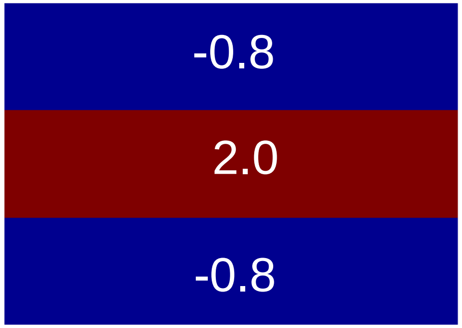
Edge Detection

Code

```
Jb = rgb2gray(J);

imagesc(Jb);axis image; colormap(gray);

bw = edge(Jb,'canny');
```

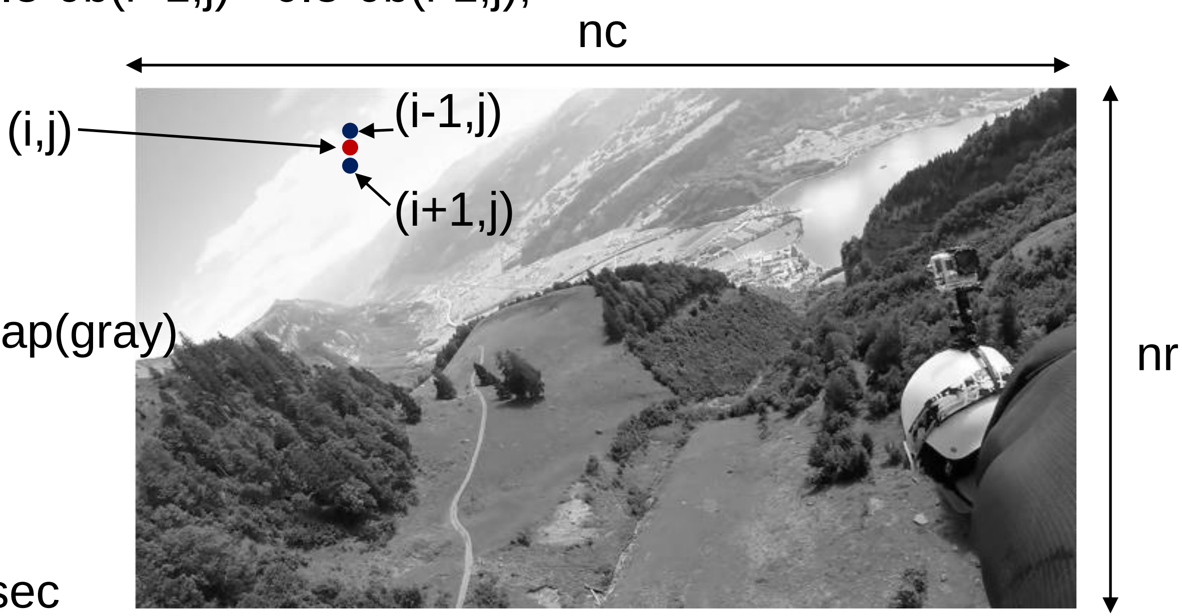
Numerical Image Filtering

Filter



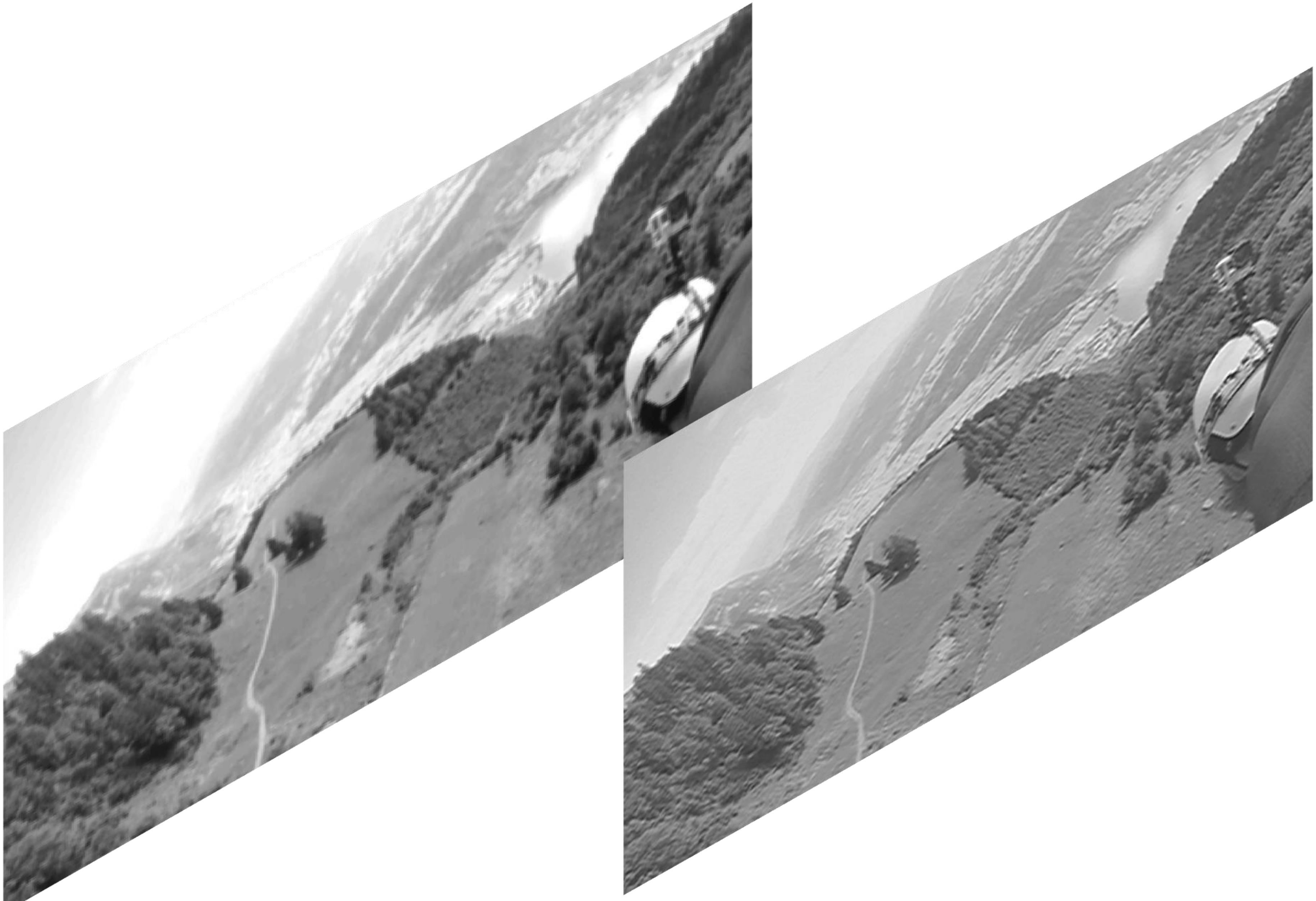
Looping through all pixels

```
[nr,nc] = size(Jb);
J_out = zeros(nr,nc);
for i=1:nr,
    for j=1:nc;
        if (i<nr) && (i>1),
            J_out(i,j) = 2*Jb(i,j) - 0.8*Jb(i+1,j) - 0.8*Jb(i-1,j);
        else
            J_out(i,j) = Jb(i,j);
        end
    end
end
figure; imagesc(J_out);colormap(gray)
```



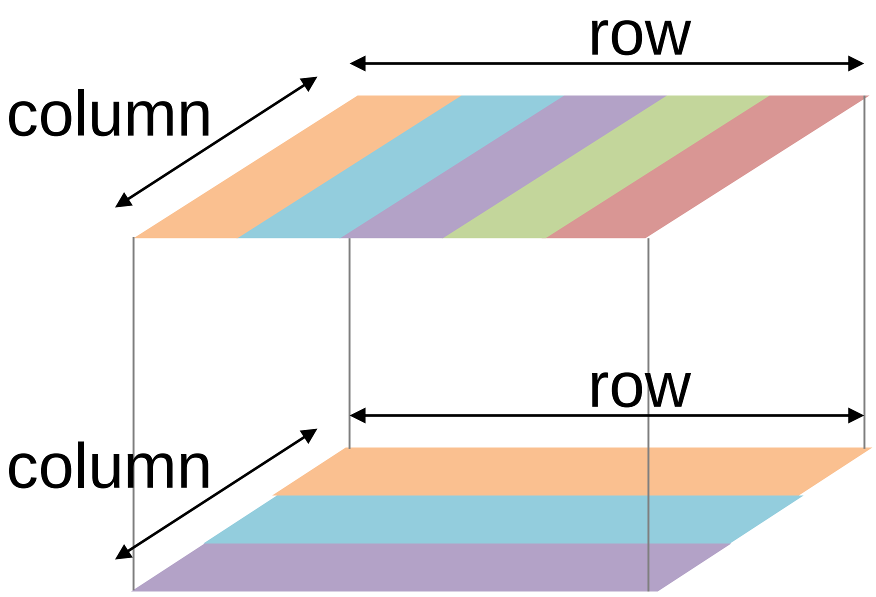
Computation time: 0.050154 sec

Numerical Image Filtering

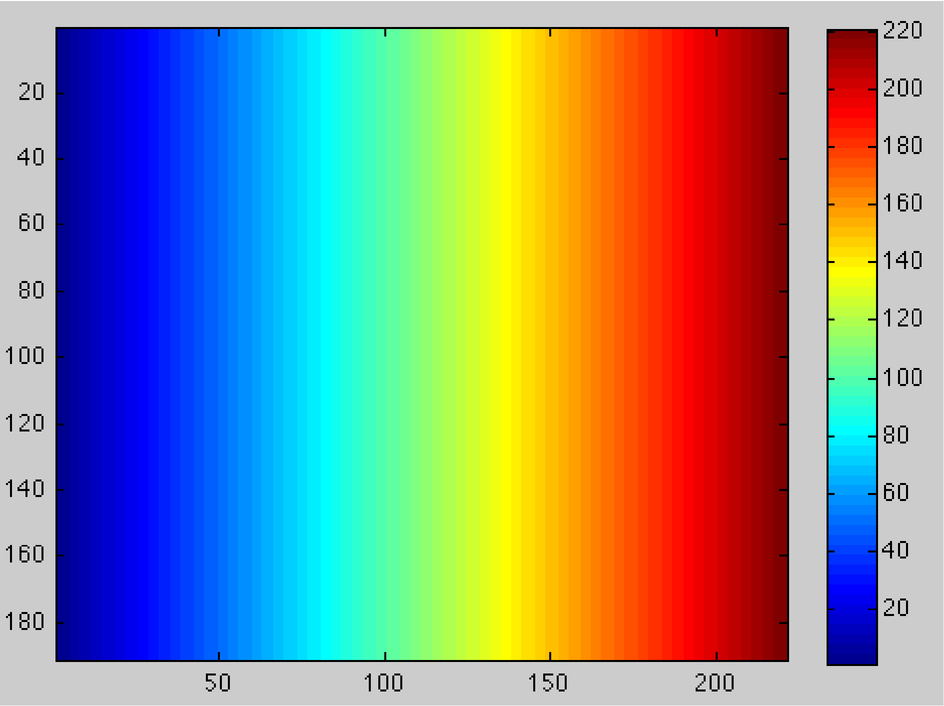


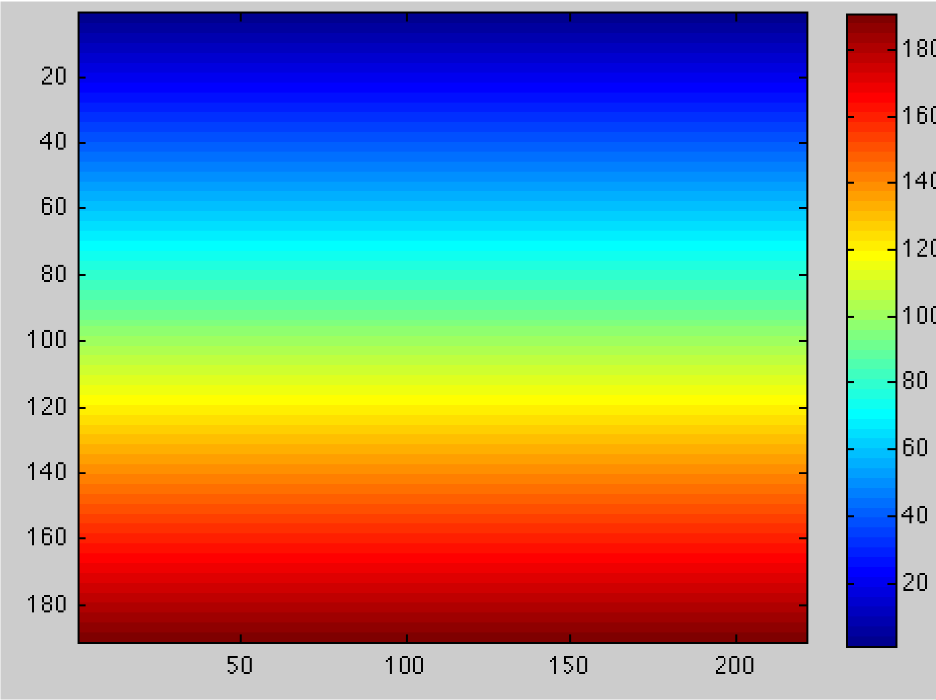
Convolution without Looping using meshgrid

```
>> [x,y] = meshgrid(1:5,1:3)

x =

     1     2     3     4     5
     1     2     3     4     5
     1     2     3     4     5


y =

     1     1     1     1     1
     2     2     2     2     2
     3     3     3     3     3
```



```
[x,y] = meshgrid(1:nc,1:nr);
figure(1); imagesc(x); axis image; colorbar;
colormap(jet);
figure(2); imagesc(y); axis image; colorbar;
colormap(jet);
```





Convolution without Looping using meshgrid

```
[x,y] = meshgrid(1:nc,1:nr);

y_up = y-1;
y_down = y+1;

y_up = min(nr,max(1,y_up));  % keep y_up index within legal range of [1,nr]
y_down = min(nr,max(1,y_down));

ind_up = sub2ind([nr,nc],y_up(:),x(:));  % create linear index
ind_down = sub2ind([nr,nc],y_down(:),x(:));

J_out = 2*Jb(:) - 0.8*Jb(ind_up) - 0.8*Jb(ind_down);
J_out = reshape(J_out, nr, nc);

figure; imagesc(J_out);colormap(gray)
```

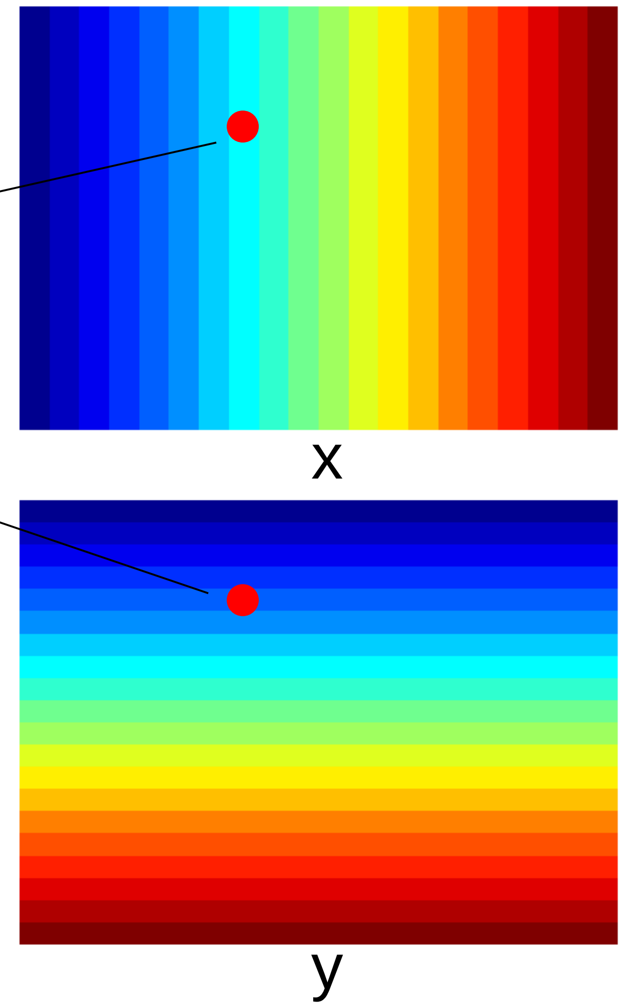
Computation time: 0.024047 sec

Convolution without Looping using meshgrid

[x,y] = meshgrid(1:nc,1:nr);

x and y are subscript indice.

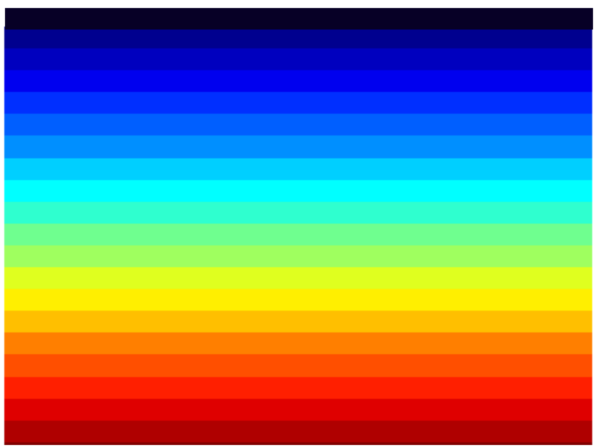
Jb_{xy}



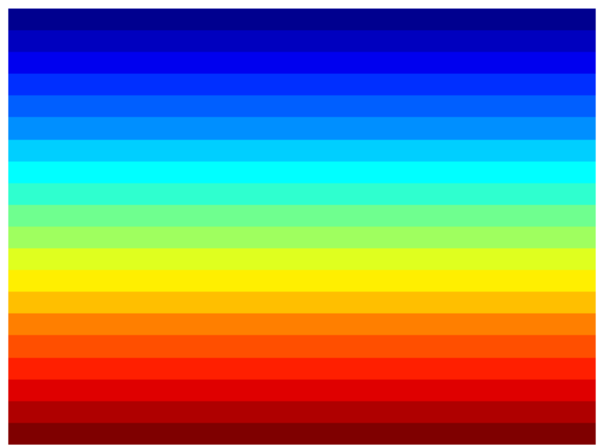
Convolution without Looping using meshgrid

```
[x,y] = meshgrid(1:nc,1:nr);

y_up = y-1;
y_down = y+1;
```

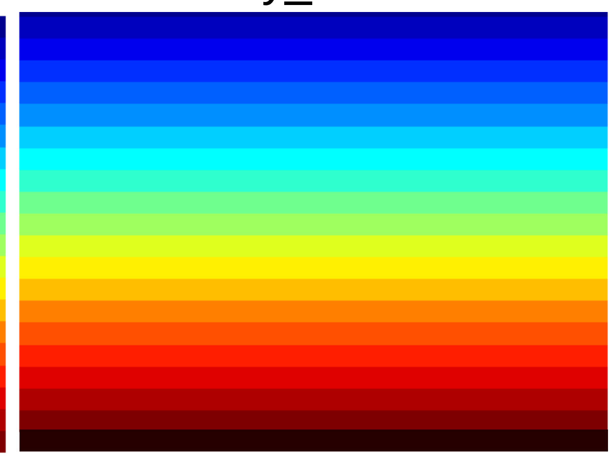
y_up



y



y_down



y_up =

0	0	0	0	0
1	1	1	1	1
2	2	2	2	2

y =

1	1	1	1	1
2	2	2	2	2
3	3	3	3	3

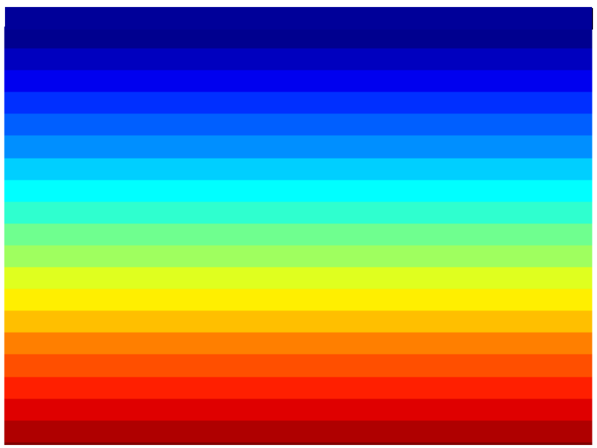
y_down =

2	2	2	2	2
3	3	3	3	3
4	4	4	4	4

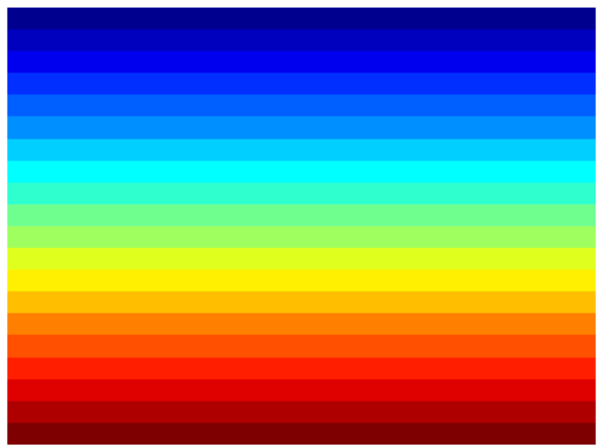
Convolution without Looping using meshgrid

```
y_up = y-1;
y_down = y+1;

y_up = min(nr,max(1,y_up));  % keep y_up index within legal range of [1,nr]
y_down = min(nr,max(1,y_down));
```

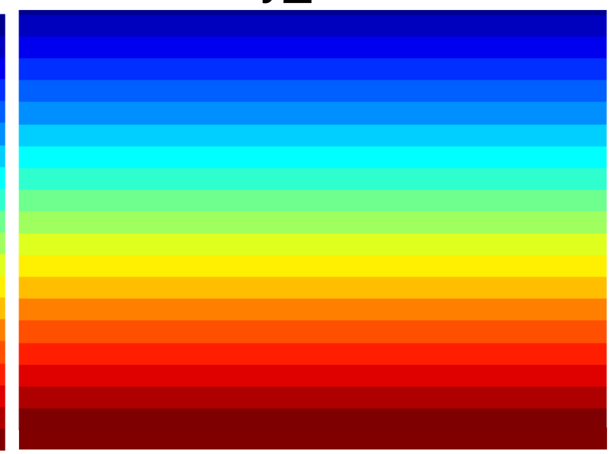
y_up



y



y_down



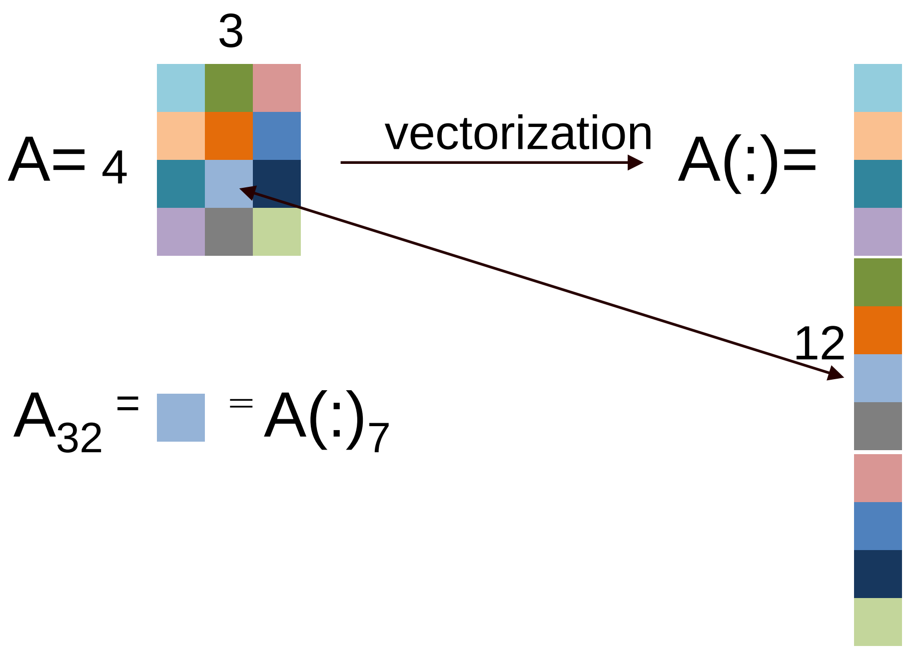
```
y_up =

     1     1     1     1     1
     1     1     1     1     1
     2     2     2     2     2
```

```
y =

     1     1     1     1     1
     2     2     2     2     2
     3     3     3     3     3
```

```
y_down =

     2     2     2     2     2
     3     3     3     3     3
     3     3     3     3     3
```

Convolution without Looping using meshgrid

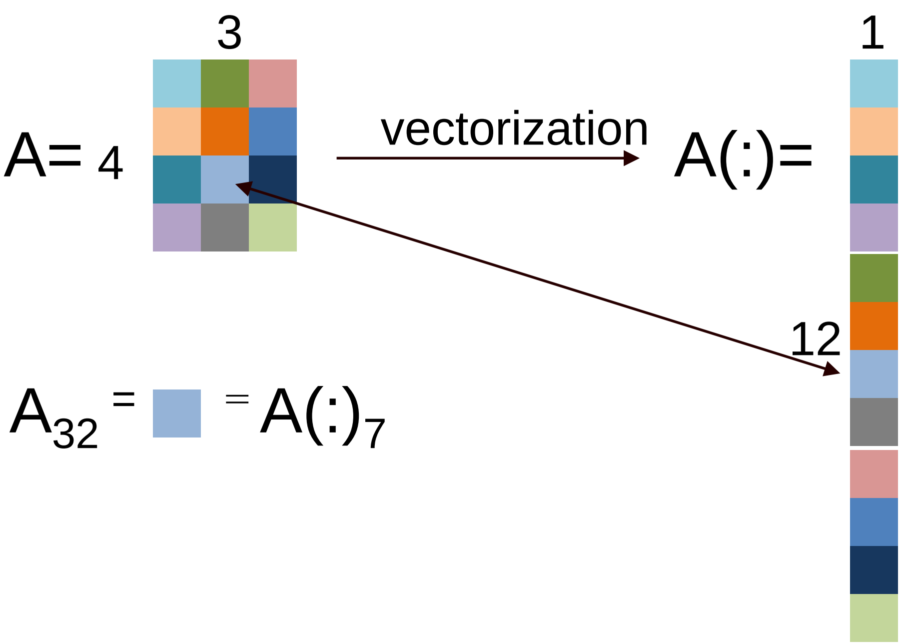
```
y_up = min(nr,max(1,y_up));  % keep y_up index within legal range of [1,nr]
y_down = min(nr,max(1,y_down));

ind_up = sub2ind([nr,nc],y_up(:),x(:));  % create linear index
ind_down = sub2ind([nr,nc],y_down(:),x(:));

linear_index = sub2ind([n_row, n_col], row_subscript, col_subscript)
```



7 = sub2ind([4 3], 3,2)

Convolution without Looping using meshgrid

```
y_up = min(nr,max(1,y_up));  % keep y_up index within legal range of [1,nr]
y_down = min(nr,max(1,y_down));

ind_up = sub2ind([nr,nc],y_up(:),x(:));  % create linear index
ind_down = sub2ind([nr,nc],y_down(:),x(:));

linear_index = sub2ind([n_row, n_col], row_subscript, col_subscript)
```

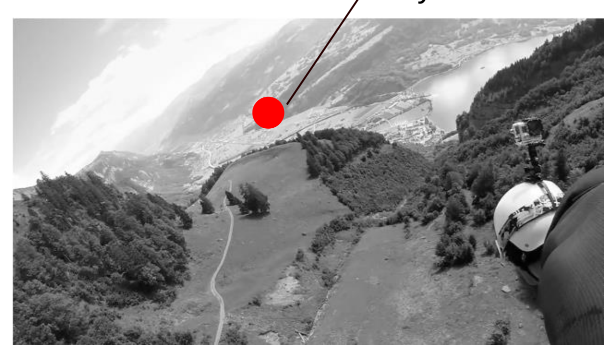


7 = sub2ind([4 3], 3,2)

ind_up = sub2ind([nr,nc],y_up(:),x(:))

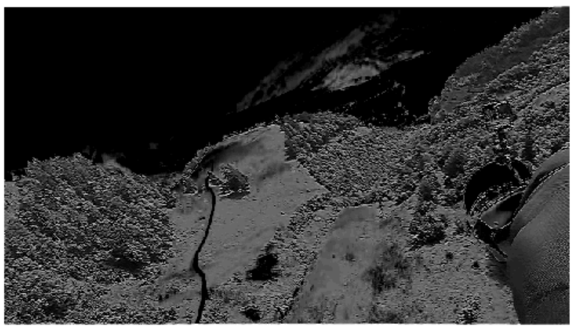
Operation on vectors

$Jb_{xy}=Jb_{idx}$



Convolution without Looping using meshgrid

```
J_out = 2*Jb(:) - 0.8*Jb(ind_up) - 0.8*Jb(ind_down);
J_out = reshape(J_out, nr, nc);

figure; imagesc(J_out);colormap(gray)
```



J_out

=

2

Jb

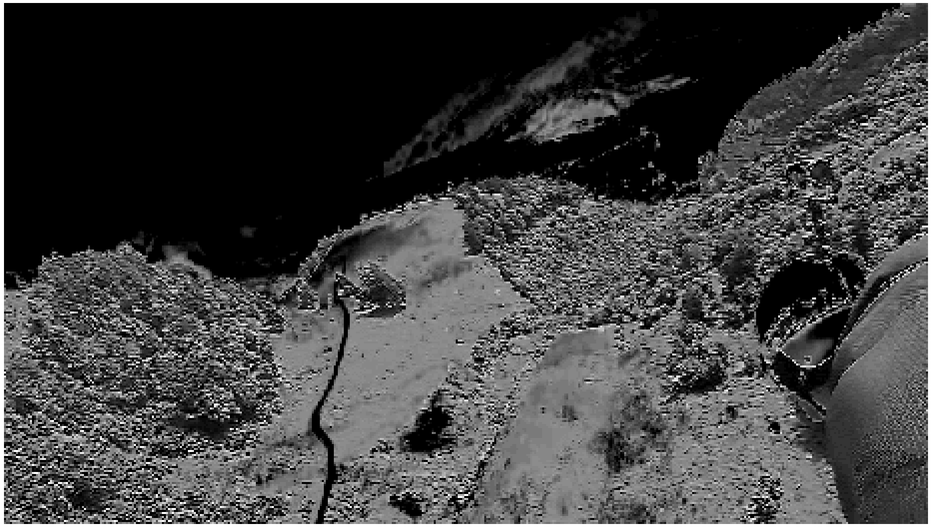
-0.8

Jb(ind_up)

-0.8

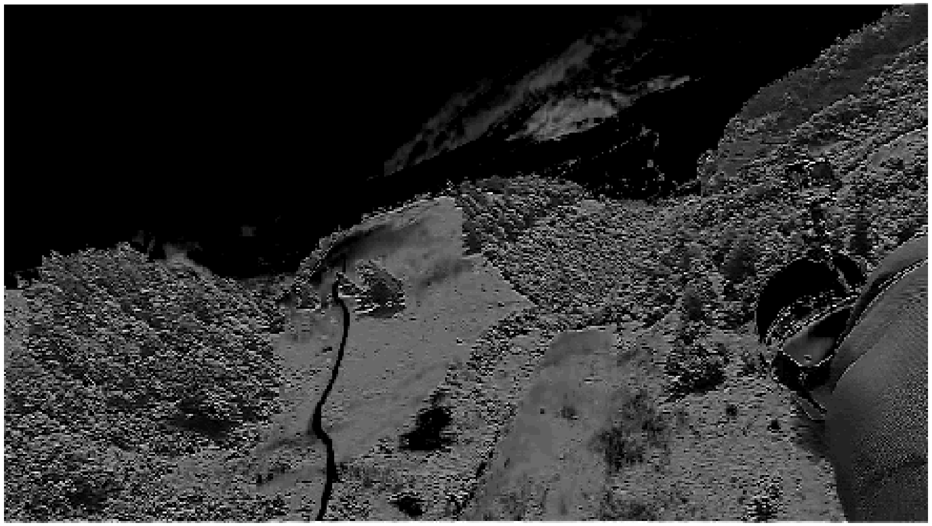
Jb(ind_down)

With loop



Computation time: 0.050154 sec

Without loop



Computation time: 0.024047 sec

Canny Edge Detection

$$B(i,j) = \begin{cases} 1 & \text{if } I(i,j) \text{ is edge} \\ 0 & \text{if } I(i,j) \text{ is not edge} \end{cases}$$

Objective: to localize edges given an image.

