APT: Hough Transform & Bilateral Filter

Presenter: Dr. Ha Viet Uyen Synh.

PART A

HOUGH TRANSFORM

Motivation

In many cases an edge detector can be used as a pre-processing stage to obtain image points or image pixels that are on the desired curve in the image space. Due to imperfections in either the image data or the edge detector, however, there may be missing points or pixels on the desired curves as well as spatial deviations between the ideal line/circle/ellipse and the noisy edge points as they are obtained from the edge detector. For these reasons, it is often non-trivial to group the extracted edge features to an appropriate set of lines, circles or ellipses. The purpose of the Hough transform is to address this problem by making it possible to perform groupings of edge points into object candidates by performing an explicit voting procedure over a set of parameterized image objects

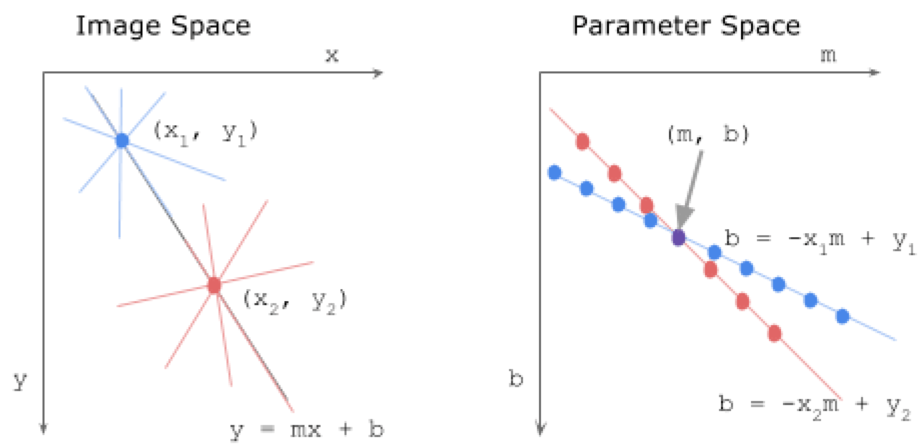
Cartesian Coordinates

In image space line is defined by the slope m and the y-intercept b

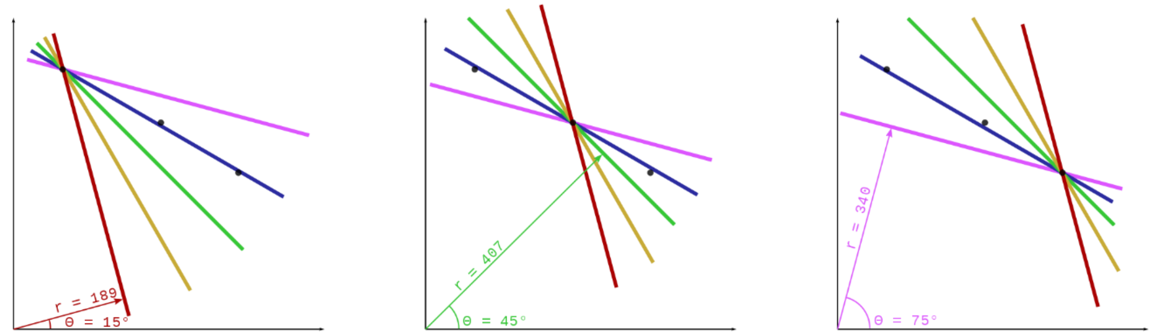
$$y = mx + b$$

So to detect the line in the image space we have to define these parameters, which is not applicable in image domain.

In the other domain with m and b coordinates, line represent a point from image domain. Points on the same line in image domain will be mapped to lines in Hough domain. These lines intersect with each other in a point with specific values m and b. These values are the slope and y-intercept of original line in image domain.



Example

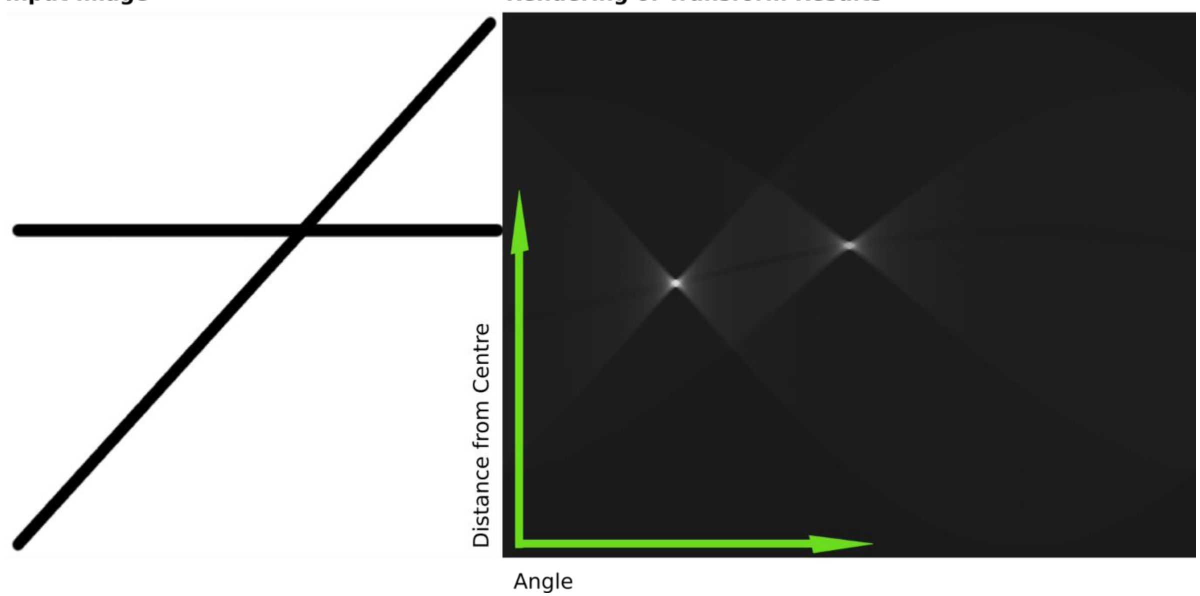


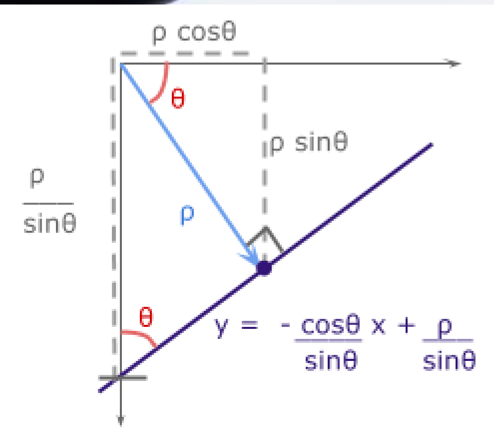
θ	r
15	189.0
30	282.0
45	355.7
60	407.3
75	429.4

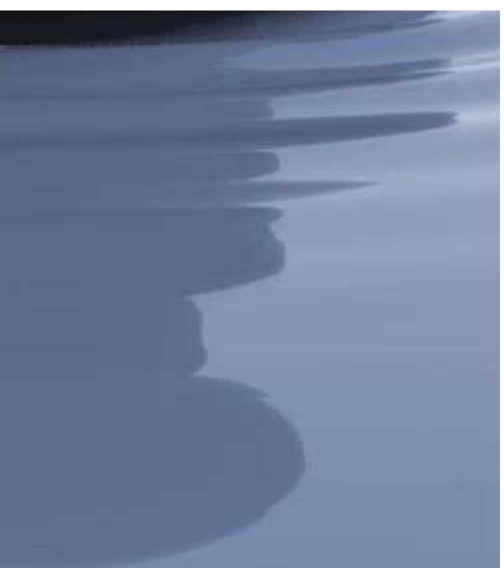
θ	r
15	318.5
30	376.8
45	407.3
60	409.8
75	385.3

θ	r
15	419.0
30	443.6
45	438.4
60	402.9
75	340.1

Input Image **Rendering of Transform Results**







Polar Coordinates

Due to undefined value of slope for vertical lines in cartesian coordinates, we have to move to polar coordinates. In polar coordinates line is define by ρ and θ where ρ is the norm distance of the line from origin. θ is the angle between the norm and the horizontal x axis. The equation of line in terms of ρ and θ now is

$$y = \frac{-\cos(\theta)}{\sin(\theta)}x + \frac{\rho}{\sin(\theta)}$$

and

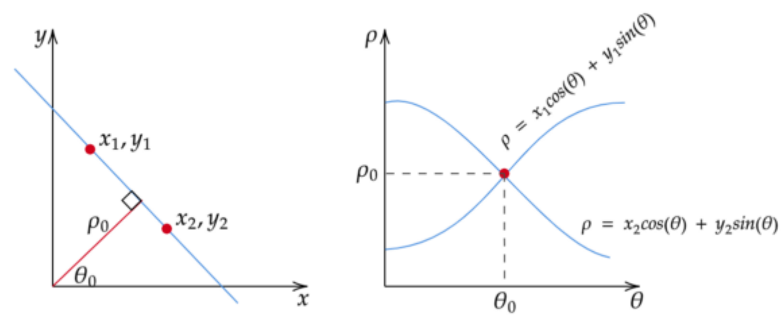
$$\rho = x\cos(\theta) = y\sin(\theta)$$

The range of values of ρ and θ

- θ in polar coordinate takes value in range of -90 to 90
- The maximum norm distance is given by diagonal distance which is

$$\rho_{max} = \sqrt{x^2 + y^2}$$

So ρ has values in range from $-\rho_{max}$ to ρ_{max}



Algorithm

Extract edges of the image using Canny

1- initialize parameter space rs, thetas

2- Create accumulator array and initialize to zero

3- for each edge pixel

4-   for each theta

5-   calculate r = x cos(theta) + y sin(theta)

6-   Increment accumulator at r, theta

7- Find Maximum values in accumulator (lines)

