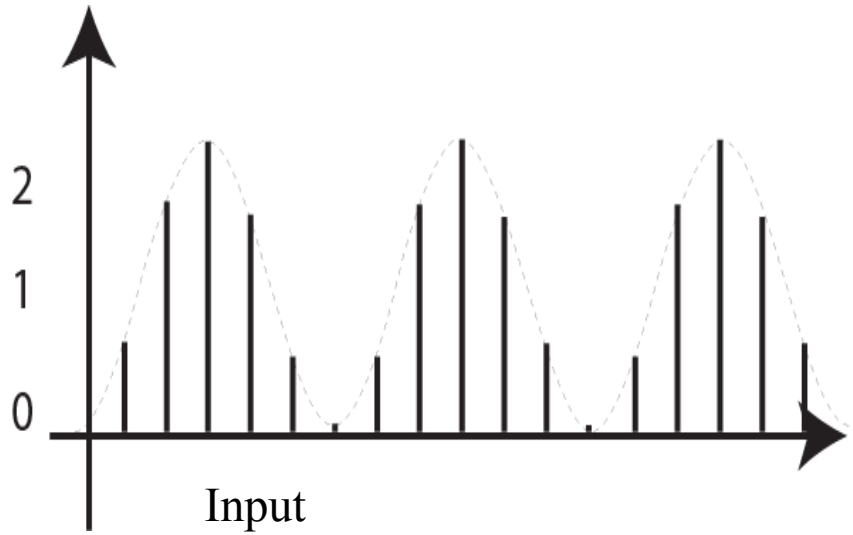


Deconvolved noise

Recall Fourier transform

- Makes convolution diagonal

Blurring: convolution

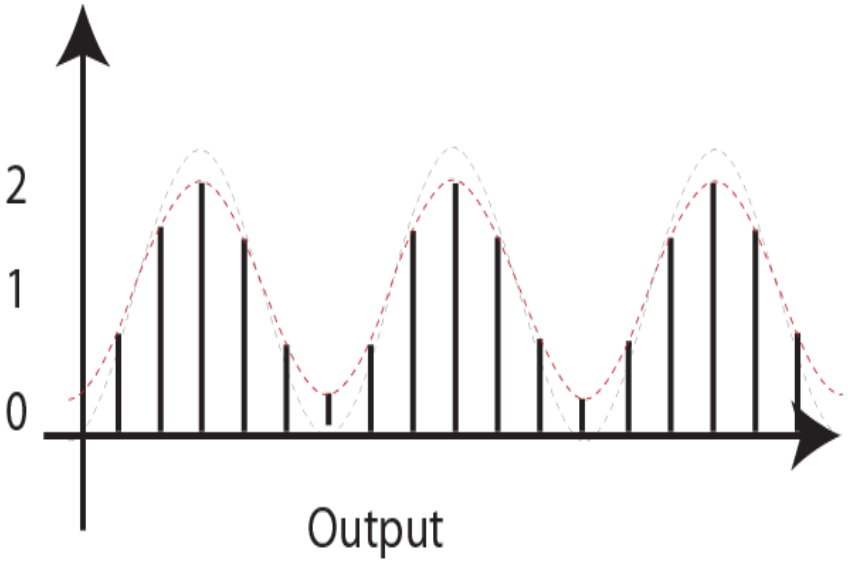




Convolution sign



Kernel

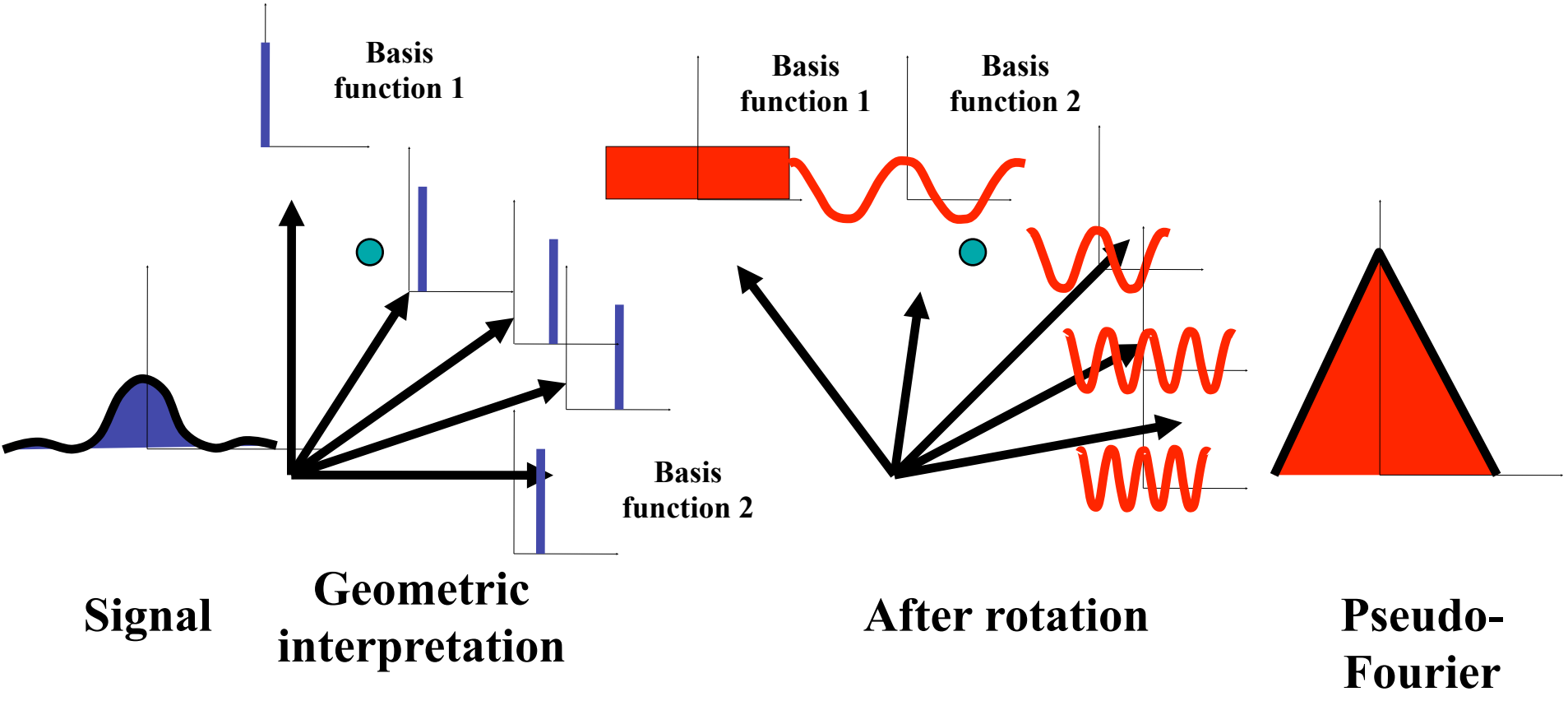


Same shape, just reduced contrast!!!

This is an eigenvector
(output is the input multiplied by a constant)

Fourier as change of basis

- Same thing with infinite-dimensional vectors

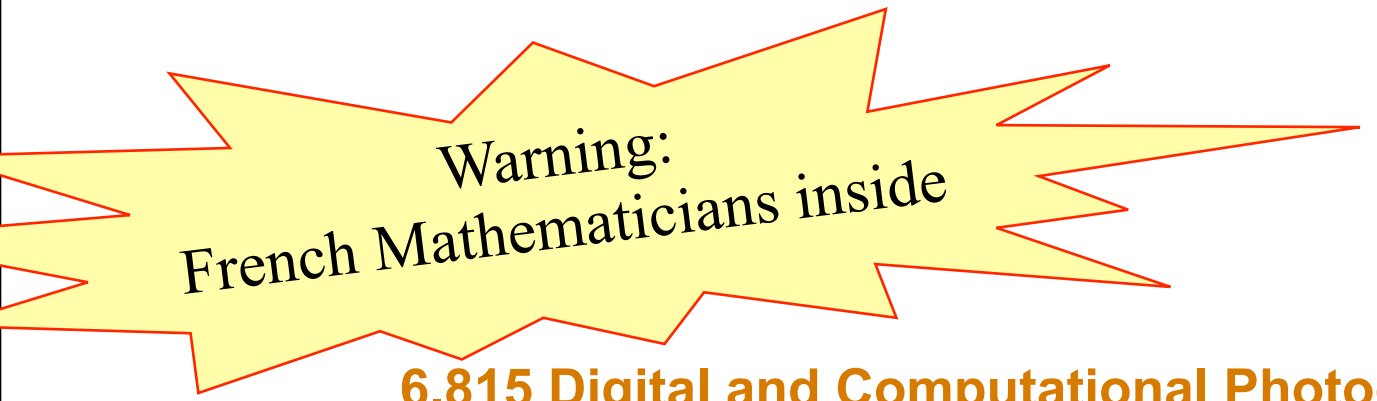


Deconvolution in Fourier

- M and M^{-1} become diagonal
- The diagonal tells us how attenuated various frequencies are
- Deconvolution divides by this factor and amplifies noise.

Better approach

- Faster convergence with conjugate gradient
- “regularize” the inversion to mitigate the effect of noise
- Understand the properties of M and M^{-1} better
  - to better fight noise
- In some cases, we’ll modify the physical blur (e.g. by changing the lens design) to make sure that deblurring doesn’t introduce as much noise



6.815 Digital and Computational Photography
6.865 Advanced Computational Photography

Gradient Image Processing

Frédo Durand
MIT - EECS

Problems with direct copy/paste

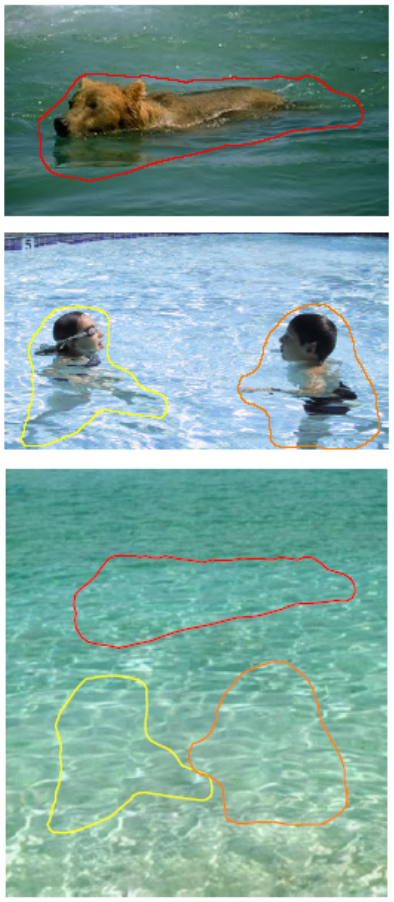


sources/destinations

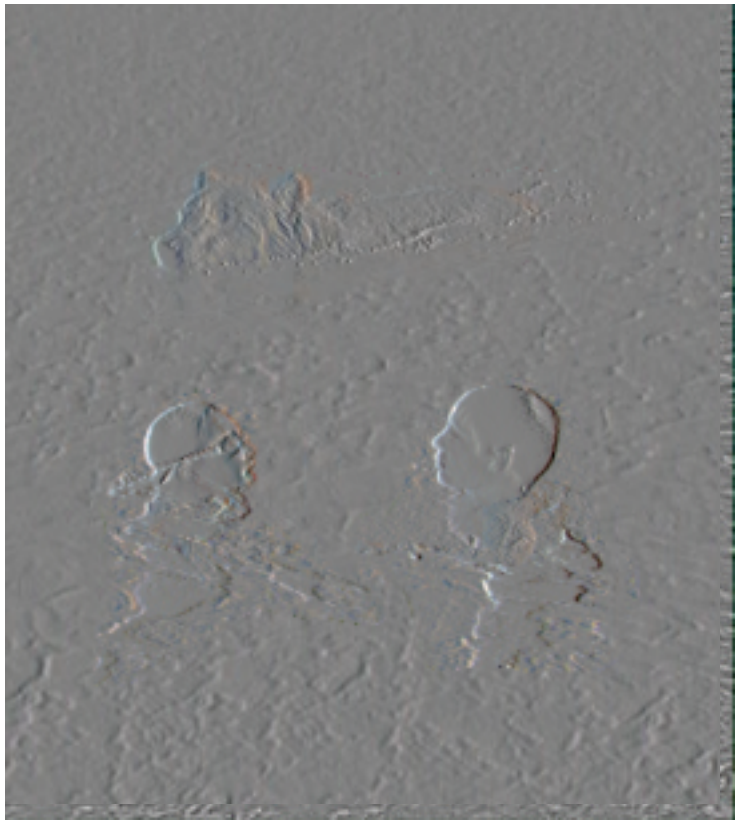
cloning

From Perez et al. 2003

Solution: paste gradient



sources/destinations



hacky visualization of gradient



seamless cloning

Demo of healing brush

- **Slightly smarter version of what we learn today**
  - higher-order derivative in particular
  - See also http://www.petapixel.com/2011/03/02/how-to-use-the-healing-brush-and-patch-tool-in-photoshop/

What is a gradient?

- **derivative of a multivariate function**
- **for example, for f(x,y)**

Different from gradient of quadratic

$$\nabla f = \left(\frac{df}{dx}, \frac{df}{dy} \right)$$

- **For a discrete image, can be approximated with finite differences**

$$\frac{df}{dx} \approx f(x+1, y) - f(x, y)$$

$$\frac{df}{dy} \approx f(x, y+1) - f(x, y)$$

Gradient vs. gradient

- **Here**

- **Quadratic**

Gradient vs. gradient

- **Here, gradient wrt 2D x,y coordinates**
  - two partial derivatives wrt x, wrt y
  - but one gradient per pixel (per channel)

- **Quadratic: gradient wrt full image values**
  - width x height x 3 partial derivatives, one for each value
  - a single giant gradient for the whole image

Gradient: intuition



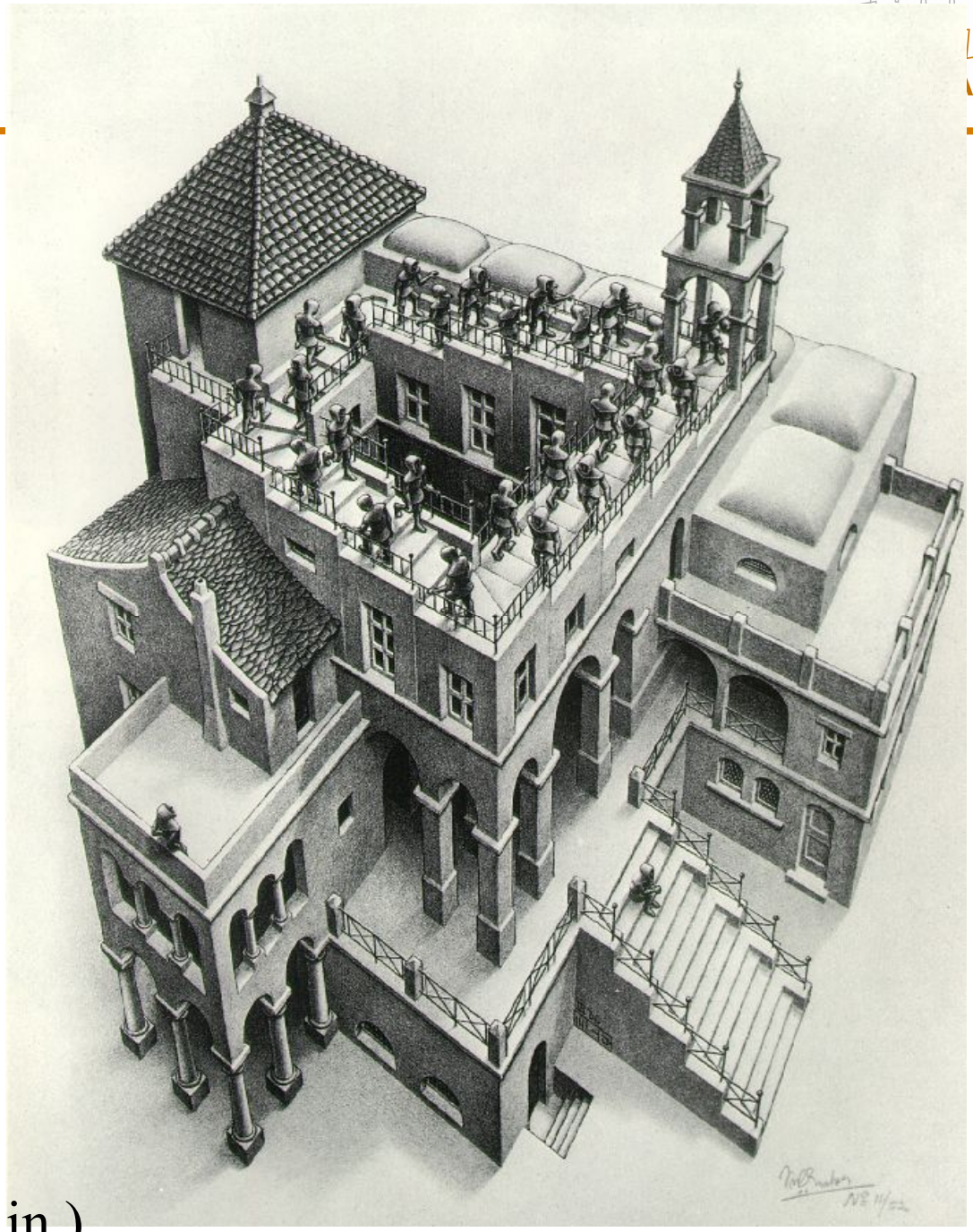
Gradients and grayscale images

- **Grayscale image: n×n scalars**
- **Gradient: n×n *2D vectors***
- **Two many numbers!**
- **What's up with this?**
- **Not all vector fields are the gradient of an image!**
- **Only if they are curl-free (a.k.a. conservative)**
  - But we'll see it does not matter for us

Color images

- 3 gradients, one for each channel.
- We'll sweep this under the rug for this lecture
- In practice, treat each channel independently

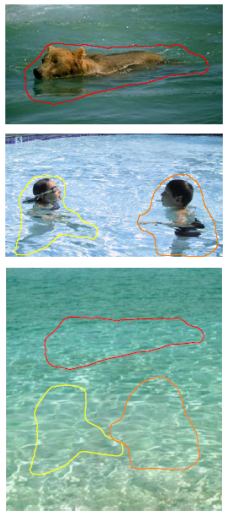
Questions?



Escher, Maurits Cornelis
Ascending and Descending
1960
Lithograph
35.5 x 28.5 cm (14 x 11 1/4 in.)

Seamless Poisson cloning

- **Paste source gradient into target image inside a selected region**
- **Make the new gradient as close as possible to the source gradient while respecting pixel values at the boundary**



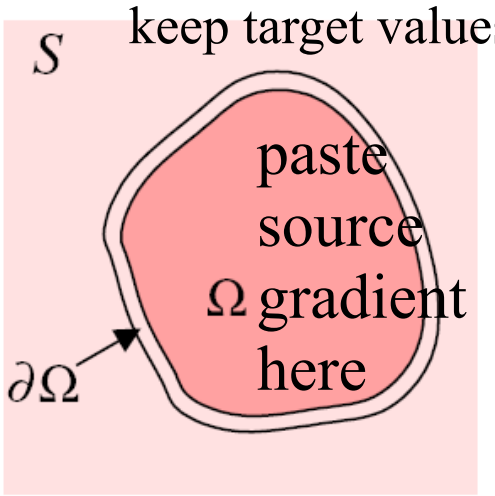
sources/destinations



cloning



seamless cloning



Seamless Poisson cloning

- **Given vector field *v* (pasted gradient), find the value of *f* in unknown region that optimize:**

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Poisson equation with Dirichlet conditions

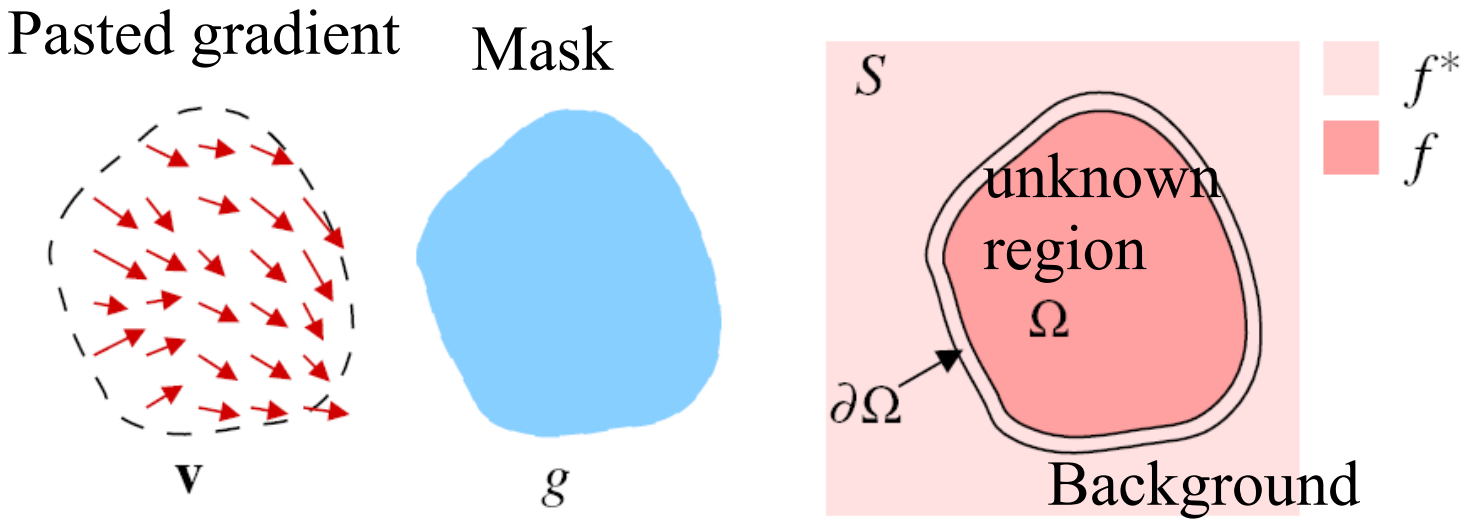


Figure 1: **Guided interpolation notations**. Unknown function f interpolates in domain Ω the destination function f^*, under guidance of vector field $\mathbf{v}$, which might be or not the gradient field of a source function g.

Discrete 2D Poisson

- Simple gradient kernels are [-1, 1] and $[-1, 1]^T$
  - Not Sobel because because it's optimized to analyze gradient
    - support is a little big
    - it blurs things too much
  - We're trying to keep equations simples and to stick as directly as possible to image values
- Big linear least squares:
  - minimize

Discrete 2D Poisson

Discrete 2D Poisson

paste gradient of source S into image I using finite differences:

$$\min_I \sum_{\text{pixel } y,x} \left(\frac{\partial I}{\partial x} - \frac{\partial S}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y} - \frac{\partial S}{\partial y}\right)^2$$

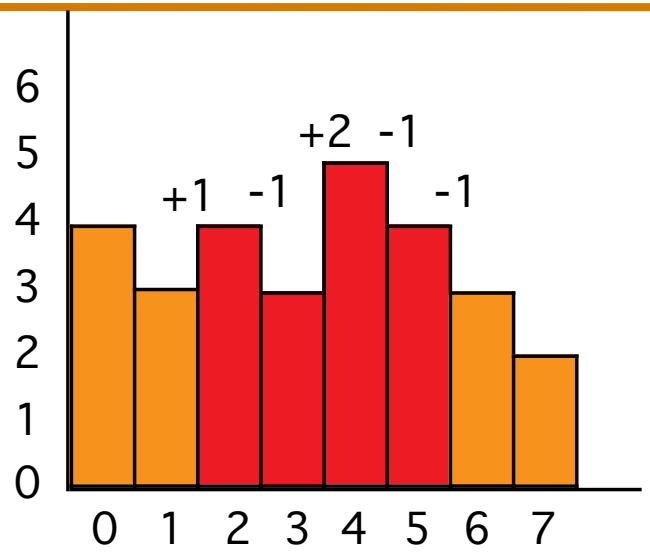
$$\min_I \sum_{\text{pixel } p=y,x} \Big[\big(I(y,x) - I(y,x-1)\big) - \big(S(y,x) - S(y,x-1)\big)\Big]^2 +$$
$$\Big[\big(I(y,x) - I(y-1,x)\big) - \big(S(y,x) - S(y-1,x)\big)\Big]^2$$

+ boundary conditions: keep old values of I

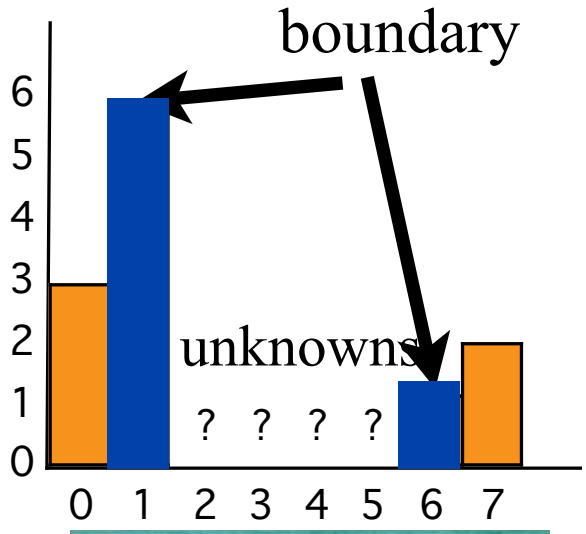
- **Big linear least squares!**

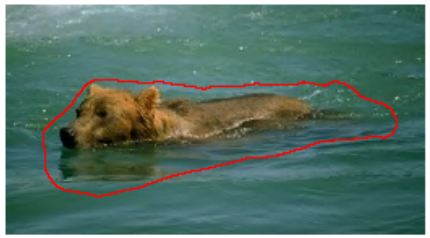
Discrete 1D example: minimization

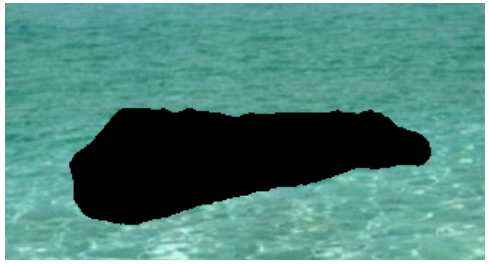
- **Copy**



to



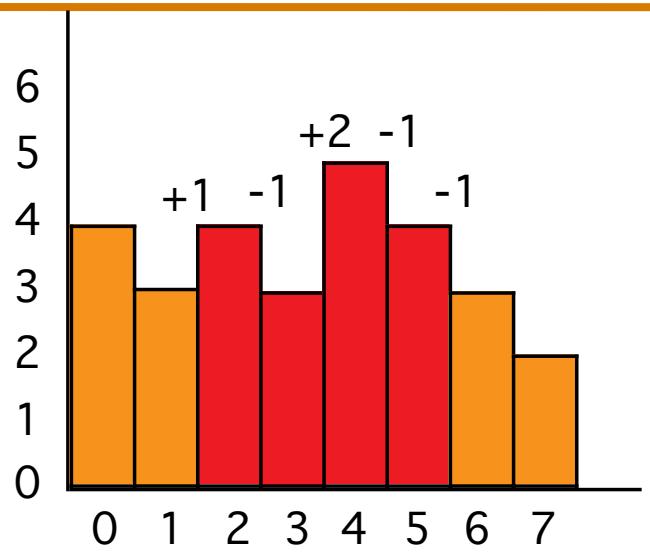




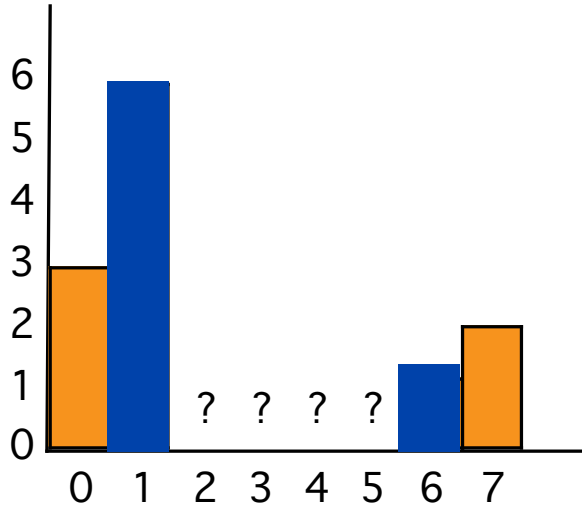
orange: pixel outside the mask
red: source pixel to be pasted
blue: boundary conditions (in background)

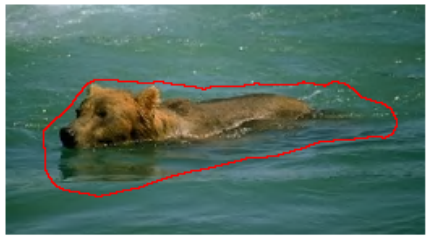
Discrete 1D example: minimization

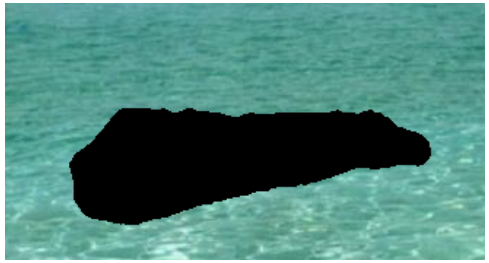
- **Copy**



to







$$\text{Min } [(f_2-f_1)-1]^2$$
$$+ [(f_3-f_2)-(-1)]^2$$
$$+ [(f_4-f_3)-2]^2$$
$$+ [(f_5-f_4)-(-1)]^2$$
$$+ [(f_6-f_5)-(-1)]^2$$

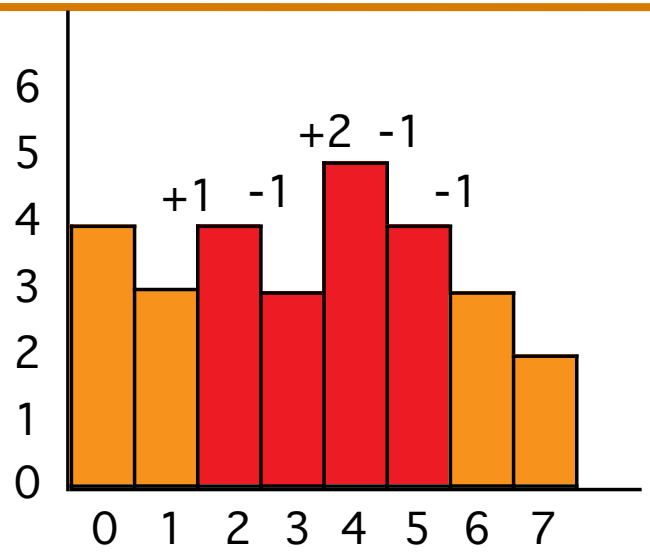
With

$f_1=6$

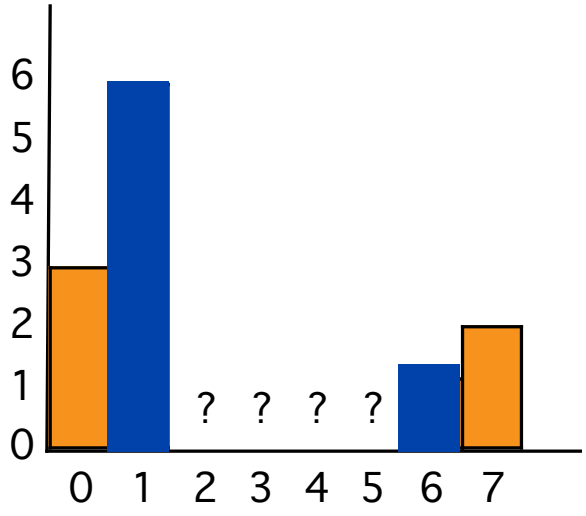
$f_6=1$

1D example: minimization

- **Copy**



to



$$\text{Min } [(f_2-f_1)-1]^2 \quad ==> \quad f_2^2+49-14f_2$$

$$+ [(f_3-f_2)-(-1)]^2 \quad ==> \quad f_3^2+f_2^2+1-2f_3f_2 +2f_3-2f_2$$

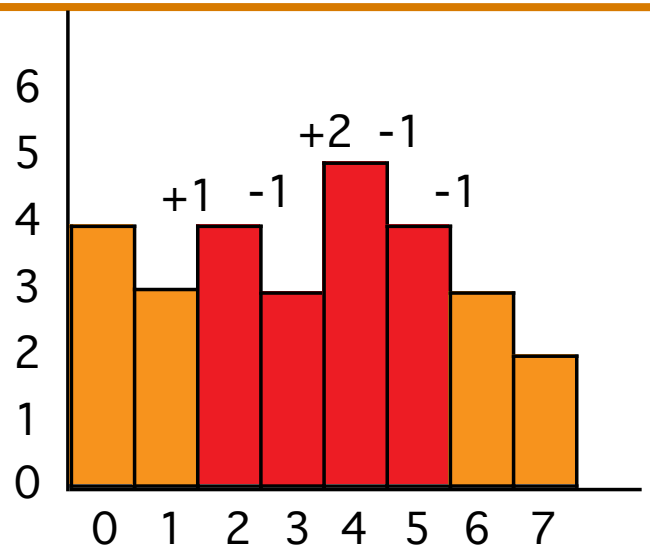
$$+ [(f_4-f_3)-2]^2 \quad ==> \quad f_4^2+f_3^2+4-2f_3f_4 -4f_4+4f_3$$

$$+ [(f_5-f_4)-(-1)]^2 \quad ==> \quad f_5^2+f_4^2+1-2f_5f_4 +2f_5-2f_4$$

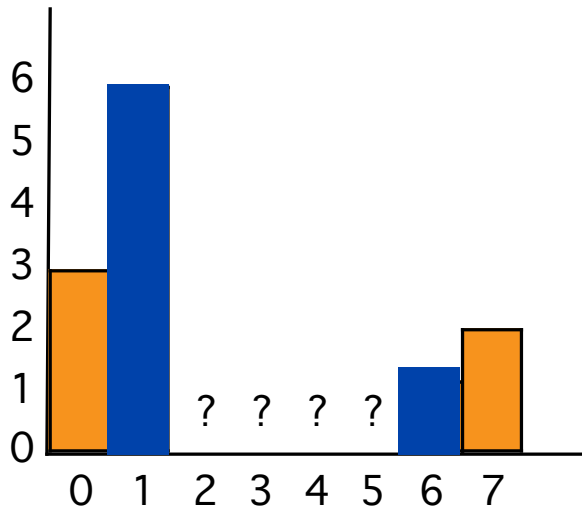
$$+ [(f_6-f_5)-(-1)]^2 \quad ==> \quad f_5^2+4-4f_5$$

1D example: big quadratic

- **Copy**



to



- **Min (f_2^2+49-14f_2**

  + f_3^2+f_2^2+1-2f_3f_2 +2f_3-2f_2

  + f_4^2+f_3^2+4-2f_3f_4 -4f_4+4f_3

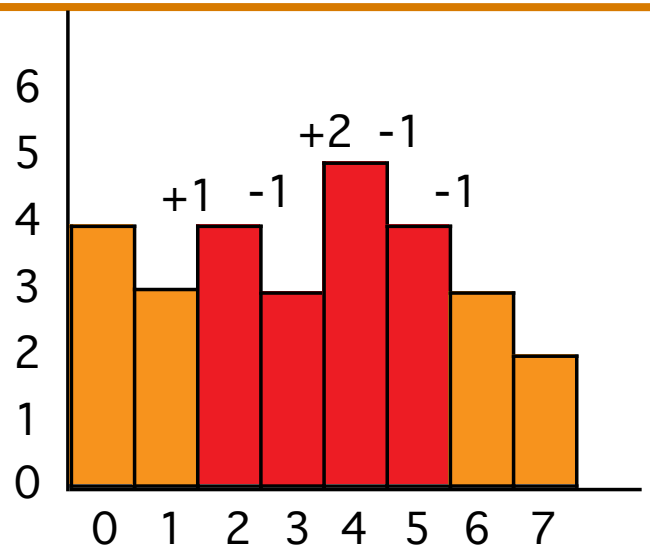
  + f_5^2+f_4^2+1-2f_5f_4 +2f_5-2f_4

  + f_5^2+4-4f_5)

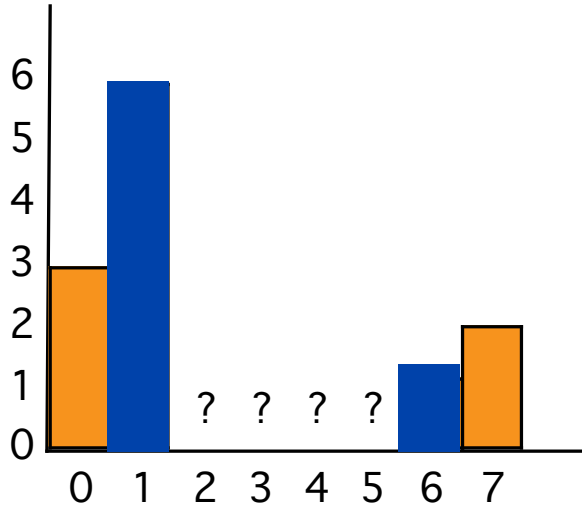
  Denote it Q

1D example: derivatives

- **Copy**



to



Min (f_2^2+49-14f_2

+ f_3^2+f_2^2+1-2f_3f_2 +2f_3-2f_2

+ f_4^2+f_3^2+4-2f_3f_4 -4f_4+4f_3

+ f_5^2+f_4^2+1-2f_5f_4 +2f_5-2f_4

+ f_5^2+4-4f_5)

