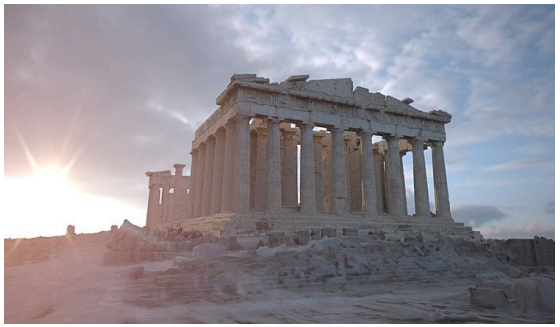
Computer Vision

CSE399b

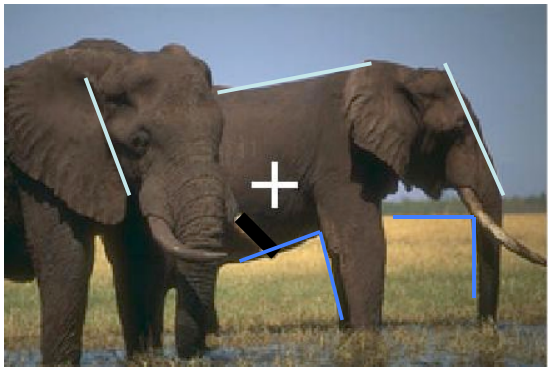


Jianbo Shi

Spring 2008

Paul Debevec
http://www.debevec.org/Parthenon/Images/

Marr's Primary Sketch **Go symbolic**



Edge **Bar** **terminators**

Canny edge Detection

1) Compute image gradient $J_x = I \otimes (\frac{\delta}{\delta x} \otimes G)$

2) Compute edge gradient magnitude:

$$\nabla J = (J_x, J_y) = \left(\frac{\partial J}{\partial x}, \frac{\partial J}{\partial y}\right) \text{ is the Gradient}$$

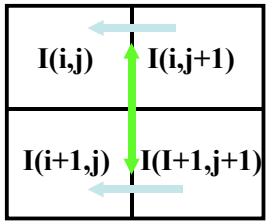
$$\|\nabla J\| = \sqrt{J_x^2 + J_y^2}$$

3) Detect local edge maximum

1) Compute Image Gradient

the first order derivative of Image I in x, and in y direction

Compute gradient: first order derivatives



$$\frac{\delta}{\delta x} = \begin{array}{|c|c|}\hline 1 & -1 \\ \hline \end{array}$$

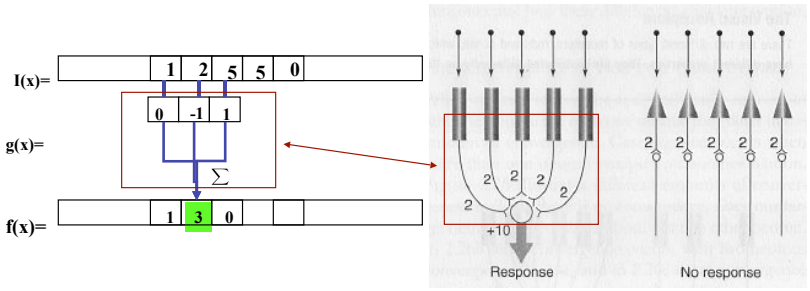
$$\mathbf{S} = \begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline \end{array}$$

$$\frac{\delta I}{\delta x}(i,j) = \frac{1}{2}((I(i,j+1) - I(i,j)) + (I(i+1,j+1) - I(i+1,j)))$$

$$= (I \otimes \frac{\delta}{\delta x}) \otimes S$$

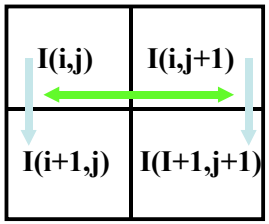
Let I be an Signal(image), Convolution kernel f,

$$g[m,n] = I \otimes f = \sum_{k,l} I[m-k, n-l] f[k,l]$$



This leads to parallel computation

Compute gradient: first order derivatives



$$\frac{\delta}{\delta y} = \begin{array}{|c|}\hline 1 \\ \hline -1 \\ \hline \end{array}$$

$$\frac{\delta I}{\delta y}(i,j) = \frac{1}{2}((I(i+1,j) - I(i,j)) + (I(i+1,j+1) - I(i,j+1)))$$

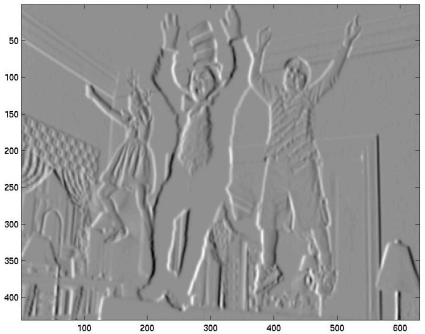
$$I_y = (I \otimes \frac{\delta}{\delta y}) \otimes S'$$

Use matlab ``conv2'' function

s = [1 ; 1];
dx = [1, -1];

gx = conv2(conv2(I,dx,'same'),s,'same');