Binary image indicating edge pixels

Original image, I

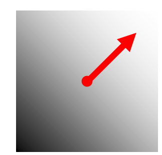
Edge map image, B

Canny Edge Detection

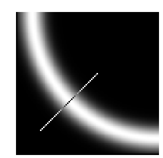
1. Filter image by derivatives of Gaussian

2. Compute magnitude of gradient

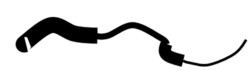
3. Compute edge orientation



4. Detect local maximum



5. Edge linking



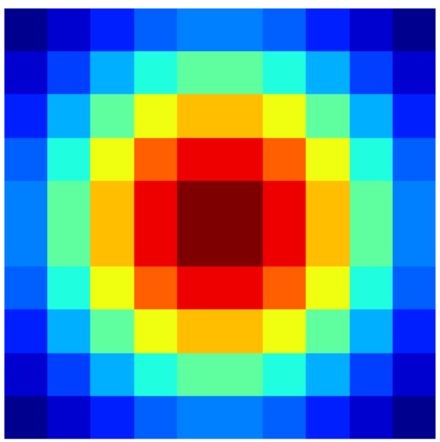
1) Compute Image Gradient

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

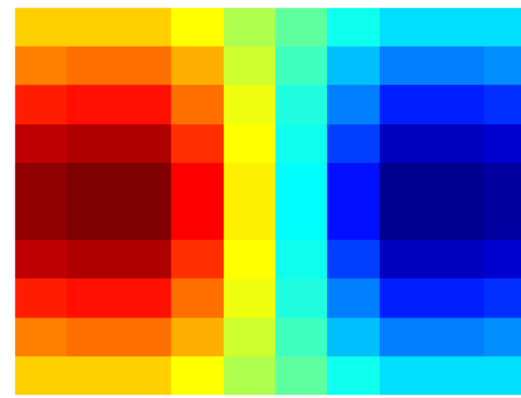
Edge Detection, Step 1,
Filter out noise and compute derivative:

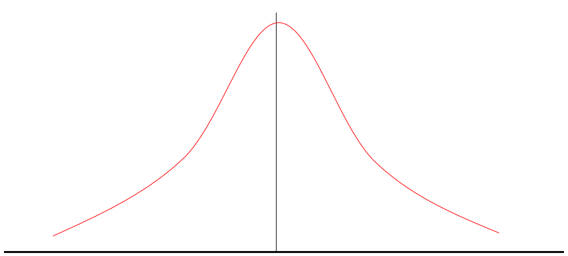
$$\left(\frac{\delta}{\delta x} \otimes G\right)$$

Gradient of Gaussian

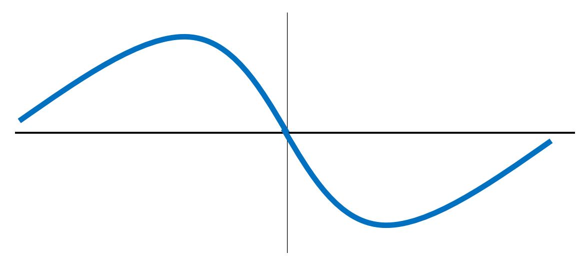


$$\frac{\delta}{\delta x} \longrightarrow$$



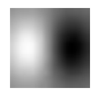


$$\frac{\delta}{\delta x} \longrightarrow$$

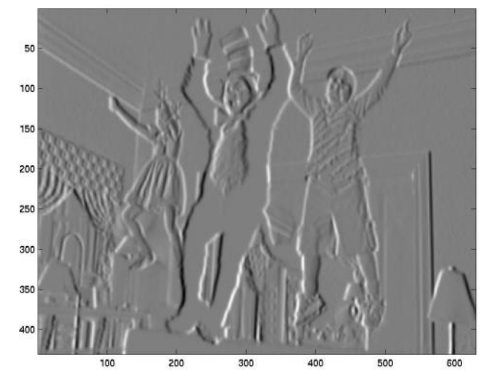


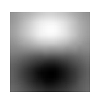
Edge Detection, Step 1,
Filter out noise and compute derivative:

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

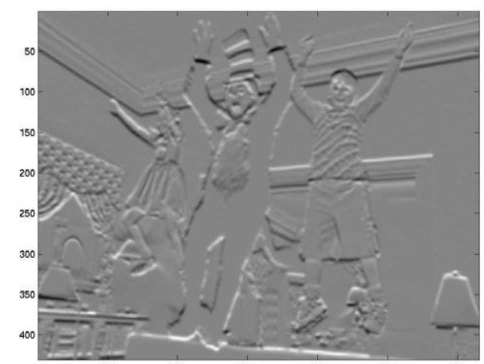


I_x





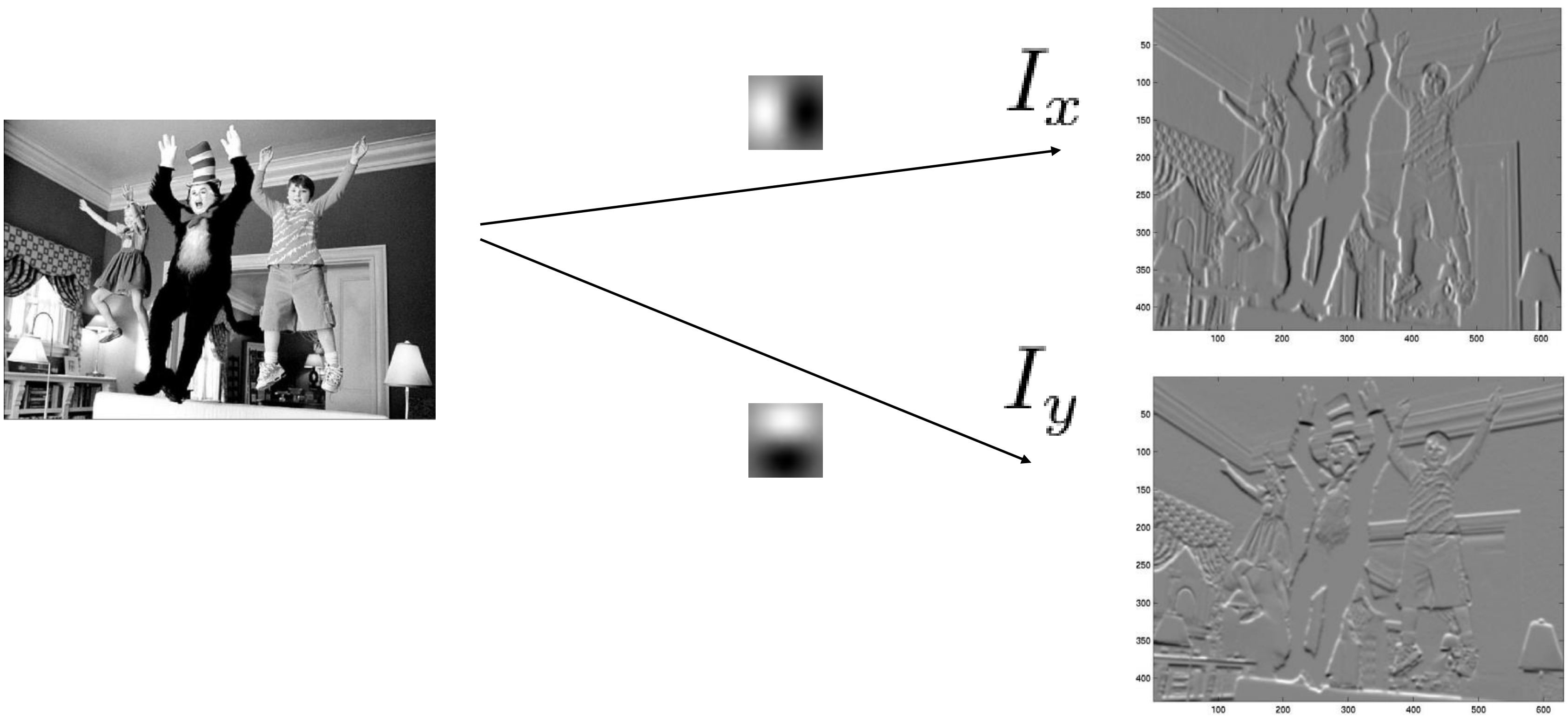
I_y



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

In matlab:

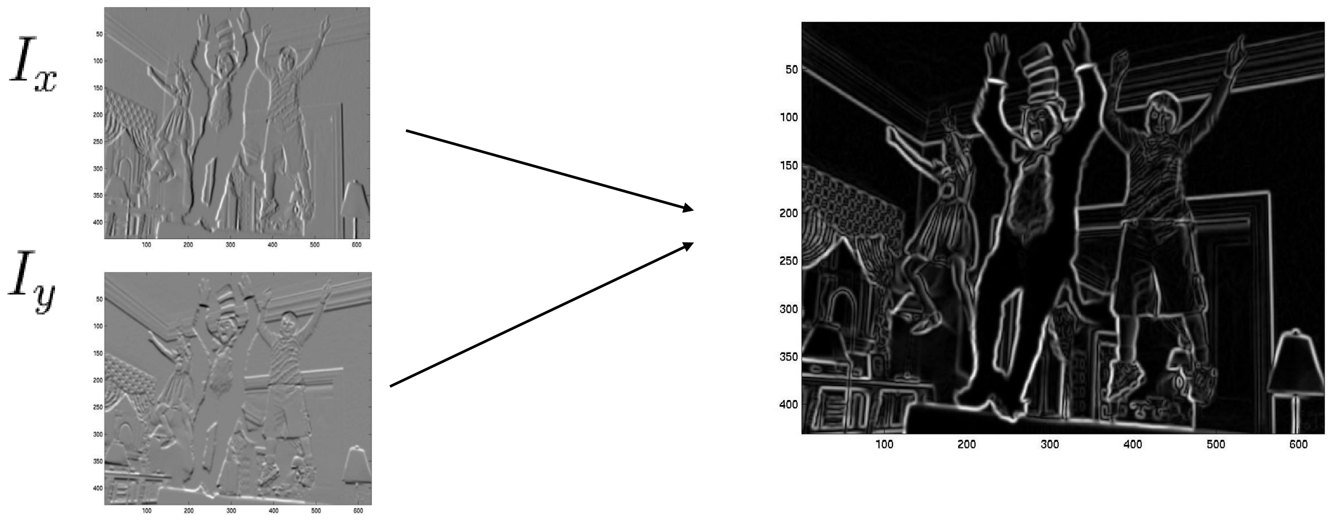
```
>> [dx,dy] = gradient(G); % G is a 2D gaussain
>> Ix = conv2(I,dx,'same'); Iy = conv2(I,dy,'same');
```



Edge Detection: Step 2
Compute the magnitude of the gradient

In Matlab:

```
>> Im = sqrt(Ix.*Ix + Iy.*Iy);
```



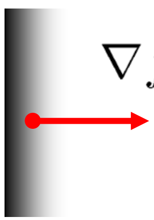
We know roughly where are the edges, but we need their precise location.

Finding the orientation of the edge

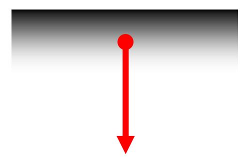
- The gradient of an image:

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

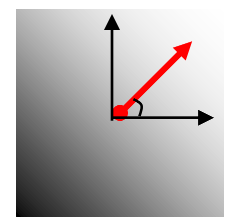
- The gradient points in the direction of most rapid change in intensity



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$



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

- The image gradient direction is given by:

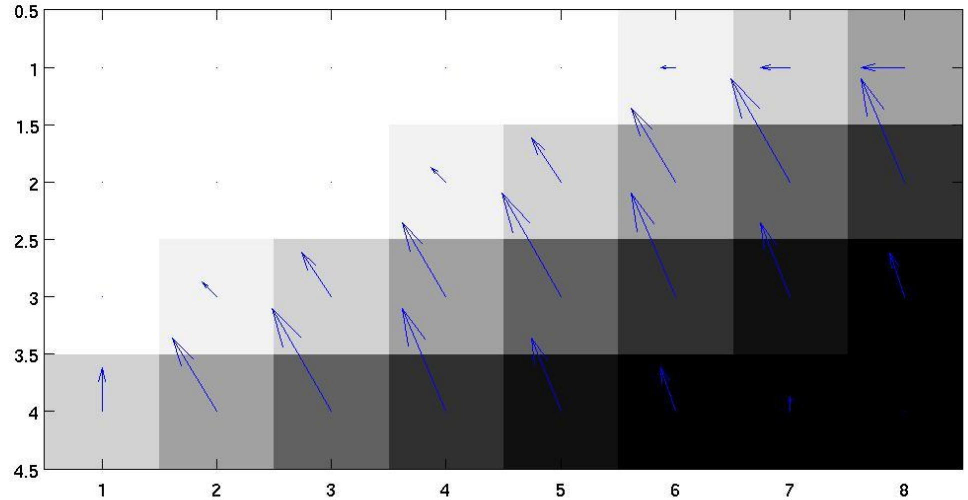
$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} \Big/ \frac{\partial f}{\partial x}\right)$$

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

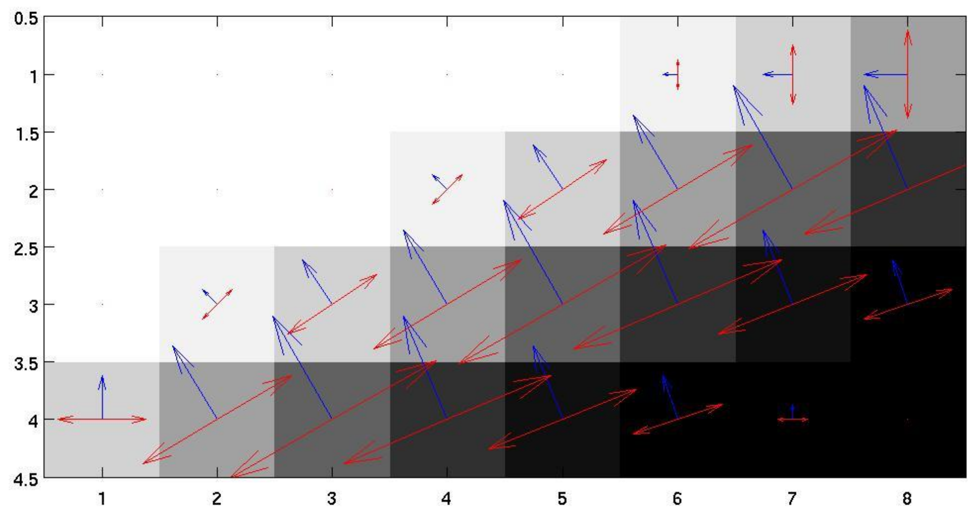
$$\theta_{edge} = tan^{-1}\left(-\frac{\delta f}{\delta x} / \frac{\delta f}{\delta y}\right)$$

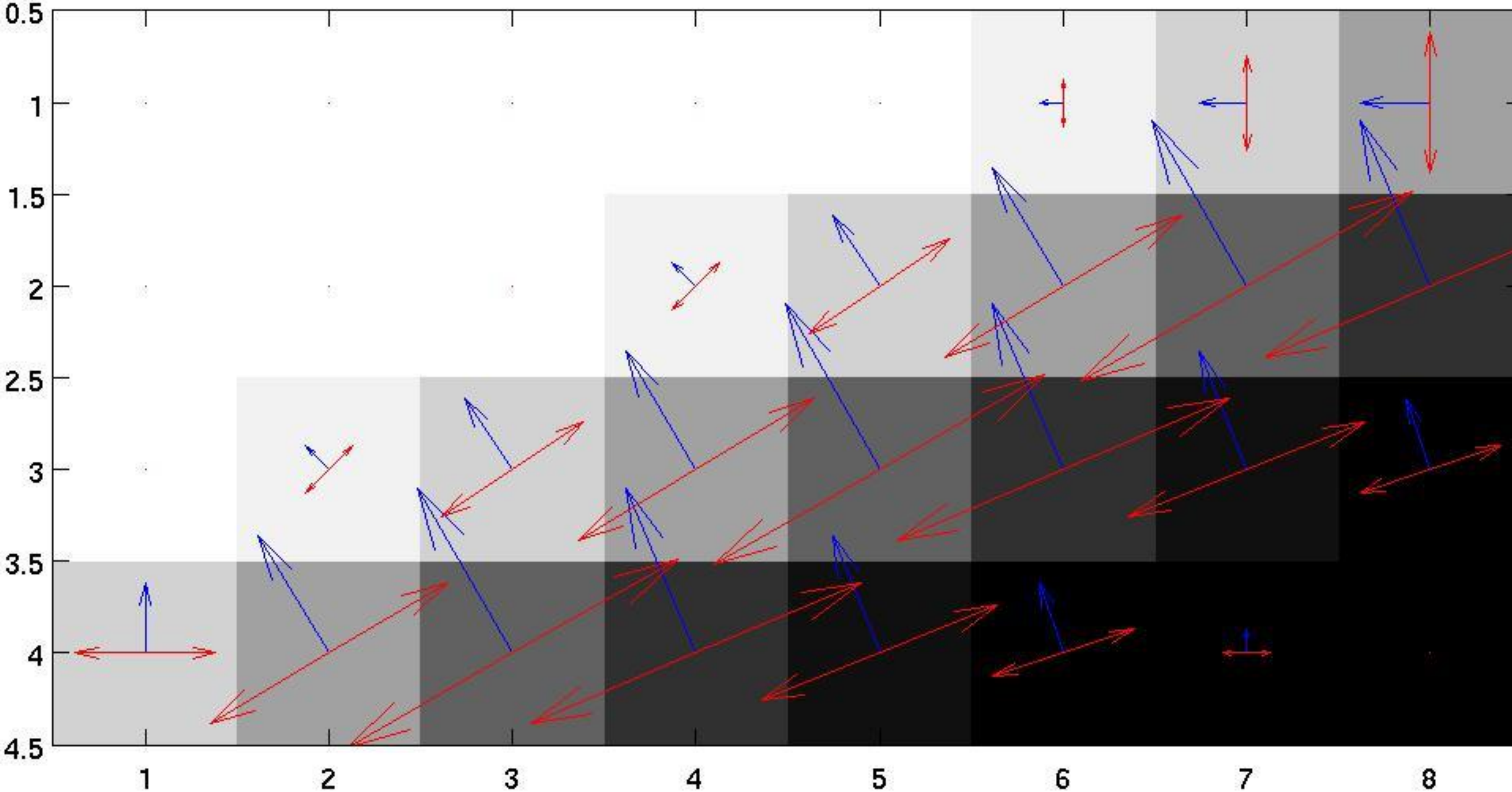
%% define image gradient operator
dy = [1;-1];
dx = [1,-1];

%% compute image gradient in x and y
Iy = conv2(I,dy,'same');
Ix = conv2(I,dx,'same');



%% 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');




0.5
1
1.5
2
2.5
3
3.5
4
4.5
1
2
3
4
5
6
7
8

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

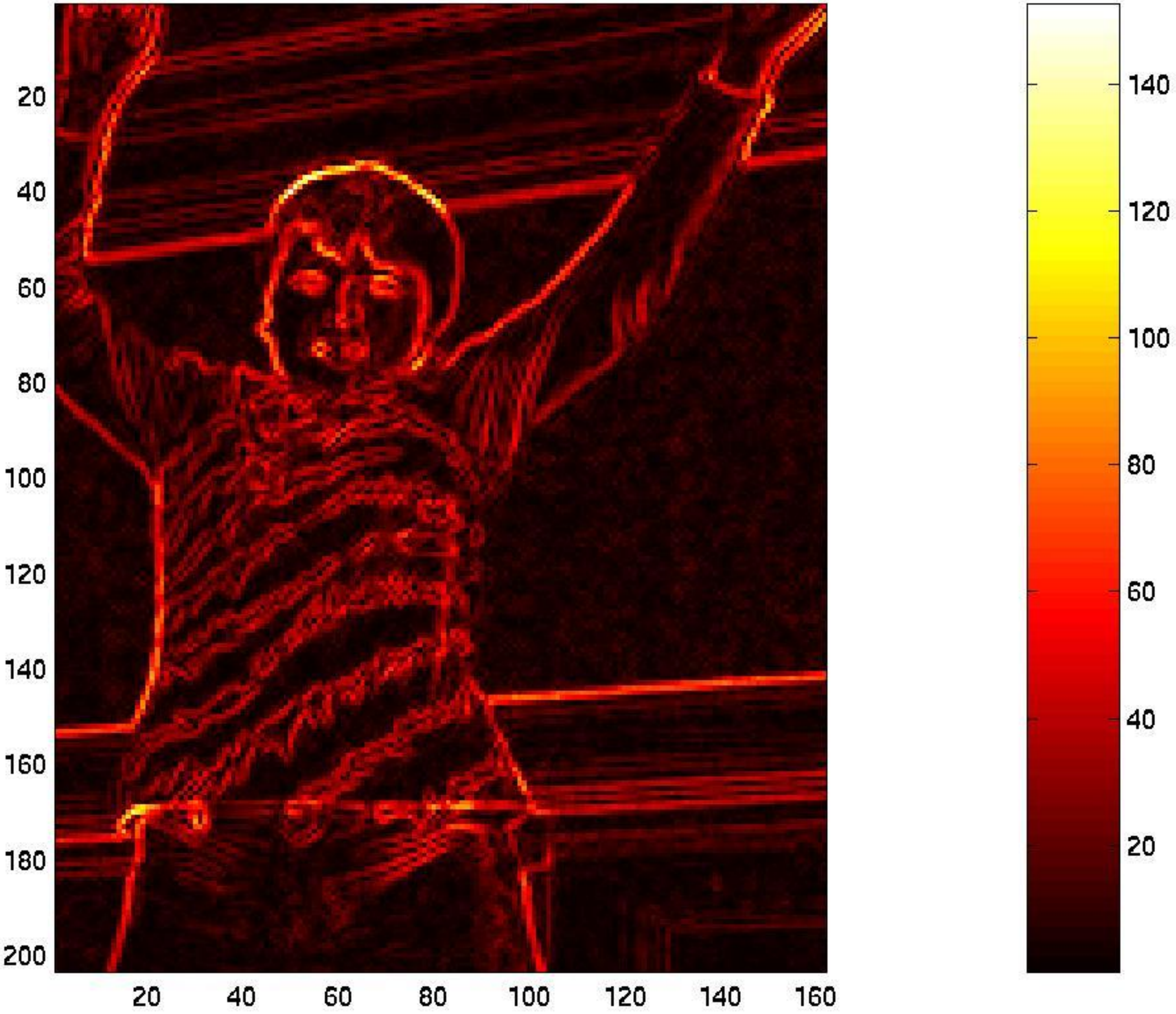
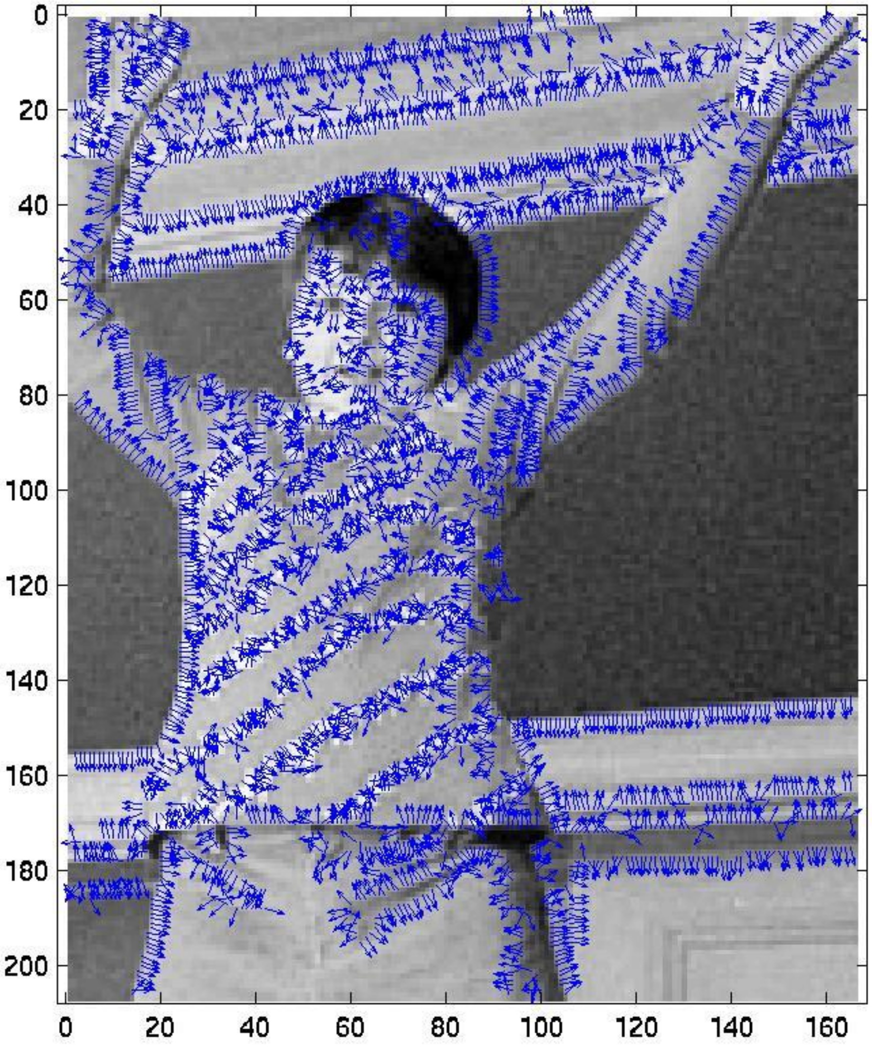
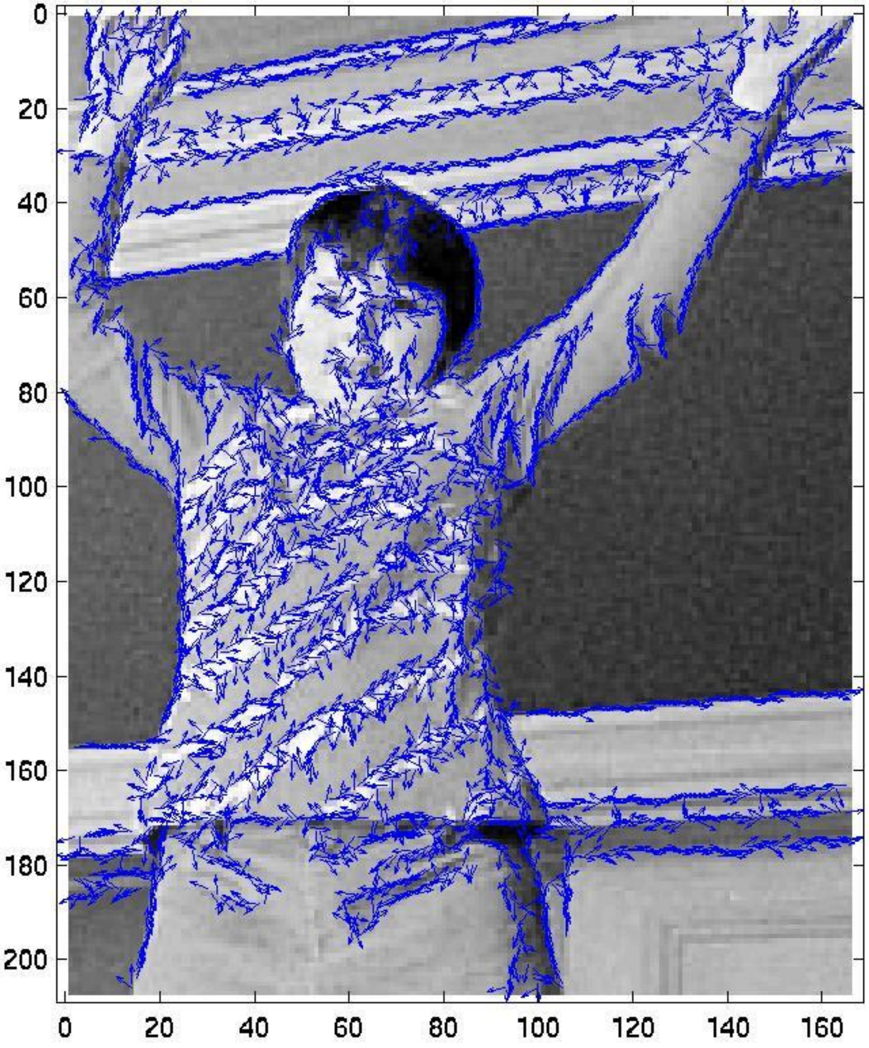