Denote it Q

$$\frac{dQ}{df_2} = 2f_2 + 2f_2 - 2f_3 - 16$$

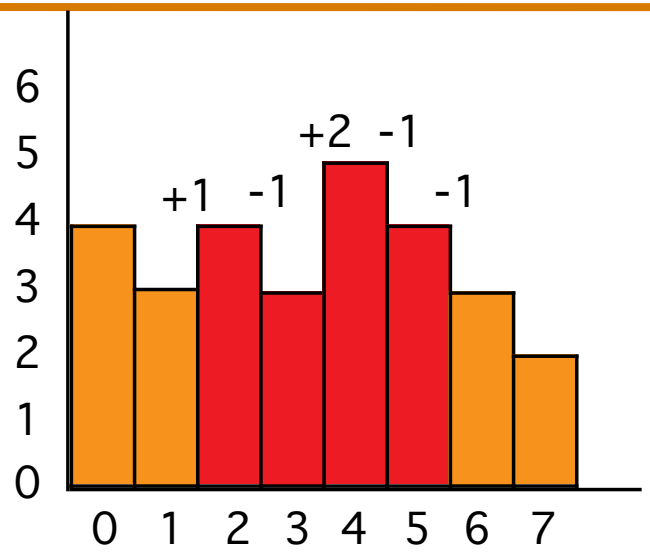
$$\frac{dQ}{df_3} = 2f_3 - 2f_2 + 2 + 2f_3 - 2f_4 + 4$$

$$\frac{dQ}{df_4} = 2f_4 - 2f_3 - 4 + 2f_4 - 2f_5 - 2$$

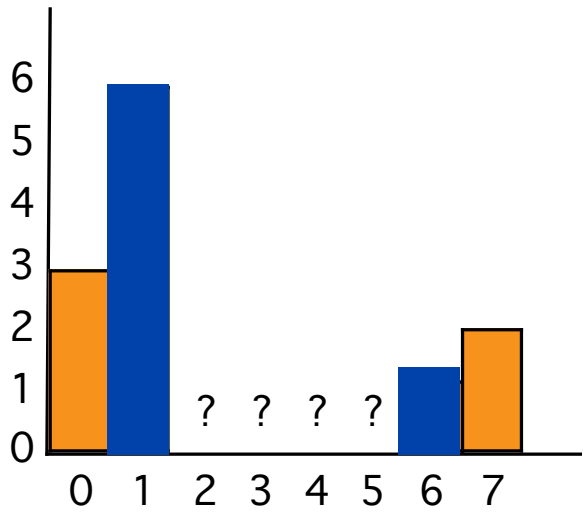
$$\frac{dQ}{df_5} = 2f_5 - 2f_4 + 2 + 2f_5 - 4$$

1D example: set derivatives to zero

- **Copy**



to



$$\frac{dQ}{df_2} = 2f_2 + 2f_2 - 2f_3 - 16 \quad = 0$$

$$\frac{dQ}{df_3} = 2f_3 - 2f_2 + 2 + 2f_3 - 2f_4 + 4 \quad = 0$$

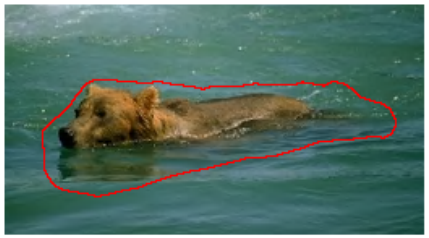
$$\frac{dQ}{df_4} = 2f_4 - 2f_3 - 4 + 2f_4 - 2f_5 - 2 \quad = 0$$

$$\frac{dQ}{df_5} = 2f_5 - 2f_4 + 2 + 2f_5 - 4 = 0$$

$$\Longrightarrow \begin{pmatrix} 4 & -2 & 0 & 0 \\ -2 & 4 & -2 & 0 \\ 0 & -2 & 4 & -2 \\ 0 & 0 & -2 & 4 \end{pmatrix} \begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 16 \\ -6 \\ 6 \\ 2 \end{pmatrix}$$

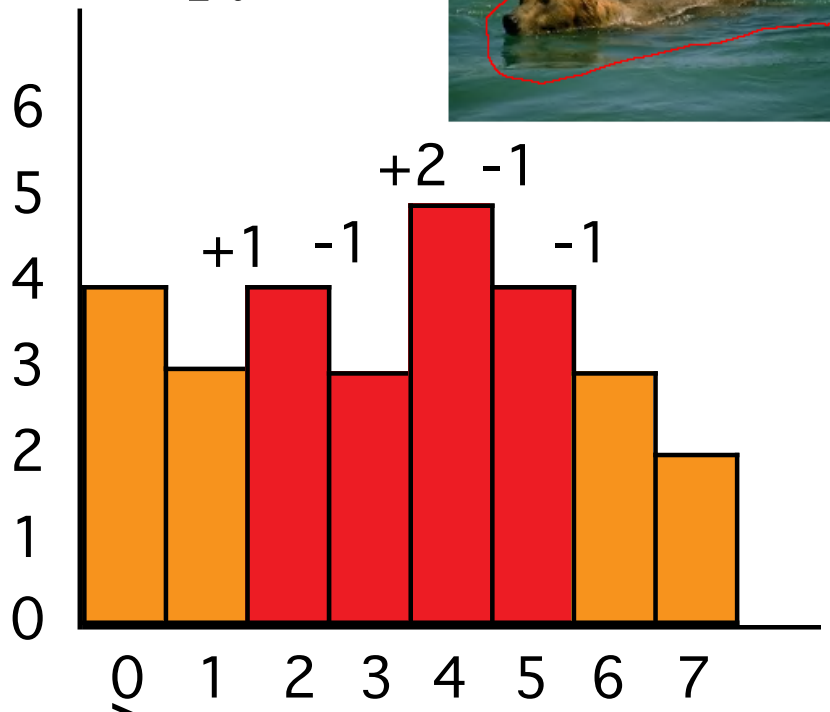
1D example recap

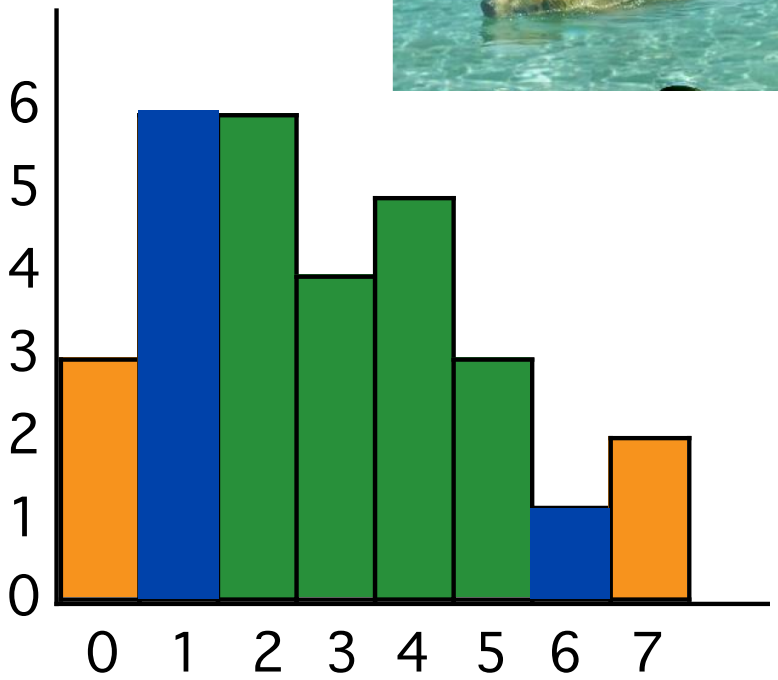
- **Copy**



to







$$\Longrightarrow$$

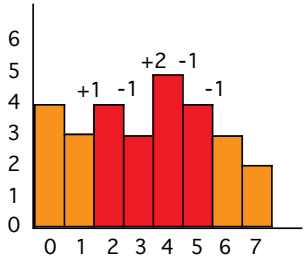
$$\begin{pmatrix} 4 & -2 & 0 & 0 \\ -2 & 4 & -2 & 0 \\ 0 & -2 & 4 & -2 \\ 0 & 0 & -2 & 4 \end{pmatrix} \begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 16 \\ -6 \\ 6 \\ 2 \end{pmatrix}$$

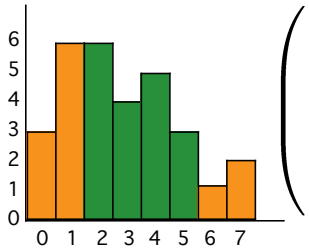
$$\begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 6 \\ 4 \\ 5 \\ 3 \end{pmatrix}$$

Questions?

- **Recap:**
  - copy gradient, not pixel values
  - enforce boundary condition
  - solve linear least square:
    minimize square difference with source gradient

1D example: remarks

- **Copy** 

to 

$$\begin{pmatrix} 4 & -2 & 0 & 0 \\ -2 & 4 & -2 & 0 \\ 0 & -2 & 4 & -2 \\ 0 & 0 & -2 & 4 \end{pmatrix} \begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 16 \\ -6 \\ 6 \\ 2 \end{pmatrix}$$

- **Matrix is sparse**
  - many zero coefficients
  - because gradient only depends on neighboring pixels
- **Matrix is symmetric**
- **Everything is a multiple of 2**
  - because square and derivative of square
- **Matrix is a convolution (kernel -2 4 -2)**
  - all the rows are the same, just shifted
- **Matrix is independent of gradient field. Only RHS is**
- **Matrix is a second derivative**

transpose and flip

- **Gradient is [-1, 1]**
- **flipped gradient is**
- **gradient convolved by flipped gradient is**

transpose and flip

- **Gradient is [-1, 1]**
- **flipped gradient is [1, -1]**
- **gradient convolved by flipped gradient is**
- **… [-1, 2, -1]**

In 2D

- **gradient is [[-1, 1]] and [[-1], [1]]**
- **for each component, you convolve by its flipped version and get**
  - [-1, 2, -1] and [[-1], [2], [-1]]
- **you sum them to get the full derivative wrt the image:**

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

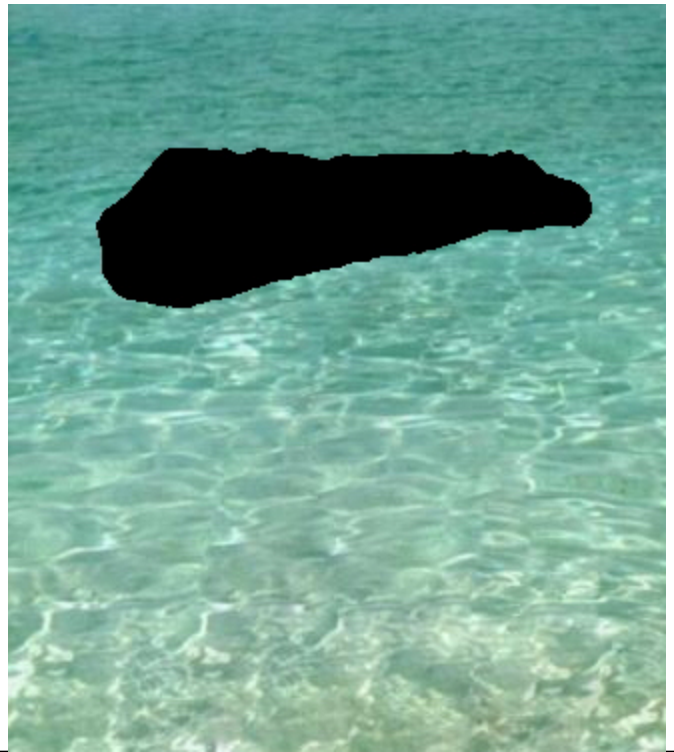
Questions?

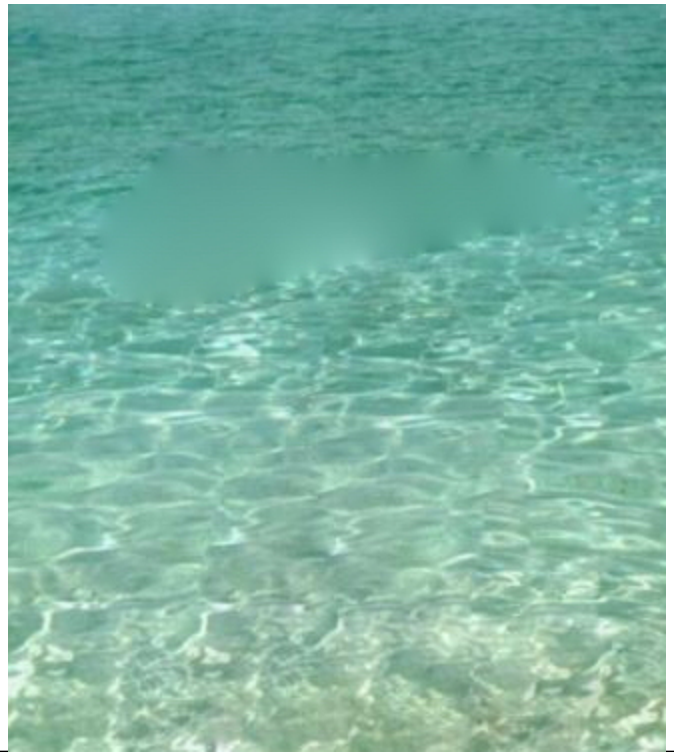
CSAIL

Membrane interpolation

- **What if source gradient *v* is null?**
- **Laplace equation (a.k.a. membrane equation)**

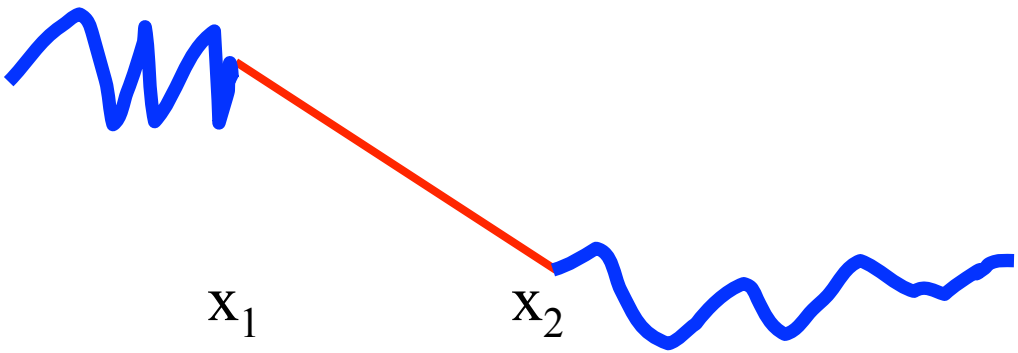
$$\min_f \iint_\Omega |\nabla f|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$



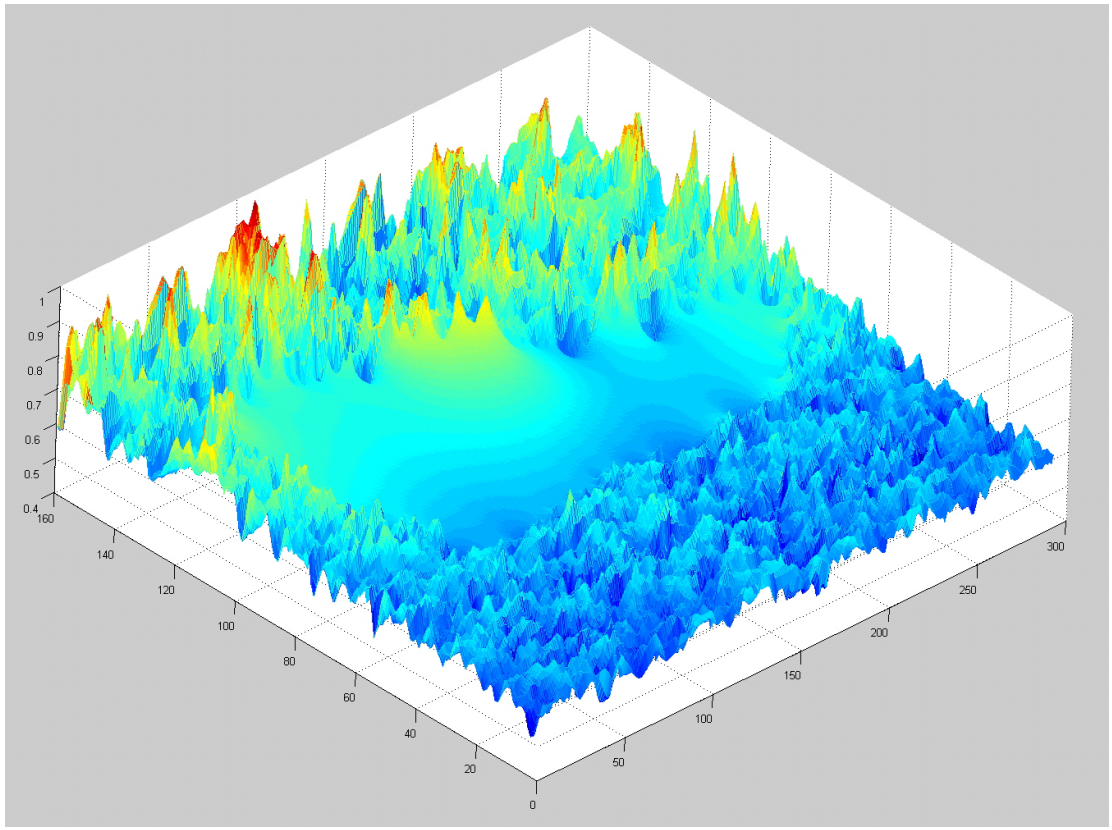


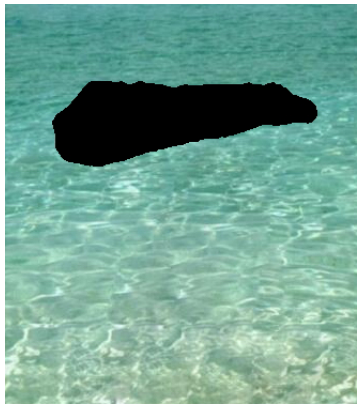
Intuition

- **In 1D; just linear interpolation!**
- **Locally, if the second derivative was not zero, this would mean that the first derivative is varying, which is bad since we want $(\nabla f)^2$ to be minimized**
- **Note that, in 1D: by setting f'', we leave two degrees of freedom. This is exactly what we need to control the boundary condition at x_1 and x_2**

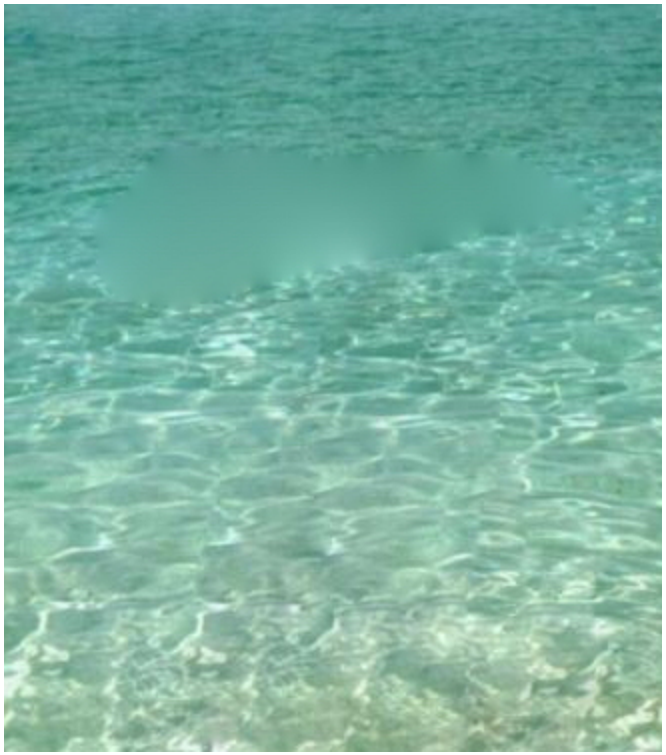


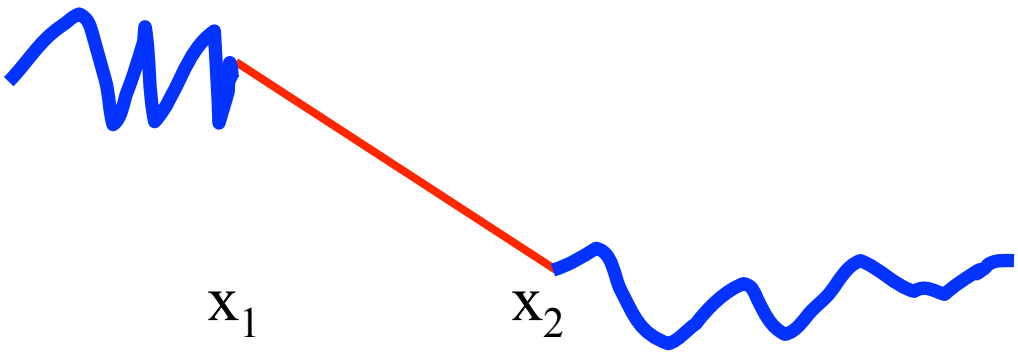
In 2D: membrane interpolation





Not as simple





Questions?

CSAIL

Linear algebra to the rescue

Images and linear algebra

- We want to use powerful tools in linear algebra
  - System Ax=b
- Our unknown *x* is an image
- We could solve Poisson integration by stacking all our pixels into a big numpy 1D vector x, build a big matrix A, solve for Ax=b, then copy the pixels back into the image
- Instead, we'll keep images as numpy arrays, but we'll use abstract derivations in terms of A and x to know what to do
- The implementation will use simple image operations such as convolution, multiplication

Gradient

- **The gradient is a linear operation**
- **Can be expressed as a matrix G**
  - Gx is the gradient of image x
  - (stacks the x and y gradient components)
- **Poisson equation:**
  min $\|Gx-Gs\|^2$
  - where x is the unknown output image and s is the source image
  - We'll talk about boundary conditions later

Poisson equation

- **Poisson equation:**
  min ||Gx-Gs||²
- **||v||² is the L2 norm between two vectors**
  - ||v||² =v.v = v^T v

$$\min ||Gx - Gs||^2$$

$$(Gx - Gs)^T (Gx - Gs)$$

$$x^T (G^T G) x - 2x^T (G^T G) s + s^T (G^T G) s$$

- **Define** $A = G^T G$

$$\min x^T A x - 2x^T A s + s^T A s$$

Laplacian

- We defined $A = G^T G$
- It is the composition of two gradient operations
- => It is a second derivative (Laplacian)

- If the gradient kernels are [-1, 1] and $[-1, 1]^T$
- The Laplacian kernel is

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

- Recall the matrix in 1D:

$$\begin{pmatrix} 4 & -2 & 0 & 0 \\ -2 & 4 & -2 & 0 \\ 0 & -2 & 4 & -2 \\ 0 & 0 & -2 & 4 \end{pmatrix}$$

Back to minimization

- Given source image s, find image x that minimizes the energy

$$Q(x) = x^T Ax - 2x^T As + s^T As$$

- How do you minimize?
- Set derivatives to zero!

Derivative wrt the image

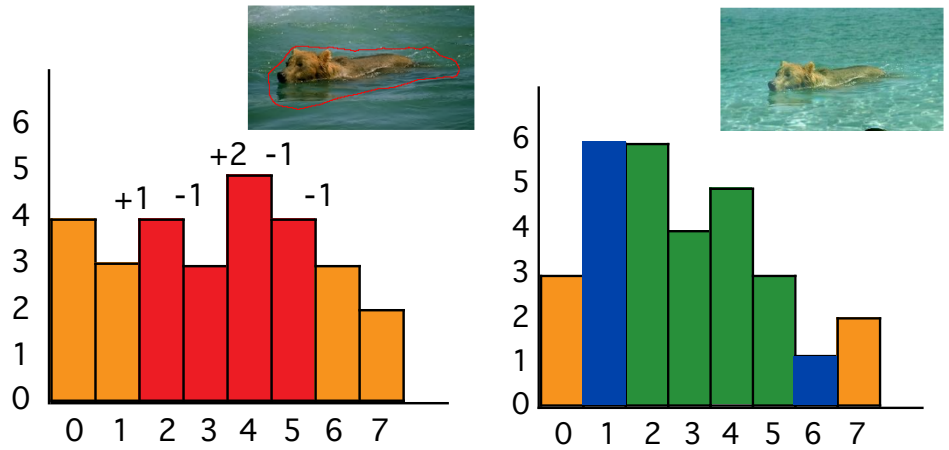
$$Q(x) = x^T Ax - 2x^T As + s^T As$$

$$\frac{dQ}{dx} = 2Ax - 2As$$

- Recall that x is a full image.
- Not surprisingly, we derive a quadratic and get a linear equation
- In summary: solution to Poisson minimization is:

$$\boxed{Ax = As}$$

Recall the 1D case



$$\begin{pmatrix} 4 & -2 & 0 & 0 \\ -2 & 4 & -2 & 0 \\ 0 & -2 & 4 & -2 \\ 0 & 0 & -2 & 4 \end{pmatrix} \begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 16 \\ -6 \\ 6 \\ 2 \end{pmatrix}$$

- Same as

$$Ax = As$$

Difference with deconvolution

- **Deconvolution least squares:**
  - $M^T M x - M^T y = 0$
  - where his a blurred image
- **Poisson least squares**
  - $G^T G x - G^T G s = 0$
  - we apply the full $G^T G$ to s, not just G^T
  - because the equivalent of the blurred image is the source convolved by the gradient (the source is not convolved already like y)

Recap

- We have shown that minimizing a quadratic

$$\min x^T Ax - 2x^T As + s^T As$$

  is the same as solving a linear system

$$Ax = As$$

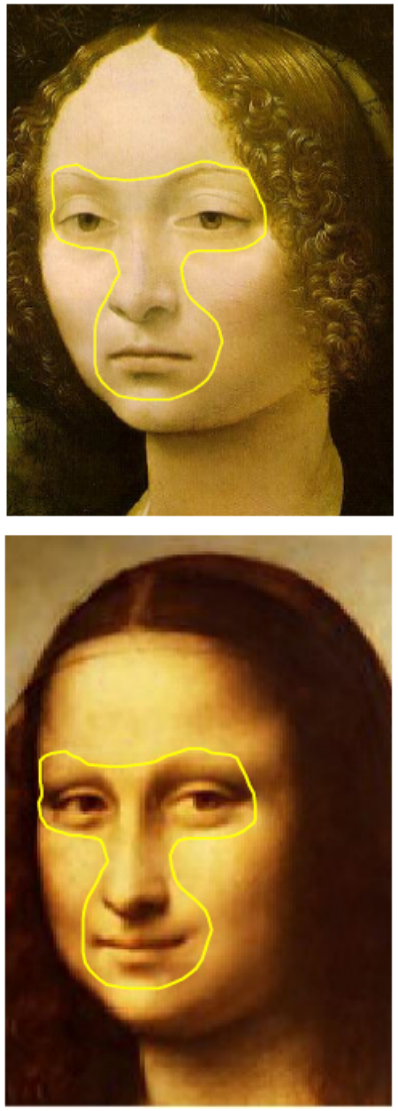
- In the case of Poisson, the matrix is a second derivative convolution with the Laplacian kernel

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

- We want the Laplacian of the output to be the same as that of the source

Eye candy

Result (eye candy)



source/destination

cloning

seamless cloning



sources | destinations | cloning | seamless cloning



Figure 2: **Concealment**. By importing seamlessly a piece of the background, complete objects, parts of objects, and undesirable artifacts can easily be hidden. In both examples, multiple strokes (not shown) were used.

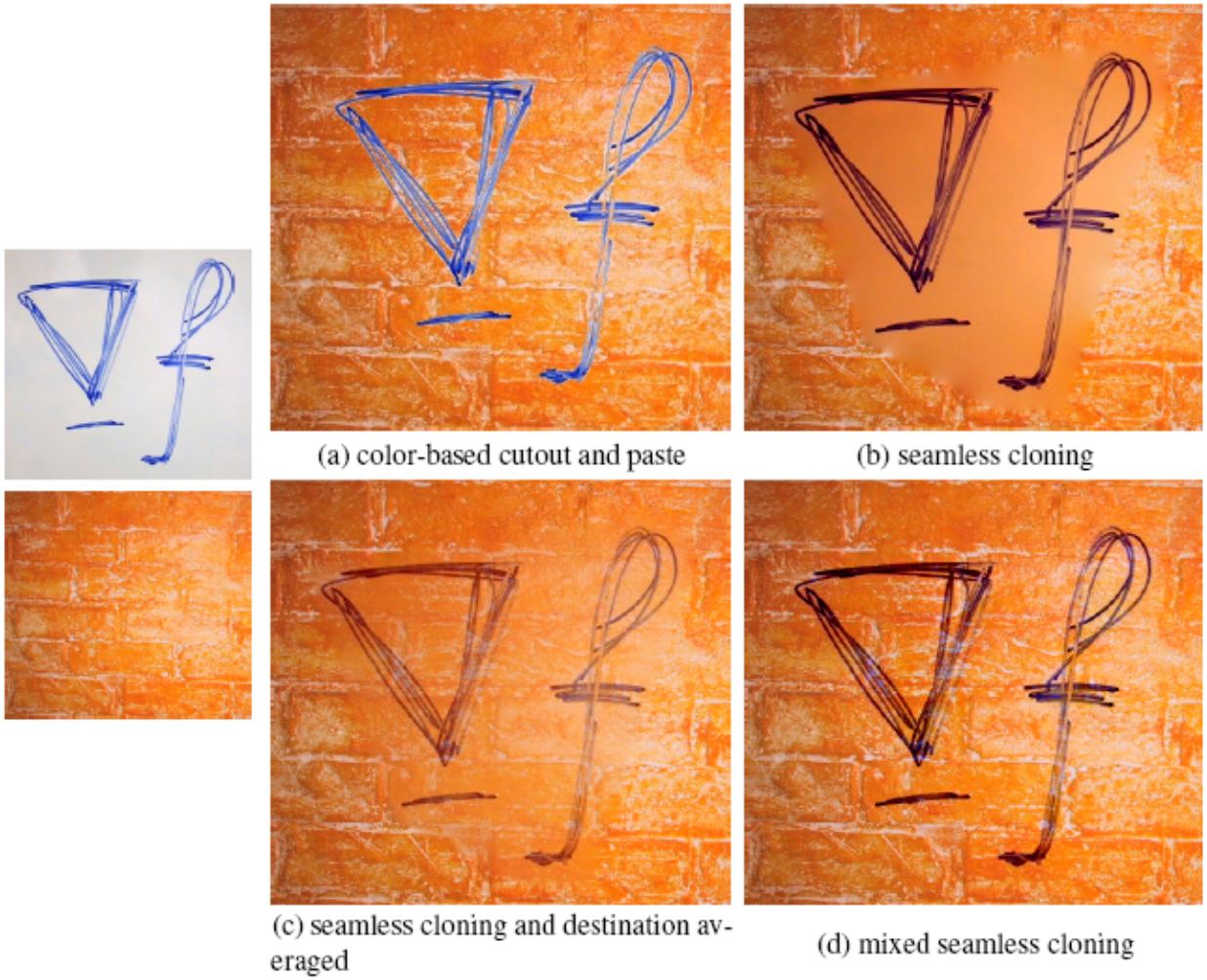
Manipulate the gradient

- **Mix gradients of g & f: take the max**



source/destination | seamless cloning | mixed seamless cloning

Figure 8: **Inserting one object close to another**. With seamless cloning, an object in the destination image touching the selected region Ω bleeds into it. Bleeding is inhibited by using mixed gradients as the guidance field.



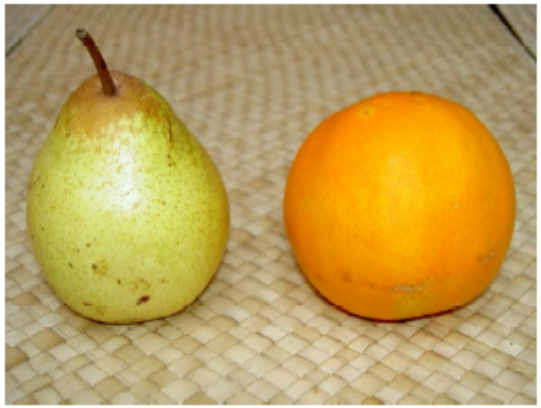
(a) color-based cutout and paste

(b) seamless cloning

(c) seamless cloning and destination averaged

(d) mixed seamless cloning

Figure 6: **Inserting objects with holes**. (a) The classic method, color-based selection and alpha masking might be time consuming and often leaves an undesirable halo; (b-c) seamless cloning, even averaged with the original image, is not effective; (d) mixed seamless cloning based on a loose selection proves effective.





swapped textures



source



destination



Figure 7: **Inserting transparent objects**. Mixed seamless cloning facilitates the transfer of partly transparent objects, such as the rainbow in this example. The non-linear mixing of gradient fields picks out whichever of source or destination structure is the more salient at each location.

Questions?