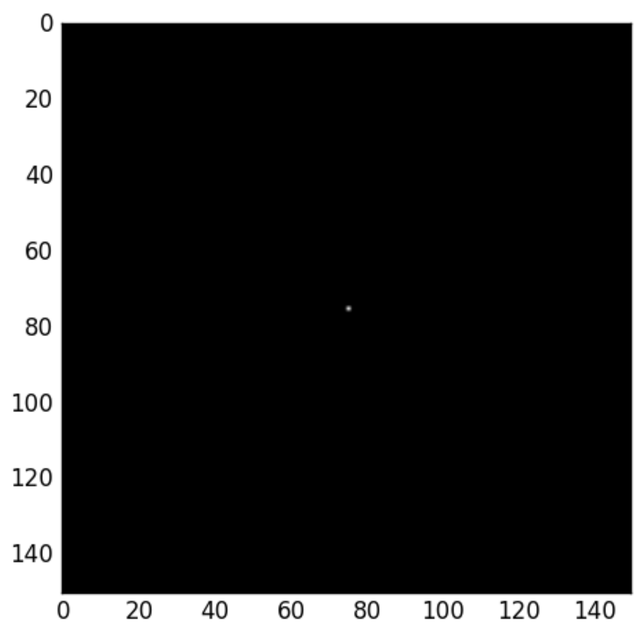
Extract related r, theta

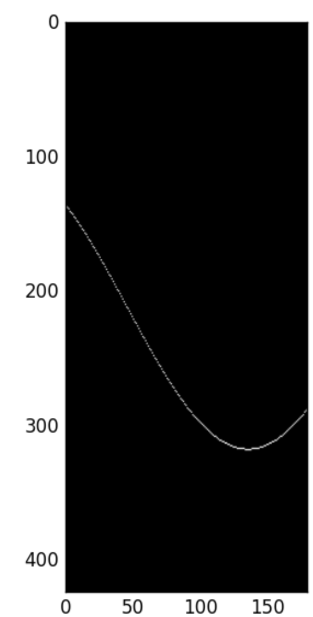
Implementation

```
import numpy as np
import matplotlib.pyplot as plt
def houghLine(image):
    #Get image dimensions _ y for rows and x for columns
    Ny = image.shape[0]
    Nx = image.shape[1]
    #Max diatance is diagonal one
    Maxdist = int(np.round(np.sqrt(Nx**2 + Ny ** 2)))
    #1. initialize parameter space rs, thetas
    # Theta in range from -90 to 90 degrees
    thetas = np.deg2rad(np.arange(-90, 90))
    #Range of radius
    rs = np.linspace(-Maxdist, Maxdist, 2*Maxdist)
    #2. Create accumulator array and initialize to zero
    accumulator = np.zeros((2 * Maxdist, len(thetas)))
    #3. Loop for each edge pixel
    for y in range(Ny):
     for x in range(Nx):
         # Check if it is an edge pixel _ NB: y -> rows , x -> columns
          if image[y,x] > 0:
            # Map edge pixel to hough space
            for k in range(len(thetas)):
              # Calculate space parameter
              r = x*np.cos(thetas[k]) + y * np.sin(thetas[k])
              # Update the accumulator _ N.B: r has value -max to max
              # map r to its idx 0 : 2*max
              accumulator[int(r) + Maxdist,k] += 1
 return accumulator, thetas, rs
```

Ex: One edge point image

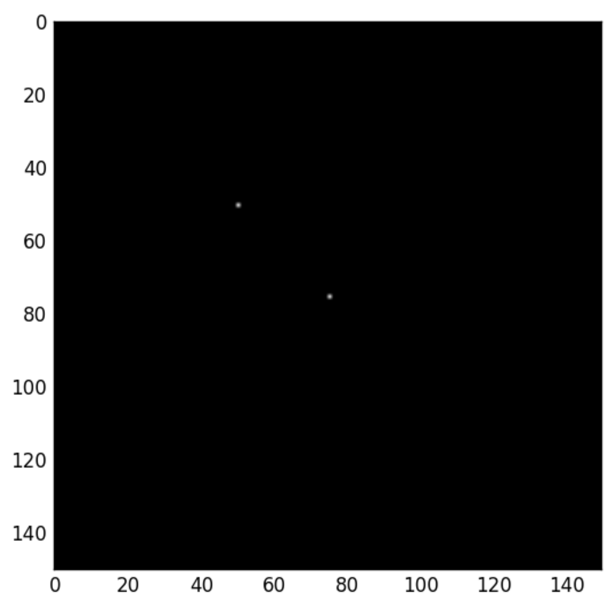
```
image = np.zeros((150,150))
image[75, 75] = 1
accumulator, thetas, rhos = houghLine(image)
plt.figure('Original Image')
plt.imshow(image)
plt.set_cmap('gray')
plt.figure('Hough Space')
plt.imshow(accumulator)
plt.set_cmap('gray')
plt.show()
```

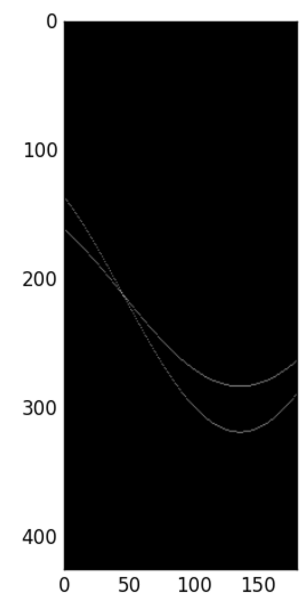




Ex: Two edge points image

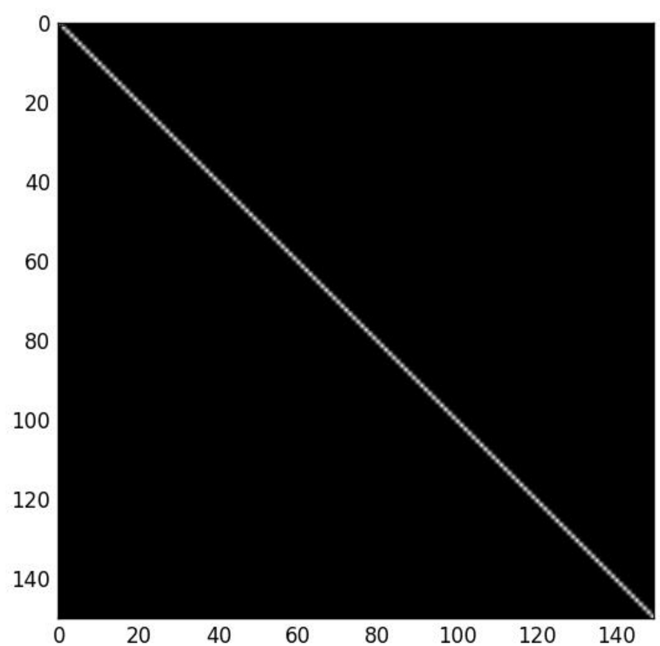
```
image = np.zeros((150,150))
image[75, 75] = 1
image[50, 50] = 1
accumulator, thetas, rhos = houghLine(image)
plt.figure('Original Image')
plt.imshow(image)
plt.set_cmap('gray')
plt.figure('Hough Space')
plt.imshow(accumulator)
plt.set_cmap('gray')
plt.show()
```

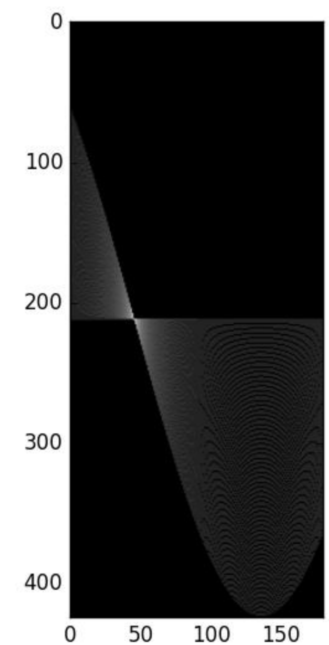




Ex: Image with a single line

```
image = np.zeros((150,150))
image[:, :] = np.eye(150)
accumulator, thetas, rhos = houghLine(image)
plt.figure('Original Image')
plt.imshow(image)
plt.set_cmap('gray')
plt.figure('Hough Space')
plt.imshow(accumulator)
plt.set_cmap('gray')
plt.show()
```





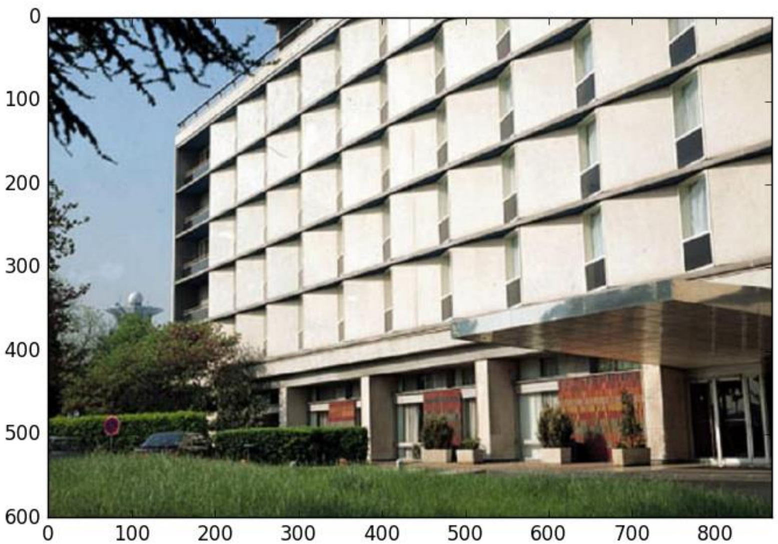
Getting value of ρ and θ

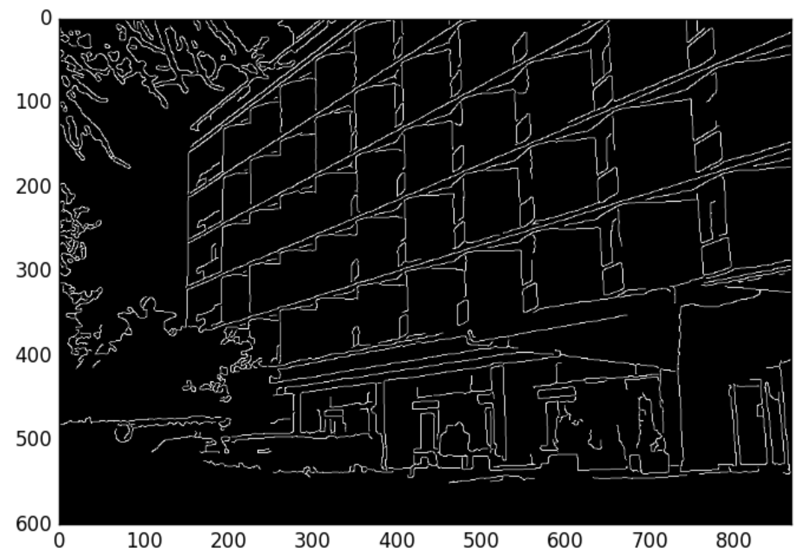
```
idx = np.argmax(accumulator)
rho = int(rhos[int(idx / accumulator.shape[1])])
theta = thetas[int(idx % accumulator.shape[1])]
print("rho={0:.0f}, theta={1:.0f}".format(rho, np.rad2deg(theta)))
```