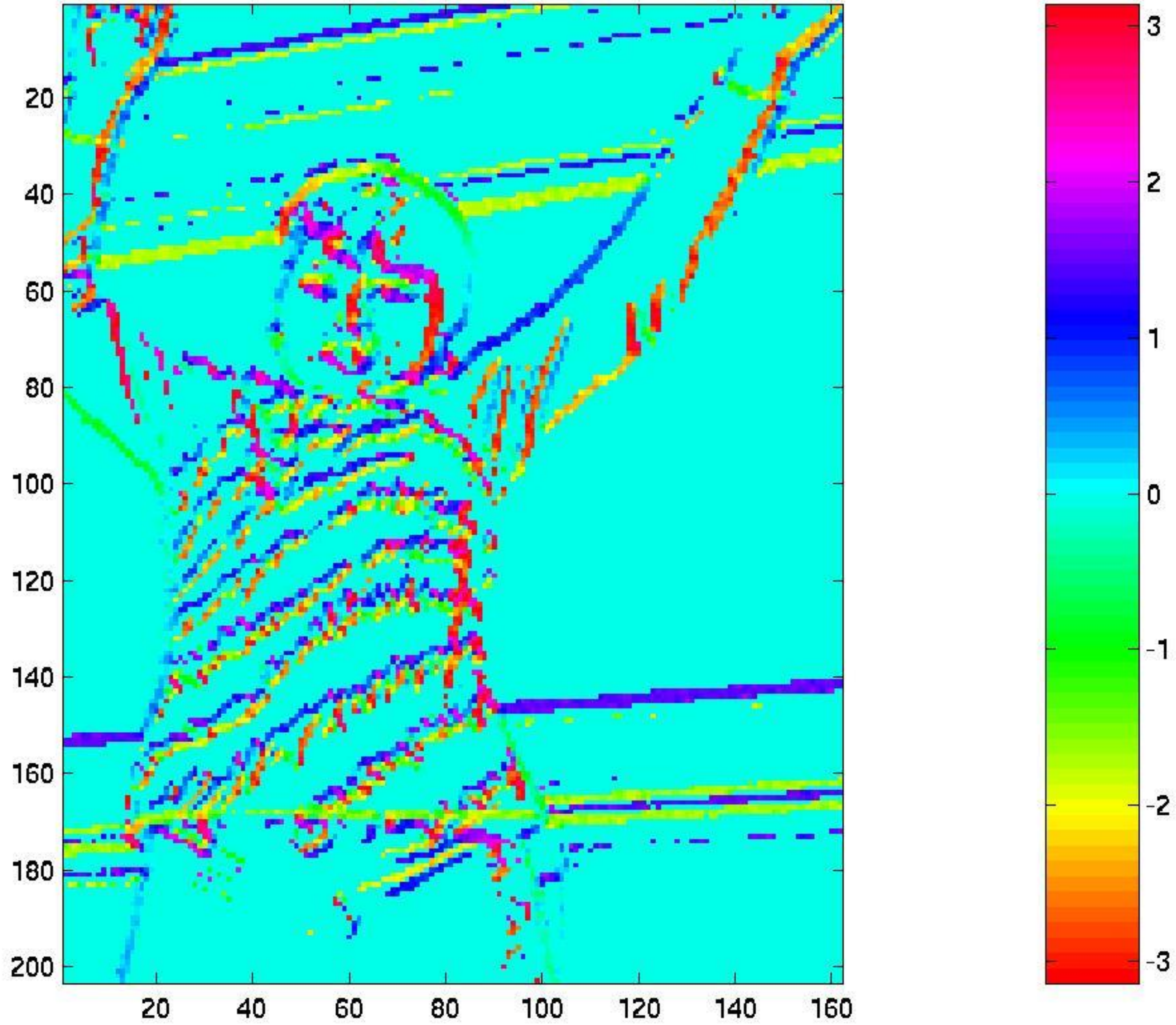
image gradient direction:



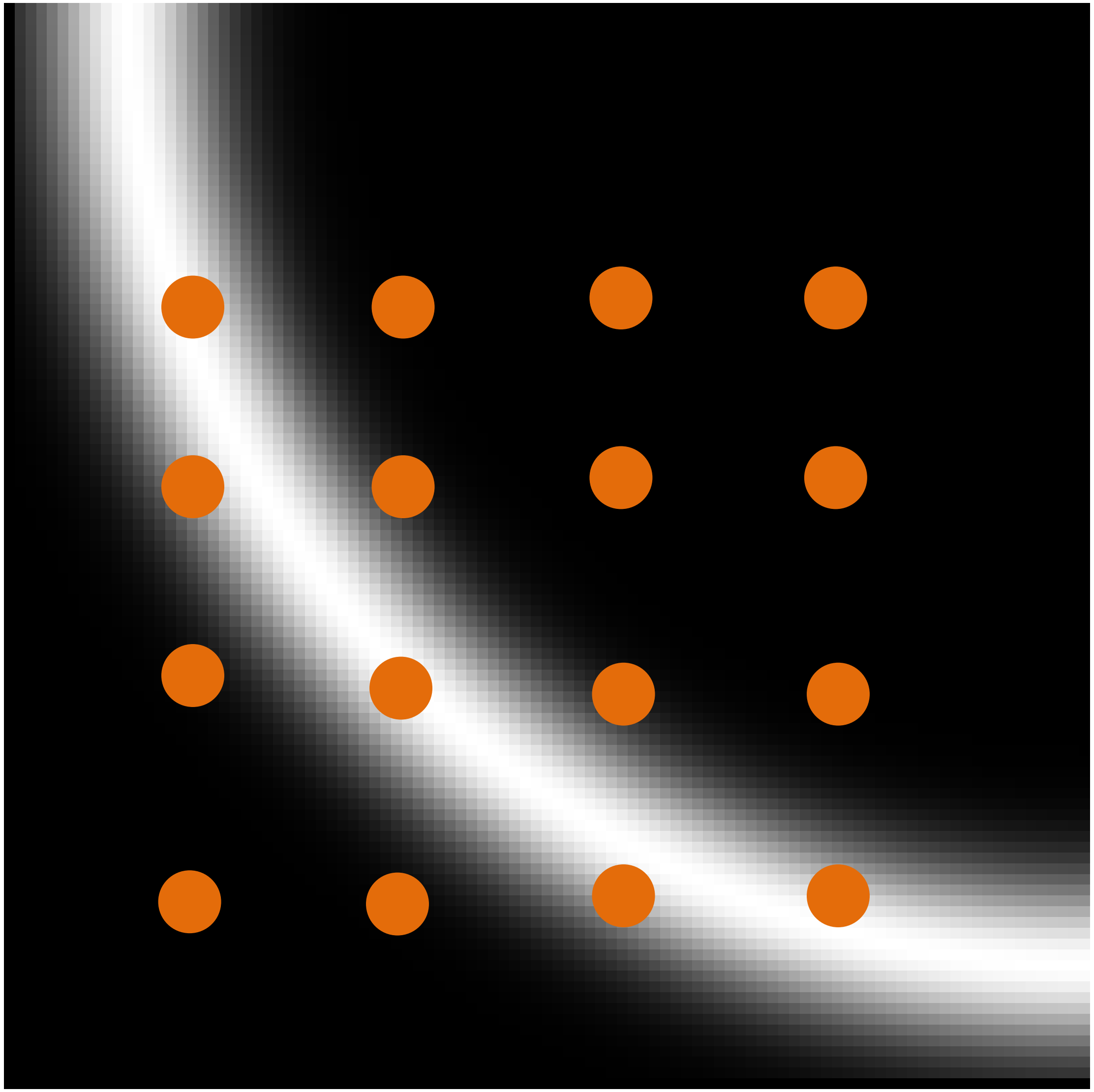
Edge orientation direction:



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

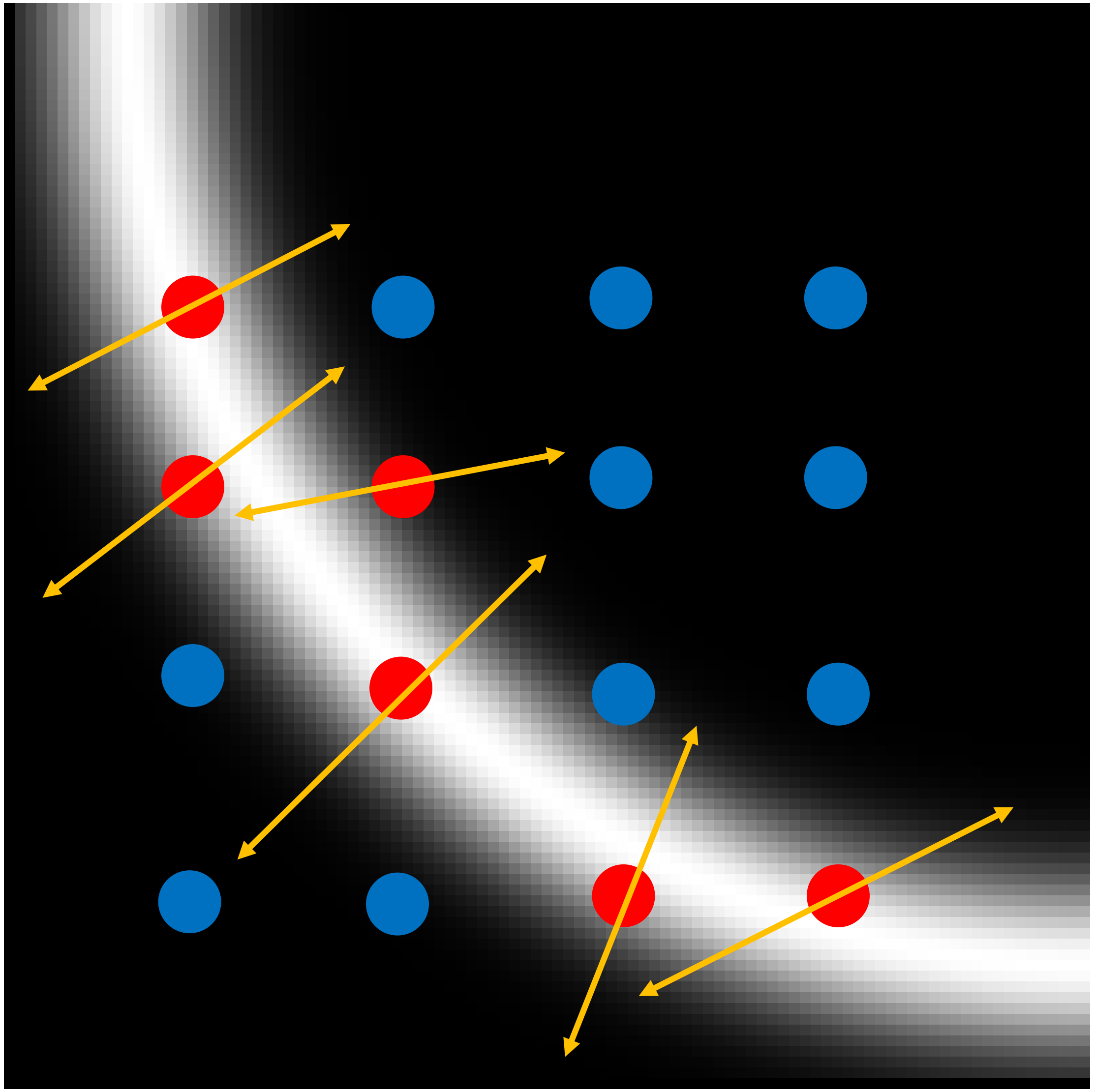


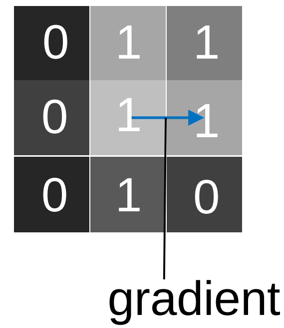
Discretized pixel locations



(Forsyth & Ponce)

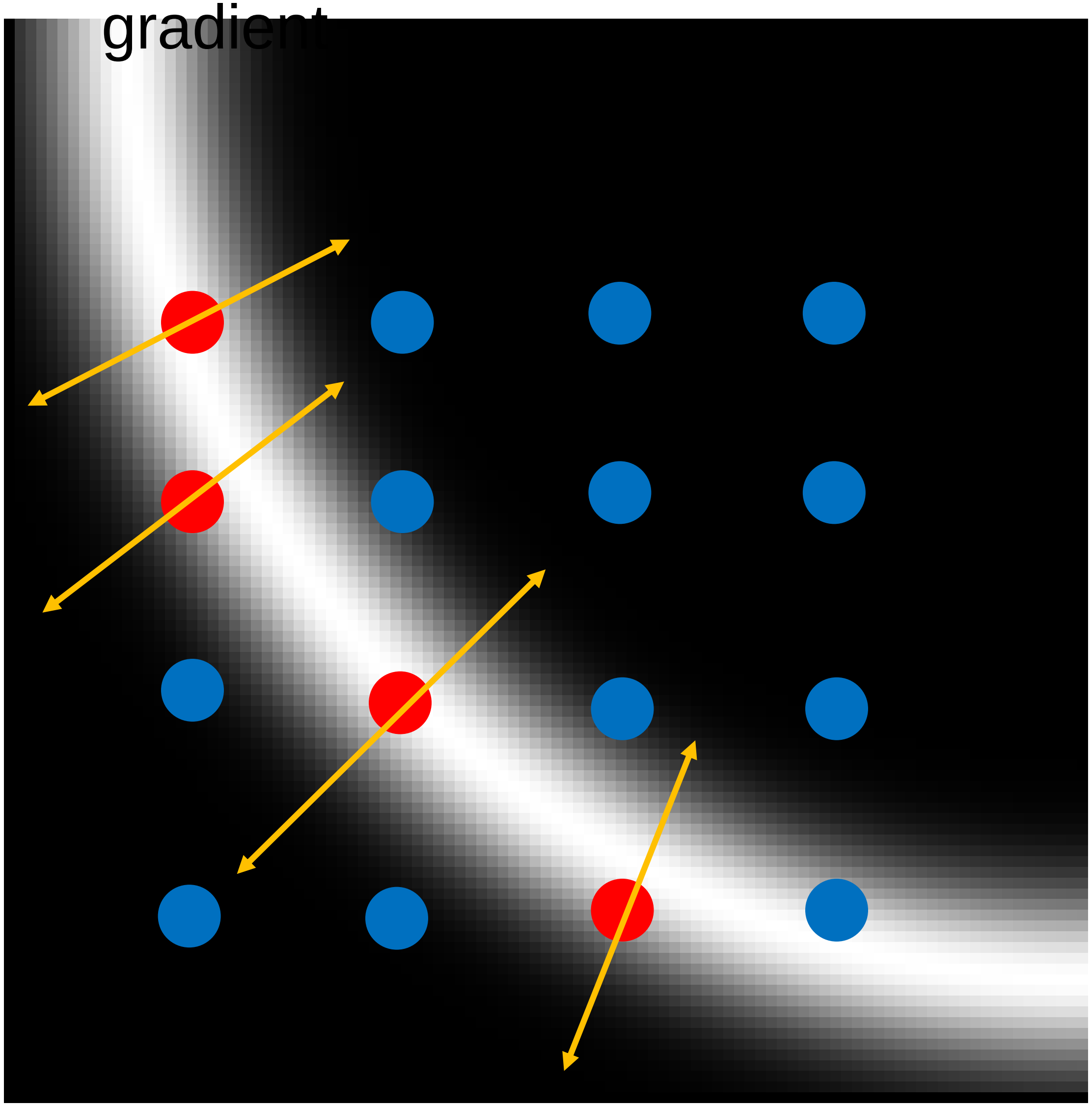
Thesholding

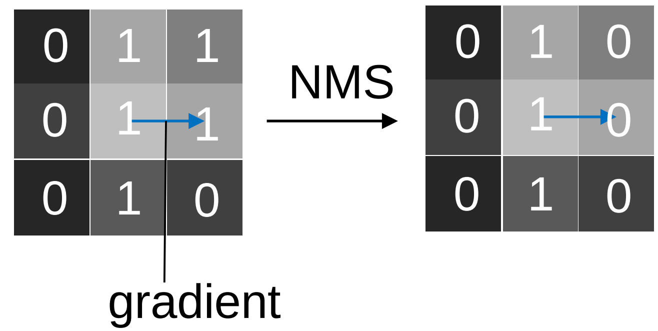




(Forsyth & Ponce)

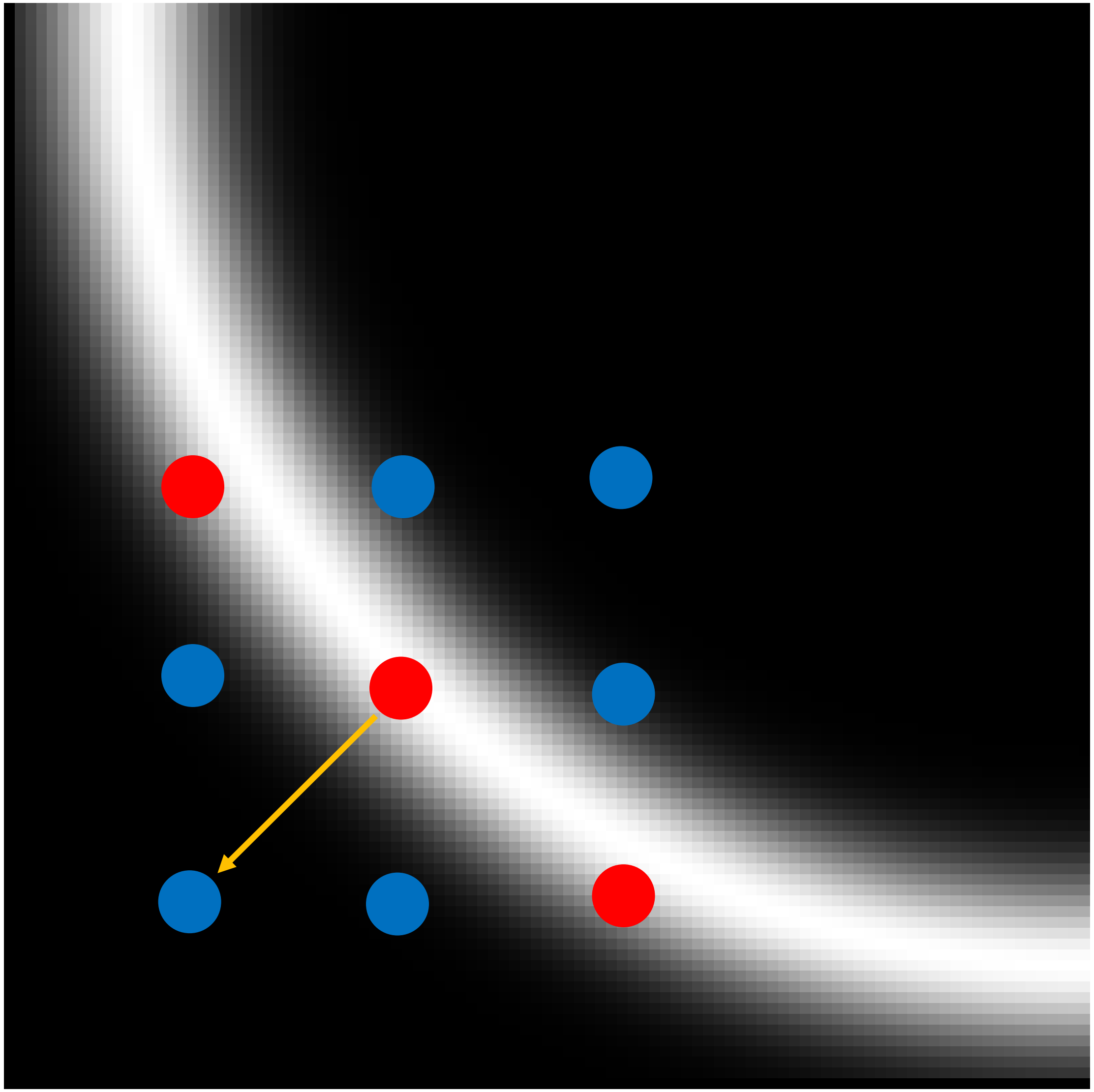
Non-maximum suppression along the line of the gradient

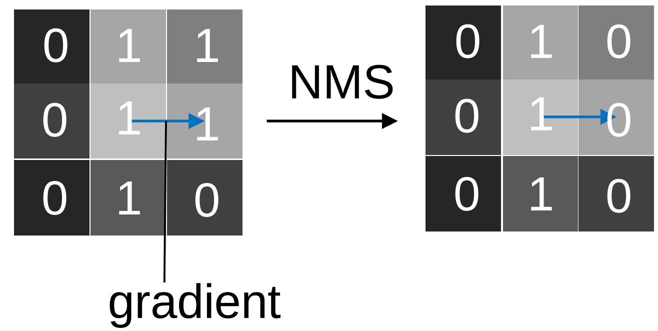




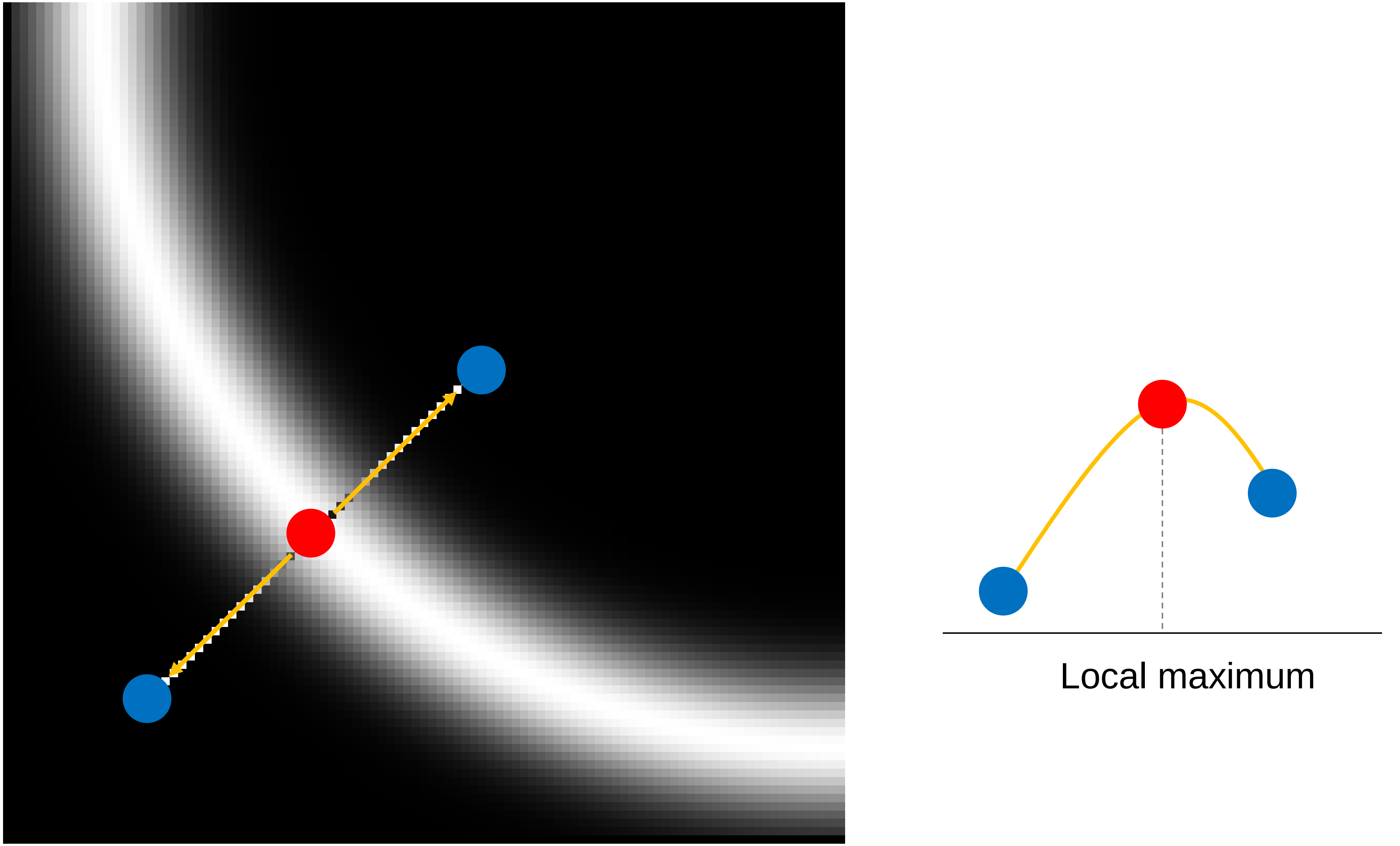
(Forsyth & Ponce)

Gradient direction

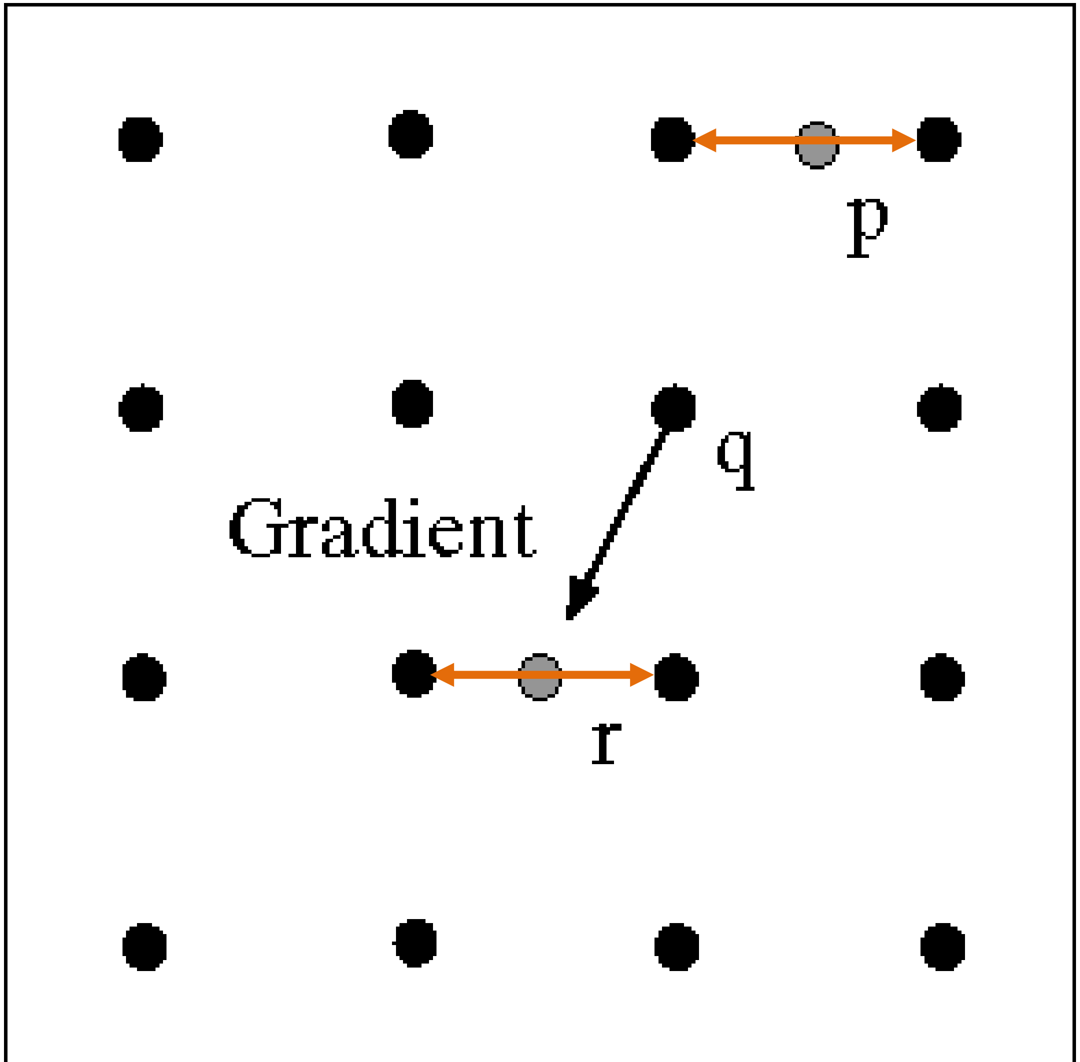




(Forsyth & Ponce)