CSAIL

Gradient descent for Poisson

Recall gradient descent

$$r_i = b - Ax_i$$

$$\alpha = \frac{r_i \cdot r_i}{r_i \cdot Ar_i}$$

$$x_{i+1} = x_i + \alpha r_i$$

- **x is an unknown image**
- **A is the Laplacian operator**
  - Multiplying by A just means computing a convolution
  - Ax is the convolved image
- **b is the Laplacian of the source**
- **r_i is also an image**

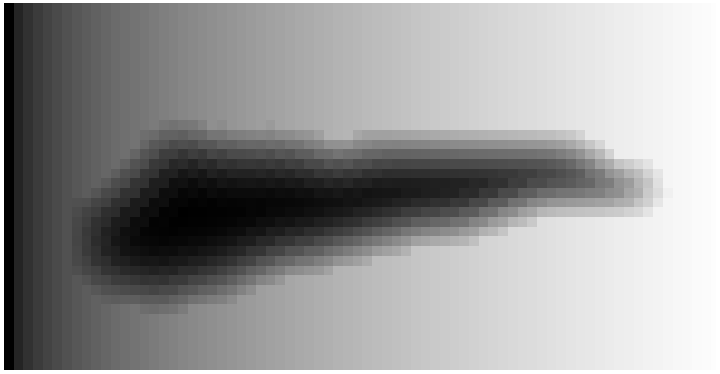
Boundary conditions

- How do we deal with the mask?
- Only update pixels inside the mask
- But maintain an image x that includes background pixels

x is the full rectangle

The Laplacian is computed over the full rectangle

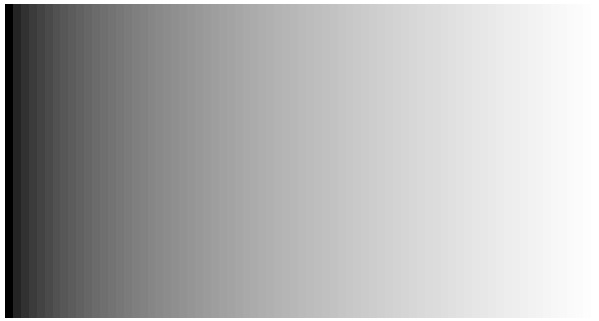
ri is restricted to the masked region





Gradient descent results

Background



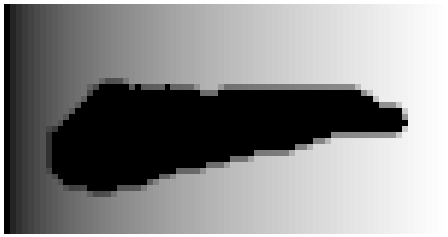
Source



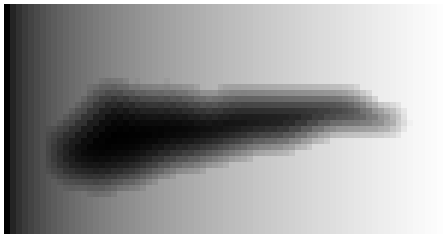
Mask



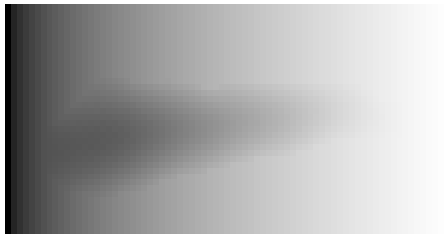
Initialization



10 iterations



100 iterations



200 iterations

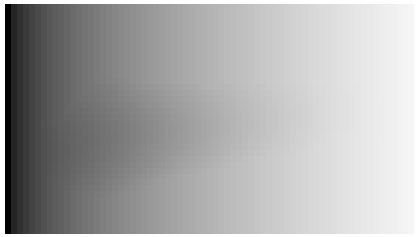


Image terms

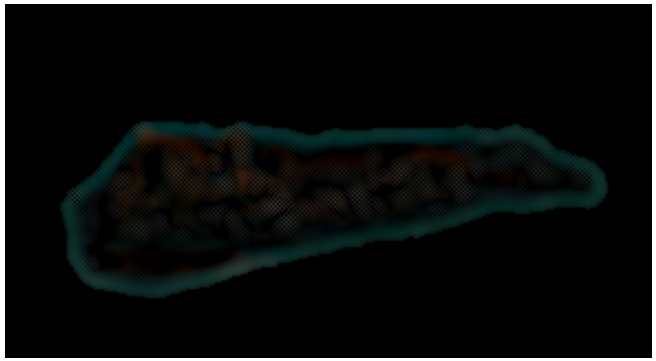
xi



Axi



ri



$$r_i = b - Ax_i$$

$$\alpha = \frac{r_i \cdot r_i}{r_i \cdot Ar_i}$$

$$x_{i+1} = x_i + \alpha r_i$$

Important: r_i was masked. All pixel values outside the white region of the mask are zero

Bottomline

- We solve Ax=b
- But we never form or store matrix A
- We just do repeated convolutions and a few image dot products.

$$r_i = b - Ax_i$$

convolution, subtraction

$$\alpha = \frac{r_i \cdot r_i}{r_i \cdot Ar_i}$$

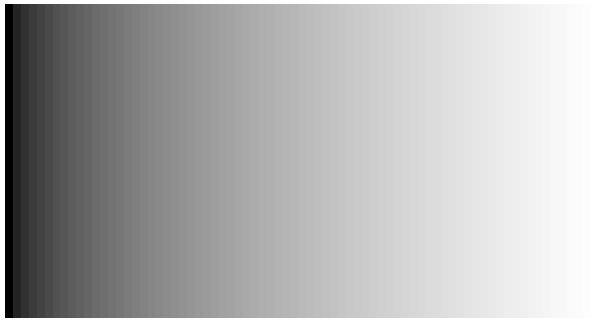
Dot product, convolution, dot product, division of two scalars

$$x_{i+1} = x_i + \alpha r_i$$

multiplication by scalar, addition

Gradient descent results

Background



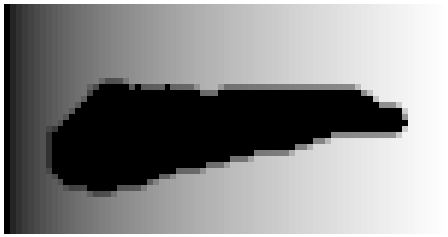
Source



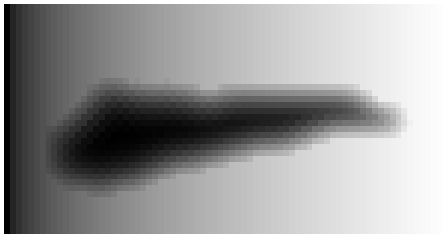
Mask



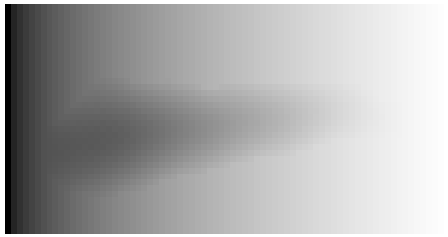
Initialization



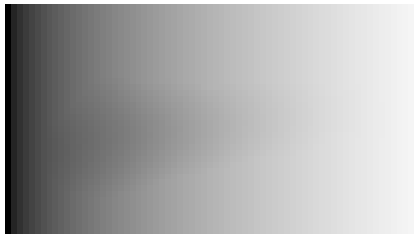
10 iterations



100 iterations



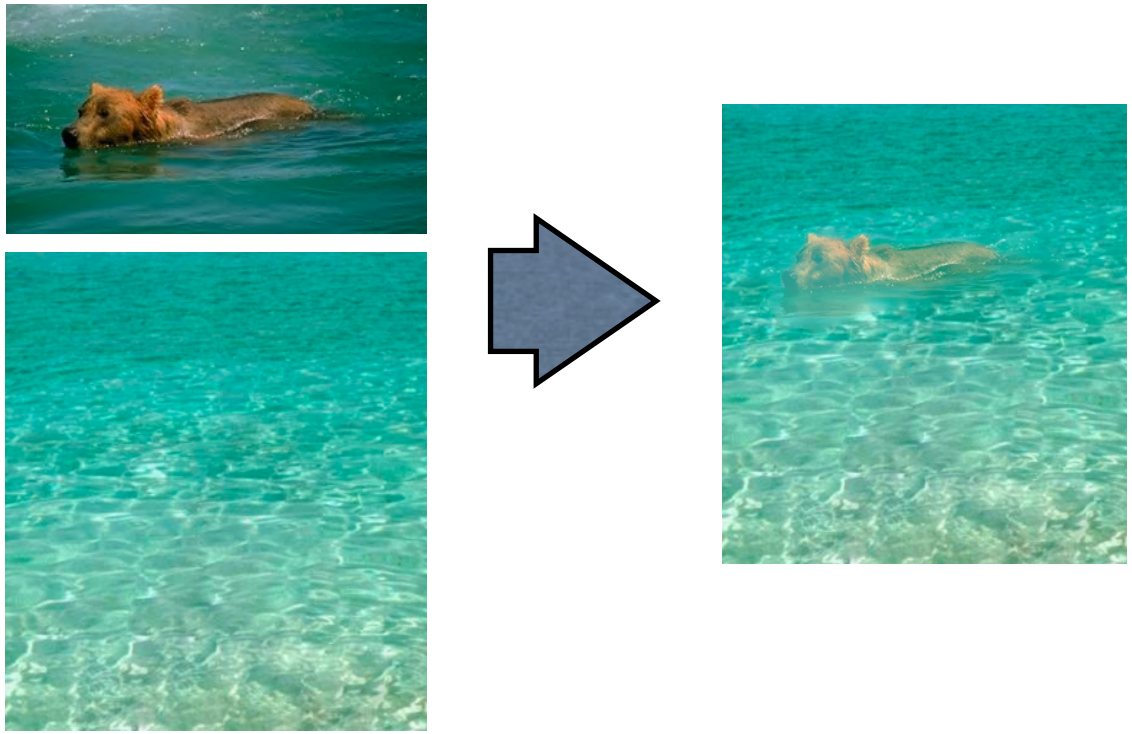
200 iterations



Bells and whistles

Contrast problem

- Contrast is a multiplicative quantity
- With Poisson, we try to reproduce linear differences
- Loss of contrast if pasting from dark to bright



Contrast preservation: use the log



Poisson in linear color space



Poisson in log color space

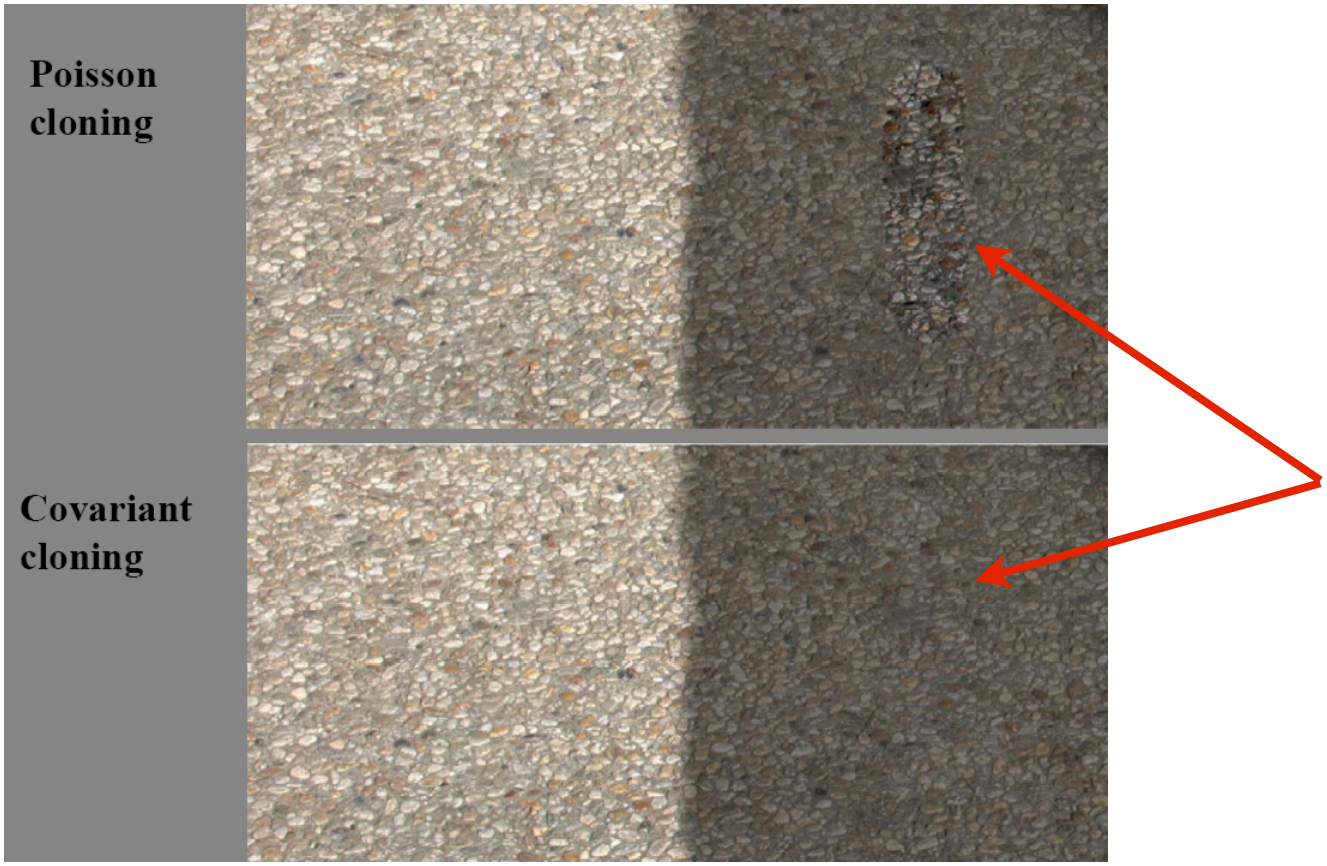
- see A Perception-based Color Space for Illumination-invariant Image Processing

  http://www.eecs.harvard.edu/~hchong/thesis/color_siggraph08.pdf

- Or use covariant derivatives (next slides)

Covariant derivatives & Photoshop

- **Photoshop Healing brush**
- **First developed (before Poisson editing) by Todor Georgiev (Adobe, now Qualcomm)**

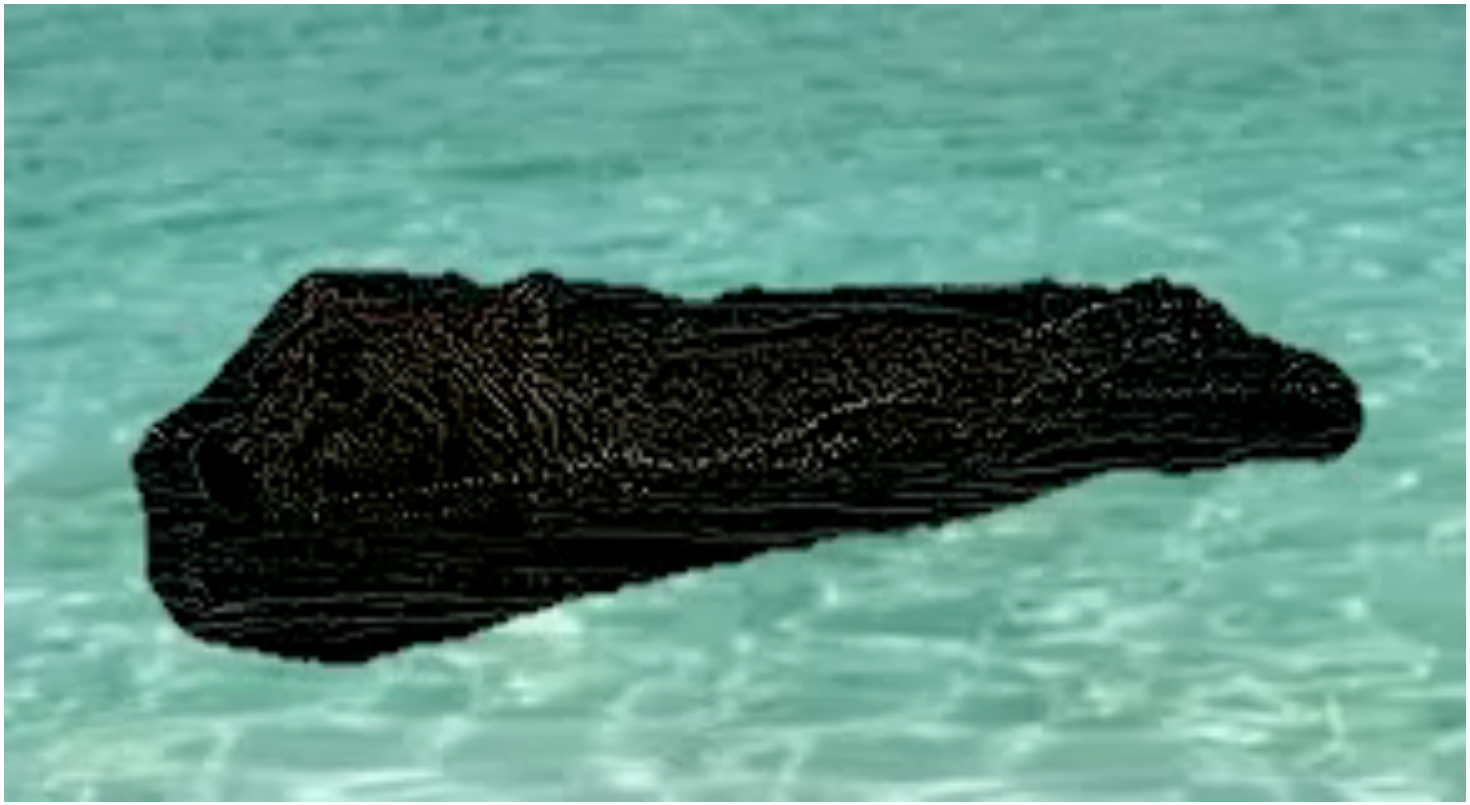


From Todor Georgiev's slides http://photo.csail.mit.edu/posters/todor_slides.pdf

Convergence

Convergence

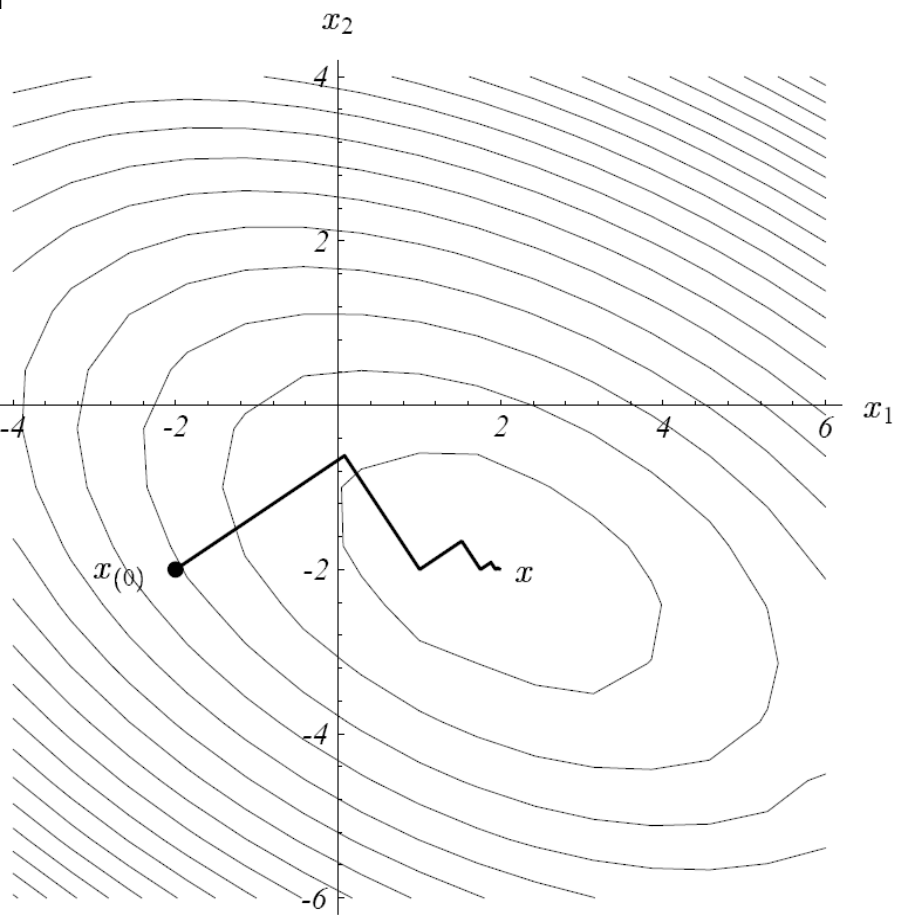
- A little slow: not fully there yet after 1000 iterations



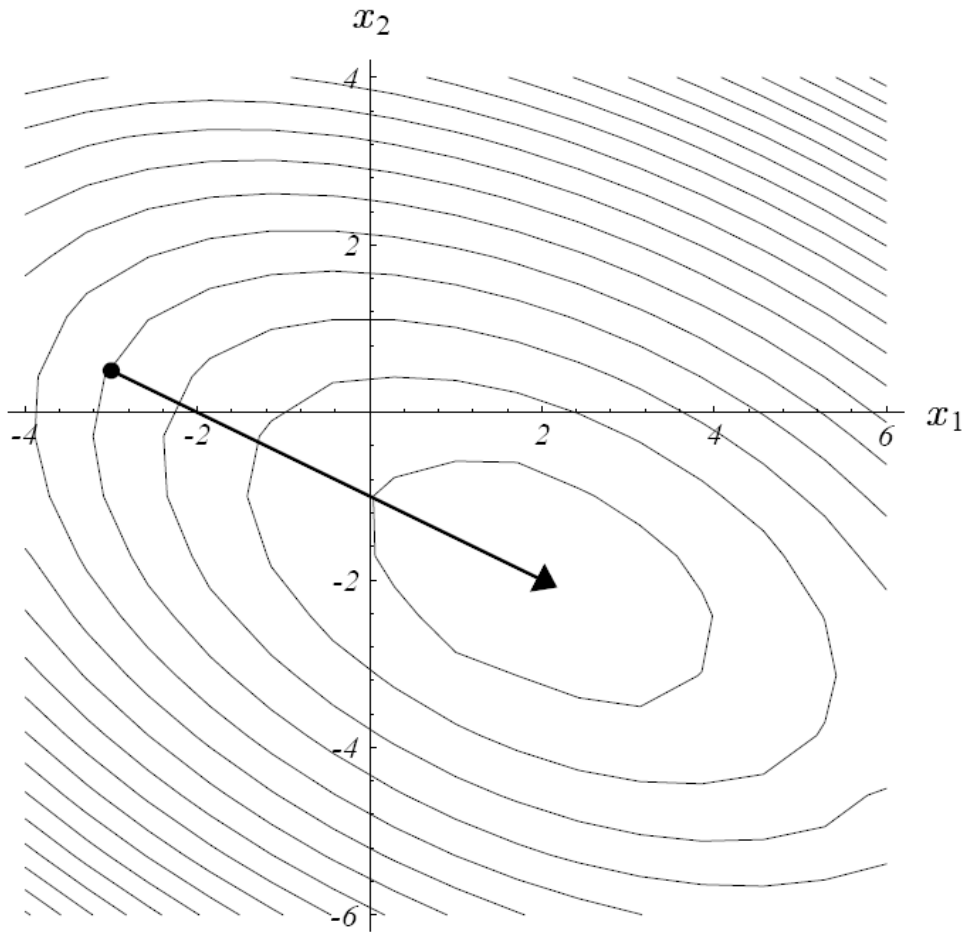
Behavior of gradient descent

- **Zigzag or goes straight depending if we're lucky**
  - Ends up doing multiple steps in the same direction

Unlucky

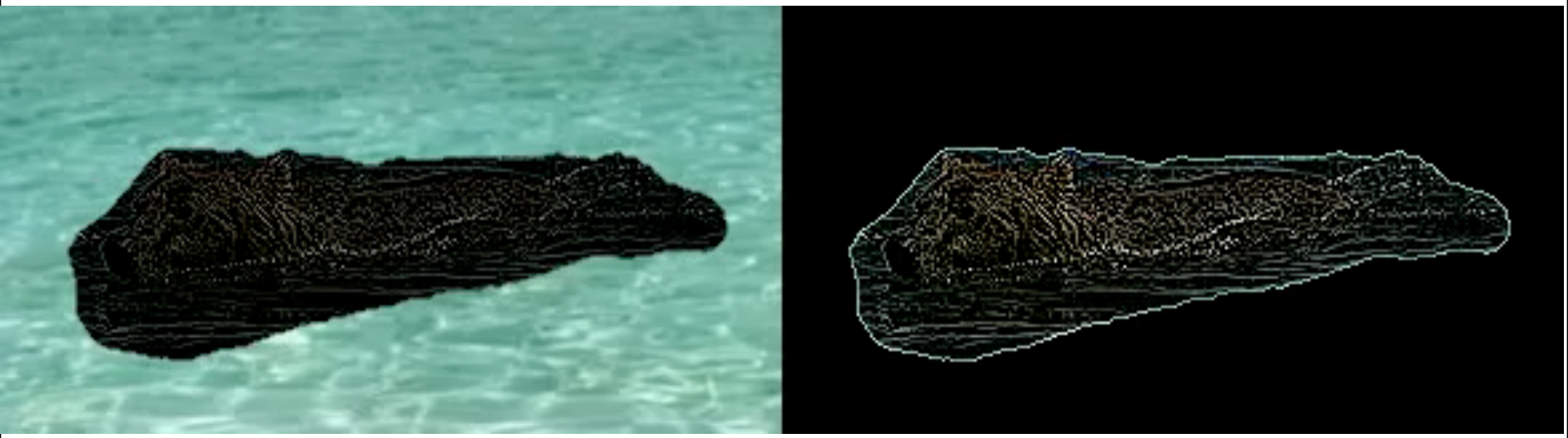


Lucky



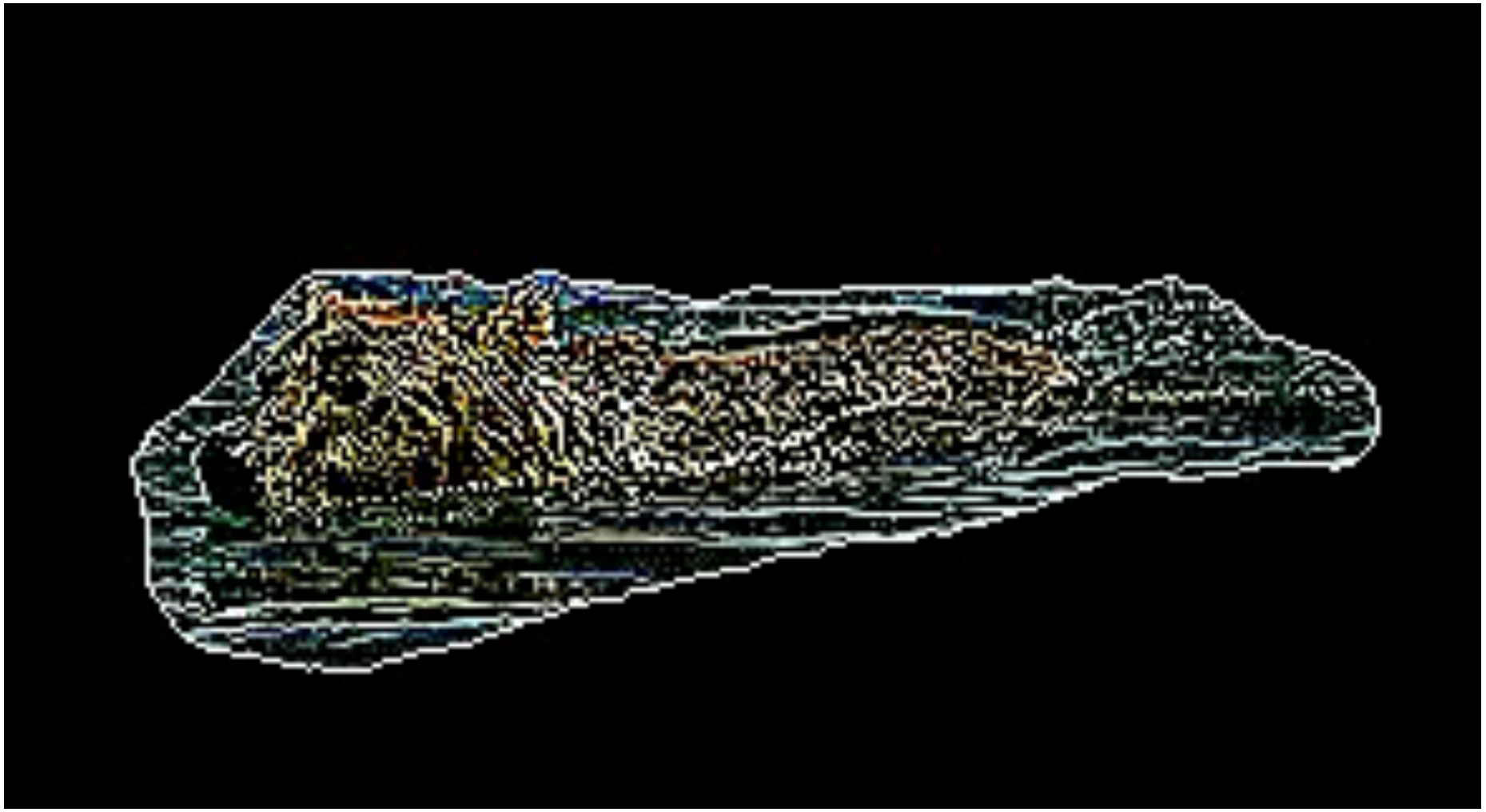
Our residuals

- times 10
- We zigzag between the two same checkerboard patterns



Our residuals

- times 10
- We zigzag between the two same checkerboard patterns

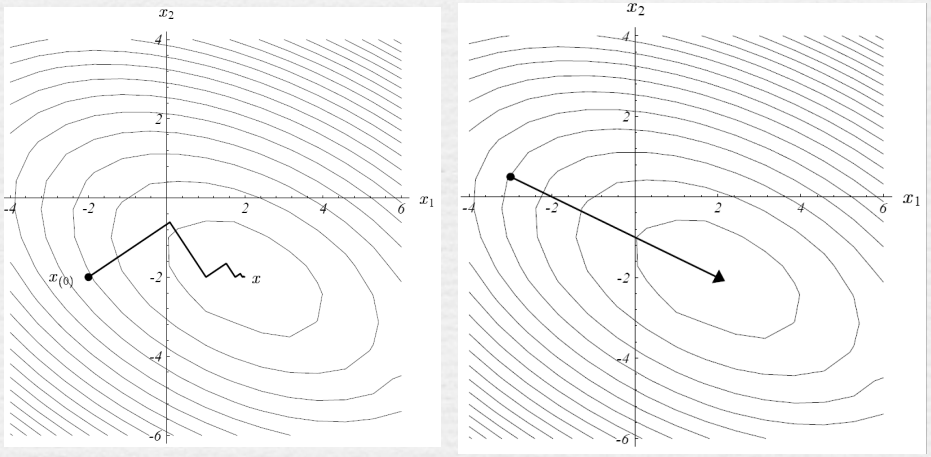


Recap: Gradient Descent

- Residual = - gradient : $r_{(i)}=b-Ax_{(i)}$
- Iteratively walk along residual: $x_{(i+1)}=x_{(i)}+a\ r_{(i)}$
- Find optimal along residual direction:

$$\alpha = \frac{r_{(i)}^T r_{(i)}}{r_{(i)}^T A r_{(i)}}$$

- Behavior: sometimes zigzag, sometimes straight

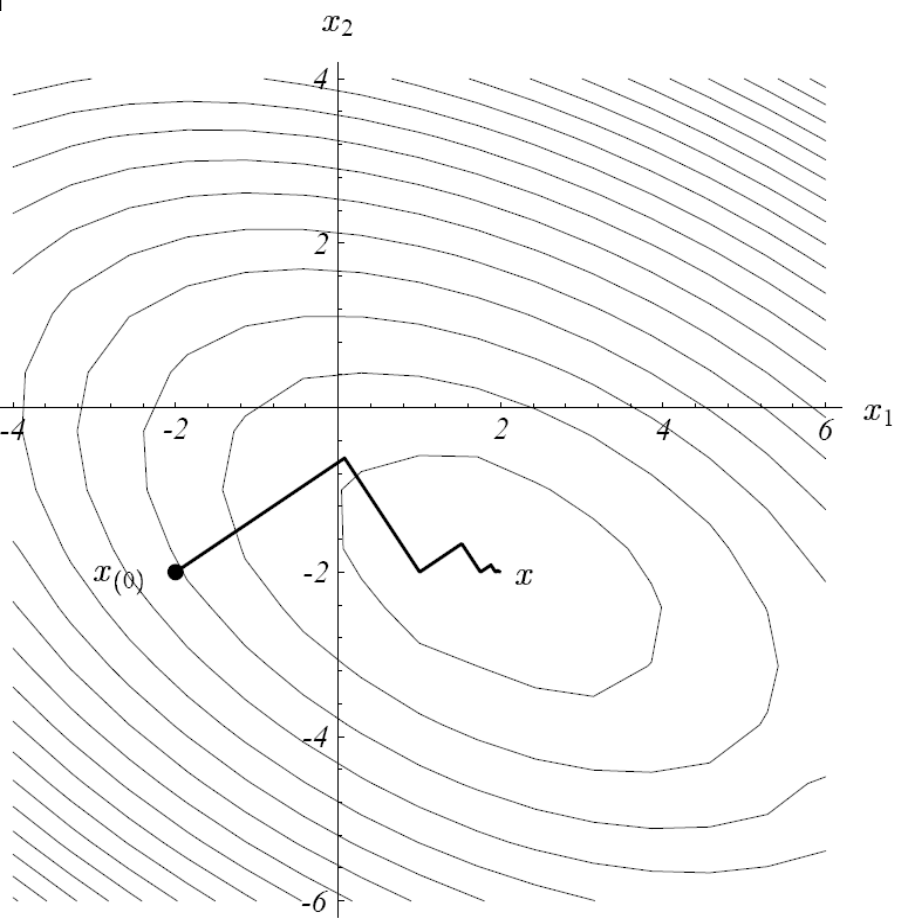


Conjugate Gradient

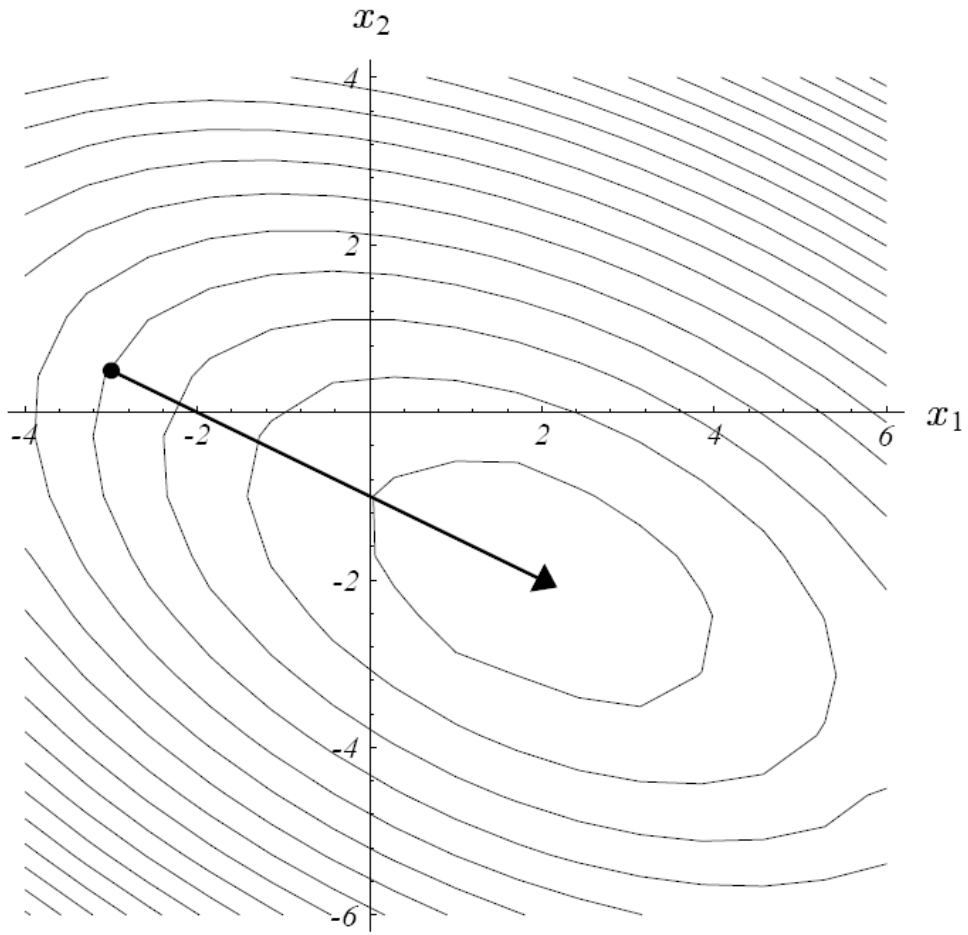
Behavior of gradient descent

- **Zigzag or goes straight depending if we're lucky**
  - –Ends up doing multiple steps in the same direction

Unlucky

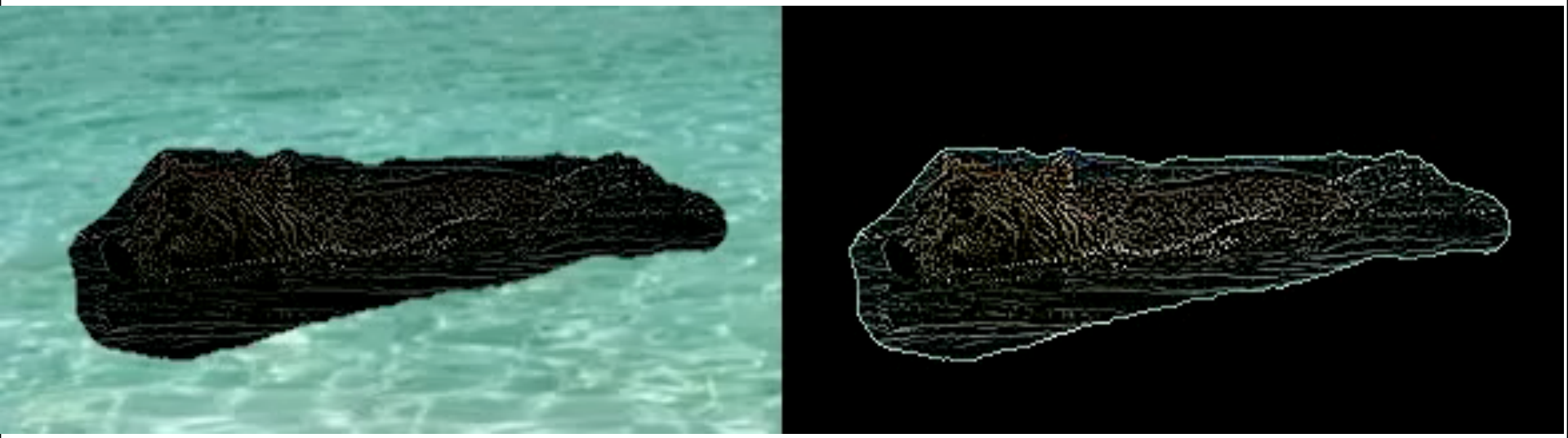


Lucky



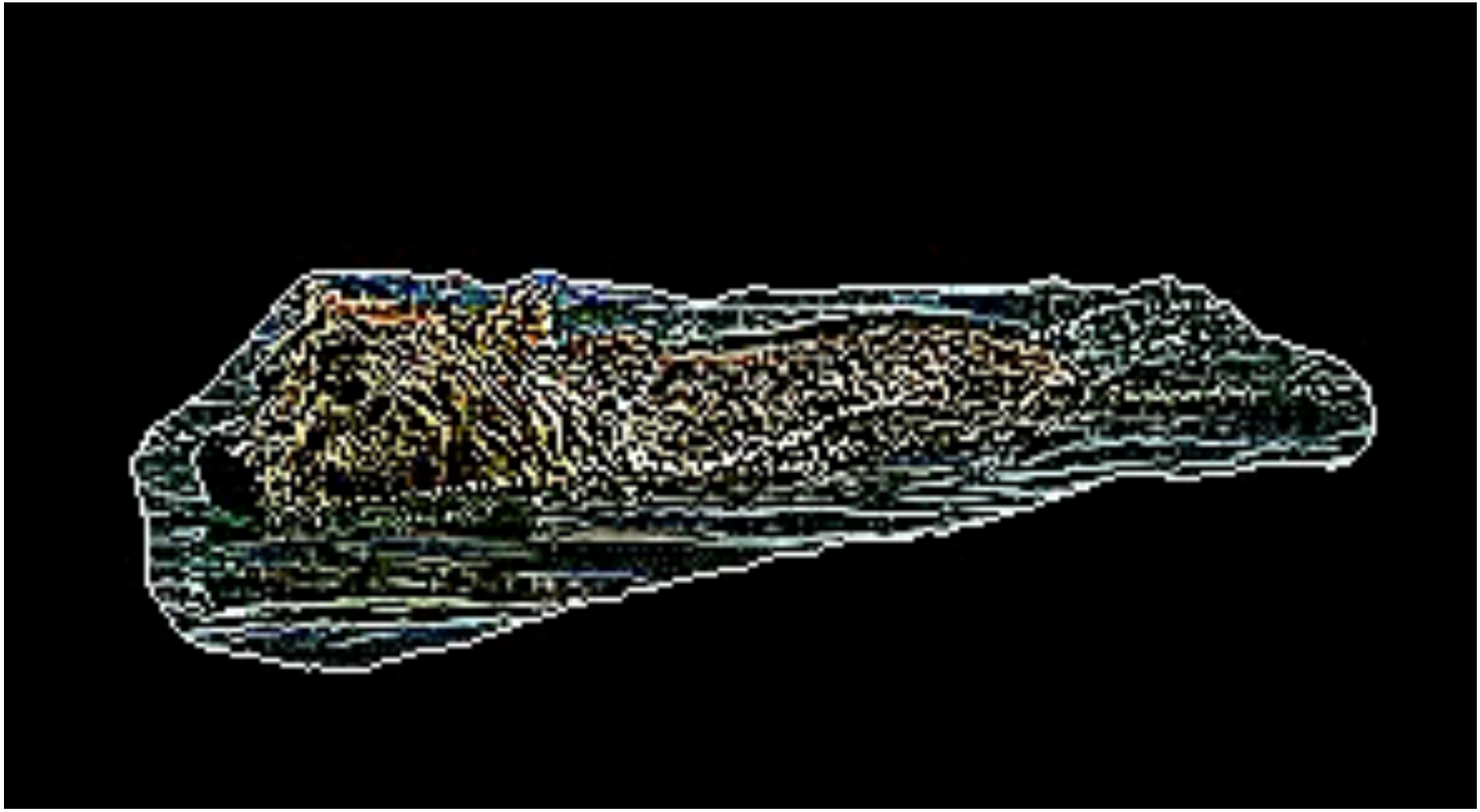
Our residuals

- **times 10**
- **We zigzag between the two same checkerboard patterns**



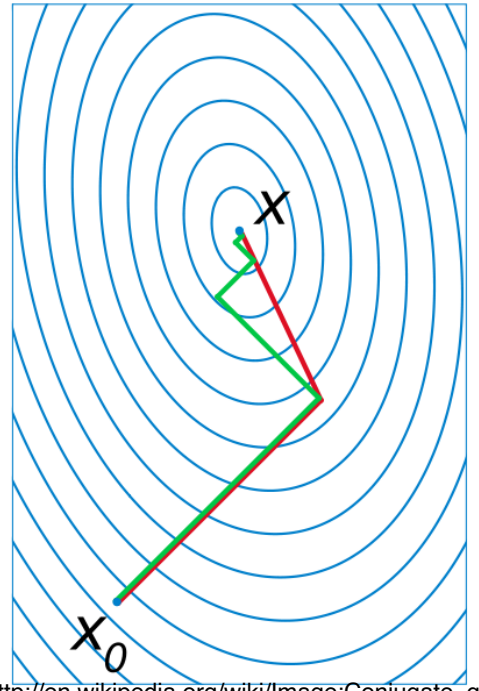
Our residuals

- times 10
- We zigzag between the two same checkerboard



Overview

- **Naive iterative solver: Zigzag**
  - Ends up doing multiple steps in the same direction
- **Conjugate gradient: make sure never go twice in the same direction**
  - Don't go exactly along gradient direction



