

Image Filtering

Week – 2B

Disclaimer: Many of the slides used here are obtained from online resources (including many open lecture materials given at MIT, etc.) without due acknowledgement. They are used here for the sole purpose of classroom teaching. All the credits and all the copyrights belong to the original authors. You should not copy it, redistribute it, put it online, or use it for any, any purposes other than helping you study the course of ENGN/COMP 4528/6528.

Announcement

- Mathematics Reference Book: Linear algebra, probability
 - <https://mml-book.github.io/book/mml-book.pdf>
 - eg. CDF (cumulative distribution function, see chapter 6.2 discrete and continuous distribution function)
- Lecture Series by Prof. Shree Nayar, Professor of Computer Science in the School of Engineering at Columbia University, Senior researcher in computer vision with high profile.
 - <https://fpcv.cs.columbia.edu/>

Outline

- Image Noise
- Linear filter
 - Gaussian filter
 - Edge detection
- Non-linear filter
 - median filter
 - bilateral filter

Outline

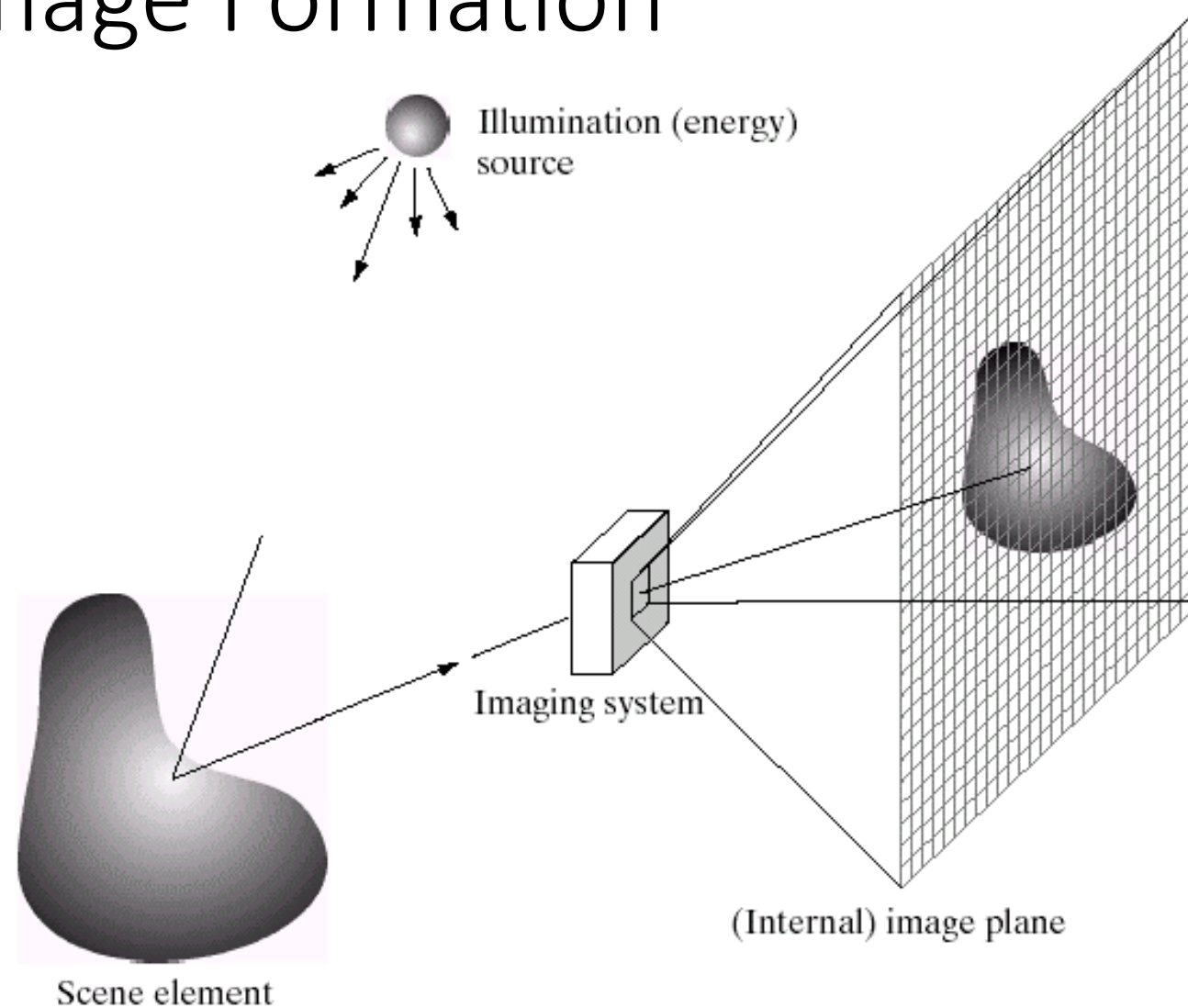
- Image Noise
- Linear filter
 - Gaussian filter
 - Edge detection
- Non-linear filter
 - median filter
 - bilateral filter