Local maximum

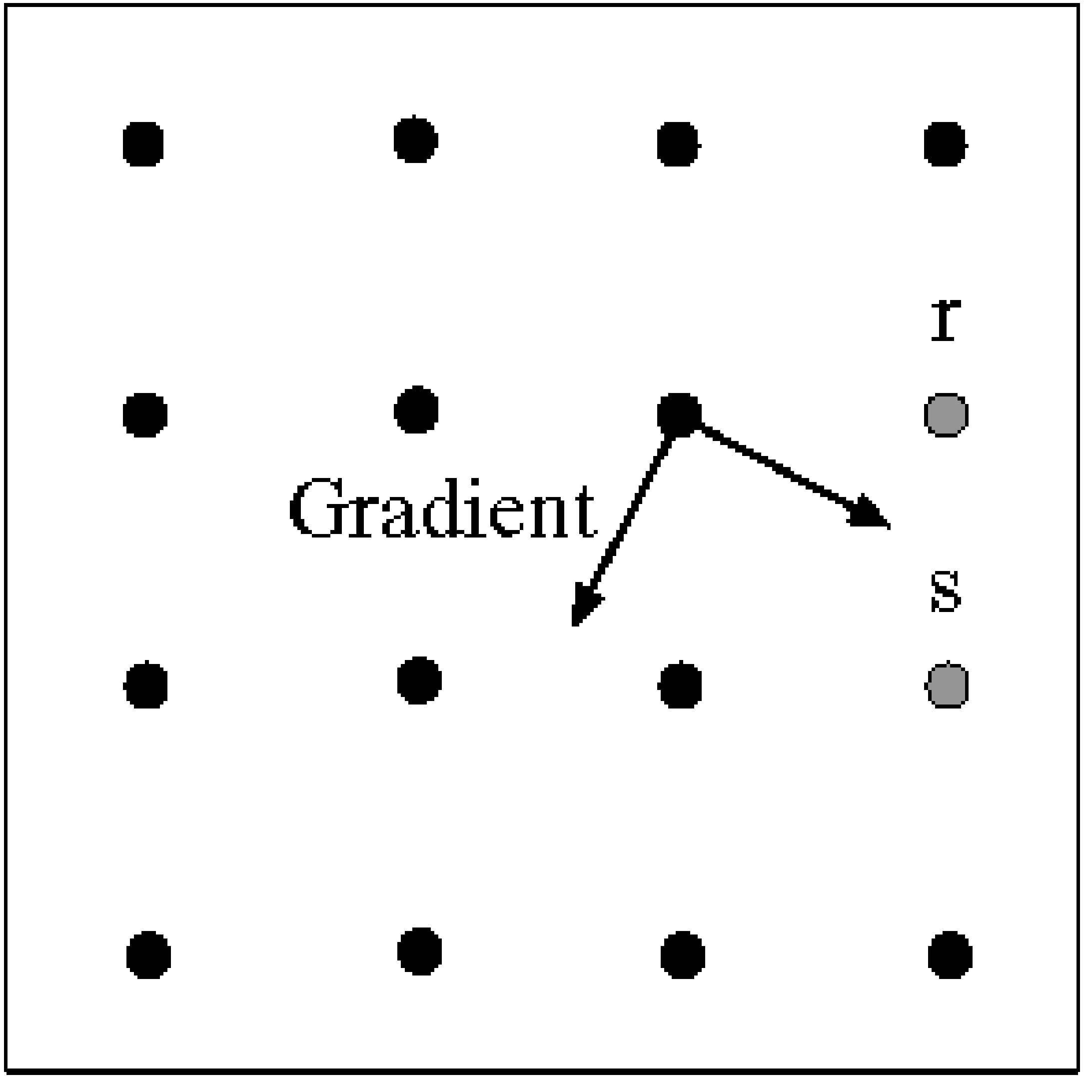
No intensity values at r and p:
Interpolate these intensities using neighbor pixels.

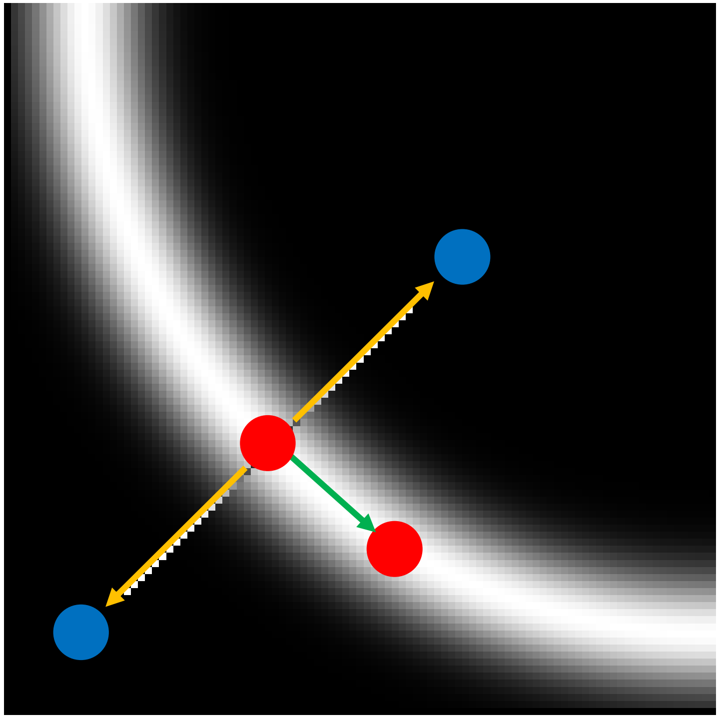


Where is next edge poin

Where is next edge point?

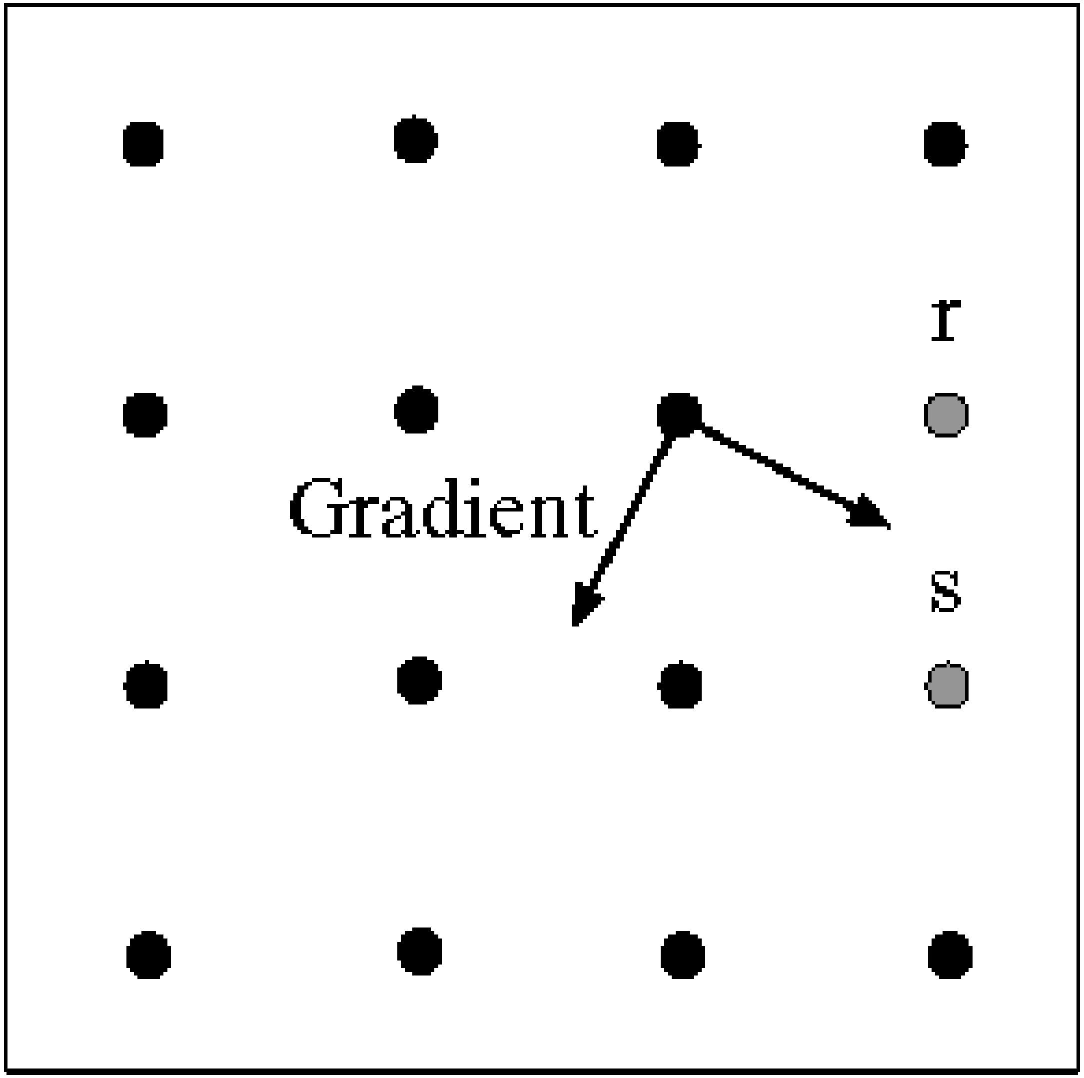
we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points

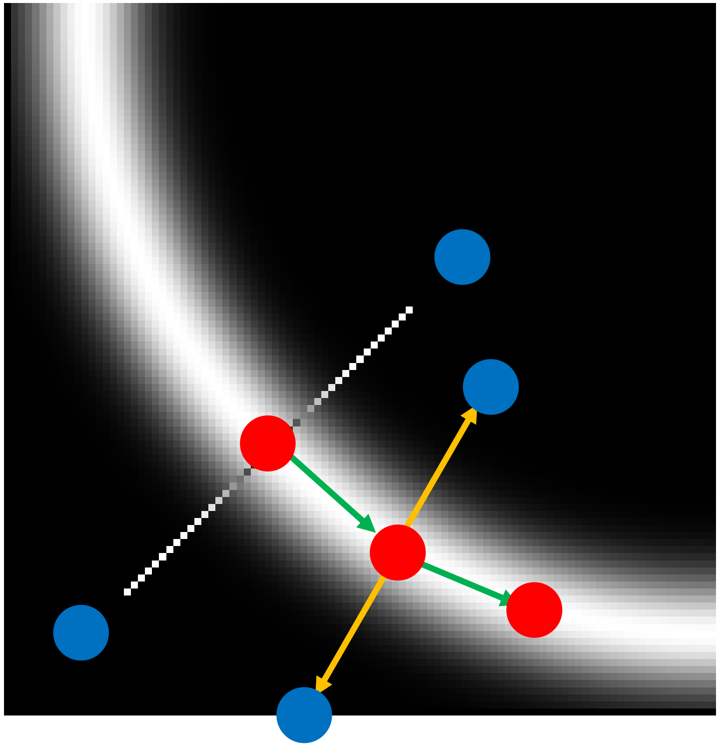




Where is next edge point?

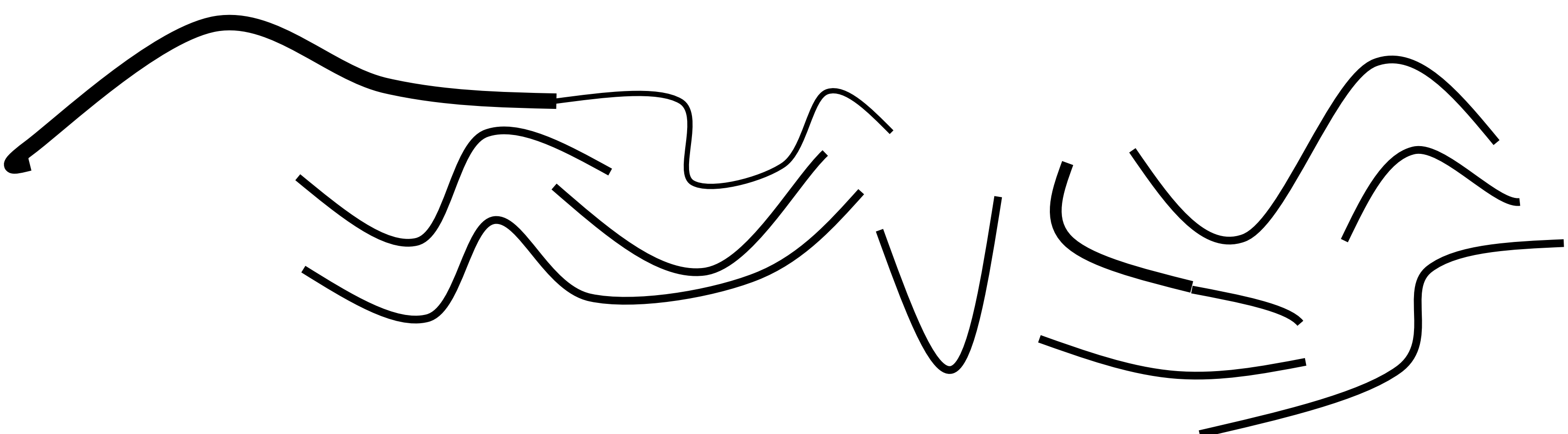
we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points





Edge Linking: 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.

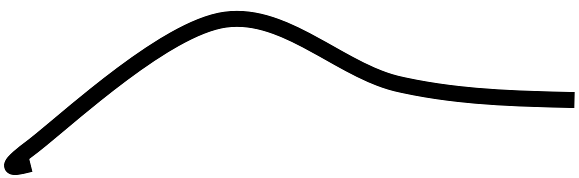


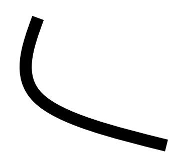
Edge Linking: Hysteresis

- 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.

threshold_high

0	1	0	0
0	1	0	1
0	0	0	0
0	1	0	0





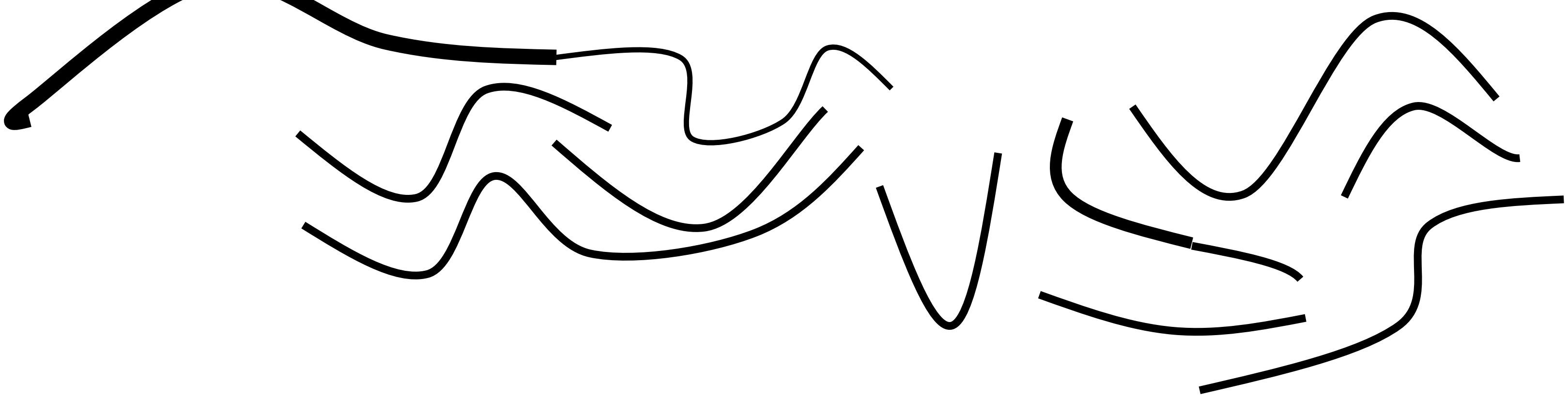
Edge Linking: Hysteresis

threshold_high

0	1	0	0
0	1	0	1
0	0	0	0
0	1	0	0

threshold_low

0	1	0	0
0	1	0	1
0	1	0	0
0	1	0	1



Edge Linking: Hysteresis

threshold_high

0	1	0	0
0	1	0	1
0	0	0	0
0	1	0	0

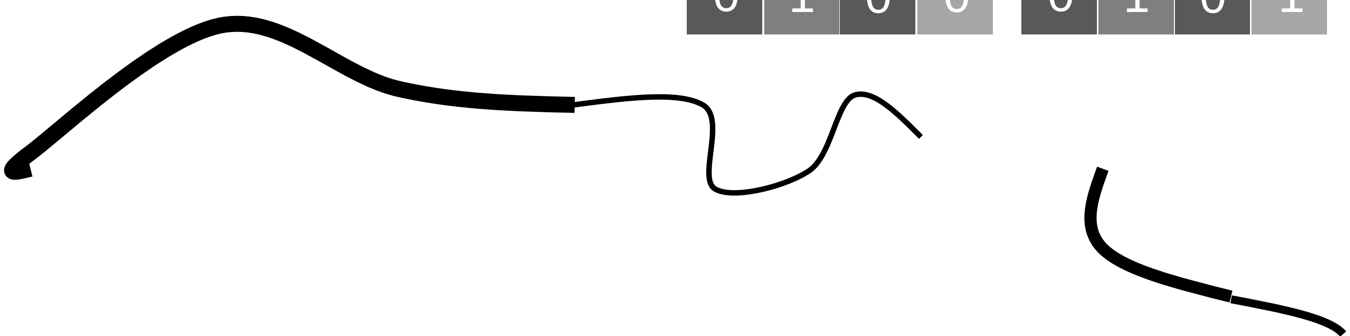
threshold_low

0	1	0	0
0	1	0	1
0	1	0	0
0	1	0	1

→

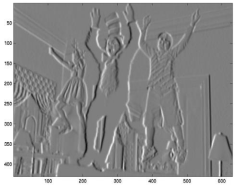
hysteresis

0	1	0	0
0	1	0	1
0	1	0	0
0	1	0	0



Canny Edge Detection

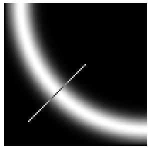
1. Filter image by derivatives of Gaussian



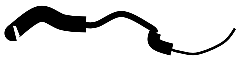
2. Compute magnitude of gradient

3. Compute edge orientation

4. Detect local maximum

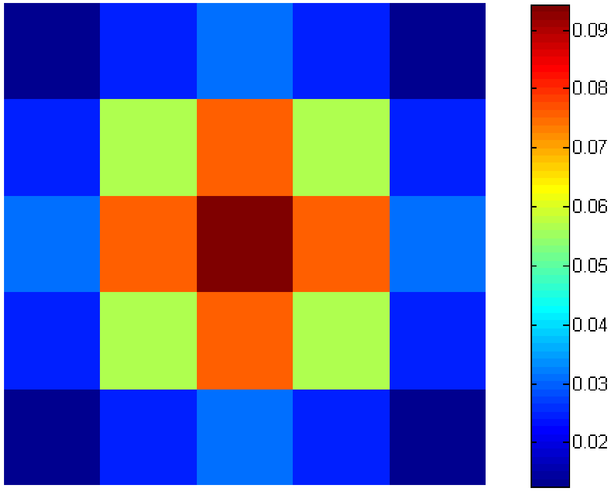


5. Edge linking



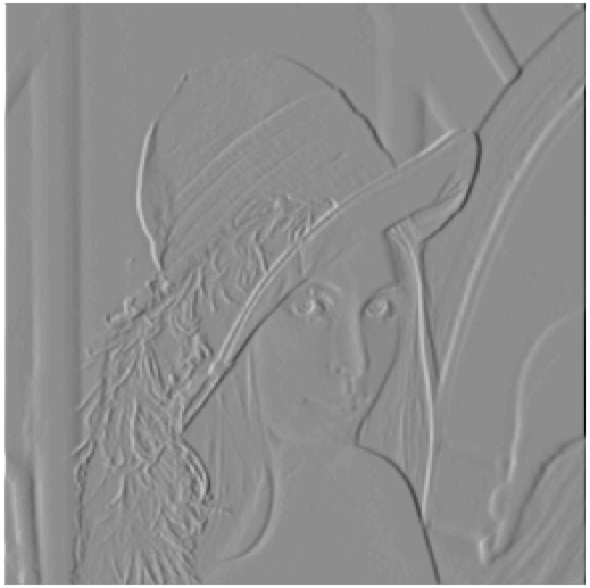
Canny Edge Implementation

```
img = imread ('Lenna.png');
img = rgb2gray(img);
img = double (img);

% Value for high and low thresholding
threshold_low = 0.035;
threshold_high = 0.175;

%% Gaussian filter definition (https://en.wikipedia.org/wiki/Canny_edge_detector)
G = [2, 4, 5, 4, 2; 4, 9, 12, 9, 4;5, 12, 15, 12, 5;4, 9, 12, 9, 4;2, 4, 5, 4, 2];
G = 1/159.* G;

%Filter for horizontal and vertical direction
dx = [1 0 -1];
dy = [1; 0; -1];
```



Canny Edge Implementation

```
% % Convolution of image with Gaussian
Gx = conv2(G, dx, 'same');
Gy = conv2(G, dy, 'same');

% Convolution of image with Gx and Gy
Ix = conv2(img, Gx, 'same');
Iy = conv2(img, Gy, 'same');
```



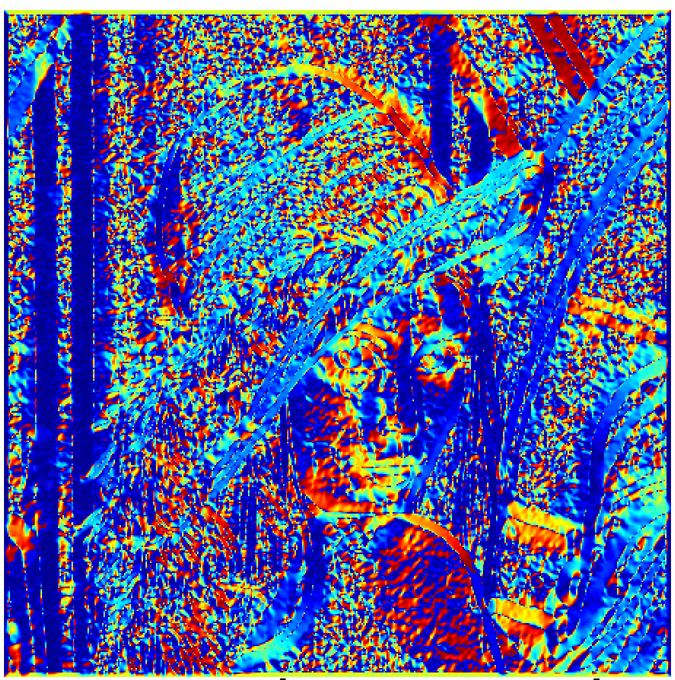
Ix

Iy

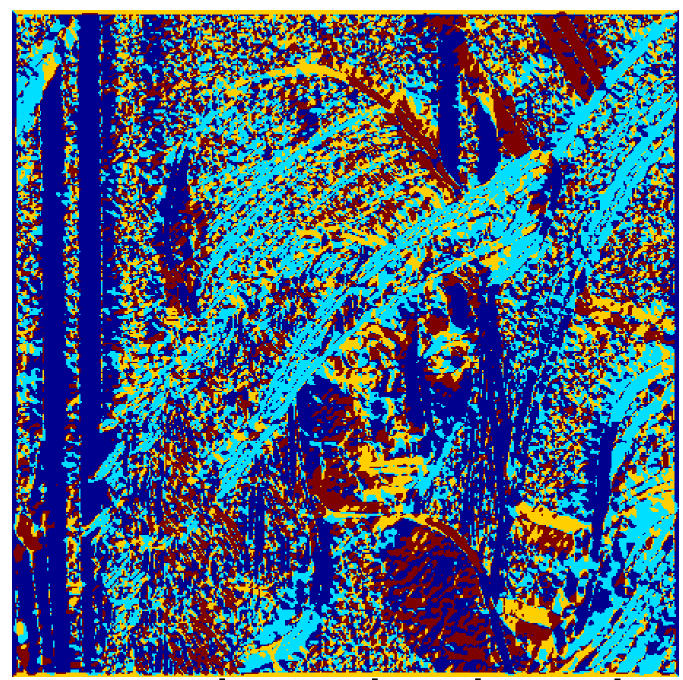
Canny Edge Implementation

```
angle = atan2(Iy, Ix);

%% Edge angle conditioning
angle(angle<0) = pi+angle(angle<0);
angle(angle>7*pi/8) = pi-angle(angle>7*pi/8);
```

```
% Edge angle discretization into 0, pi/4,
pi/2, 3*pi/4
angle(angle>=0&angle<pi/8) = 0;
angle(angle>=pi/8&angle<3*pi/8) = pi/4;
angle(angle>=3*pi/8&angle<5*pi/8) = pi/2;
angle(angle>=5*pi/8&angle<=7*pi/8) =
3*pi/4;
```



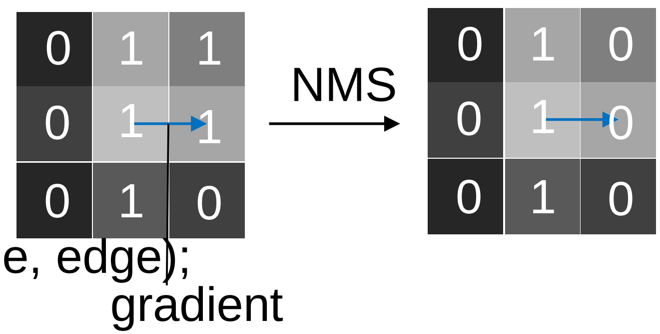
Continuous angle



Discretized angle

Canny Edge Implementation

```
%Calculate magnitude
magnitude = sqrt(Ix.*Ix+Iy.*Iy);
edge = zeros(nr, nc);

%% Non-Maximum Supression
edge = non_maximum_suppression(magnitude, angle, edge);

edge = edge.*magnitude;
```

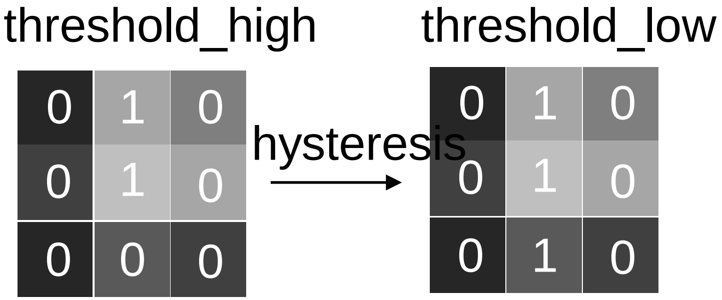



Gradient magnitude

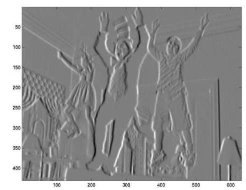

Localized edge

Canny Edge Implementation

```
%% Hysteresis thresholding
% for weak edge
threshold_low = threshold_low * max(edge(:));
% for strong edge
threshold_high = threshold_high * max(edge(:));
linked_edge = zeros(nr, nc);
linked_edge = hysteresis_thresholding(threshold_low, threshold_high, linked_edge, ed
```