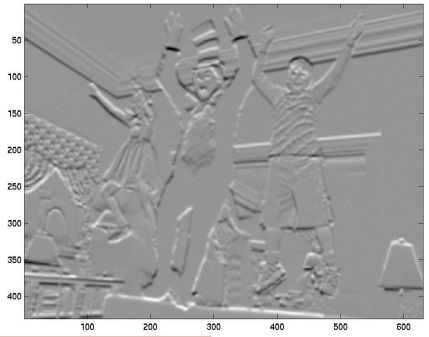
gx(I)



$$I_x = = (I \otimes \frac{\delta}{\delta x}) \otimes S$$

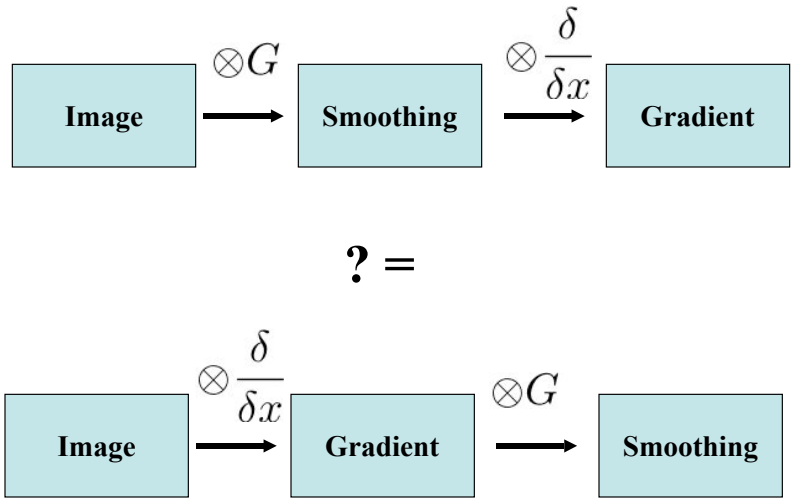
```
s = [1 ; 1];
dx = [1, -1];
dy = [1;-1];

gx = conv2(conv2(I,dx,'same'),s,'same');
gy = conv2(conv2(I,dy,'same'),s','same');
```

gy(I)



$$I_y = (I \otimes \frac{\delta}{\delta y}) \otimes S'$$

Can we switch the image smoothing and image differencing,

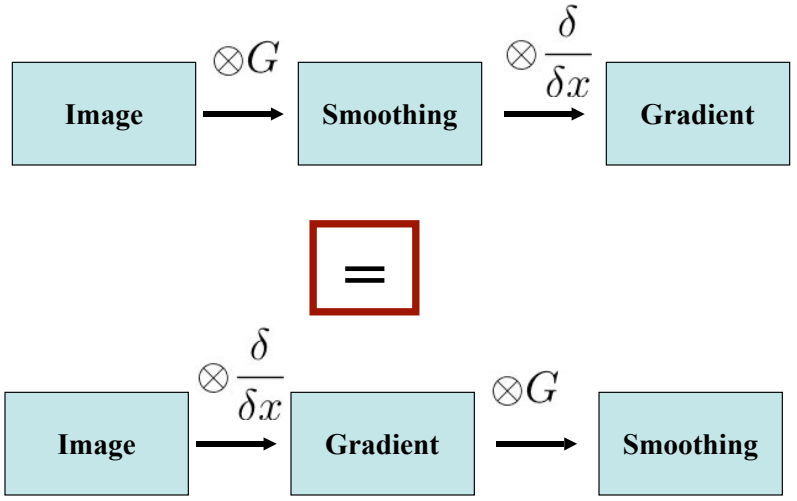


Convolution is communitative:

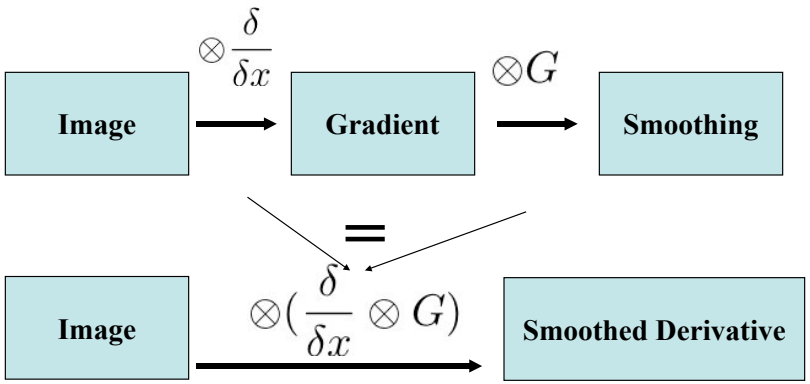
$$I \otimes g = g \otimes I$$

Convolution is associative:

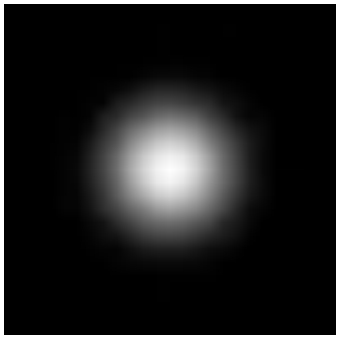
$$(I \otimes f_1) \otimes f_2 = I \otimes (f_1 \otimes f_2)$$



We can simplify even more:



Recall, a smooth filter of Gaussian is:



$$e^{-\frac{x^2+y^2}{2\sigma^2}}$$

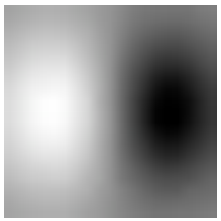
More information:

http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

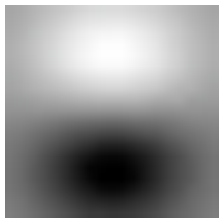
Smoothed derivative filter

$$\frac{\delta}{\delta x} \otimes G = \frac{\delta G}{\delta x} \longrightarrow \frac{\delta G}{\delta x} = -\frac{2x}{\sigma_x^2} G(x,y)$$