Green:
standard
iterations
Red:
conjugate
gradient

http://en.wikipedia.org/wiki/Image:Conjugate_gradient_illustration.svg

Good news: the code is simple

```
function [x] = conjgrad(A,b,x0)
    r = b - A*x0;
    w = -r;
    z = A*w;
    a = (r'*w)/(w'*z);
    x = x0 + a*w;
    B = 0;
    for i = 1:size(A);
       r = r - a*z;
       if( norm(r) < 1e-10 )
            break;
       B = (r'*z)/(w'*z);
       w = -r + B*w;
       z = A*w;
       a = (r'*w)/(w'*z);
       x = x + a*w;
```

Conjugate gradient

- **Smarter choice of direction**
  - Ideally, step directions should be orthogonal to one another (no redundancy)
  - But tough to achieve
  - Next best thing: make them A-orthogonal (conjugate) That is, orthogonal when transformed by $\sqrt{A}$
    - Turn the ellipses into circles

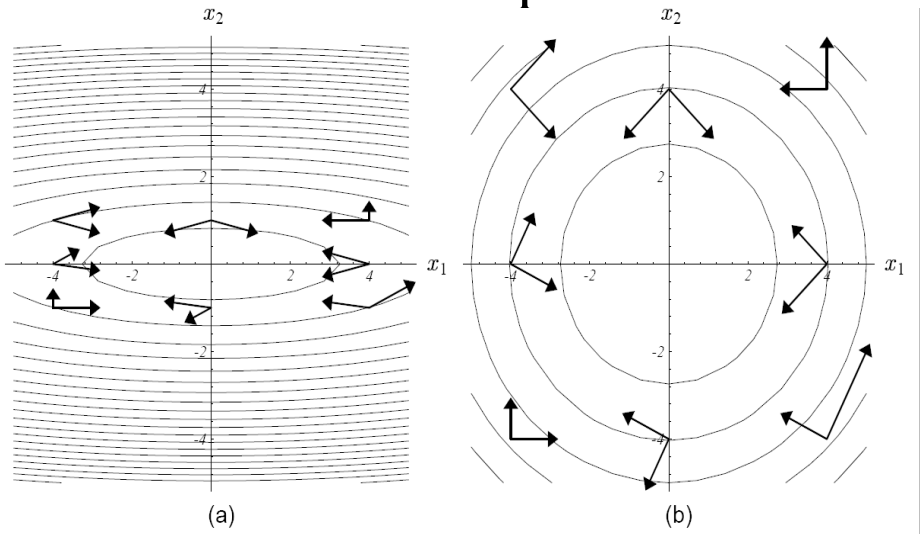


Figure 22: These pairs of vectors are A-orthogonal . . . because these pairs of vectors are orthogonal.

$$d_{(i)}^T A d_{(j)} = 0$$

Conjugate gradient overview

- **For each step:**
  - Take the residual (gradient)
  - Make it A-orthogonal to the previous ones
  - Find minimum along this direction

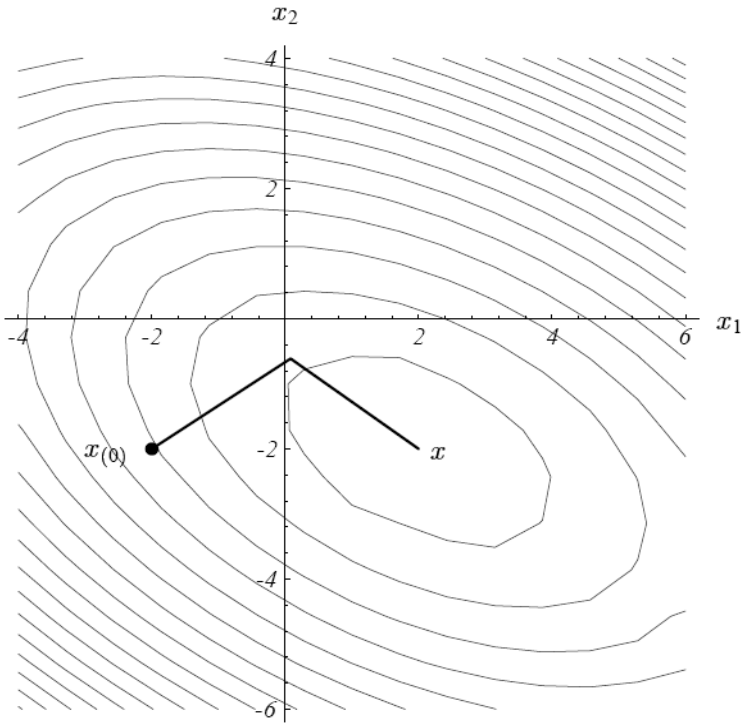
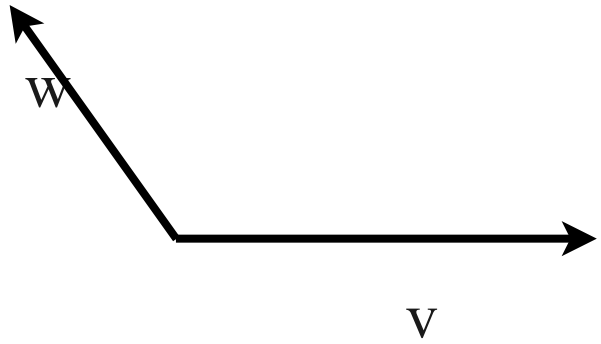


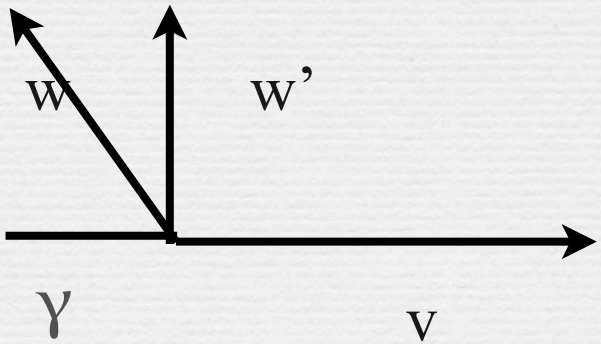
Figure 30: The method of Conjugate Gradients.

How to make vectors orthogonal



How to make vectors orthogonal

- Subtract the non-orthogonal component
  - Use dot product
- Make w orthogonal to v:
- $w' = w - \gamma v$
- where $\gamma = v^T w / v^T v$
  - denominator needed when v is not unit length
- Gram Schmidt generalizes this to n vectors



Making vectors A-orthogonal

- Start with residual r(i+1),
  turn it into d (i+1)
  which is A-orthogonal to all previous iteration directions
  - subtract the component that are non-A-orthogonal to previous directions
  - more complex dot product to take A into account
  - Turns out we only need to do it for previous iteration direction (lucky!)

Making vectors A-orthogonal

- Start with residual r(i+1)
- subtract the non-A-orthogonal component
  - turns out we need to take care of only the previous direction d(i)

$$d_{(i+1)} = r_{(i+1)} + \beta_{(i+1)} d_{(i)}$$

  - where

$$\beta_{(i+1)} = \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}}$$

Similar to previous formula, but involves r's to make orthogonal to d's

  - See Shewchuck's text for derivation

Comparison & recap

- Gradient descent
- r(i)=b-Ax(i)
- x(i+1)=x(i)+ α(i)r(i)

$$\alpha = \frac{r_{(i)}^T r_{(i)}}{r_{(i)}^T A r_{(i)}}$$

- Conjugate gradient
- r(i)=b-Ax(i)
- x(i+1)=x(i)+ α(i) **d**(i)

$$\alpha_{(i)} = \frac{d_{(i)}^T r_{(i)}}{d_{(i)}^T A d_{(i)}}$$

$$\beta_{(i+1)} = \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}}$$

$$d_{(i+1)} = r_{(i+1)} + \beta_{(i+1)} d_{(i)}$$

Saving some computation

- Bottleneck: matrix-vector products
- Can avoid one:
- r(i+1)=b-Ax(i+1)
  =b-A(x(i)+α (i) d(i))
  =(b-Ax(i))+α (i) Ad(i)
  =r(i)+α (i) Ad(i)
- Same as the one needed for α (i)

$$r_{(i)}=b-Ax_{(i)}$$

$$x_{(i+1)}=x_{(i)}+\alpha_{(i)}d_{(i)}$$

$$\alpha_{(i)} = \frac{d_{(i)}^T r_{(i)}}{d_{(i)}^T Ad_{(i)}}$$

$$\beta_{(i+1)} = \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}}$$

$$d_{(i+1)} = r_{(i+1)} + \beta_{(i+1)} d_{(i)}$$

Bells and whistles

- Update r(i) incrementally (previous slide)
  - Compute product Ad once only
  - Pitfall: could drift
  - maybe reset once in a while with full calculation
- Only need to be able to apply matrix A to a vector
  - Often you don't even store A, but use a procedure
- Conjugate gradient is guaranteed to converge in n iterations for n unknowns
  - But we usually want to stop way earlier

The Algorithm

```
function [x] = conjgrad(A,b,x0)
    r = b - A*x0;
    d = -r;
    z = A*d;
    a = (r'*d)/(d'*z);
    x = x0 + a*d;
    B = 0;
    for i = 1:size(A);
       r = r - a*z;
       if( norm(r) < 1e-10 )
            break;
       B = (r'*z)/(d'*z);
       d = -r + B*d;
       z = A*d;
       a = (r'*d)/(d'*z);
       x = x + a*d;
```

x0: initial guess

residual

first iteration: direction = residual

save common term

alpha

update x

beta

guaranteed to converge in size(A) steps

update residual

early termination criterion

beta

make residual A-orthogonal

save common term

alpha

update x

Conjugate gradient

- **For each step:**
  - Take the residual (gradient)
  - Make it A-orthogonal to the previous ones
  - Find minimum along this direction
- **Plus life is good:**
  - In practice, you only need the previous one
  - You can show that the new residual r(i+1) is already A-orthogonal to all previous directions d but d(i)

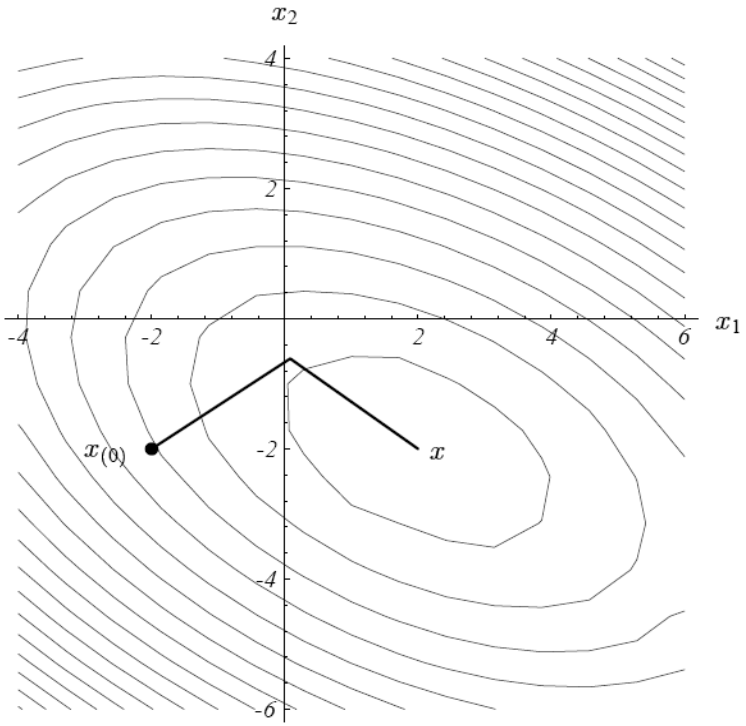
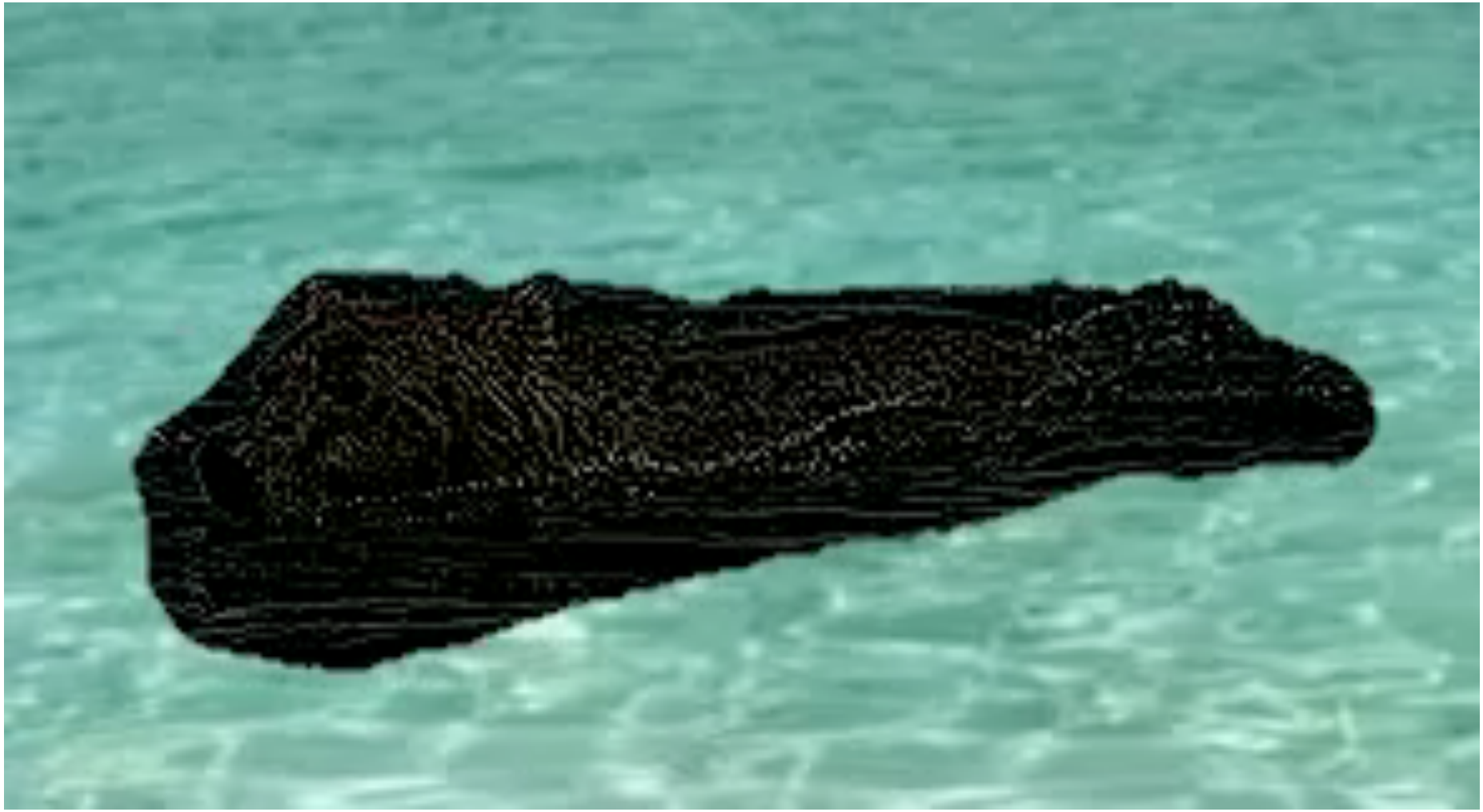


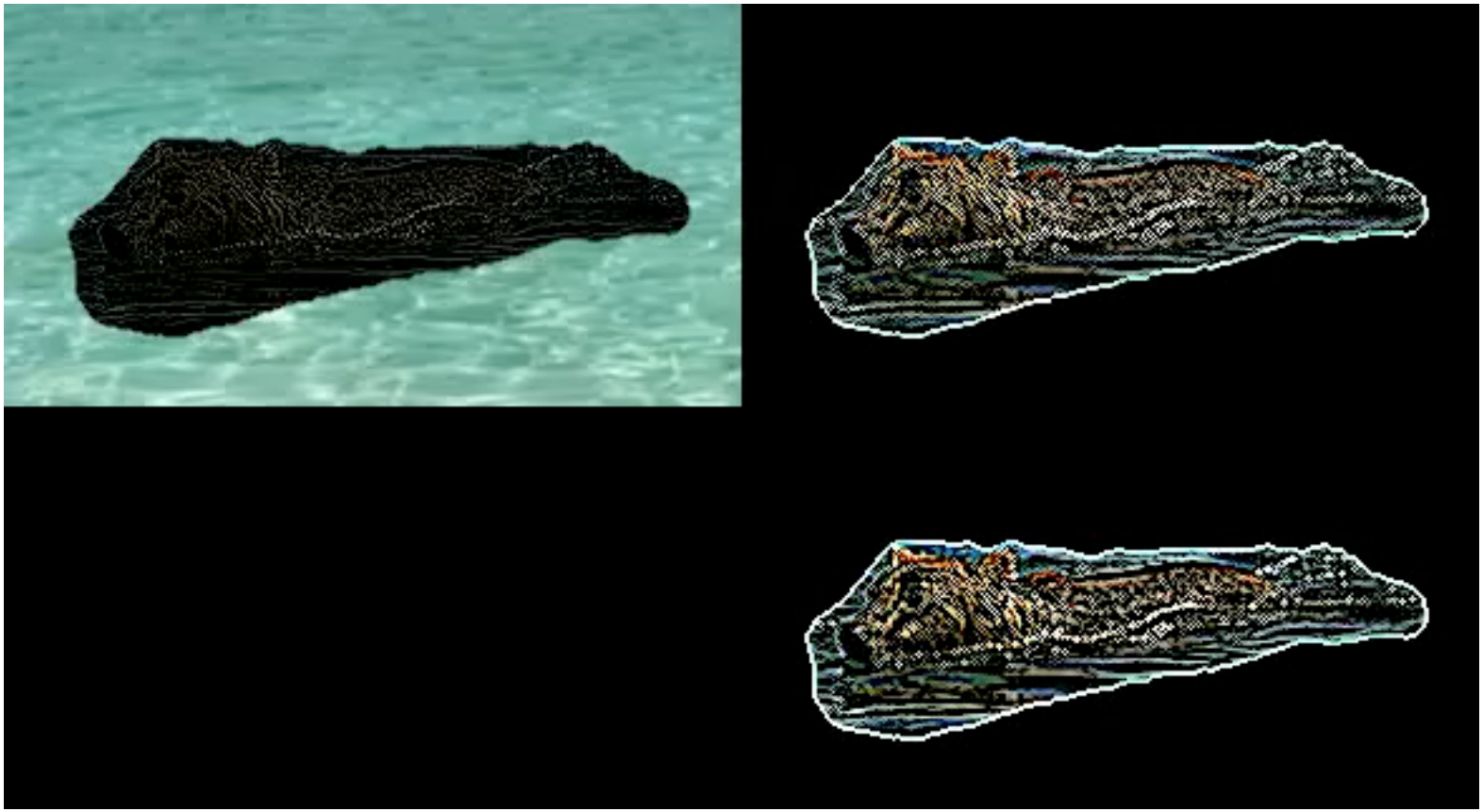
Figure 30: The method of Conjugate Gradients.

Convergence



Residuals and direction

- times 10, displayed at 10fps



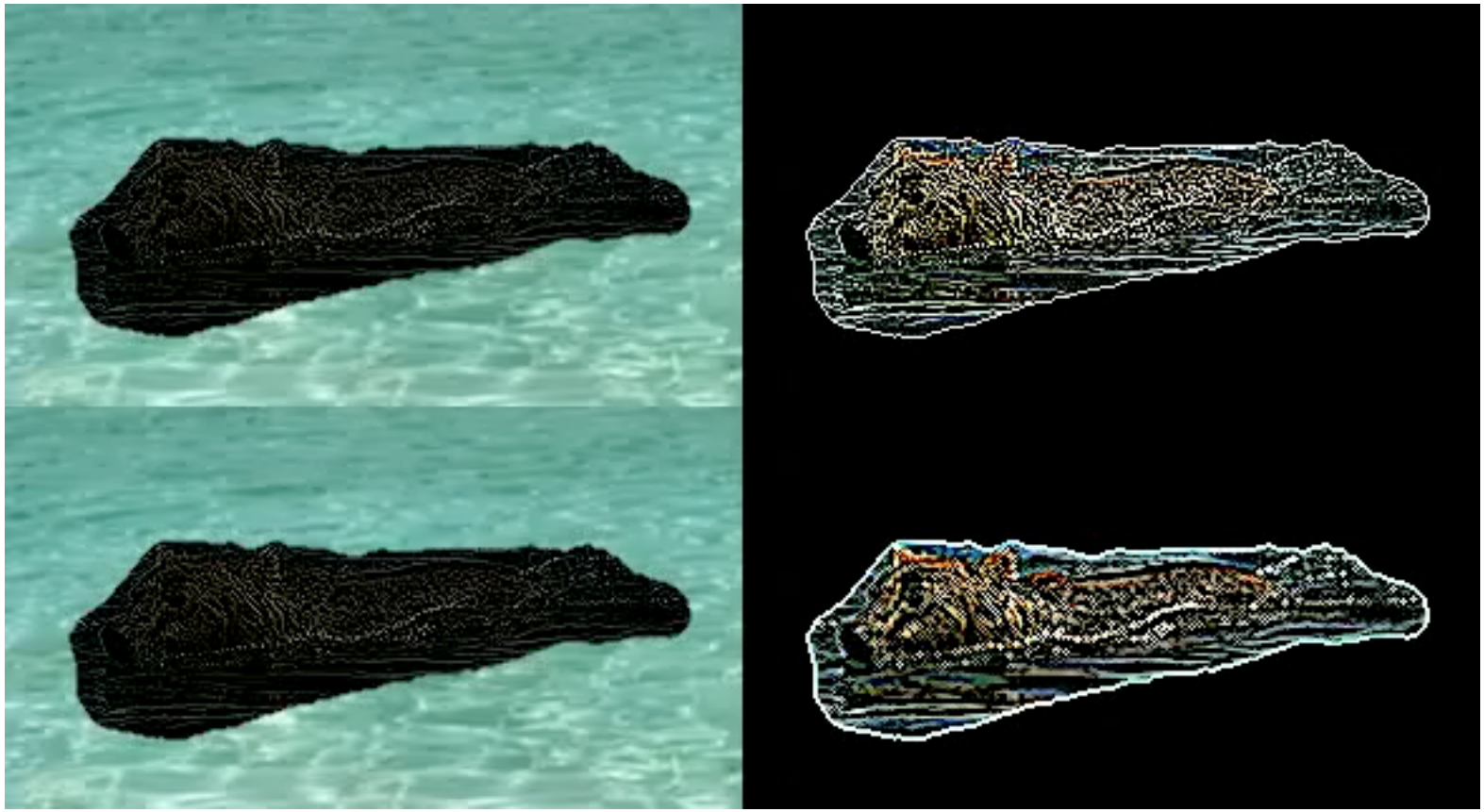
ri

di

Compared to gradient descent

gradient descent

conjugate gradient



When use Conjugate Gradient?

- Ax=b
- A is positive definite
- A is sparse
- Disadvantage compared to factorization+backsubstitution:
  - you start from scratch for every new b
  - error if not converged
  - A has to be positive definite
  - costly if not sparse
- Bottomline: use a direct solver when you can afford it, conjugate gradient otherwise

The two references

An Introduction to
the Conjugate Gradient Method
Without the Agonizing Pain
Edition $1\frac{1}{4}$

Jonathan Richard Shewchuk
August 4, 1994

- http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf



- Iterative methods for sparse linear systems (2nd edition)
  Yousef Saad
- http://www-users.cs.umn.edu/~saad/books.html
- http://books.google.com/books?id=Uoe7xBOhS5AC&dq=saad+iterative&printsec=frontcover&source=in&hl=en&ei=Y2GtSerjMdW5twft78CHBg&sa=X&oi=book_result&resnum=11&ct=result#PPR5,M1

Regularization

Idea

- Add an energy term to the quadratic
- Encourage the output not to go crazy
- Penalize crazy high gradients

$$\min \quad \|M\hat{x} - y\|^2 + \text{craziness}(x)^2$$

Putting is together

- Deconvolution least squares
- Laplace least squares (Poisson with zero gradient)
- => regularized deconvolution

Putting is together

- Deconvolution least squares

$$\min \sum \|Mx - y\|^2$$

- Laplace least squares (Poisson with zero gradient)

$$\min \sum \|Gx\|^2$$

- => regularized deconvolution

$$\min \sum \|Mx - y\|^2 + \lambda \|Gx\|^2$$

$$M^T M x + \lambda G^T G x - M^T y = 0$$

In practice

- Combine deblurring & Poisson/Laplace
- A:
  - known blur for deblurring part
  - Laplacian for regularization
- b:
  - input blurry image for deblurring
  - 0 for regularization
- Weighted average of the 2 As and the 2 bs
  - weight controls regularization strength/SNR
    - ideally derived from Wiener equation
    - In practice often tweaked

Results

- Gaussian blur, sigma=1



Blurry input



Blurry input (for comparison)



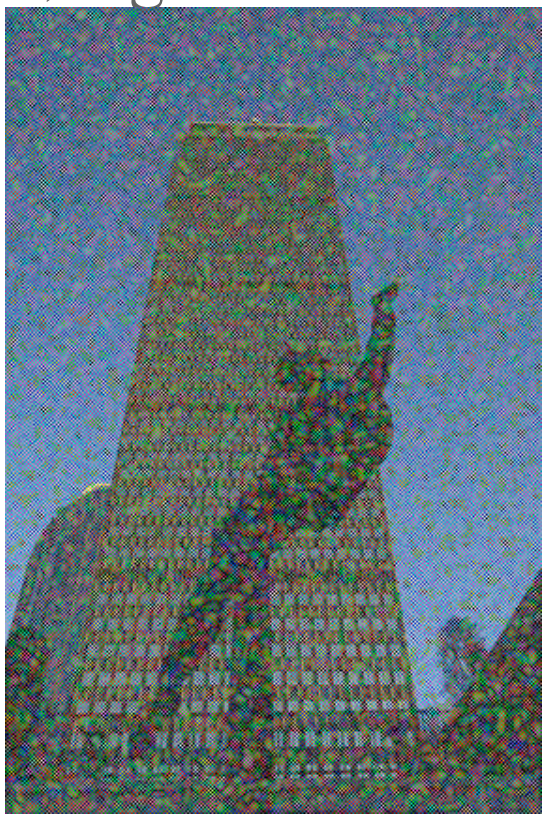
Blurry input (for comparison)

