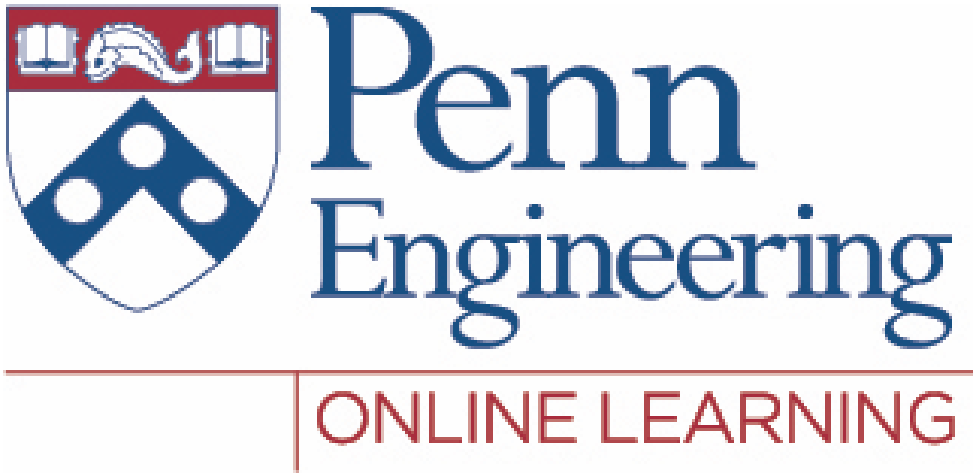
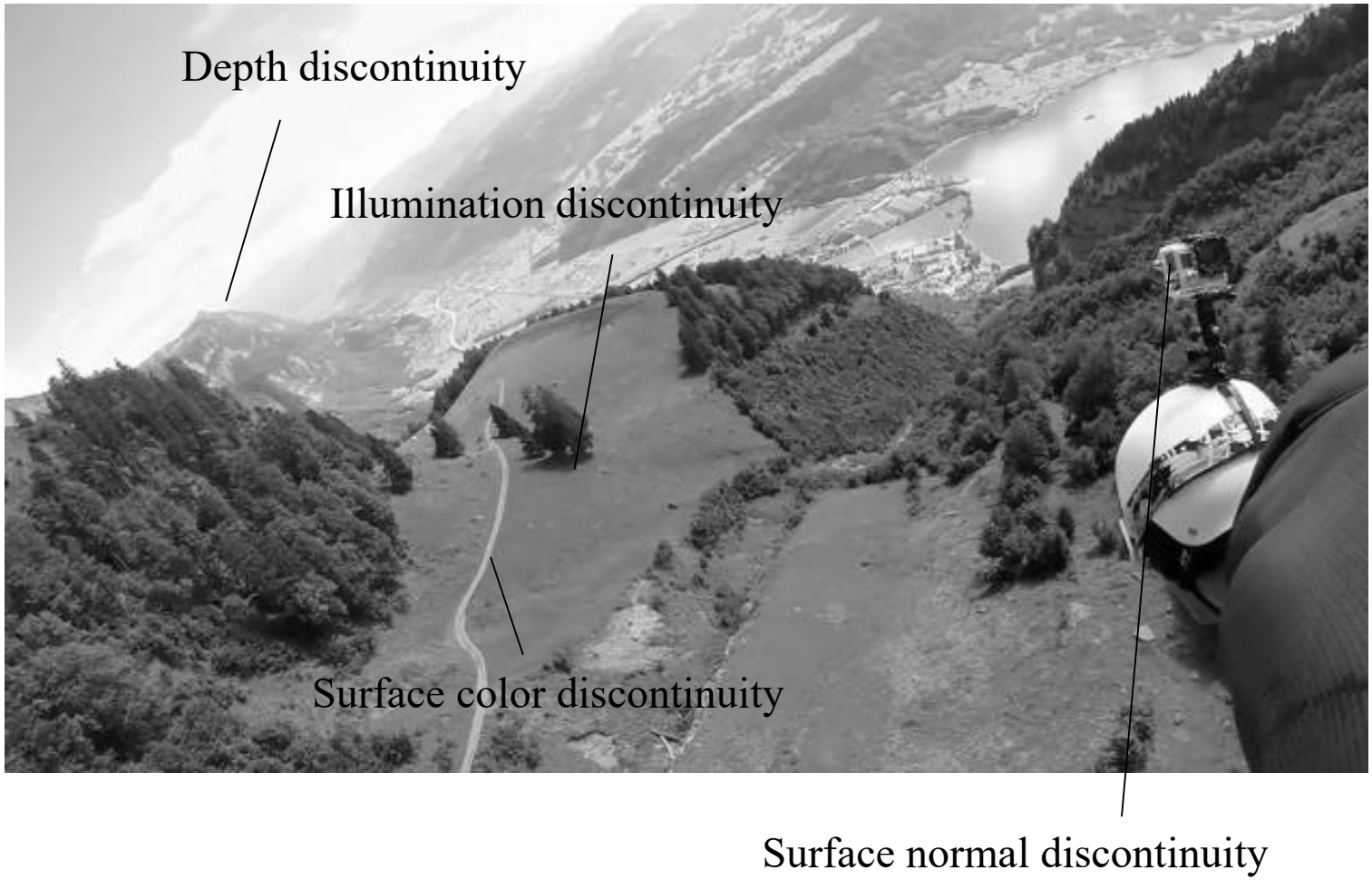


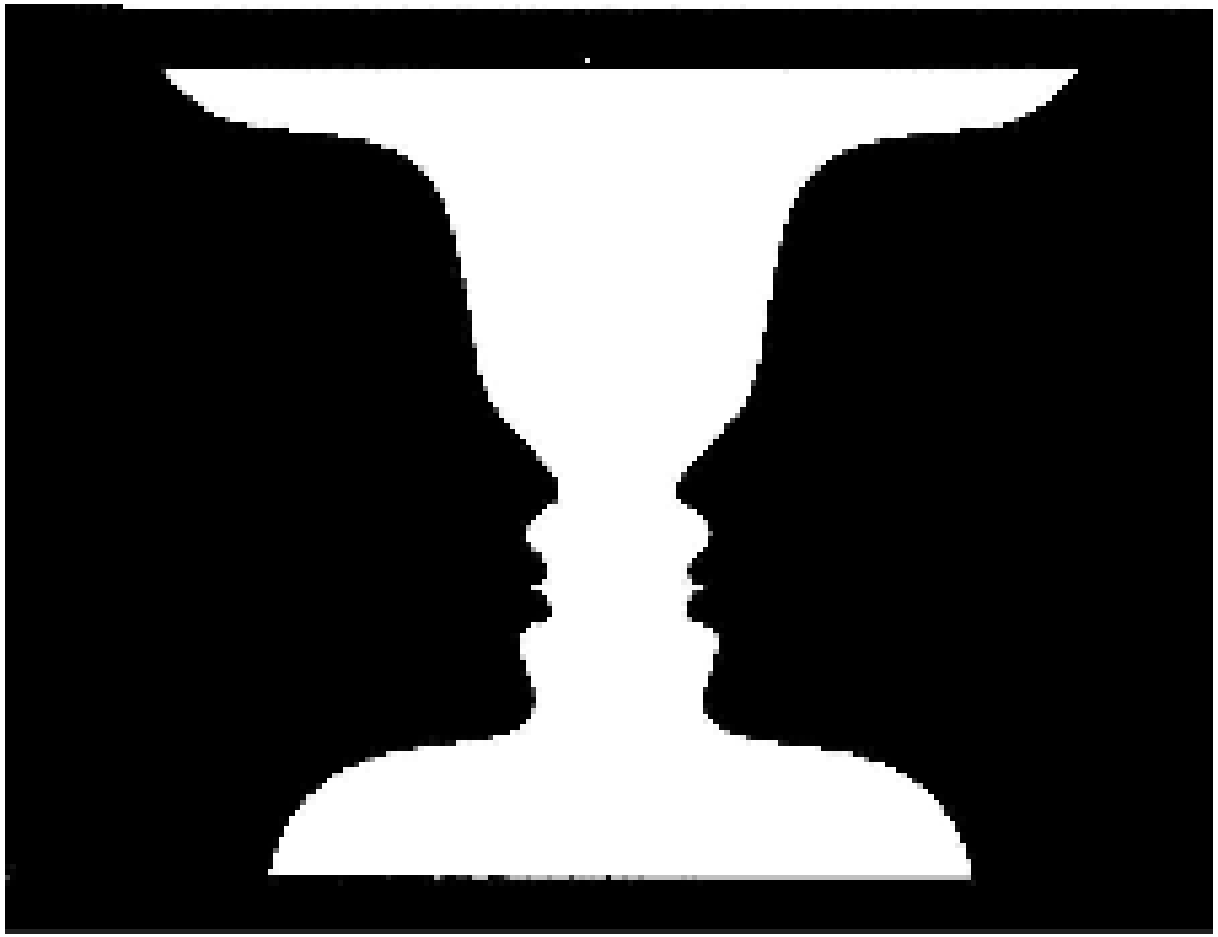
Video 2.9
Jianbo Shi

Edge Formation Factors



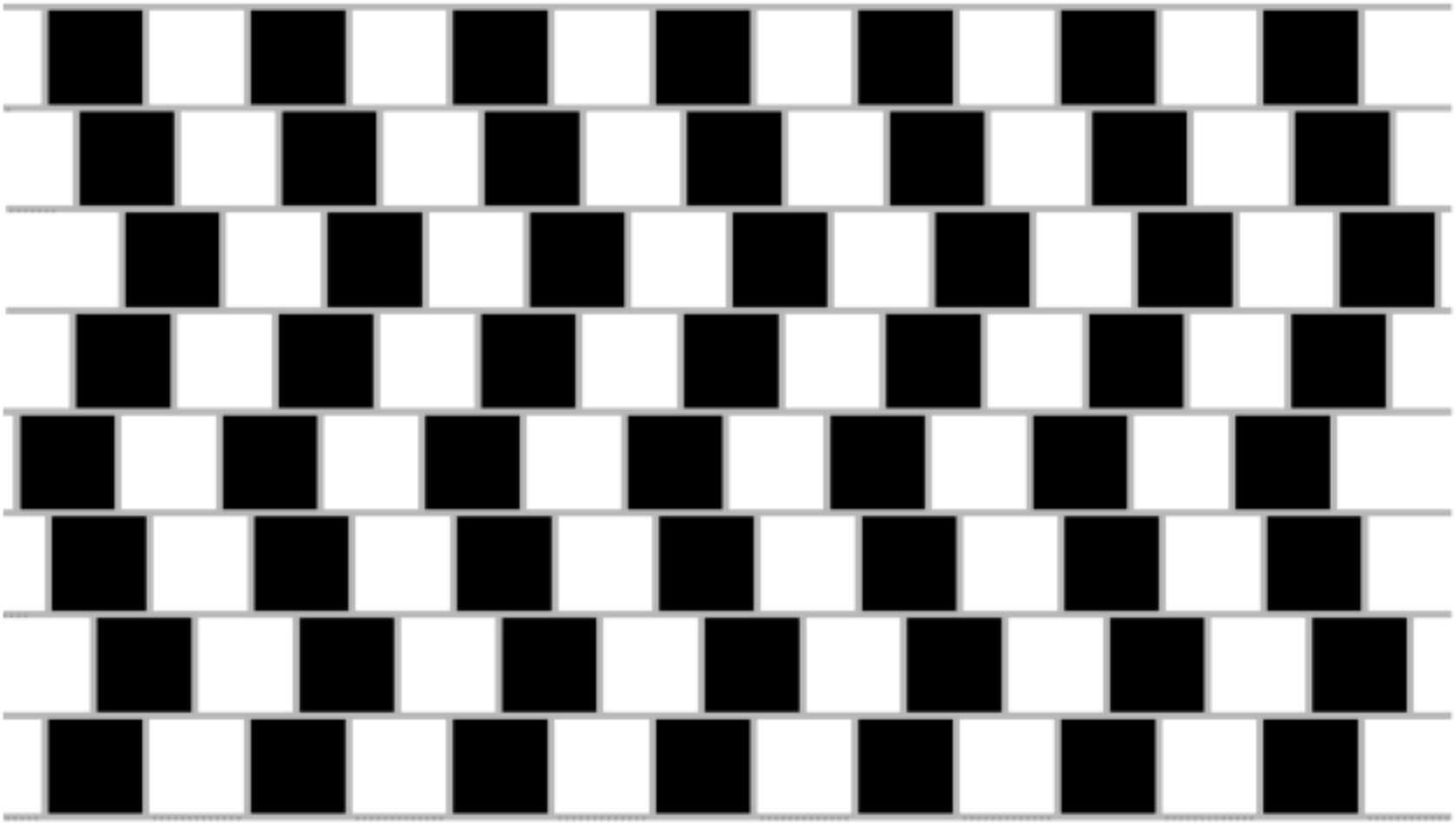
What's in front?

What it is?



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

j

i

121	121	118	111	. . .	21
134	136	137	132	. . .	23
133	131	136	136	. . .	25
136	145	148	151	. . .	34
137	140	147	149	. . .	54
. . .	. . .	. . .	. . .	. . .	. . .
231	233	243	244	. . .	179

Any 2D matrix can be seen as an image

Linear Filtering

Image I

200	130	20
255	100	10
200	100	30

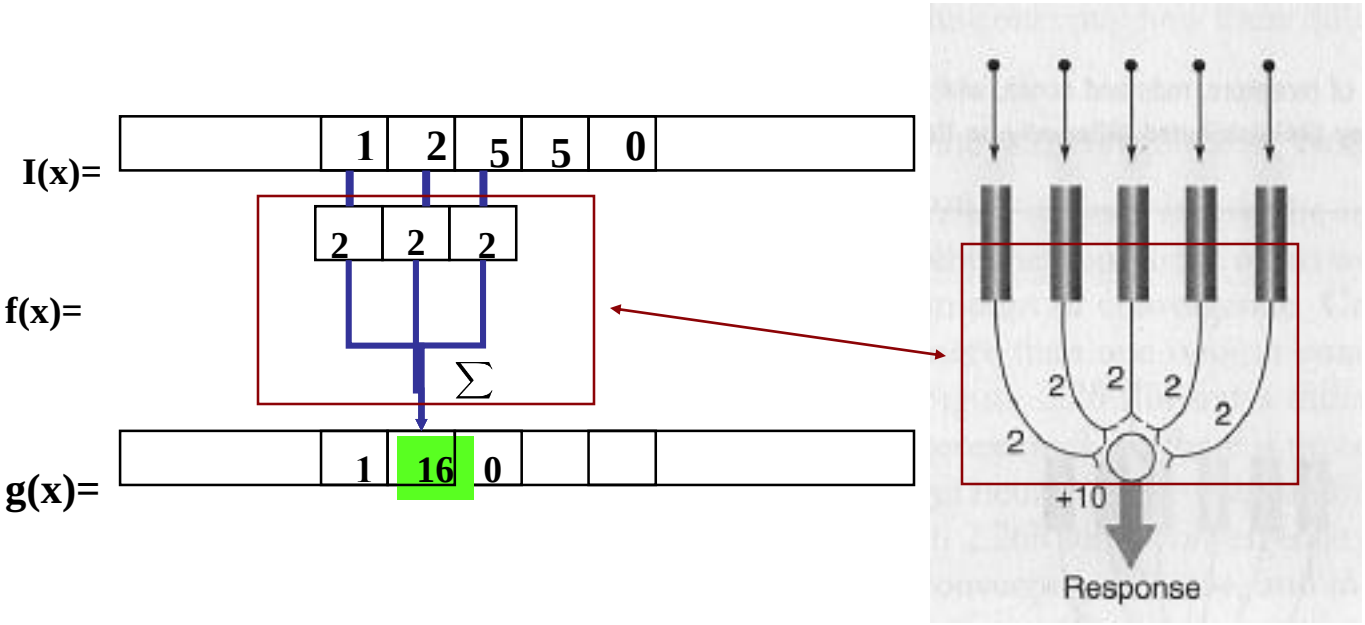
Kernel f

0.5	0	0
0	1.5	0.5
0	-0.5	0

	205	

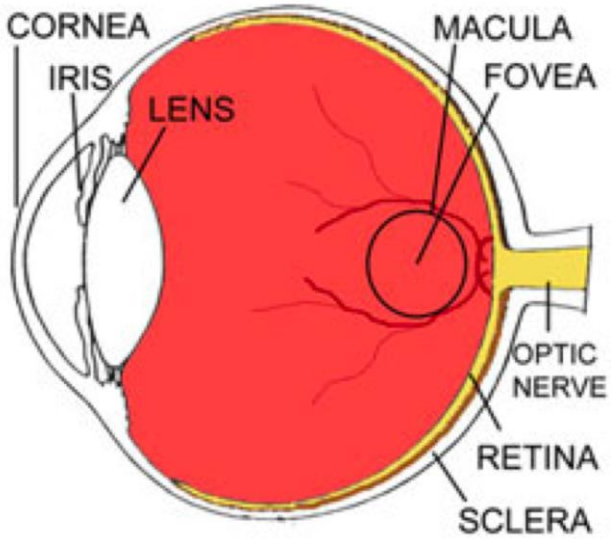
Image *filtering:* Replacing each pixel value by weighted average of its neighbors

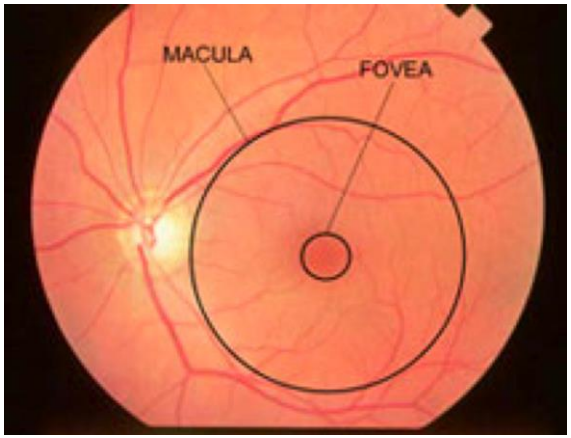
Simple Neural Network



This leads to parallel computation

Anatomy of eye

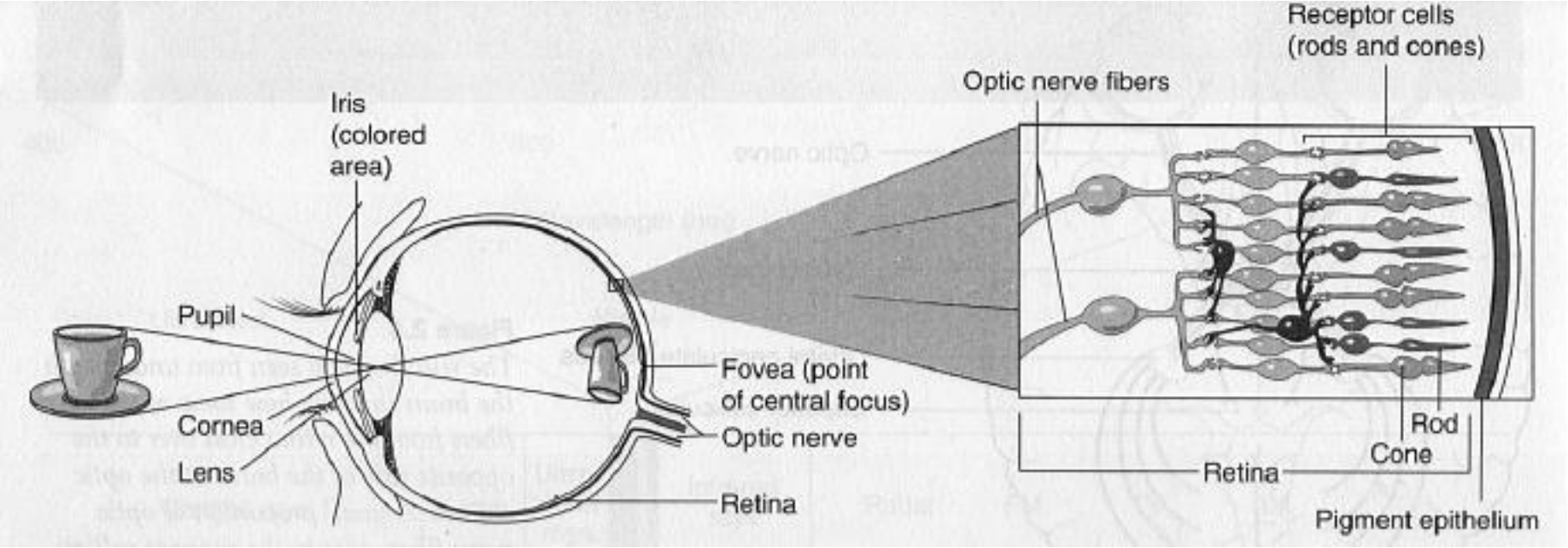




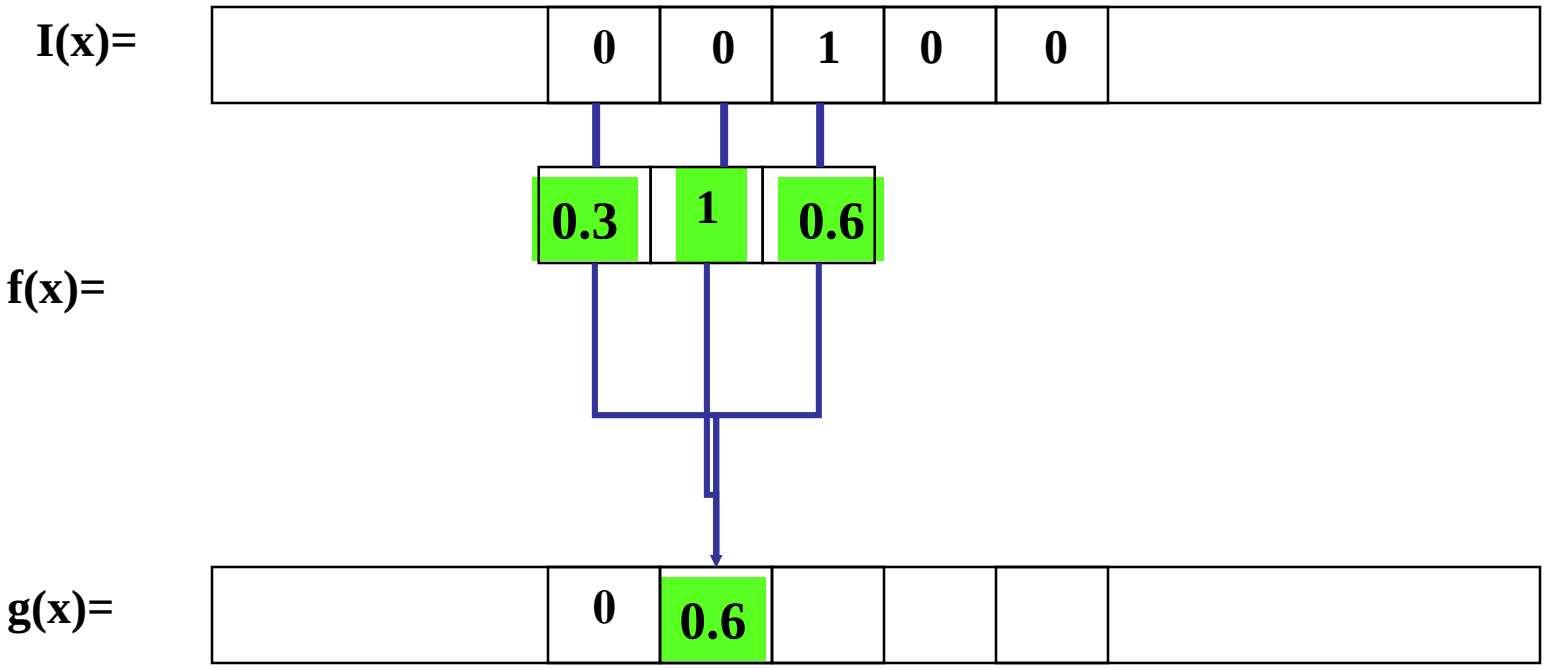
The macula is the center of vision (and retina) and the fovea (FAZ) is the focal point approximately only 0.4mm in diameter. Reading, driving, etc. is all performed here.