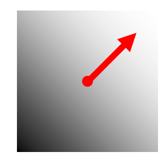


Canny Edge Detection

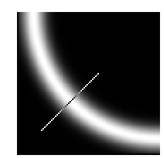


1. Filter image by derivatives of Gaussian

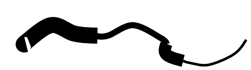
2. Compute magnitude of gradient



3. Compute edge orientation

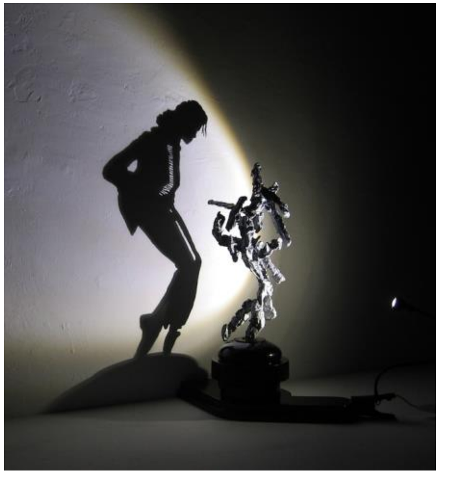


4. Detect local maximum

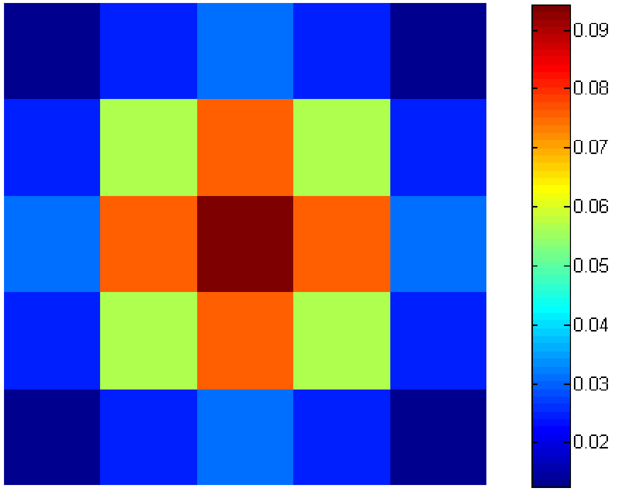


5. Edge linking

Canny Edge Implementation

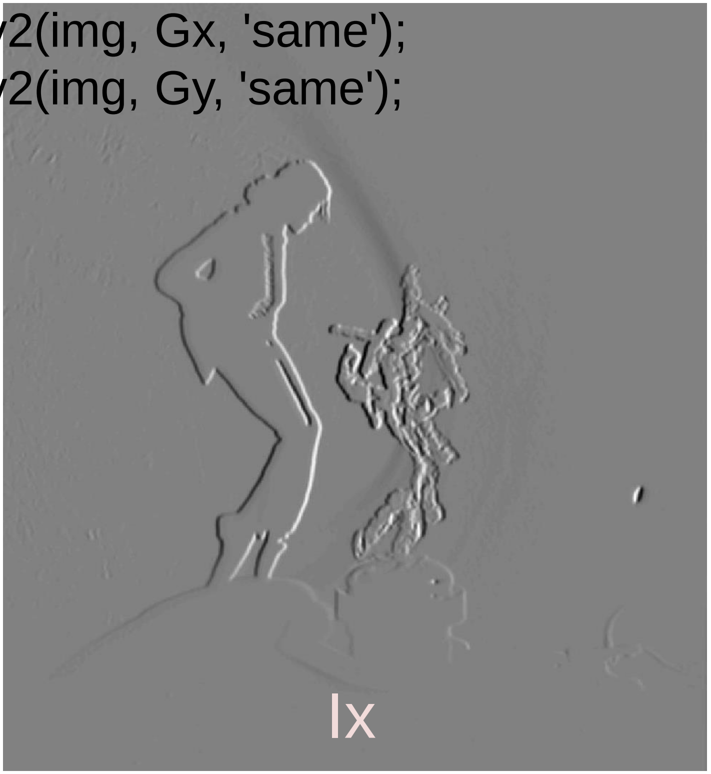


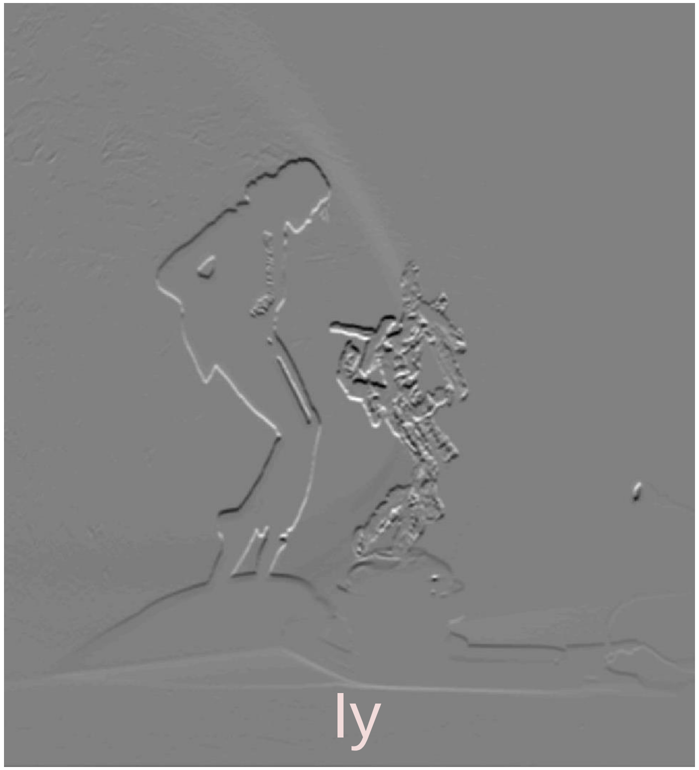
```
img = imread ('image.png');
img = rgb2gray(img);
img = double (img);

% Value for high and low thresholding
threshold_low = 0.035;
threshold_high = 0.175;

%% Gaussian filter definition (https://en.wikipedia.org/wiki/Canny_edge_detector)
G = [2, 4, 5, 4, 2; 4, 9, 12, 9, 4;5, 12, 15, 12, 5;4, 9, 12, 9, 4;2, 4, 5, 4, 2];
G = 1/159.* G;

%Filter for horizontal and vertical direction
dx = [1 -1];
dy = [1; -1];
```



Canny Edge Implementation

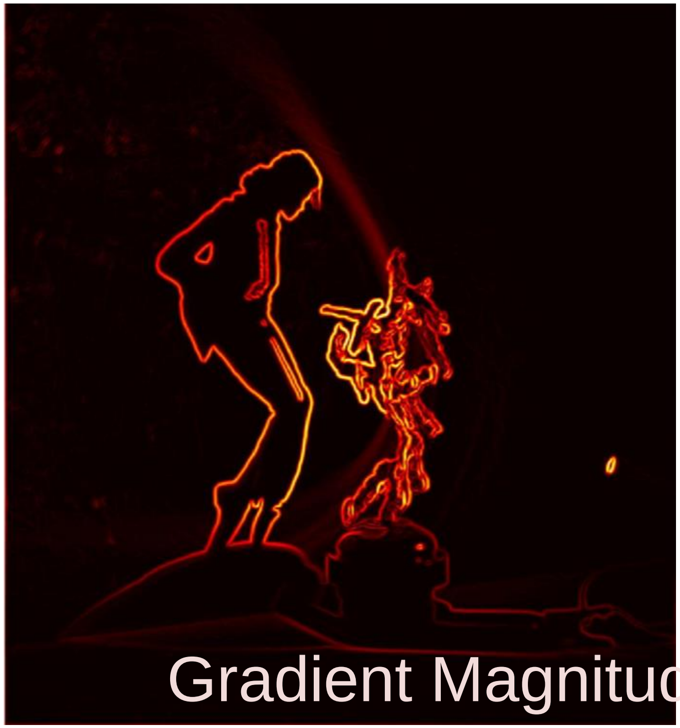
```
% % Convolution of image with
Gaussian
Gx = conv2(G, dx, 'same');
Gy = conv2(G, dy, 'same');

% Convolution of image with Gx and
Gy
Ix = conv2(img, Gx, 'same');
Iy = conv2(img, Gy, 'same');
```

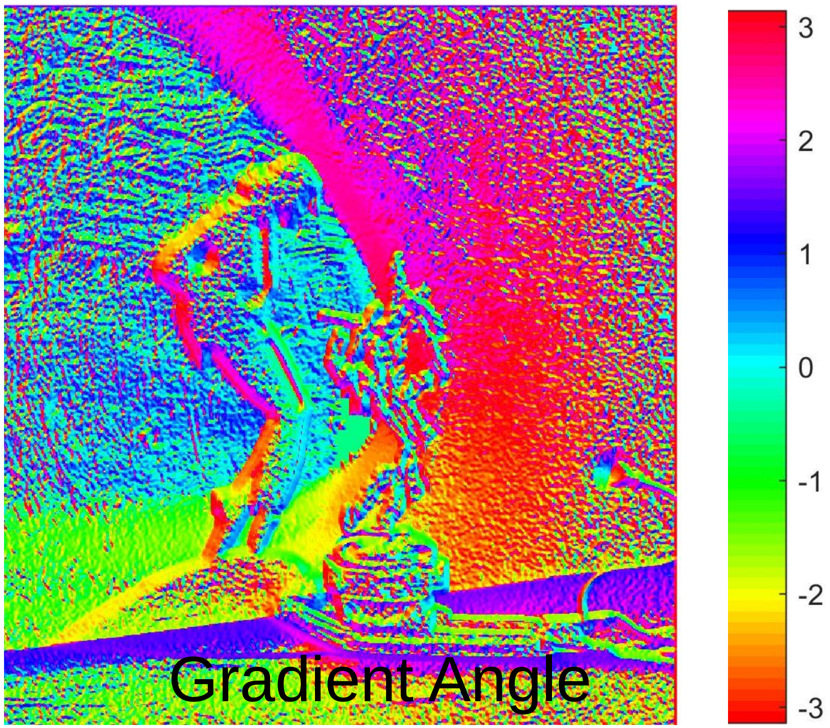


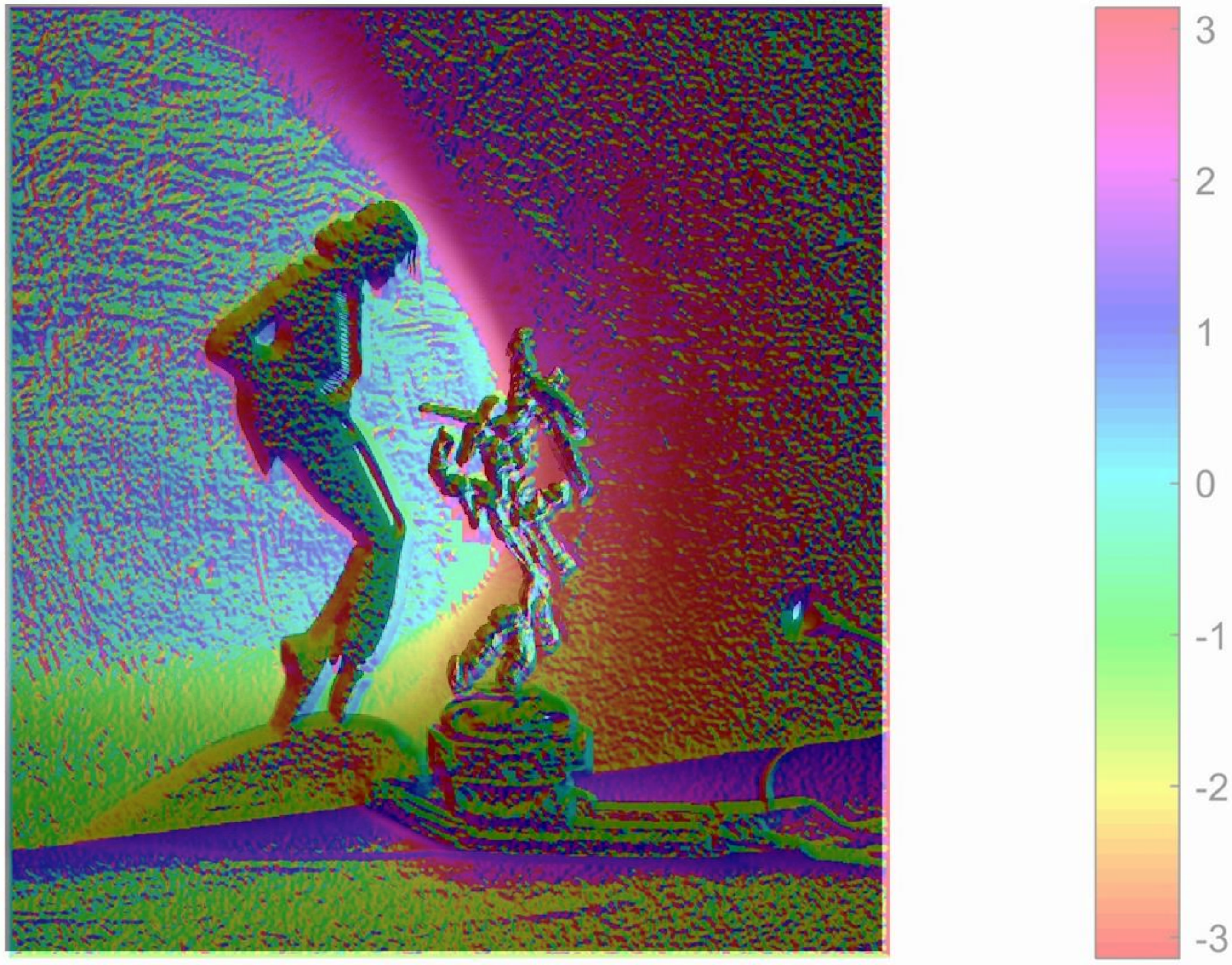


Canny Edge Implementation

```
angle = atan2(Iy, Ix);
mag = sqrt(Iy.^2 + Ix.^2);
```





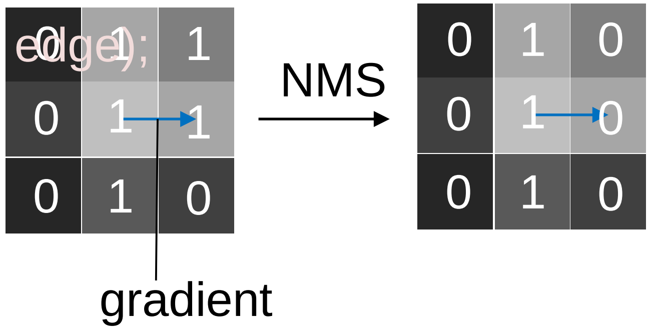


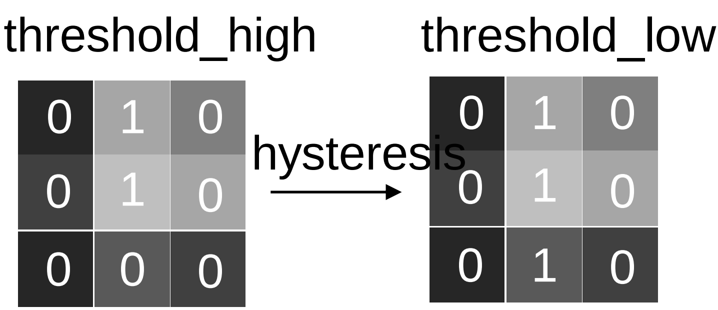
3
2
1
0
-1
-2
-3

Canny Edge Implementation

```
%% Non-Maximum Supression
edge = non_maximum_suppression(magnitude, angle, edge);
```

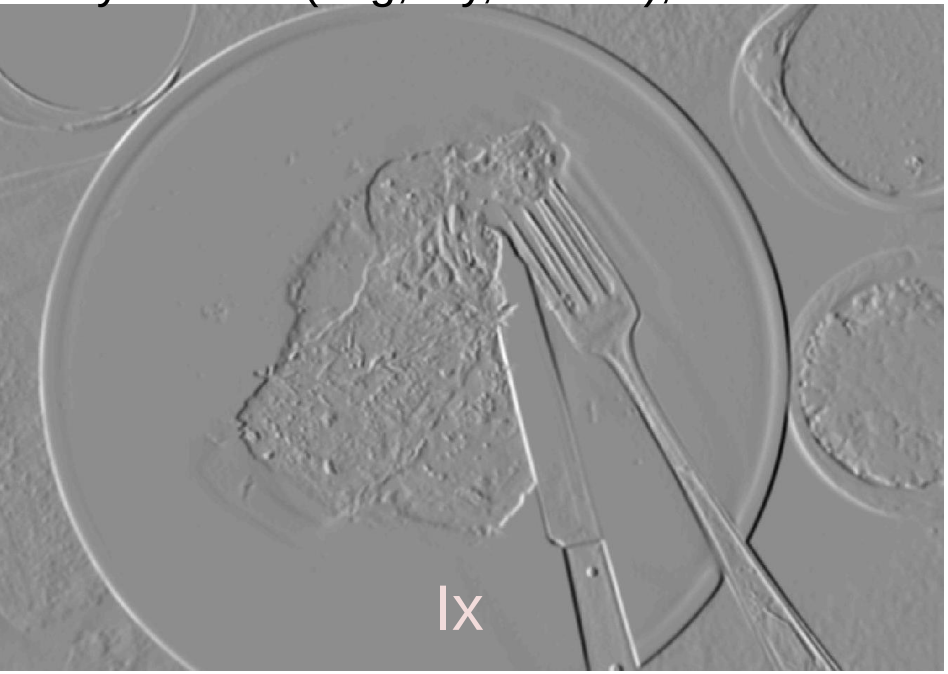
```
low = threshold_low * max(edge(:));
high = threshold_high * max(edge(:));
linked_edge = hysteresis_thresholding(low, high);
```

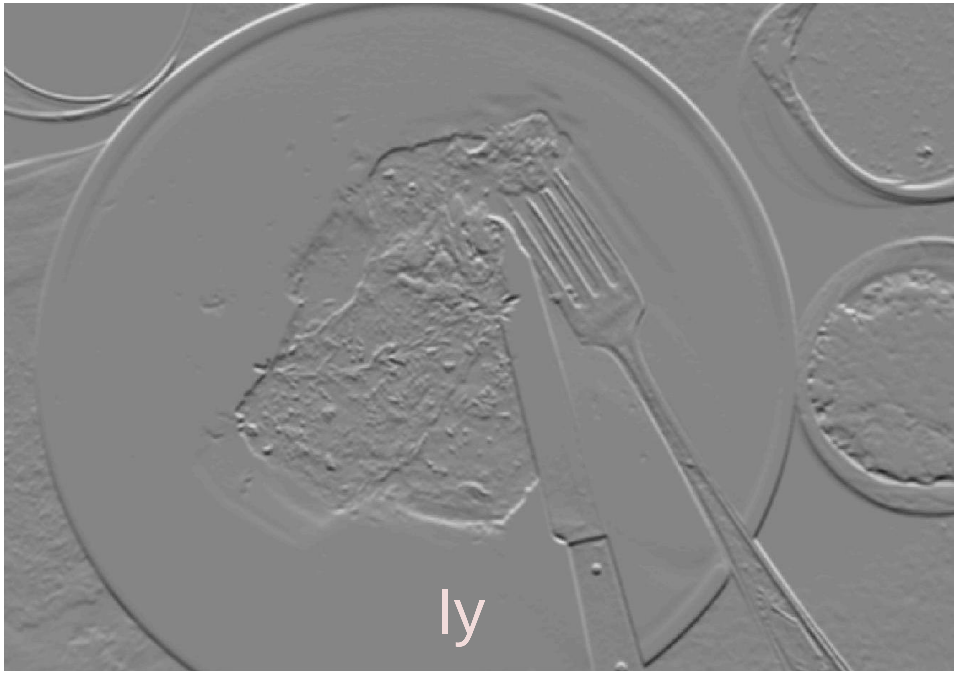




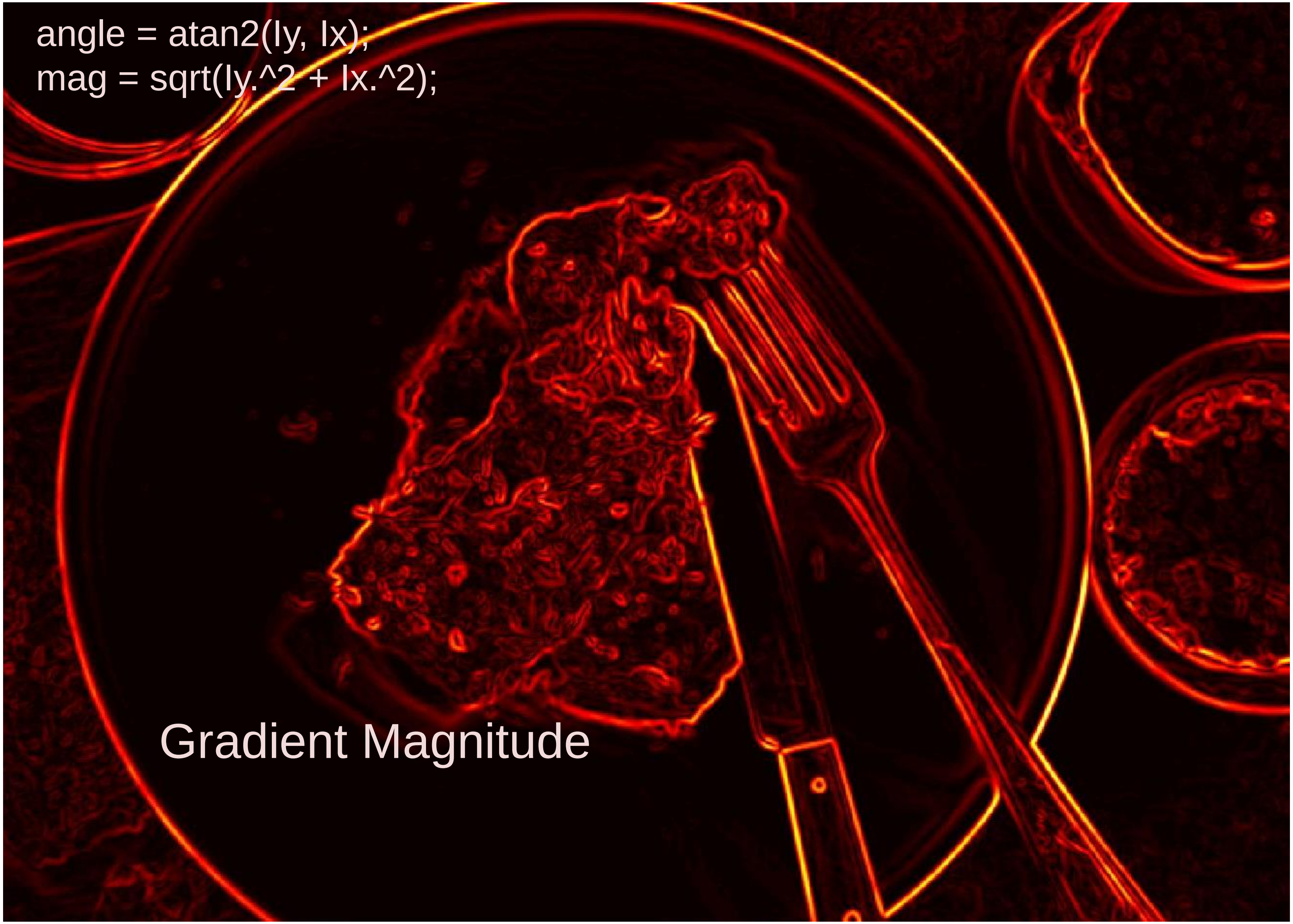
Localized edge

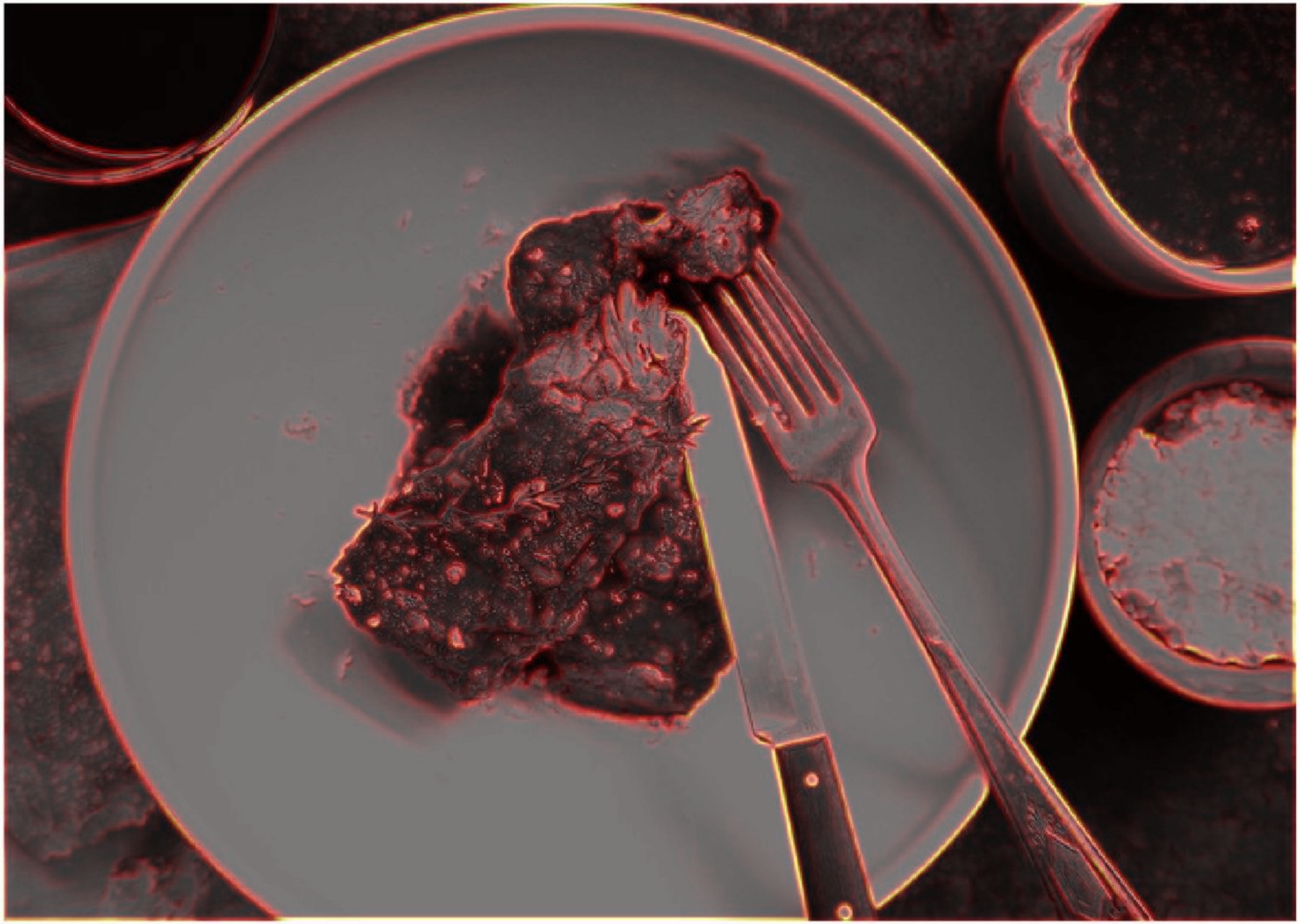
```
% % Convolution of image with Gaussian
Gx = conv2(G, dx, 'same');
Gy = conv2(G, dy, 'same');

% Convolution of image with Gx and Gy
Ix = conv2(img, Gx, 'same');
Iy = conv2(img, Gy, 'same');
```