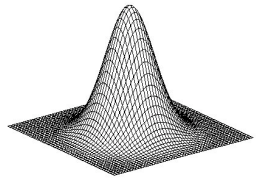
Gx



Gy

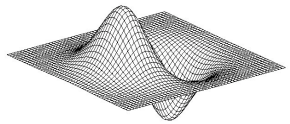


2D edge detection filters



Gaussian

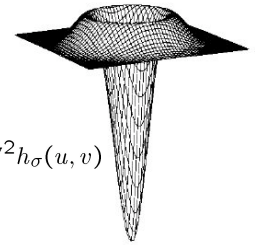
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

Laplacian of Gaussian



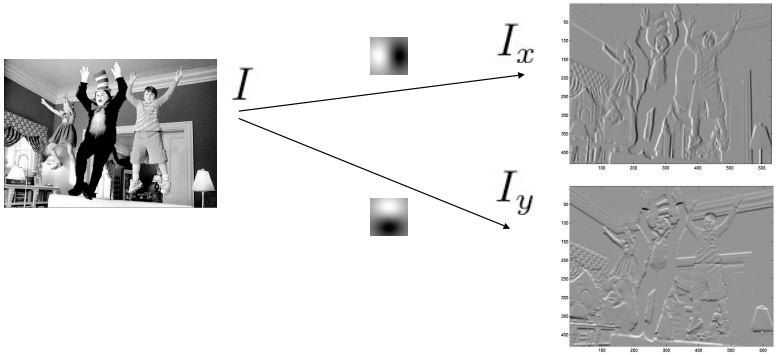
$$\nabla^2 h_\sigma(u,v)$$

∇^2 • is the **Laplacian** operator:

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

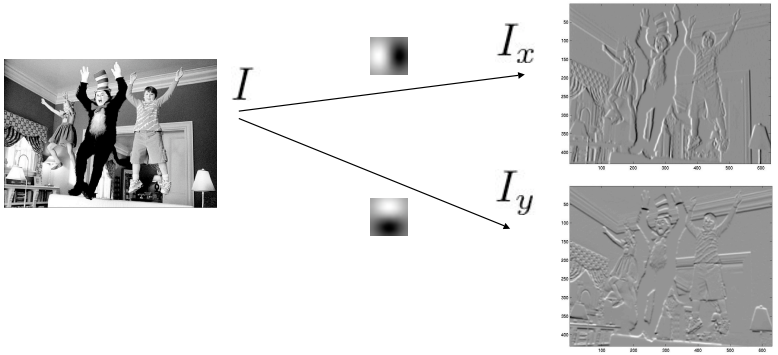
Edge Detection, Step 1, Filter out noise and compute derivative:

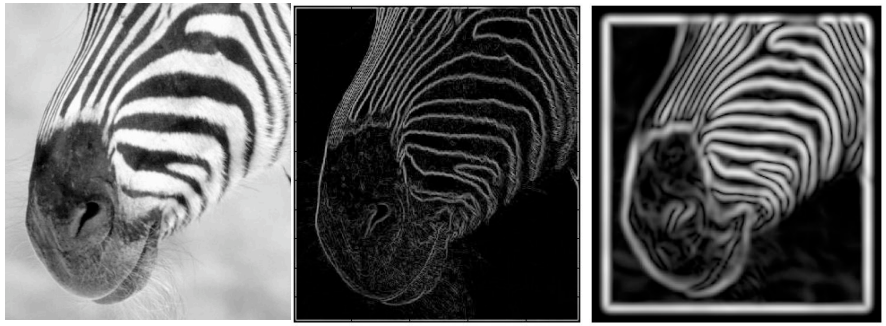
Image $\otimes(\frac{\delta}{\delta x} \otimes G)$ **Smoothed Derivative**



Edge Detection, Step 1, Filter out noise and compute derivative:

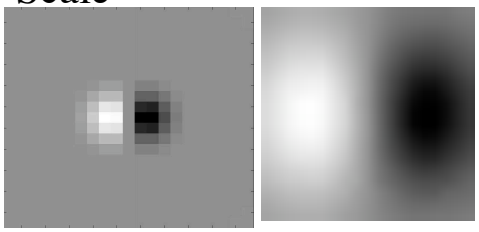
In matlab:
>> [dx,dy] = gradient(G); % G is a 2D gaussain
>> Ix = conv2(I,dx,'same'); Iy = conv2(I,dy,'same');





Scale

- Smoothing
- Eliminates noise edges.
- Makes edges smoother.
- Removes fine detail.

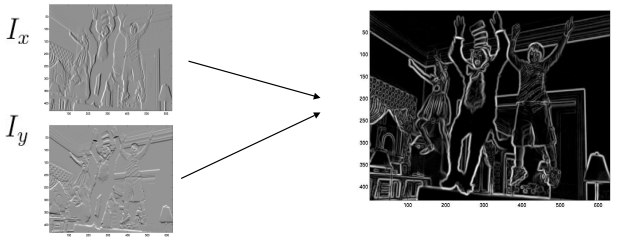


(Forsyth & Ponce)