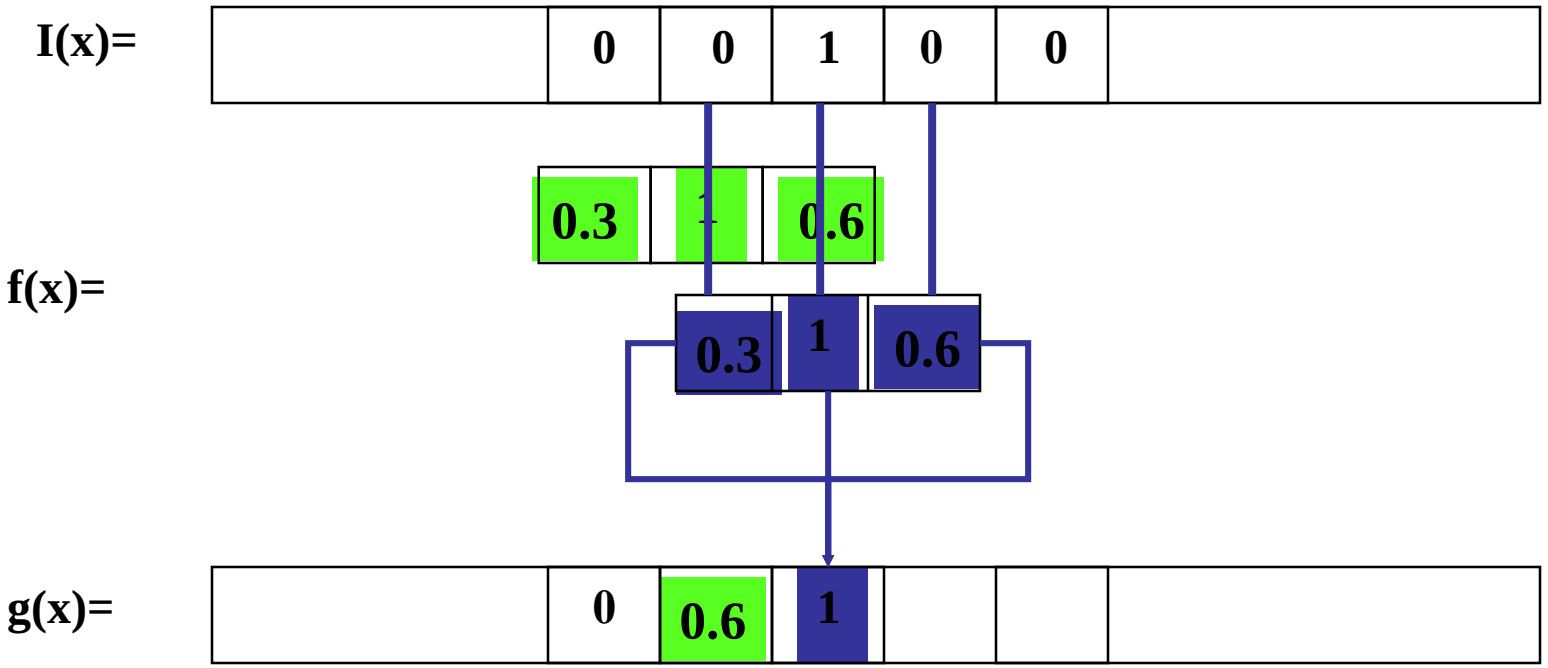
http://www.retinatexas.com/Anatomy_eng.html

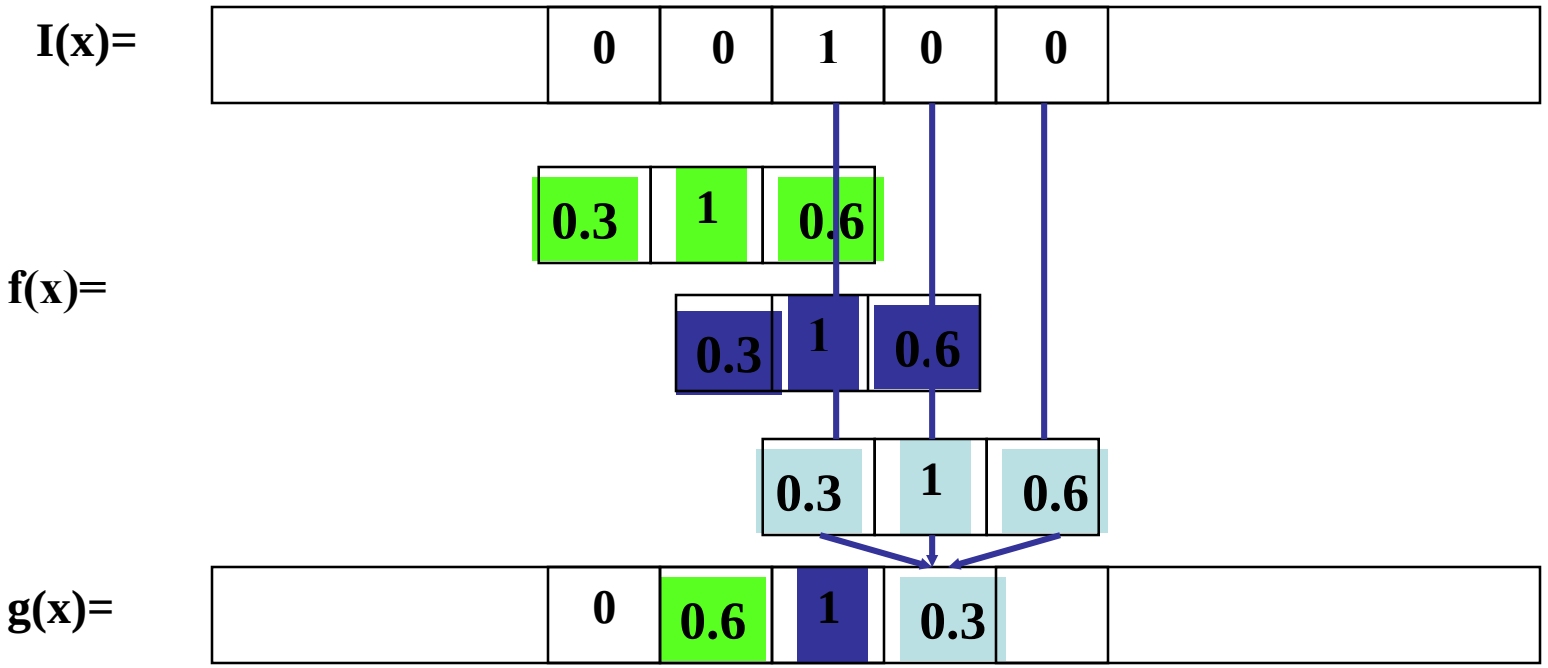
Photo-sensors

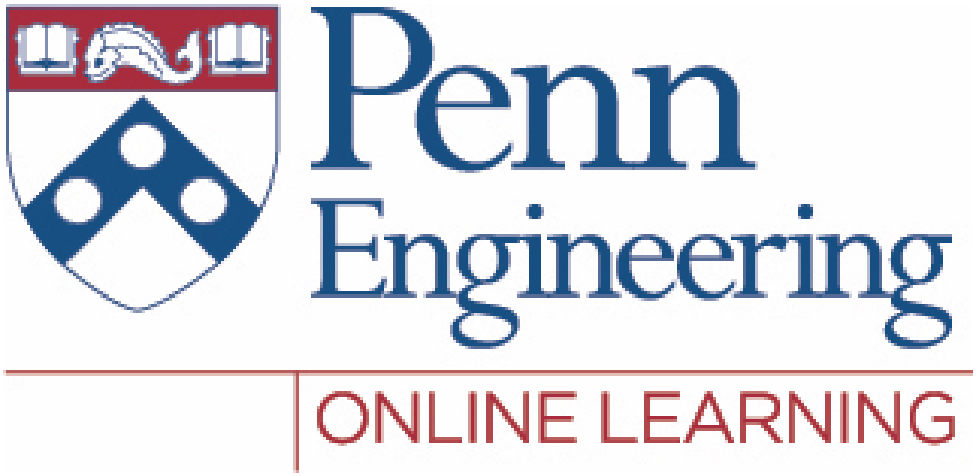


1) **Light pass through retina cells, excites Rod and Cone**
2) **Cone: color(spectral) sensitive, R,G,B, 6 Million**
3) **Rod: more photo sensitive, peak at 580nm(yellow), 120 M**
4) **What happens if you miss one type of Cone cells?**









Video 2.10
Jianbo Shi

Linear Filter

Image

0	0	0
0	1	0
0	0	0

Filter
Kernal

Linear Filter

Image

0	0	0
0	1	0
0	0	0

Kernal

Image no change

Linear Filter

Image

0	0	0
1	0	0
0	0	0

Kernal

Linear Filter

Image

0	0	0
1	0	0
0	0	0

Kernal

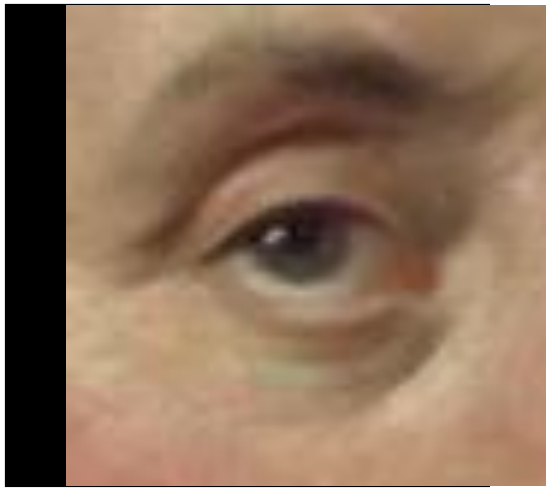


Image Shifted

Linear Filter

Image

0	0	0
0.3	0.3	0.3
0	0	0

Kernal

Linear Filter

Image

0	0	0
0.3	0.3	0.3
0	0	0

Kernal

Blurred
(horizontal)

Linear Filter

Image

0	0.3	0
0	0.3	0
0	0.3	0

Kernal

Linear Filter

Image

0	0.3	0
0	0.3	0
0	0.3	0

Kernal

Blurred (vertical)

Linear Filter

Image

0	0	0
0	2	0
0	0	0

−

0	0	0
0.3	0.3	0.3
0	0	0

Filter
Kernal

Linear Filter

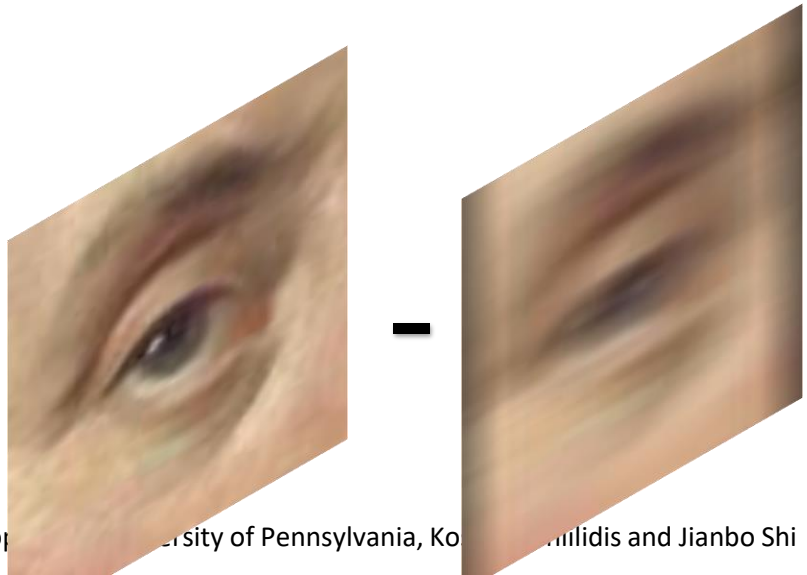
Image

0	0	0
0	2	0
0	0	0

−

0	0	0
0.3	0.3	0.3
0	0	0

Kernal Image



Linear Filter

Image

0	0	0
0	2	0
0	0	0

−

0	0	0
0.3	0.3	0.3
0	0	0

Kernal Image

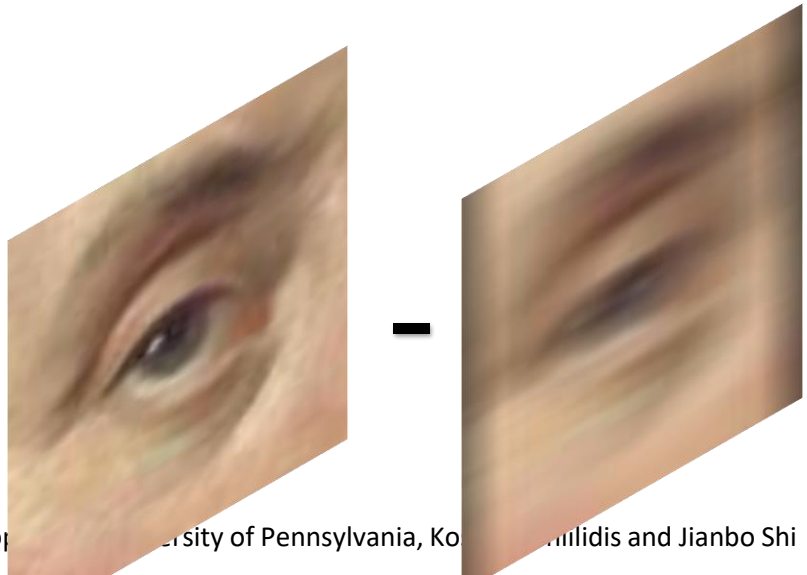


Image filtering

$$g[m,n] = \sum_{k,l} I(m+k, n+l) * f(k,l)$$

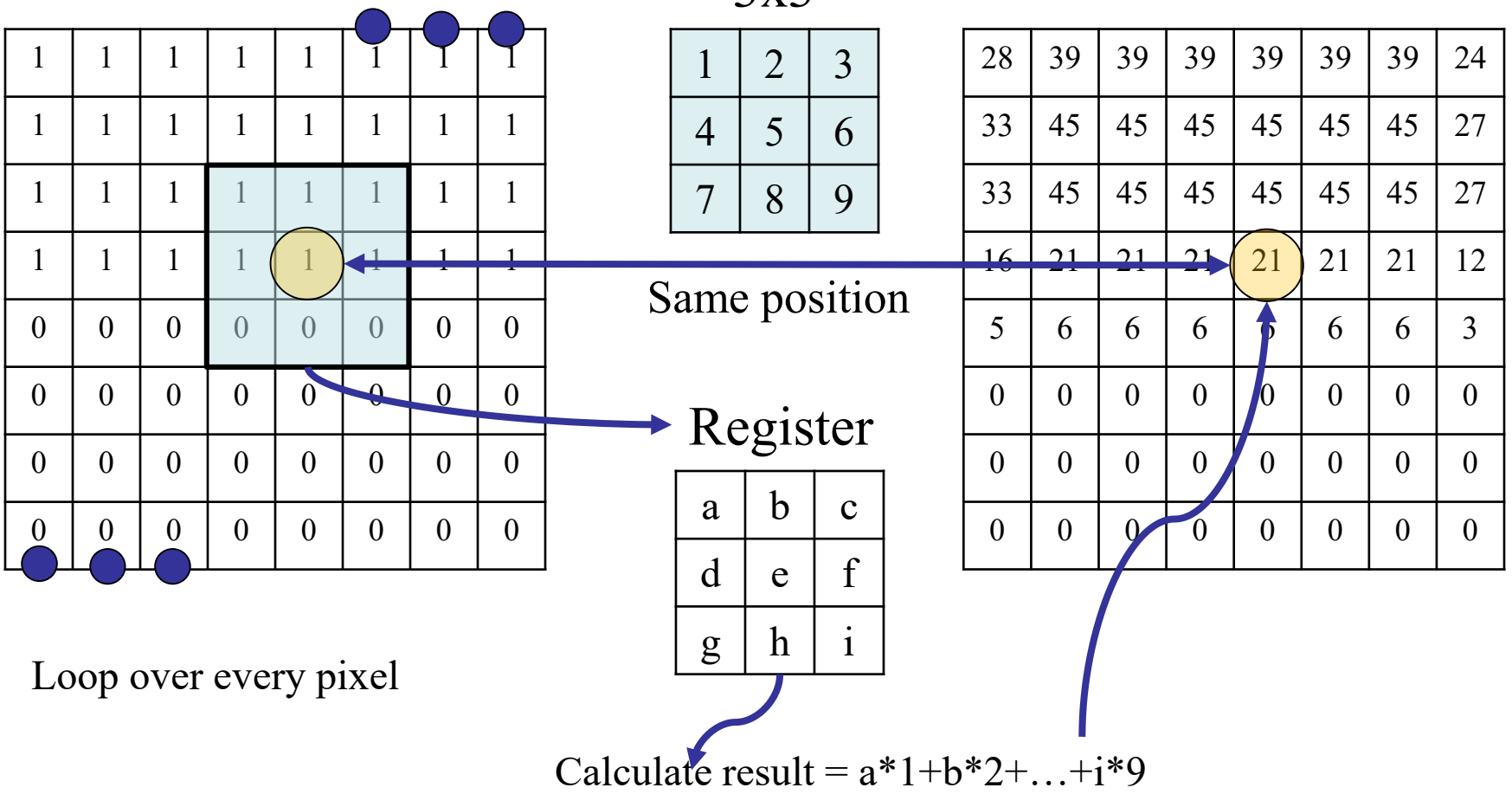
Output Image

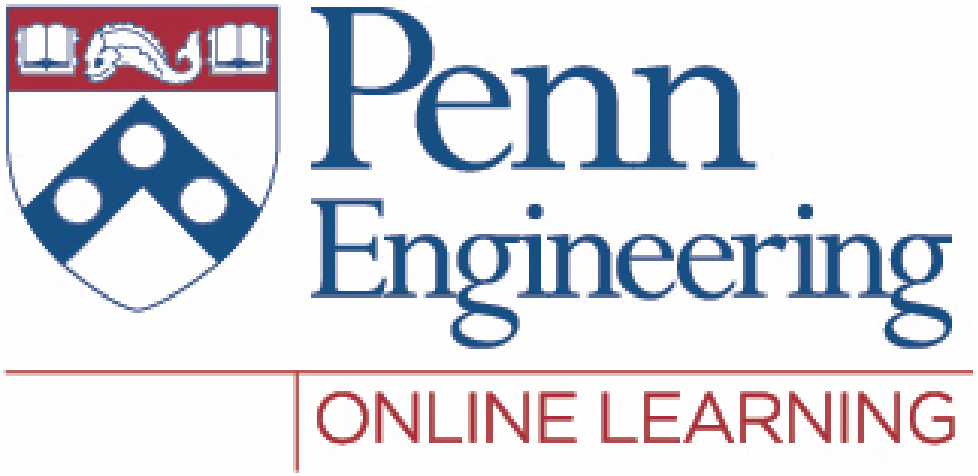
Input Image

Kernal Image

Image filtering

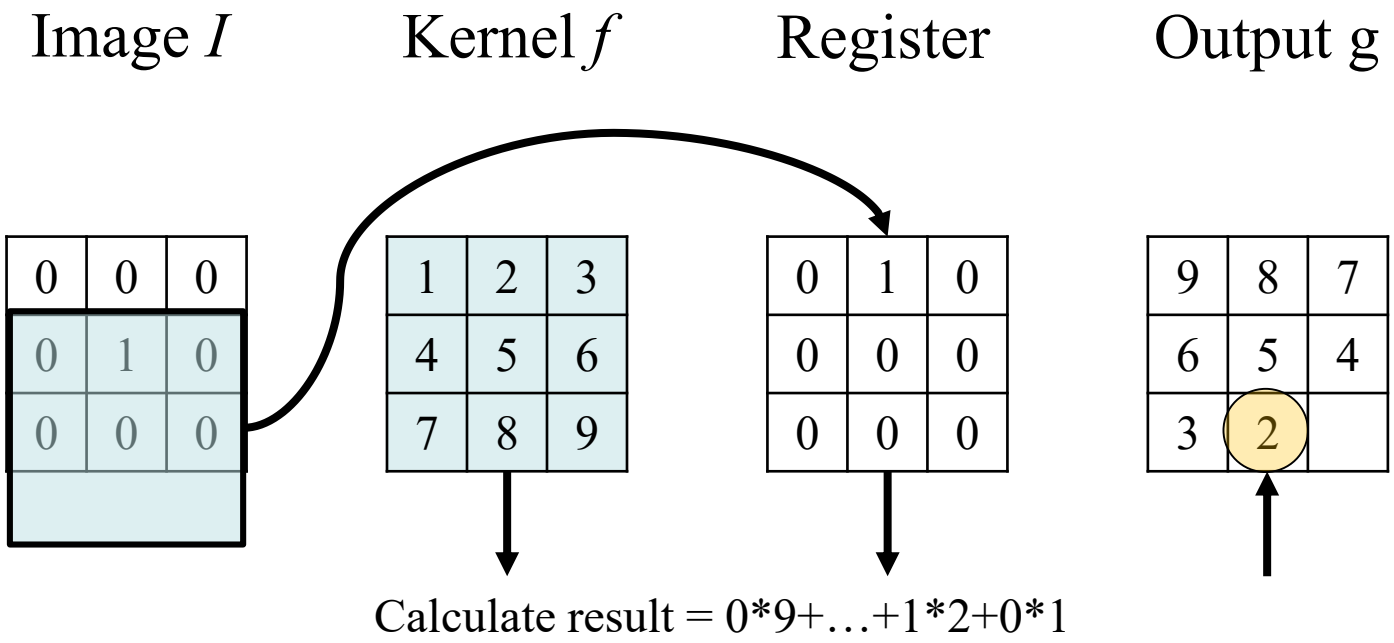
$$g[m,n] = \sum_{k,l} I(m+k, n+l) * f(k,l)$$



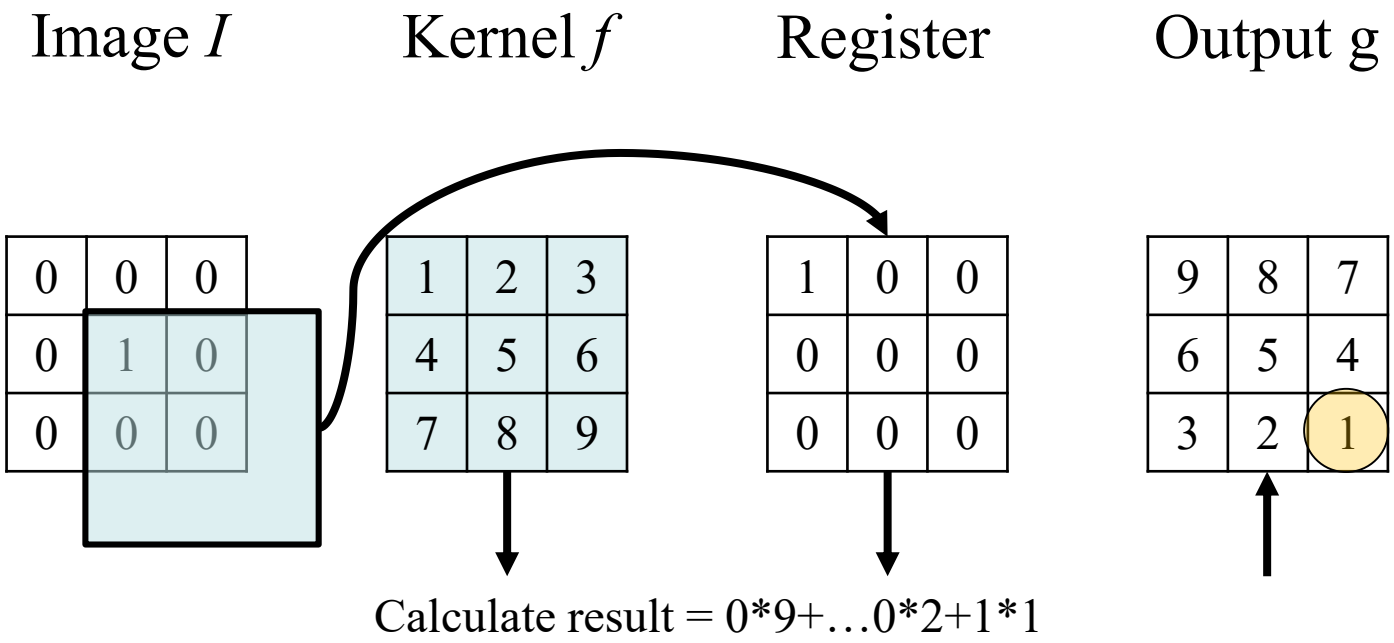


Video 2.11
Jianbo Shi

Special case: impulse function



Special case: impulse function



<Note> The output is the kernel flipped left-right, up-down!

Convolution $I \otimes f$

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

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

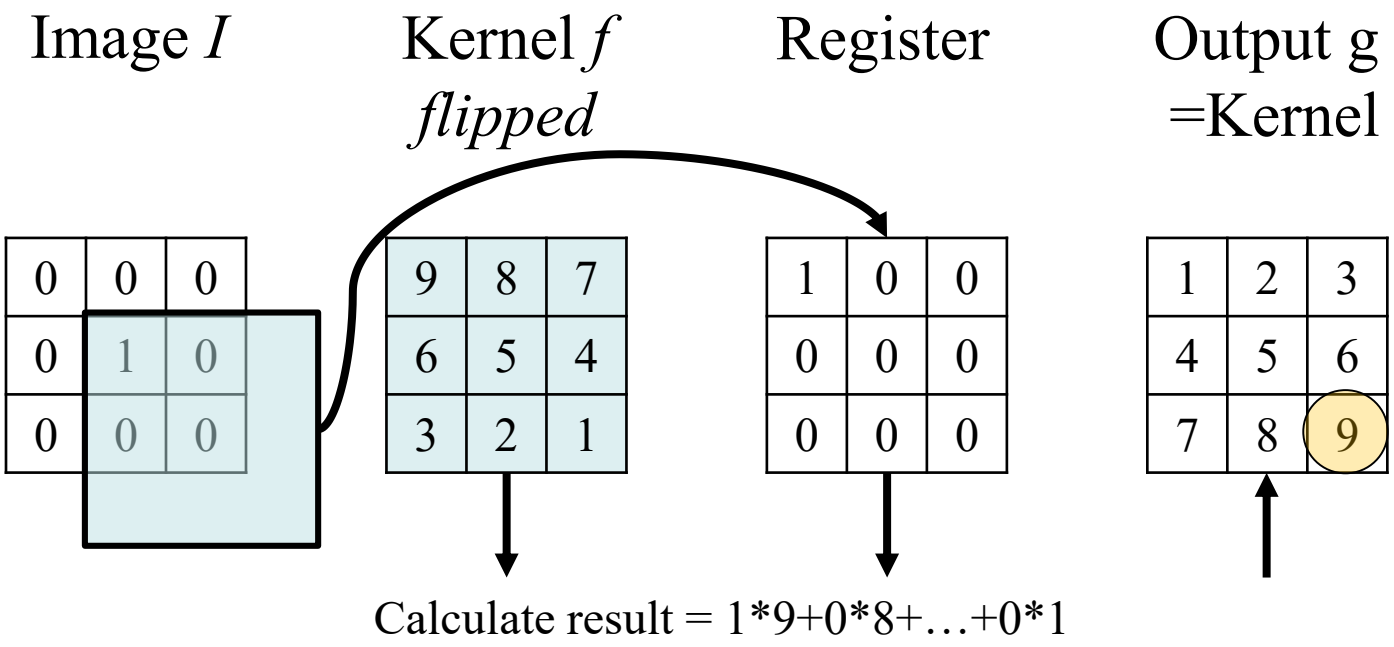
Output Image

Input Image

Kernal Image

Convolution

- Convolution is filtering with kernel flipped



Impulse functions shift images

Image I

a	b	c
d	e	f
g	h	i

Kernel f

1	0	0
0	0	0
0	0	0

Kernel f'

