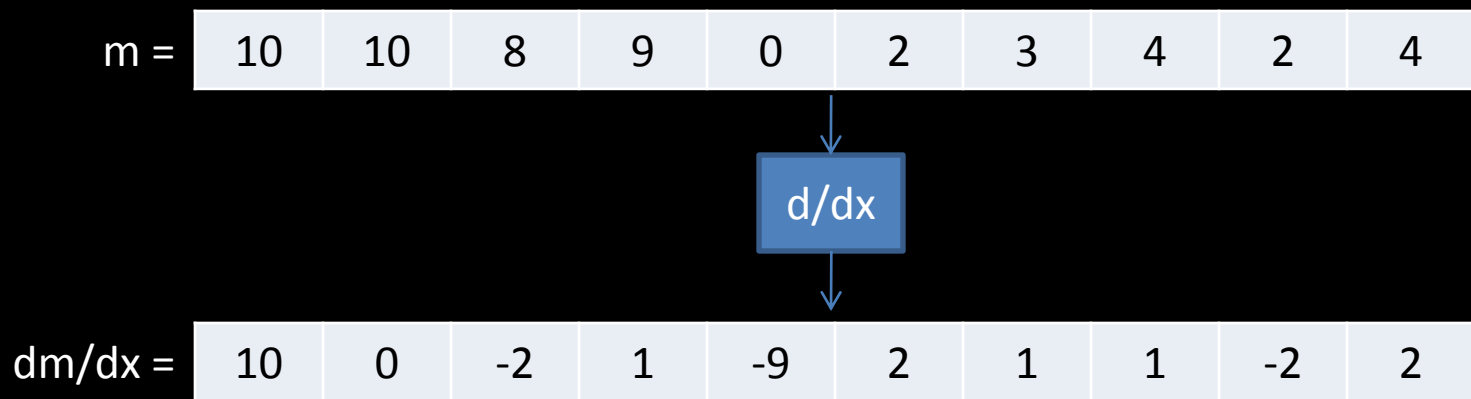


CS448f: Image Processing For Photography and Vision

The Gradient Domain

Image Gradients

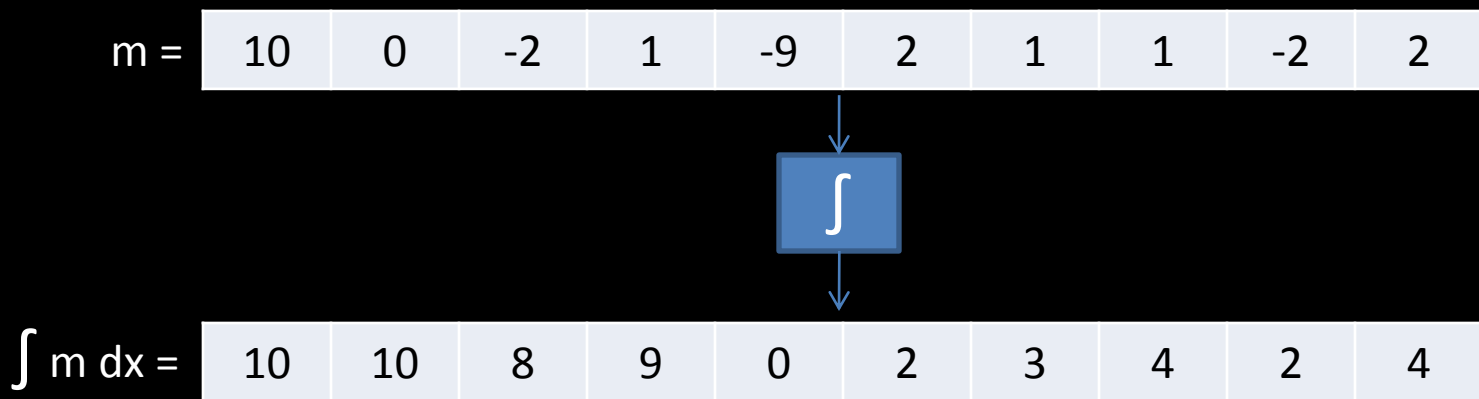
- We can approximate the derivatives of an image using differences



- Equivalent to convolution by $[-1 \ 1]$
- Note the zero boundary condition

Image Derivatives

- We can get back to the image by integrating
 - like an integral image



- Differentiating throws away constant terms
 - The boundary condition allowed us to recover it

Image Derivatives

- Can think of it as an extreme coarse/fine decomposition
 - coarse = boundary term
 - fine = gradients

In 2D

- Gradient in X = convolution by $[-1 \ 1]$
- Gradient in Y = convolution by $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$
- If we take both, we have $2n$ values to represent n pixels
 - Must be redundant!