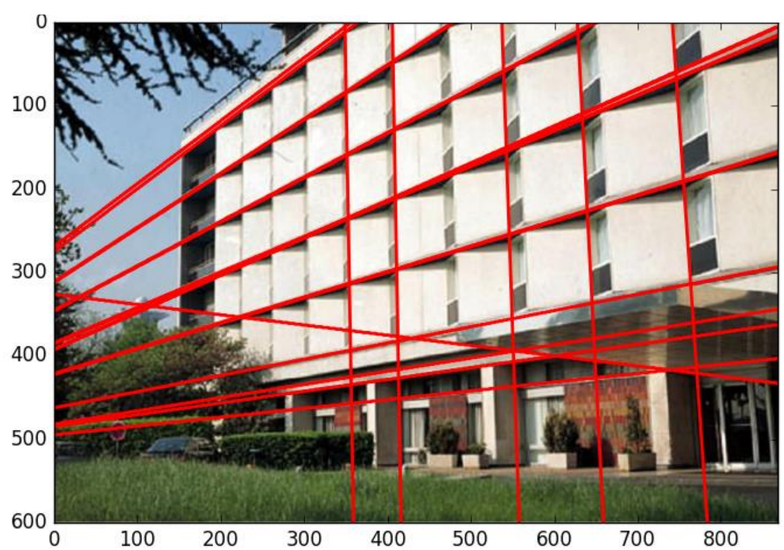
Output is

rho = 0, theta = -45

Ex: Real images







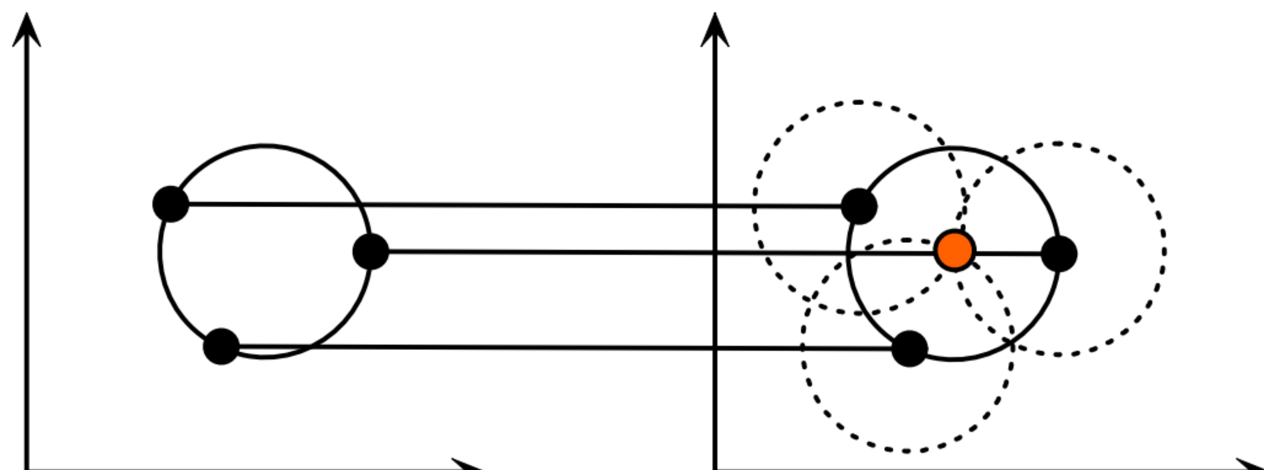
Hough Circle Transform

A circle is represented mathematically as

$$(x-x_{center})^2+(y-y_{center})^2=r^2$$

where (x_{center},y_{center}) is the center of the circle, and r is the radius of the circle.

From equation, we can see we have 3 parameters, so we need a 3D accumulator for Hough transform.



Each point in geometric space (left) generates a circle in parameter space (right). The circles in parameter space intersect at the (a, b) that is the center in geometric space.

Implementation

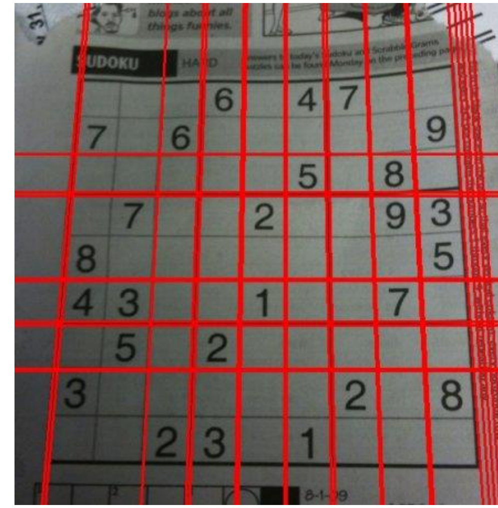
```
# Read image
img = cv2.imread('lanes.jpg', cv2.IMREAD_COLOR) # road.png is the filename

# Convert the image to gray-scale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Find the edges in the image using canny detector
edges = cv2.Canny(gray, 50, 200)

# Detect points that form a line
lines = cv2.HoughLinesP(edges, 1, np.pi/180, max_slider, minLineLength=10,
maxLineGap=250)

# Draw lines on the image
for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0), 3)

# Show result
cv2.imshow("Result Image", img)
```