Results

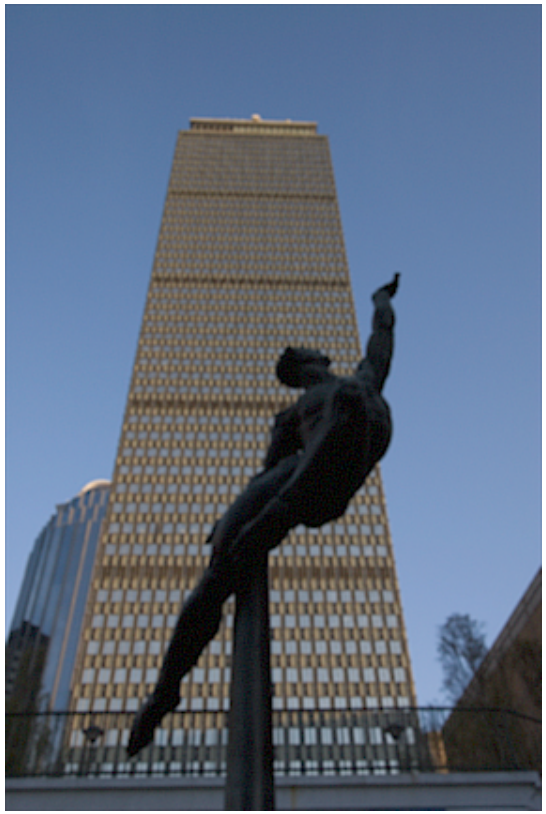
- Gaussian blur, sigma=1



Blurry input



Result without regularization



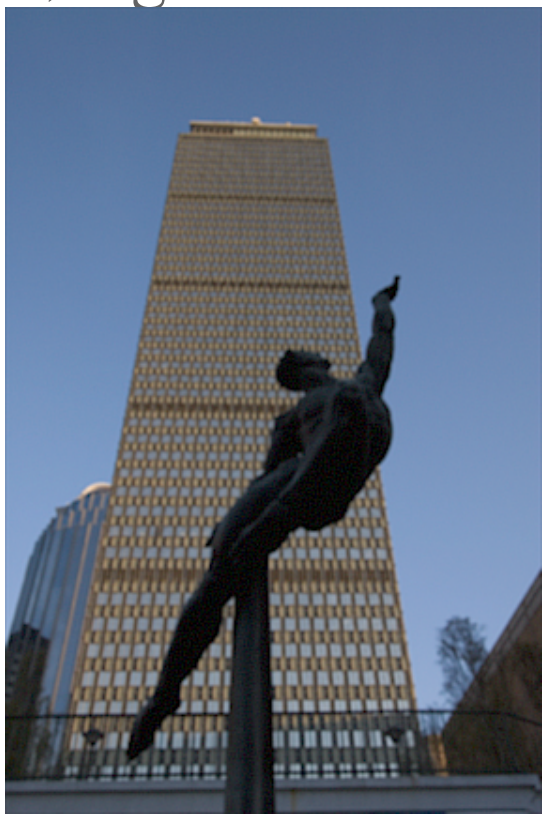
Result with regularization

Results

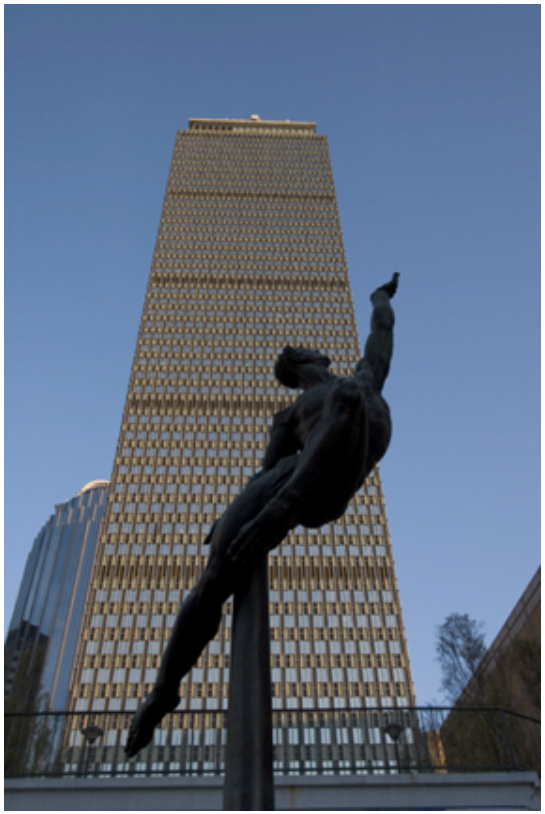
- Gaussian blur, sigma=1



Blurry input



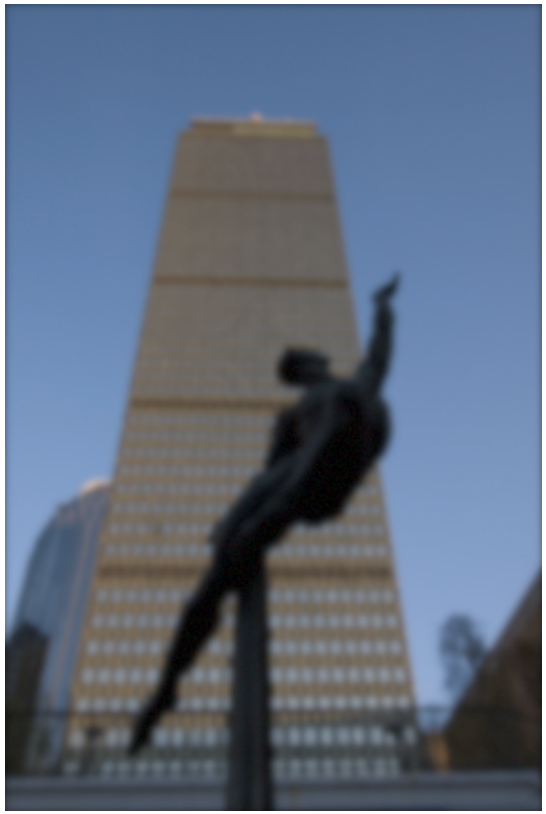
Result with regularization



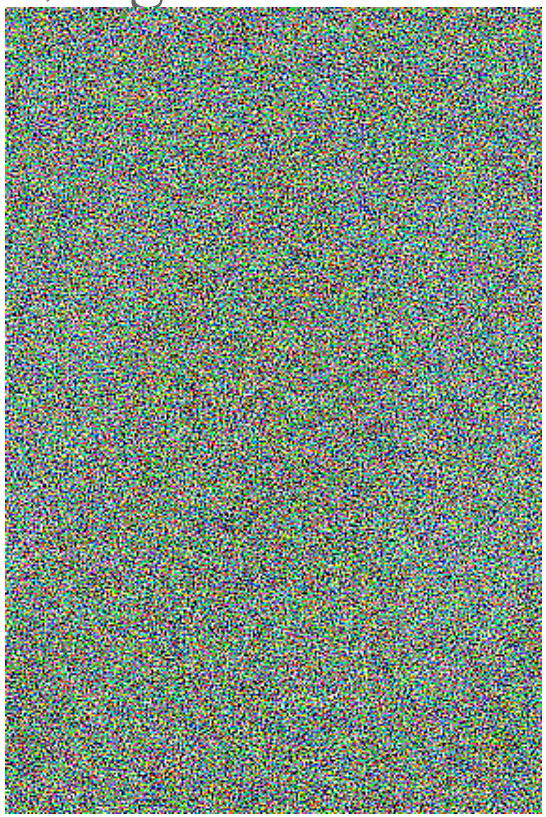
Original sharp image

Results

- Gaussian blur, sigma=2



Blurry input



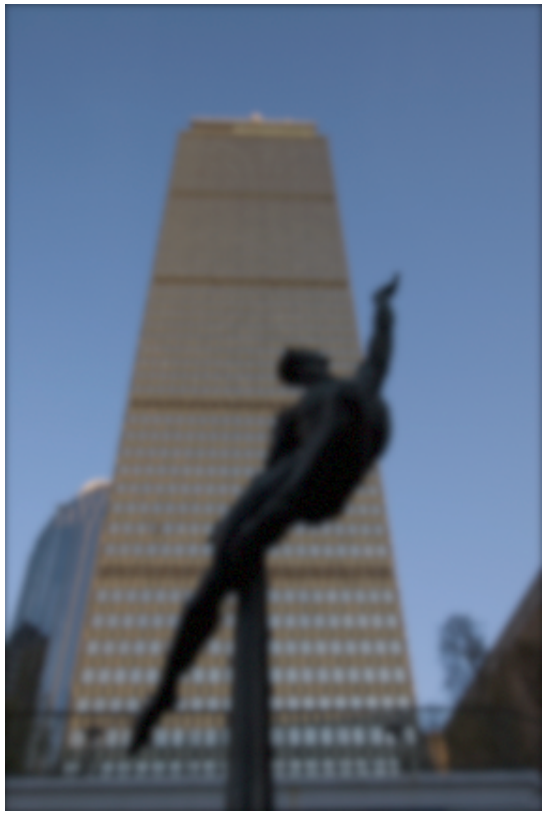
Result without regularization



Result with regularization

Results

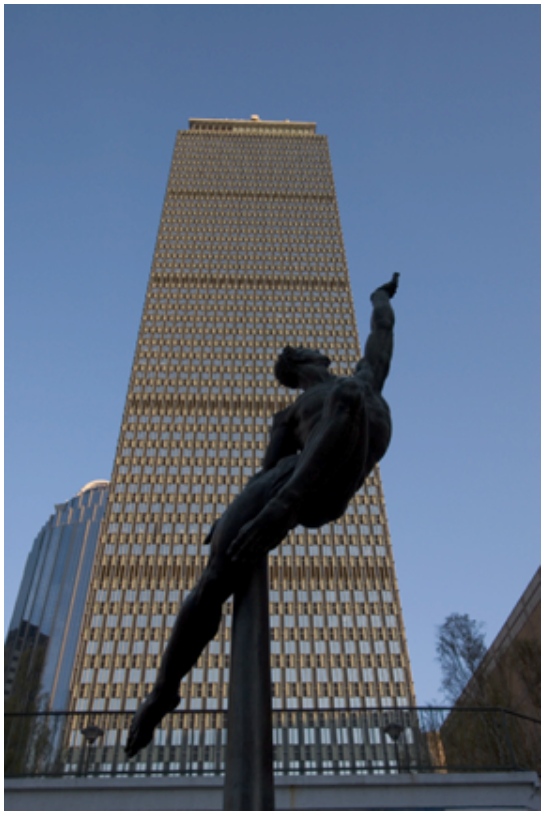
- Gaussian blur, sigma=2



Blurry input



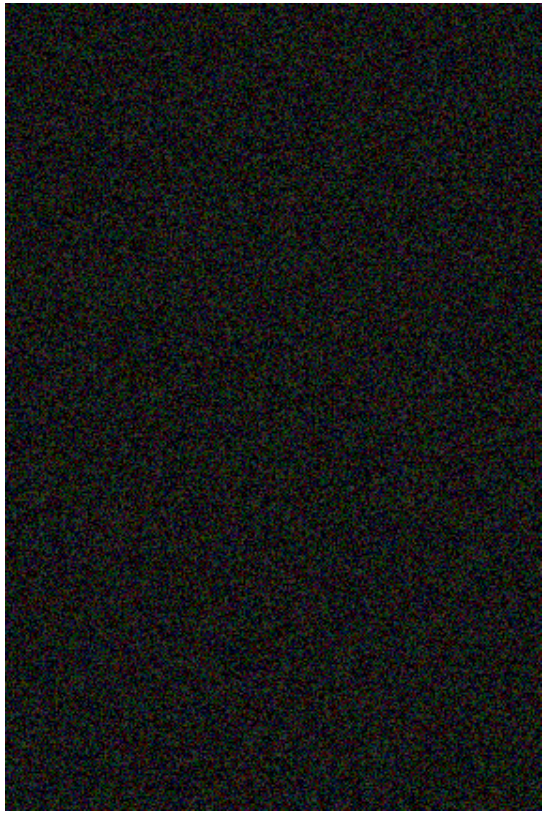
Result with regularization



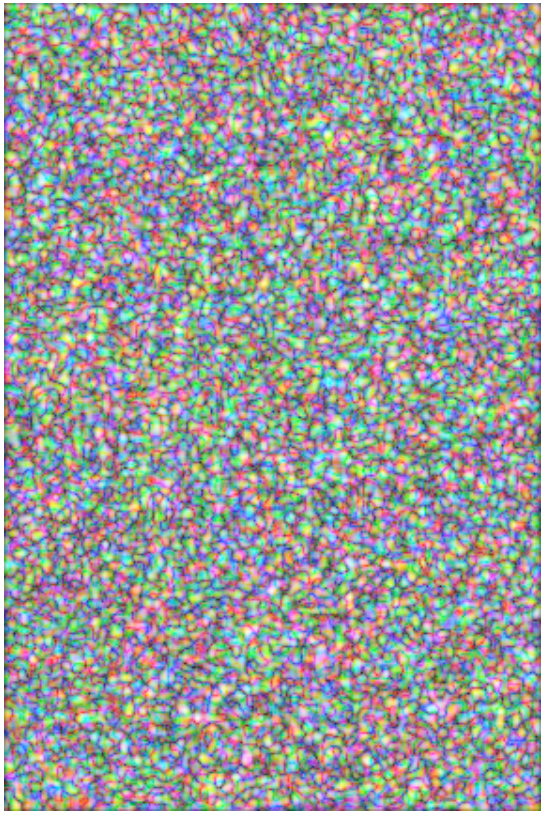
Original sharp image

Noise alone

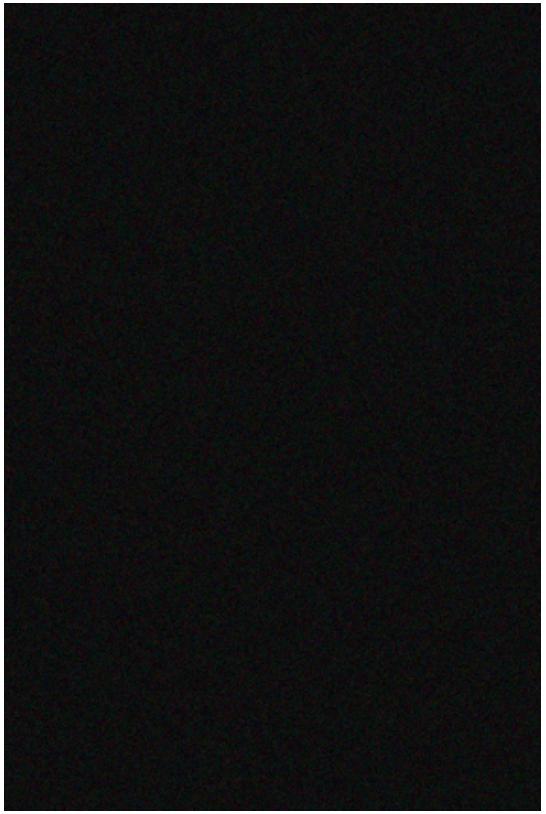
- Gaussian blur, sigma=1



Noise*20



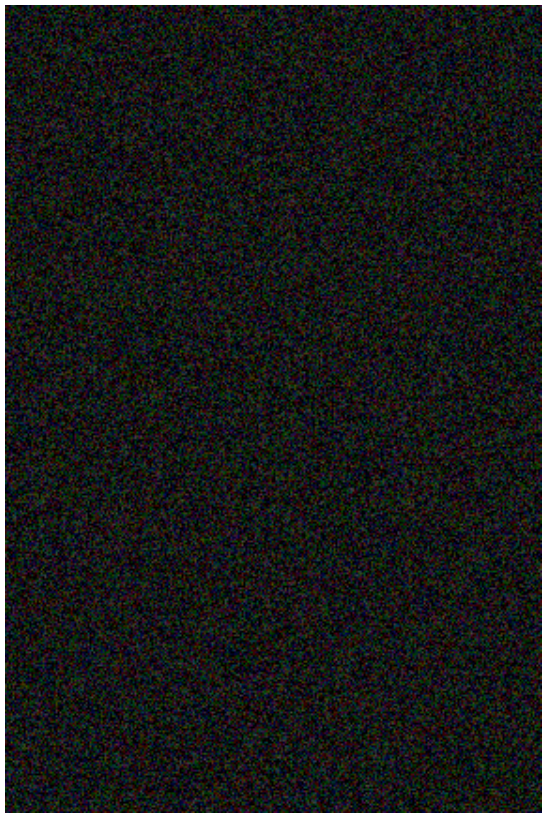
Deconvolved noise



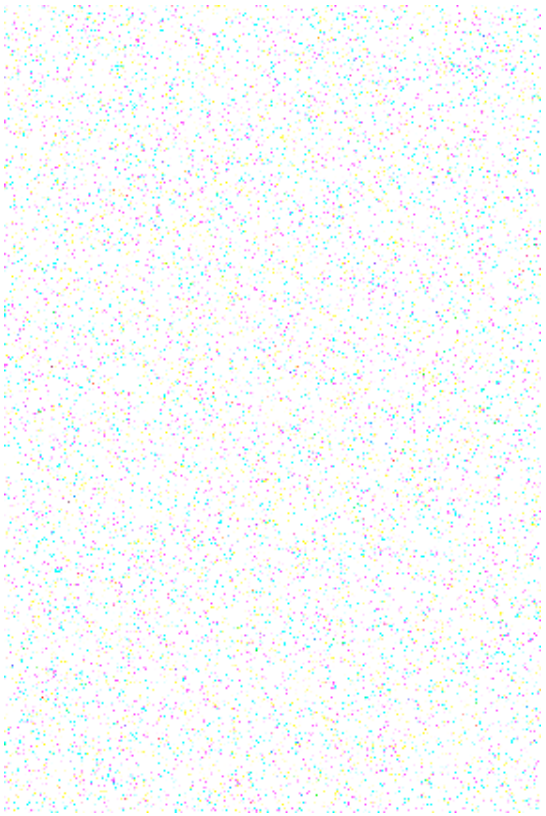
Deconvolved noise
with regularization

Noise alone

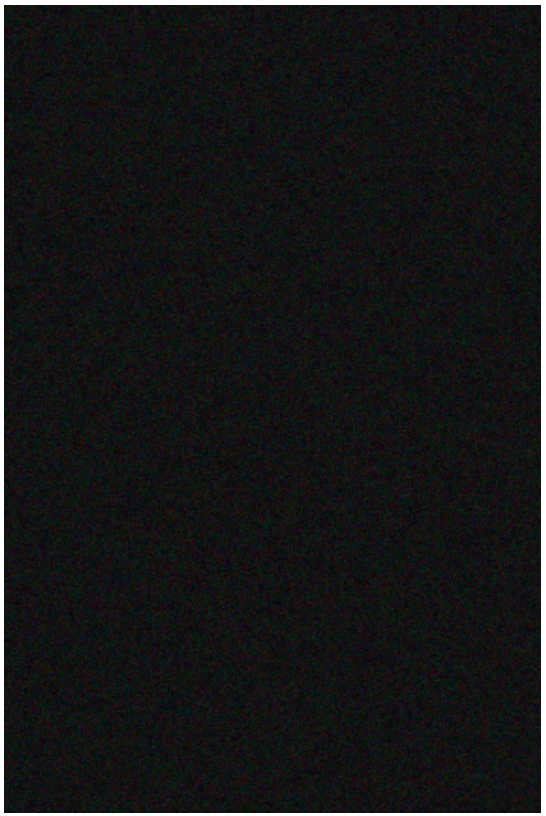
- Gaussian blur, sigma=2



Noise*20



Deconvolved noise



Deconvolved noise
with regularization

Very general idea

- Unstable linear system
- Add regularization to stabilize
  - choice of regularizer depends on problem
    - sometimes minimize $\|x\|^2$
    - sometimes $\|\nabla_x\|^2$
    - …
- Can be seen as a prior with Bayesian probabilities

Can we do better?



Input

$$\sum ||\nabla x||^2$$



Deconvolved with linear prior on gradient

http://groups.csail.mit.edu/grap

Sparsity prior for deconvolution



Input

Image and Depth from a Conventional Camera
with a Coded Aperture
Anat Levin, Rob Fergus, Fredo Durand, Bill Freeman

http://groups.csail.mit.edu/graphics/CodedAperture/

But harder because non-linear

$$\sum ||\nabla x||^2$$

spread error

$$\sum ||\nabla x||^{0.8}$$

localize error



Deconvolved with linear prior on gradient



Our sparse prior

Sparsity prior for deconvolution

- Can be solved using reweighed least squares
- Once in a while, change the regularization matrix to approximate $|| ||^{0.8}$ with a quadratic

- http://en.wikipedia.org/wiki/Iteratively_reweighted_least_squares

Sparse prior in practice

- One solution:
  Iteratively reweighted least square
  - http://en.wikipedia.org/wiki/Iteratively_reweighted_least_squares
  - Outer loop over a least square problem
    - Conjugate gradient is now the inner loop
  - Reweight the gradient regularizer as a function of the current gradient strength
    - Bigger gradients get lower weight
- Matlab code at
  - http://groups.csail.mit.edu/graphics/CodedAperture/DeconvolutionCode.html
- More at
  - http://www.wisdom.weizmann.ac.il/~nadler/Course/deconv_notes.pdf

Analyzing (de)convolution: Fourier !

Analyzing blur

- Bunch of high-dimensional linear equations

$$\hat{x} \approx M^{-1} y = M^{-1}(Mx + n)$$

Analyzing blur

- Bunch of high-dimensional linear equations

$$\hat{x} \approx M^{-1} y = M^{-1}(Mx + n)$$

- What are the properties of M^{-1}?
  What does it do to n? to y?
  How well can we deconvolve?
  How does the kernel affect deconvolution?
  Why was gradient regularization a good idea?
- Hard to answer with the general form of M
- But easy if M is diagonal

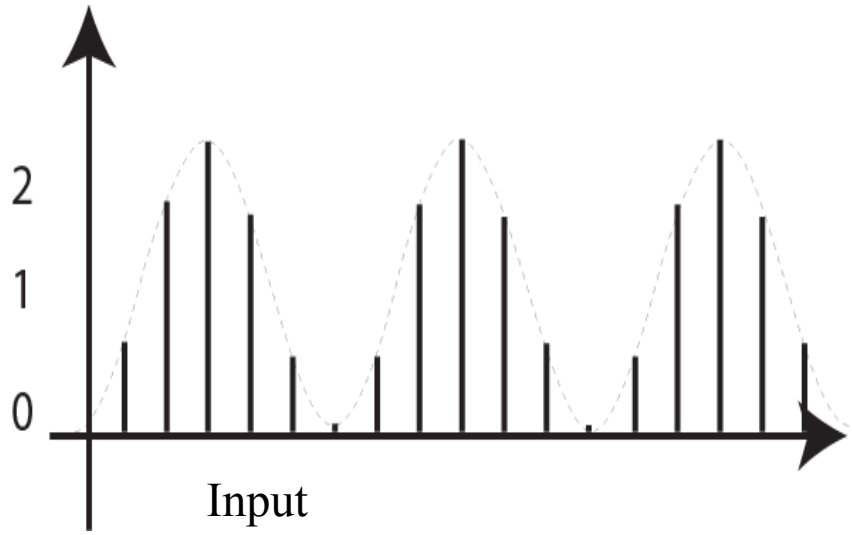
Inverse of a diagonal matrix

$$M = \begin{pmatrix} d_1 & & & \\ & d_2 & & \\ & & d_3 & \\ & & & d_4 \\ & & & & \ddots \end{pmatrix} \qquad M^{-1} = \begin{pmatrix} 1/d_1 & & & \\ & 1/d_2 & & \\ & & 1/d_3 & \\ & & & 1/d_4 \\ & & & & \ddots \end{pmatrix}$$

How do we make convolution diagonal?

- Eigenvectors!

Blurring: convolution

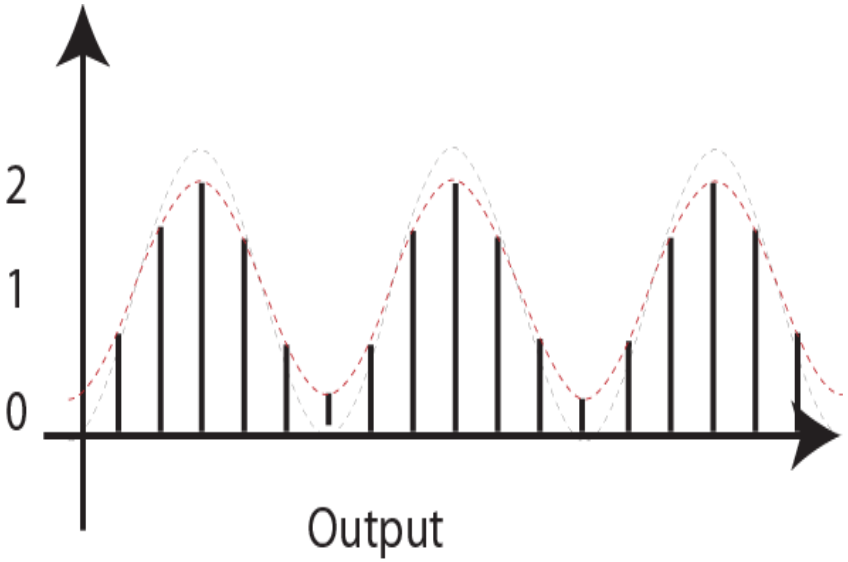




Convolution sign



Kernel

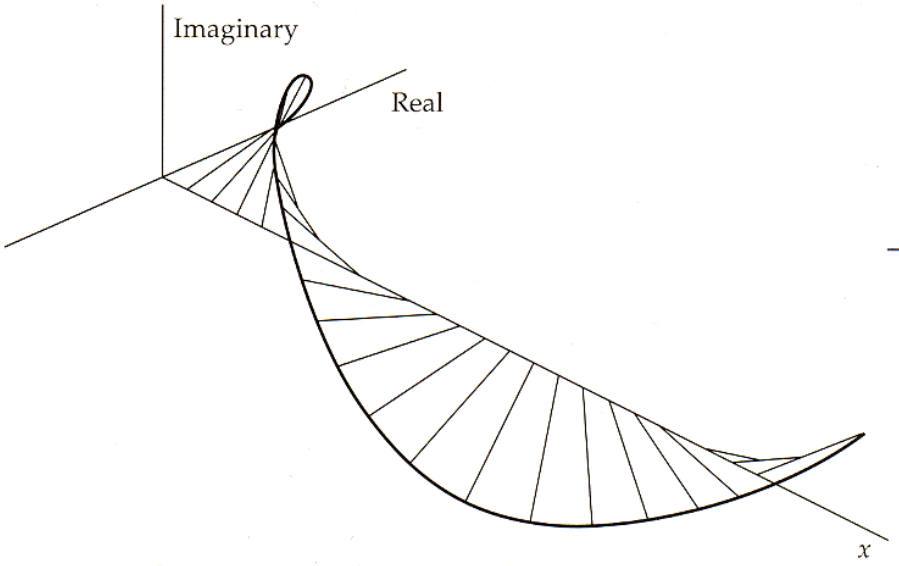


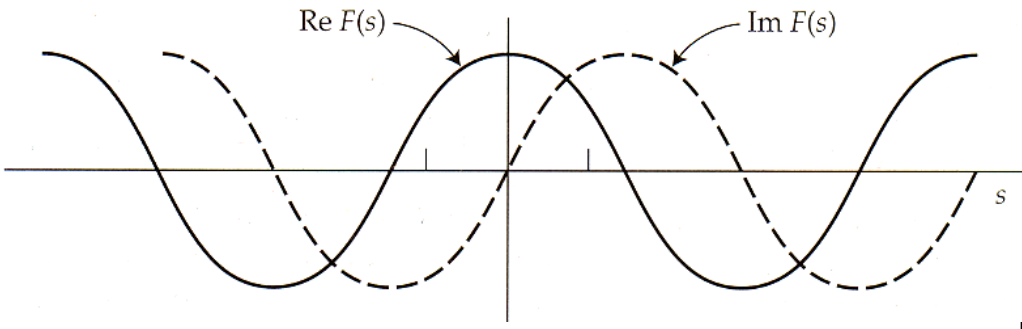
Same shape, just reduced contrast!!!

This is an eigenvector
(output is the input multiplied by a constant)

Eigenvectors

- More precisely, the eigenvectors are $e^{i\omega x}$
- aka $\cos \omega x + i \sin \omega x$





Big Motivation for Fourier analysis

- (Complex) Sine waves are eigenvectors of the convolution operator
- Convolution is diagonal in the Fourier domain

Big Motivation for Fourier analysis

- Sine waves are eigenvectors of the convolution operator
- Convolution is diagonal in the Fourier domain
  - And the diagonal is the Fourier transform of the kernel
  - Convolution theorem:

> Convolution in the primal
> is a multiplication in Fourier

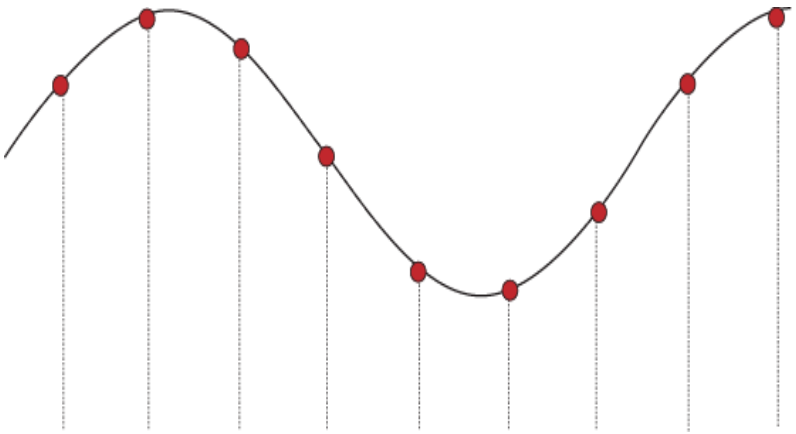
More Fourier Motivation

Other motivation for Fourier analysis: sampling

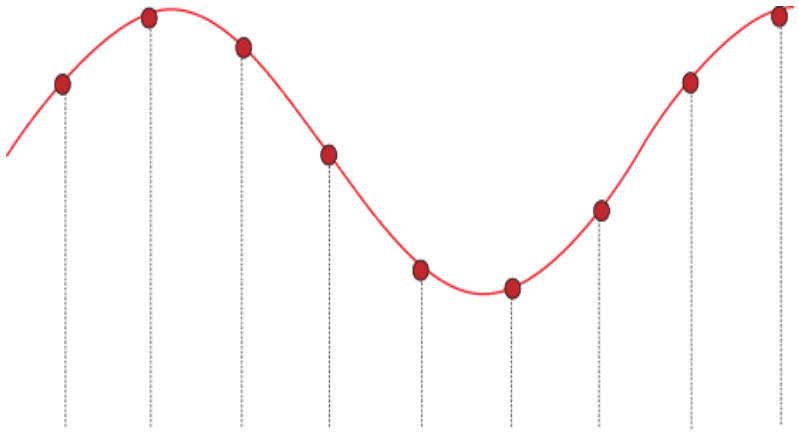
- The sampling grid is a periodic structure
  - Fourier is pretty good at handling that
  - We saw that a sine wave has serious problems with sampling
- Sampling is a linear process
  - but not shift-invariant

Sampling Density

- If we're lucky, sampling density is enough



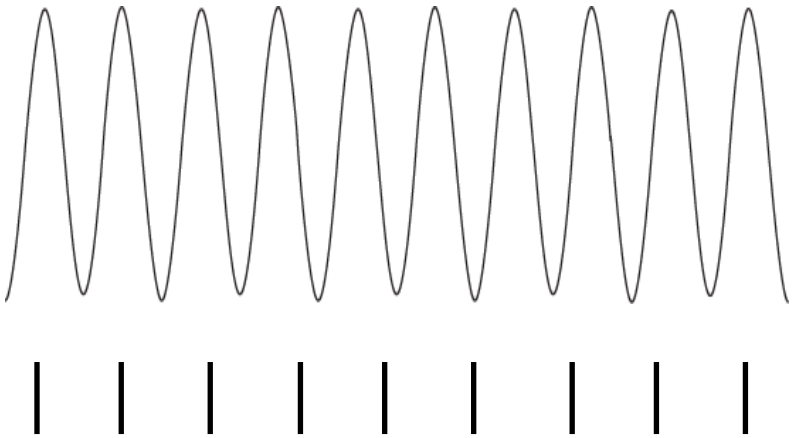
Input



Reconstructed

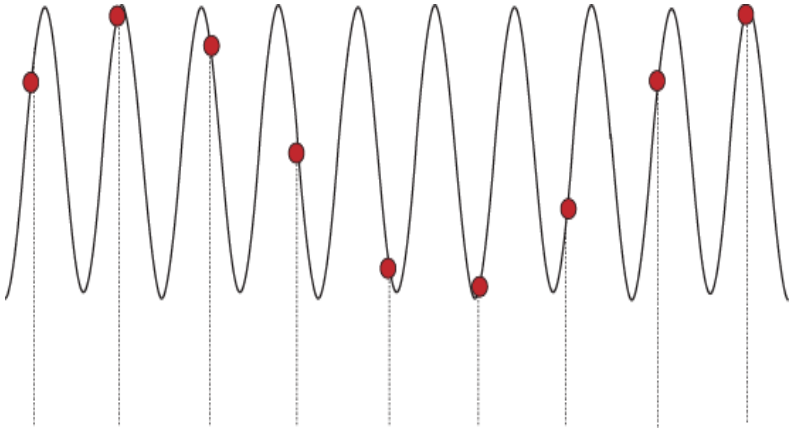
Sampling Density

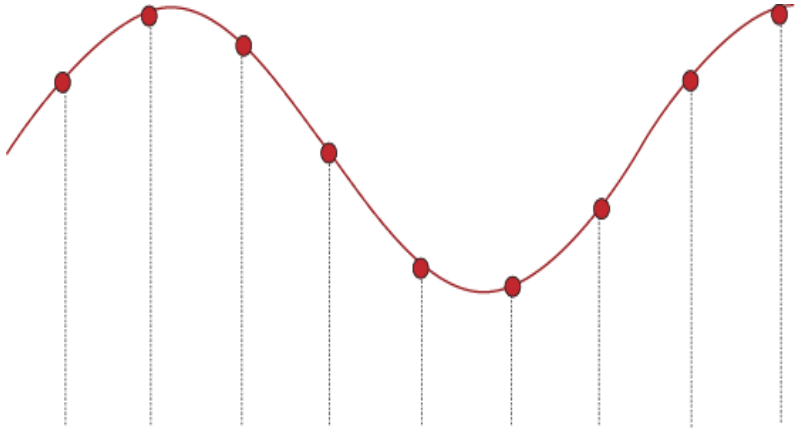
- If we insufficiently sample the signal



Sampling Density

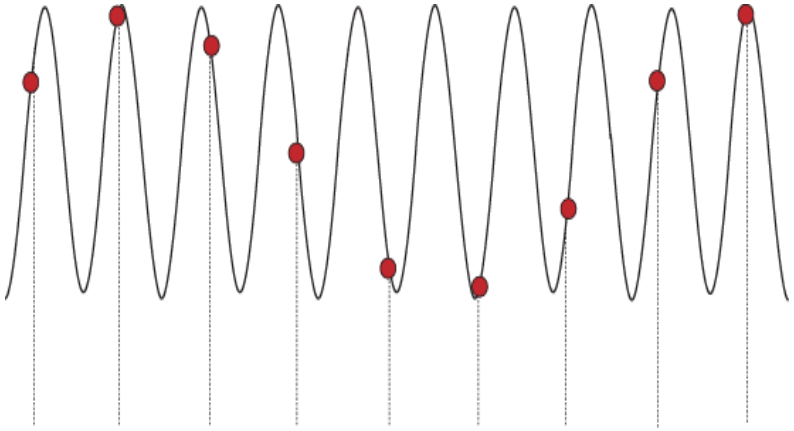
- If we insufficiently sample the signal, it may be mistaken for something simpler during reconstruction (that's aliasing!)

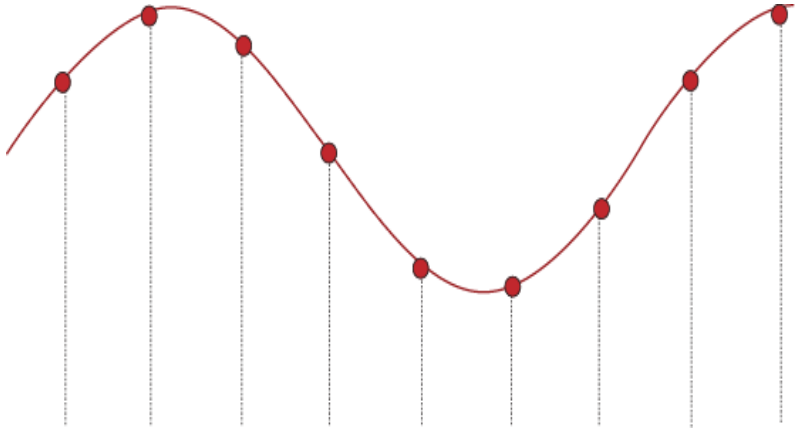




Sampling recap

- Sampling has bad properties when undersampling (aliasing)
- But they can be analyzed more easily for sine waves





Recap: motivation for sine waves

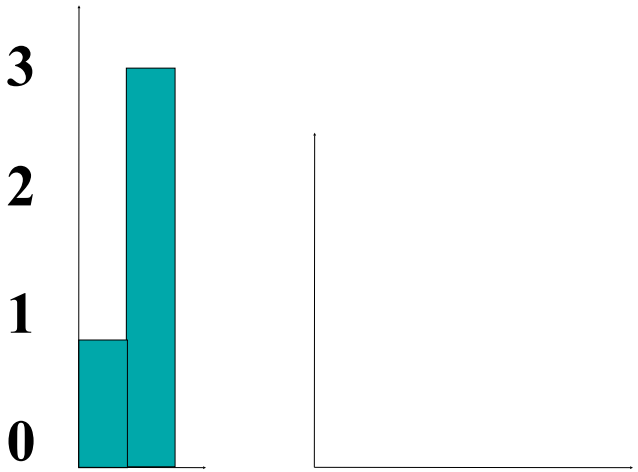
- Blurring sine waves is simple
  - You get the same sine wave, just scaled down
  - The sine functions are the eigenvectors of the convolution operator
- Sampling sine waves is interesting
  - Get another sine wave
  - Not necessarily the same one! (aliasing)

If we represent functions (or images) with a sum of sine waves, convolution and sampling are easy to study

Fourier as change of basis

- Shuffle the data to reveal other information
- E.g., take average & difference: matrix

$$\begin{bmatrix} 0.5 & 1 \\ 0.5 & -1 \end{bmatrix}$$



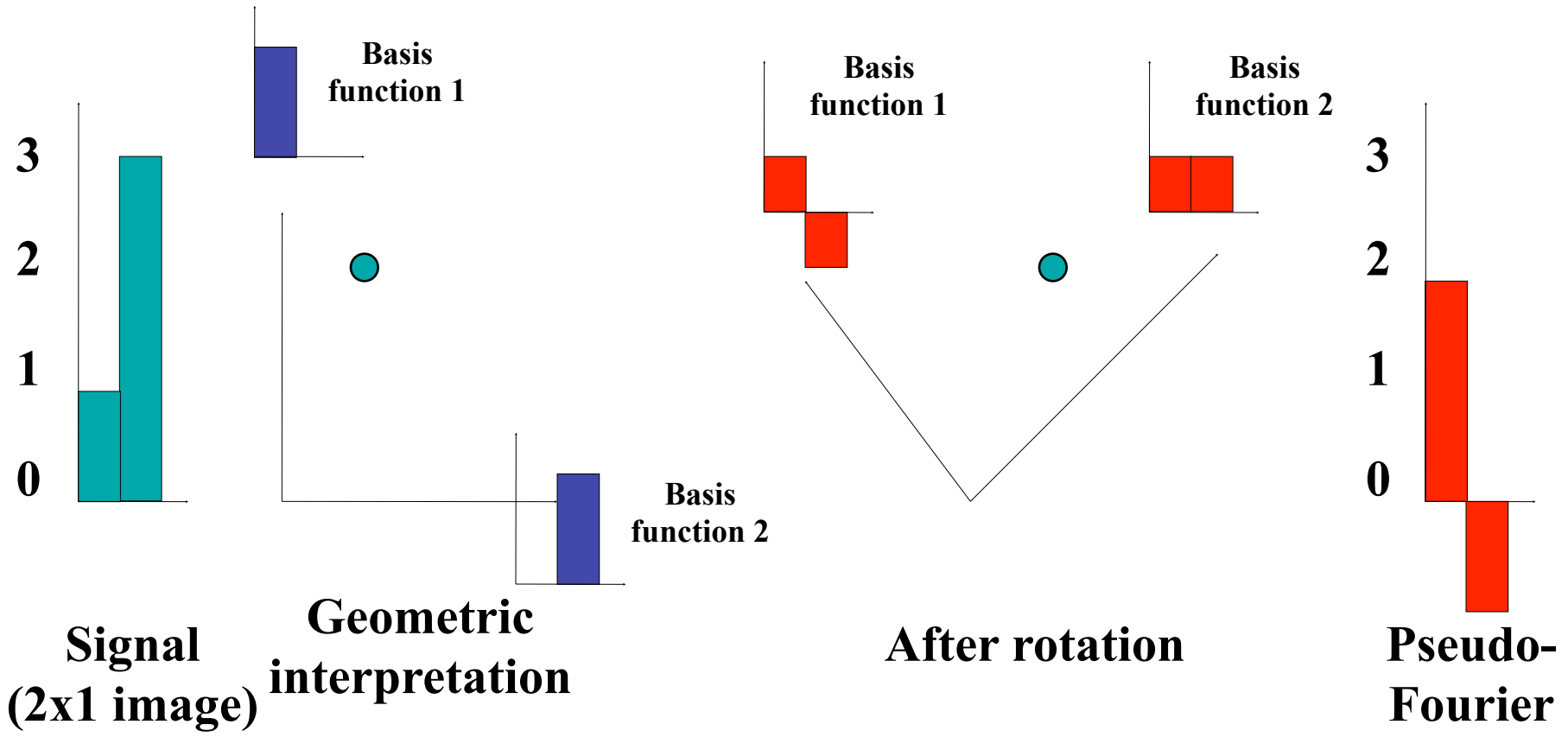
Signal (2x1 image)

Geometric interpretation

Fourier as change of basis

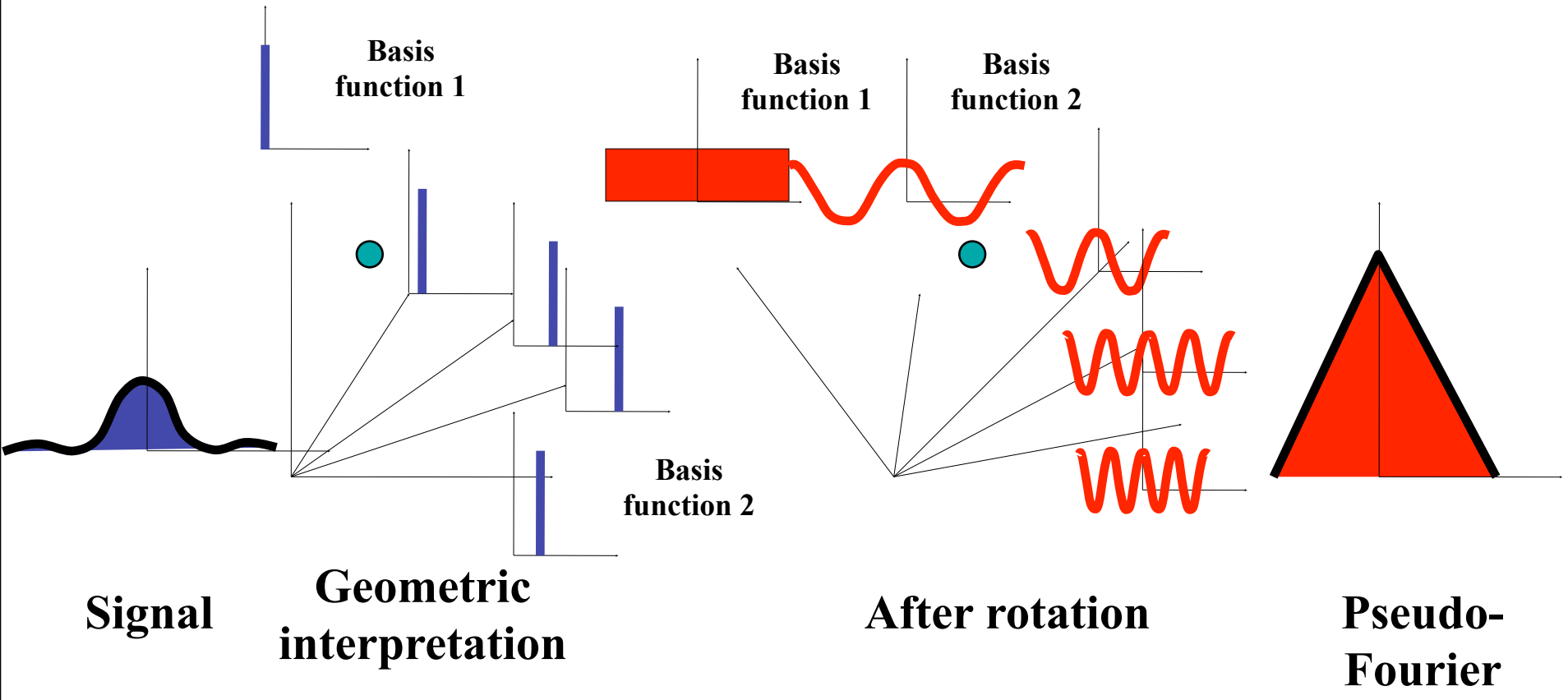
- Shuffle the data to reveal other information
- E.g., take average & difference: matrix

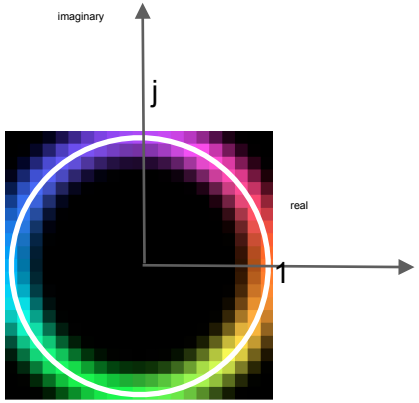
$$\begin{bmatrix} 0.5 & 1 \\ 0.5 & -1 \end{bmatrix}$$



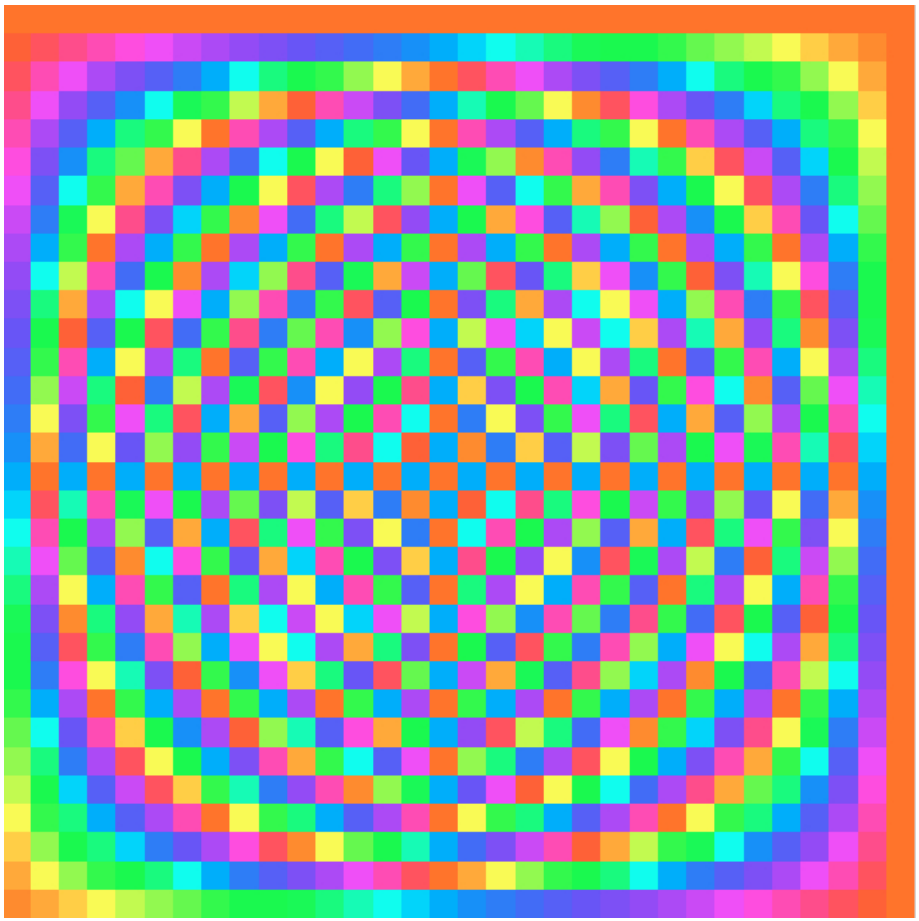
Fourier as change of basis

- Same thing with infinite-dimensional vectors





color key



Fourier transform matrix



input signal

$$F[m,n] = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} f[k,l] e^{-\pi i \left(\frac{km}{M} + \frac{\ln}{N} \right)}$$