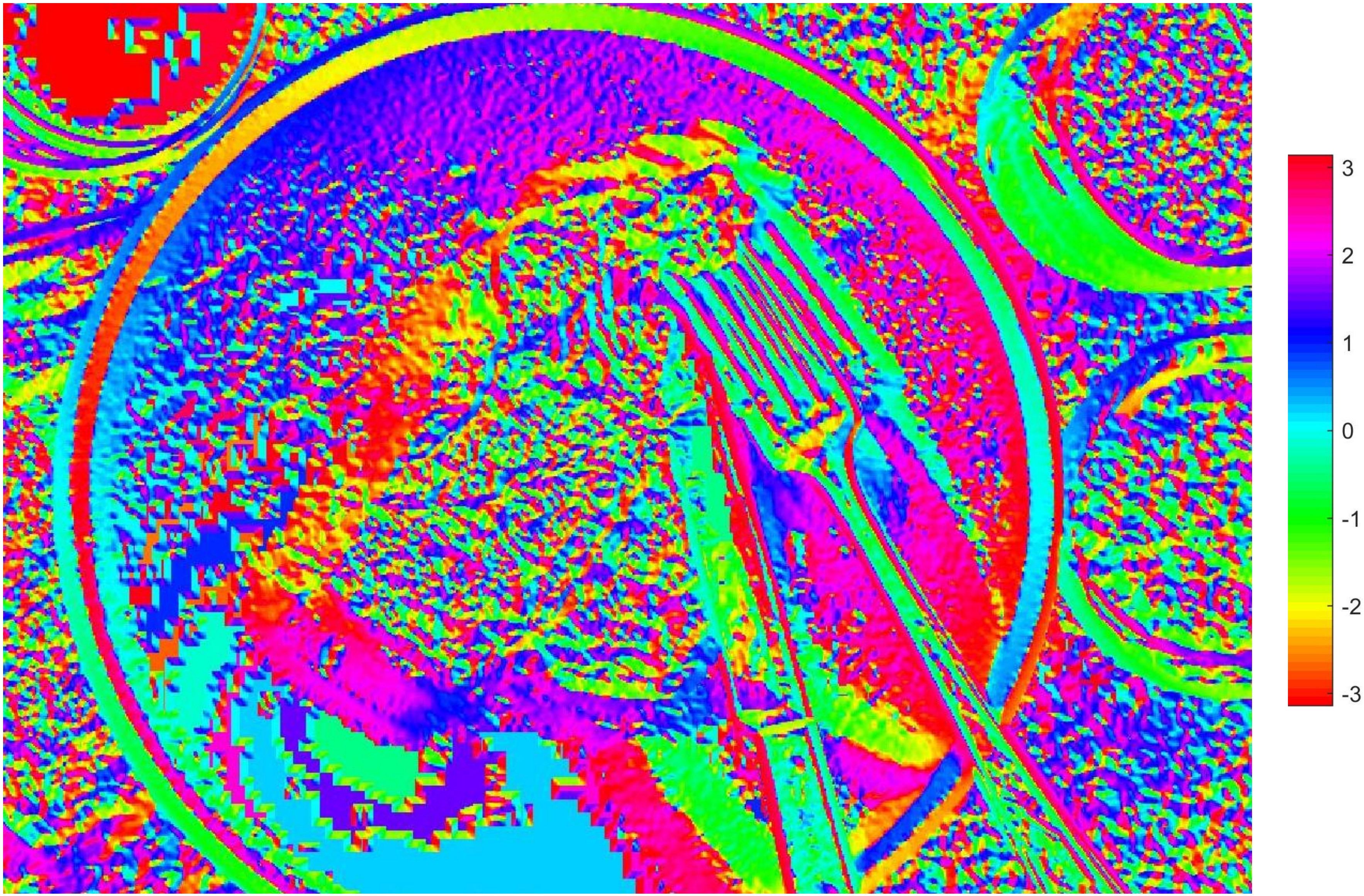
Canny Edge Implementation

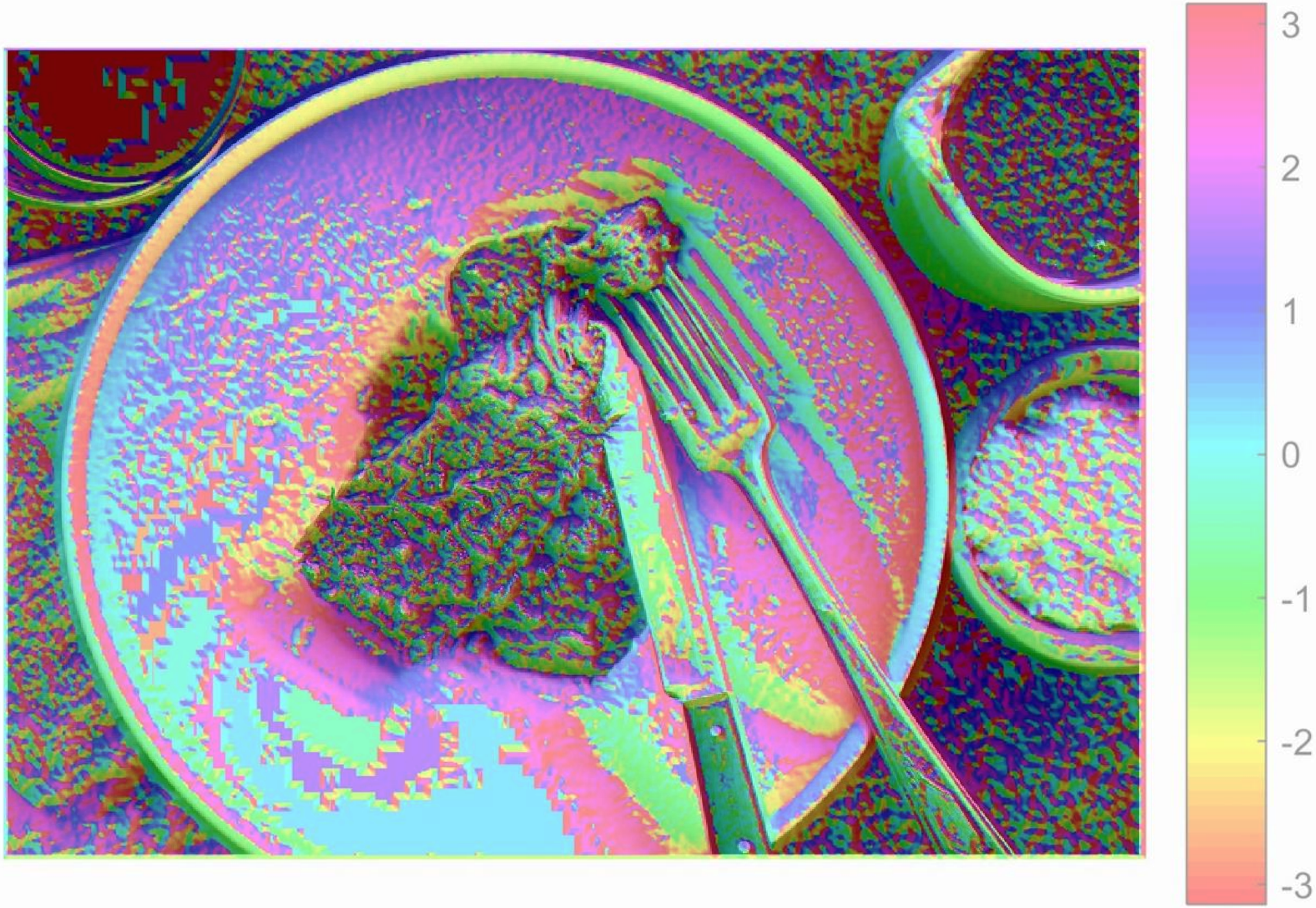




Canny Edge Implementation

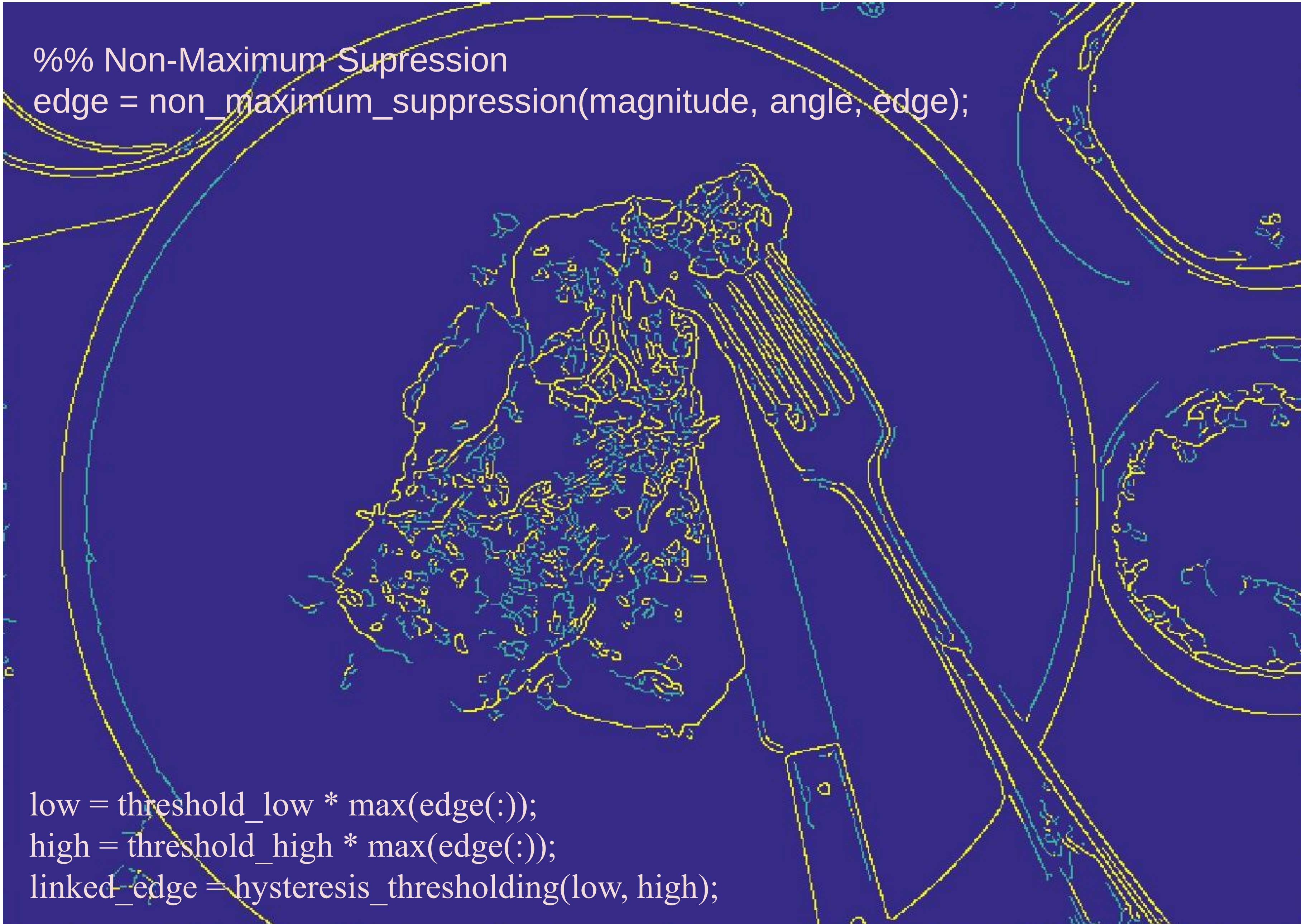
```
angle = atan2(Iy, Ix);
mag = sqrt(Iy.^2 + Ix.^2);
```



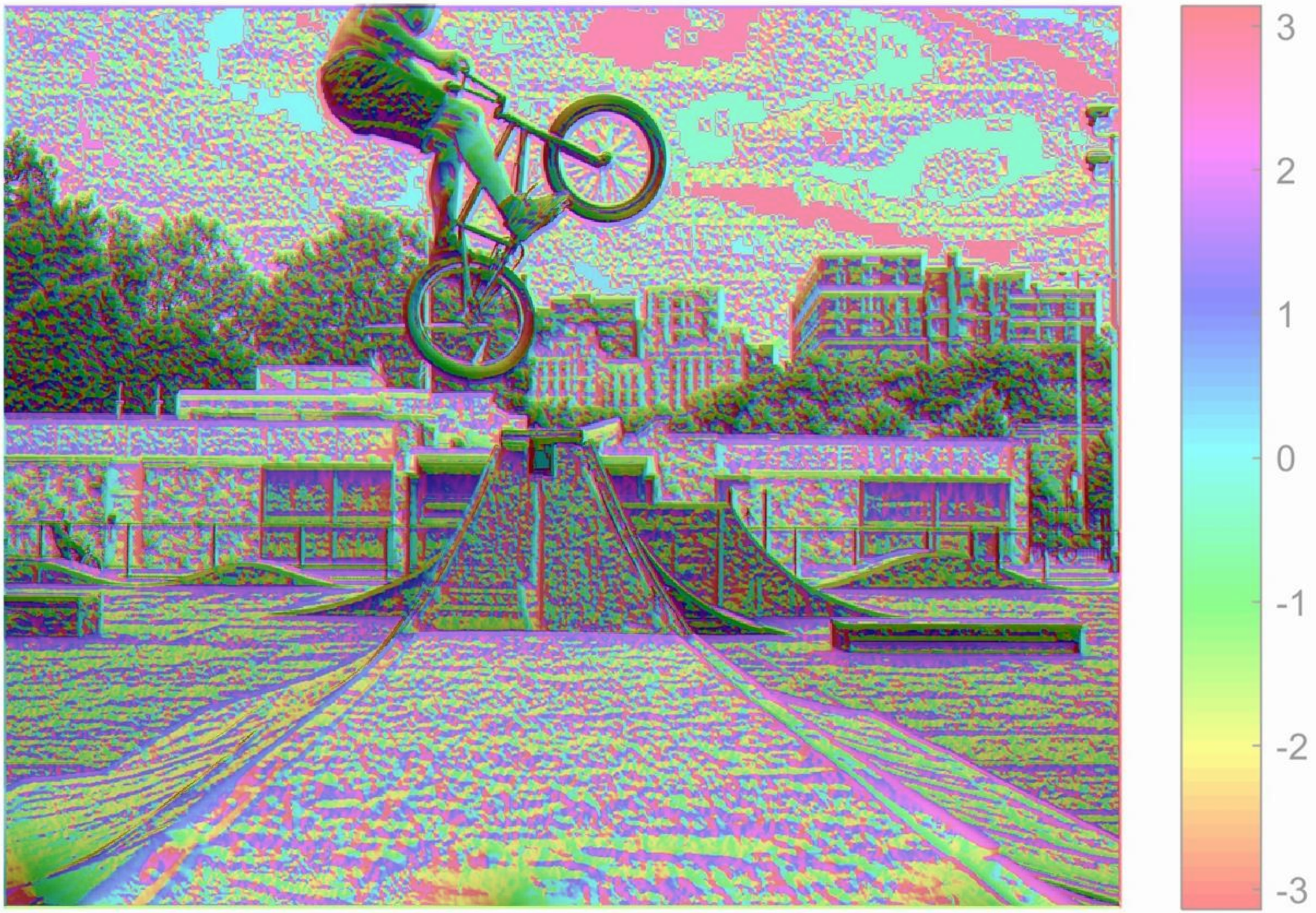

3
2
1
0
-1
-2
-3

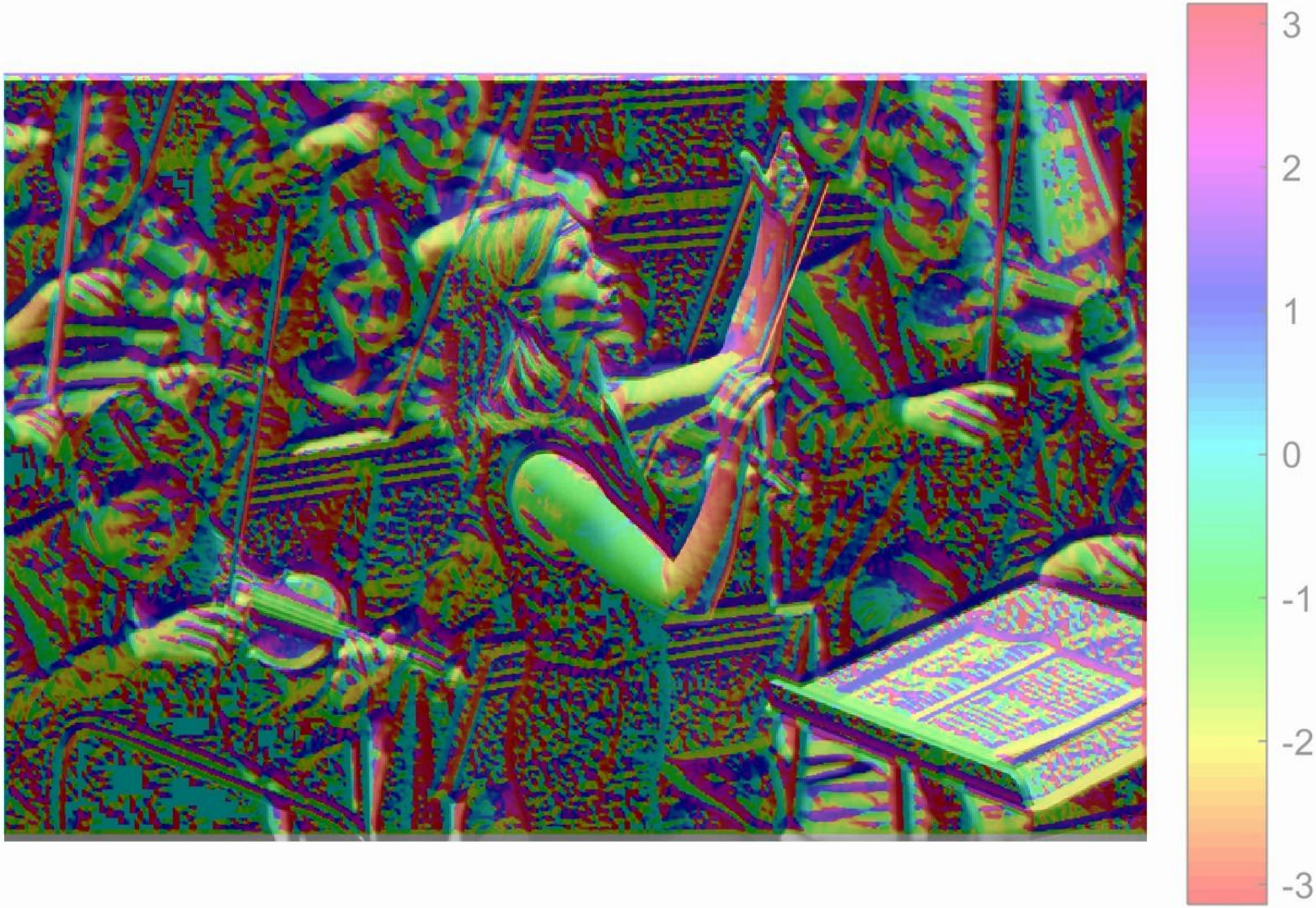
Canny Edge Implementation

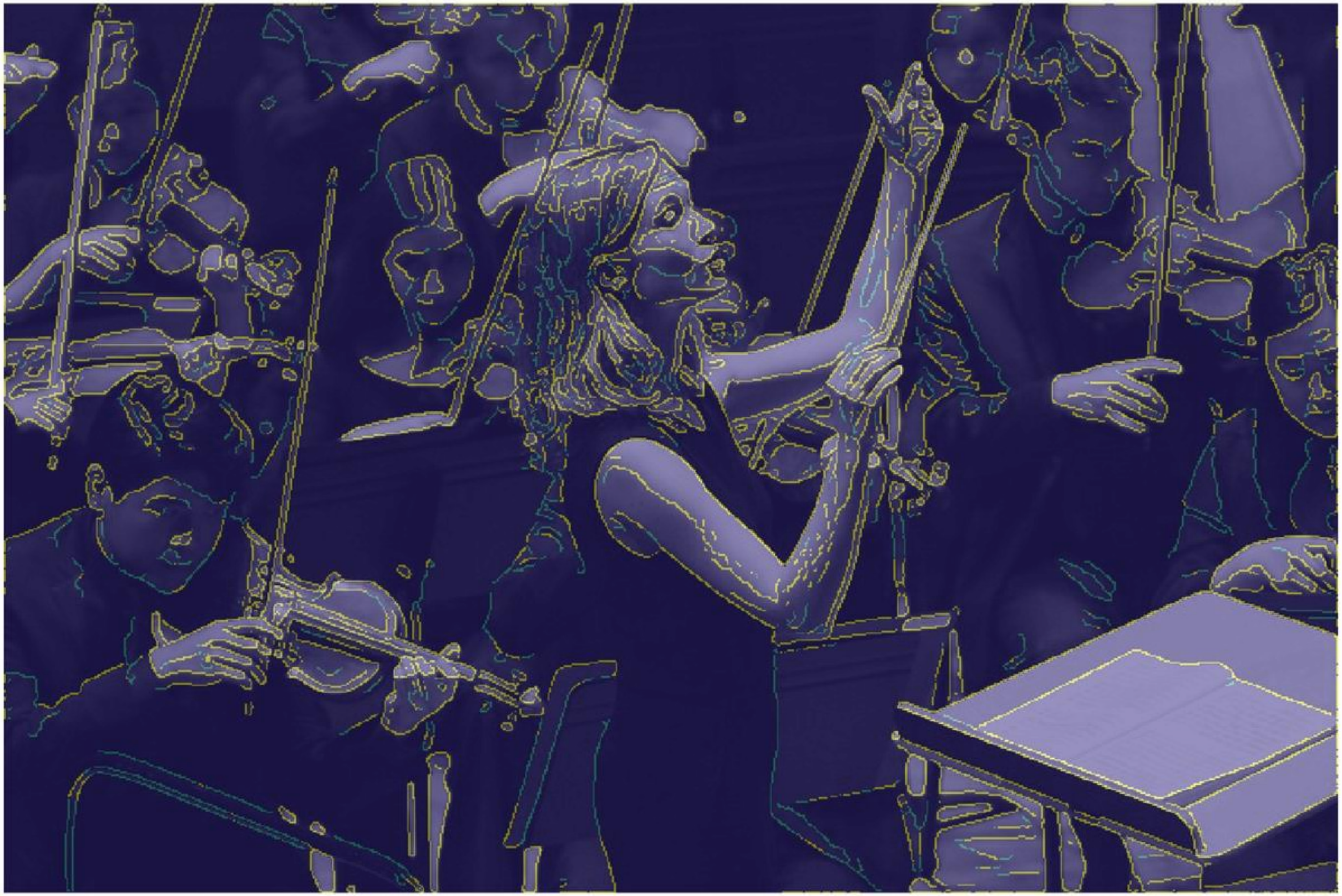
Localized edge

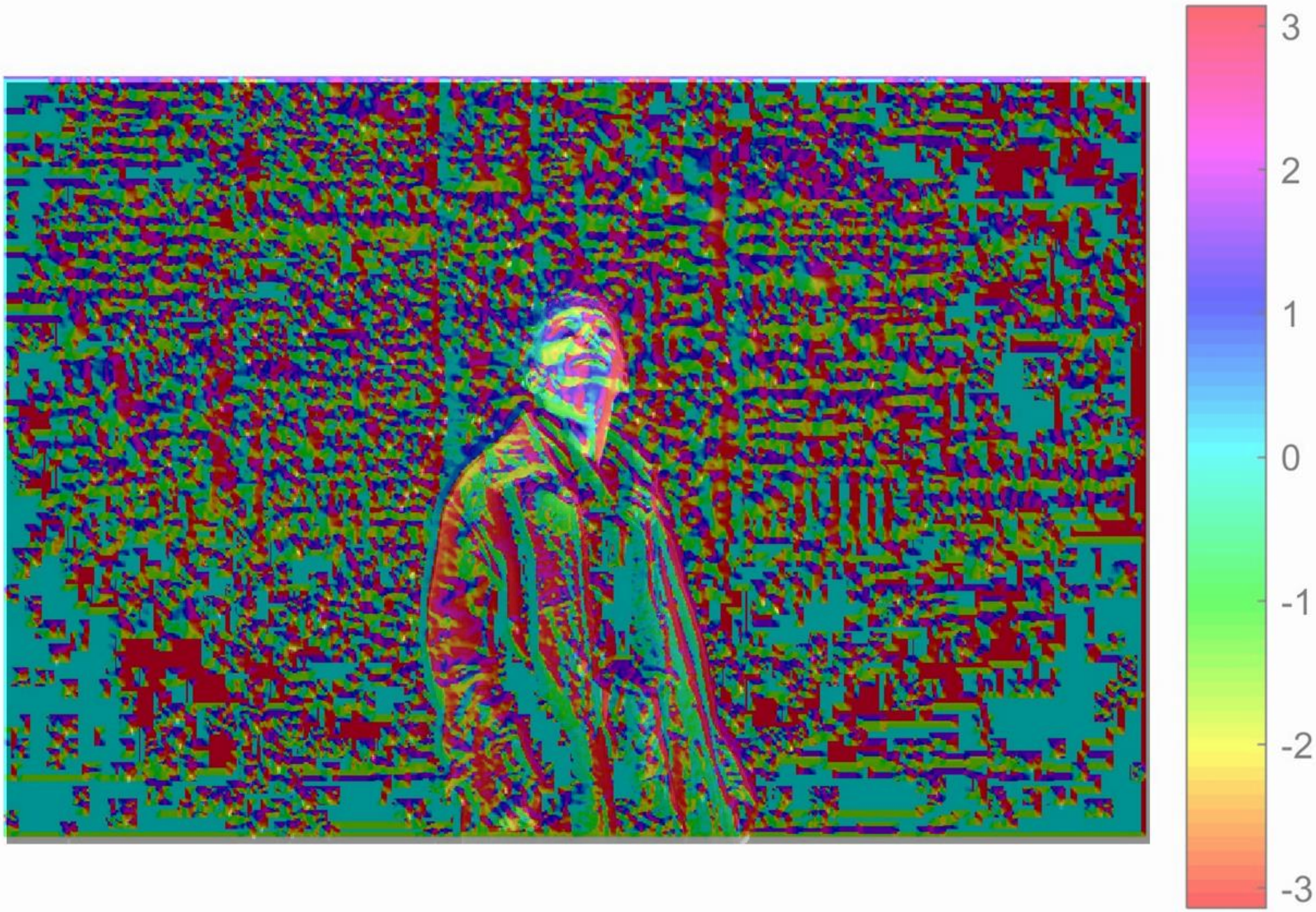


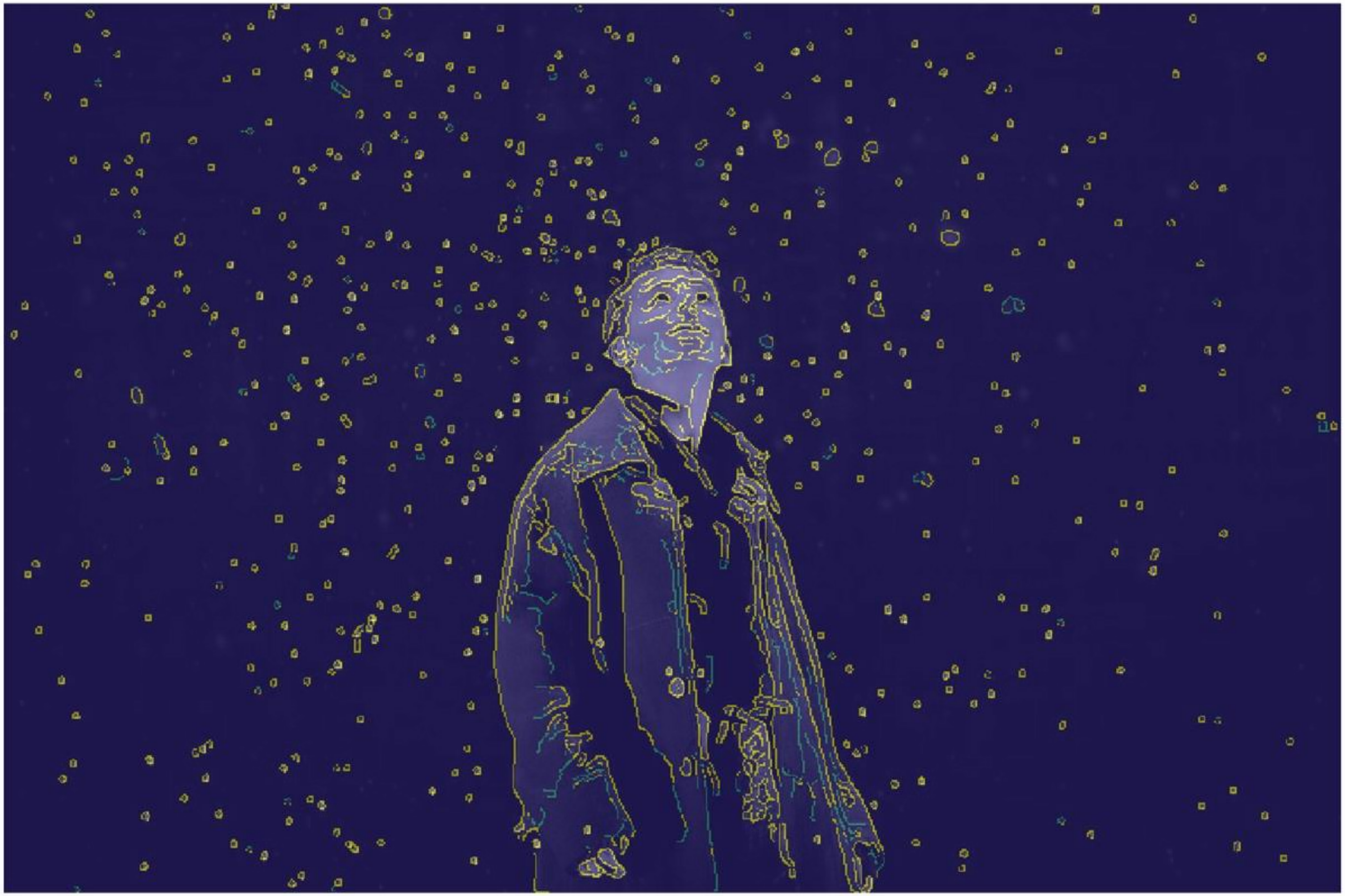
```
%% Non-Maximum Supression
edge = non_maximum_suppression(magnitude, angle, edge);

low = threshold_low * max(edge(:));
high = threshold_high * max(edge(:));
linked_edge = hysteresis_thresholding(low, high);
```


3
2
1
0
-1
-2
-3


3
2
1
0
-1
-2
-3

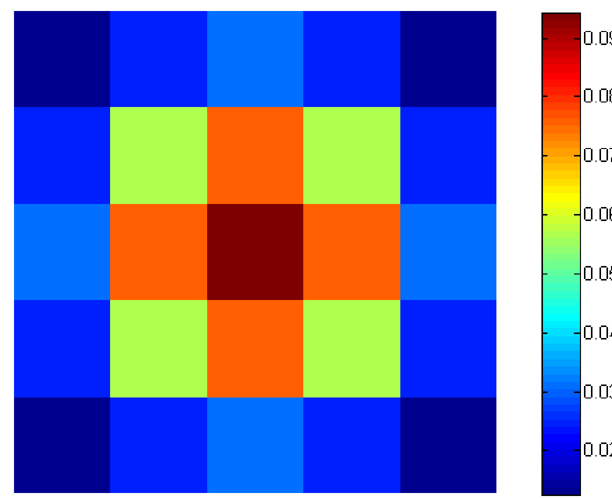



3
2
1
0
-1
-2
-3



Canny Edge Implementation

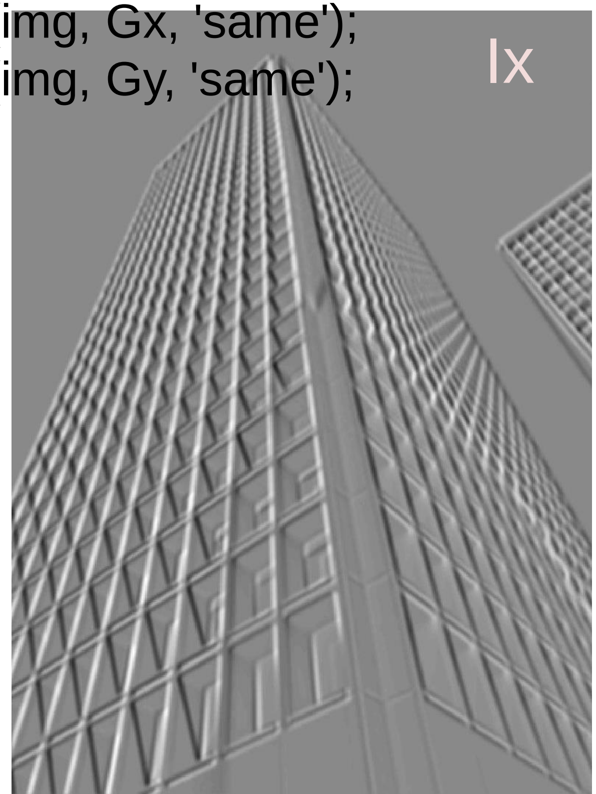
```
img = imread ('image.png');
img = rgb2gray(img);
img = double (img);

% Value for high and low thresholding
threshold_low = 0.035;
threshold_high = 0.175;

%% Gaussian filter (https://en.wikipedia.org/wiki/Canny_edge_detector)
G = [2, 4, 5, 4, 2; 4, 9, 12, 9, 4;5, 12, 15, 12, 5;4, 9, 12, 9, 4;2, 4, 5, 4, 2];
G = 1/159.* G;

%Filter for horizontal and vertical direction
dx = [1 -1];
dy = [1; -1];
```