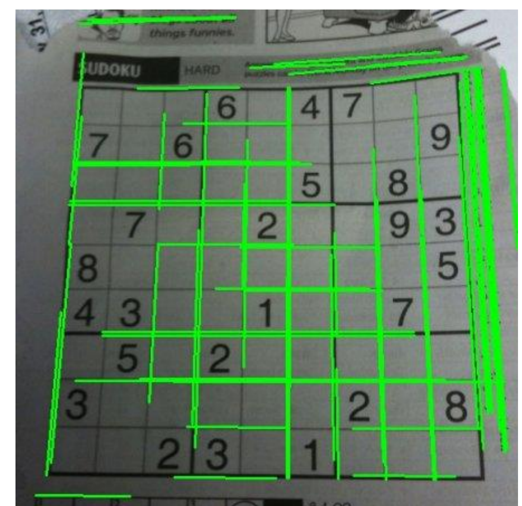
OpenCV vs Hough Transform

cv2.HoughLines

cv2.HoughLinesP

cv2.HoughCircles

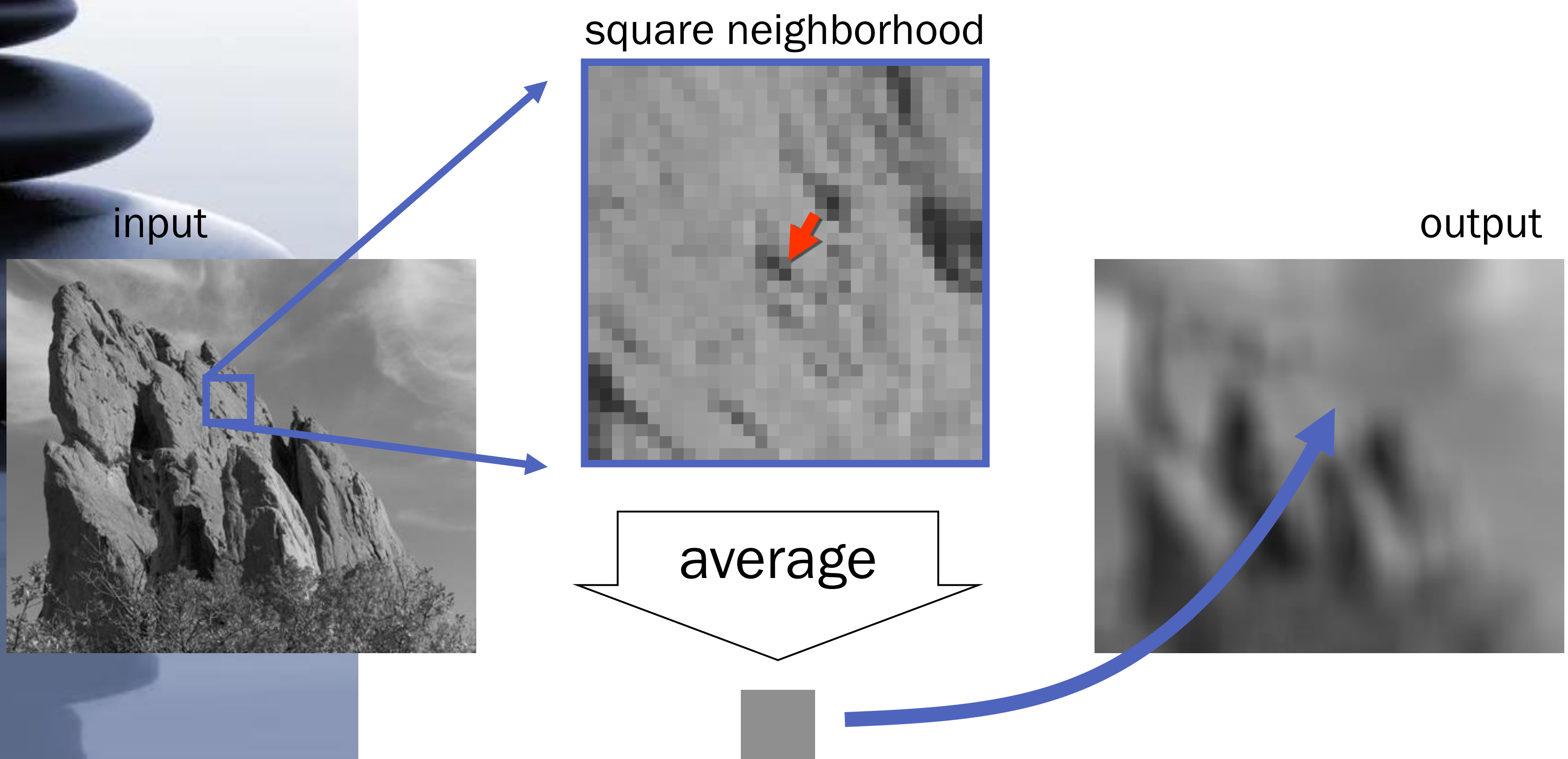




PART B

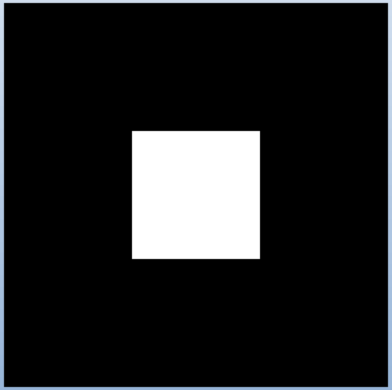
BILATERAL FILTER

Box Average



Equation of Box Average

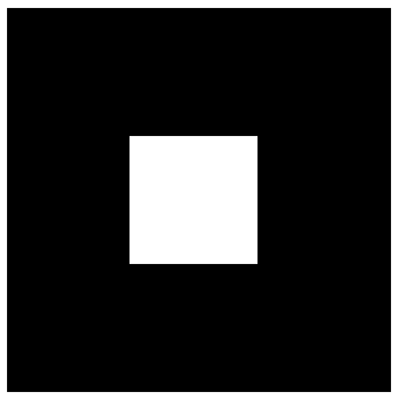
$$\underbrace{BA[I]_{\mathbf{p}}}_{\text{result at pixel } \mathbf{p}} = \underbrace{\sum_{\mathbf{q}\in S}}_{\text{sum over all pixels } \mathbf{q}} \underbrace{B_{\sigma}(\mathbf{p}-\mathbf{q})}_{\text{normalized box function}} \underbrace{I_{\mathbf{q}}}_{\text{intensity at pixel } \mathbf{q}}$$



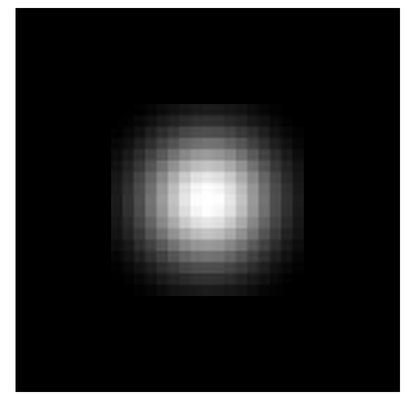
Strategy to Solve these Problems

Use an isotropic (*i.e.* circular) window.

Use a window with a smooth falloff.

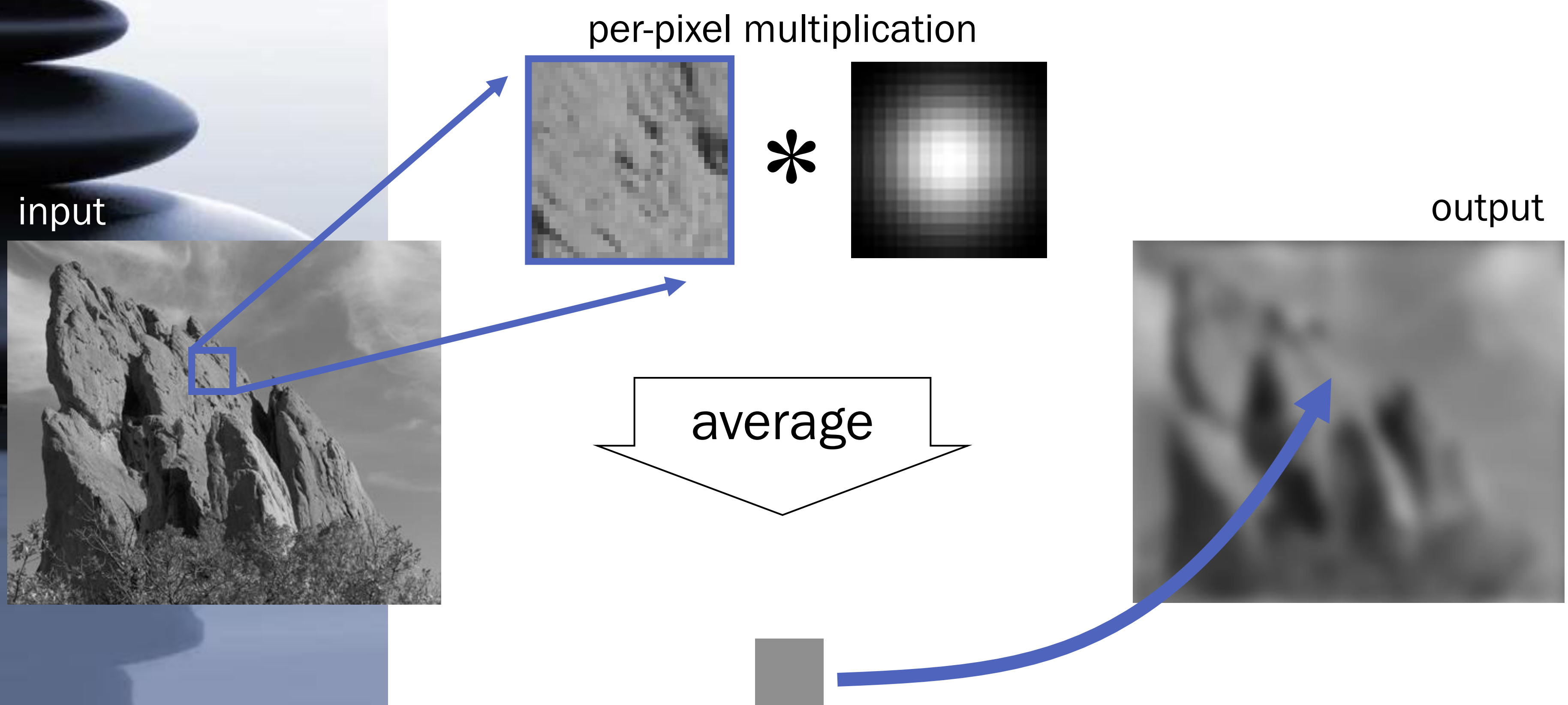


box window



Gaussian window

Gaussian Blur



Equation of Gaussian Blur

Same idea: **weighted average of pixels.**

$$GB[I]_{\mathbf{p}} = \sum_{\mathbf{q} \in S} \boxed{G_\sigma(\| \mathbf{p} - \mathbf{q} \|)} I_{\mathbf{q}}$$

normalized
Gaussian function

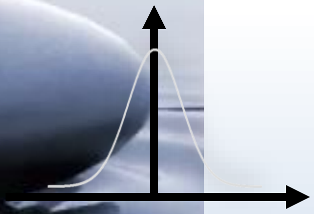
$$G_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2} \right)$$

Spatial Parameter

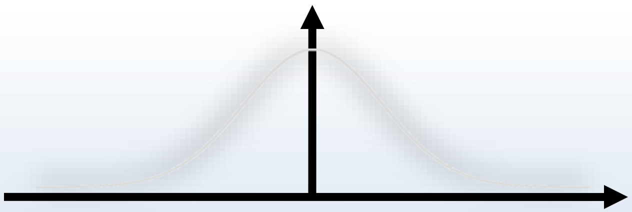
input

$$GB[I]_{\mathbf{p}} = \sum_{\mathbf{q}\in S} G_{\sigma}\left(\|\mathbf{p}-\mathbf{q}\|\right) I_{\mathbf{q}}$$

size of the window

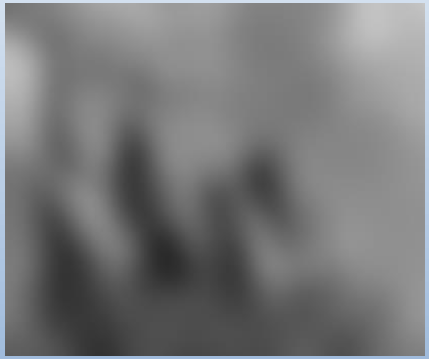


small σ



large σ

limited smoothing



strong smoothing

Properties of Gaussian Blur

Does smooth images

But smoothes too much:
edges are blurred.