0	0	0
0	0	0
0	0	1

Result $I \otimes f$

e	f	0
h	i	0
0	0	0

- In this case the resulting image shifted to the upper left

Impulse functions shift images

Image I

1	0	0
0	0	0
0	0	0

Kernel f

a	b	c
d	e	f
g	h	i

Kernel f'

i	h	g
f	e	d
c	b	a

Result $I \otimes f$

e	f	0
h	i	0
0	0	0

- In this case the resulting image shifted to the upper left

Convolution is associative

I

a	b	c
d	e	f
g	h	i

f

1	0	0
0	0	0
0	0	0

$I \otimes f$

e	f	0
h	i	0
0	0	0

f

1	0	0
0	0	0
0	0	0

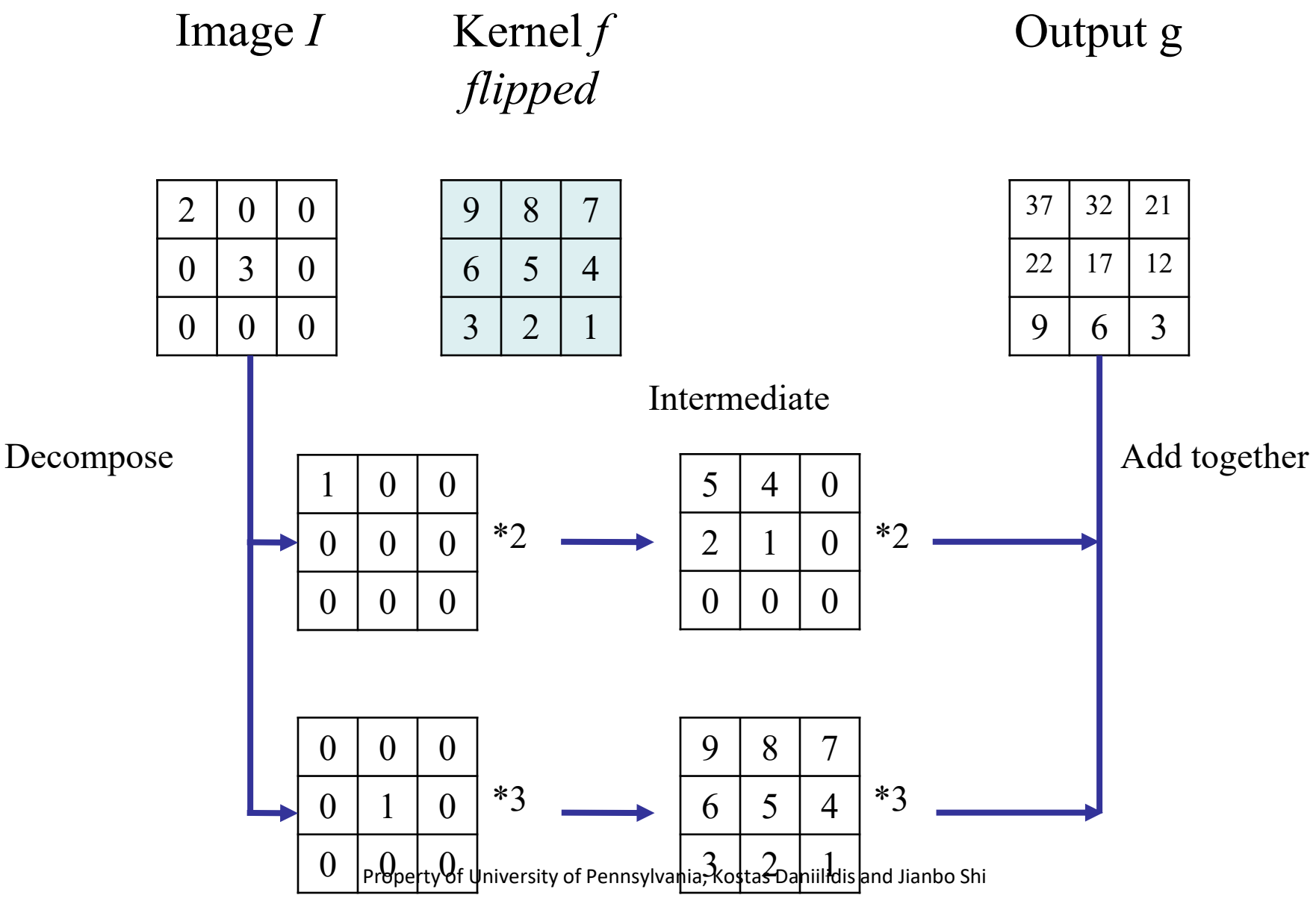
I

a	b	c
d	e	f
g	h	i

$f \otimes I$

e	f	0
h	i	0
0	0	0

Linear independence



Linear independence

Image I

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

5x5

Kernel f

1	0	0
0	1	0
0	0	0

Output $g=g_1+g_2$

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

Kernel f_1

1	0	0
0	0	0
0	0	0

Output g_1

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

Kernel f_2

0	0	0
0	1	0
0	0	0

Output g_2

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

- Convolution is commutative

$$I \otimes f$$

Image I

a	b	c
d	e	f
g	h	i

Kernel f

1	0	0
1	0	0
1	1	0

Decompose

Image I

a	b	c
d	e	f
g	h	i

$\otimes$

1	0	0
0	0	0
0	0	0

e	f	0
h	i	0
0	0	0

0	0	0
1	0	0
0	0	0

b	c	0
e	f	0
h	i	0

0	0	0
0	0	0
1	0	0

0	0	0
b	c	0
e	f	0

0	0	0
0	0	0
0	1	0

0	0	0
a	b	c
d	e	f

- Convolution is commutative

$$f \otimes I$$

Image I

a	b	c
d	e	f
g	h	i

Kernel f

1	0	0
1	0	0
1	1	0

Decompose

1	0	0
0	0	0
0	0	0

e	f	0
h	i	0
0	0	0

0	0	0
1	0	0
0	0	0

$\otimes$

a	b	c
d	e	f
g	h	i

b	c	0
e	f	0
h	i	0

0	0	0
0	0	0
1	0	0

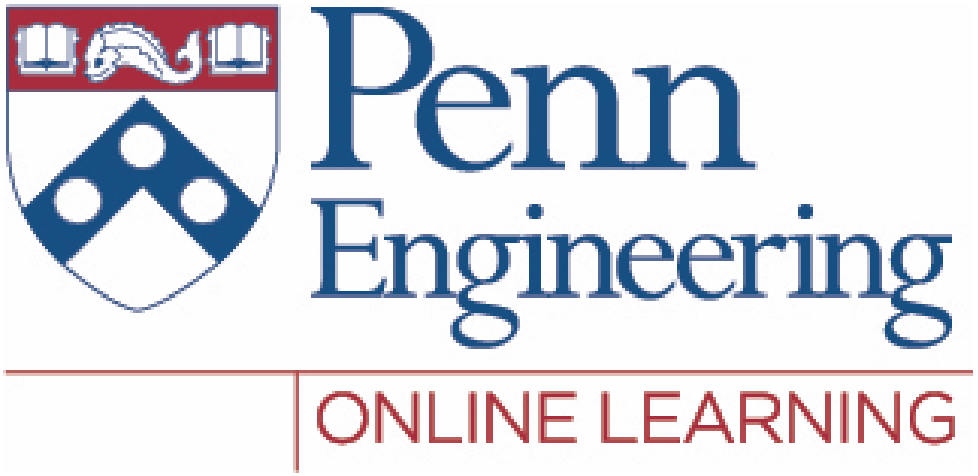
0	0	0
b	c	0
e	f	0

0	0	0
0	0	0
0	1	0

0	0	0
a	b	c
d	e	f

Proof of Commutative property

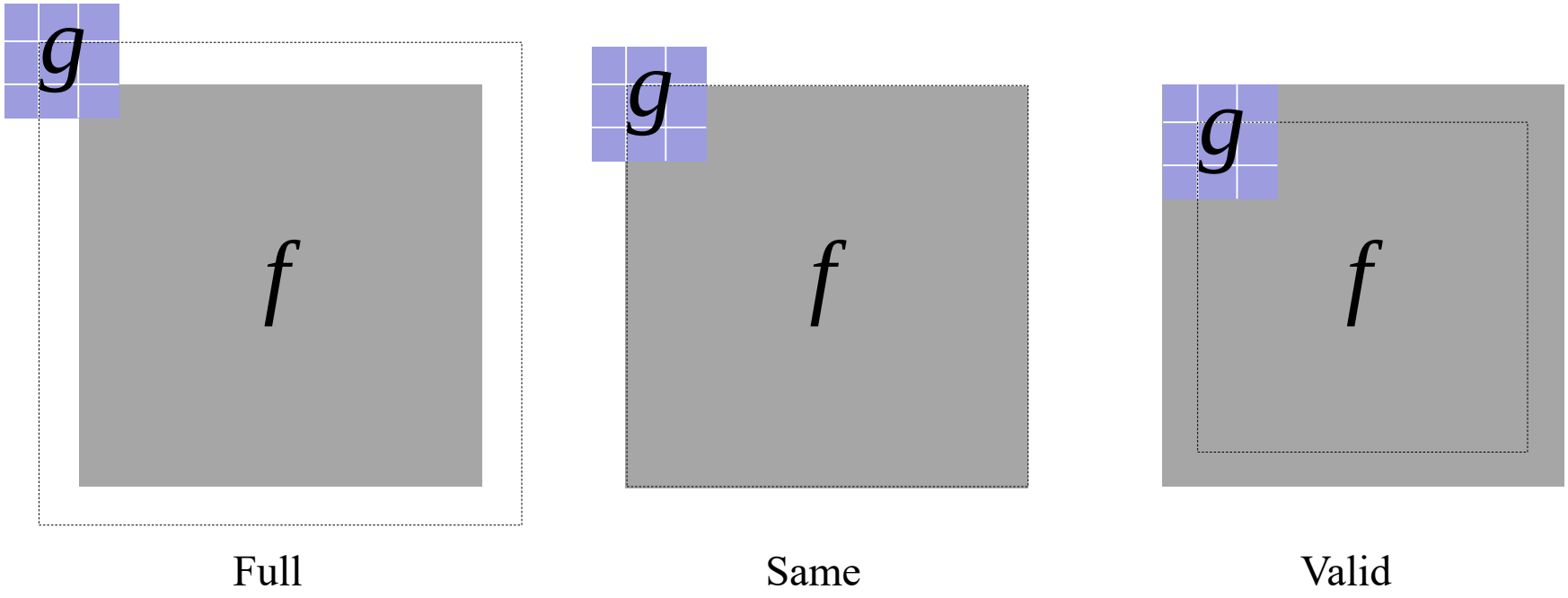
- $g[m,n] = I \otimes f = f \otimes I$
- $g[m,n] = I \otimes f = \sum_{k,l} I(m-k, n-l) * f(k,l)$
- Let $k' = m - k, l' = n - l,$
  then $k = m - k', l = n - l'$
- $g[m,n] = \sum_{k',l'} I(k',l') * f(m-k', m-l') = f \otimes I$



Video 2.12
Jianbo Shi

Output Size of Image Convolution

$f \otimes g$



Full Same Valid

filter2(g, f, shape) in MATLAB

Full: output_size = f_size + g_size – 1

Same: output_size = f_size

Valid: output_size = f_size – (g_size – 1)

2D visualization of convolution (full)

Image I

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

8x8

Kernel f

1	2	3
4	5	6
7	8	9

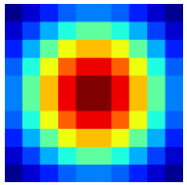
3x3

Output g

1	3	6	6	6	6	6	6	5	3
5	12	21	21	21	21	21	21	16	9
12	27	45	45	45	45	45	45	33	18
12	27	45	45	45	45	45	45	33	18
11	24	39	39	39	39	39	39	28	15
7	15	24	24	24	24	24	24	17	9
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

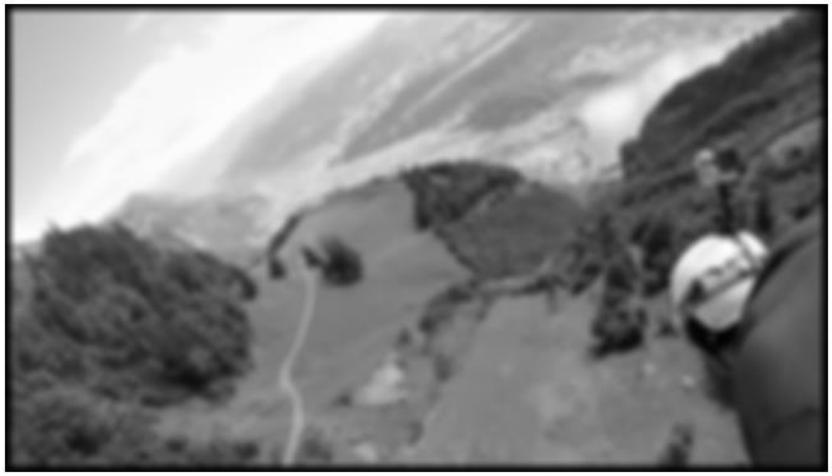
10x10

Output Size of Image Convolution



g : 10 x 10 Gaussian kernel

f : 640 x 360 resolution



Full

filter2(g, f, shape) in MATLAB

Full: output_size = f_size + g_size – 1

```
>> full = filter2(g, im, 'full');
>> size(full)

ans =

   369   649
```

2D visualization of convolution (same)

Image I

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

8x8

Kernel f

1	2	3
4	5	6
7	8	9

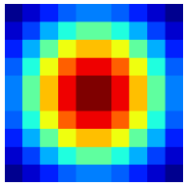
3x3

Output g

12	21	21	21	21	21	21	16
27	45	45	45	45	45	45	33
27	45	45	45	45	45	45	33
24	39	39	39	39	39	39	28
15	24	24	24	24	24	24	17
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

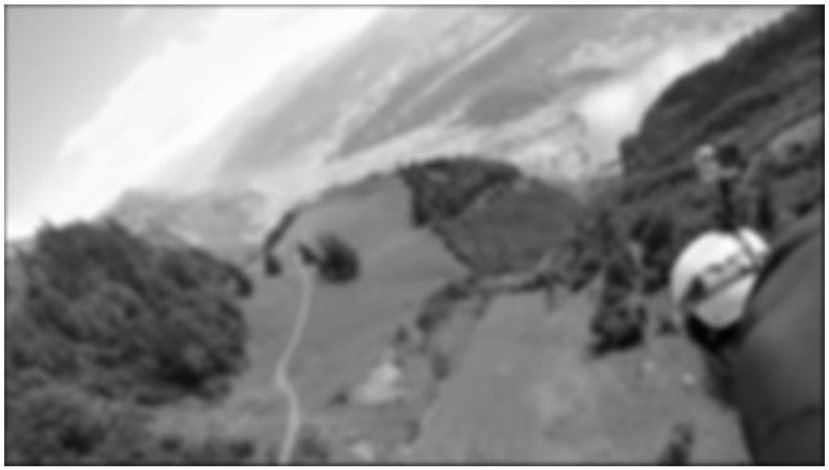
8x8

Output Size of Image Convolution



g : 10 x 10 Gaussian kernel

f : 640 x 360 resolution



Same

filter2(g, f, shape) in MATLAB

Full: output_size = f_size + g_size – 1

Same: output_size = f_size

```
>>same = filter2(g, im, 'same');
>> size(same)

ans =

   360   640
```

2D visualization of convolution (valid)

Image I

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

8x8

Kernel f

1	2	3
4	5	6
7	8	9

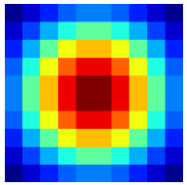
3x3

Output g

45	45	45	45	45	45
45	45	45	45	45	45
39	39	39	39	39	39
24	24	24	24	24	24
0	0	0	0	0	0
0	0	0	0	0	0

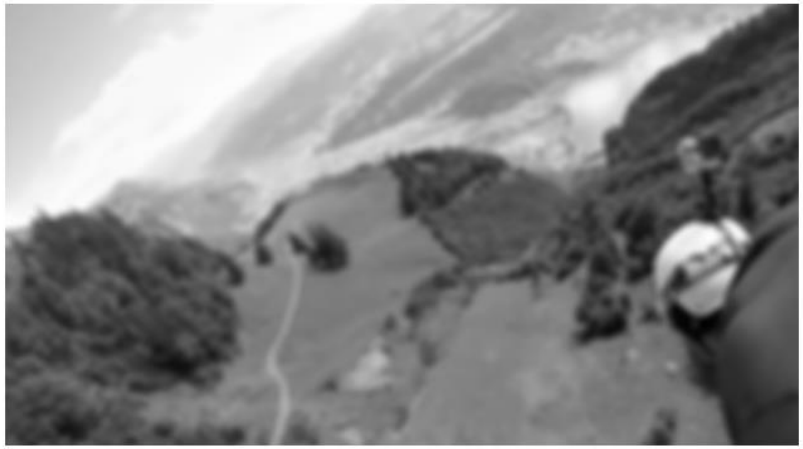
6x6

Output Size of Image Convolution



g : 10 x 10 Gaussian kernel

f : 640 x 360 resolution



Valid

filter2(g, f, shape) in MATLAB

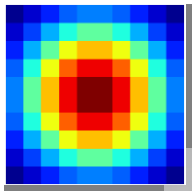
Full: output_size = f_size + g_size – 1

Same: output_size = f_size

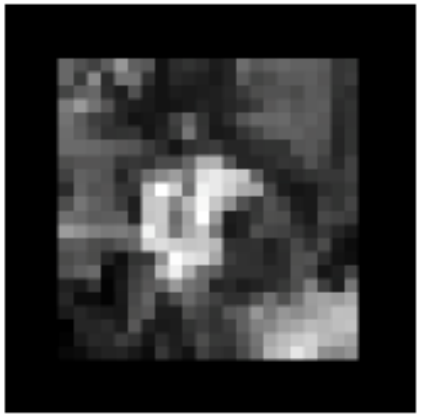
Valid: output_size = f_size – (g_size – 1)

```
>> valid = filter2(g, im, 'valid');
>> size(valid)

ans =

   351   631
```

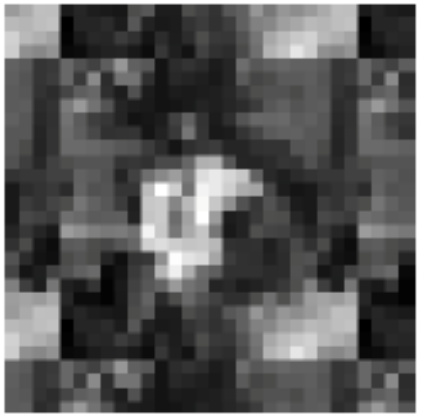
Image Boundary Effect



The filter window falls off at the edge of image.



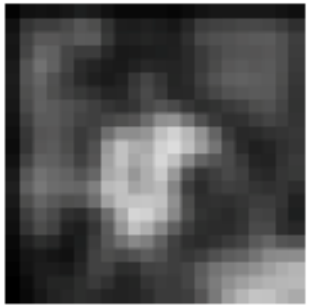
zero



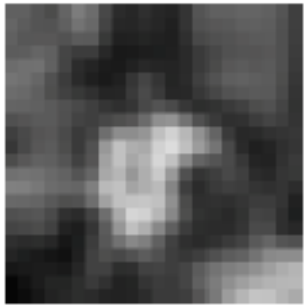
wrap

clamp

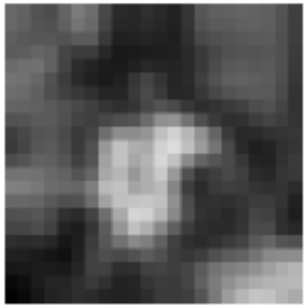
mirror



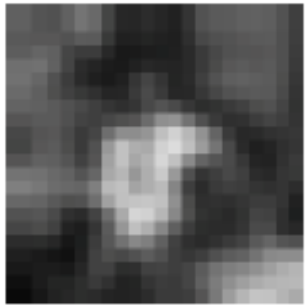
blurred zero



normalized zero



blurred clamp



blurred mirror

Image Extrapolation (Mirroring)

Code

```
J = imread('image.bmp');
figure; imshow(J);
```

J

Image Extrapolation (Mirroring)

Code

```
boarder = 40;
[nr,nc,nb] = size(J);
J_big = zeros(nr+2*boarder, nc + 2*boarder,nb);
J_big(boarder+1:boarder+nr,boarder+1:boarder+nc,:) = J;
```

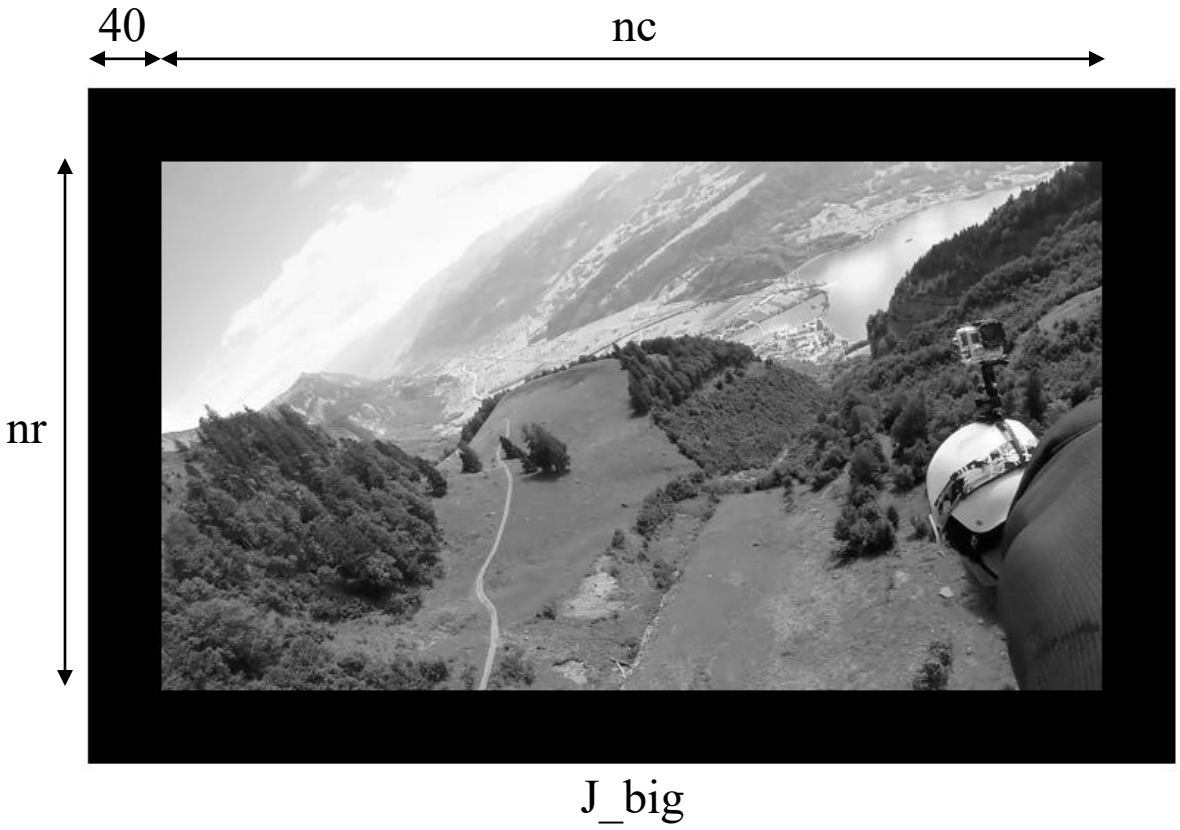
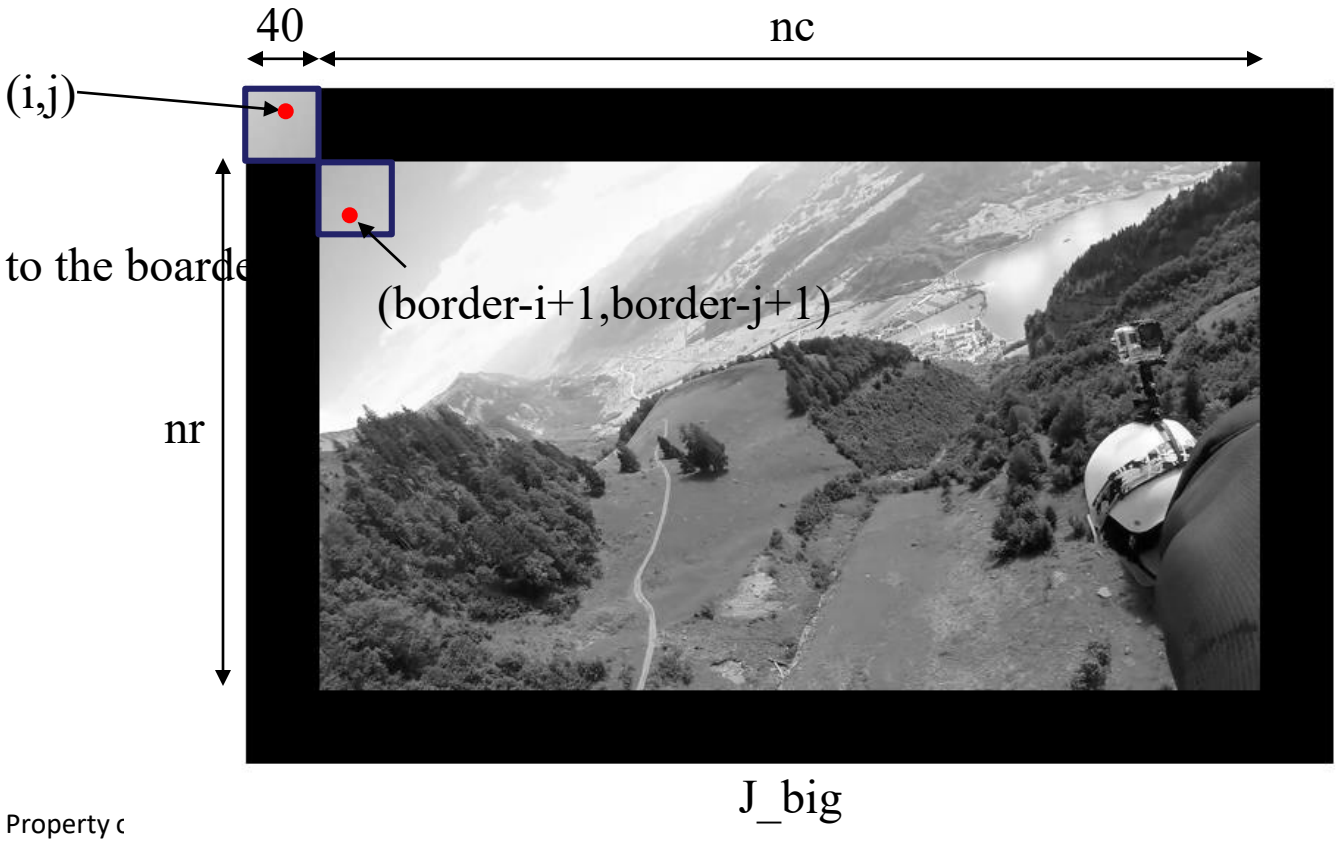


Image Extrapolation (Mirroring)