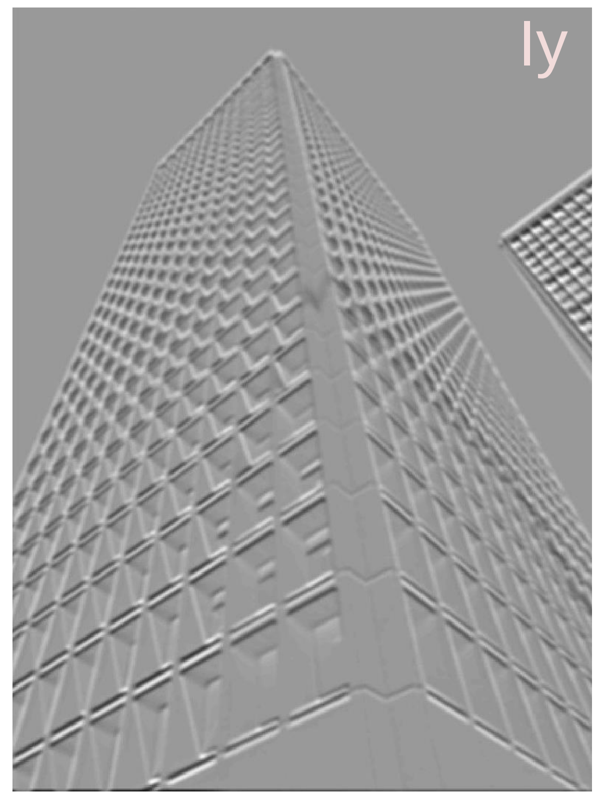


Canny Edge Implementation

% % Convolution of image with Gaussian
Gx = conv2(G, dx, ‘full');
Gy = conv2(G, dy, ‘full');

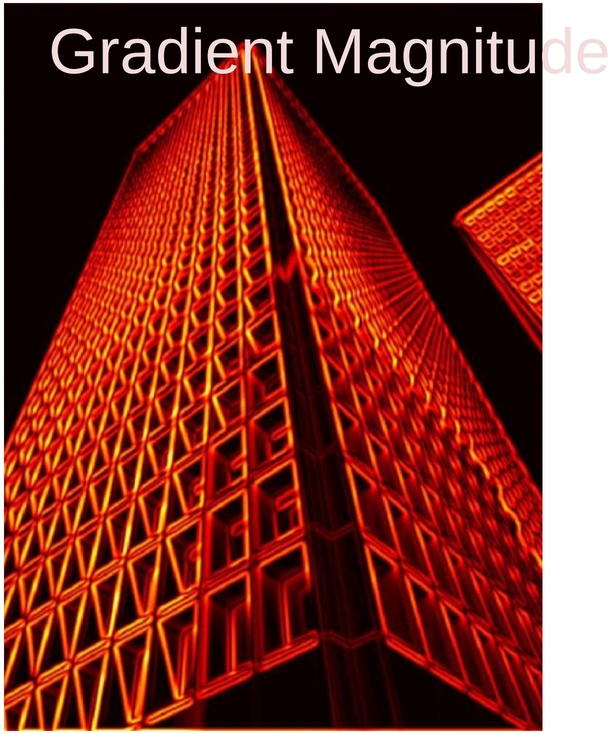
% Convolution of image with Gx and Gy
Ix = conv2(img, Gx, 'same');
Iy = conv2(img, Gy, 'same');

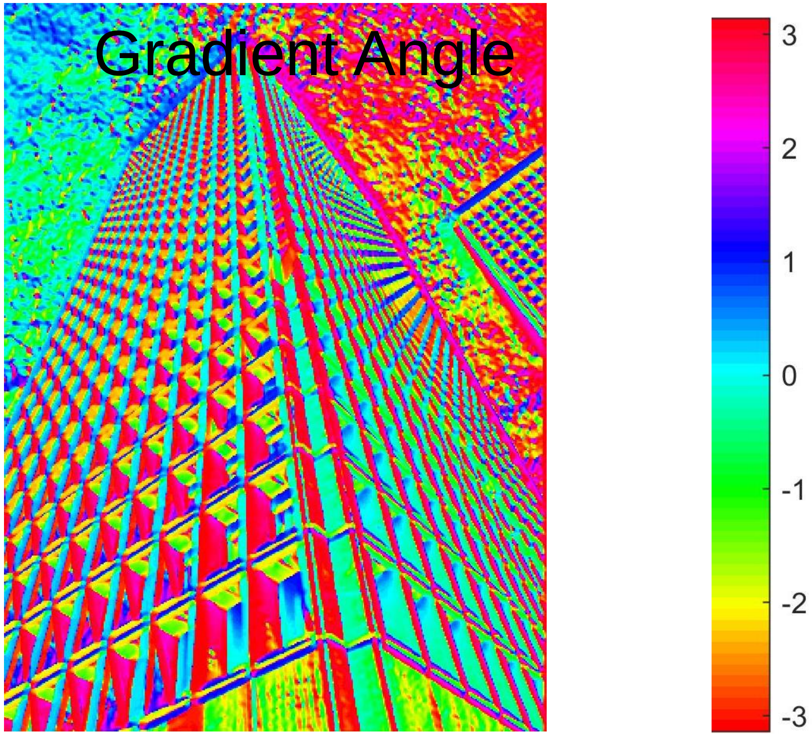


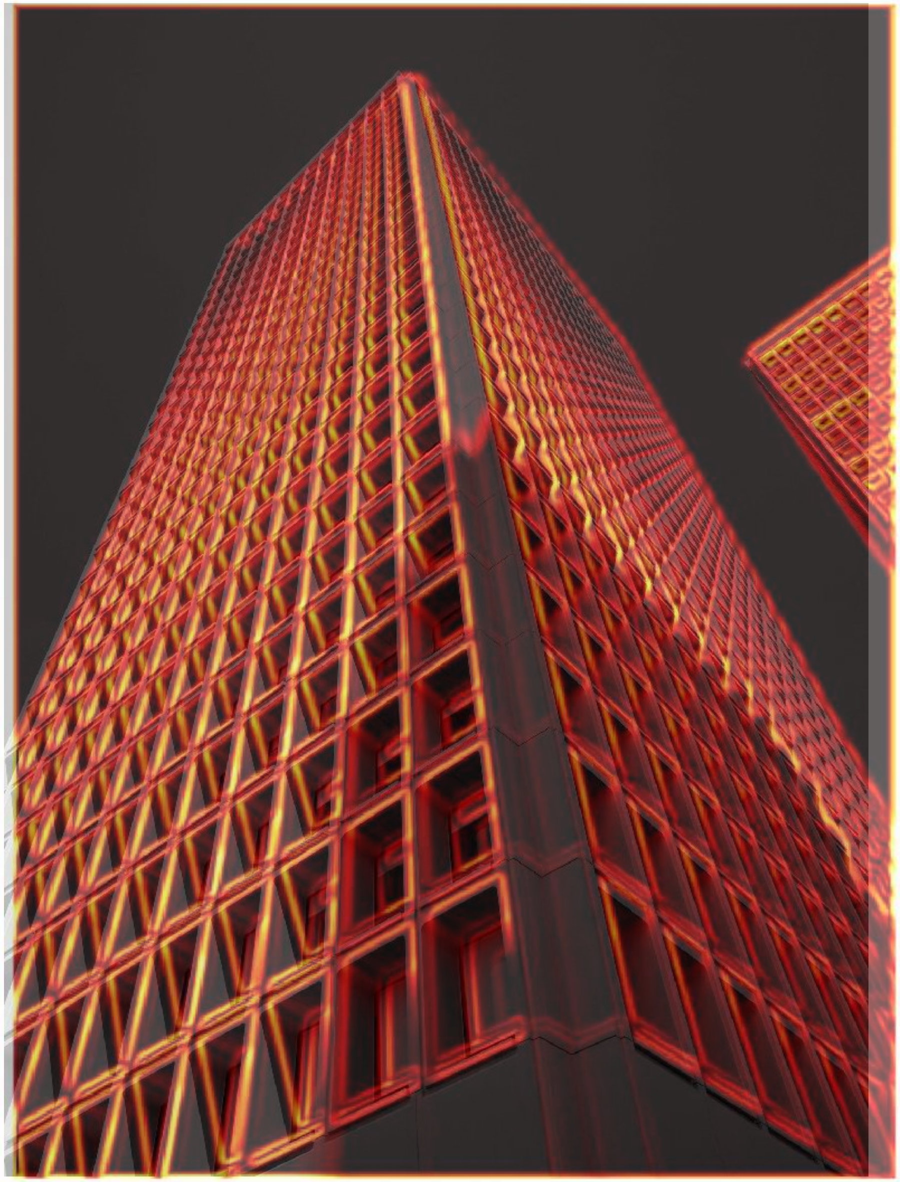


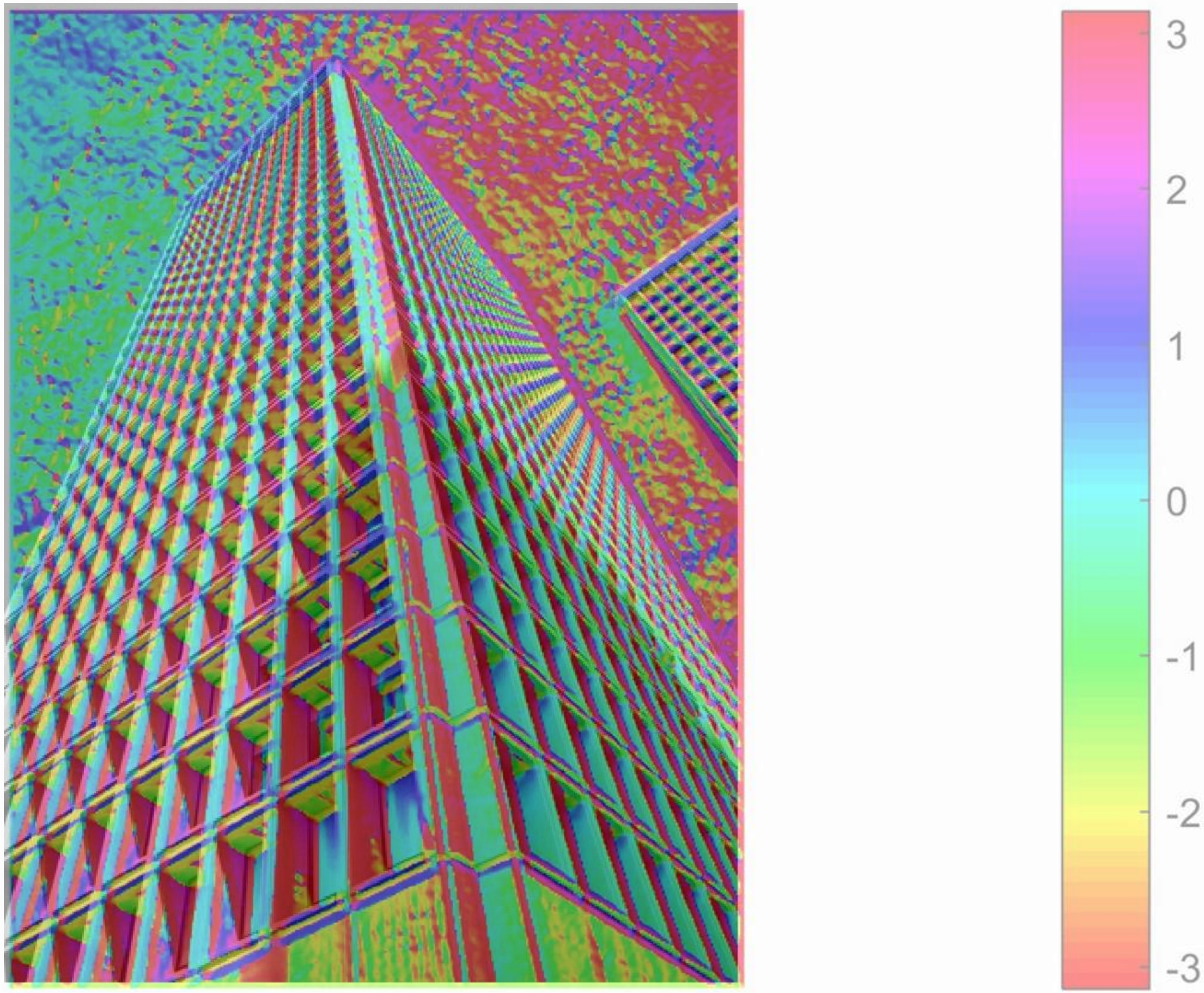
Canny Edge Implementation

```
angle = atan2(Iy, Ix);
mag = sqrt(Iy.^2 + Ix.^2);
```

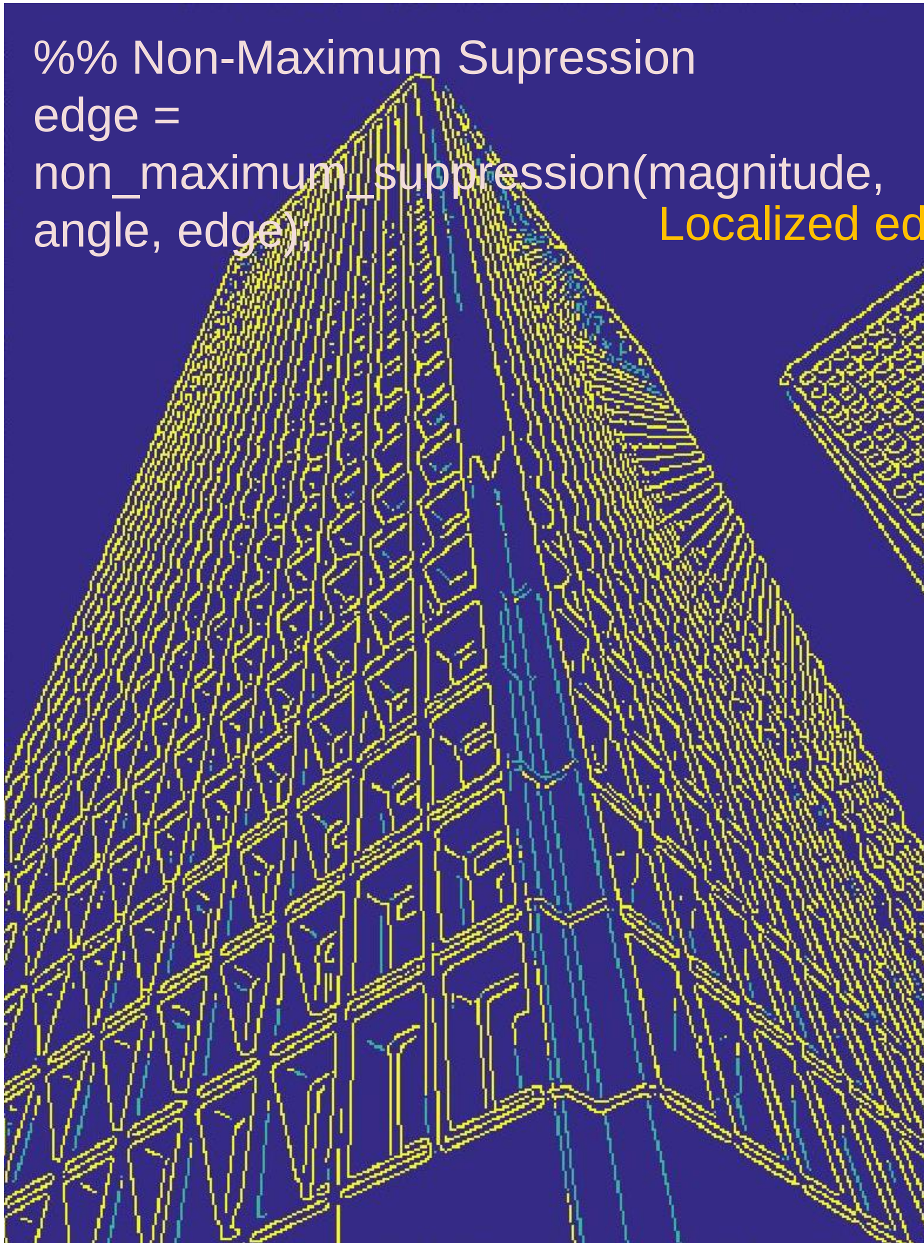


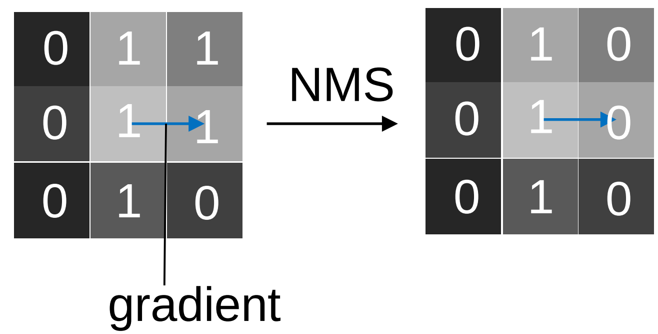


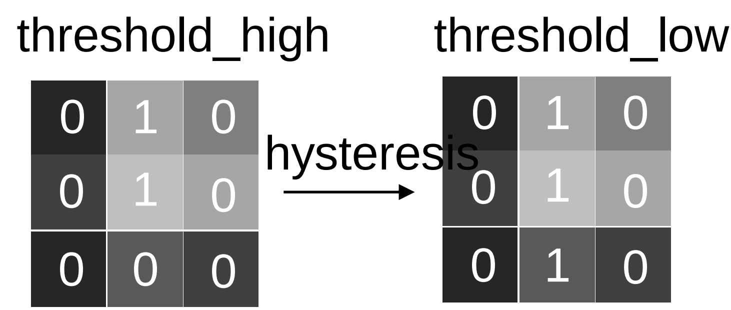



3
2
1
0
-1
-2
-3

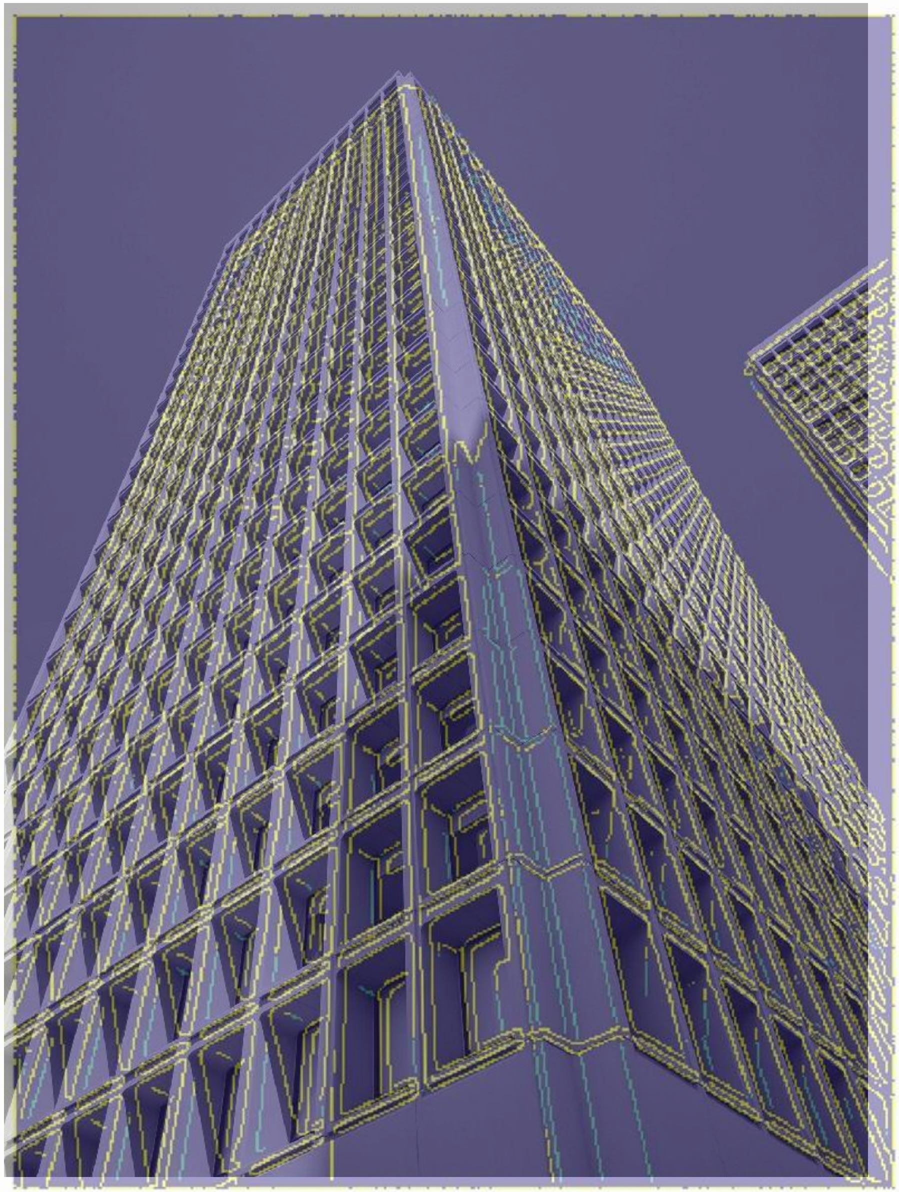
Canny Edge Implementation

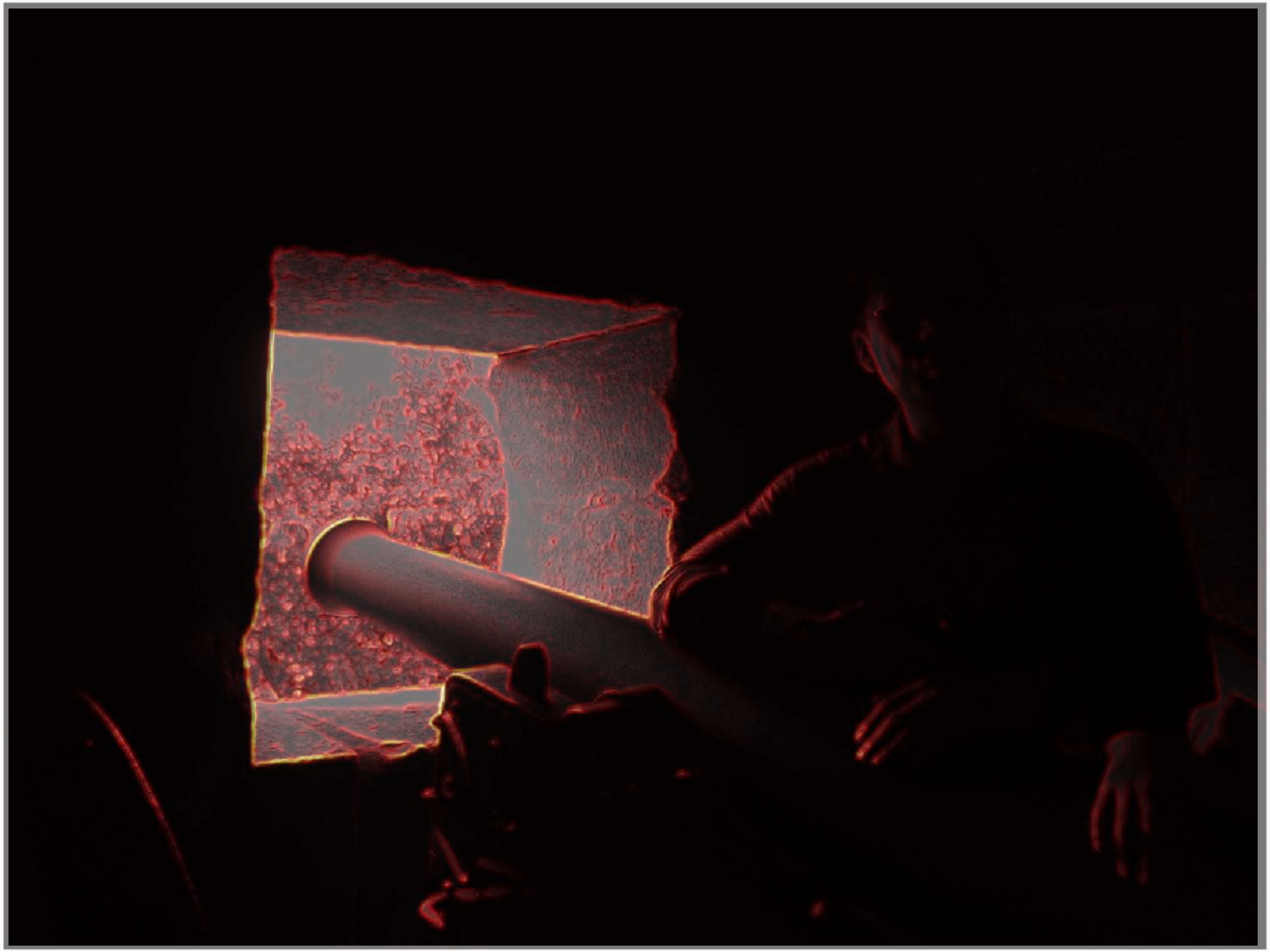


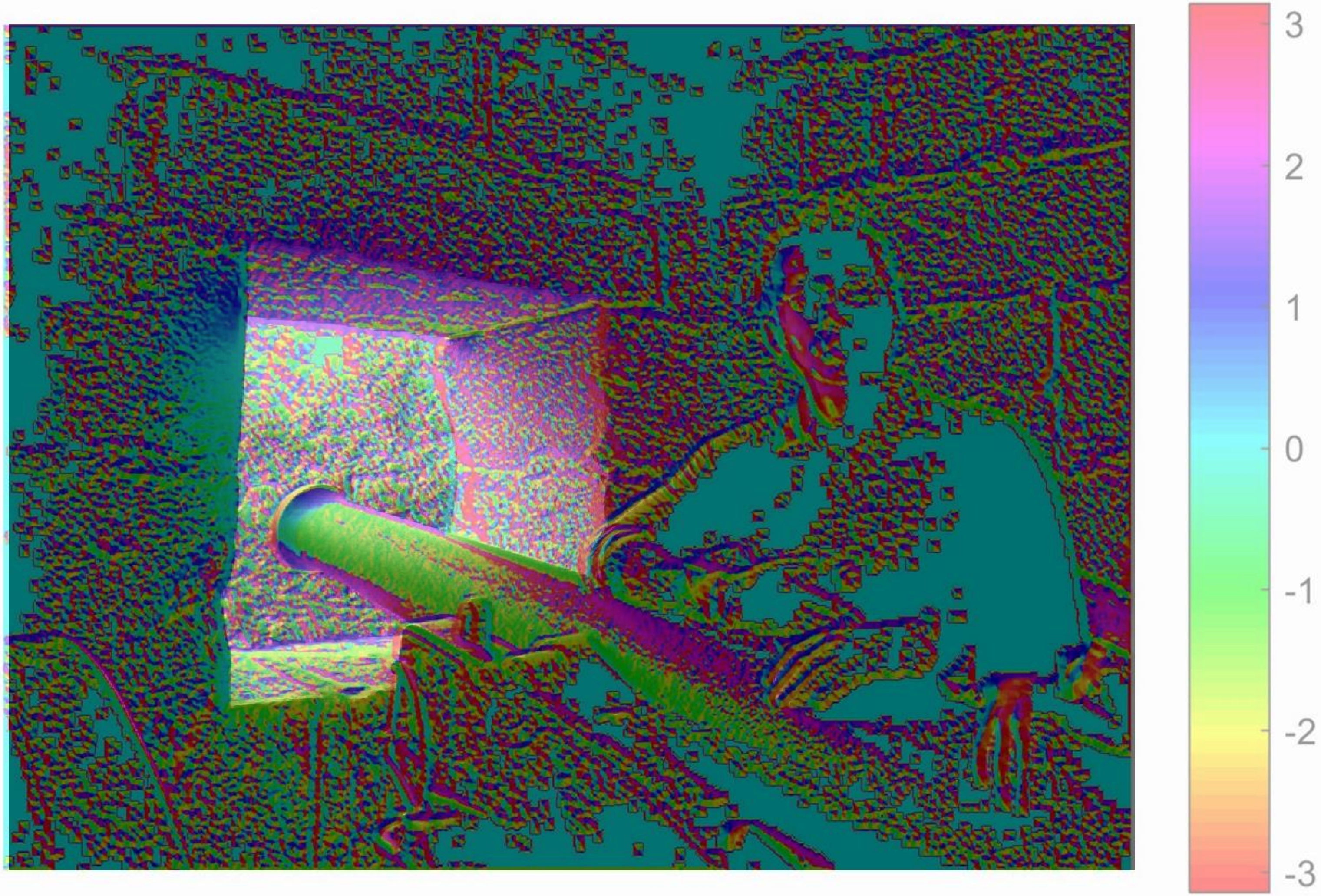


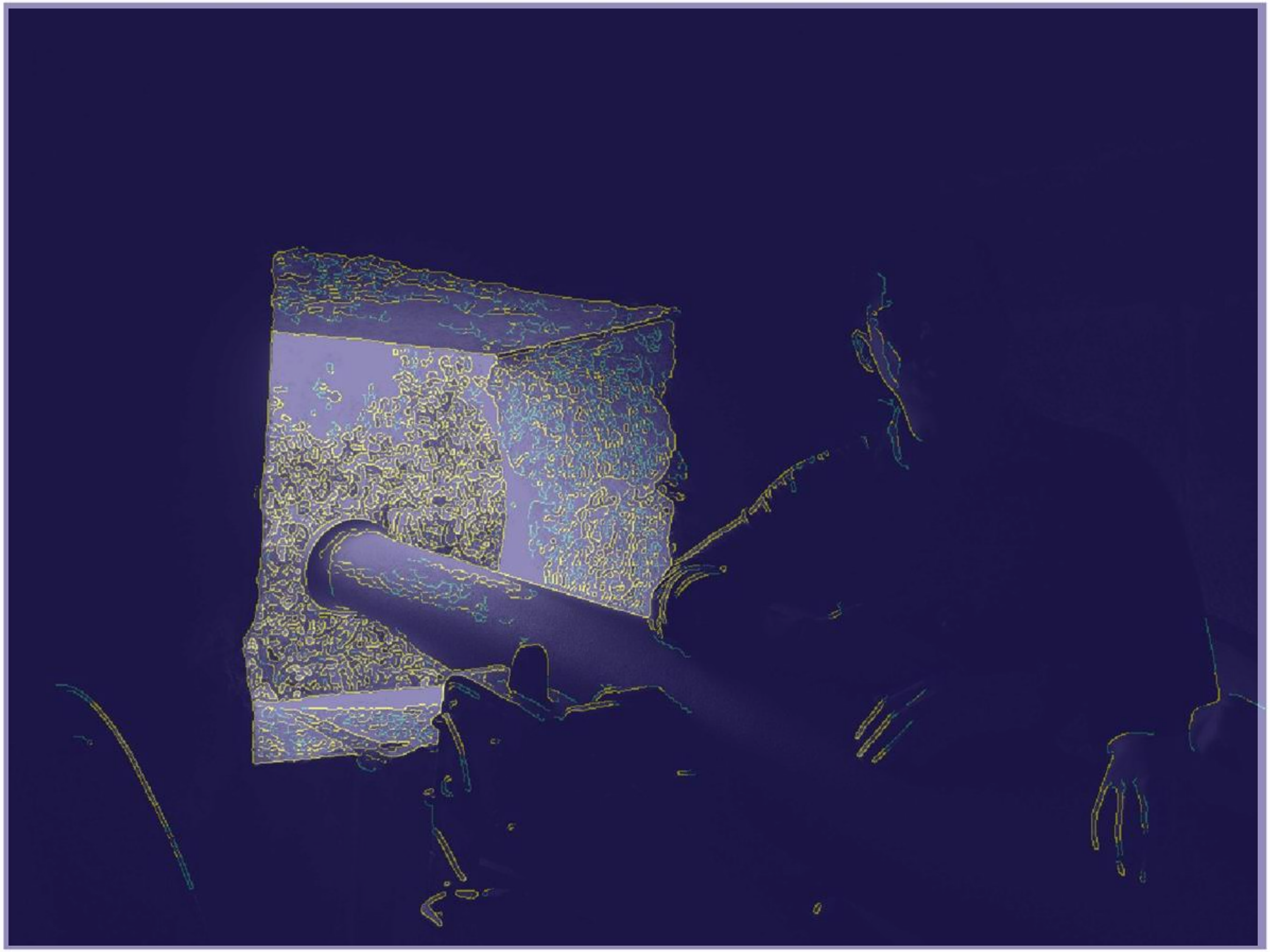



```
low = threshold_low * max(edge(:));
high = threshold_high * max(edge(:));
linked_edge = hysteresis_thresholding(low, high);
```