Slide: Bill Freeman

Other presentations of Fourier

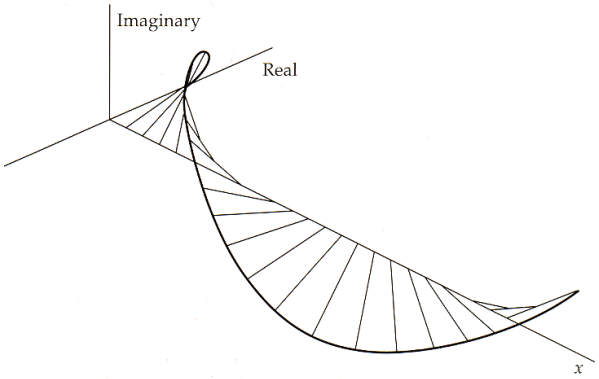
- Start with Fourier series with periodic signal
- Heat equation
  - more or less special case of convolution
  - iterate -> exponential on eignevalues

Motivations

- Insights & mathematical beauty
- Sampling rate and filtering bandwidth
- Computation bases
  - FFT: faster convolution
  - E.g. finite elements, fast filtering, heat equation, vibration modes
    - Preconditioning
- Optics: wave nature of light & diffraction

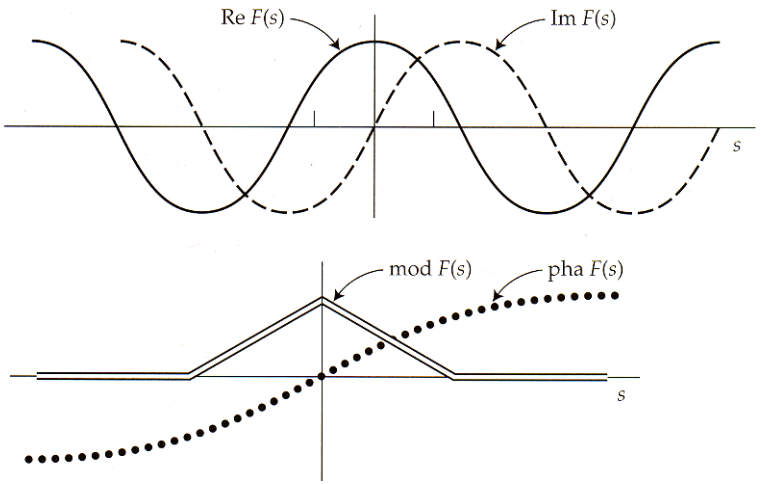
Phase

- Don’t forget the phase! Fourier transform results in complex numbers

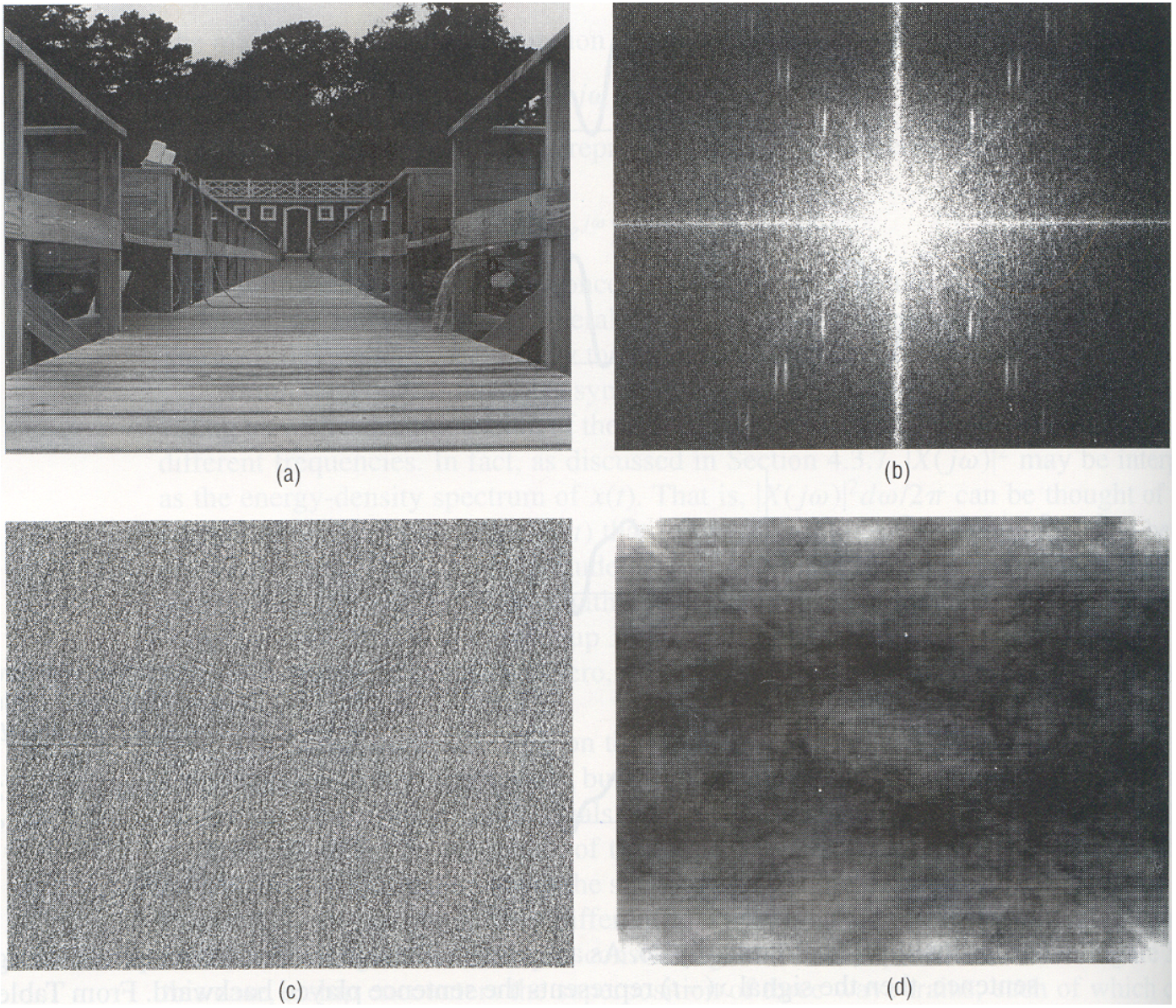


- Can be seen as sum of sines and cosines

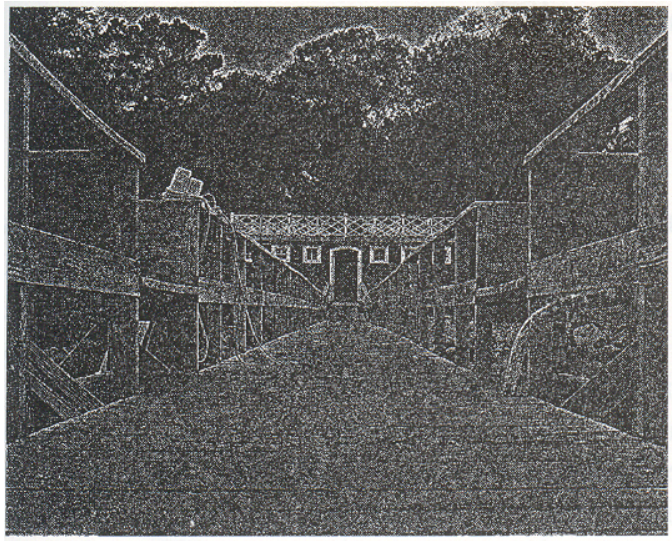
- Or modulus/phase



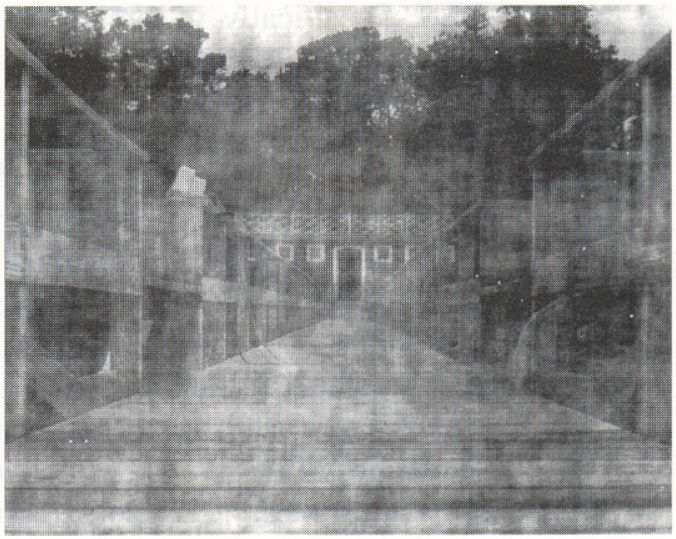
Phase is important!



Phase is important!



(e)



(f)



(g)

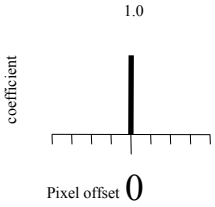
Figure 6.2 (a) The image shown in Figure 1.4; (b) magnitude of the two-dimensional Fourier transform of (a); (c) phase of the Fourier transform of (a); (d) picture whose Fourier transform has magnitude as in (b) and phase equal to zero; (e) picture whose Fourier transform has magnitude equal to 1 and phase as in (c); (f) picture whose Fourier transform has phase as in (c) and magnitude equal to that of the transform of the picture shown in (g).

Filters in Fourier

Analysis of our simple filters



original

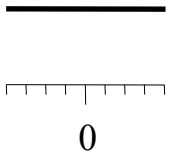




Filtered
(no change)

1.0 constant

spectrum: F(ω)=1

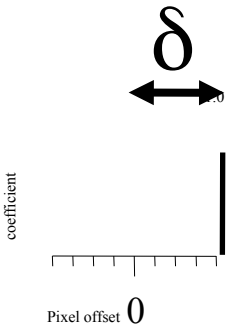


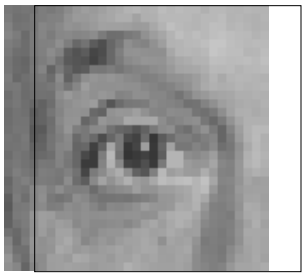
(yes, I am now using the definition
without 1/sqrt(2pi)

Analysis of our simple filters



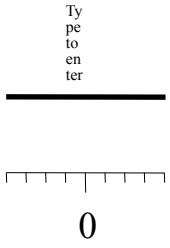
original





shifted

spectrum:



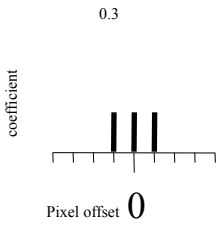
$$F(\omega) = e^{-2\pi j \omega \delta}$$

Constant magnitude, linearly shifted phase

Analysis of our simple filters

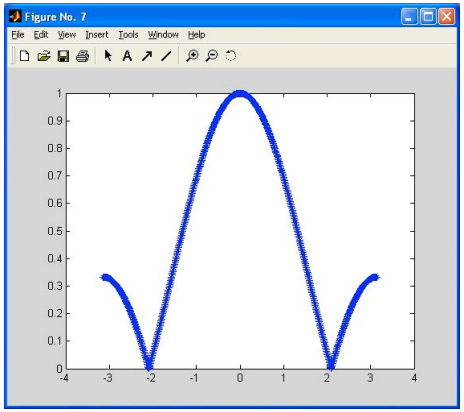


original



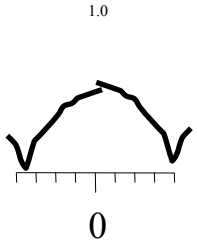


blurred



Low-pass filter

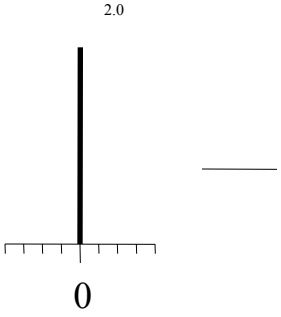
spectrum:
$F(\omega)=\mathrm{sinc}(\omega)$
$=\sin(\omega)/\omega$

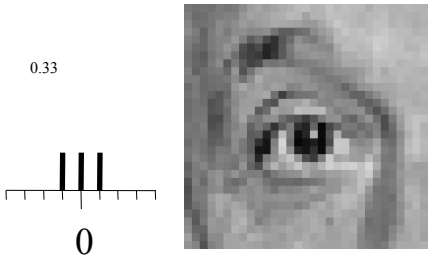


Analysis of our simple filters

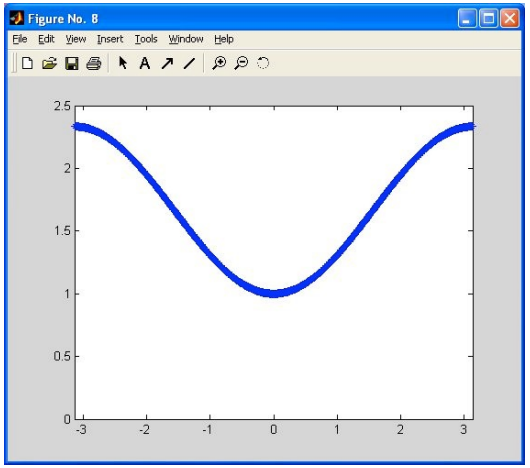


original

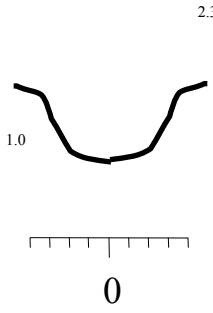




sharpened



high-pass filter



spectrum:
$F(\omega)=2-\mathrm{sinc}(\omega)$

Deconvolution

Back to convolution/deconvolution

- Convolution in space is a multiplication in Fourier
- Note y the observed blurry image and
  x the original sharp one
- $y = g \otimes x$ in the spatial domain
- $Y = GX$ in the Fourier domain
  - A frequency does not depend on the other ones

Careful: G is not a gradient anymore, it's the convolution matrix we used to call M

Invert the convolution theorem

- Given $y=g \otimes x$ and g, we seek an estimate x' of x
- How do you invert a multiplication?
  - Division!
- $X'(\omega)=Y(\omega)/G(\omega)$

- DECONVOLUTION IS A DIVISION IN THE FOURIER DOMAIN !
- Which means it is also a convolution in the spatial domain, by the inverse Fourier transform of 1/G

Questions?

- Given $y=g \otimes x$ and g, we seek an estimate x' of x
- How do you invert a multiplication?
  - Division!
- $X'(\omega)=Y(\omega)/G(\omega)$

- DECONVOLUTION IS A DIVISION IN THE FOURIER DOMAIN !
- Which means it is also a convolution in the spatial domain, by the inverse Fourier transform of 1/G

Potential problem?

- Deconvolution is a division in the Fourier domain
- Division by zero is bad!
  - Information is lost at the zeros of the kernel spectrum G

Noise problem

- Even when there is no zero, noise is a big problem
- If G has small number, division amplifies noise
- if $y=g \otimes x+v$ where v is additive noise
- $Y=GX+V$
- $X'=(GX+V)/G$
  $= X+V/G$
- V is amplified by 1/G. This is why you typically get more high-frequency noise with deconvolution

Noise problem

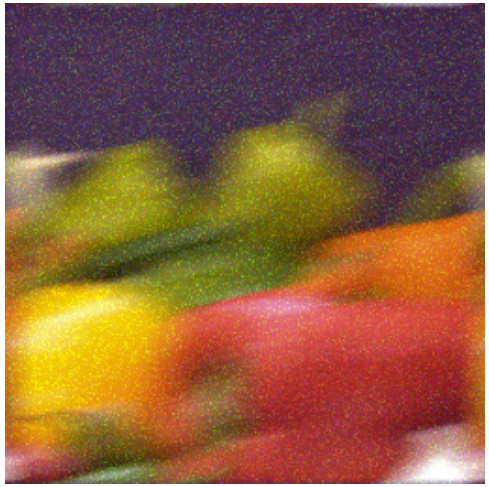
blurry, no noise



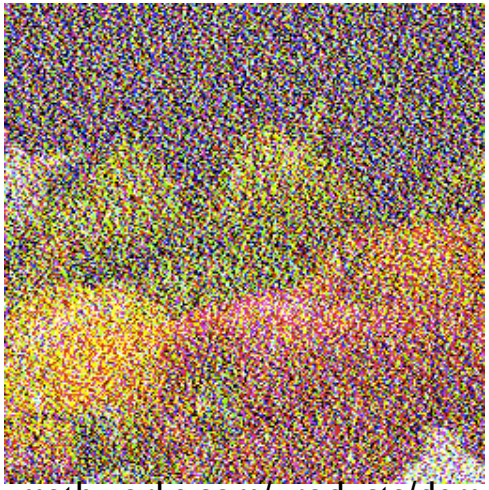
deconvolved



blurry, with noise



deconvolved



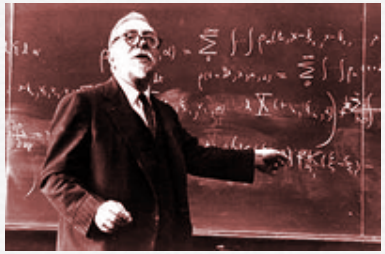
http://www.mathworks.com/products/demos/image/deblur_wiener/deblur.html

Wiener

Back to convolution

- Naive: divide by Kernel spectrum G
- Assume we know the expected noise power spectrum and expected signal power spectrum
- Can we tweak 1/G to reduce output noise ?
- Maybe if we use something smaller than 1/G
  - we won't amplify noise as much
  - but the inversion won't be as correct

http://www.norbertwiener.umd.edu/NW/index.html



Wiener deconvolution

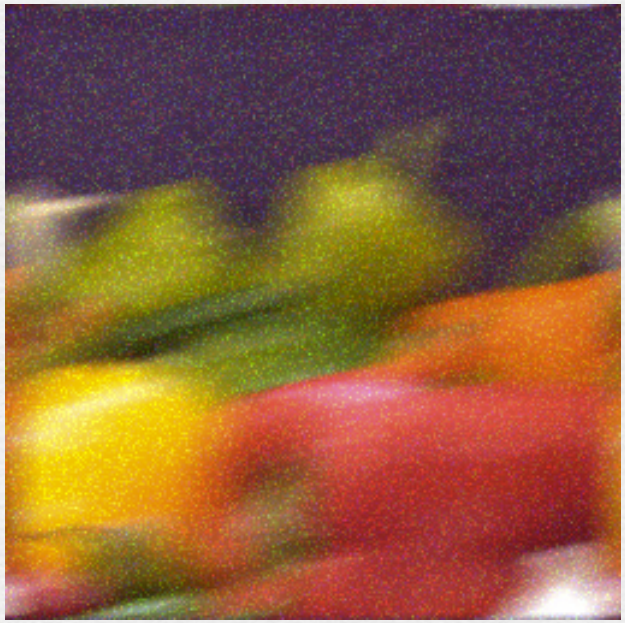
- Find the gain H that minimize $||X'(\omega)-X(\omega)||^2$ where $X'=HY$
- We need to know the signal noise ratio $SNR(\omega)=E(|X(\omega)|^2)/E(|V(\omega)|^2)$
- Optimal filter

$$\frac{1}{G(\omega)}\left[\frac{|G(\omega)|^2}{|G(\omega)|^2+1/SNR(\omega)}\right]$$

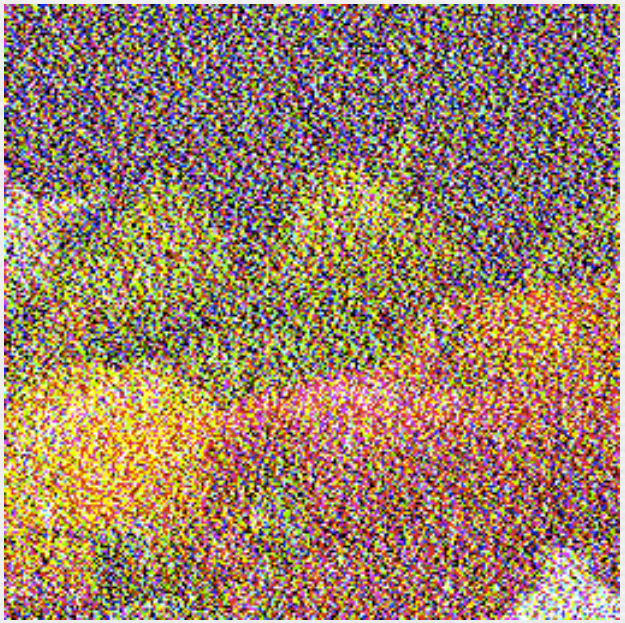
- See e.g. http://en.wikipedia.org/wiki/Wiener_deconvolution
  - careful, their notations are different from mine

Results from Wiener

blurry with noise



naive deconvolution



Wiener deconvolution



http://www.mathworks.com/products/demos/image/deblur_wiener/deblur.html

Long story short

- Under the right SNR model,
  Wiener deconvolution is gradient regularization
  - Because the noise usually has flat power in frequency
  - The signal has power $1/f^2$ where f is frequency
  - The gradient corresponds to multiplying by f in Fourier, hence power is multiplied by f^2

Blind deconvolution

Blind deconvolution

- So far we have assumed we know the kernel
- When both x and g are unknown, the problem is badly ill-posed
- It is called blind deconvolution

- See e.g.
  - http://www.wisdom.weizmann.ac.il/~levina/papers/deconvLevinEtal09-MIT-TR.pdf
  - http://cs.nyu.edu/~fergus/research/deblur.html
  - http://www.wisdom.weizmann.ac.il/~levina/papers/deconvLevinEtal11-TR.pdf
  - http://www.wisdom.weizmann.ac.il/~levina/papers/LevinEtalCVPR2011Code.zip

Fergus et al. Siggraph 2006

Removing Camera Shake from a Single Photograph

Rob Fergus[1] Barun Singh[1] Aaron Hertzmann[2] Sam T. Roweis[2] William T. Freeman[1]

[1]MIT CSAIL [2]University of Toronto



Figure 1: *Left*: An image spoiled by camera shake. *Middle*: result from Photoshop "unsharp mask". *Right*: result from our algorithm.

http://people.csail.mit.edu/fergus/research/deblur.html

Overview

Original



Our algorithm



Close-up

Original



Naïve Sharpening



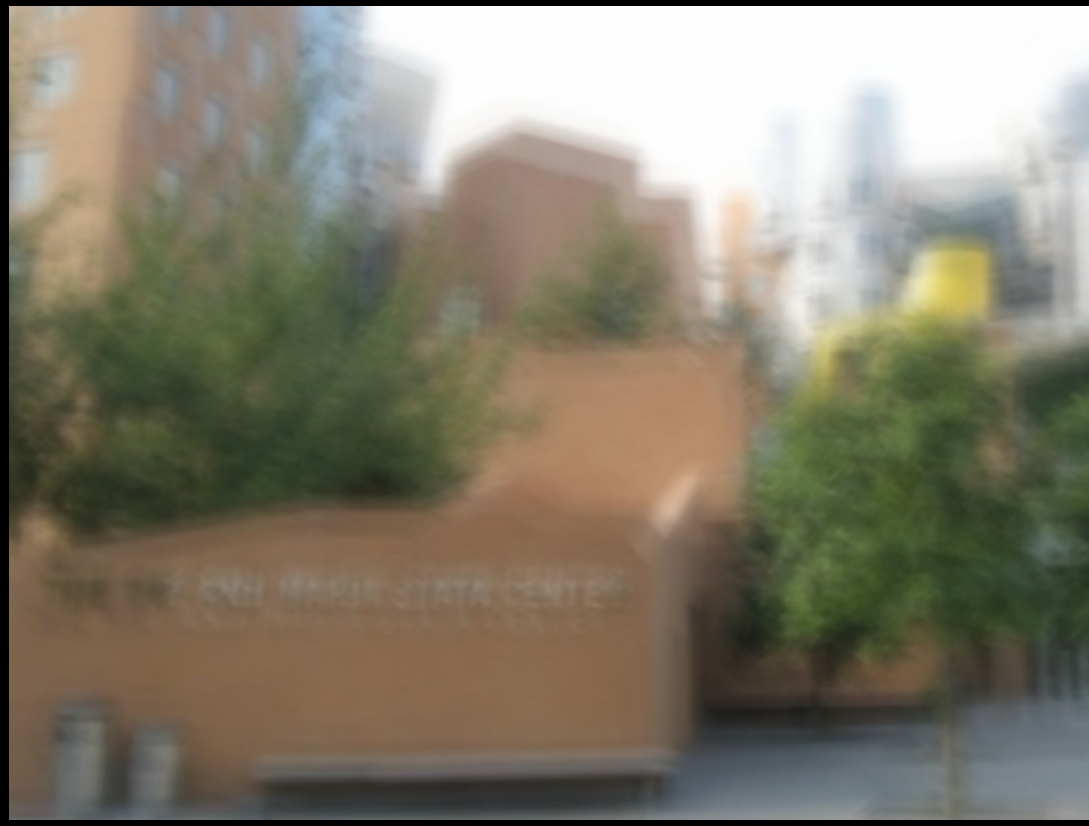
Our algorithm



Let’s take a photo



Blurry result



Slow-motion replay



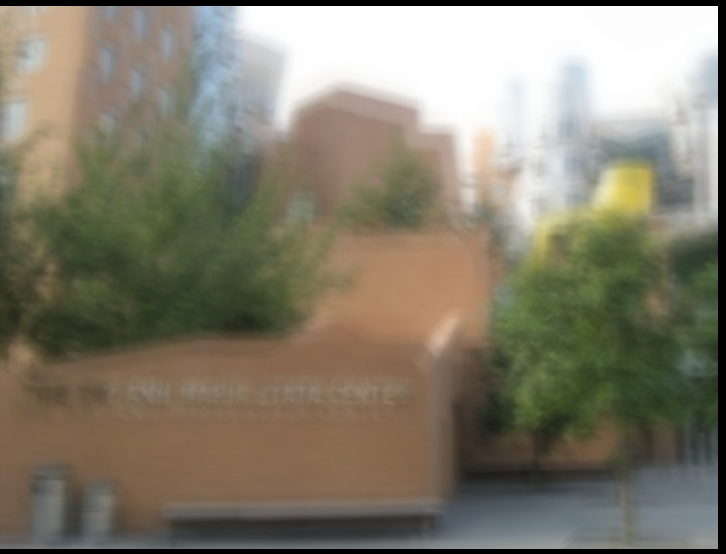
Slow-motion replay





Motion of camera

Image formation process



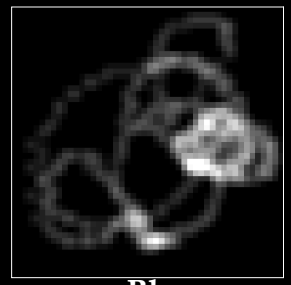
Blurry image

=



Sharp image

⊗



Blur kernel

Input to algorithm

Desired output

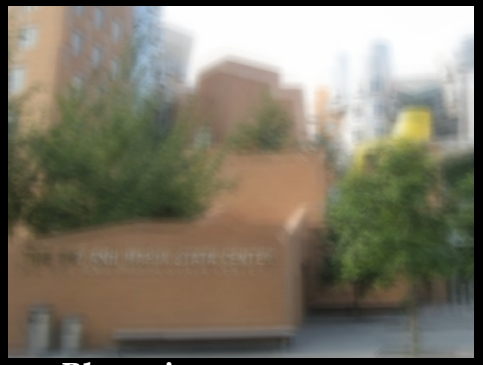
Assume unknown

Convolution operator

Multiple possible solutions

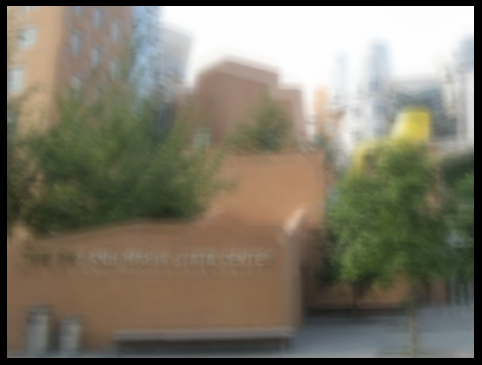
Sharp image

Blur kernel

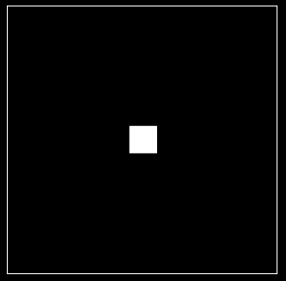


Blurry image

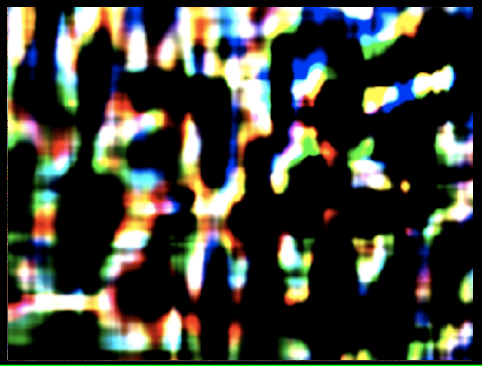
=



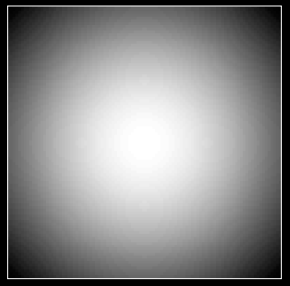
⊗



=



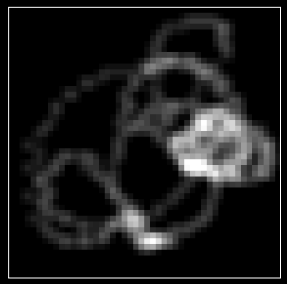
⊗



=



⊗



Idea

- Discriminate between sharp and unsharp images

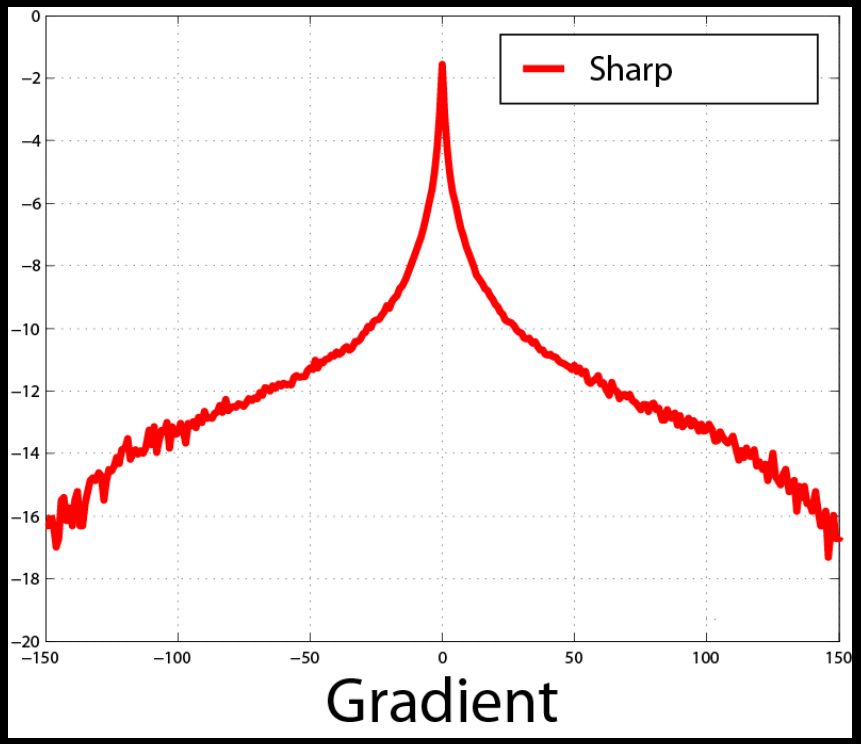
- As we will see, it does not completely work, but it will get us to a working solution

Natural image statistics

Characteristic distribution with heavy tails



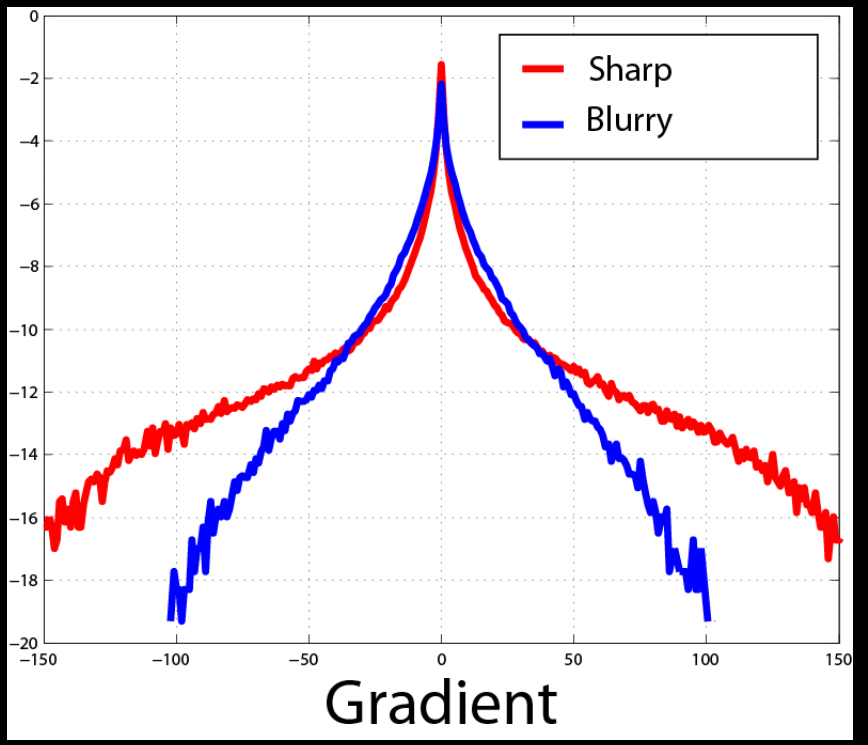
Histogram of image gradients



Blurry images have different statistics



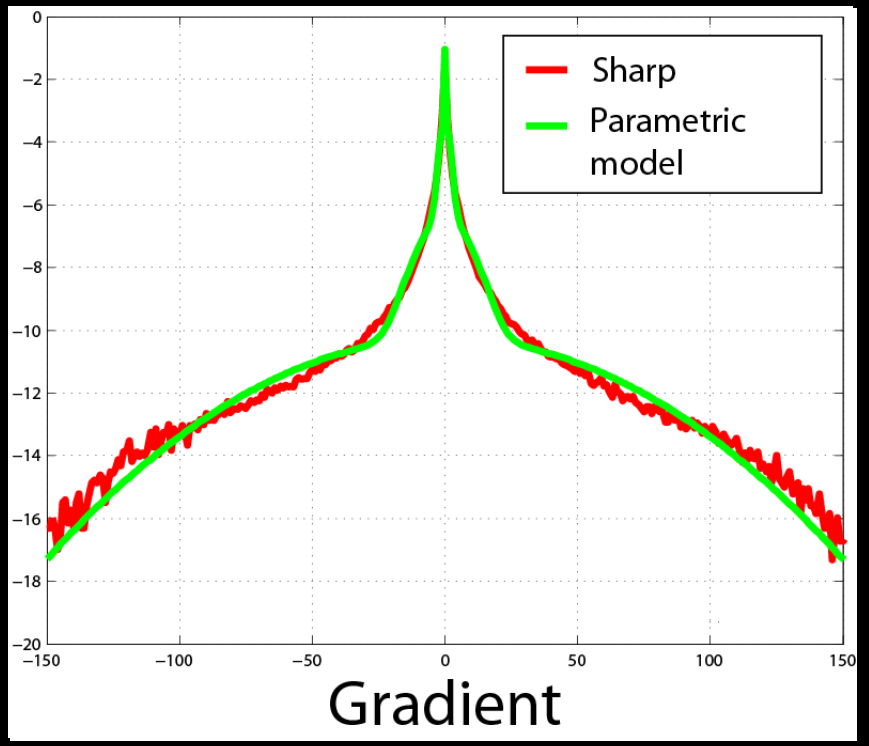
Histogram of image gradients



Parametric distribution



Histogram of image gradients



Use parametric model of sharp image statistics

Three sources of information

1. Reconstruction constraint:



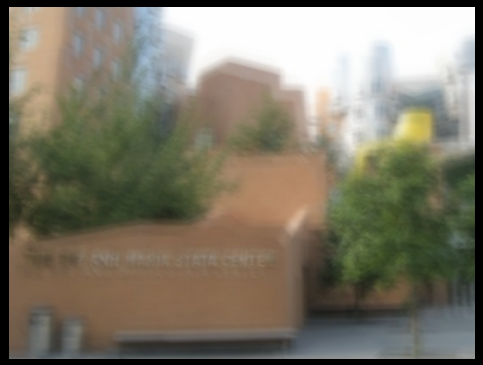
Estimated sharp image

$\otimes$



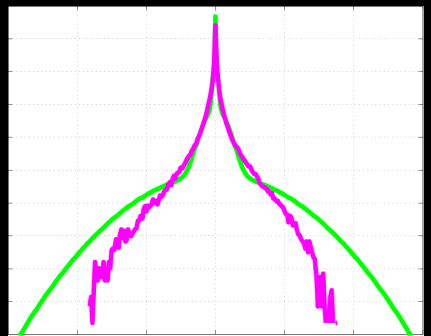
Estimated blur kernel

$=$



Input blurry image

2. Image prior:



Distribution of gradients

3. Blur prior:



Positive & Sparse

How do we use this information?