Redundancy

- $d (dm/dx) / dy = d (dm/dy) / dx$
- Y derivative of X gradient = X derivative of Y gradient

Gradient Domain Editing

- Gradient domain techniques
 - Take image gradients
 - Mess with them
 - Try to put the image back together
- After you've messed with the gradients, the constraint on the previous slide doesn't necessarily hold anymore.

The Poisson Solve

- Convolution by $[-1 \ 1]$ is a linear operator: D_x
- Taking the Y gradient is some operator: D_y
- We have desired gradient images g_x and g_y
- We want to find the image that best produces them
- Solve for an image m such that:

$$\begin{bmatrix} D_x \\ D_y \end{bmatrix} m = \begin{bmatrix} g_x \\ g_y \end{bmatrix}$$

The Poisson Solve

- How? Using Least Squares:

$$\begin{bmatrix} D_x^T & D_y^T \end{bmatrix} \begin{bmatrix} D_x \\ D_y \end{bmatrix} m = \begin{bmatrix} D_x^T & D_y^T \end{bmatrix} \begin{bmatrix} g_x \\ g_y \end{bmatrix}$$

$$(D_x^T D_x + D_y^T D_y) m = D_x^T g_x + D_y^T g_y$$

- This is a Poisson Equation

The Poisson Solve

- $D_x =$ Convolution by $[-1 \ 1]$
- $D_x^T =$ Convolution by $[1 \ -1]$
- The product = Convolution by $[1 \ -2 \ 1]$
 - Approximate second derivative
- $D_x^T D_x + D_y^T D_y =$ convolution by

$$\begin{array}{ccc} & & 1 \\ & & | \\ 1 & -4 & 1 \\ & & | \\ & & 1 \end{array}$$

The Poisson Solve

- We need to invert:

$$(D_x^T D_x + D_y^T D_y)m = D_x^T g_x + D_y^T g_y$$

- How big is the matrix?
- Anyone know any methods for inverting large sparse matrices?

Solving Large Linear Systems

- $A = D_x^T D_x + D_y^T D_y$
- $b = D_x^T g_x + D_y^T g_y$
- We need to solve $Ax = b$

1) Gradient Descent

- x = some initial estimate
- For (lots of iterations):

$$r = b - Ax$$

$$e = r^T r$$

$$\alpha = e / r^T A r$$

$$x += \alpha r$$

2) Conjugate Gradient Descent

- x = some initial estimate
- $d = r = Ax - b$
- $e_{\text{new}} = r^T r$
- For (fewer iterations):
 - $\alpha = e_{\text{new}} / d^T A d$
 - $x += \alpha d$
 - $r = b - Ax$
 - $e_{\text{old}} = e_{\text{new}}$
 - $e_{\text{new}} = r^T r$
 - $d = r + d e_{\text{new}} / e_{\text{old}}$
- (See An Introduction to the Conjugate Gradient Method Without the Agonizing Pain)

3) Coarse to Fine Conj. Grad. Desc.

- Downsample the target gradients
- Solve for a small solution
- Upsample the solution
- Use that as the initial estimate for a new conj. grad. descent
- Not too many iterations required at each level
- This is what ImageStack does in -poisson

4) FFT Method

- We're trying to undo a convolution
- Convolutions are multiplications in Fourier space
- Therefore, go to Fourier space and divide

Applications

- How might we like to mess with the gradients?
- Let's try some stuff

Applications

- Poisson Image Editing
 - Perez 2003
- GradientShop
 - Bhat 2009
- Gradient Domain HDR Compression
 - Fattal et al 2002
- Efficient Gradient-Domain Compositing Using Quadtrees
 - Agarwala 2007
- Coordinates for Instant Image Cloning
 - Farbman et al. 2009