Edge Detection: Step 2 Compute the magnitude of the gradient

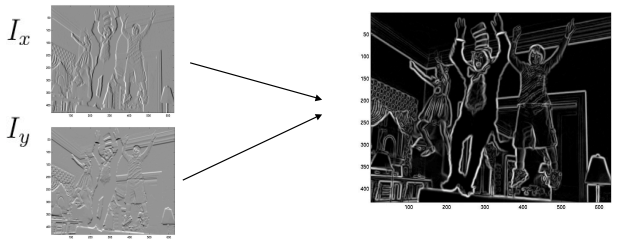
$$\nabla J = (J_x, J_y) = \left(\frac{\partial J}{\partial x}, \frac{\partial J}{\partial y}\right) \text{ is the Gradient}$$

$$\|\nabla J\| = \sqrt{J_x^2 + J_y^2}$$

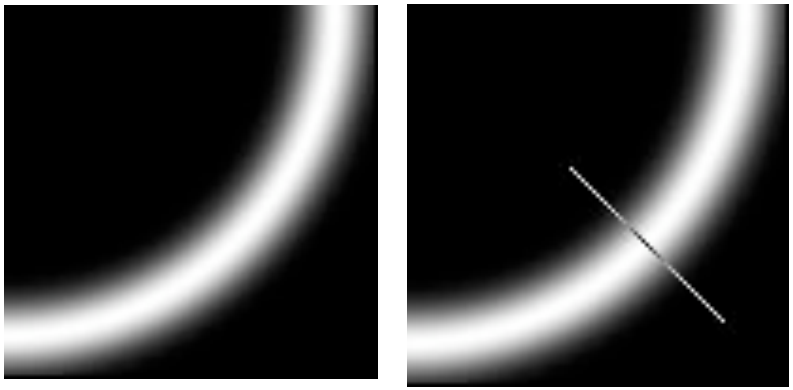


Edge Detection: Step 2
Compute the magnitude of the gradient

In Matlab:
>> Im = sqrt(Ix.*Ix + Iy.*Iy);



we know roughly where are the edges, but we need their precise location.

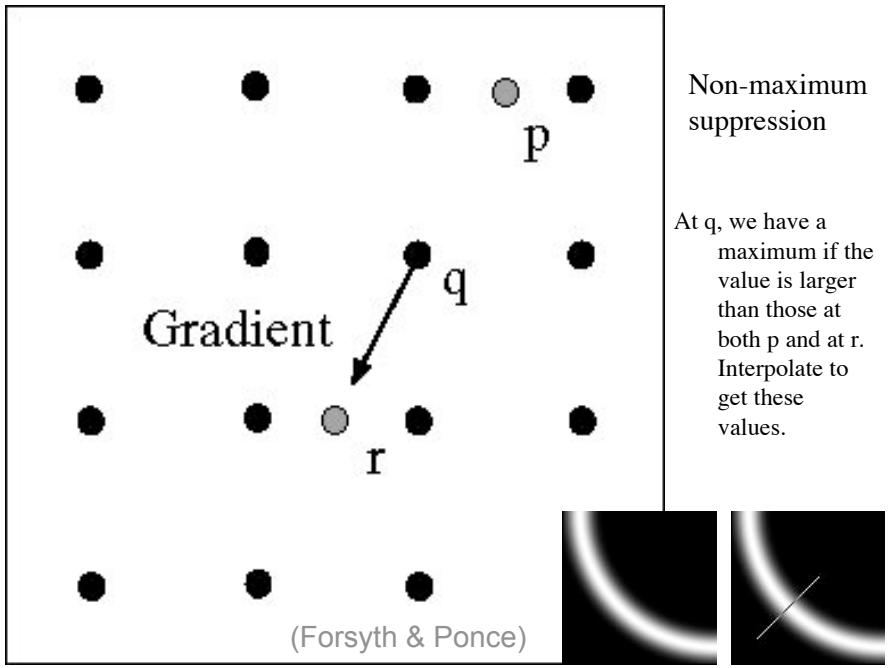


We wish to mark points along the curve where the magnitude is biggest. We can do this by looking for a maximum along a slice normal to the curve (non-maximum suppression). These points should form a curve. There are then two algorithmic issues: at which point is the maximum, and where is the next one?

(Forsyth & Ponce)

Finding the Peak

- 1) The gradient magnitude is large along thick trail; how do we identify the significant points?
- 2) How do we link the relevant points up into curves?



Non-maximum suppression

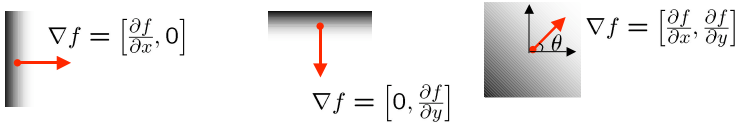
At q, we have a maximum if the value is larger than those at both p and at r. Interpolate to get these values.

Finding the orientation of the edge

- The gradient of an image:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

- The gradient points in the direction of most rapid change in intensity

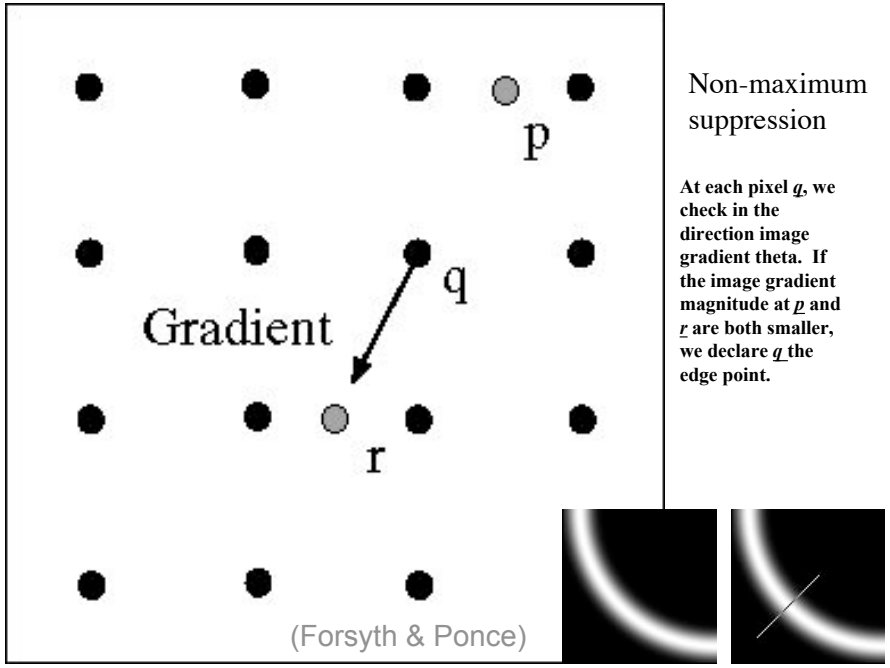


- The image gradient direction is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