Only spatial distance matters

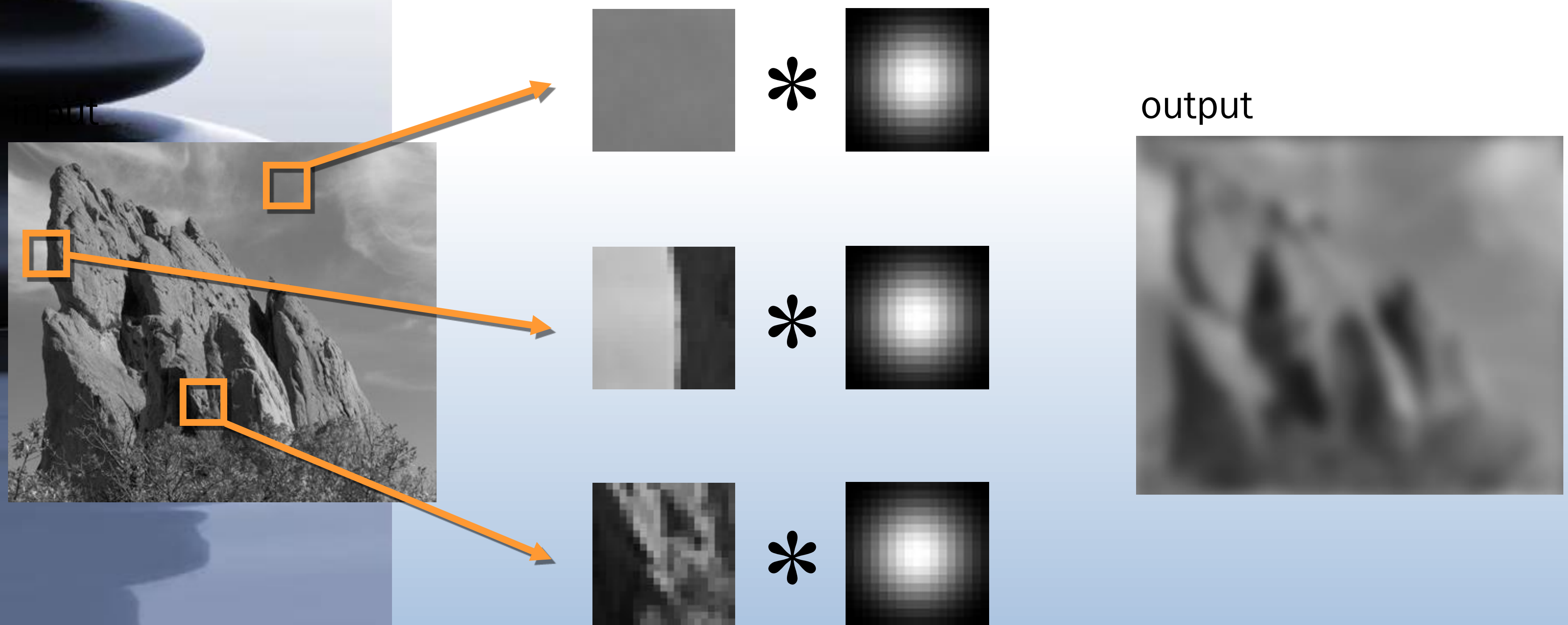
No edge term

input

output

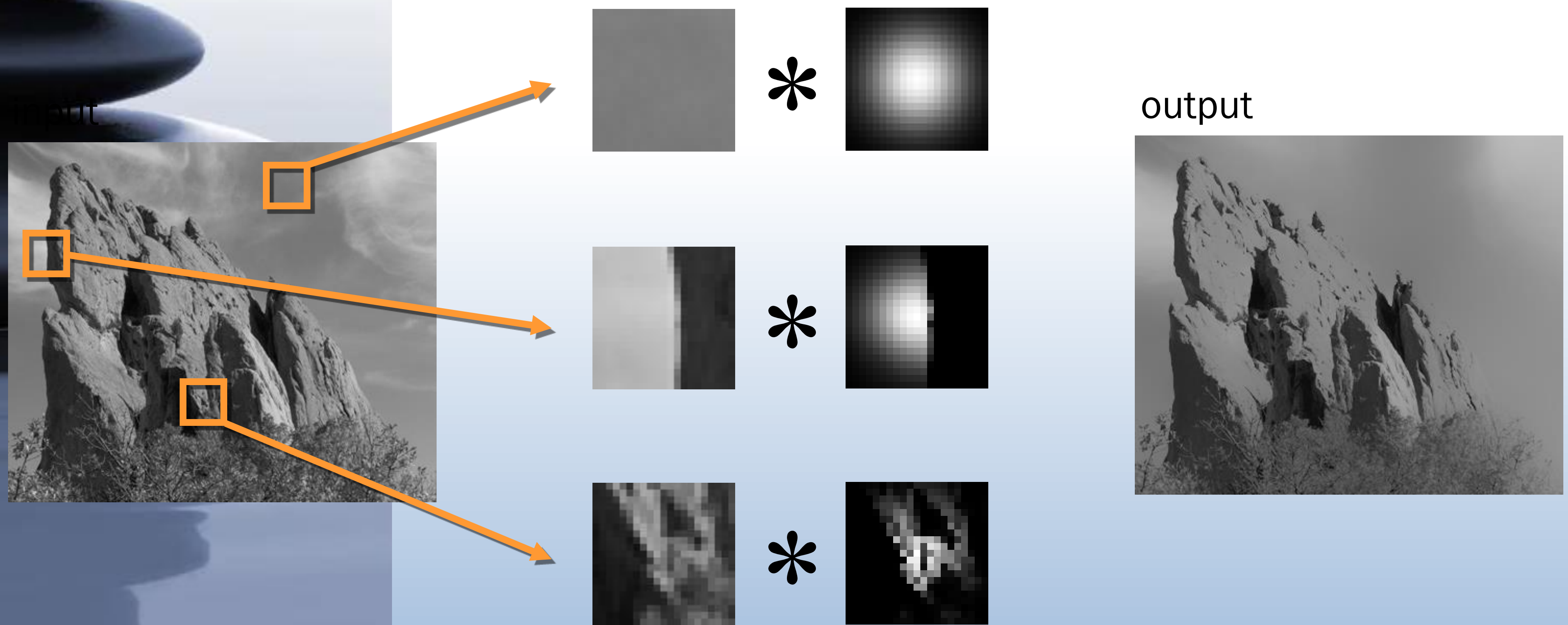
$$GB[I]_{\mathbf{p}} = \sum_{\mathbf{q} \in S} \underbrace{G_\sigma(\|\mathbf{p}-\mathbf{q}\|)}_{\text{space}} I_{\mathbf{q}}$$

Blur Comes from Averaging across Edges



Same Gaussian kernel everywhere.

Bilateral Filter
No Averaging across Edges



The kernel shape depends on the image content.

Bilateral Filter Definition: an Additional Edge Term

Same idea: **weighted average of pixels.**

$$BF[I]_{\mathbf{p}} = \underbrace{\overset{\text{new}}{\frac{1}{W_{\mathbf{p}}}}}_{\text{normalization factor}} \sum_{\mathbf{q}\in S} \underbrace{\overset{\text{not new}}{G_{\sigma_{\mathrm{s}}}\left(\|\mathbf{p}-\mathbf{q}\|\right)}}_{\textit{space}\text{ weight}} \, \underbrace{\overset{\text{new}}{G_{\sigma_{\mathrm{r}}}\left(|I_{\mathbf{p}}-I_{\mathbf{q}}|\right)}}_{\textit{range}\text{ weight}} \, I_{\mathbf{q}}$$

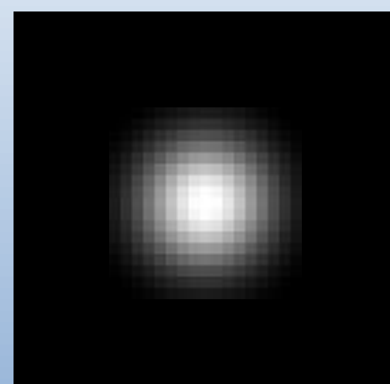
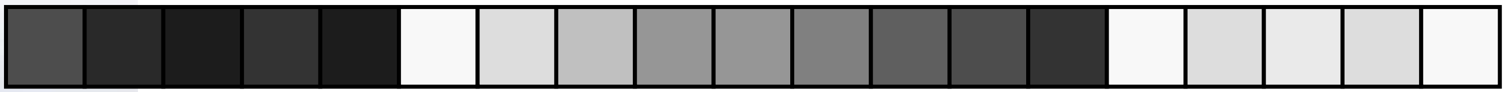
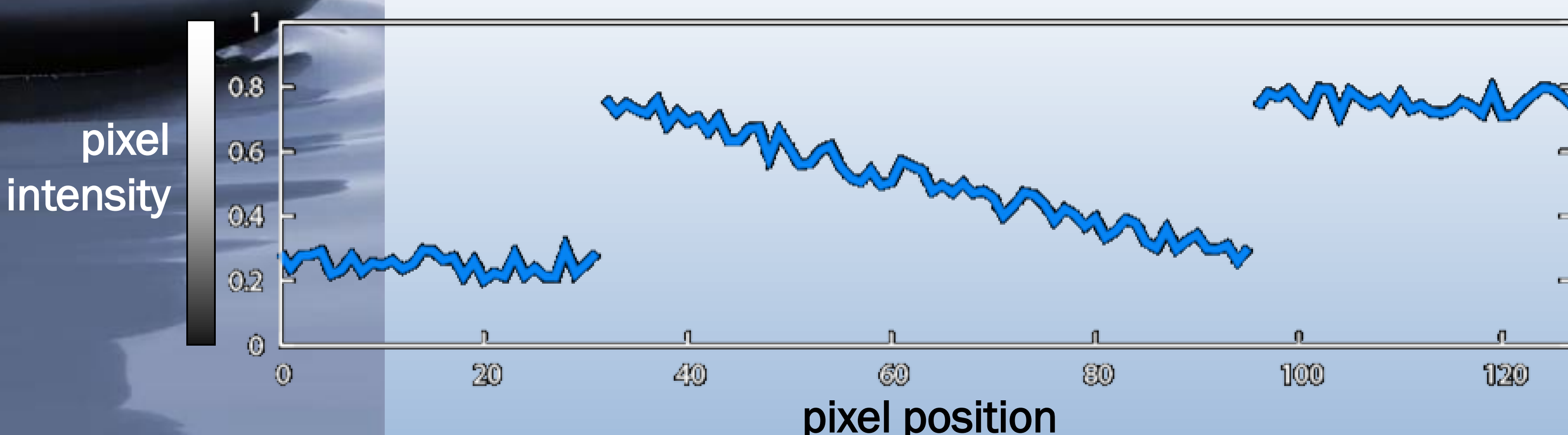