Image Filtering

- Filtering (correlation, convolution)
- Gaussian filter
- Application of filters (denoising, edge, contour, corner, texture, template matching and tracking ...,)

Image Noise

Recall: Image Formation



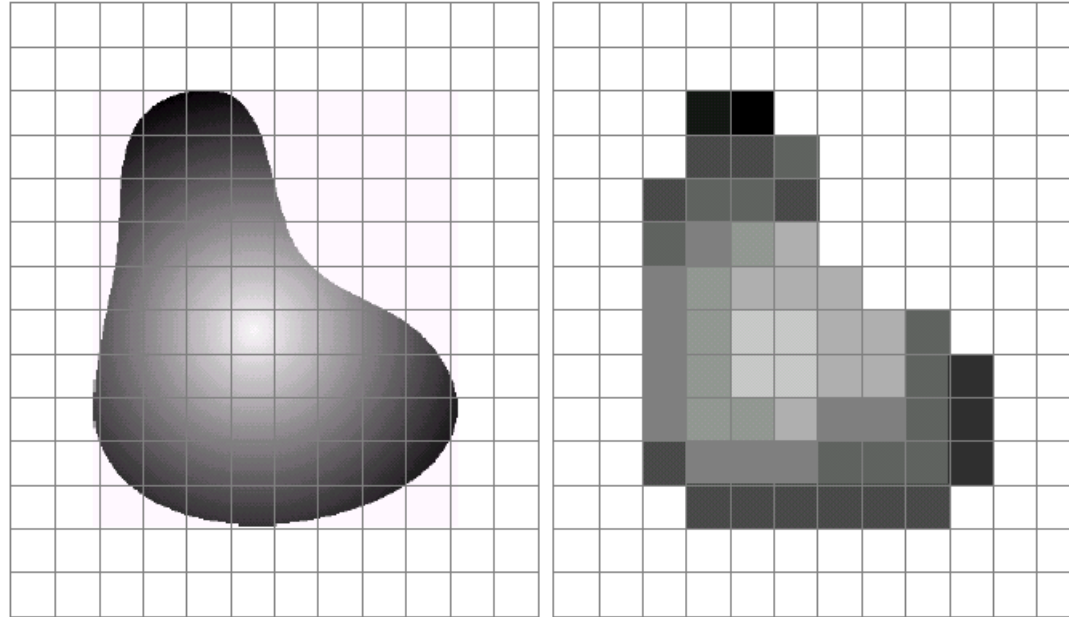
Recall: Digital camera



A digital camera replaces film with a sensor array

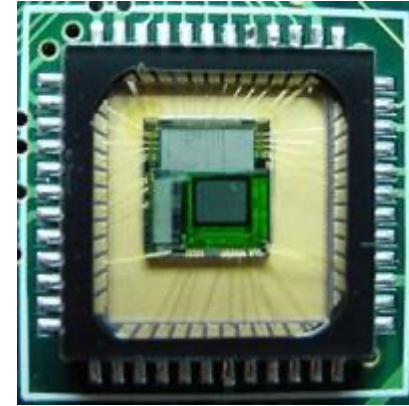
- Each cell in the array is light-sensitive diode that converts photons to electrons
- <http://electronics.howstuffworks.com/digital-camera.htm>

Digital images



a b

FIGURE 2.17 (a) Continuous image projected onto a sensor array. (b) Result of image sampling and quantization.