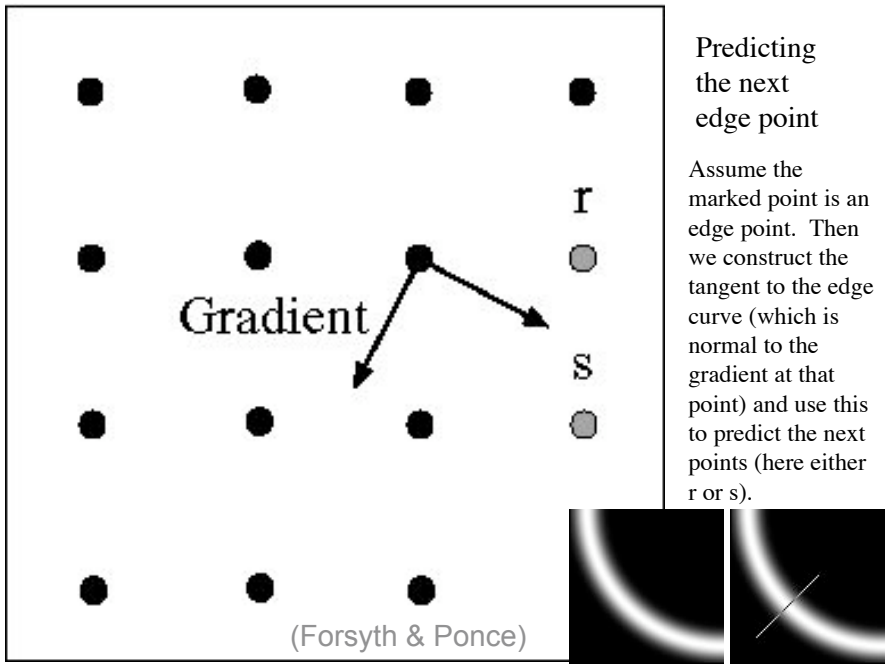
  – how does this relate to the direction of the edge?

$$\theta_{edge} = tan^{-1}(-\frac{\delta f}{\delta x} / \frac{\delta f}{\delta y})$$



Non-maximum suppression

At each pixel *q*, we check in the direction image gradient theta. If the image gradient magnitude at *p* and *r* are both smaller, we declare *q* the edge point.

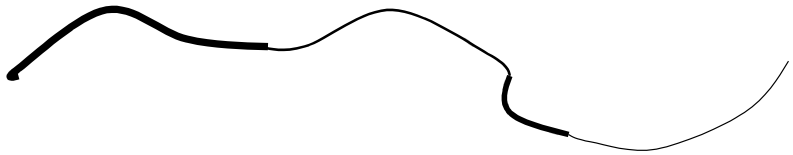


Predicting the next edge point

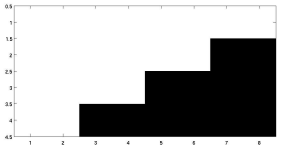
Assume the marked point is an edge point. Then we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points (here either r or s).

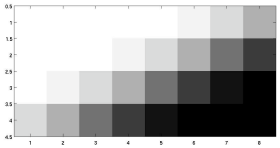
Hysteresis

- Check that maximum value of gradient value is sufficiently large
  - drop-outs? use **hysteresis**
    - use a high threshold to start edge curves and a low threshold to continue them.



```
%% create an image
A = [ ones(1,10); [ones(1,9),zeros(1,1)]; [ones(1,7),zeros(1,3)];…
  [ones(1,5),zeros(1,5)]; [ones(1,3),zeros(1,7)];[ones(1,1),zeros(1,9)]];

 %% smoothing kernel
  smoothk = [0.25, 0.5, 0.25];

 %% image I is the center portion of the image A, remove the boundary
  I= A(2:end-1,2:end-1);

 %% compute smoothed version of the image AA, low pass
 AA = conv2(A,smoothk,'same');AA = conv2(AA,smoothk','same');

 %% take out the center portion of the smoothed image AA,
 J = AA(2:end-1,2:end-1);

 %% compare the two images
 figure(1);clf;imagesc(I);colormap(gray);axis image;
 figure(2);clf;imagesc(J);colormap(gray);axis image;
```





Canny edge Detection

Filter image by derivatives of Gaussian $J_x = I \otimes (\frac{\delta}{\delta x} \otimes G)$

Compute filter Magnitude: $\nabla J = (J_x, J_y) = \left(\frac{\partial J}{\partial x}, \frac{\partial J}{\partial y}\right)$ is the Gradient

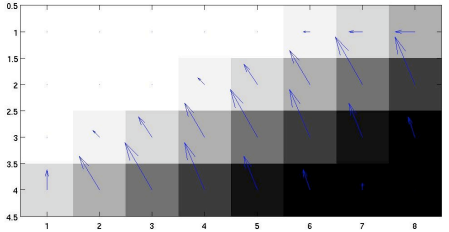
$\|\nabla J\| = \sqrt{J_x^2 + J_y^2}$

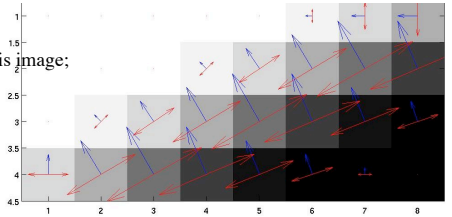
Compute edge orientation: $\theta = atan(J_x, J_y);$