Illustration a 1D Image

1D image = line of pixels

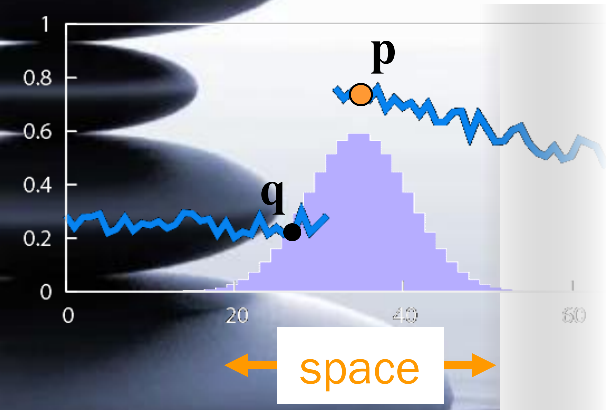


Better visualized as a plot

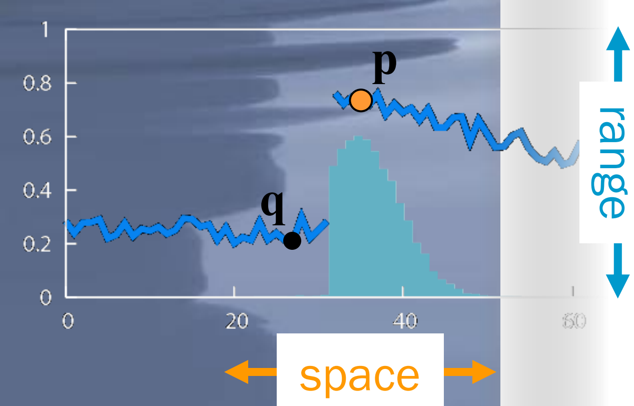


Gaussian Blur and Bilateral Filter

Gaussian blur



Bilateral filter

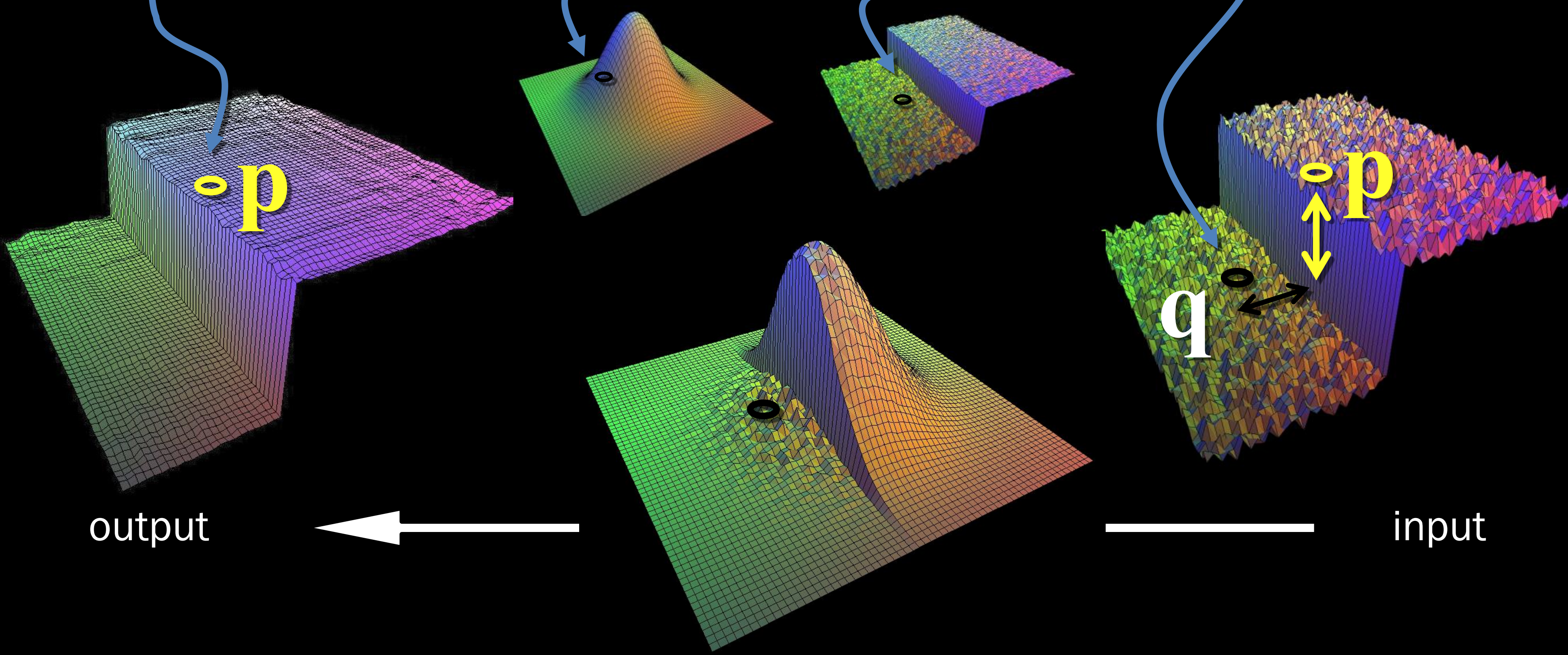


$$GB[I]_{\mathbf{p}} = \sum_{\mathbf{q}\in S} \underbrace{G_{\sigma}(\|\mathbf{p}-\mathbf{q}\|)}_{\text{space}} I_{\mathbf{q}}$$

$$BF[I]_{\mathbf{p}} = \underbrace{\frac{1}{W_{\mathbf{p}}}}_{\text{normalization}} \sum_{\mathbf{q}\in S} \underbrace{G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)}_{\text{space}} \underbrace{G_{\sigma_r}(|I_{\mathbf{p}}-I_{\mathbf{q}}|)}_{\text{range}} I_{\mathbf{q}}$$

Bilateral Filter on a Height Field

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)\; G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\; I_{\mathbf{q}}$$



Space and Range Parameters

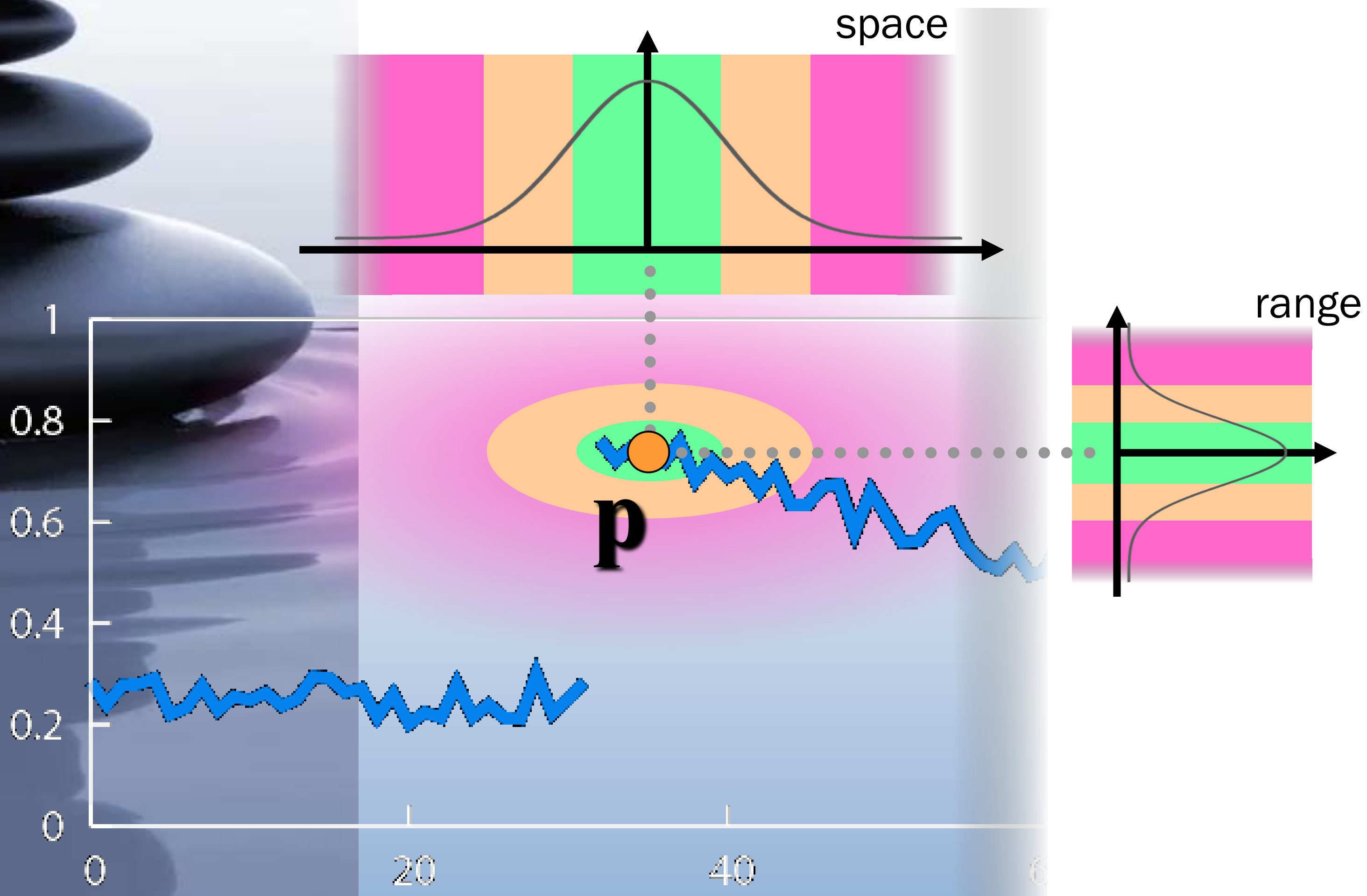
$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_s}(\| \mathbf{p} - \mathbf{q} \|) \, G_{\sigma_r}(| I_{\mathbf{p}} - I_{\mathbf{q}} |) \, I_{\mathbf{q}}$$

space σ_s : spatial extent of the kernel, size of the considered neighborhood.

range σ_r : “minimum” amplitude of an edge

Influence of Pixels

Only pixels close in space and in range are considered.