Code

```
for i=1:border,
    for j=1:border,
        J_big(i,j,:) = J(border-i+1,border-j+1,:);
    end
end
```



Mirroring with respect to the boarde

Image Extrapolation (Mirroring)

Code

```
for i=1:boarder,
   for j=border+1:border+nc,
         J_big(i,j,:) = J(border-i+1,j-border,:);
    end
end
```

Mirroring with respect to the boarde

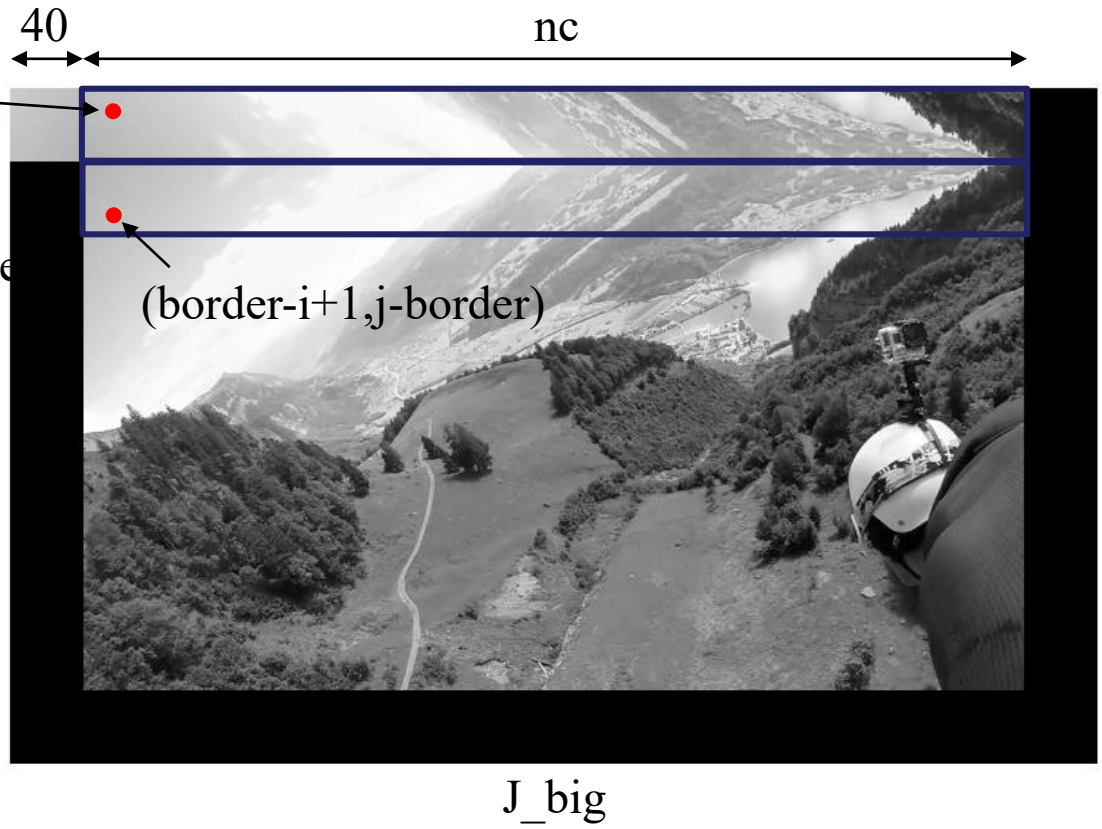


Image Extrapolation (Mirroring)

Code

```
for i=nr+border+1:border*2+nr,
    for j=border+1:border+nc,
        J_big(i,j,:) = J(2*nr-i+border+1,j-border,:);
    end
end
```

Mirroring with respect to the boarde

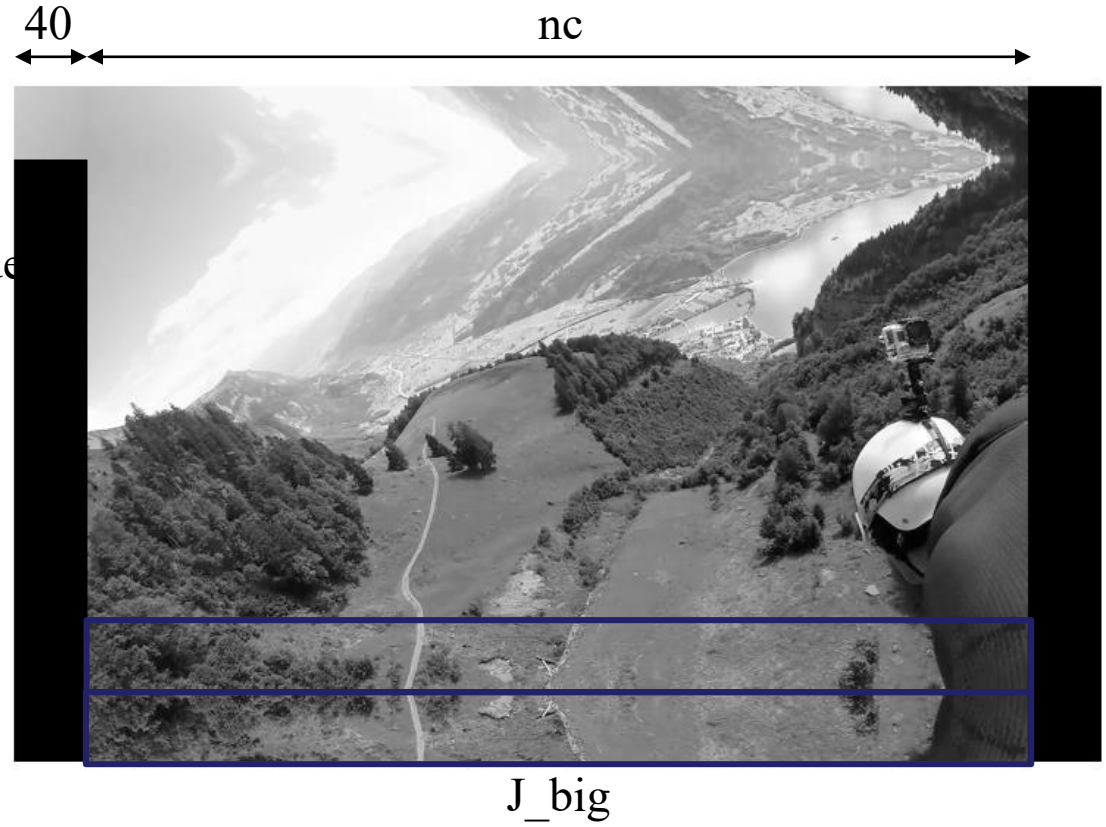
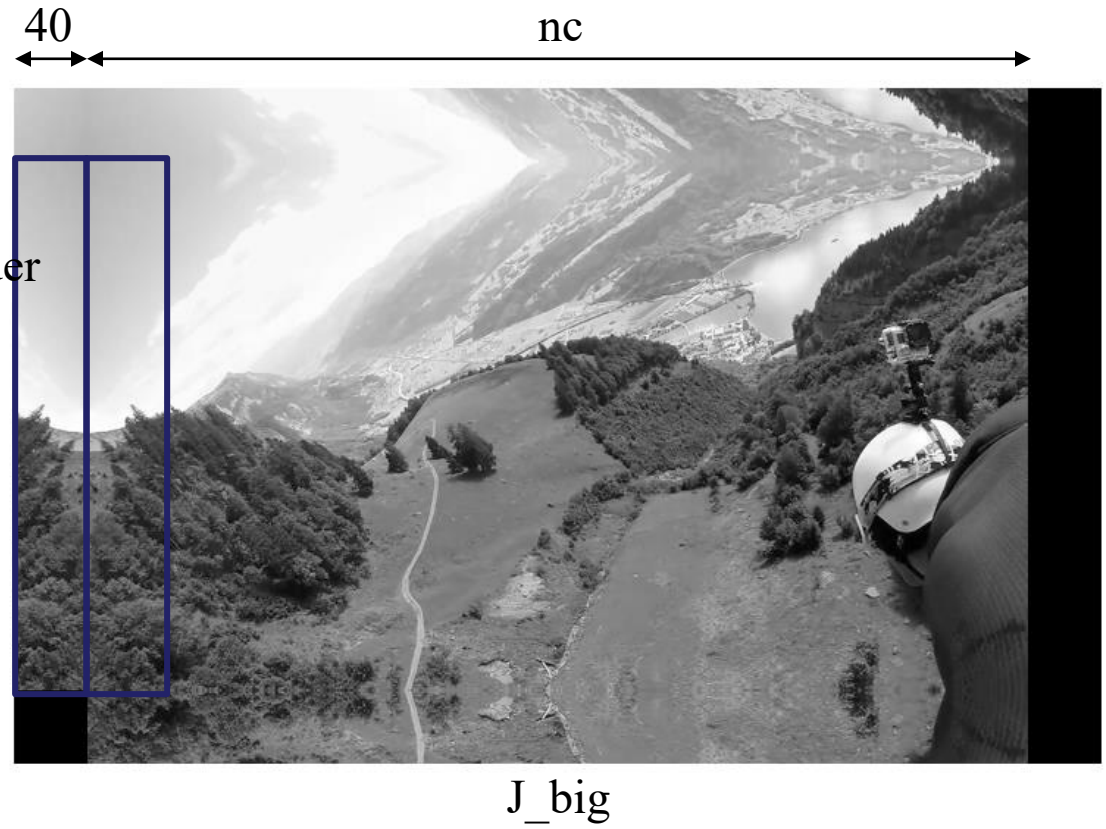


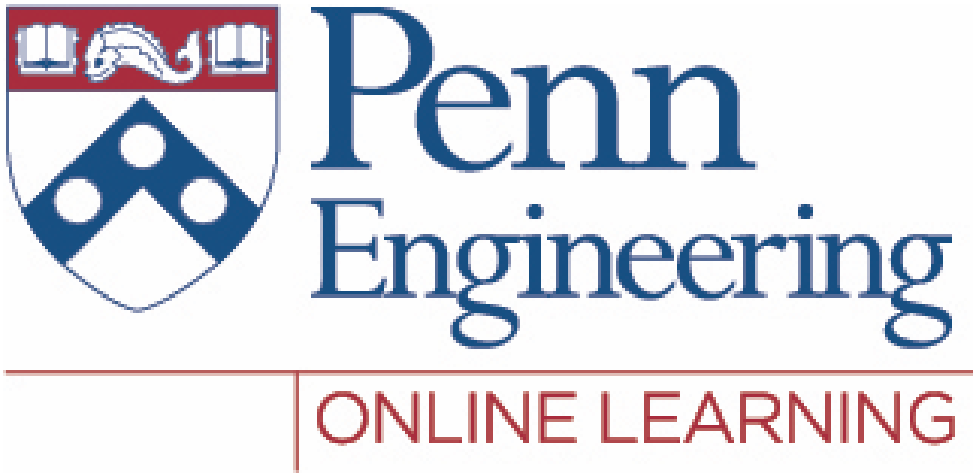
Image Extrapolation (Mirroring)

Code

```
for i=border+1:border+nr;
    for j=1:border,
        J_big(i,j,:) = J(i-border,border-j+1,:);
    end
end
```

Mirroring with respect to the boarder

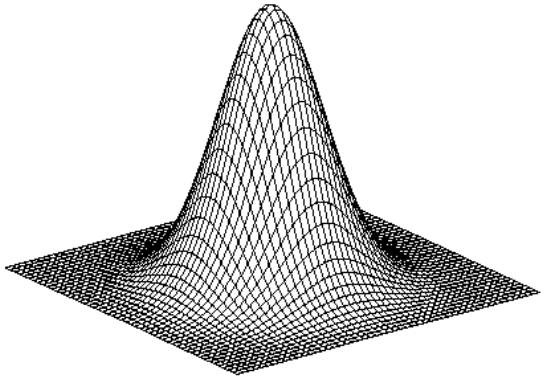




Video 2.13
Jianbo Shi

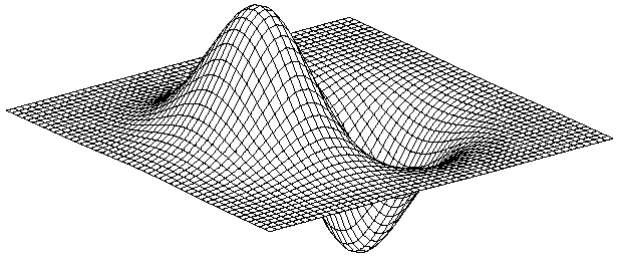
Examples of image operation as convolution

2D filters, more on this later…



Gaussian

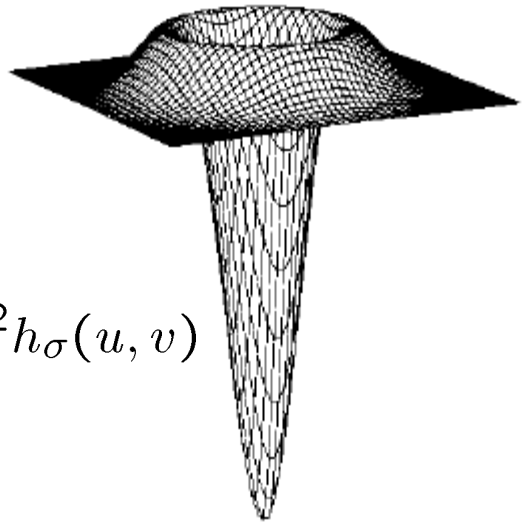
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

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

Laplacian of Gaussian

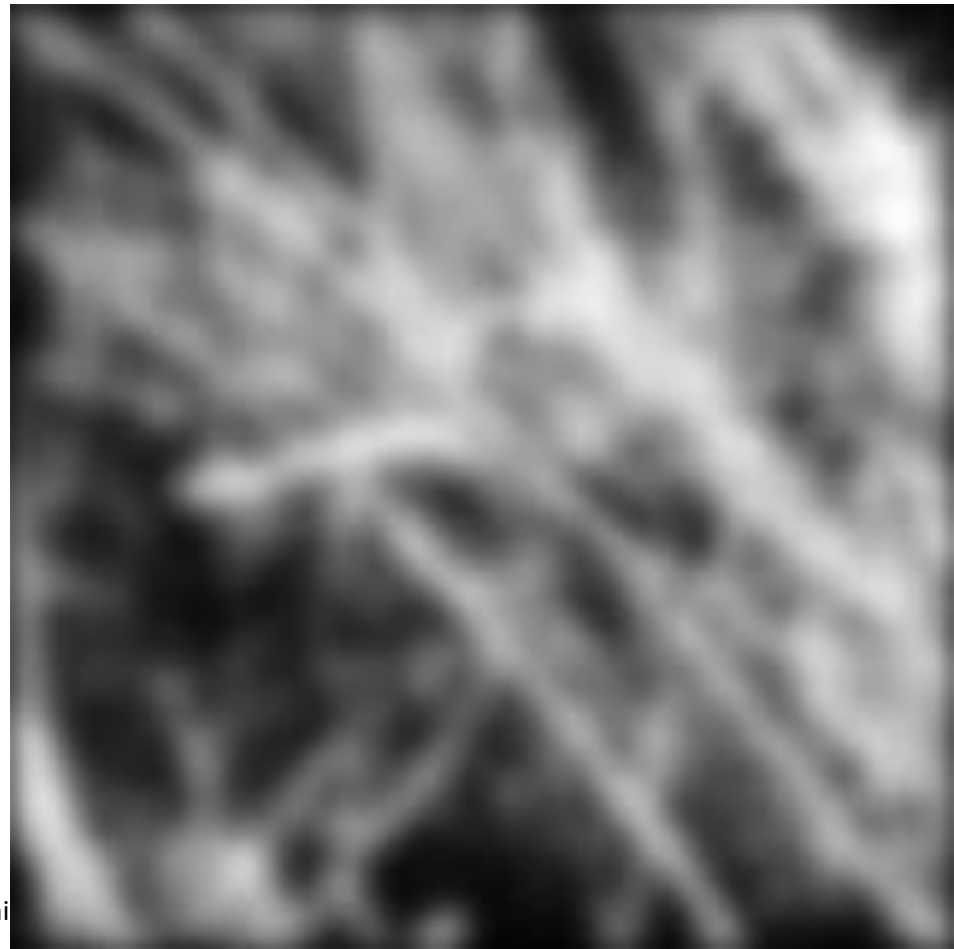


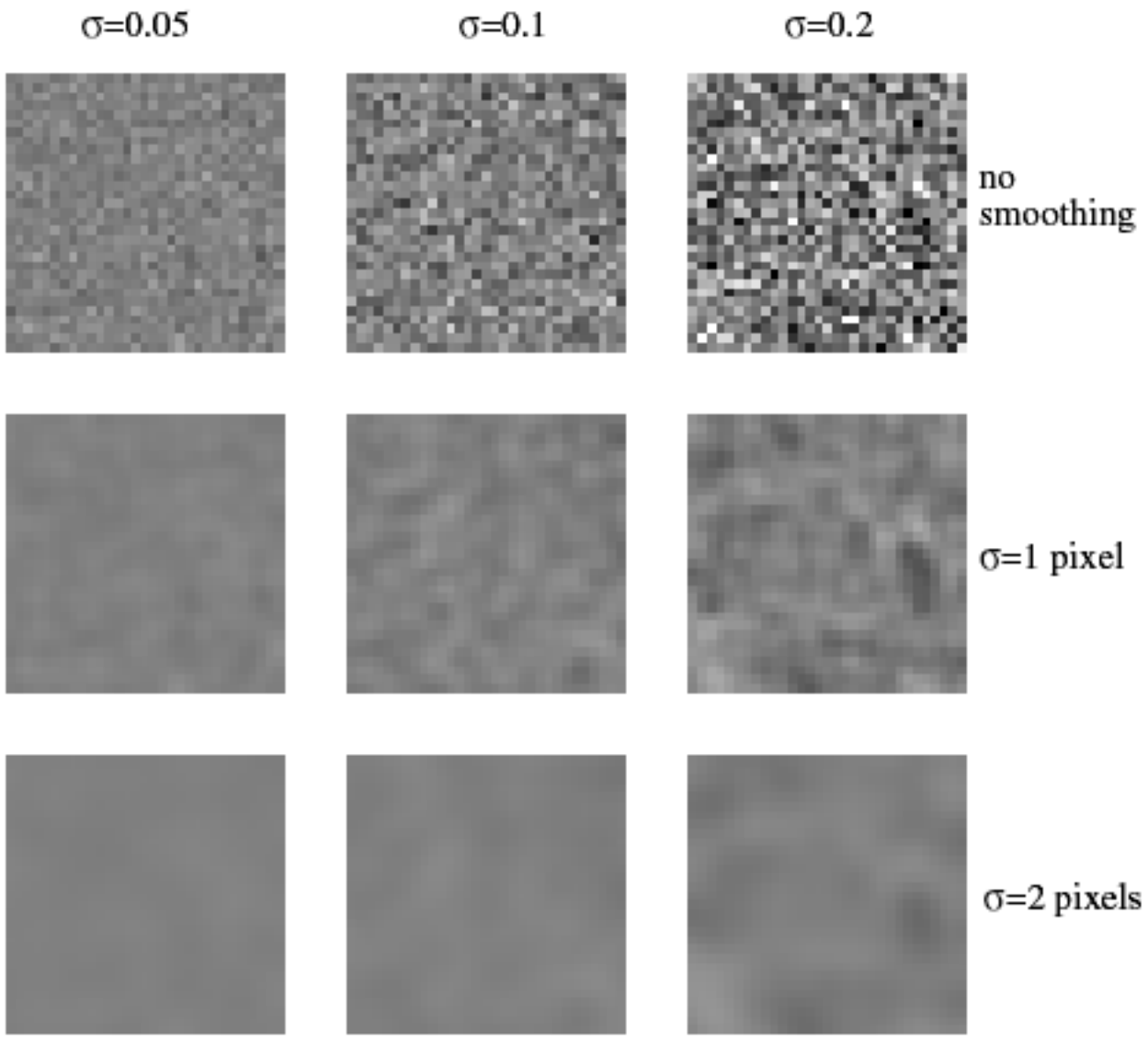
$$\nabla^2 h_\sigma(u,v)$$

- is the **Laplacian** operator:

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

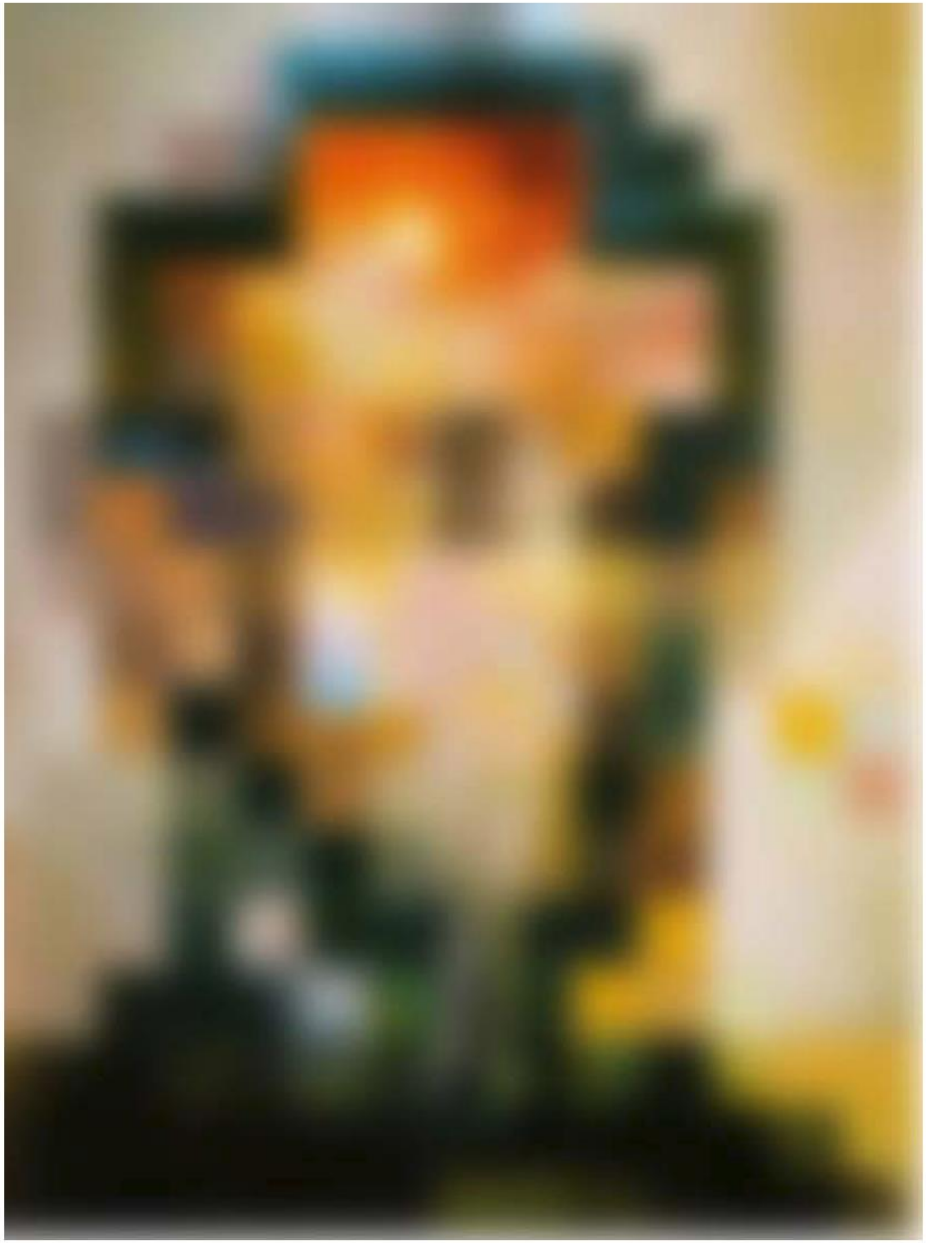
Smoothing with a Gaussian





The effects of smoothing
Each row shows smoothing with gaussians of different width; each column shows different realizations of an image of gaussian noise.