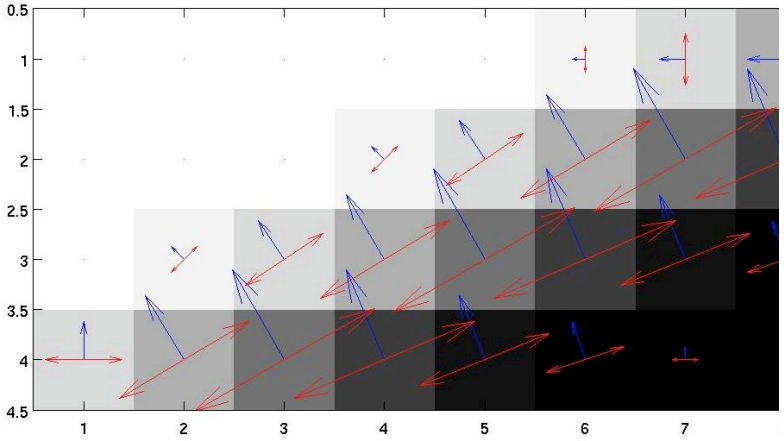
Detect local maximum,

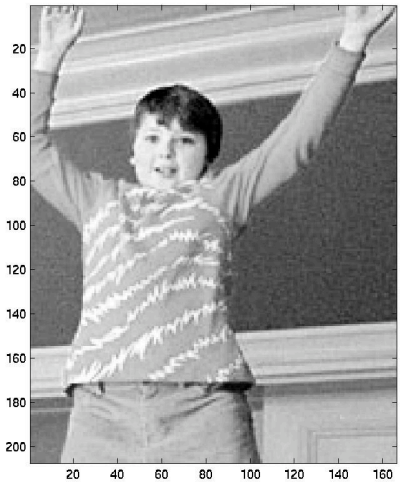
Edge linking

```
%% define image gradient operator
dy = [1;-1];
dx = [1,-1];

%% compute image gradient in x and y
AA_y = conv2(AA,dy,'same');
AA_x = conv2(AA,dx,'same');

Jy = AA_y(1:end-2,2:end-1);
Jy(1,:) = 0;

Jx = AA_x(2:end-1,1:end-2);
Jx(:,1) = 0;

%% display the image gradient flow
figure(3);clf;imagesc(J);colormap(gray);axis image;
hold on;
quiver(Jx,Jy);
quiver(-Jy,Jx,'r');
quiver(Jy,-Jx,'r');
```

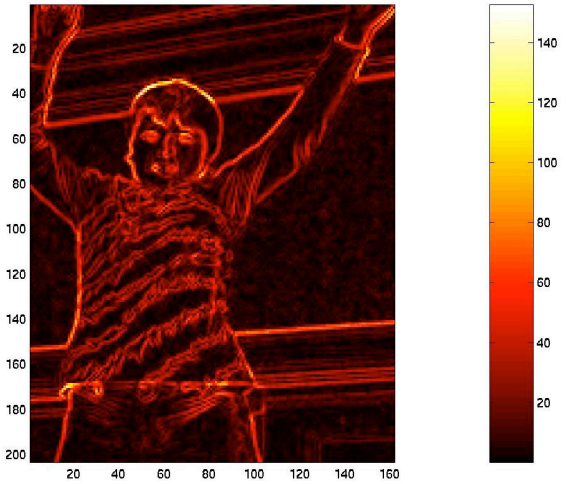








[gx,gy] = gradient(J);
mag = sqrt(gx.*gx+gy.*gy); imagesc(mag);colorbar



[gx,gy] = gradient(J);
th = atan2(gy,gx); % or you can use:[th,mag] = cart2pol(gx,gy);
imagesc(th.*(mag>20));colormap(hsv); colorbar