Exploring the Parameter Space

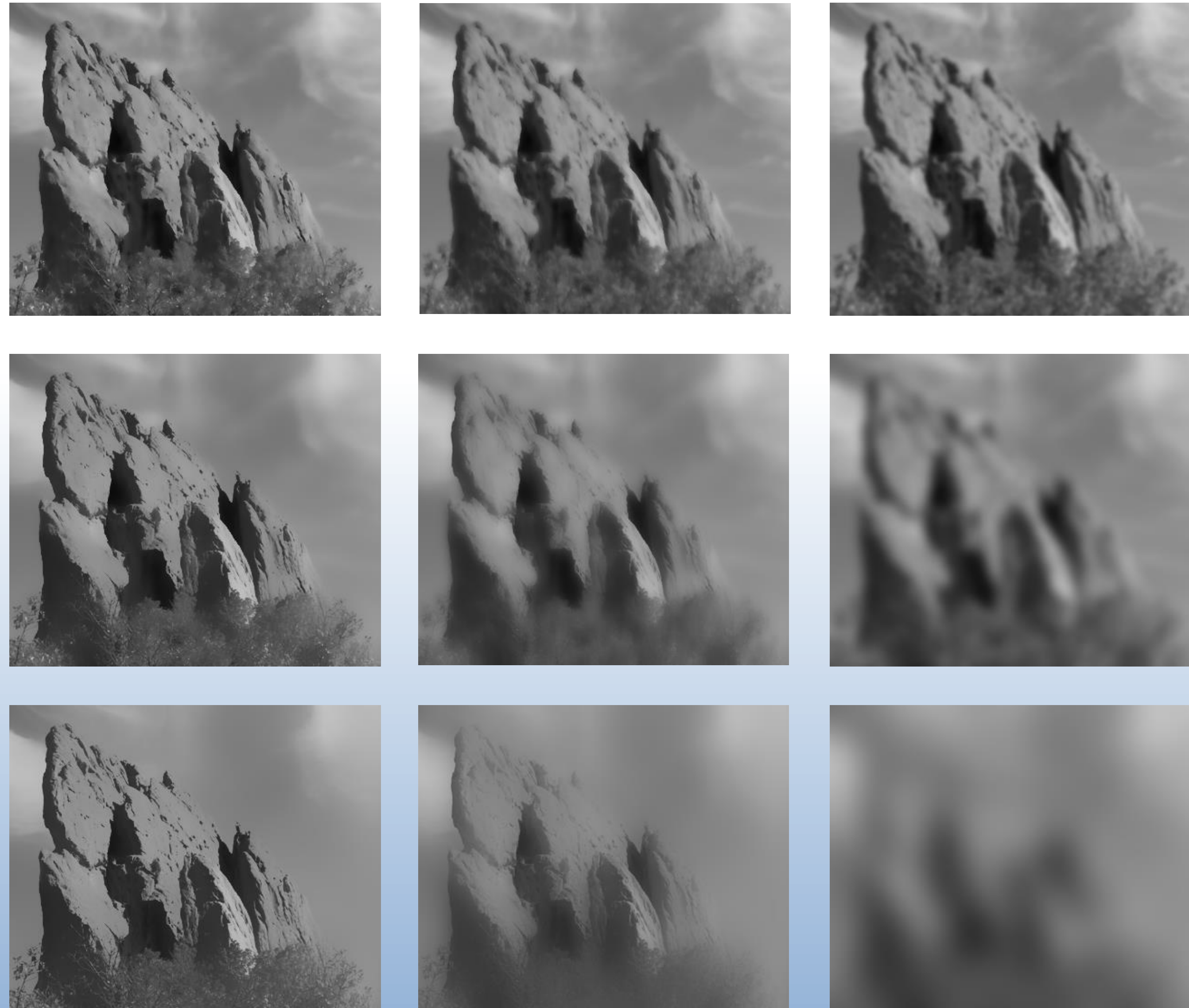
input

$\sigma_r = 0.1$ | $\sigma_r = 0.25$ | $\sigma_r = \infty$ (Gaussian blur)

$\sigma_s = 2$

$\sigma_s = 6$

$\sigma_s = 18$



Varying the Range Parameter

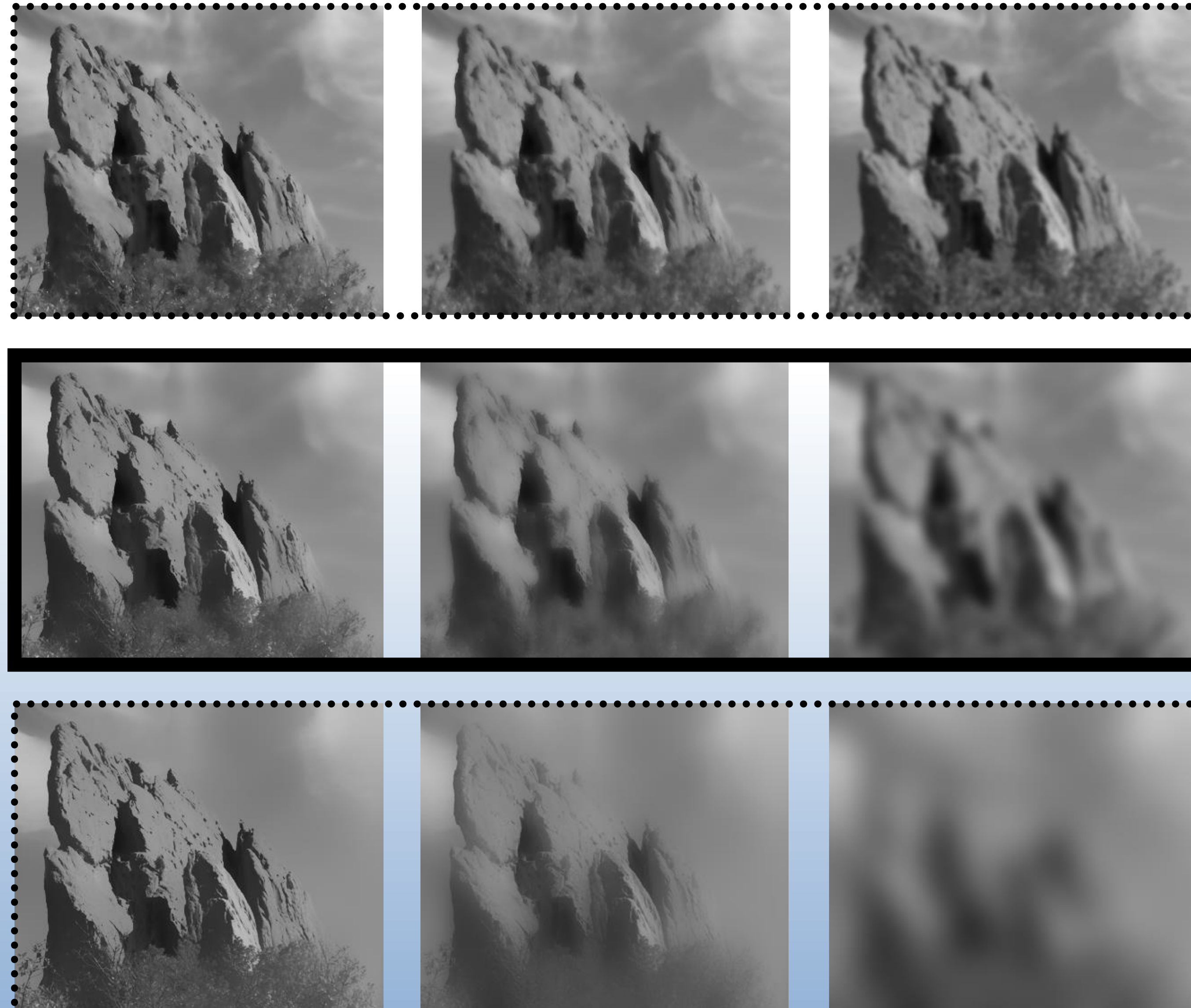
input

$\sigma_r = 0.1$ | $\sigma_r = 0.25$ | $\sigma_r = \infty$ (Gaussian blur)