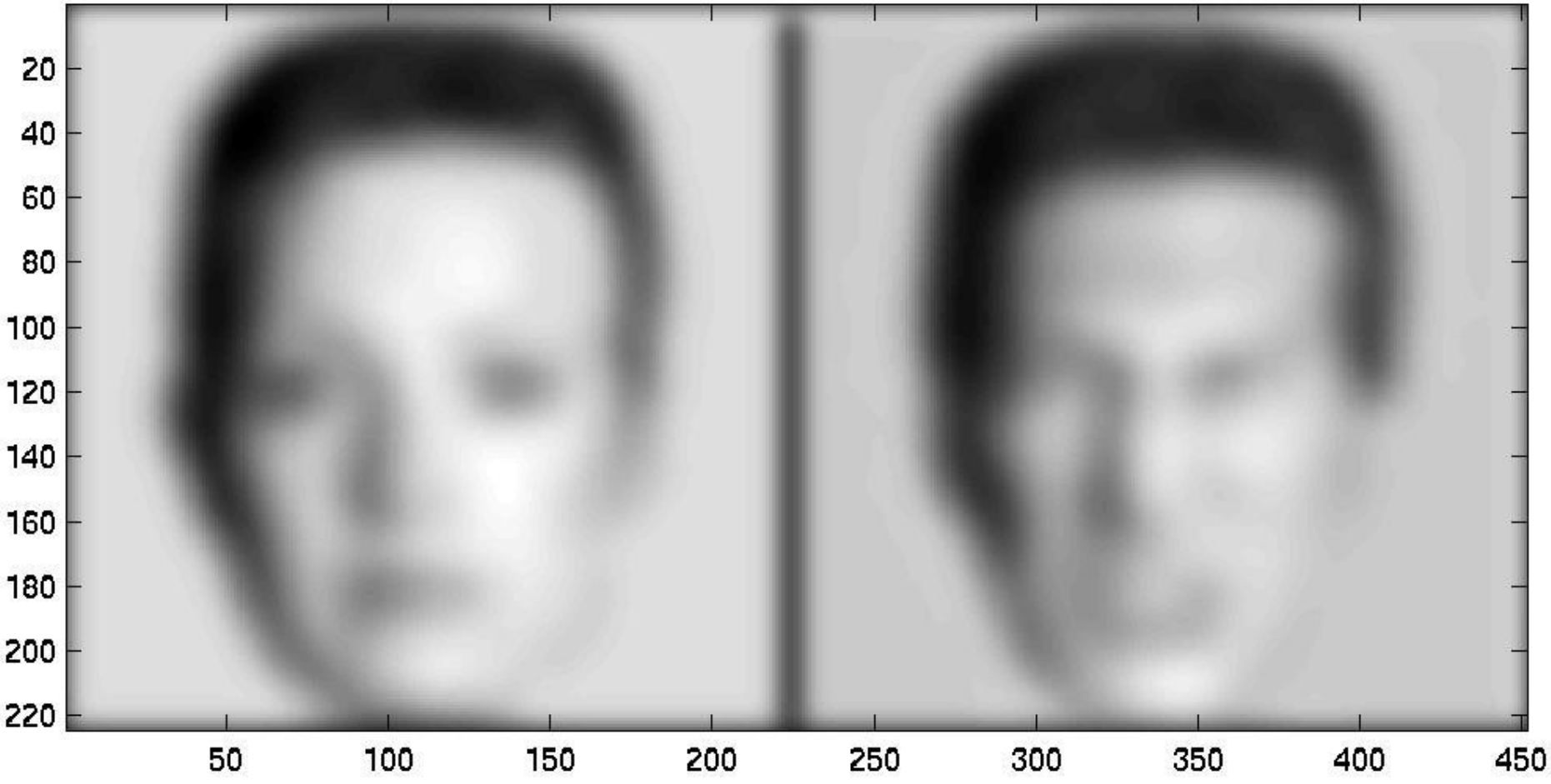
Salvador Dali, *"Gala Contemplating the Mediterranean Sea, which at 30 meters becomes the portrait of Abraham Lincoln"*, 1976

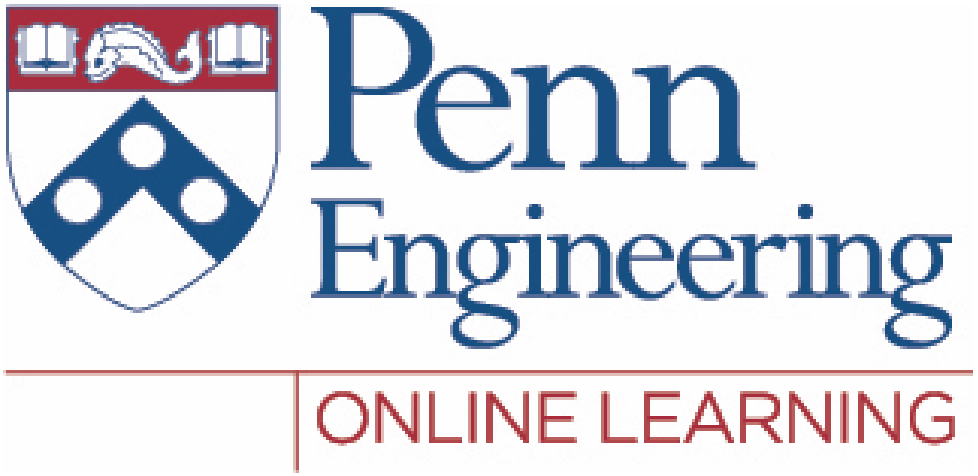


Salvador Dali, *"Gala Contemplating the Mediterranean Sea, which at 30 meters becomes the portrait of Abraham Lincoln"*, 1976

Image smoothing can remove noise, and also …


20
40
60
80
100
120
140
160
180
200
220
50
100
150
200
250
300
350
400
450


20
40
60
80
100
120
140
160
180
200
220
50
100
150
200
250
300
350
400
450

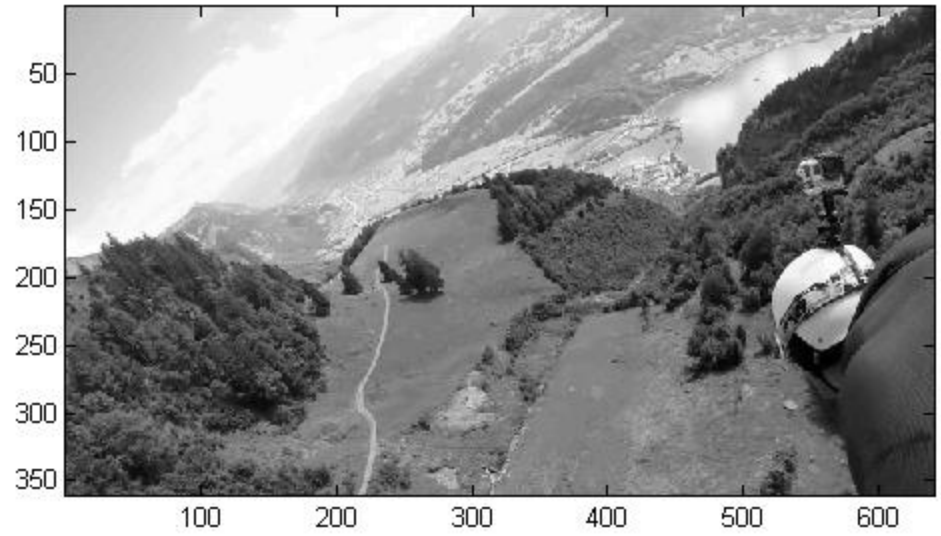


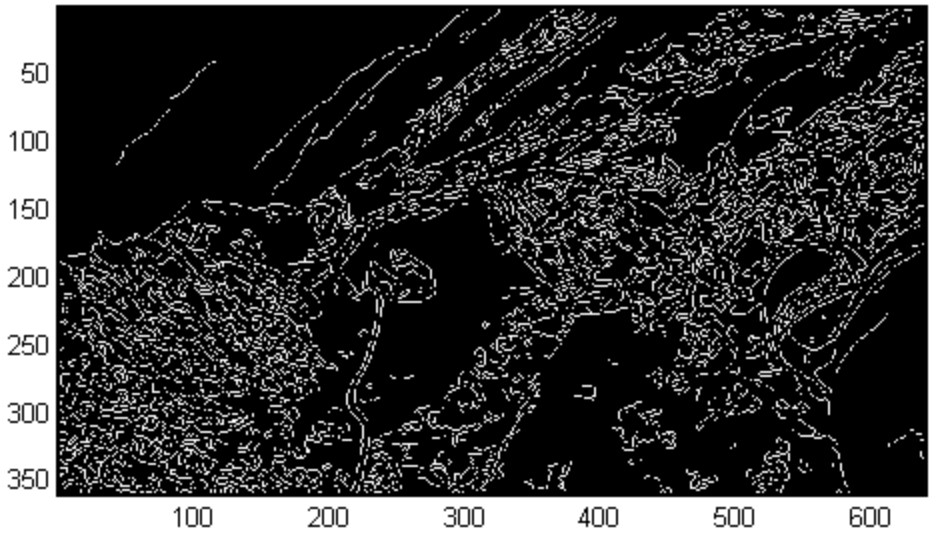
Video 2.14
Jianbo Shi

Edge Detection

Code

```
Jb = rgb2gray(J);

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

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





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.

Original image, I

Binary image indicating edge pixels

Edge map image, B

Canny Edge Detection

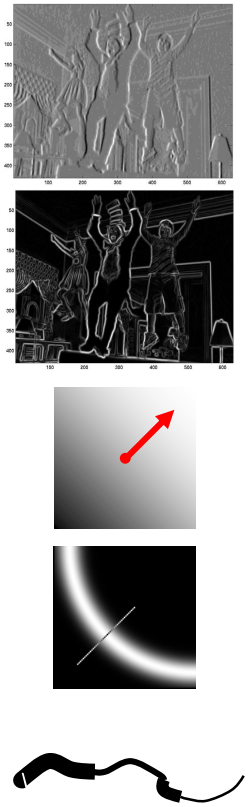
1. Filter image by derivatives of Gaussian

2. Compute magnitude of gradient

3. Compute edge orientation

4. Detect local maximum

5. Edge linking



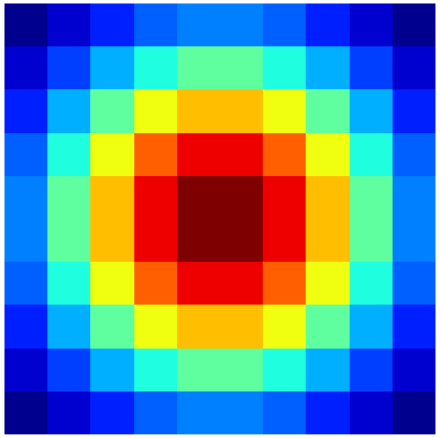
1) Compute Image Gradient

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

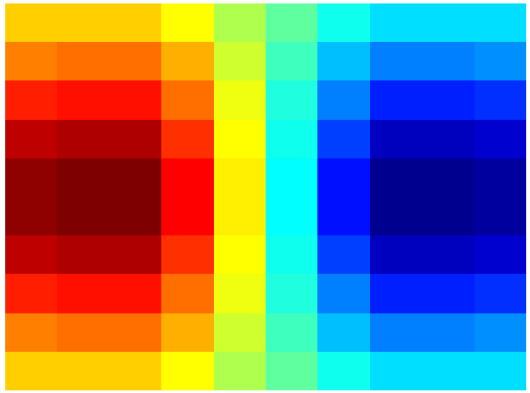
Edge Detection, Step 1, Filter out noise and compute derivative:

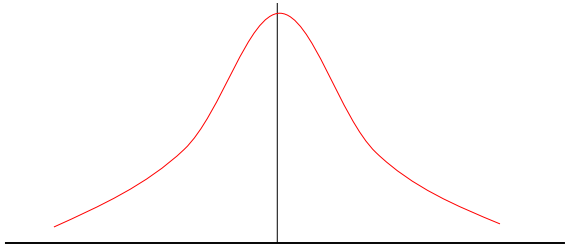
$$(\frac{\delta}{\delta x} \otimes G)$$

Gradient of Gaussian

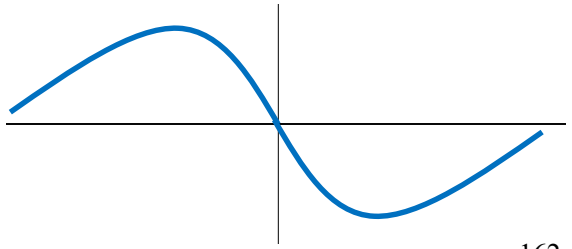


$$\frac{\delta}{\delta x}$$





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



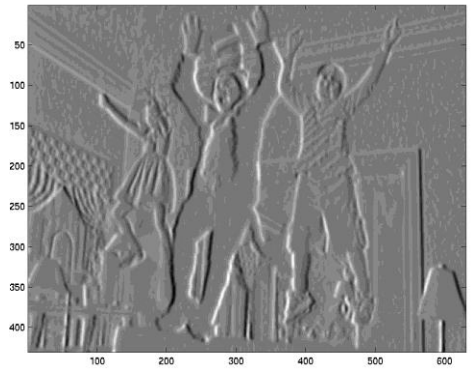
Edge Detection, Step 1, Filter out noise and compute derivative:

Image

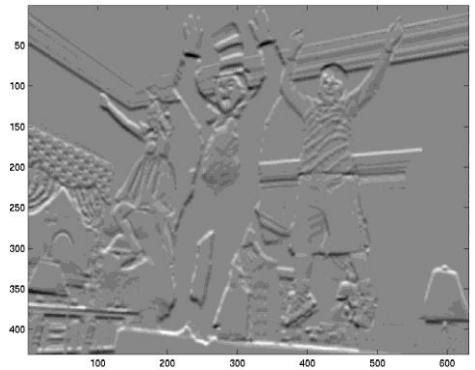
$$\otimes\left(\frac{\delta}{\delta x} \otimes G\right)$$

Smoothed Derivative

I_x



I_y

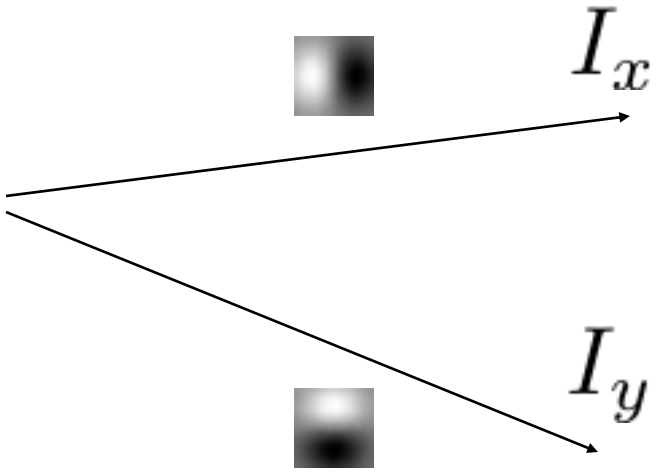


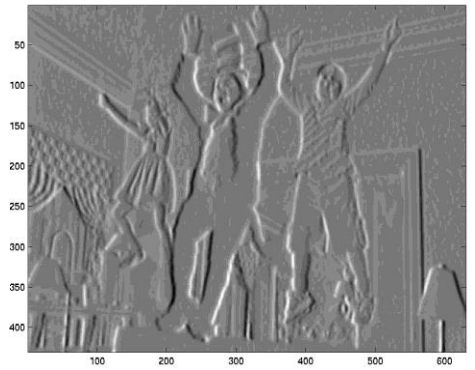
Edge Detection, Step 1, Filter out noise and compute derivative:

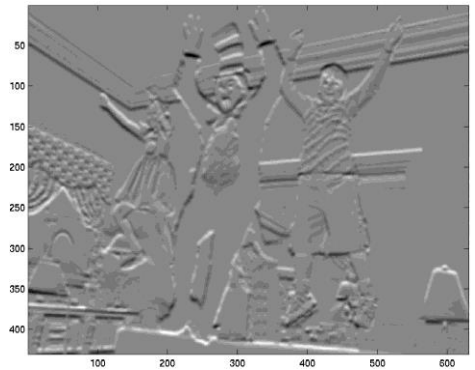
In matlab:

>> [dx,dy] = gradient(G); % G is a 2D gaussain

>> Ix = conv2(I,dx,'same'); Iy = conv2(I,dy,'same');

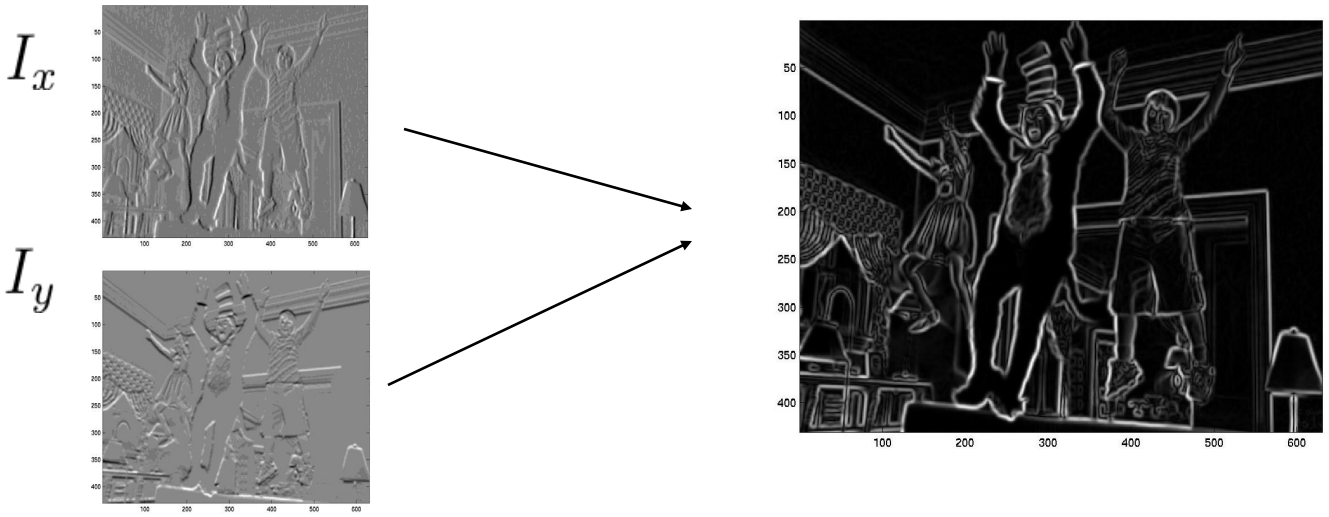




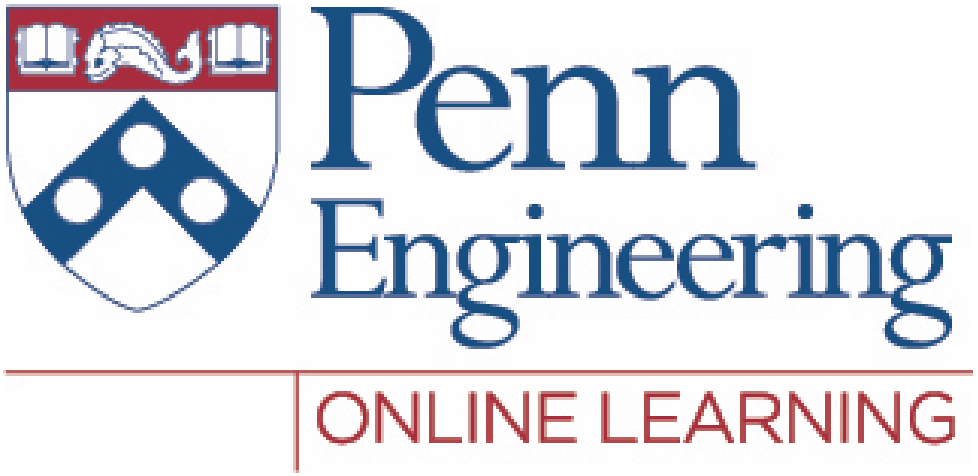


Edge Detection: Step 2
Compute the magnitude of the gradient

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



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



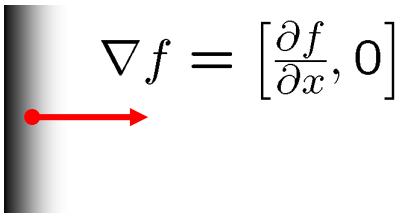
Video 2.15
Jianbo Shi

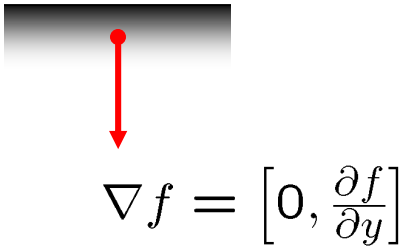
Finding the orientation of the edge

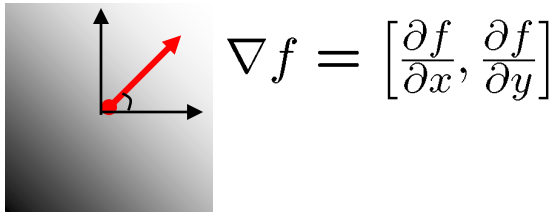
- The gradient of an image:

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

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







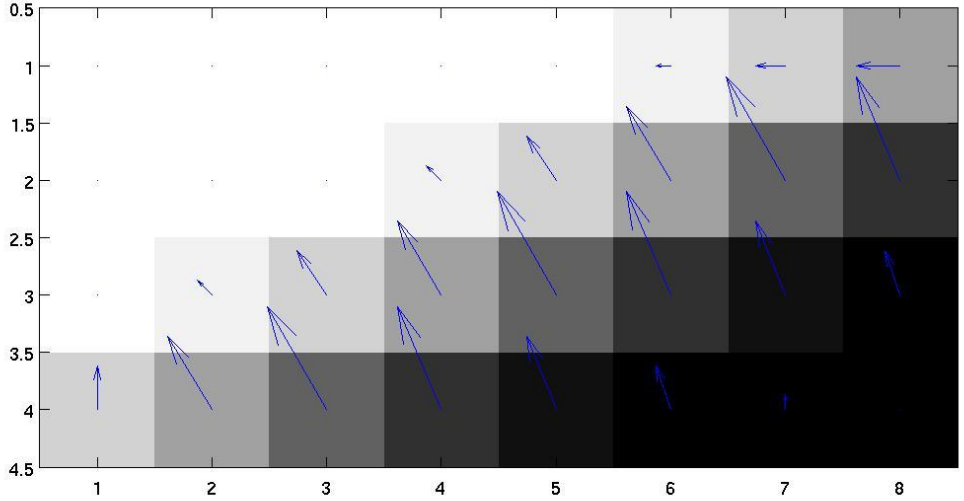
- The image gradient direction is given by:

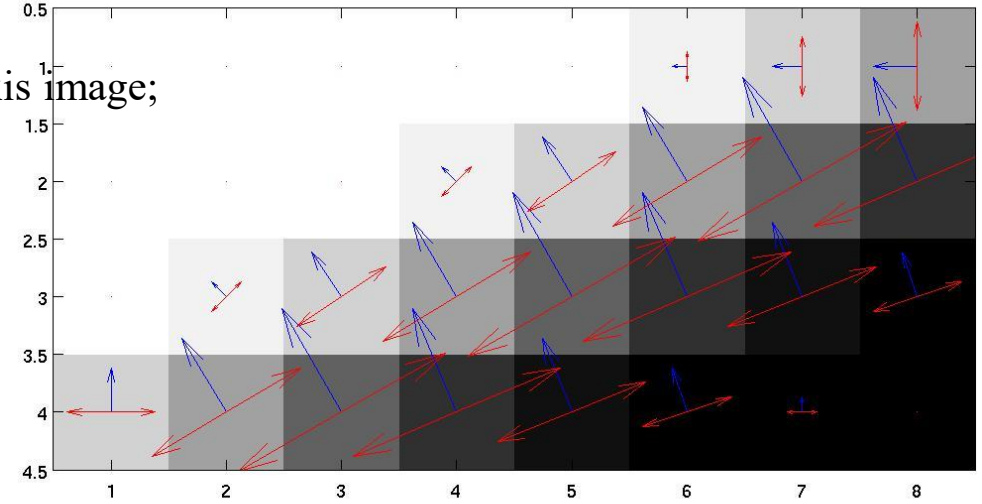
$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

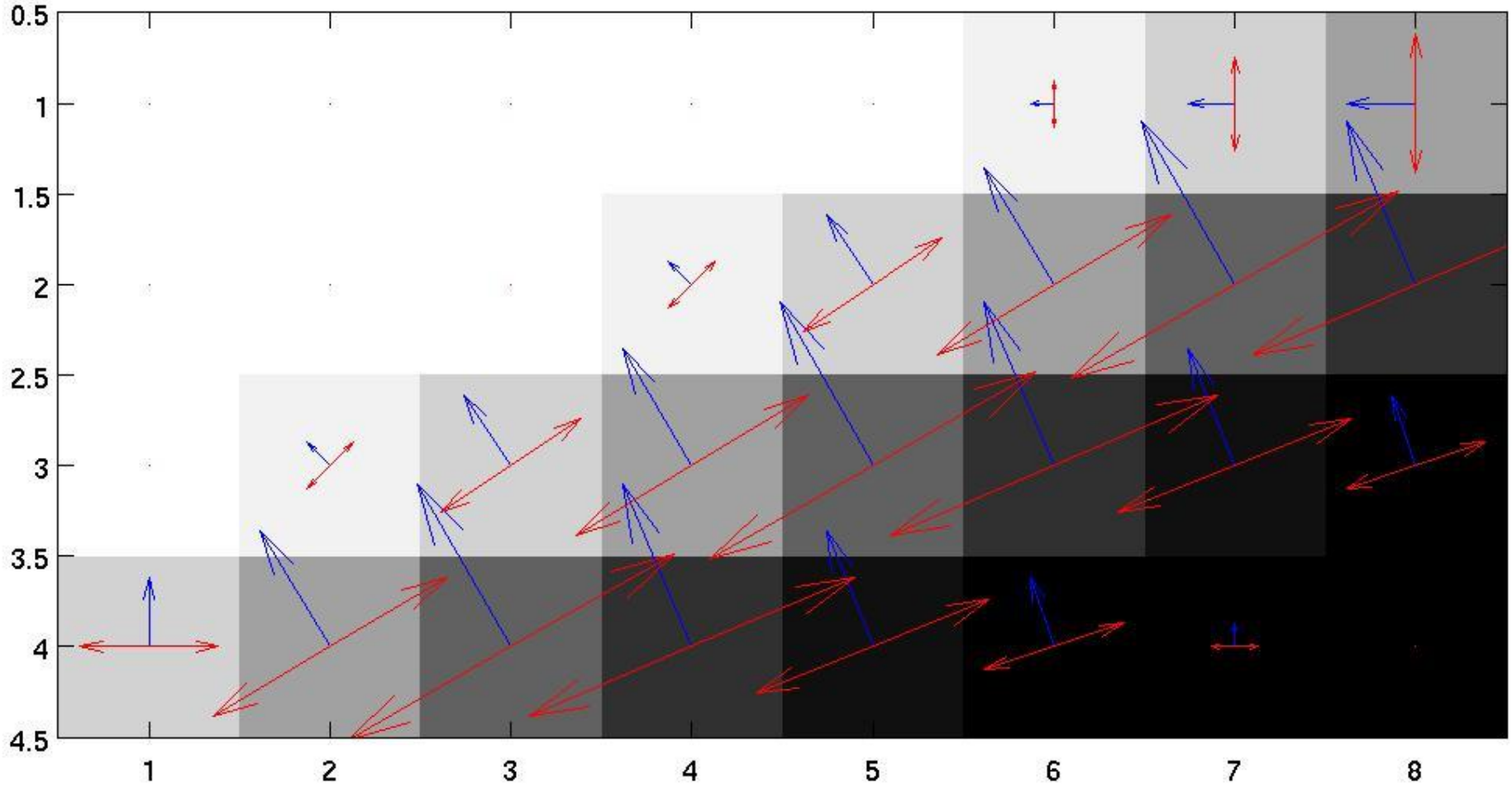
  - how does this relate to the direction of the edge?

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