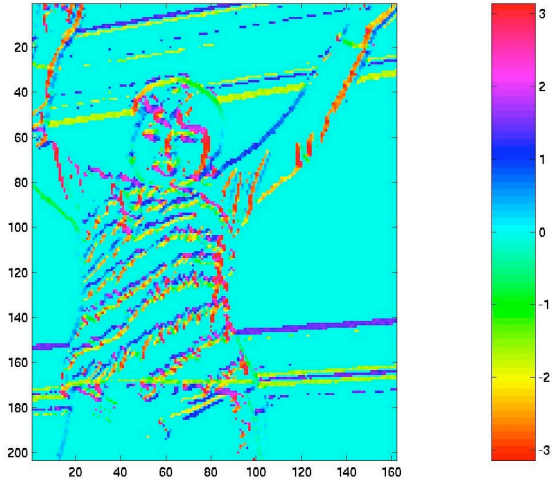
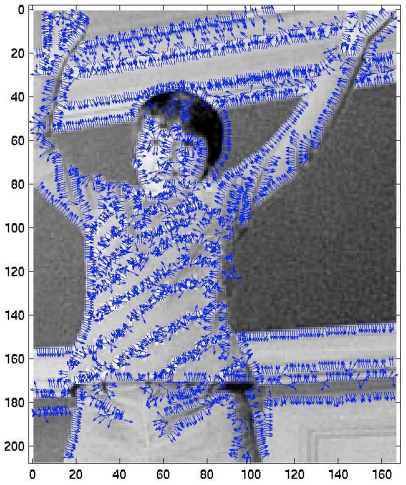
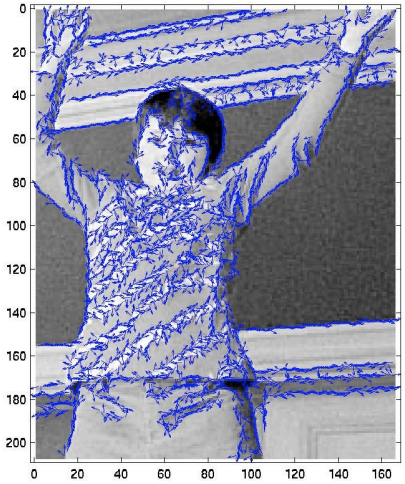
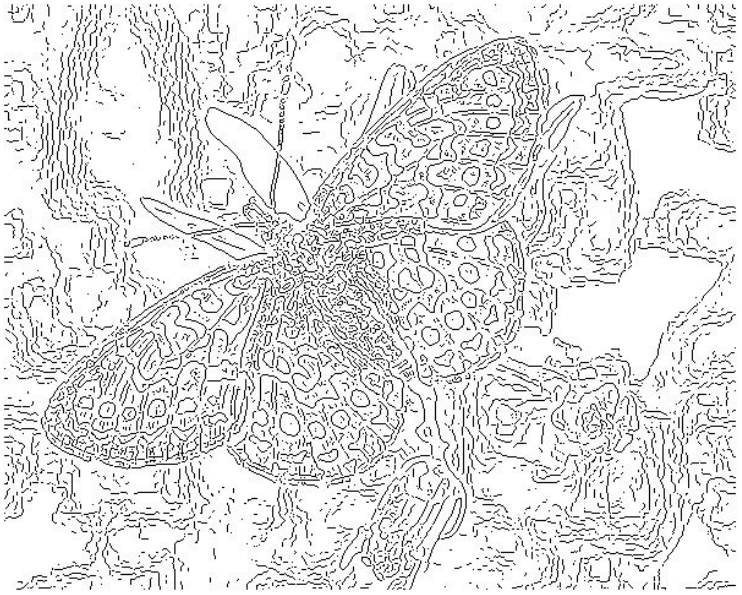


image gradient direction:

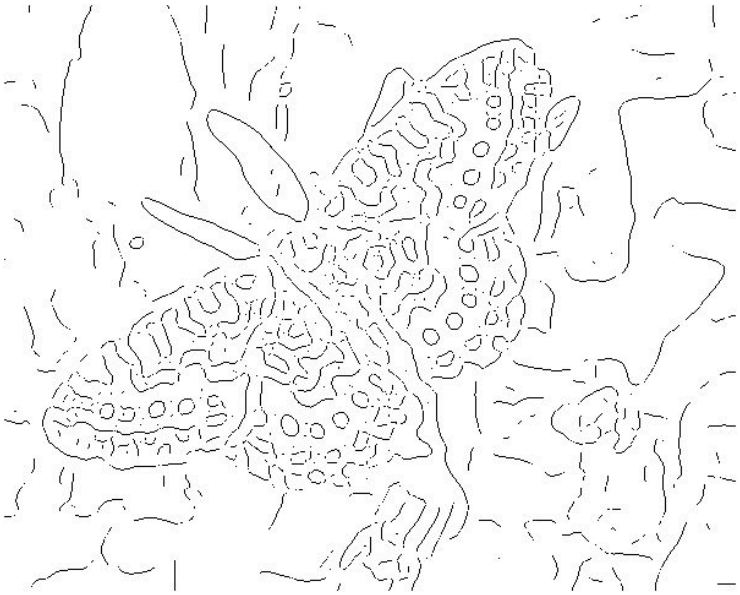


Edge orientation direction:



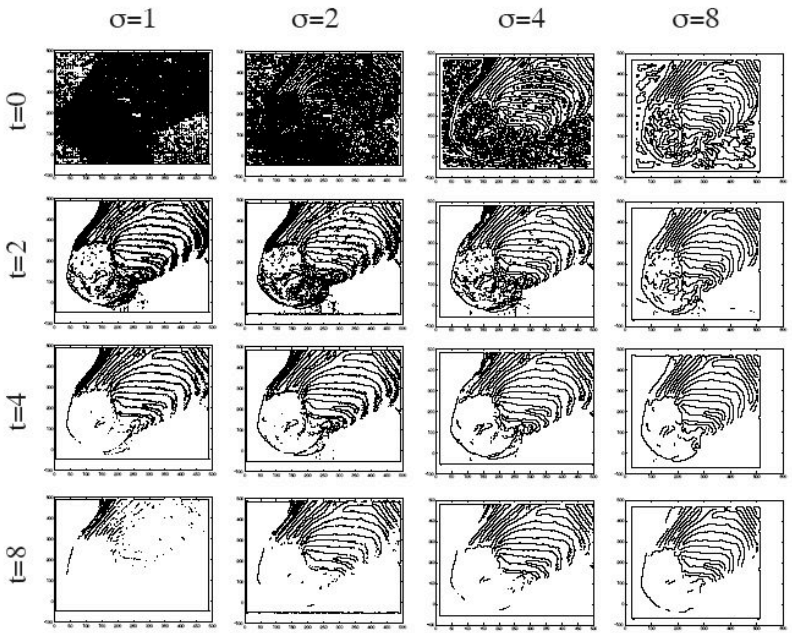


fine scale
high
threshold



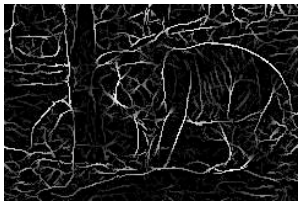
coarse scale, high threshold

coarse scale low threshold



Why is image segmentation so difficult ?