$\sigma_s = 2$

$\sigma_s = 6$

$\sigma_s = 18$



Varying the Space Parameter

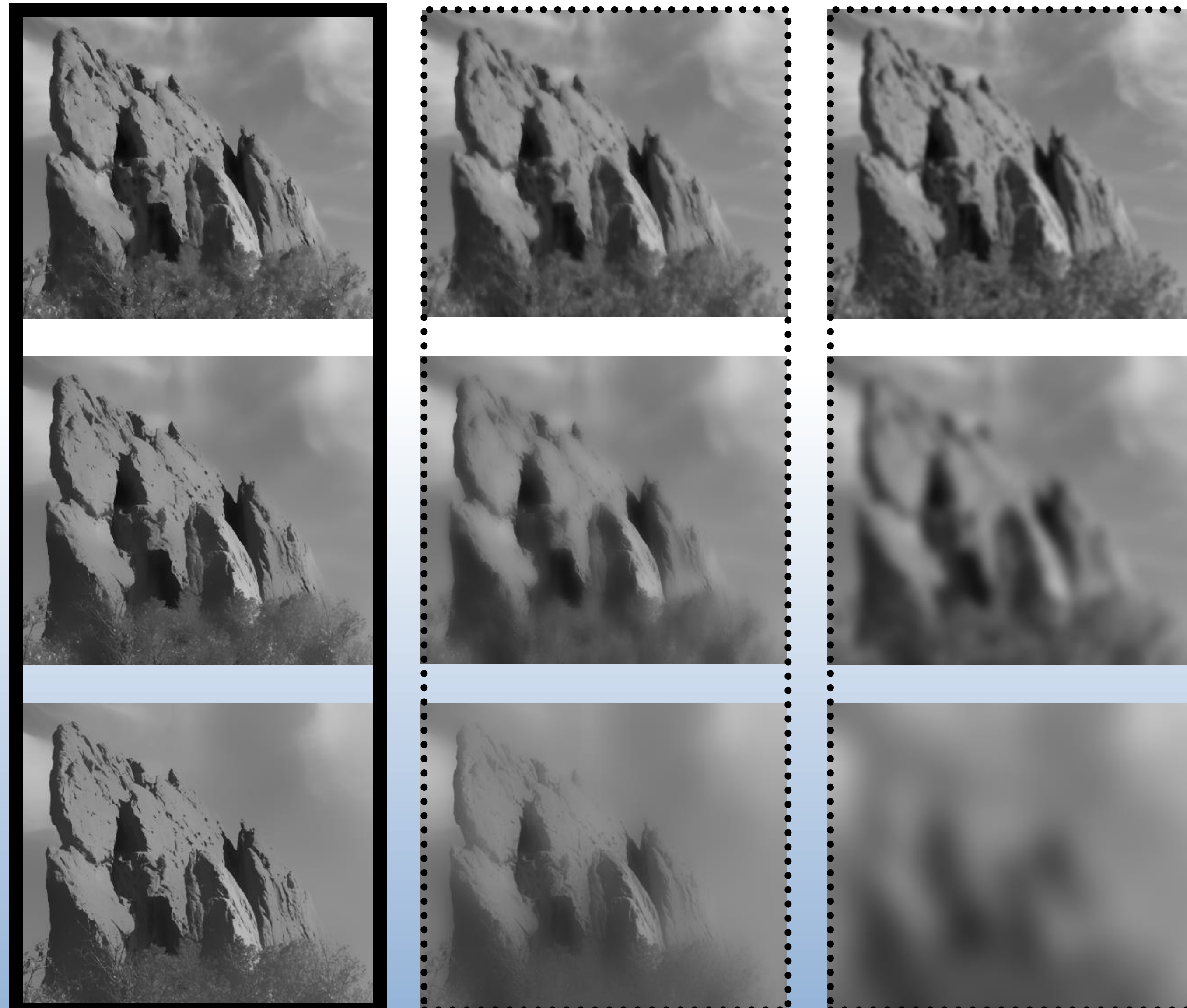
input

$\sigma_r = 0.1$ | $\sigma_r = 0.25$ | $\sigma_r = \infty$ (Gaussian blur)

$\sigma_s = 2$

$\sigma_s = 6$

$\sigma_s = 18$



How to Set the Parameters

Depends on the application. For instance:

space parameter: proportional to image size

e.g., 2% of image diagonal

range parameter: proportional to edge amplitude

e.g., mean or median of image gradients

independent of resolution and exposure

PART B

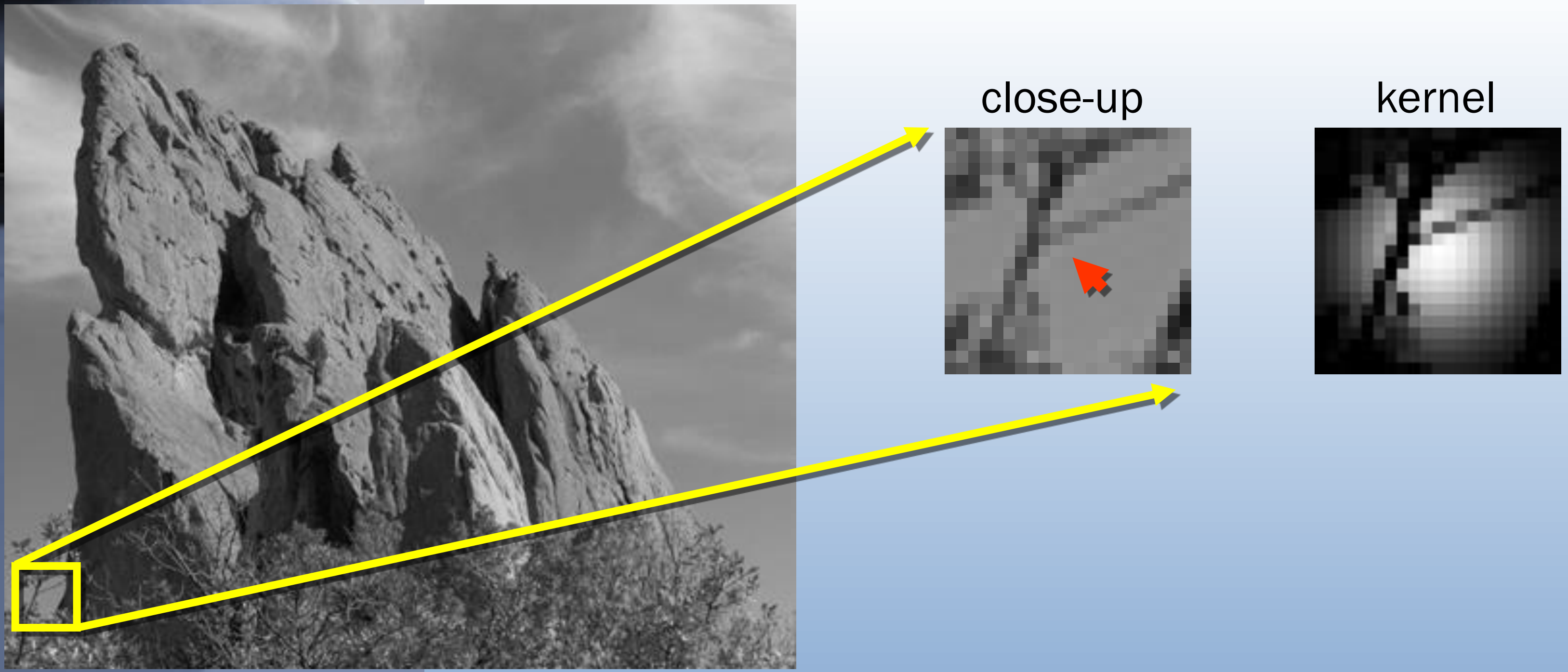
A FEW MORE ADVANCED REMARKS

Bilateral Filter Crosses Thin Lines

Bilateral filter averages across features thinner than ~$2\sigma_s$

Desirable for smoothing: more pixels = more robust

Different from diffusion that stops at thin lines



Iterating the Bilateral Filter

$$I_{(n+1)} = BF\left[I_{(n)}\right]$$

Generate more piecewise-flat images

Often not needed in computational photo.

Bilateral Filtering Color Images

For gray-level images

intensity difference

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_r}\left(\boxed{|I_{\mathbf{p}} - I_{\mathbf{q}}|}\right) \boxed{I_{\mathbf{q}}}$$

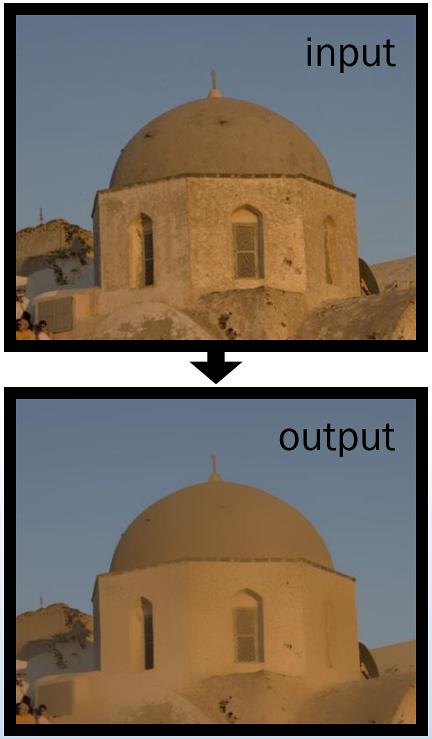
scalar

For color images

color difference

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_r}\left(\boxed{\|\mathbf{C}_{\mathbf{p}} - \mathbf{C}_{\mathbf{q}}\|}\right) \boxed{\mathbf{C}_{\mathbf{q}}}$$

3D vector
(RGB, Lab)



The bilateral filter is extremely easy to adapt to your need.

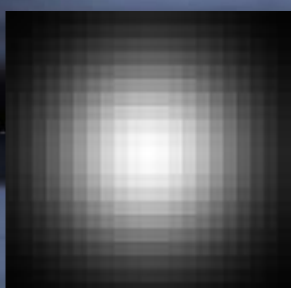
Hard to Compute

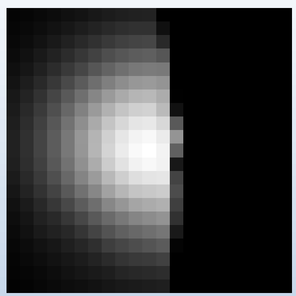
Nonlinear

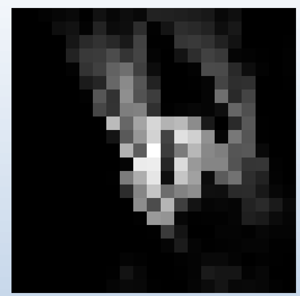
$$BF[I]_{\mathbf{p}} = \boxed{\frac{1}{W_{\mathbf{p}}}} \sum_{\mathbf{q}\in S} G_{\sigma_{\mathrm{s}}}(\|\mathbf{p}-\mathbf{q}\|)\, \boxed{G_{\sigma_{\mathrm{r}}}(|I_{\mathbf{p}}-I_{\mathbf{q}}|)}\, I_{\mathbf{q}}$$

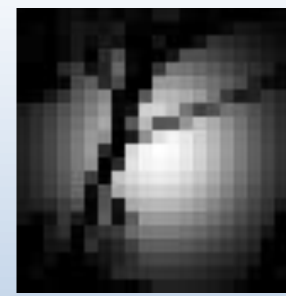
Complex, spatially varying kernels

Cannot be pre-computed, no FFT…









Brute-force implementation is slow > 10min

Questions? More Information?

hvusynh@hcmiu.edu.vn