```
%% 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');
```






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

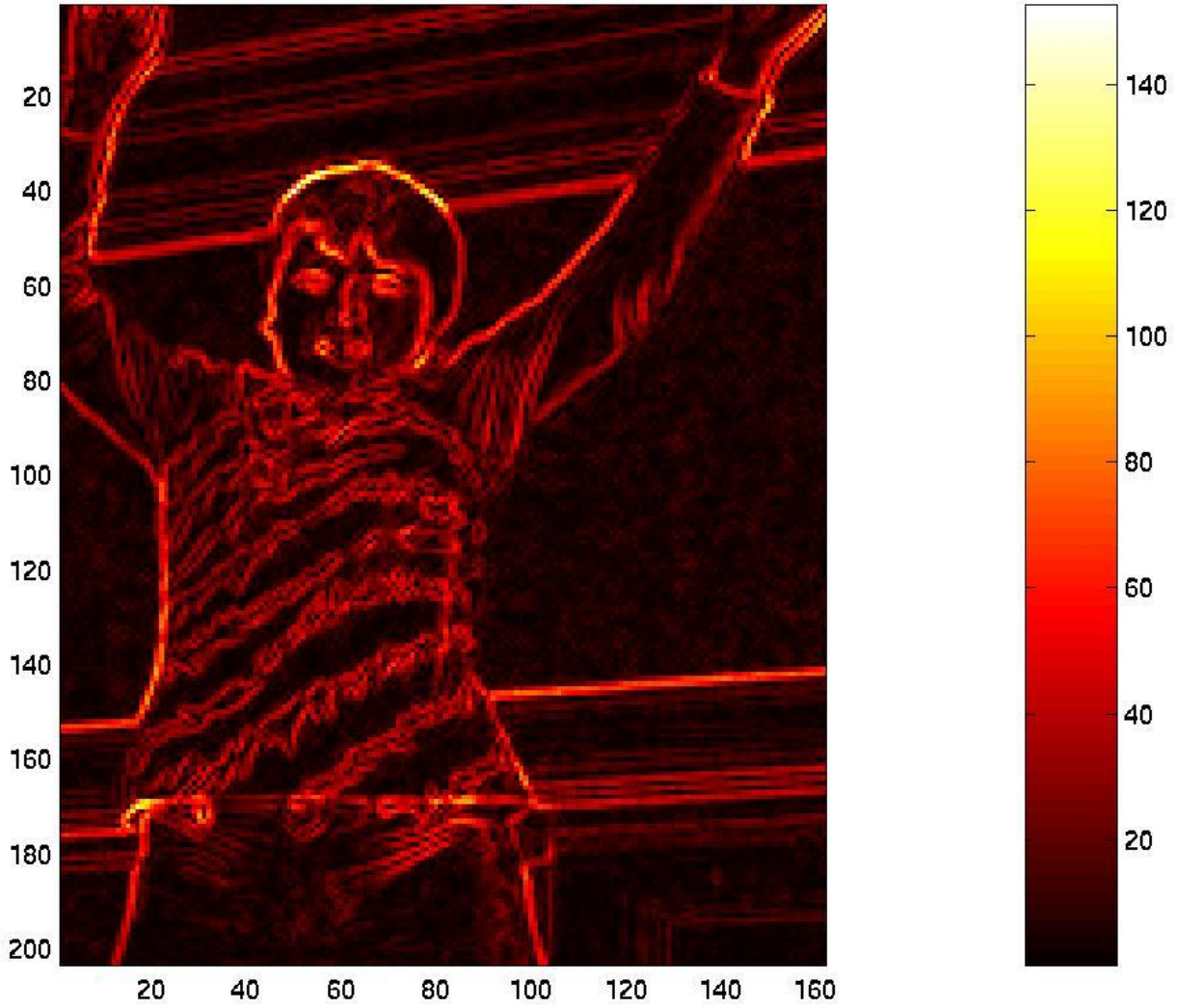
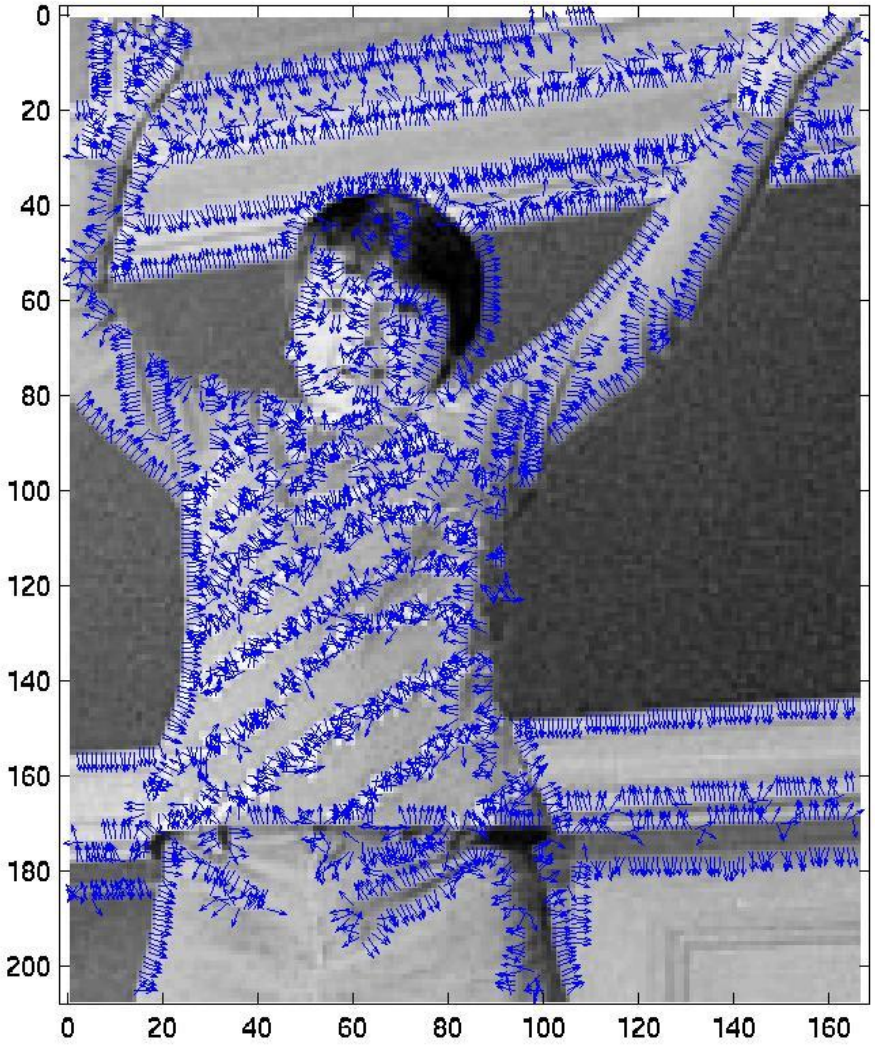
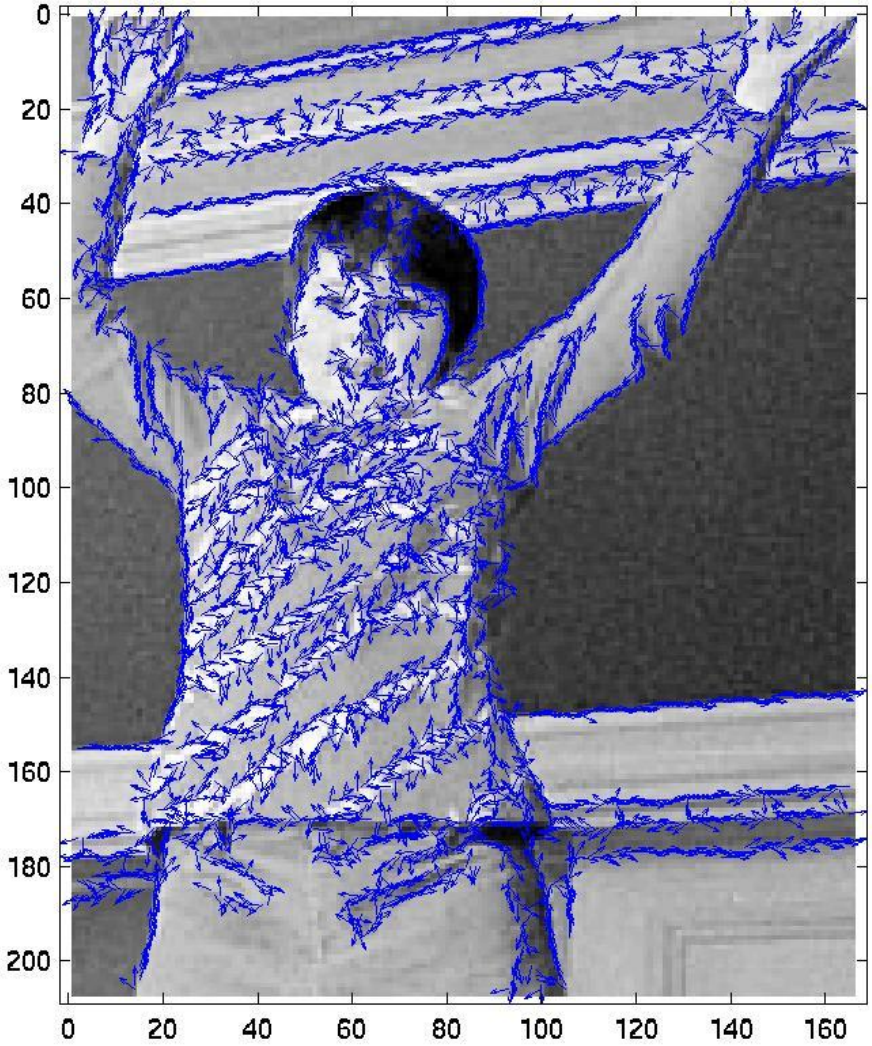


image gradient direction:



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

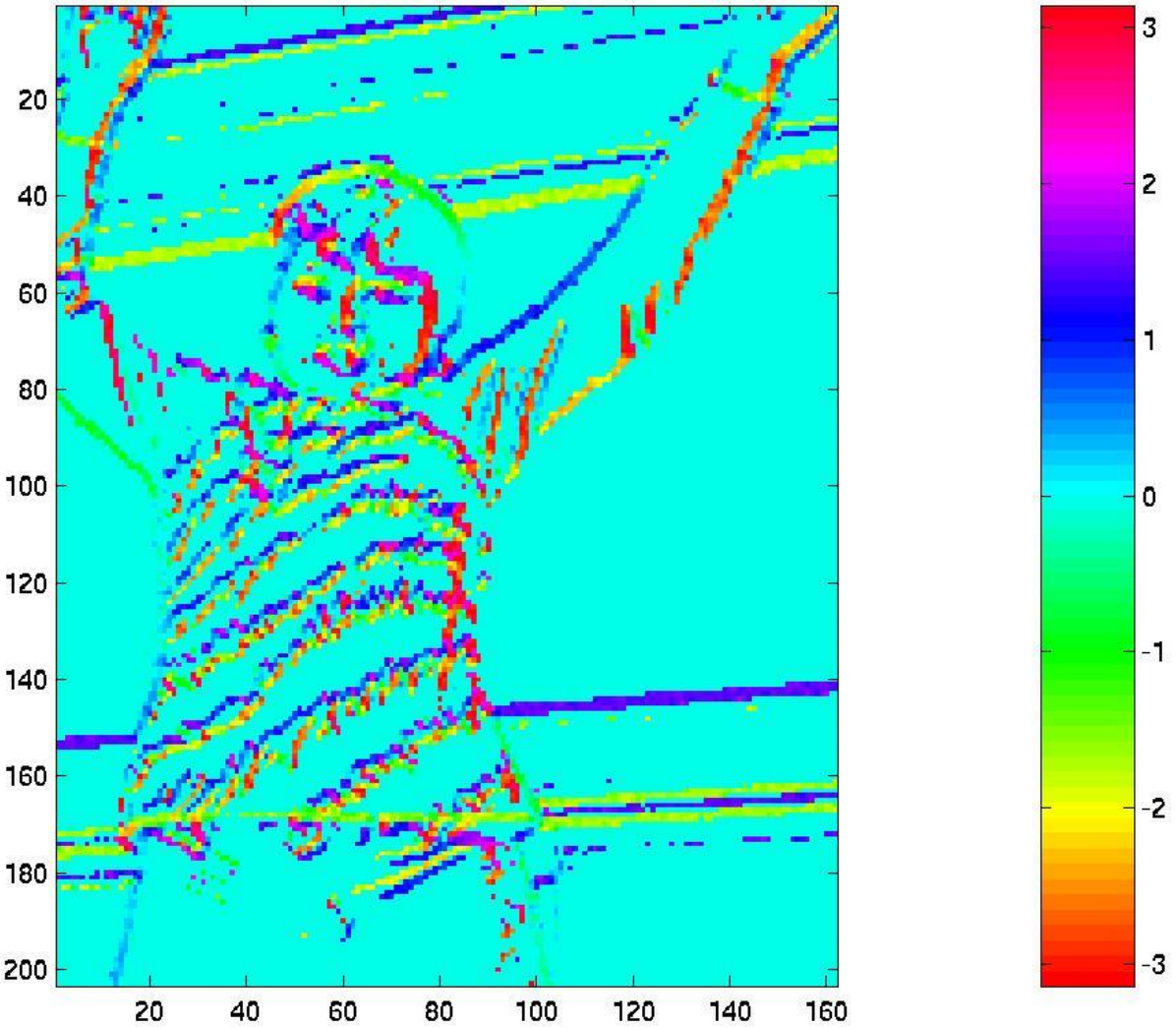
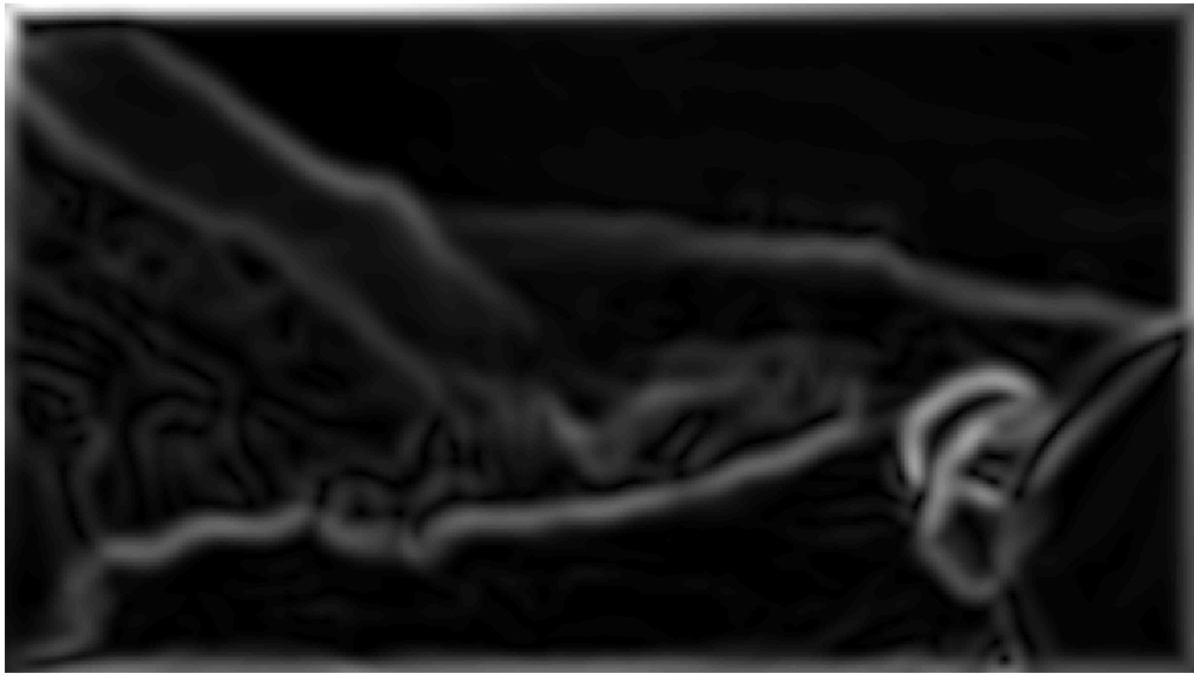
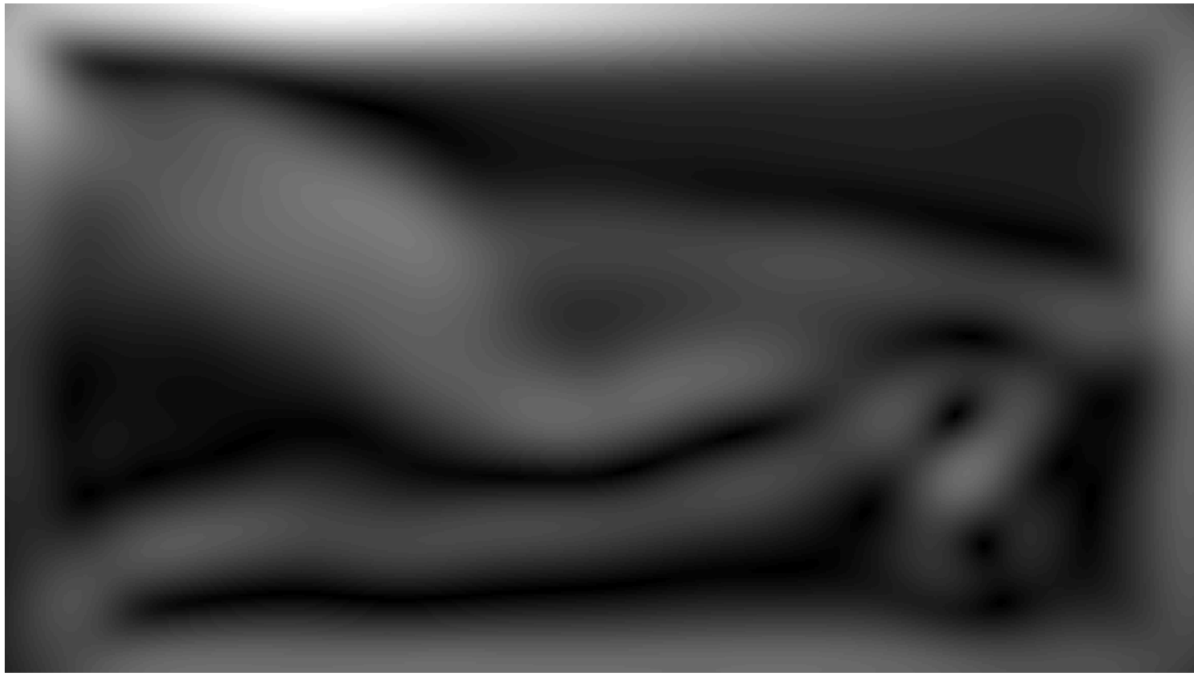
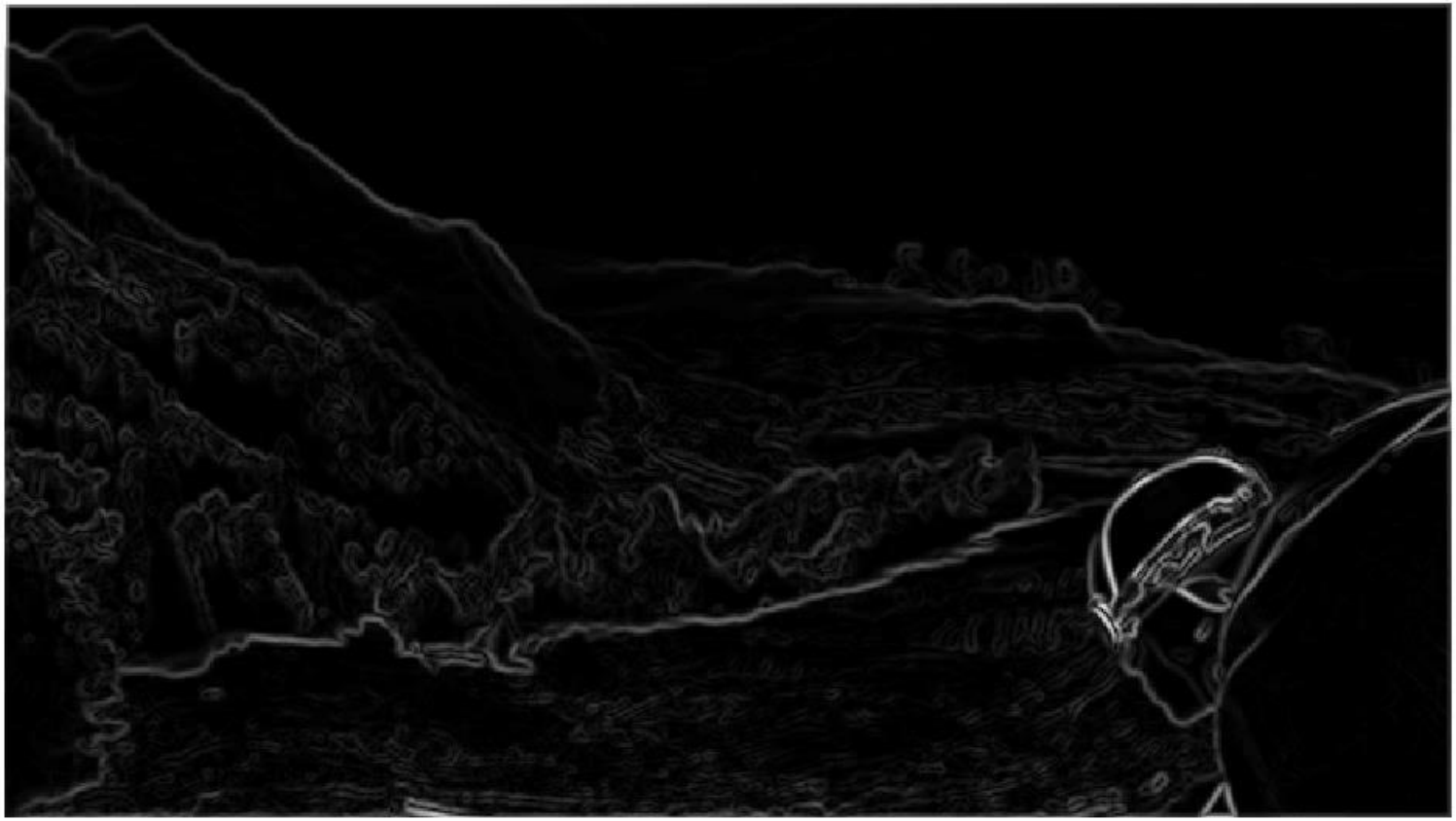


Image Scale

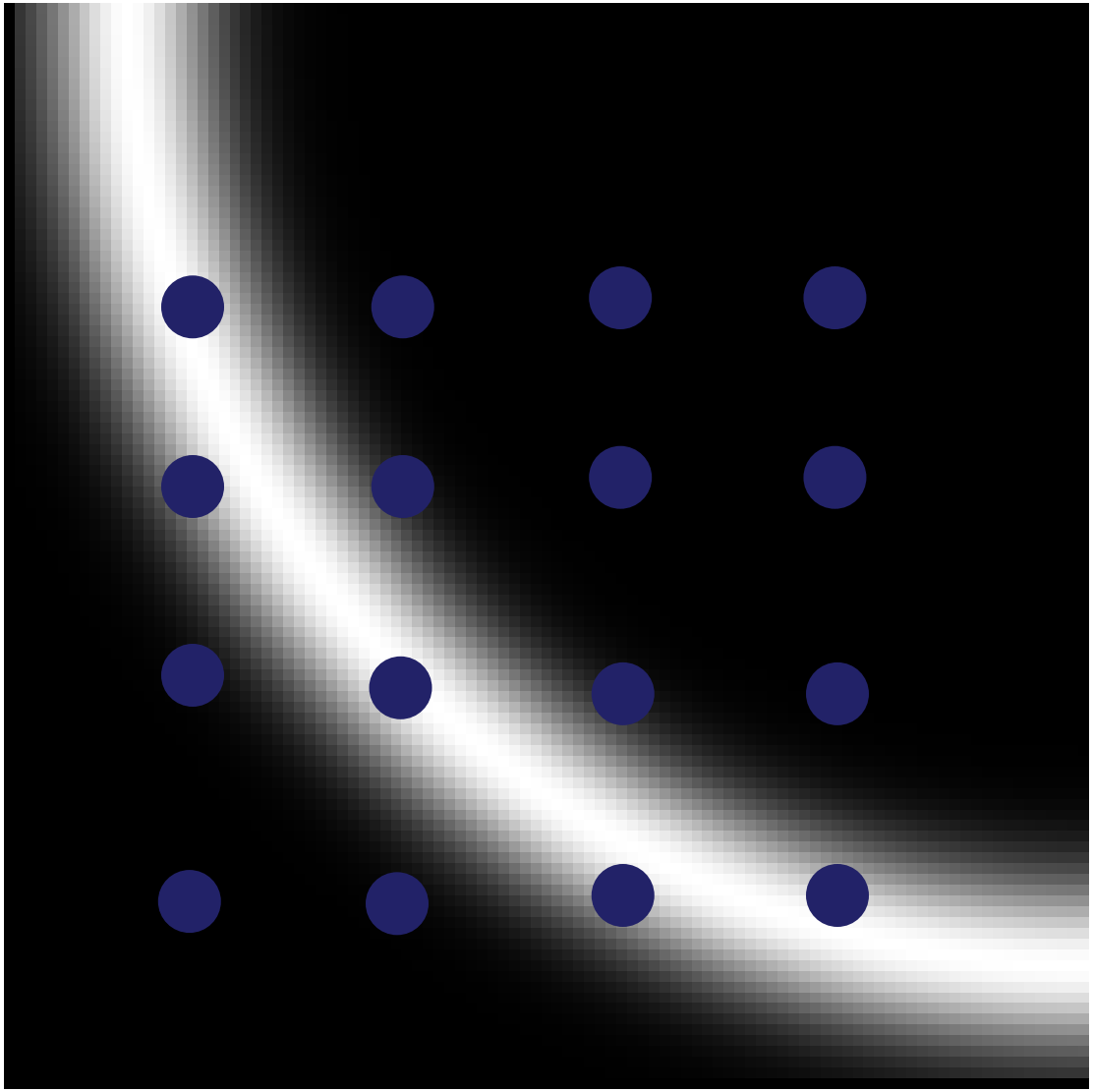






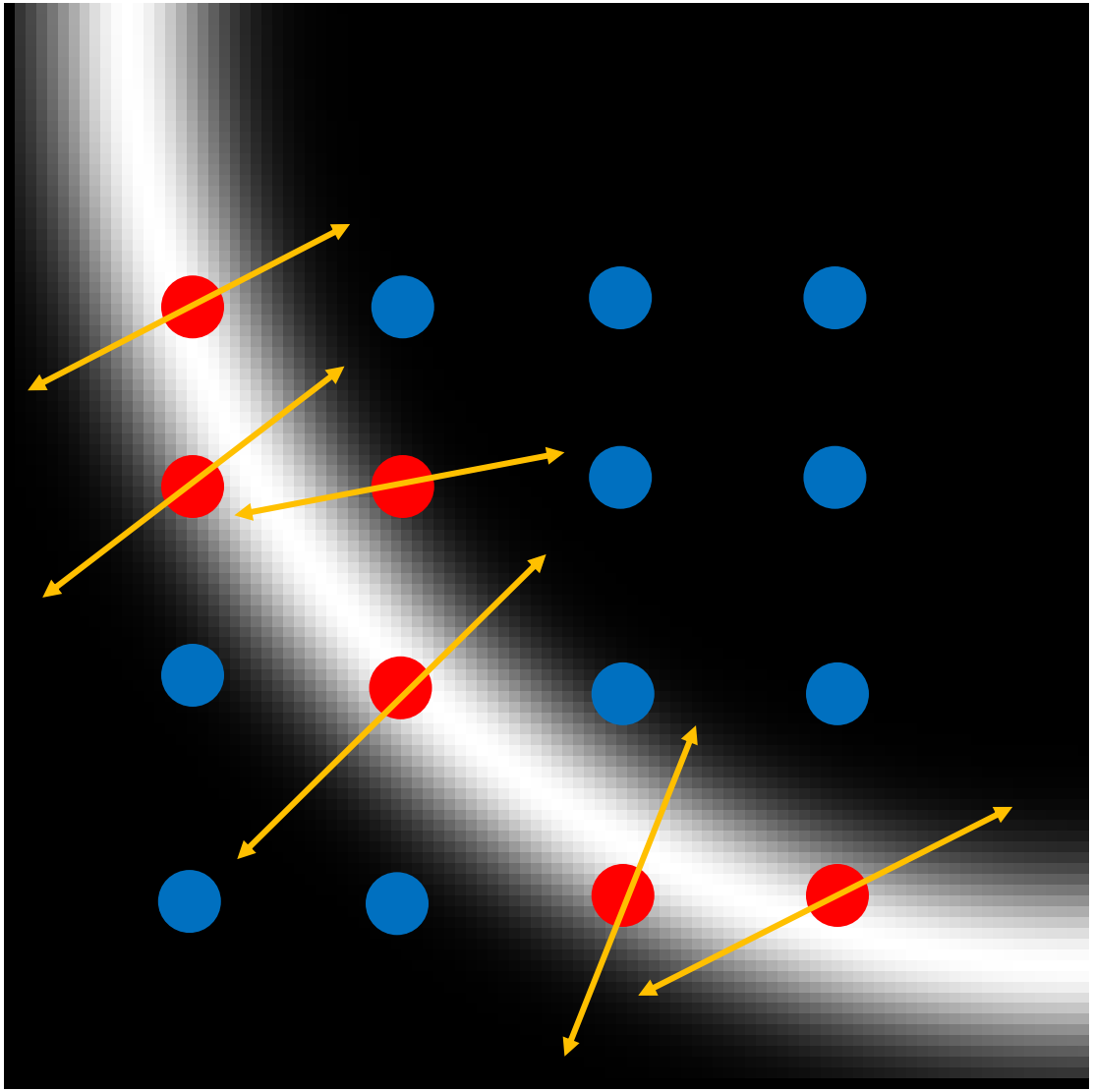
Different scale of image encodes different edge response.

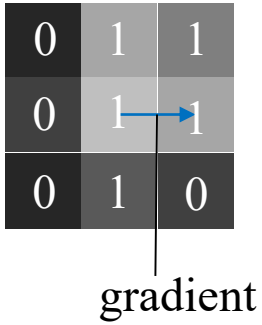
Discretized pixel locations



(Forsyth & Ponce)

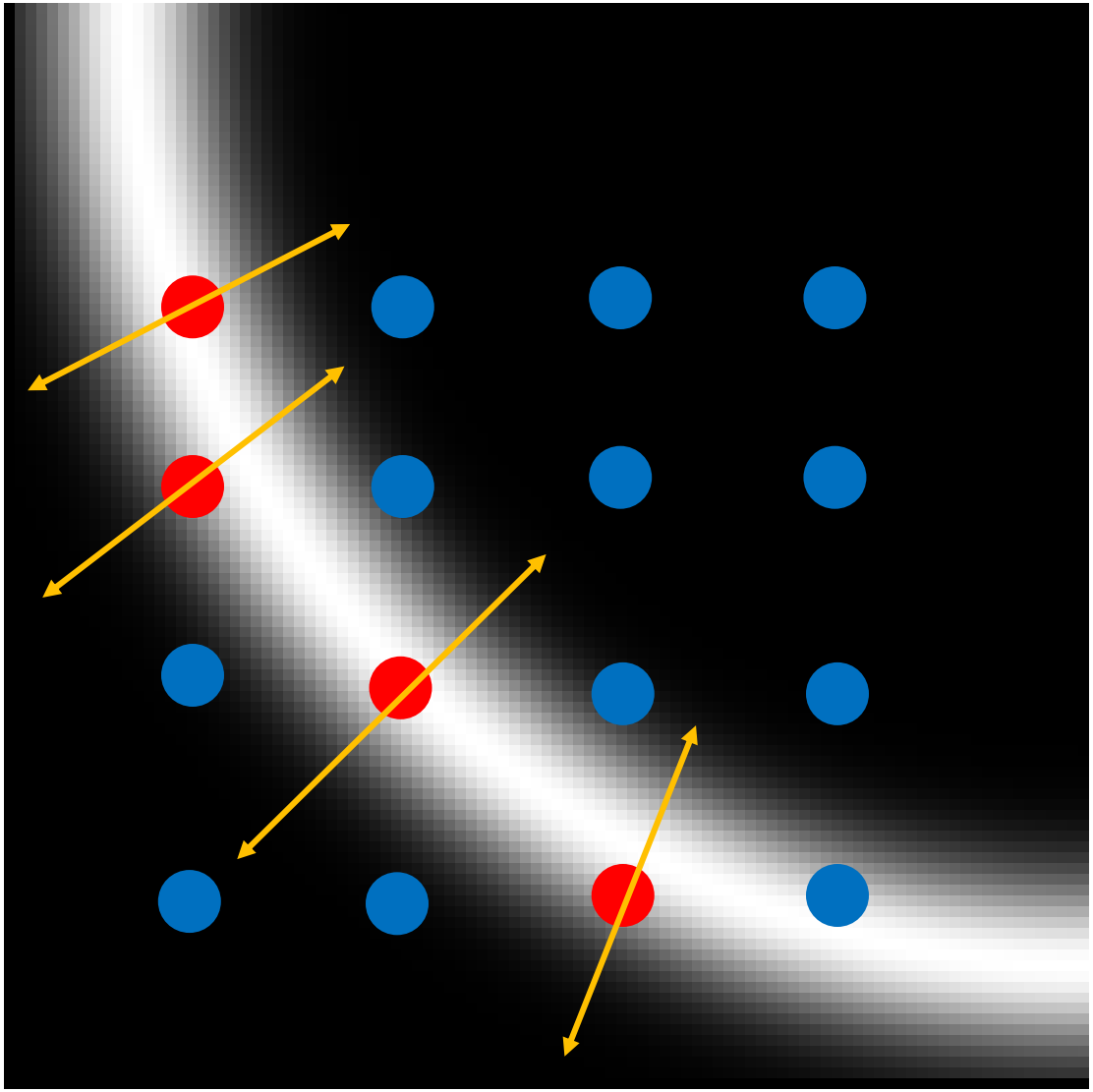
Thesholding

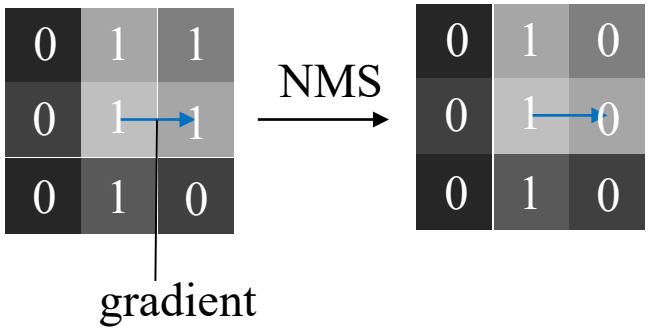




(Forsyth & Ponce)

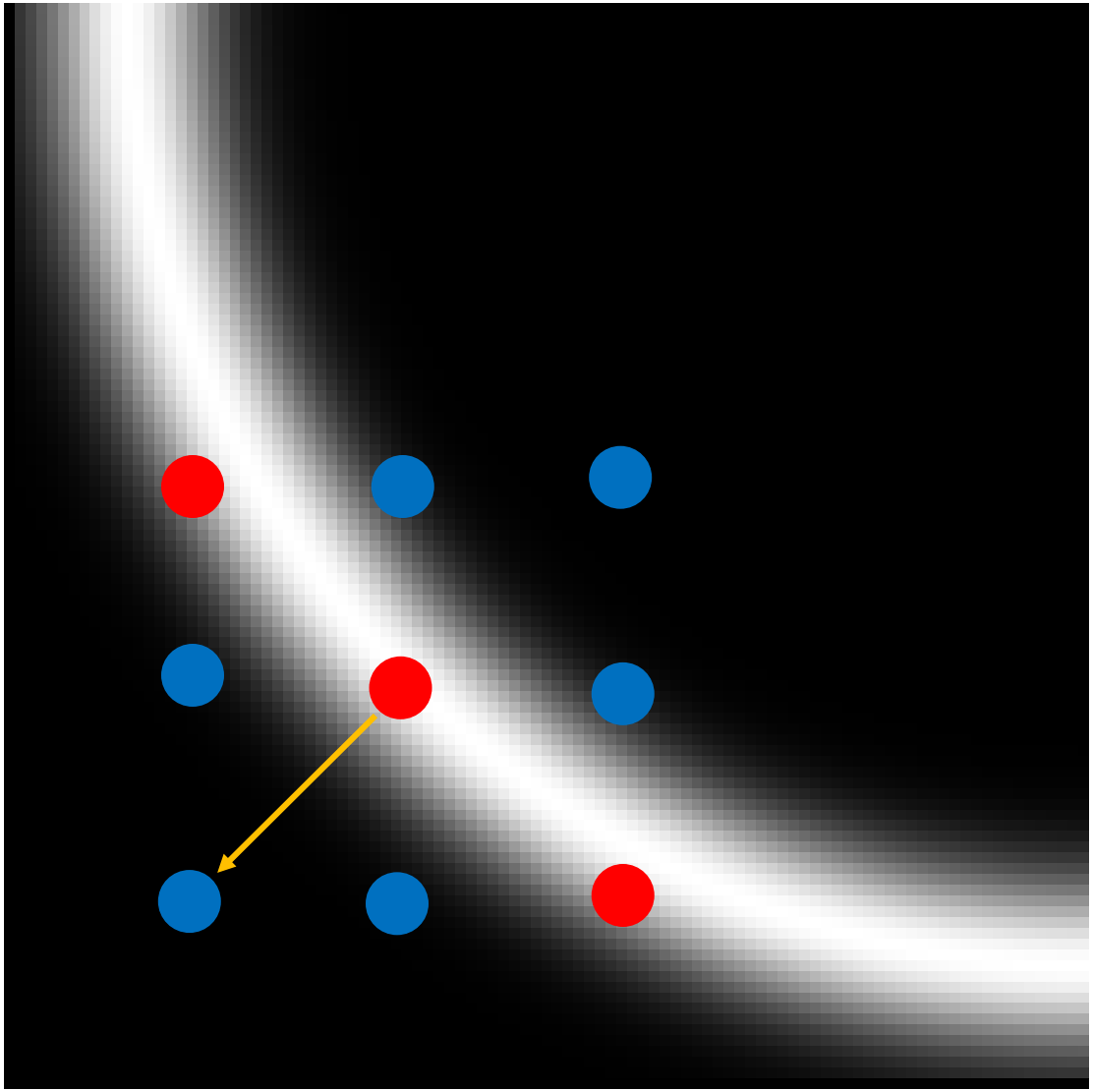
Non-maximum suppression along the line of the gradient

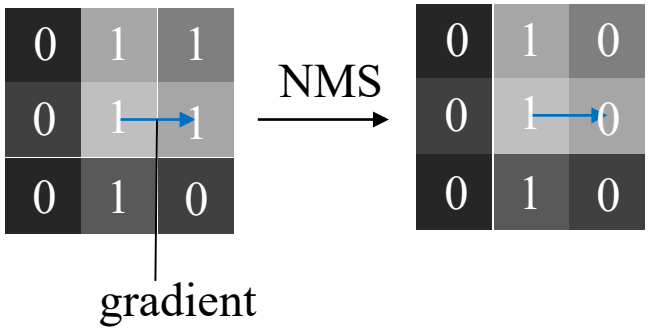




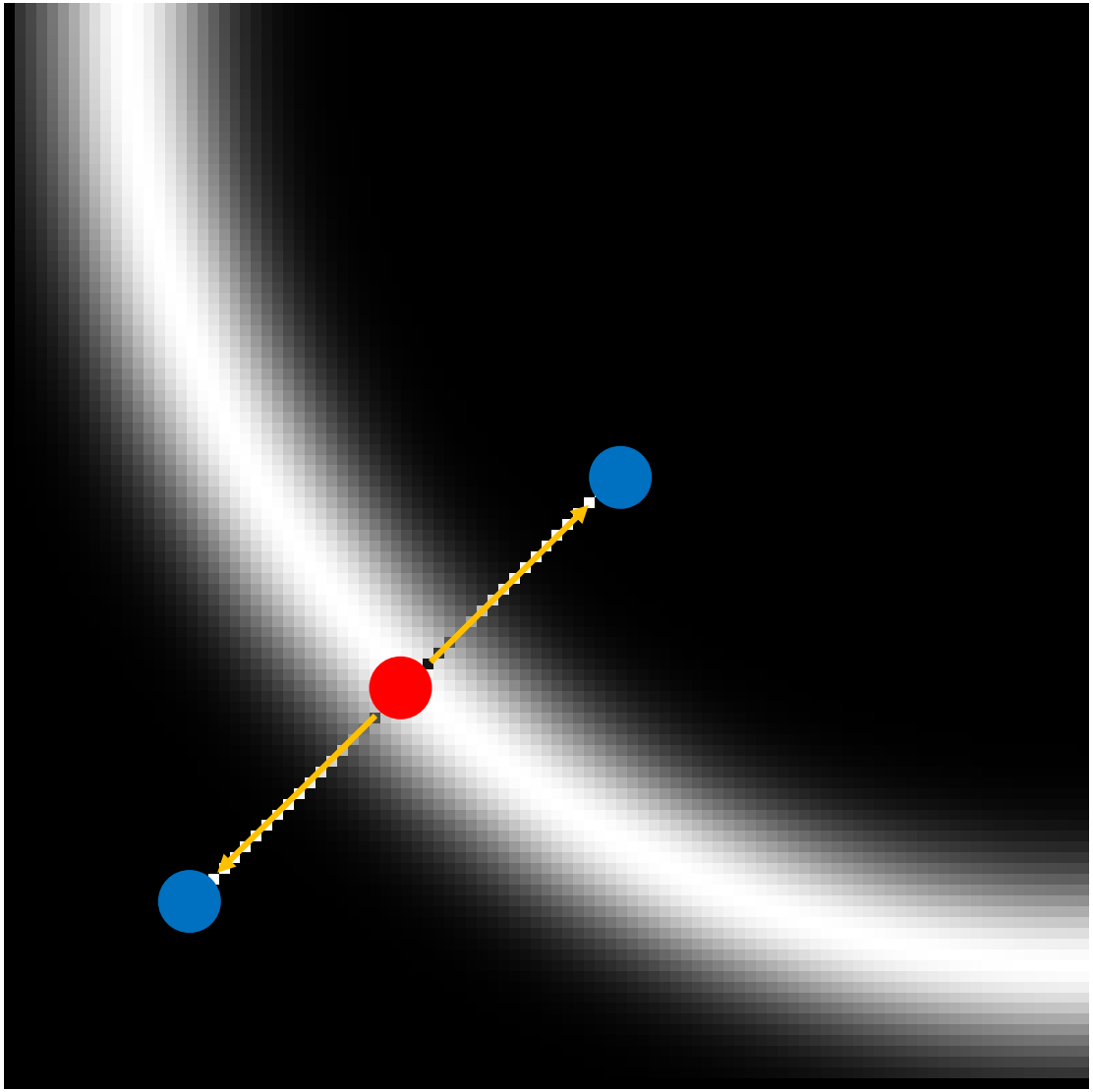
(Forsyth & Ponce)

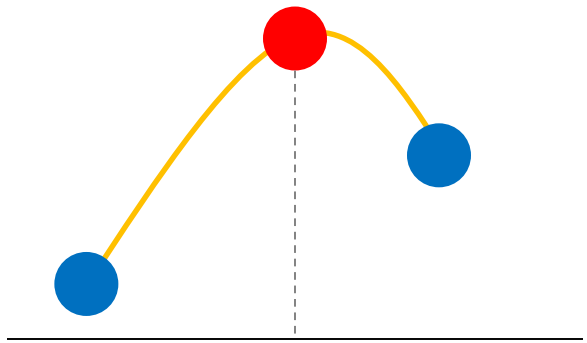
Gradient direction





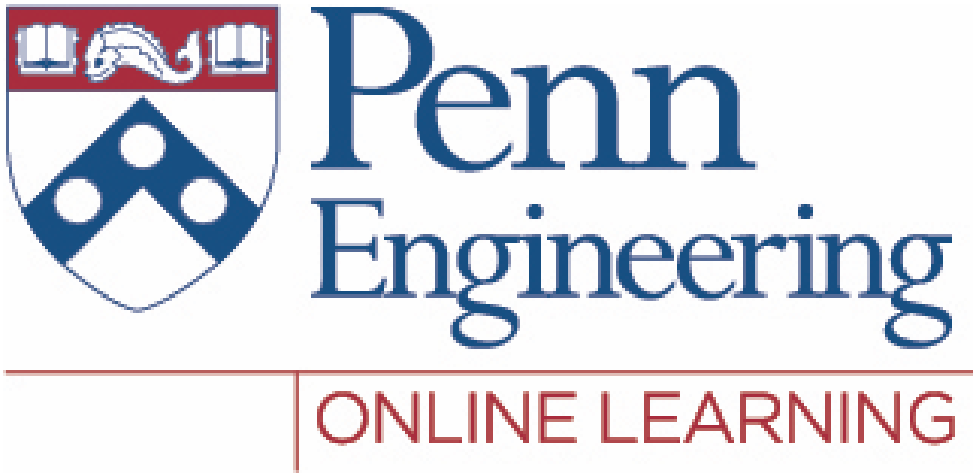
(Forsyth & Ponce)





Local maximum

(Forsyth & Ponce)



Video 2.16
Jianbo Shi

Edge Detection

Python:

```
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
from skimage.feature import canny
```

Edge Detection

Python:

```
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
from skimage.feature import canny

Jb = np.array(Image.open('demo.png').convert('L'))
plt.figure(); plt.imshow(Jb, cmap='gray')
bw = canny(Jb)
```

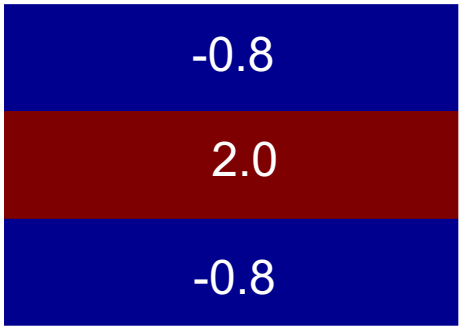
Numerical Image Filtering

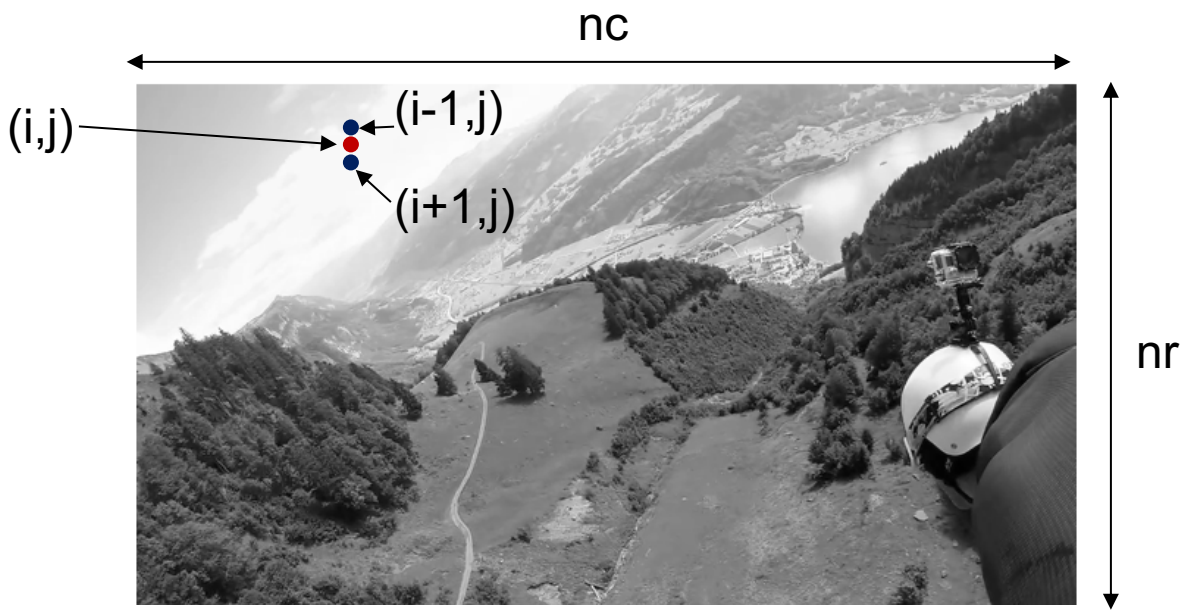
Python: (Looping through all pixels)

```
nr, nc = Jb.shape[0], Jb.shape[1]
J_out = np.zeros((nr,nc))
for i in range(nr):
        for j in range(nc):
                if (i < nr - 1) & (i > 0):
                        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]
plt.imshow(J_out, cmap='gray')
plt.show()
```

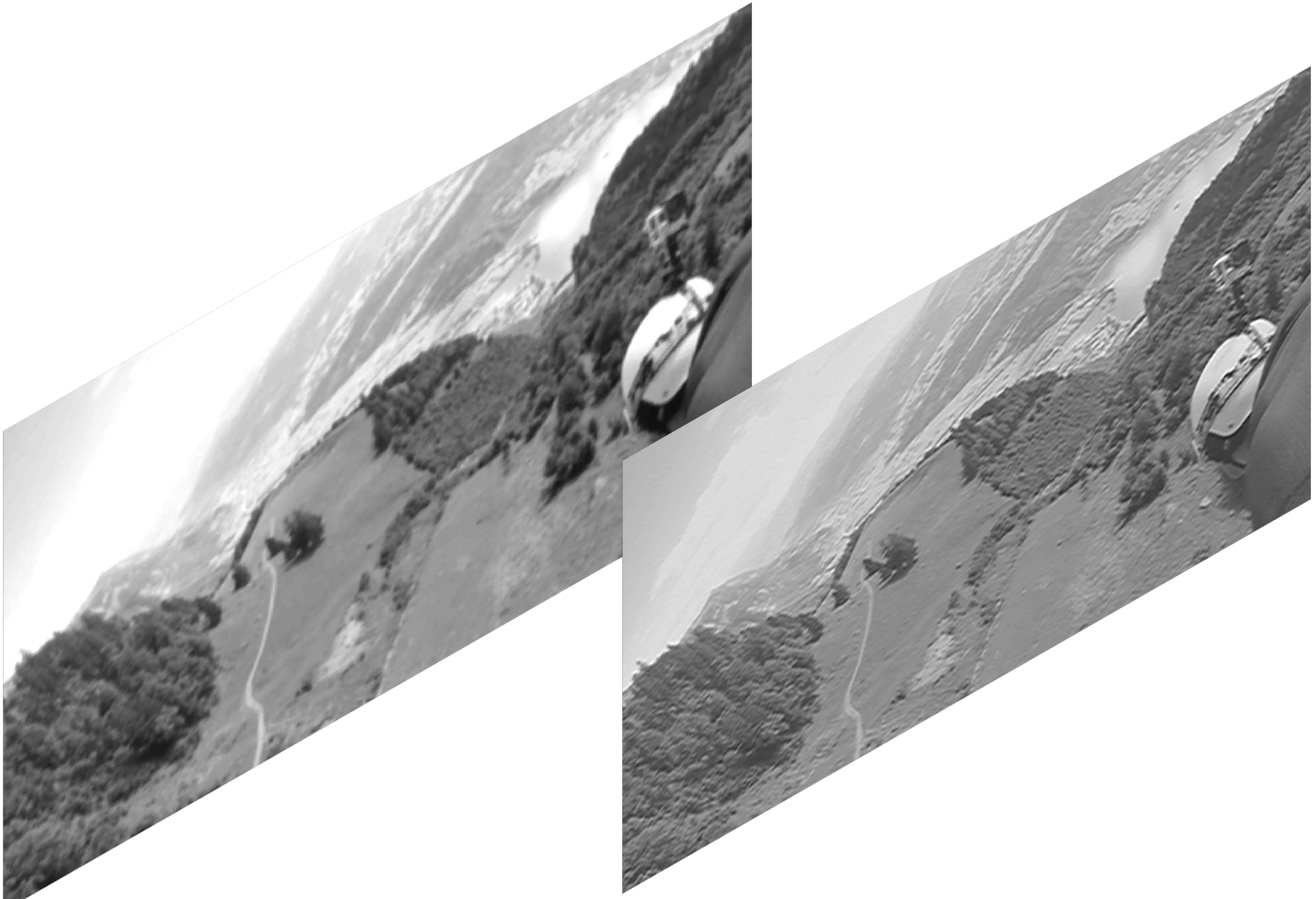
Computation time: 1.45 sec

Filter





Numerical Image Filtering

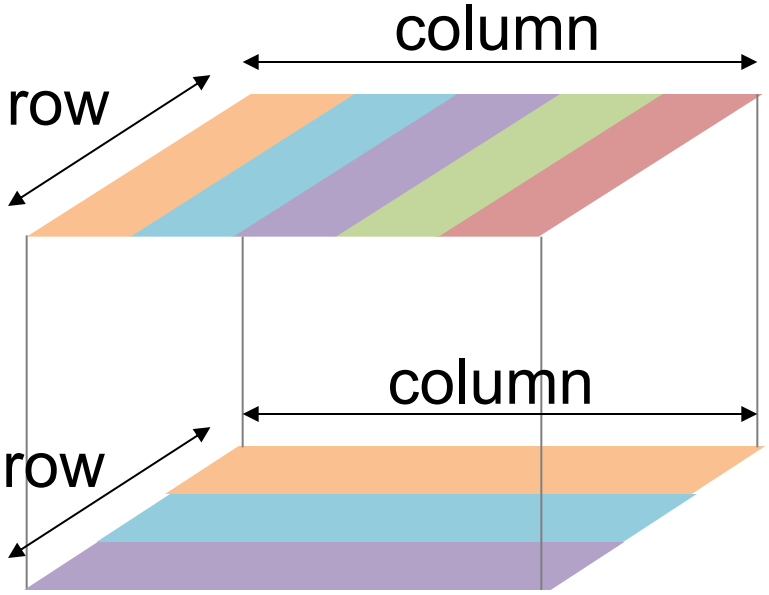


Convolution without Looping using meshgrid

Python:

```
import numpy as np
nx, ny = (5,3)
x = np.linspace(0,4,nx)
y = np.linspace(0,2,ny)
[x,y] = np.meshgrid(x, y)
print('x:\n', x)
print('y:\n', y)
```

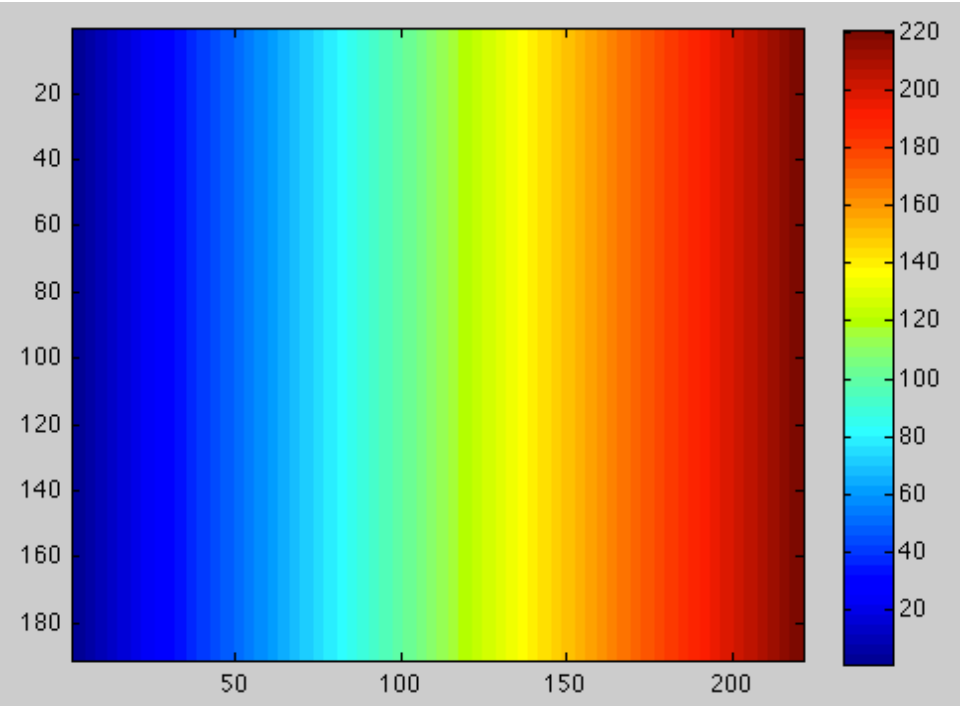
```
x:
 [[0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]]
y:
 [[0. 0. 0. 0. 0.]
 [1. 1. 1. 1. 1.]
 [2. 2. 2. 2. 2.]]
```

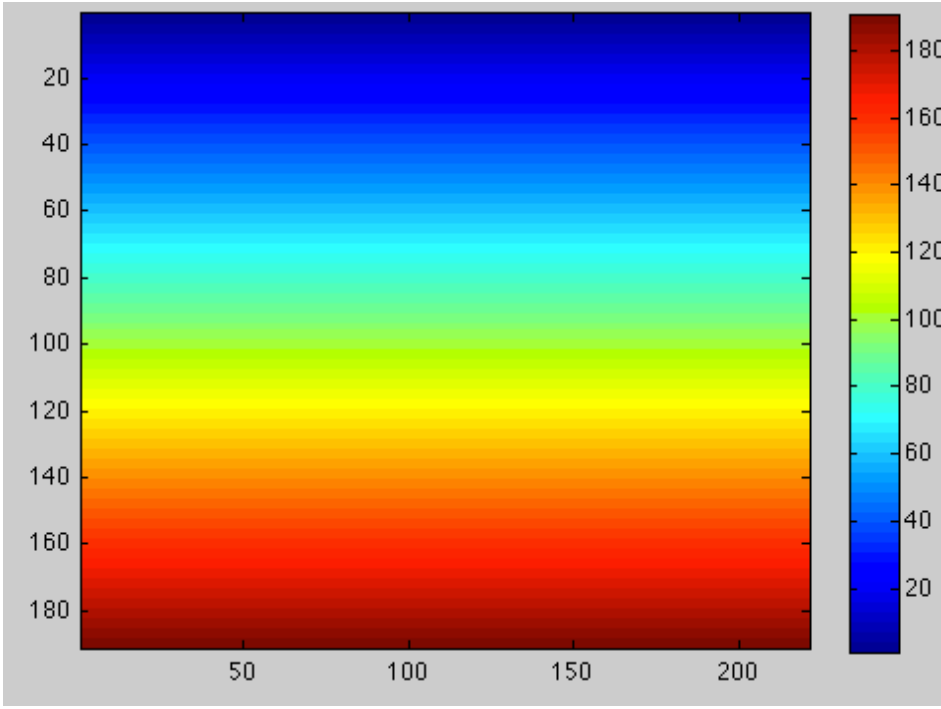


Python:

```
nx, ny = (nc,nr)
x = np.linspace(0,nc-1,nx)
y = np.linspace(0,nr-1,ny)
[x,y] = np.meshgrid(x, y)
```

```
# visualization (you may change cmap to
the color you like)
fig, ax = plt.subplots()
im = ax.imshow(x, cmap='brg')
fig.colorbar(im, orientation='vertical')
plt.show()
```





Convolution without Looping using meshgrid

Python:

```
x,y = np.meshgrid(np.arange(nc), np.arange(nr))
y_up = y - 1
y_down = y + 1
y_up = np.clip(y_up, 0, nr - 1)
y_down = np.clip(y_down, 0, nr - 1)

coor_up = np.stack([y_up.flatten(), x.flatten()])
ind_up = np.ravel_multi_index(coor_up, (nr, nc))

coor_down = np.stack([y_down.flatten(), x.flatten()])
ind_down = np.ravel_multi_index(coor_down, (nr, nc))

Jb = Jb.flatten()
J_out = 2 * Jb - 0.8 * Jb[ind_up] - 0.8 * Jb[ind_down]
J_out = J_out.reshape(nr,nc)
plt.imshow(J_out, cmap='gray')
plt.show()
```

Computation time: 0.01787 sec VS 1.45 sec (for loop)

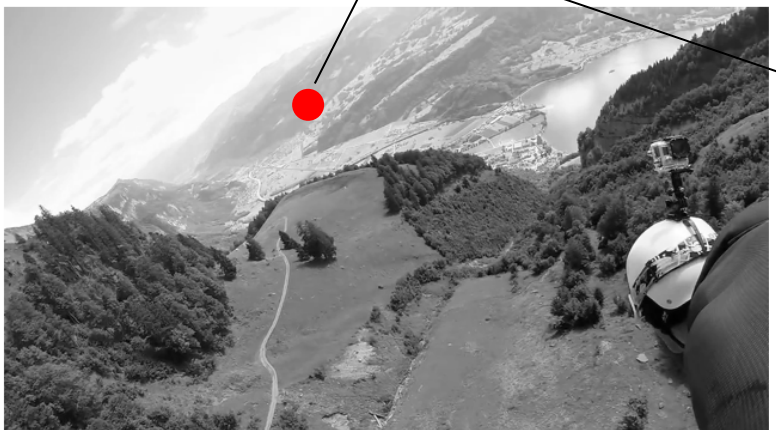
Convolution without Looping using meshgrid

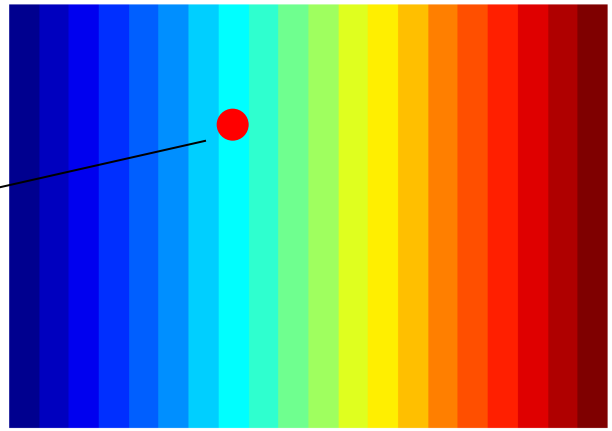
Python:

```
x,y = np.meshgrid(np.arange(nc), np.arange(nr))
```

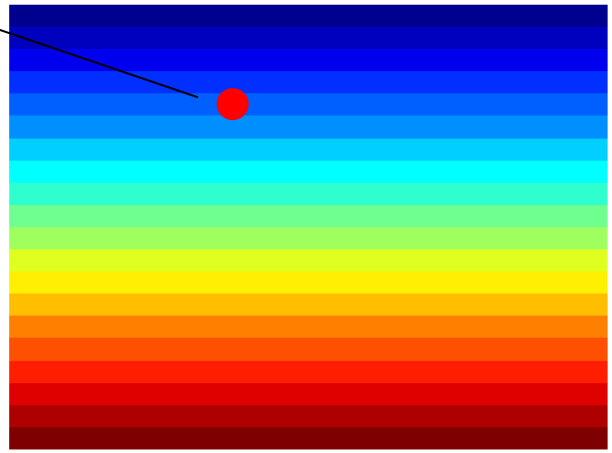
x and y are subscript indice

Jb_{xy}





x

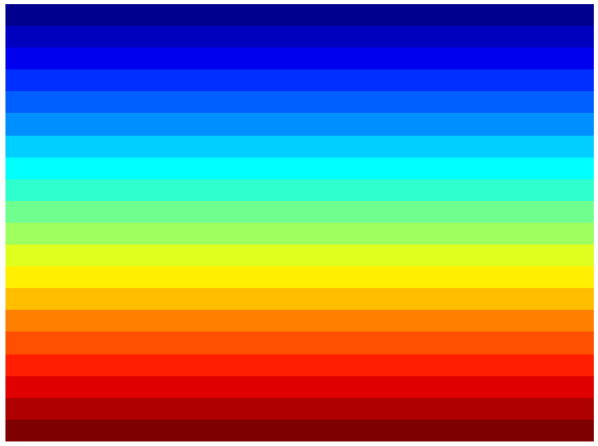


y

Convolution without Looping using meshgrid

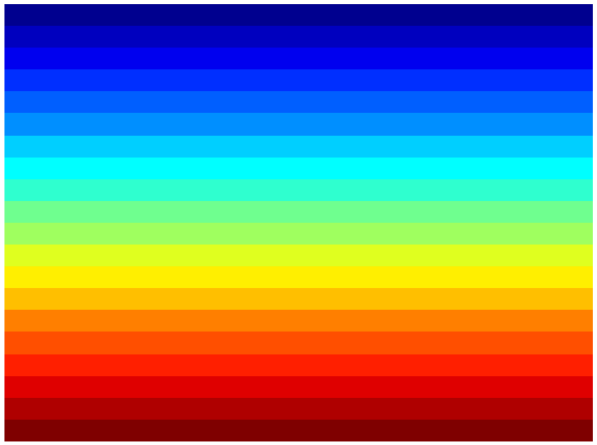
Python:

```
x,y = np.meshgrid(np.arange(nc), np.arange(nr))
y_up = y - 1
y_down = y + 1
```



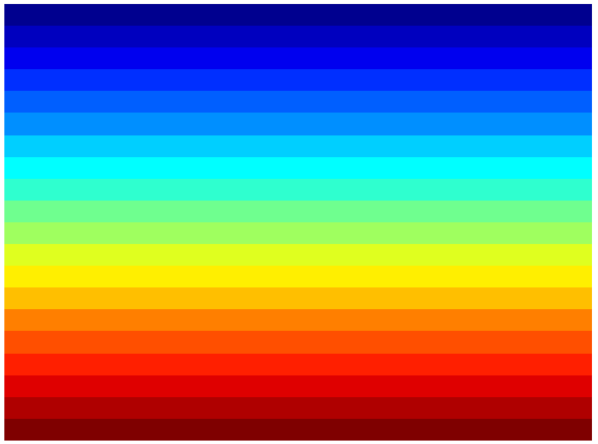
y_up =

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



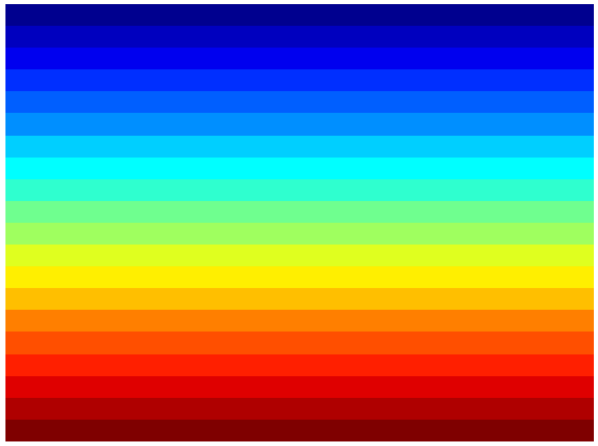
y_down =

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

Convolution without Looping using meshgrid

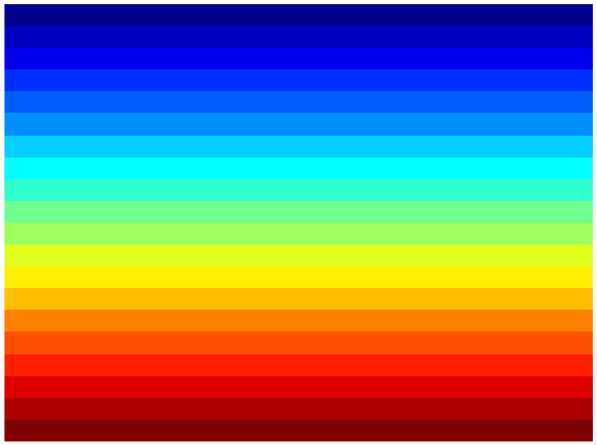
Python:

```
x,y = np.meshgrid(np.arange(nc), np.arange(nr))
y_up = y - 1
y_down = y + 1
y_up = np.clip(y_up, 0, nr - 1)
y_down = np.clip(y_down, 0, nr - 1)
```



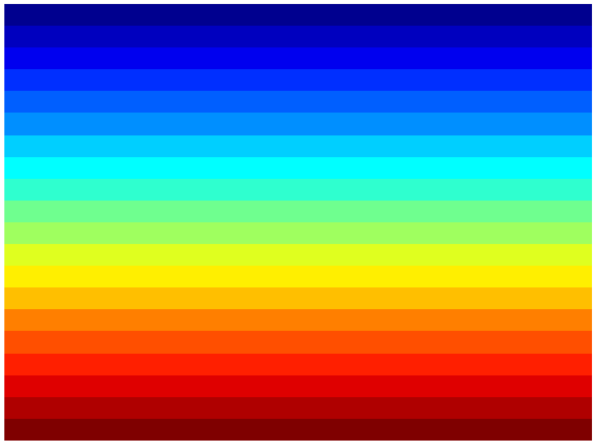
y_up =

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



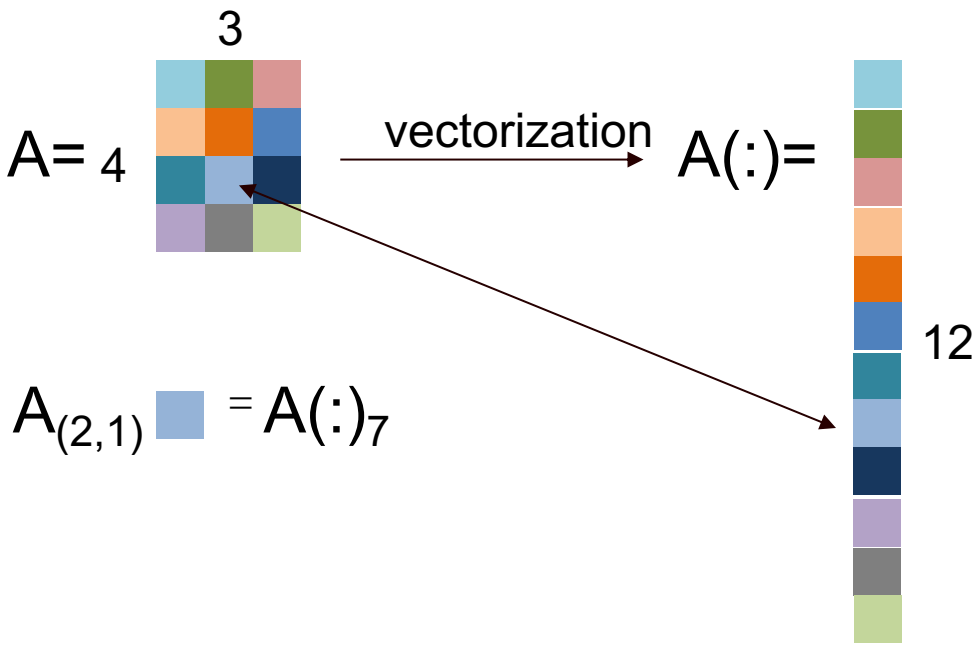
y_down =

2	2	2	2	2
3	3	3	3	3
3	3	3	3	3

Convolution without Looping using meshgrid

Python:

```
coor_up = np.stack([y_up.flatten(), x.flatten()])
ind_up = np.ravel_multi_index(coor_up, (nr, nc))

coor_down = np.stack([y_down.flatten(), x.flatten()])
ind_down = np.ravel_multi_index(coor_down, (nr, nc))
```



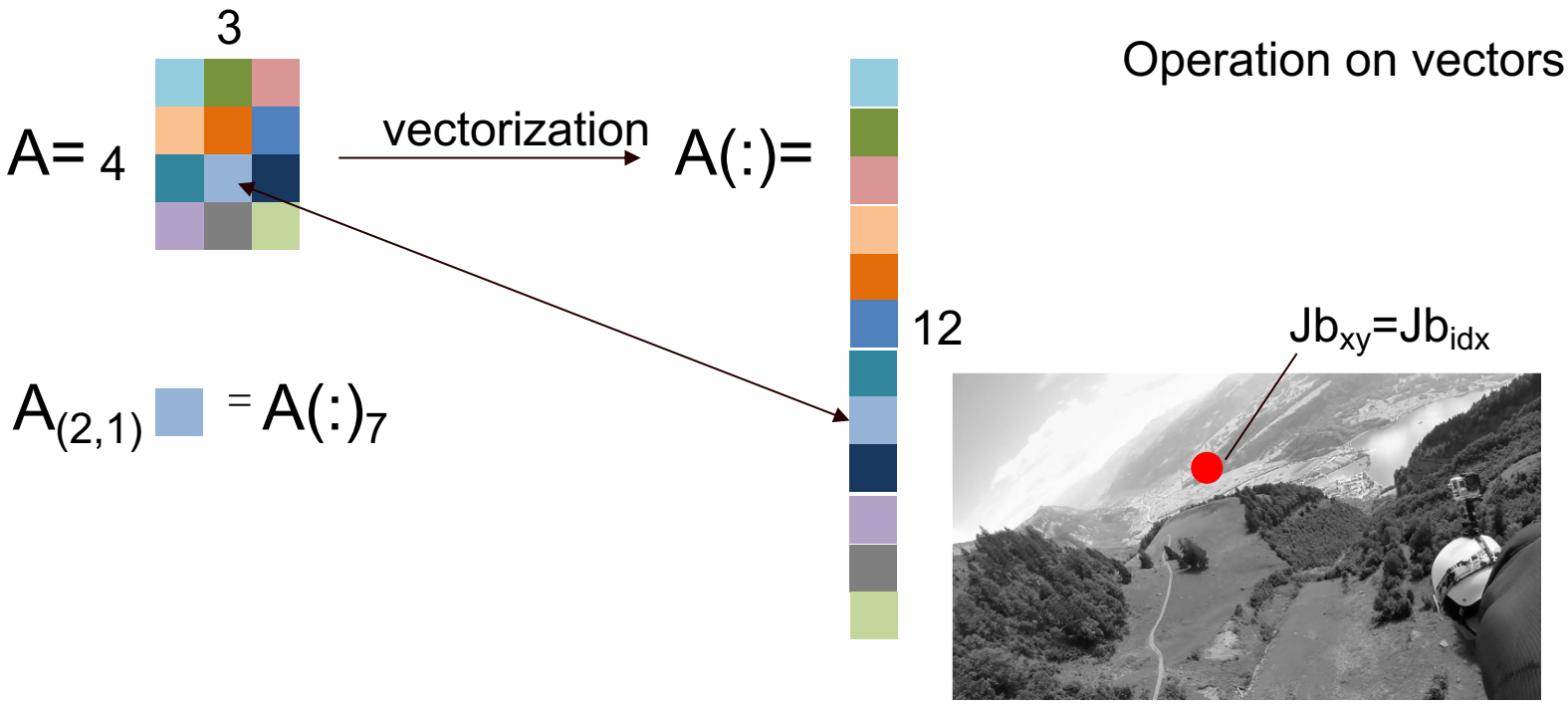
Convolution without Looping using meshgrid

Python:

```
coor_up = np.stack([y_up.flatten(), x.flatten()])
ind_up = np.ravel_multi_index(coor_up, (nr, nc))

coor_down = np.stack([y_down.flatten(), x.flatten()])
ind_down = np.ravel_multi_index(coor_down, (nr, nc))
```

```
J_out = 2 * Jb - 0.8 * Jb[ind_up] - 0.8 * Jb[ind_down]
```

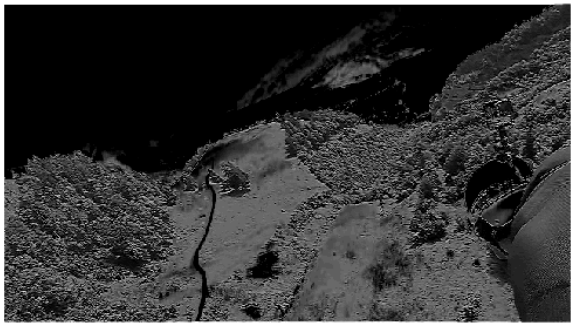
Operation on vectors



Convolution without Looping using meshgrid

Python:

```
Jb = Jb.flatten()
J_out = 2 * Jb - 0.8 * Jb[ind_up] - 0.8 * Jb[ind_down]
J_out = J_out.reshape(nr,nc)
plt.imshow(J_out, cmap='gray')
plt.show()
```



=

J_out

2

-0.8

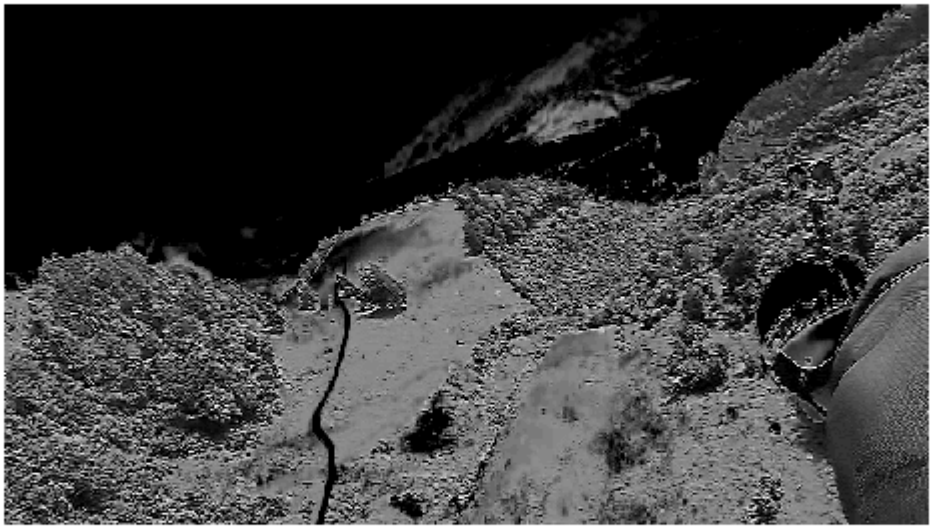
-0.8

Jb

Jb(ind_up)

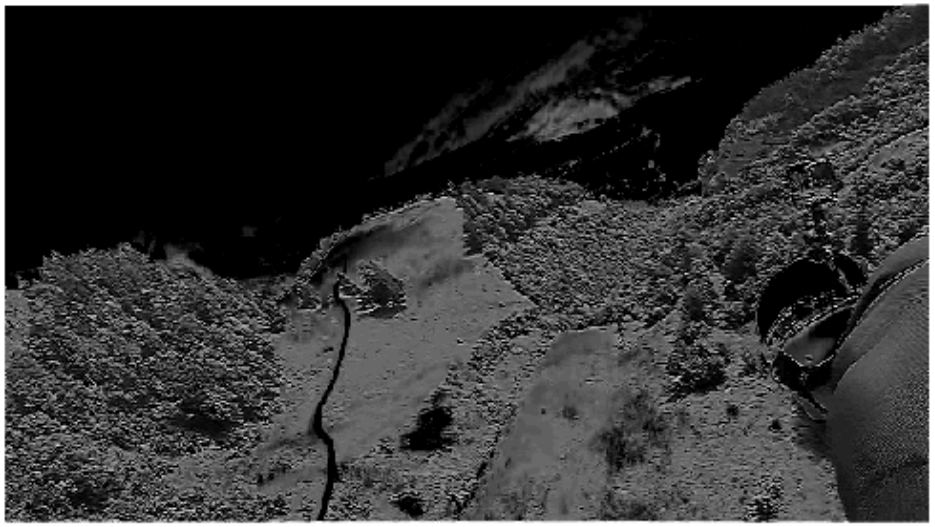
Jb(ind_down)

With loop



Computation time: 0.01787 sec

Without loop



Computation time: 1.45 sec

Vectorization is necessary for python code !!!