⇒ Image

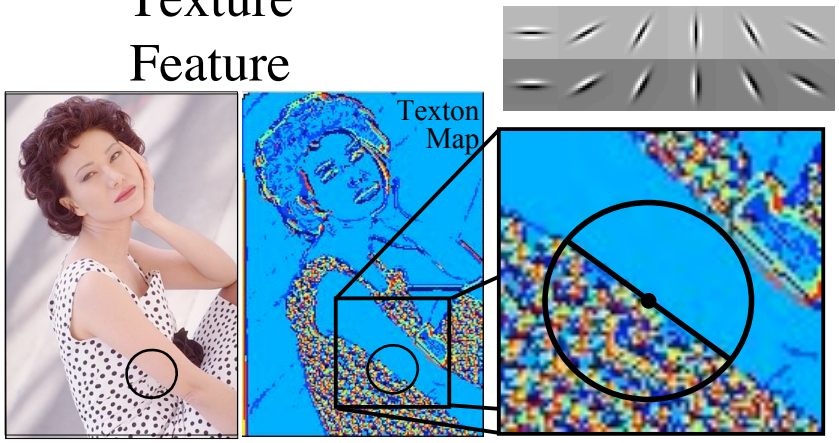


⇒ Edge detector

⇒ Human segmentation

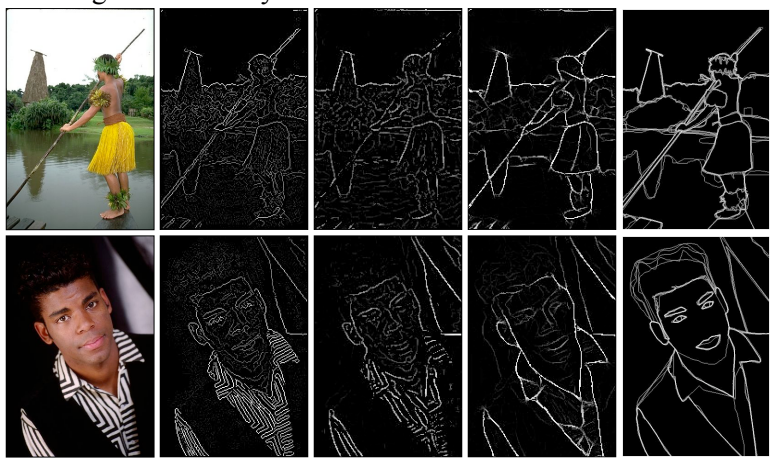
Texture edge detections

Texture Feature

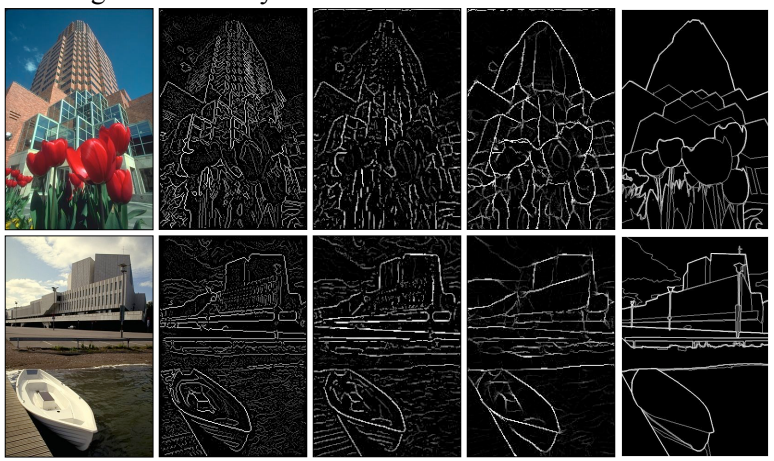


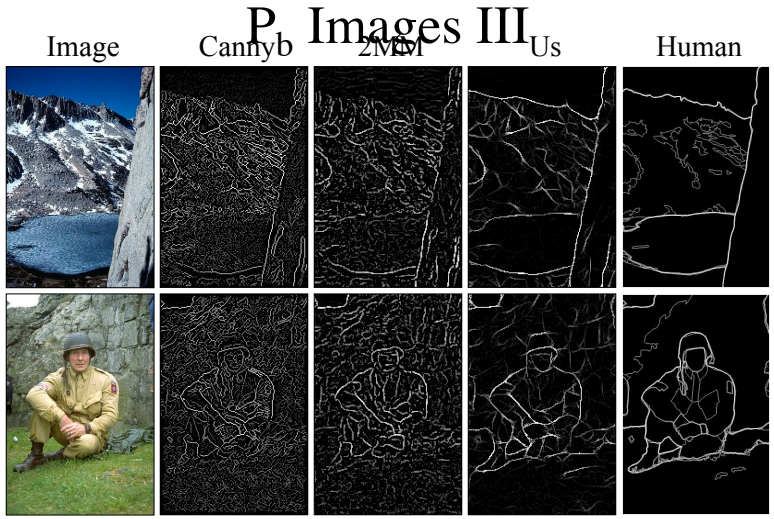
- Texture Gradient TG(x,y,r,θ)
  - χ^2 difference of texton histograms
  - Textons are vector-quantized filter outputs

P_b Images I



P_b Images II





P_b Images III

Canny edge Detection

1) Compute image gradient $J_x = I \otimes (\frac{\delta}{\delta x} \otimes G)$

2) Compute edge gradient magnitude:

$\nabla J = (J_x, J_y) = \left(\frac{\partial J}{\partial x}, \frac{\partial J}{\partial y}\right)$ is the Gradient

$\|\nabla J\| = \sqrt{J_x^2 + J_y^2}$

3) Compute edge orientation: $\theta_{edge} = tan^{-1}(-J_x/J_y)$

4) Detect local edge maximum,

5) Edge linking