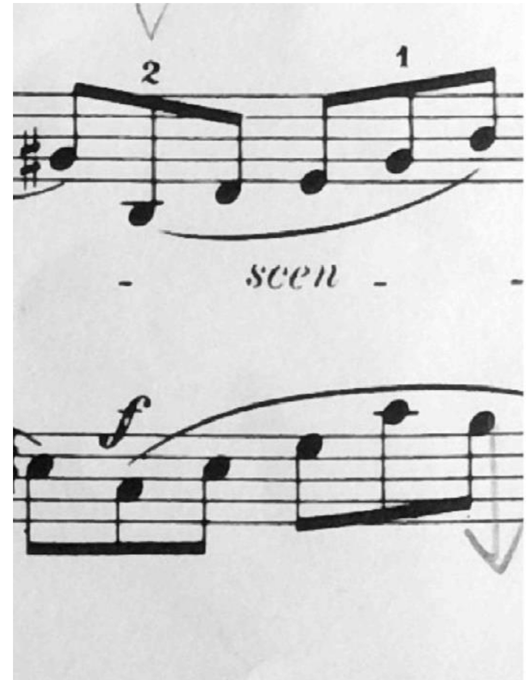
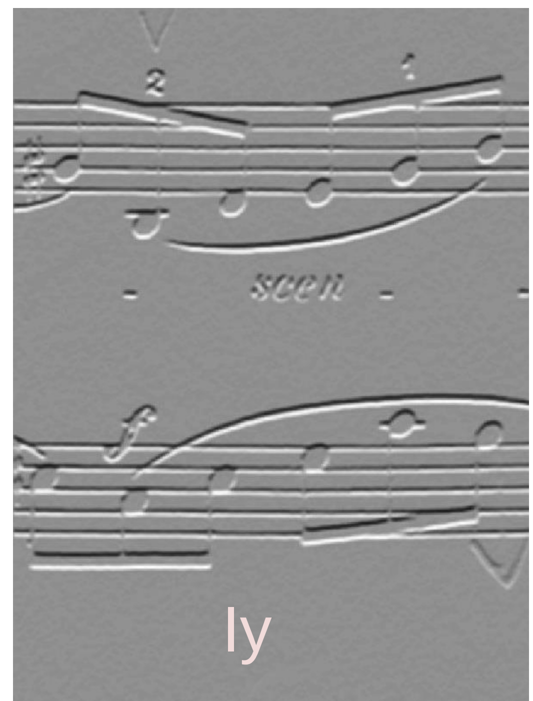


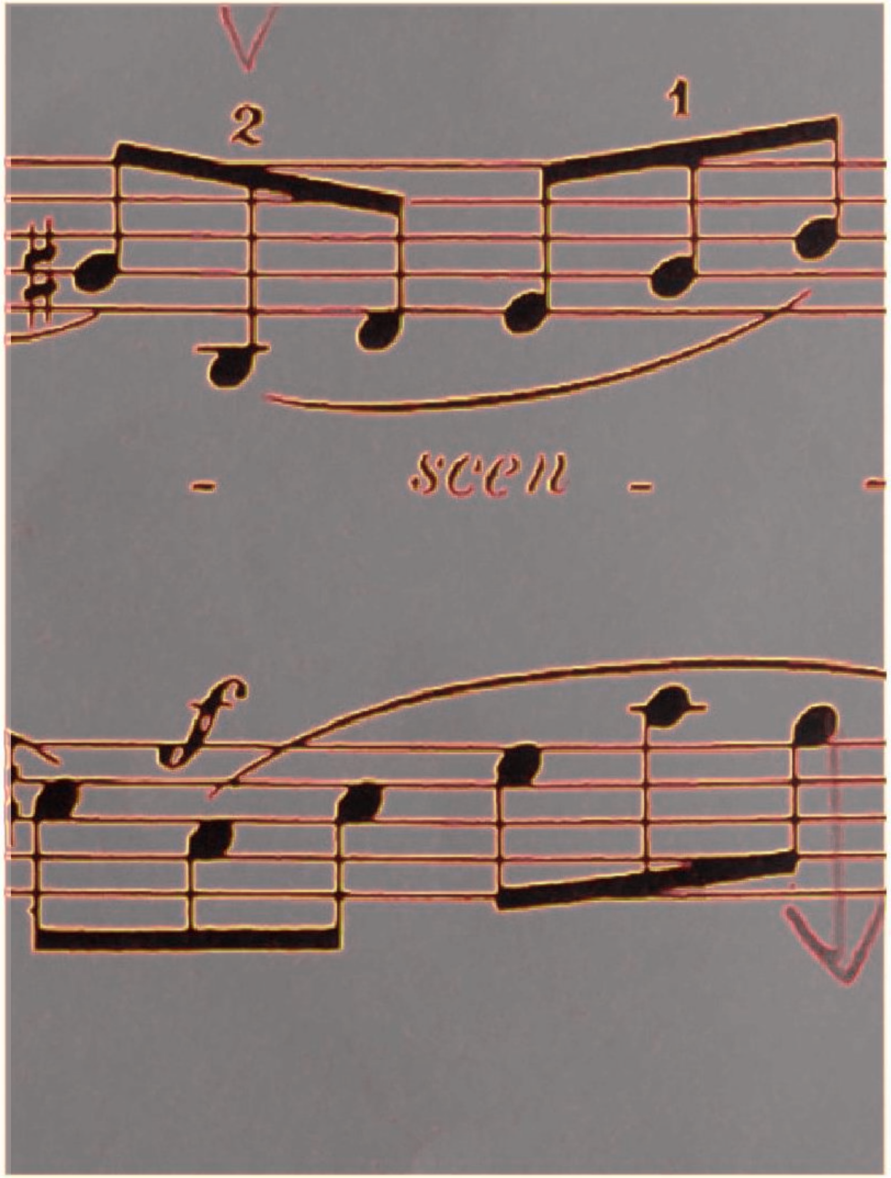

3
2
1
0
-1
-2
-3



Ix

Iy

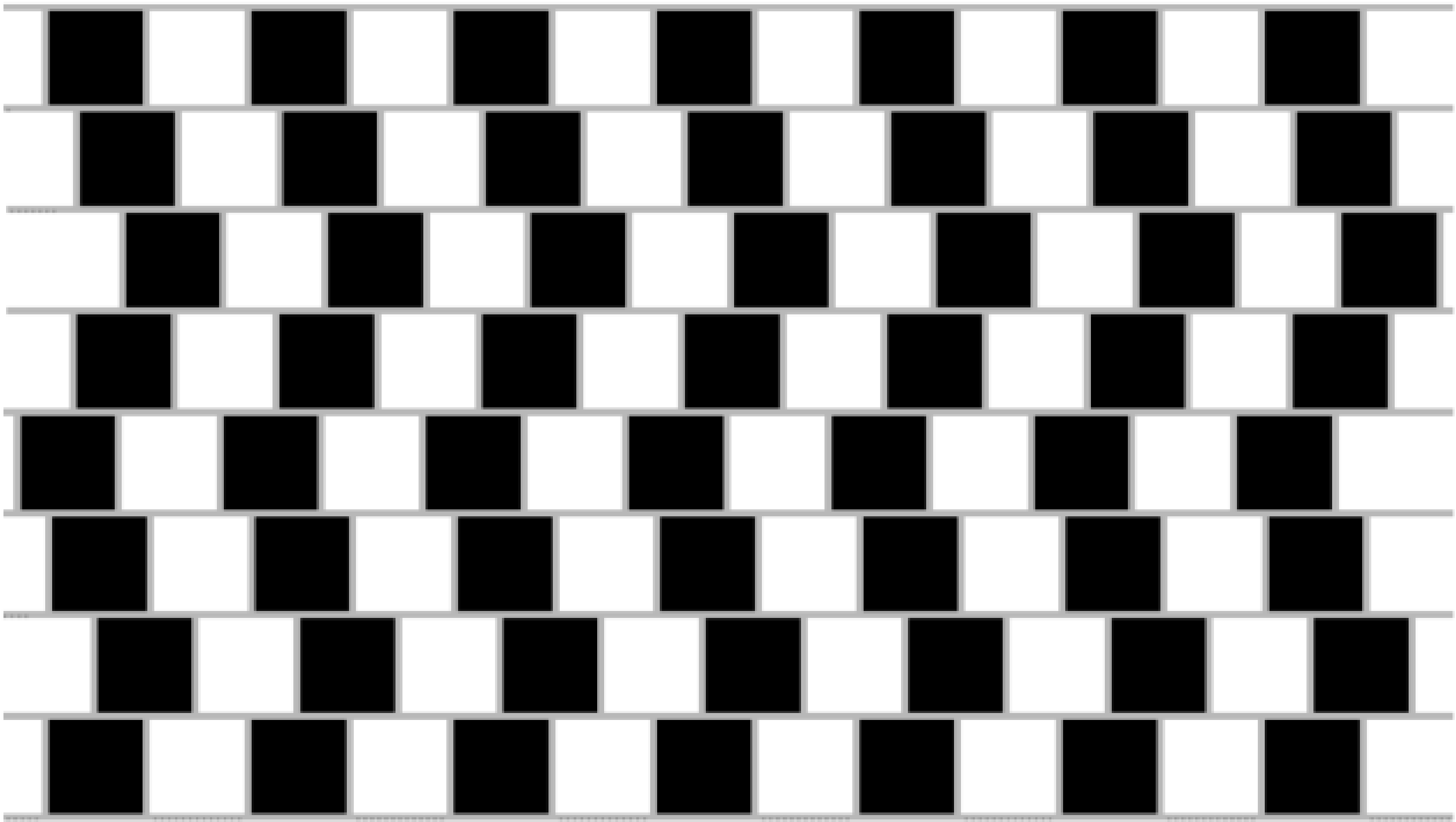
```
% % Convolution of image with Gaussian
Gx = conv2(G, dx, ‘full');
Gy = conv2(G, dy, ‘full');

% Convolution of image with Gx and Gy
Ix = conv2(img, Gx, 'same');
Iy = conv2(img, Gy, 'same');
```




2
1
- scen - -
f


2
1
- seen - -
f
3
2
1
0
-1
-2
-3

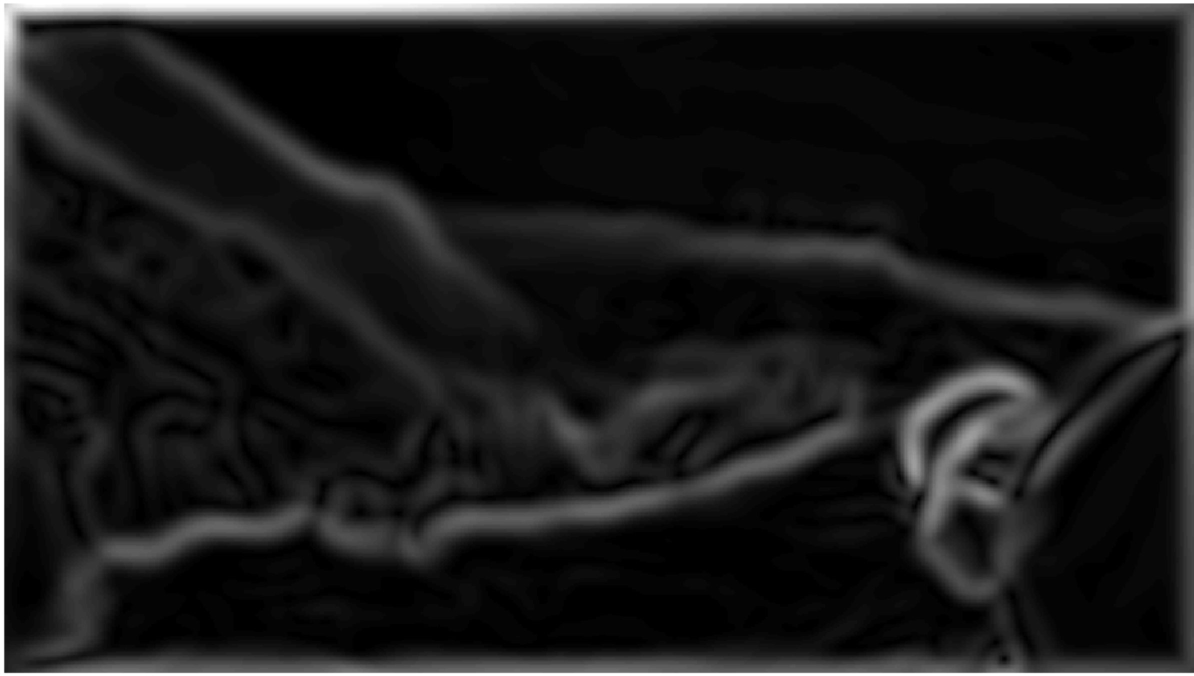
http://www.cfar.umd.edu/~fer/optical/index.html

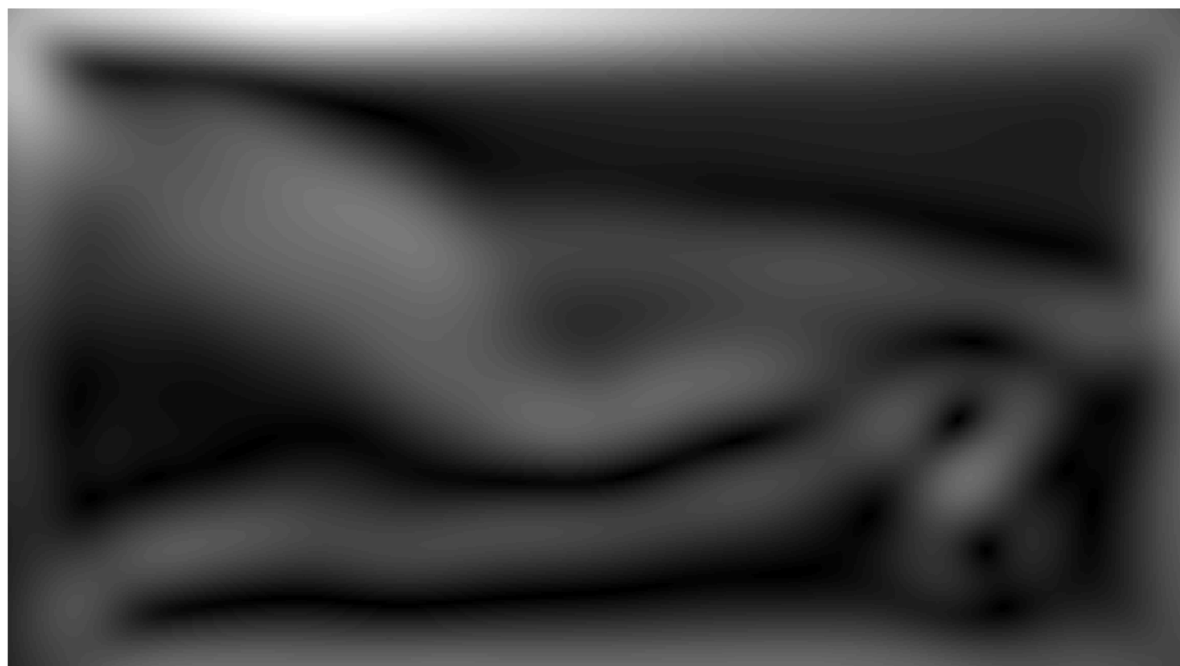
The rows of black and white squares are all parallel.
The vertical zigzag patterns disrupt our horizontal perception.

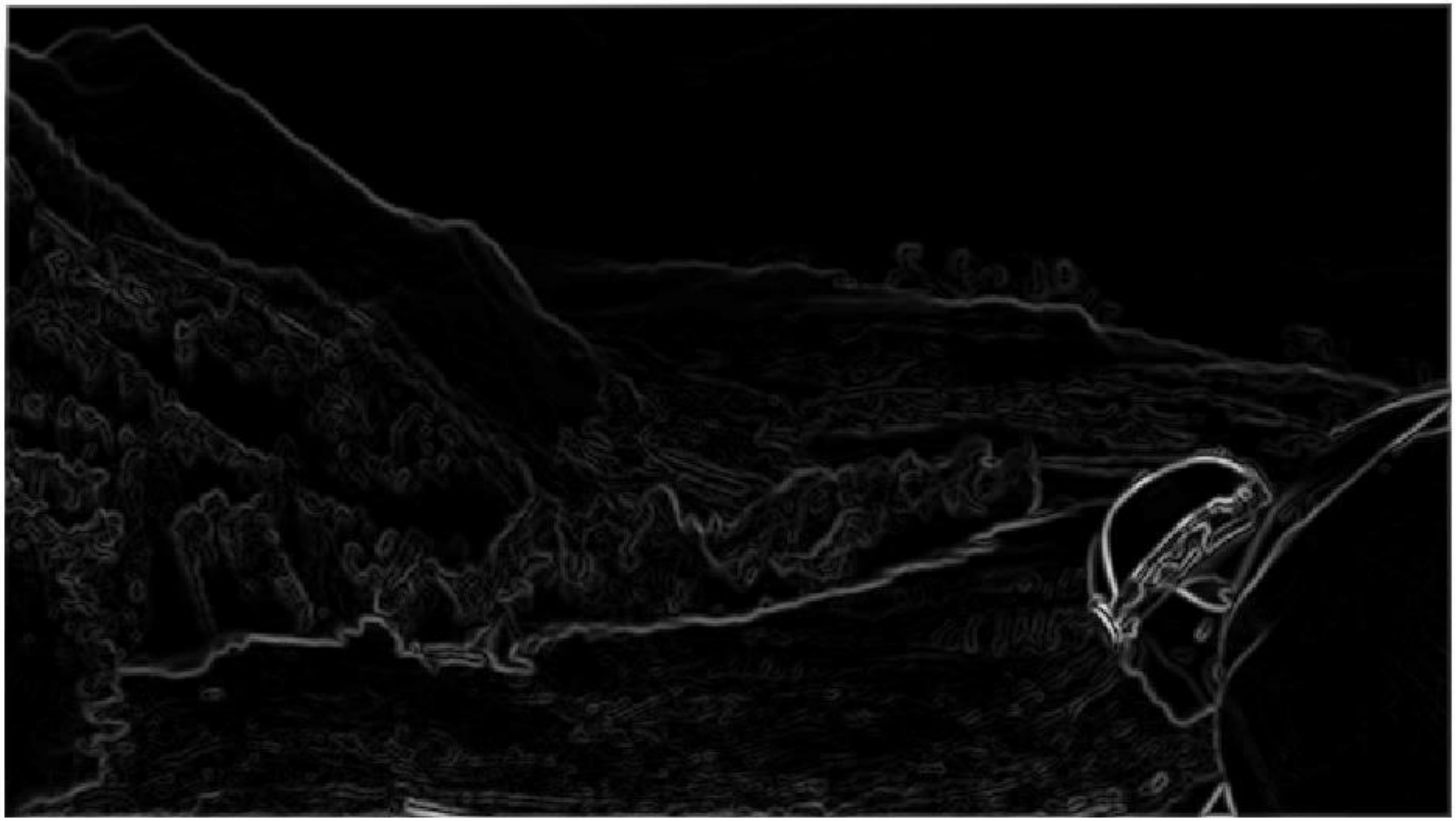


Cornelia Fermüller

Image Scale



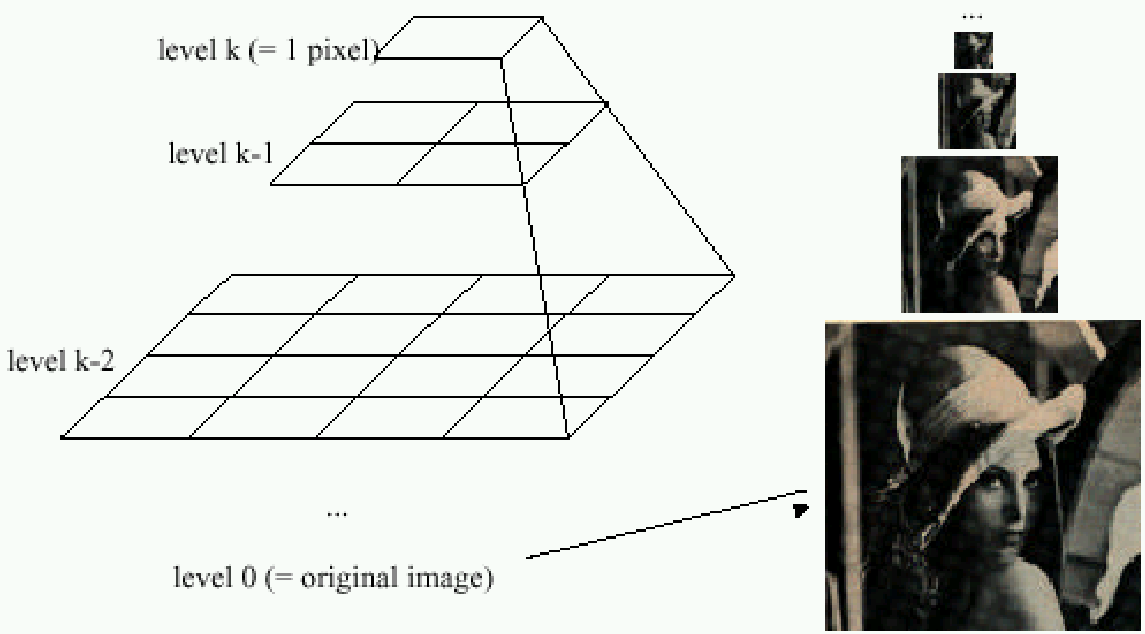




Different scale of image encodes different edge response.

Image Pyramids

Idea: Represent NxN image as a “pyramid” of $1\text{x}1, 2\text{x}2, 4\text{x}4, \ldots, 2^k\text{x}2^k$ images (assuming $N=2^k$)



Known as a Gaussian Pyramid [Burt and Adelson, 1983]

- In computer graphics, a *mip map* [Williams, 1983]
- A precursor to *wavelet transform*

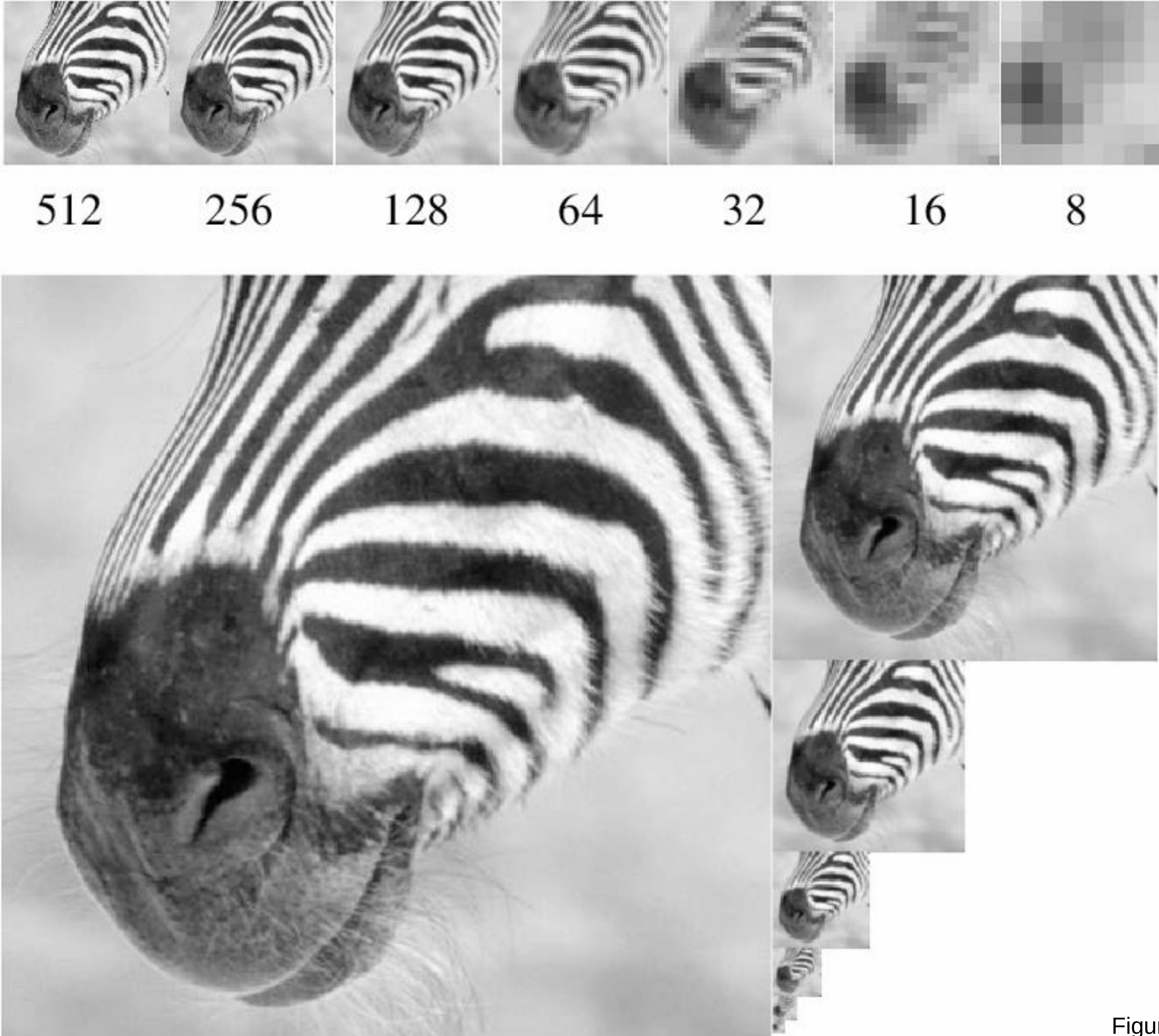


Figure from David Forsyth