- Obvious thing to do:
  - Combine 3 terms into an objective function
  - Run conjugate gradient descent
  - This is Maximum a-Posteriori (MAP)

Maximum A-Posteriori

y – observed blurry image
x – unobserved sharp image
b – blur kernel

$$p(x, b|y) \propto p(y|x, b)p(x)p(b)$$

Likelihood

Latent image prior

Blur prior

$$e^{-\frac{\sum_i (x_i * b - y_i)^2}{2\sigma^2}} \qquad e^{-\lambda \sum_i |f(x_i)|} \qquad \geq 0$$

Type to enter text

Sparse and

i – image patch index
f – derivative filter

Maximum A-Posteriori

y – observed blurry image
x – unobserved sharp image
b – blur kernel

$$p(x, b|y) \propto p(y|x, b)p(x)p(b)$$

Likelihood

Latent image prior

Blur prior

Type to enter text

Sparse and

$$e^{-\frac{\sum_i (x_i * b - y_i)^2}{2\sigma^2}} \qquad e^{-\lambda \sum_i |f(x_i)|} \qquad \geq 0$$

i – image patch index
f – derivative filter

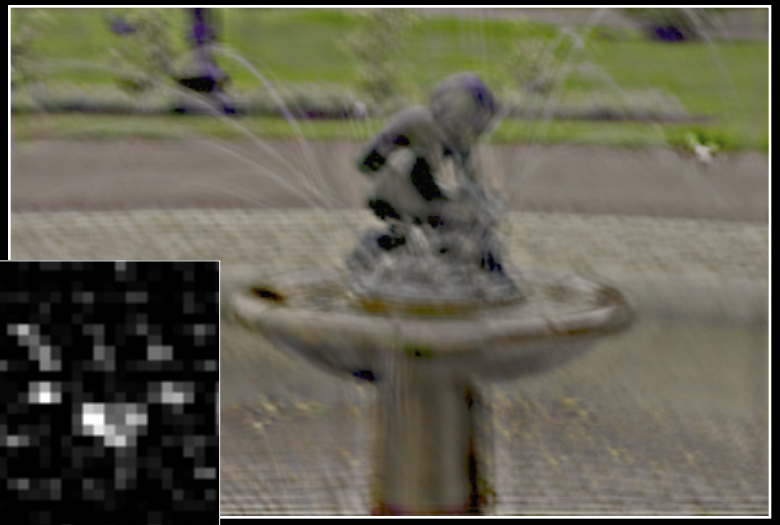
Take the log and it becomes a big linear least square
similar to our gradient-based Wiener
(but b is also unknown)

Results from MAP estimation

Input blurry image



Maximum a-Posteriori (MAP)



Not so great!

Variational Bayes

Ensemble Learning for Blind Image Separation and Deconvolution

James Miskin and David J. C. MacKay

Summary. In this chapter Ensemble Learning is applied to the problem of Blind Source Separation and Deconvolution of images. It is assumed that the observed images were constructed by mixing a set of images (consisting of independent identically distributed pixels), convolving the mixtures with unknown blurring filters and then adding Gaussian noise.

Ensemble Learning is used to approximate the intractable posterior distribution over the unknown images and unknown filters by a simpler separable distribution. The mixture of Laplacians used for the source prior respects the positivity of the image and favours sparse images. The model is trained by minimising the Kullback-Leibler divergence between the true posterior distribution and the approximating ensemble.

Unlike Maximum-Likelihood methods, increasing the number of hidden images does not lead to overfitting the data and so the number of hidden images in the

http://citeseer.ist.psu.edu/cache/papers/cs/16537/http:zSzzSzwol.ra.phy.cam.ac.ukzSzjwm1003zSzspringer_chapter8.pdf/miskin00ensemble.pdf

Setup of variational approach

Lots of math ;-(

Need likelihood and prior in same space, so use gradients:

$$x * b = y \rightarrow \nabla x * b = \nabla y$$

$$p(\nabla x, b|\nabla y) \propto p(\nabla y|\nabla x, b)p(\nabla x)p(b)$$

Likelihood

$$p(\nabla y|\nabla x, b) = \prod_i \mathcal{N}(\nabla y_i|\nabla x_i * b, \sigma^2)$$

Prior on latent image gradients – mixture of Gaussians

$$p(\nabla x) = \prod_i \sum_{c=1}^{C} \mathcal{N}(\nabla x_i|0, s_c)$$

Prior on blur elements – mixture of Exponentials

$$p(b) = \prod_j \sum_{d=1}^{D} \mathcal{E}(b|\lambda_c)$$

i – image pixel
j – blur pixel

We use C=4, D=4

Also have Gamma hyperpriors on

σ^2 (inferred)
s_c (fixed, but different for each scale)
λ_c (fixed)

Variational inference

Lots of math ;-(

Approximate posterior $p(\nabla x, b|\nabla y)$ $q(\nabla x, b)$

Cost function $KL(q(\nabla x, b)||p(\nabla x, b|\nabla y))$

Assume $q(\nabla x, b) = q(\nabla x)q(b)$

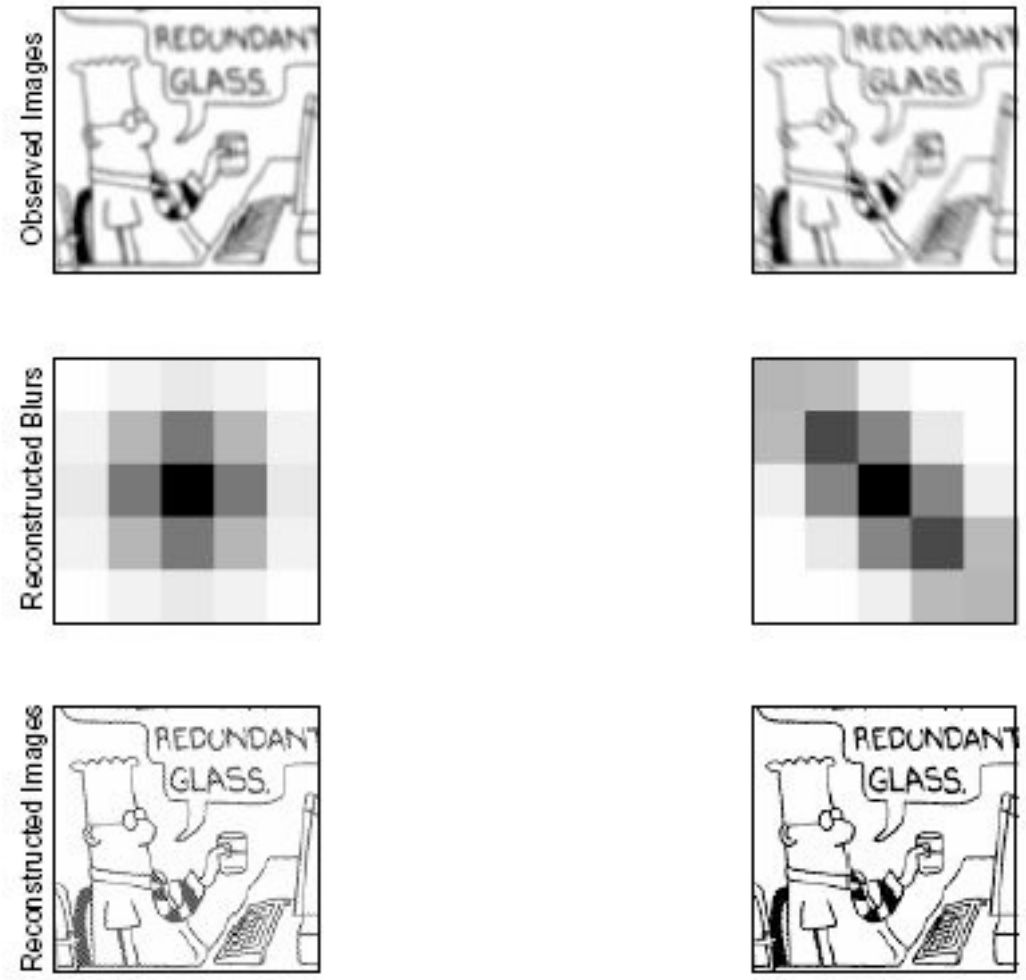
$q(\nabla x)$ is Gaussian on each pixel

$q(b)$ is rectified Gaussian on each pixel

Use gradient descent, alternating between updating $q(\nabla x)$ marginalizing out over $q(b)$ vice versa

Adapted code from Miskin & Mackay 2000

Miskin and Mackay, 2000

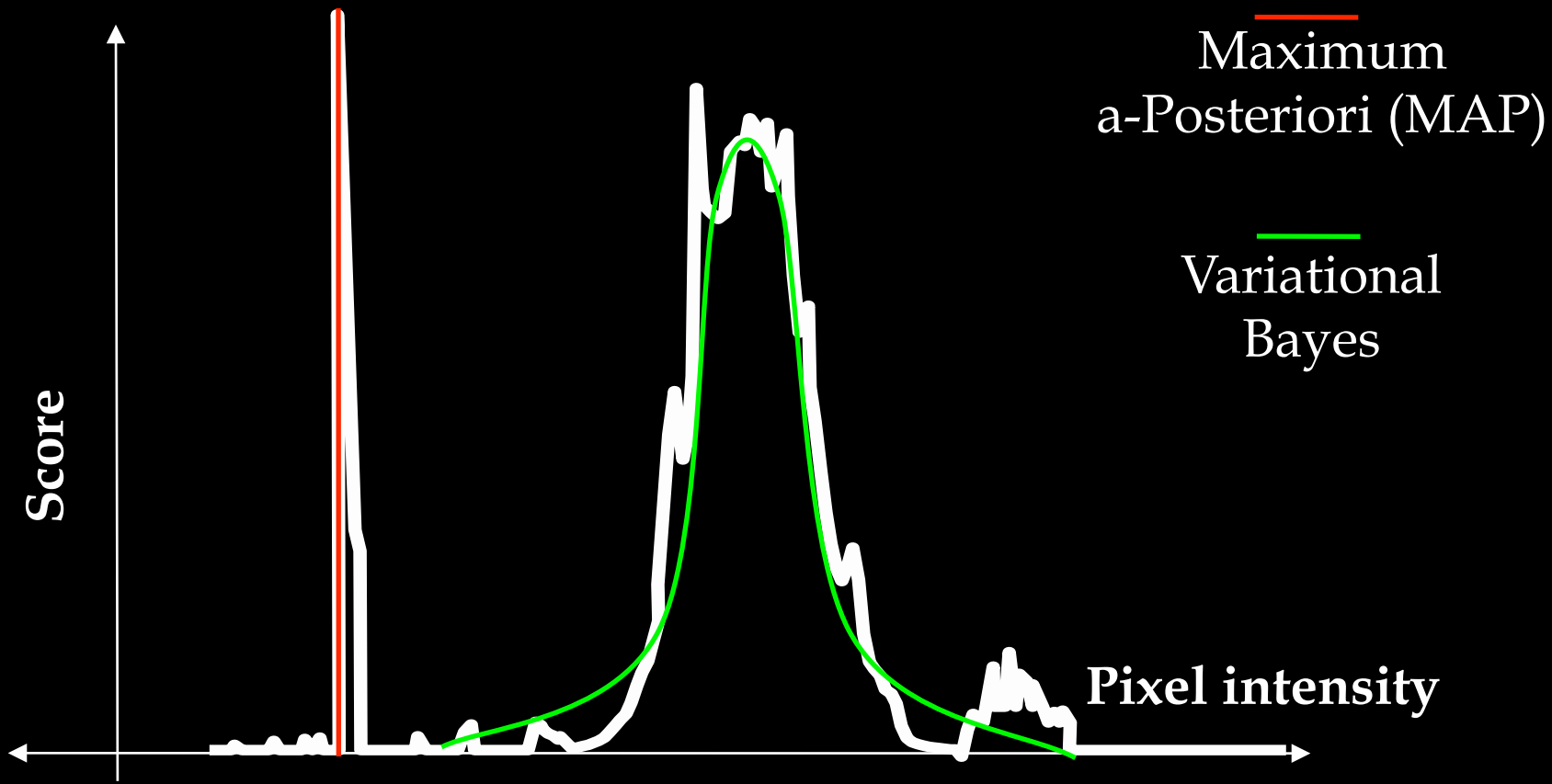


But some cartoons

Fig. 9. Demonstration of the deconvolution of two blurred images. In each test the same image was blurred by a different filter. The reconstructed filters match the true filters. The reconstructed images are close to the hidden images. [Dilbert image Copyright©1997 United Feature Syndicate, Inc., used with permission.]

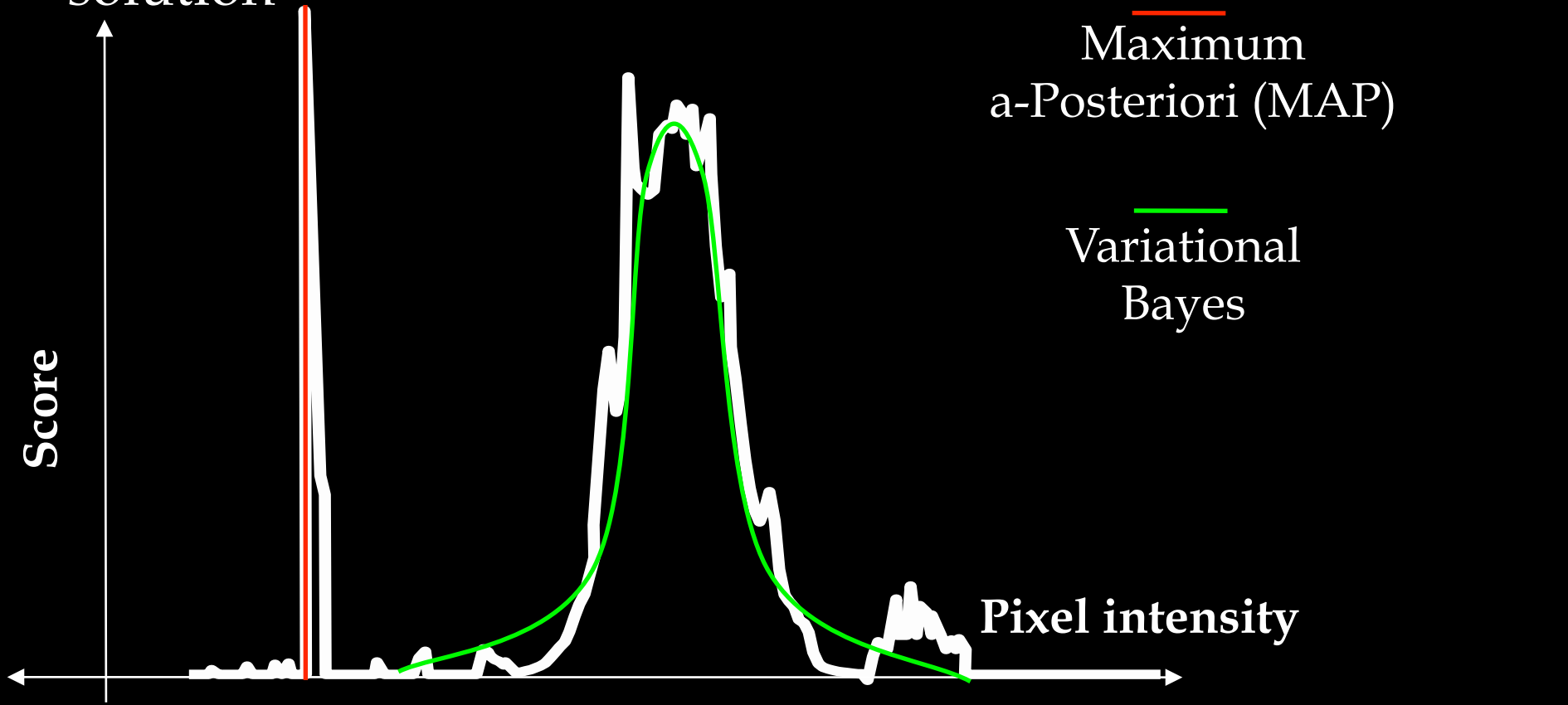
Variational Bayesian method

Objective function for a single variable



Variational Bayesian method

Objective function for a single variable
MAP is confused by spike
one should consider local mass around a solution



Updated explanation

- Why MAP is bad & marginalization is useful
  - i.e. integrate over all possible sharp images to evaluate the quality of a potential kernel
  - http://www.wisdom.weizmann.ac.il/~levina/papers/deconvLevinEtal09-MIT-TR.pdf
- How to implement it easily
  - http://www.wisdom.weizmann.ac.il/~levina/papers/deconvLevinEtal11-TR.pdf
  - The method is simple but the justification is involved

Inferring the kernel: multiscale method

Input image



Convert to grayscale

Remove gamma correction

User selects patch from image

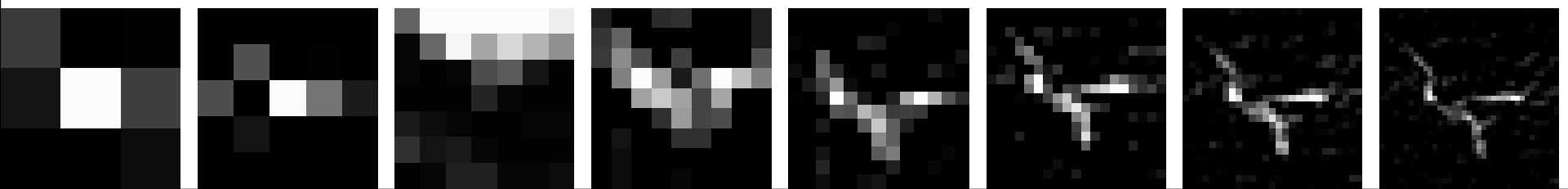
Initialize 3x3 blur kernel

Loop over scales

Variational Bayes

Upsample estimates

Use multi-scale approach to avoid local minima:

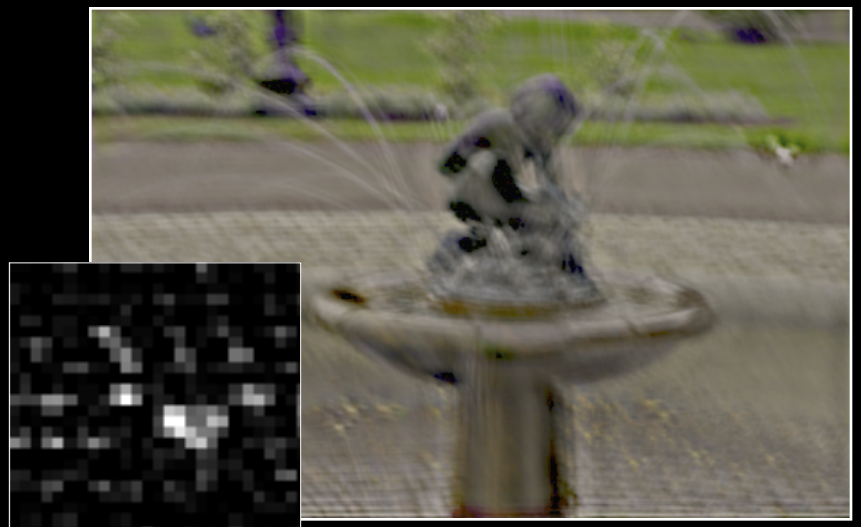


Results from MAP estimation

Input blurry image



Maximum a-Posteriori (MAP)



Our method: Variational Bayes




Blurry synthetic image

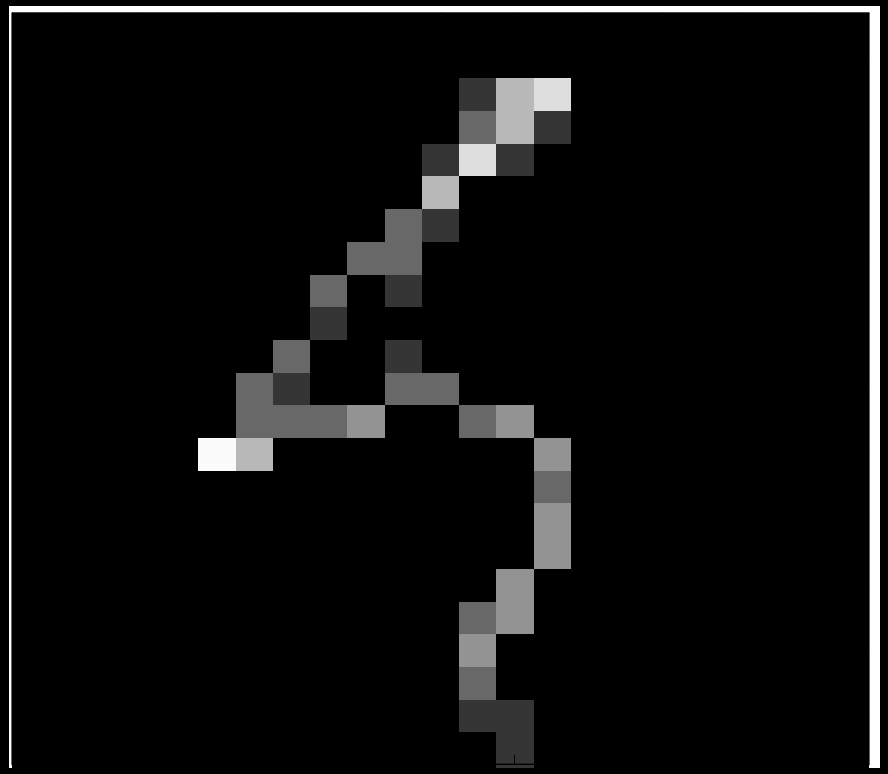

Our output


Ground truth

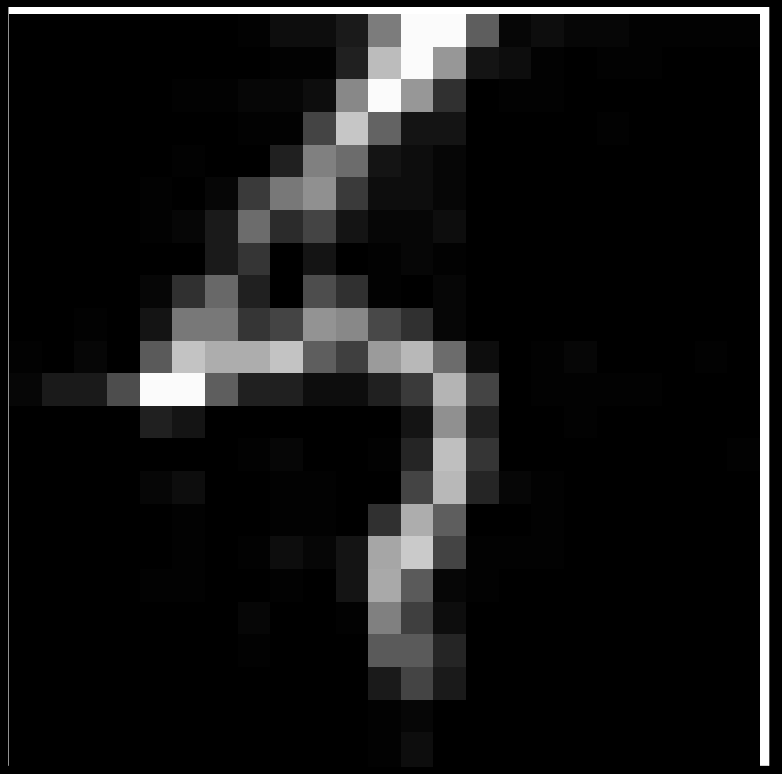
Matlab's `deconvblind`



True kernel



Estimated kernel



Results on real images

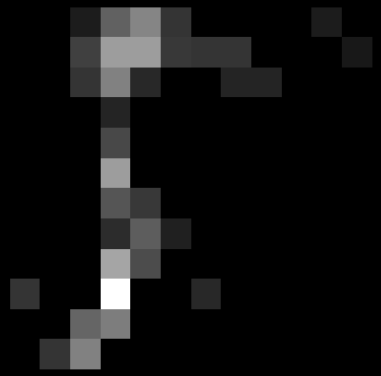
- Submitted by people from their own photo collections
- Type of camera unknown

- Output does contain artifacts
  - Increased noise
  - Ringing
- Compares well to existing methods

Original photograph

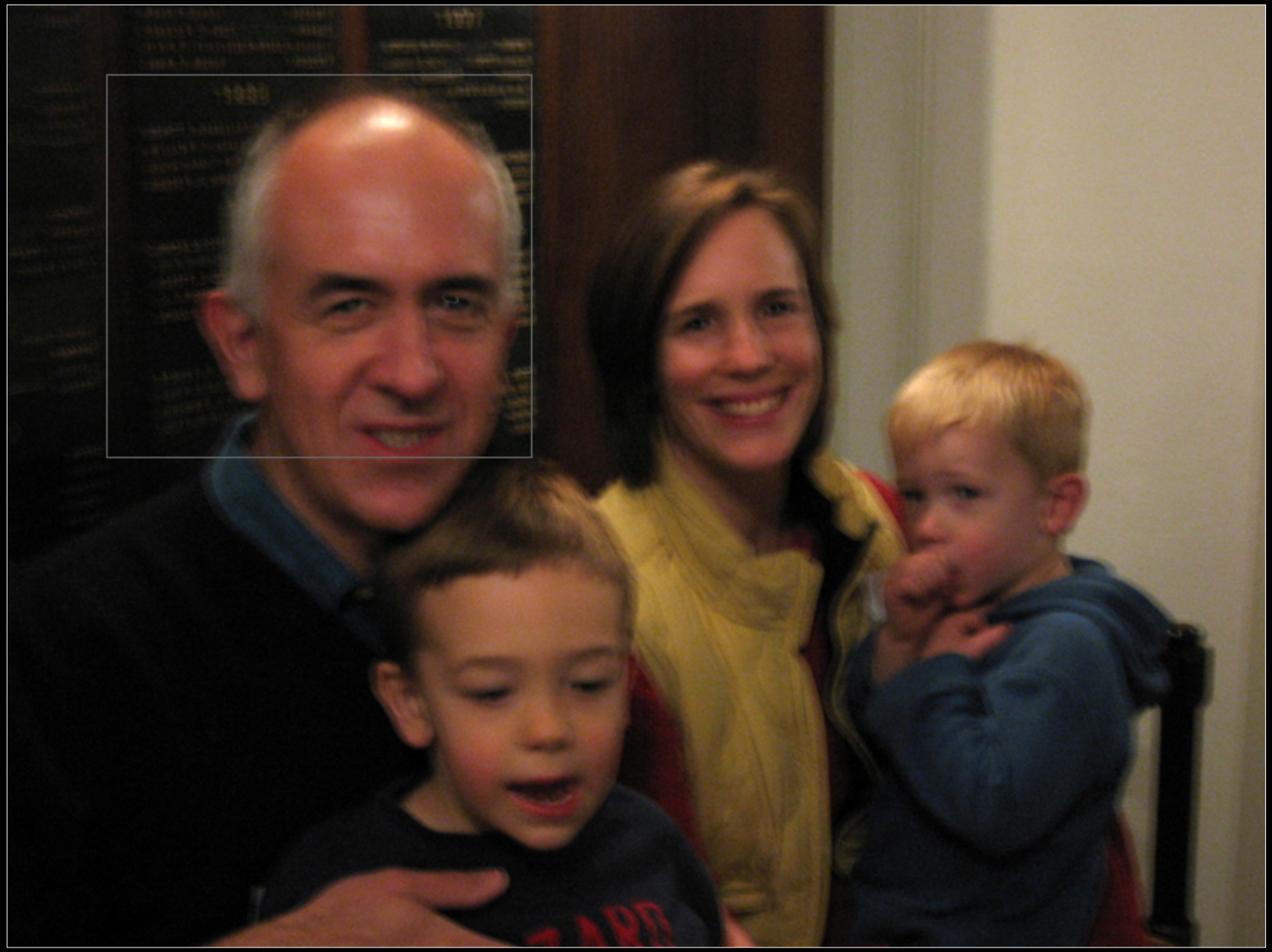


Blur kernel

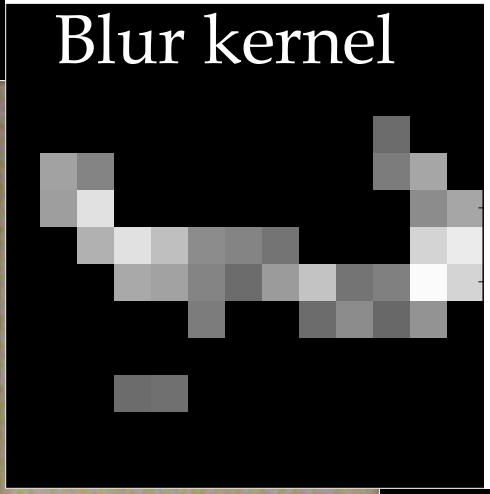


Our output

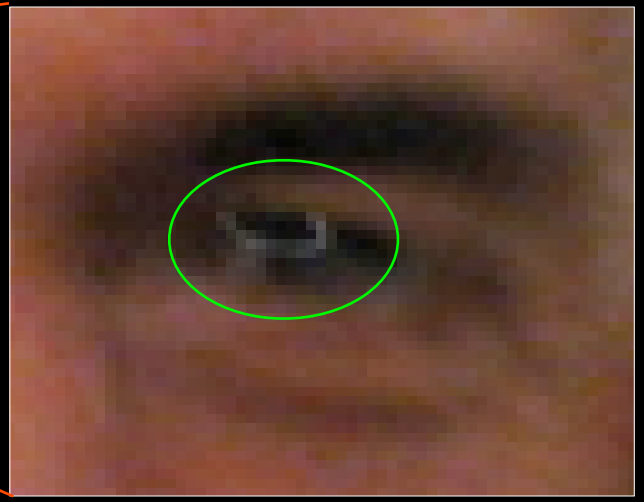
Original photograph



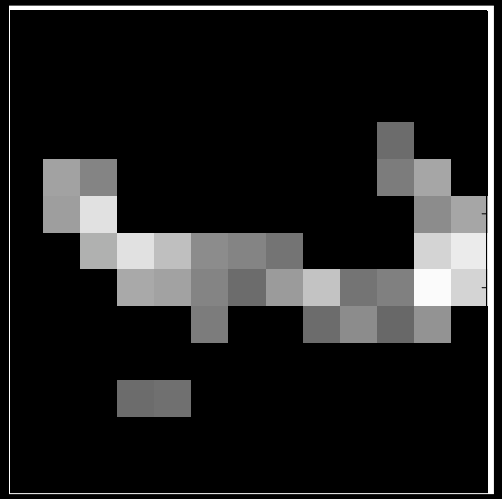
Our output
















2
4
6
8
10
12
14
2
4
6
8
10
12
14

Before



After



Image artifacts & estimated kernels

Blur kernels

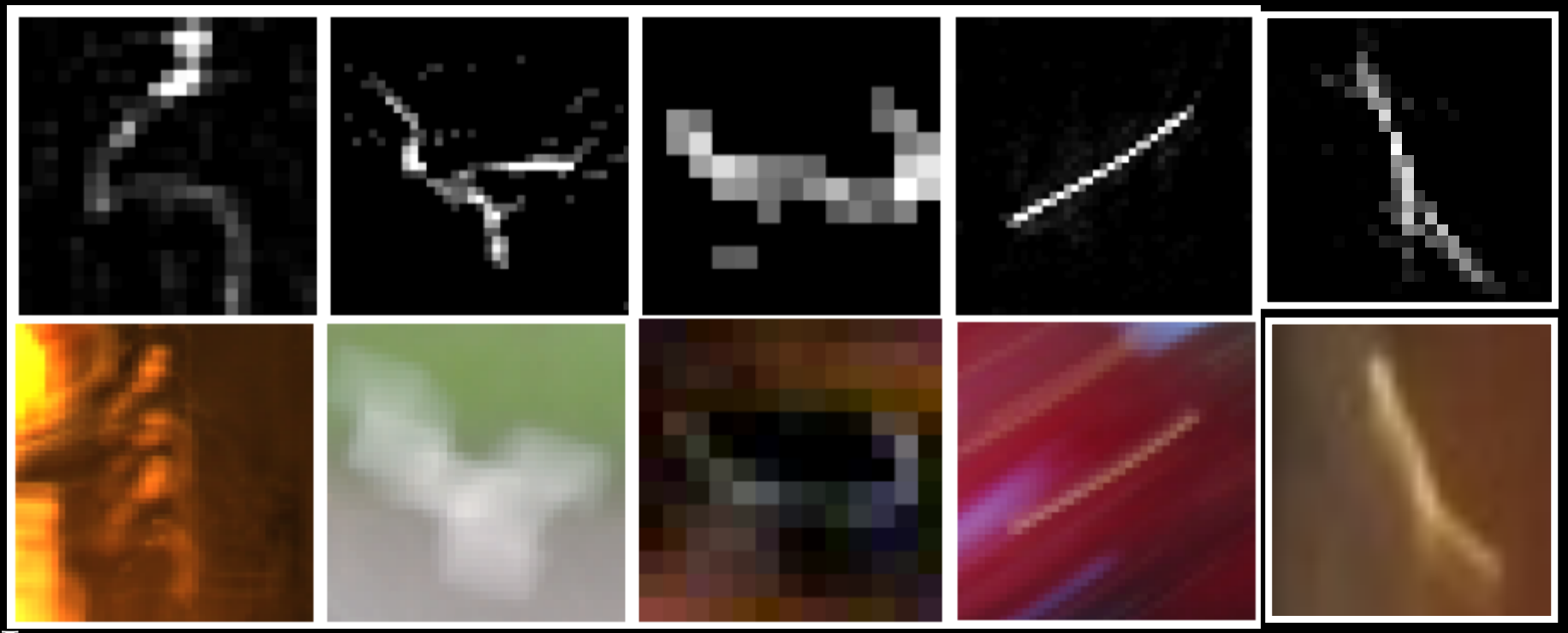
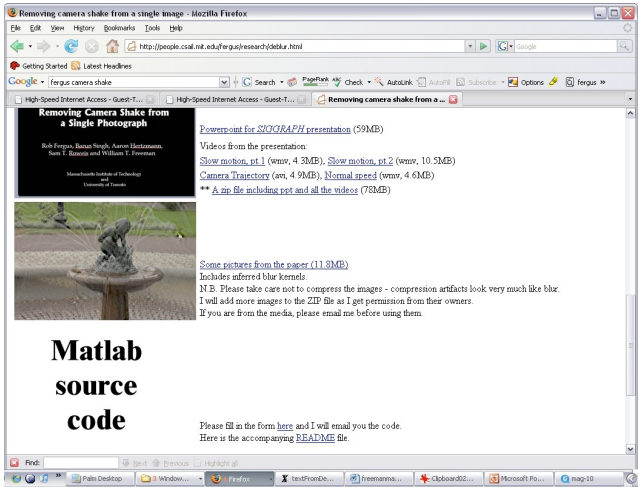


Image patterns

Note: blur kernels were inferred from large image patches,
NOT the image patterns shown

Available code

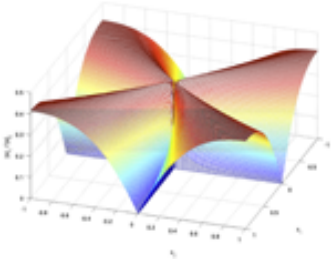
- Original Fergus et al.
  - http://cs.nyu.edu/~fergus/research/deblur.html



- Simplified/improved by Levin et al.
  - http://www.wisdom.weizmann.ac.il/~levina/papers/LevinEtalCVPR2011Code.zip

See also

- Krishnan et al.
- http://cs.nyu.edu/~dilip/research/blind-deconvolution/

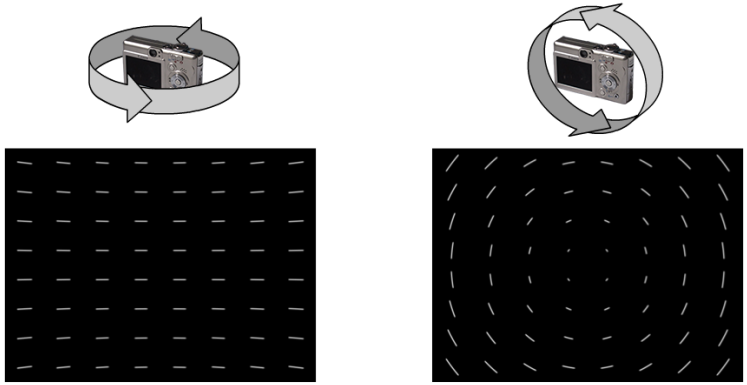




Extensions

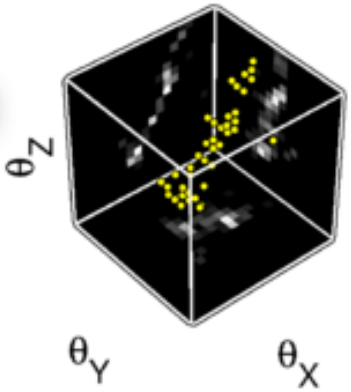
Non-shift-invariant blur

- Whyte et al, CVPR 2010
- Full model of 3D rotation of camera
- Kernel=>
  Density of 3D angles
- http://www.di.ens.fr/willow/research/deblurring/





Camera axes



The weights, or "blur kernel" defined over orientations of the camers

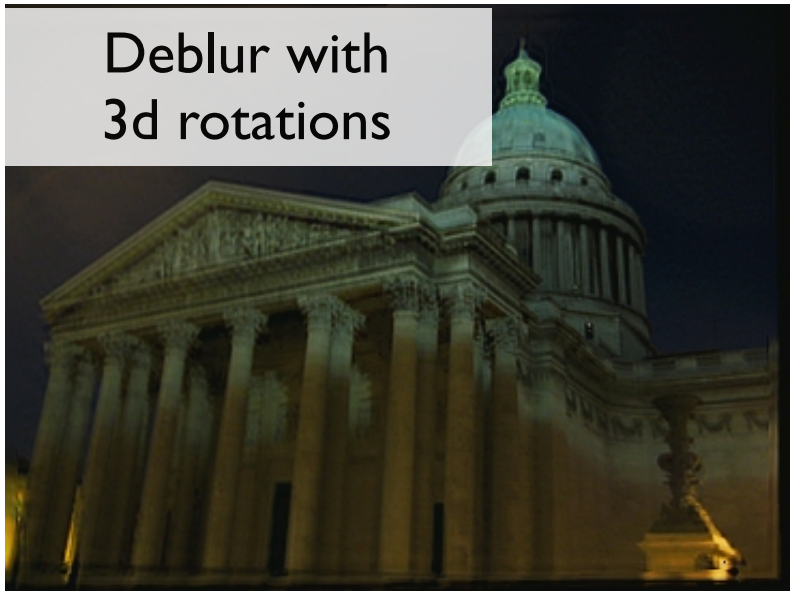


A blurry image, with the equivalent PSFs overlaid at several locations

Results





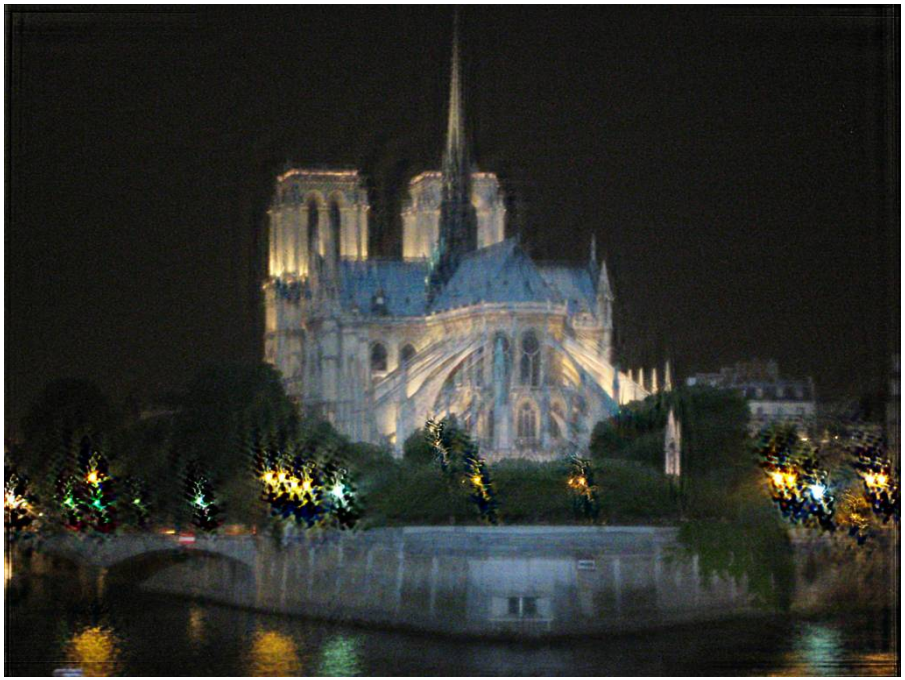


Dealing with clipped pixels

- Whyte et al. http://www.di.ens.fr/willow/research/sa
  - Oberved image is not linear at pixels that are too bright (dynamic range issue)



Blurry input



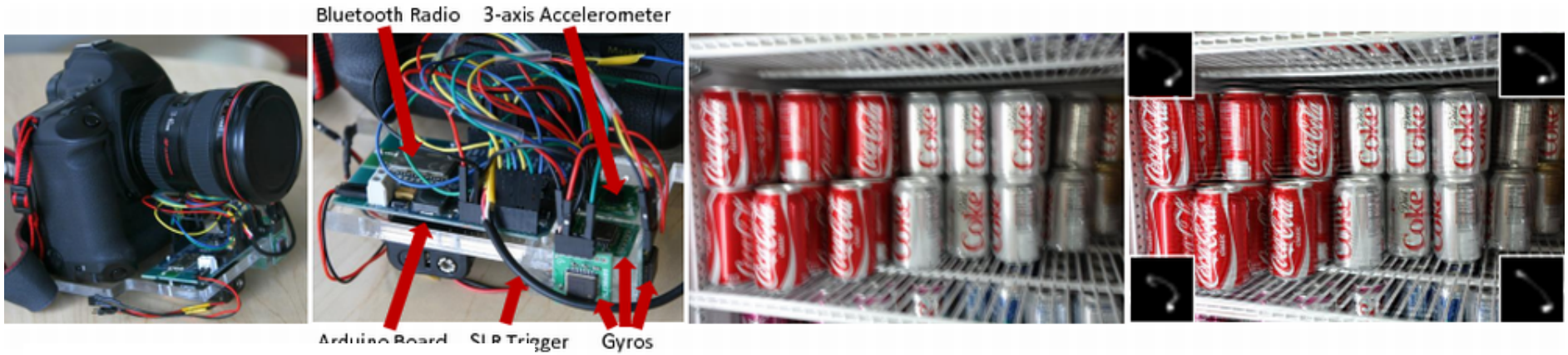
Standard deblurring

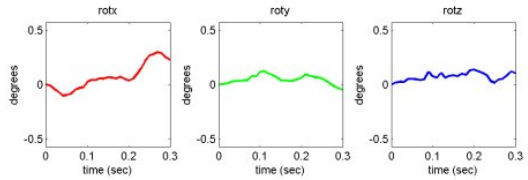


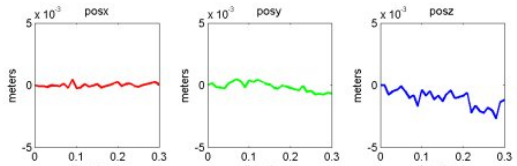
Deblurring that deals with clipped pixels

Using a gyro to measure blur

- Joshi et al. Siggraph 2010
- http://research.microsoft.com/en-us/um/redmond/groups/ivm/imudeblurring/







Before

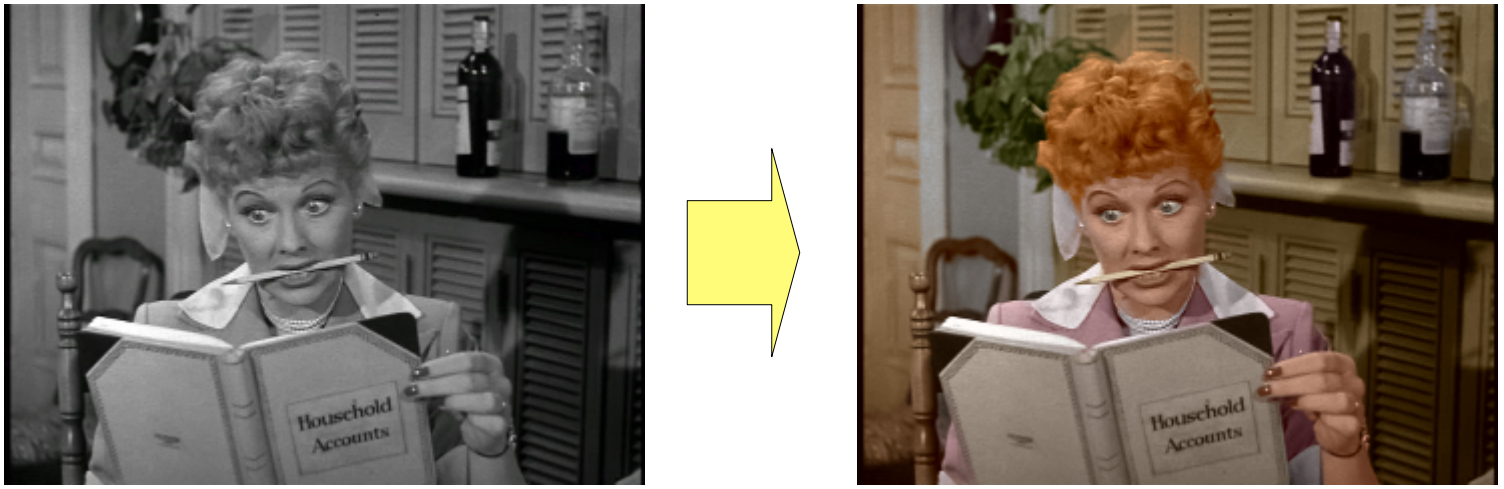


After



COLORIZATION

Colorization



Colorization: a computer-assisted process of adding color to a monochrome image or movie. (Invented by Wilson Markle, 1970)

Slide courtesy of Anat Levin

Motivation

- **Colorizing black and white movies and TV shows**



Earl Glick (Chairman, Hal Roach Studios), 1984:
"You couldn't make Wyatt Earp today for $1 million an episode. But for $50,000 a segment, you can turn it into color and have a brand new series with no residuals to pay"

Slide courtesy of Anat Levin

Google Search: Colorization Tutorial - Netscape

File Edit View Go Bookmarks Tools Window Help

http://www.google.com/search?hl=en&lr=&ie=UTF-8&q=Colorization+Tutorial

Search

Mail Home Radio Netscape Search Bookmarks

Google Search: Colorization Tutorial

Google

Web Images Groups News Froogle **more »**

Colorization Tutorial Search Advanced Search Preferences

Web Results **1** - **10** of about **2,610** for **Colorization Tutorial**. (**0.10** seconds)

Worth1000.com | Photoshop Contests | Are you Worthy™ | tutorial

... Yet Another **Colorization Tutorial Colorizing** Using CMYK Adjustment Layers By Elysium Print View. Variety is the spice of life, and **...**
www.worth1000.com/**tutorial**.asp?sid=161018 - 21k - Cached - Similar pages

Worth1000.com | Photoshop Contests | Are you Worthy™ | tutorial

... Read more. Yet Another **Colorization Tutorial** 5/5/2003 Variety is the spice of life, and with that in mind, I offer you yet another **colorization tutorial**. **...**
www.worth1000.com/**tutorial**.asp - 48k - 22 Jun 2004 - Cached - Similar pages
[More results from www.worth1000.com]

COLORIZATION TUTORIAL BY JARAH

... For this **tutorial** you will need: Paint Shop Pro 8 ~ download a trial version here. **...** Now choose a **color** from the **color** palette you would like for the star to be **...**
www.jcwdesigns.com/**colorization**/color.html - 4k - Cached - Similar pages

COLORIZATION TUTORIAL BY JARAH

... Make note of the **color** code for the Hue and Saturation as we will be using these codes in our **colorization**. Now **...** me. Happy **colorizing**!! **...**
www.jcwdesigns.com/**colorization**/color2.html - 4k - Cached - Similar pages

Slide courtesy of Anat Levin

Document: Done (0.375 secs)

Typical Colorization Process



Images from: "Yet Another Colorization Tutorial"

http://www.worth1000.com/tutorial.asp?sid=161018

Slide courtesy of Anat Levin

Typical Colorization Process

- **Delineate region boundary**



Images from: "Yet Another Colorization Tutorial"

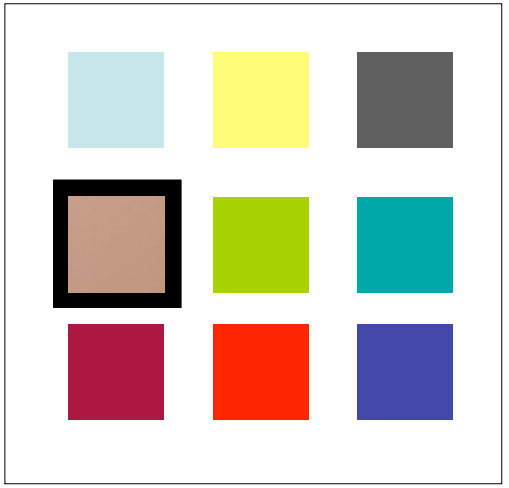
http://www.worth1000.com/tutorial.asp?sid=161018

Slide courtesy of Anat Levin

Typical Colorization Process

- **Delineate region boundary**
- **Choose region color from palette.**





Images from: "Yet Another Colorization Tutorial"

http://www.worth1000.com/tutorial.asp?sid=161018

Slide courtesy of Anat Levin

Typical Colorization Process

- **Delineate region boundary**
- **Choose region color from palette.**



Images from: "Yet Another Colorization Tutorial"

http://www.worth1000.com/tutorial.asp?sid=161018

Slide courtesy of Anat Levin

Typical Colorization Process

- **Delineate region boundary**
- **Choose region color from palette.**



Images from: "Yet Another Colorization Tutorial"

http://www.worth1000.com/tutorial.asp?sid=161018

Slide courtesy of Anat Levin

Video Colorization Process

- **Delineate region boundary**
- **Choose region color from palette.**
- **Track regions across video frames**

Slide courtesy of Anat Levin

Colorization Process Discussion



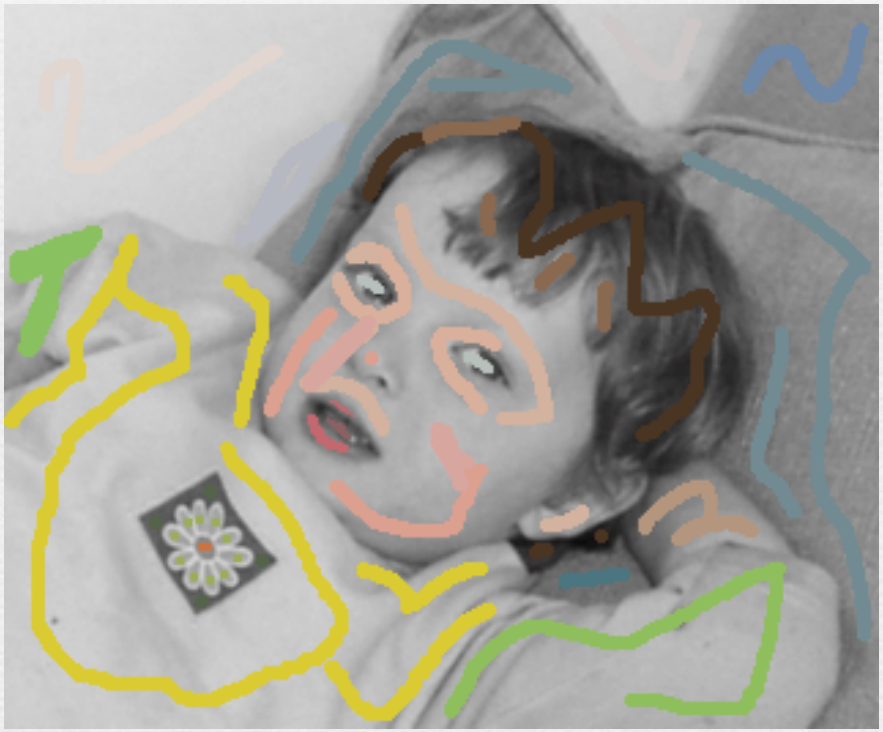
Time consuming and labor intensive

- **Fine boundaries.**
- **Failures in tracking.**

Slide courtesy of Anat Levin

Colorization using Optimization

- Anat Levin, Dani Lischinski, Yair Weiss
  http://www.cs.huji.ac.il/~yweiss/Colorization/



Input BW image
with user color strokes



Result

Principle

- Colors vary smoothly
- Except at strong edges



- Technical idea:
  - unknowns: pixel color (e.g. UV chrominance in YUV)
  - Energy function that encourages neighboring pixels to have the same value
    - Strength depends on greyscale similarity:
      the color is more likely the same if the greyscale is the same
  - User stroke = boundary condition
  - It's all a big least square problem

- Similar to Laplace/Poisson
- Solve for U, minimize

$$J(U) = \sum_r \left[U(r) - \sum_{s \in N(r)} \frac{1}{4} U(s) \right]^2$$

- constrained to boundary conditions at the user's strokes

- Idea: weight less pixels *s* that are very different from a given center pixel *r*
- The energy definition now varies spatially (non-homogenous)

$$J(U) = \sum_r \left[U(r) - \sum_{s \in N(r)} w_{rs} U(s) \right]^2$$

- where w_{rs} is high when *r* and *s* have similar greyscale values in the input, and low if they are different
- N(r) is the neighborhood of a pixel, e.g. 3x3

- Gaussian on intensity (Y) difference

$$w_{\mathbf{rs}} \propto e^{-(Y(\mathbf{r})-Y(\mathbf{s}))^2/2\sigma_{\mathbf{r}}^2}$$

- Or Normalized correlation

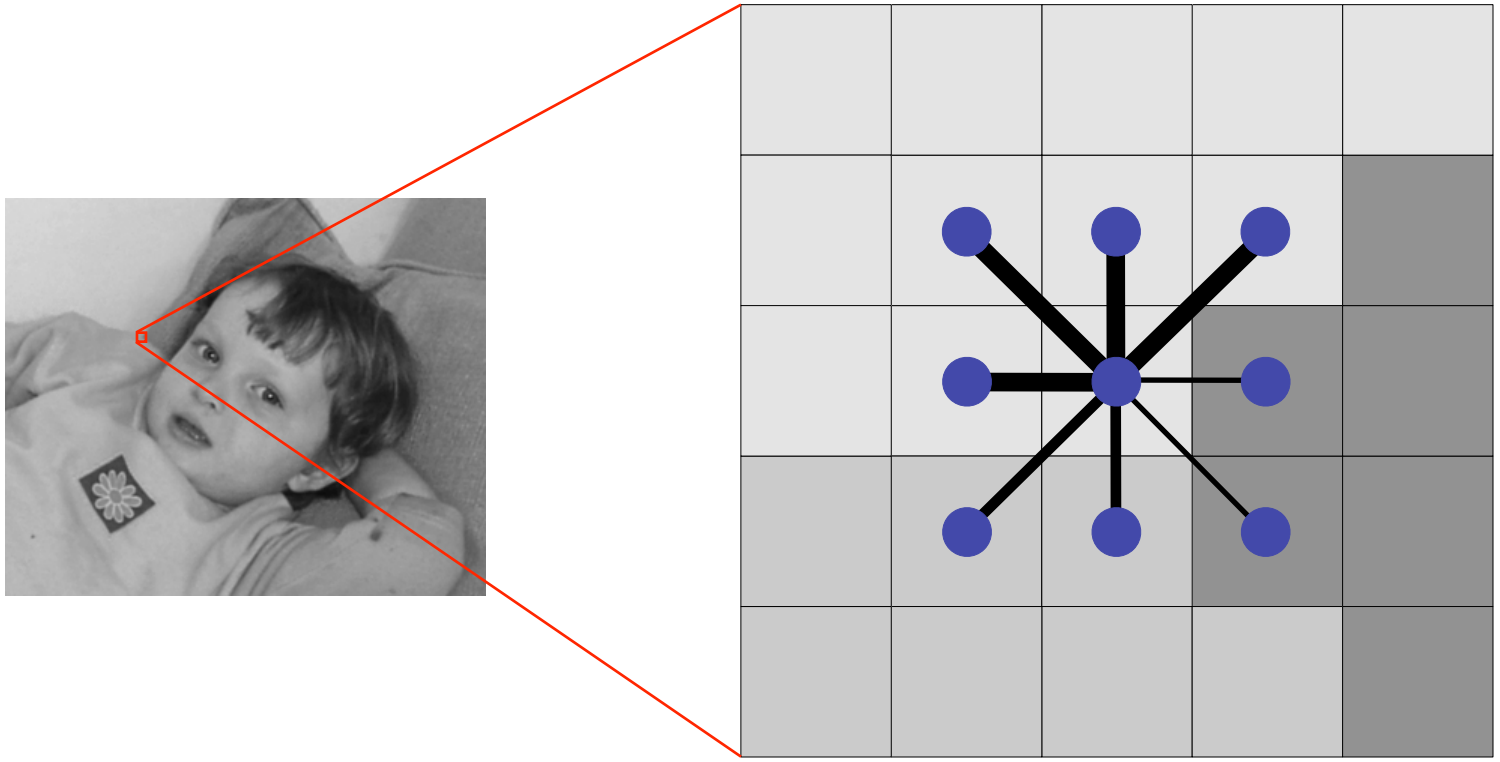
$$w_{\mathbf{rs}} \propto 1 + \frac{1}{\sigma_{\mathbf{r}}^2}(Y(\mathbf{r}) - \mu_{\mathbf{r}})(Y(\mathbf{s}) - \mu_{\mathbf{r}})$$

- where μ is the local mean intensity, σ^2 the variance
- All normalized to sum to 1 in a window

Affinity Functions

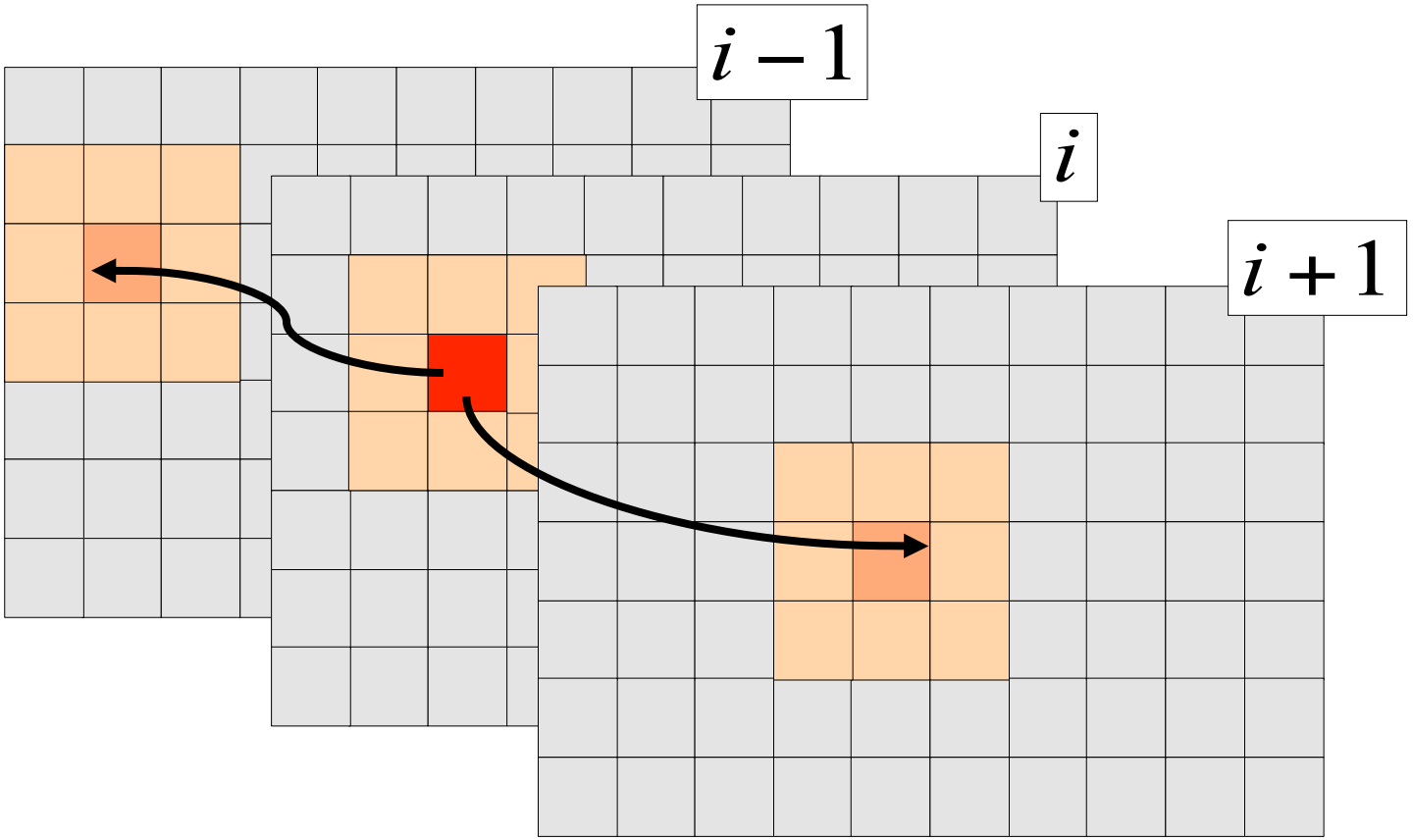
$$w_{rs} \propto e^{-(Y(r)-Y(s))^2/\sigma_r^2}$$

σ_r proportional to local variance



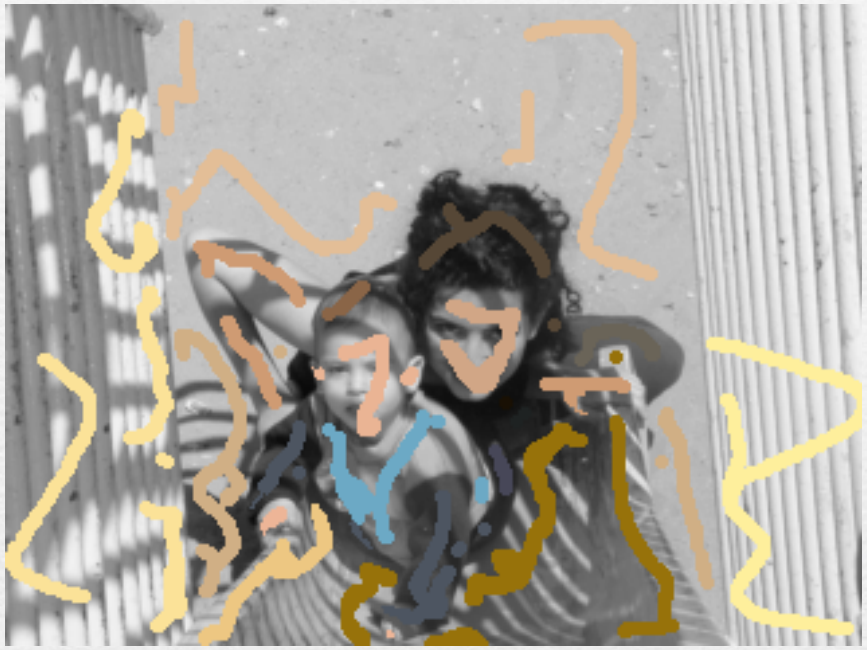
Affinity Functions in Space-Time

$$w_{rs} \propto e^{-(Y(r)-Y(s))^2 / 2\sigma_r^2}$$

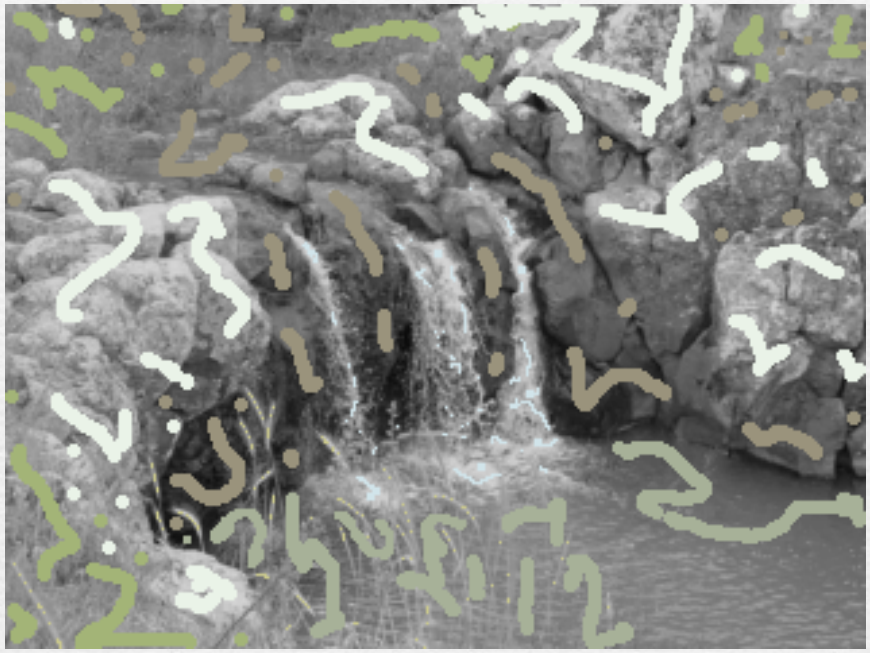


- Input: black and white image Y
- Non-homogenous least square energy on U, V
  - weight depends on pixel similarity
- User specifies U,V at stroke locations (boundary conditions
- Big linear systen

Results











Progressively improving a colorization. The artist begins with the scribbles shown in (a1), which yield the result in (a2). Note that
the table cloth gets the same pink color as the girl's dress. Also, some color is bleeding from the cyan pacier onto the wall behind it. By
adding color scribbles on the table cloth and on the wall (b1) these problems are eliminated (b2). Next, the artist decides to change the color
of the beads by sprinkling a few red pixels (c1), yielding the nal result (c2). Note that it was not necessary to mark each and every bead.

Colorization Challenges



Recoloring





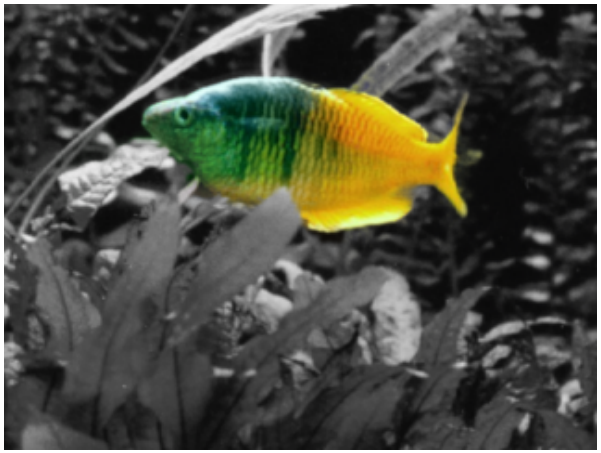


Affinity between pixels – based on intensity AND color similarities.

Slide courtesy of Anat Levin

Recoloring





Slide courtesy of Anat Levin

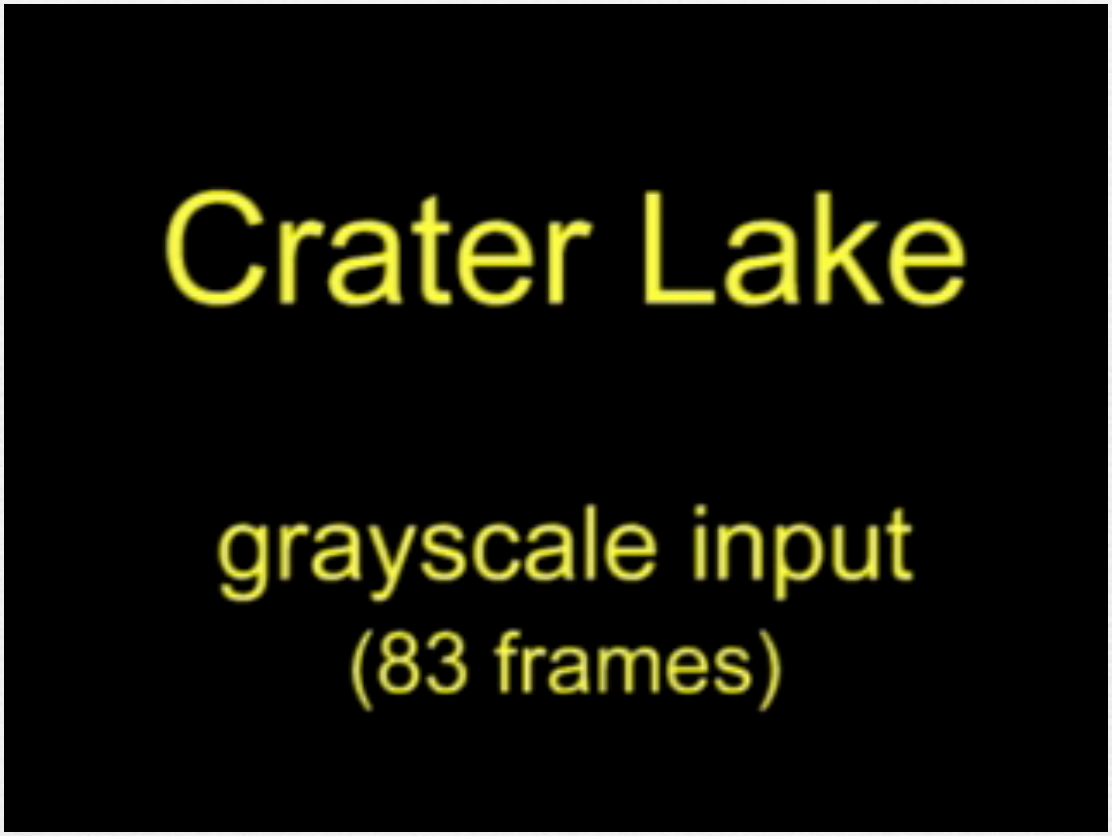
Recoloring





Slide courtesy of Anat Levin

- N(s) now takes motion into account (optical flow)



- N(s) now takes motion into account (optical flow)



Colorizing Video





13 out of 92 frames



Slide courtesy of Anat Levin

Matting as Colorization









Red channel<->matte

Slide courtesy of Anat Levin

Matting as Colorization







Slide courtesy of Anat Levin

LOCAL EDITS

Interactive Local Adjustment of Tonal Values

- Lischinski, Farbman, Uyttendaele, Szeliski
- http://www.cs.huji.ac.il/~danix/itm/
- 

play youtube video

---

http://www.youtube.com/watch?v=CoRxOFxxCOo&feature=youtube_gdata

Interactive Local Adjustment of Tonal Values

- User specifies tone/color manipulations at stroke location
- Interpolated with non-homogenous least squares
  - respects strong edges







10.1.1.89.3828.pdf
4 / 8
124%
Find

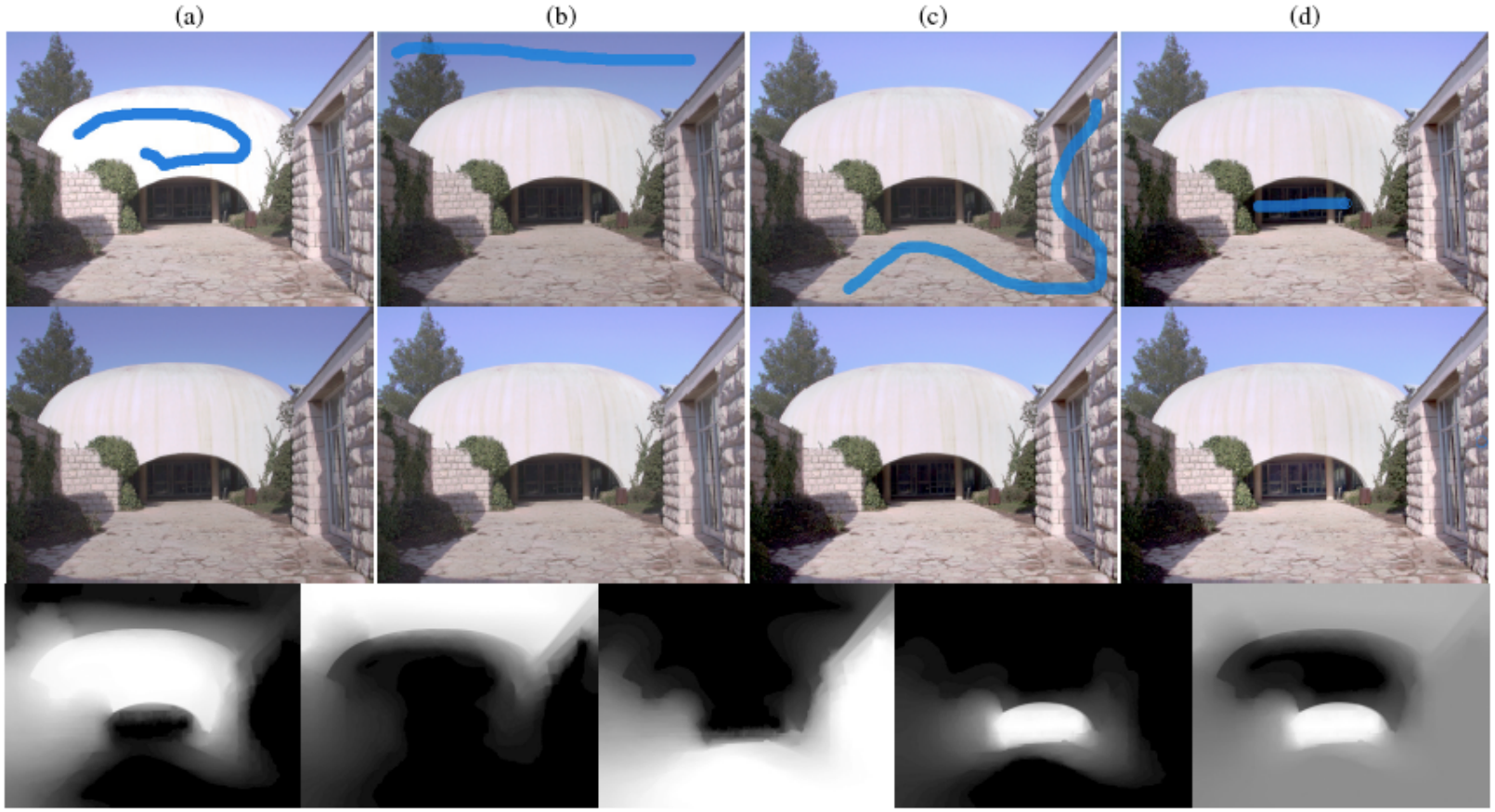


Figure 3: Step-by-step demonstration of our workflow. Top row: strokes drawn by the user; second row: the result after adjusting one or more parameters for the corresponding stroke. We refer the reader to the electronic version of the paper in order to better appreciate the differences between these images. (a) The user draws a stroke across the overexposed dome, and decreases the exposure to make some texture visible. Since this is the first and only constraint so far, the entire image is darkened. (b) The user draws a stroke across the sky and increases the exposure. The dome appears unaffected. (c) The next stroke is drawn across the foreground, followed by a slight adjustment in the exposure and an increase in the contrast. (d) Finally, the exposure and contrast are increased under the entrance arch. At any stage the user may select one of the previous strokes and modify the parameter values associated with the stroke, or delete the stroke altogether. The bottom row shows the four influence functions corresponding to the strokes in (a)–(d), and the rightmost image is the resulting exposure adjustment map. Note how strong edges in the image limit the influence of the individual strokes, and correspond to discontinuities in the final adjustment map.