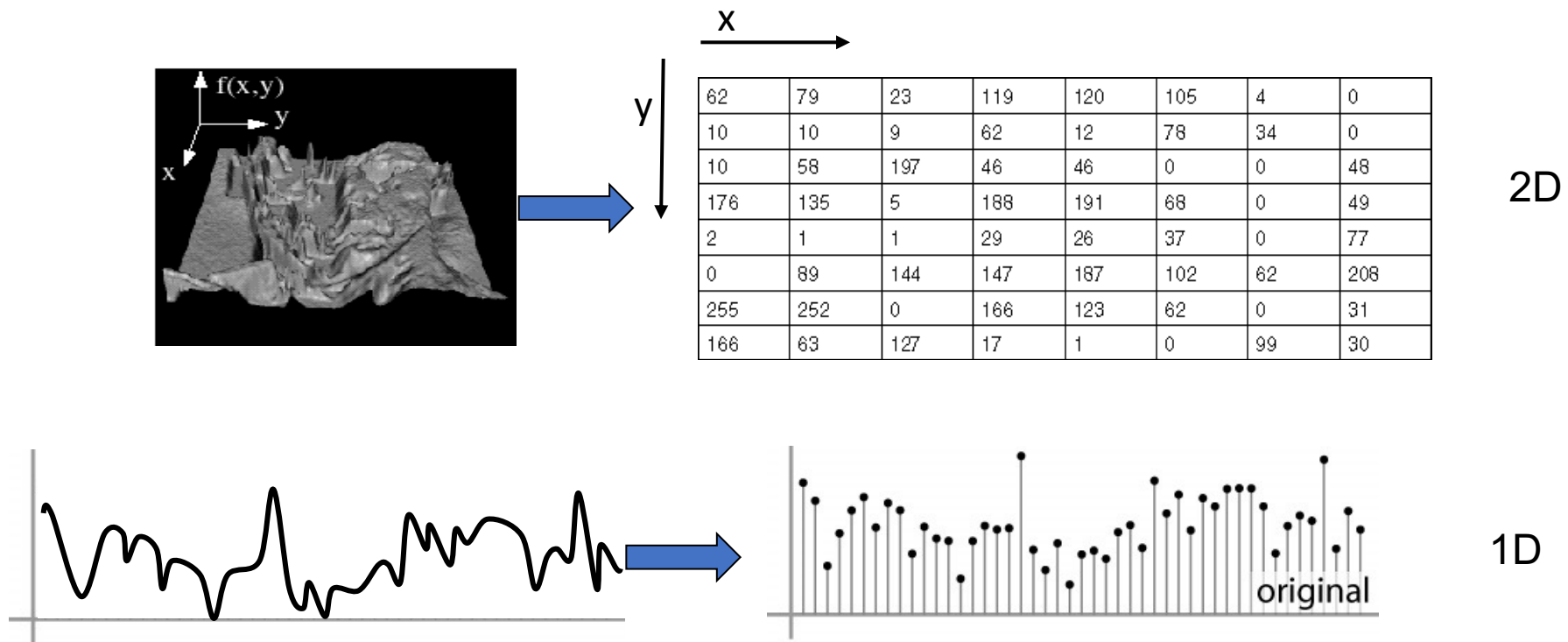
Recall: Images as functions

- We can think of an image as a function, f , from \mathbb{R}^2 to \mathbb{R} : **Input: coordinate output: intensity value**
 - $f(x, y)$ gives the intensity at position (x, y)
 - Realistically, we expect the image only to be defined over a **rectangle**, with a **finite range**:
 - $f: [a, b] \times [c, d] \rightarrow [0, 255]$
limit **intensity**
- A **color image** is just **three functions** pasted together. We can write this as a “vector-valued” function:

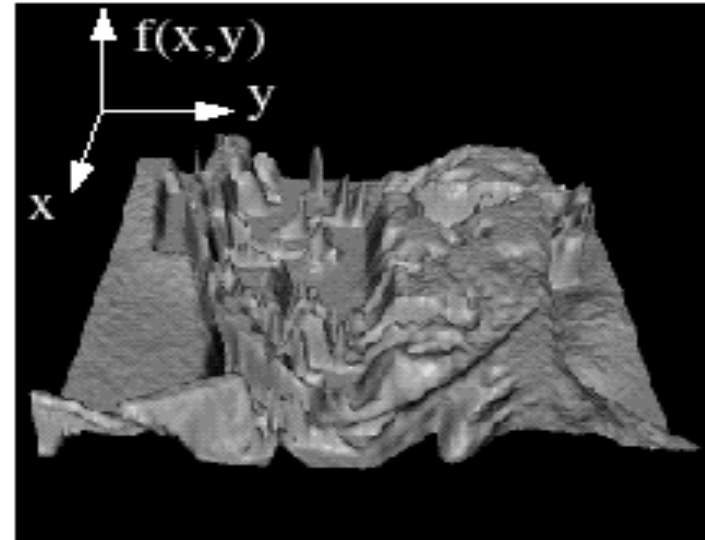
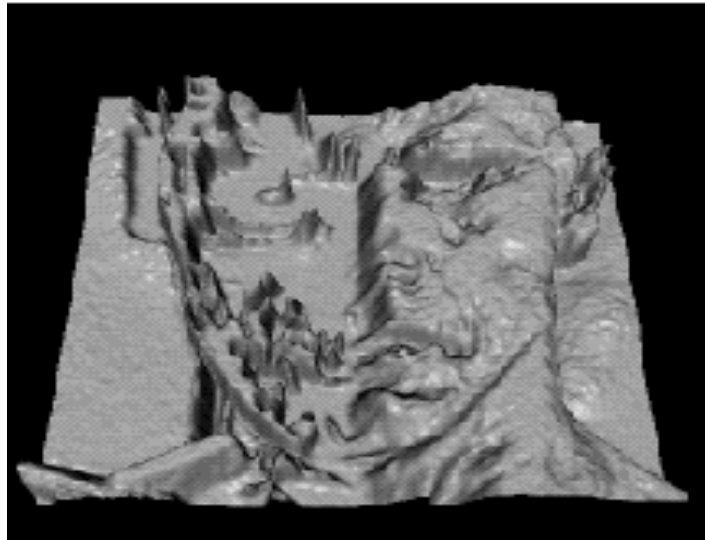
$$f(x, y) = \begin{bmatrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{bmatrix}$$

Recall: Digital images

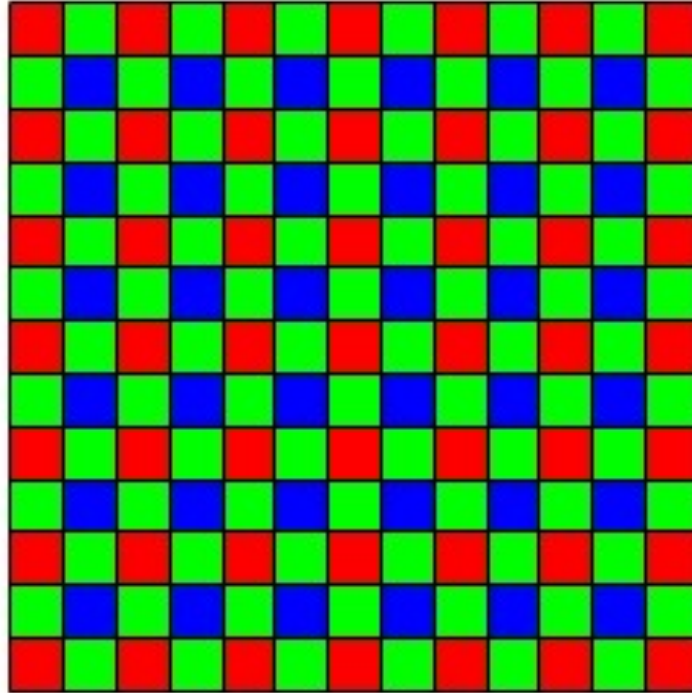
- **Sample** the 2D space on a regular grid
- **Quantize** each sample (round to nearest integer)
- Image is thus represented as a matrix of integer values.



Recall: Images as functions



Digital color images



Bayer filter

Recall: Digital color images

Color images,
RGB color
space



R



G



B

Common types of noise

- **Salt and pepper noise:** random occurrences of black and white pixels
- **Impulse noise:** random occurrences of white pixels, sometimes, it is also referred to as Salt and pepper noise.
- **Gaussian noise:** variations in intensity drawn from a Gaussian normal distribution



Original



Salt and pepper noise

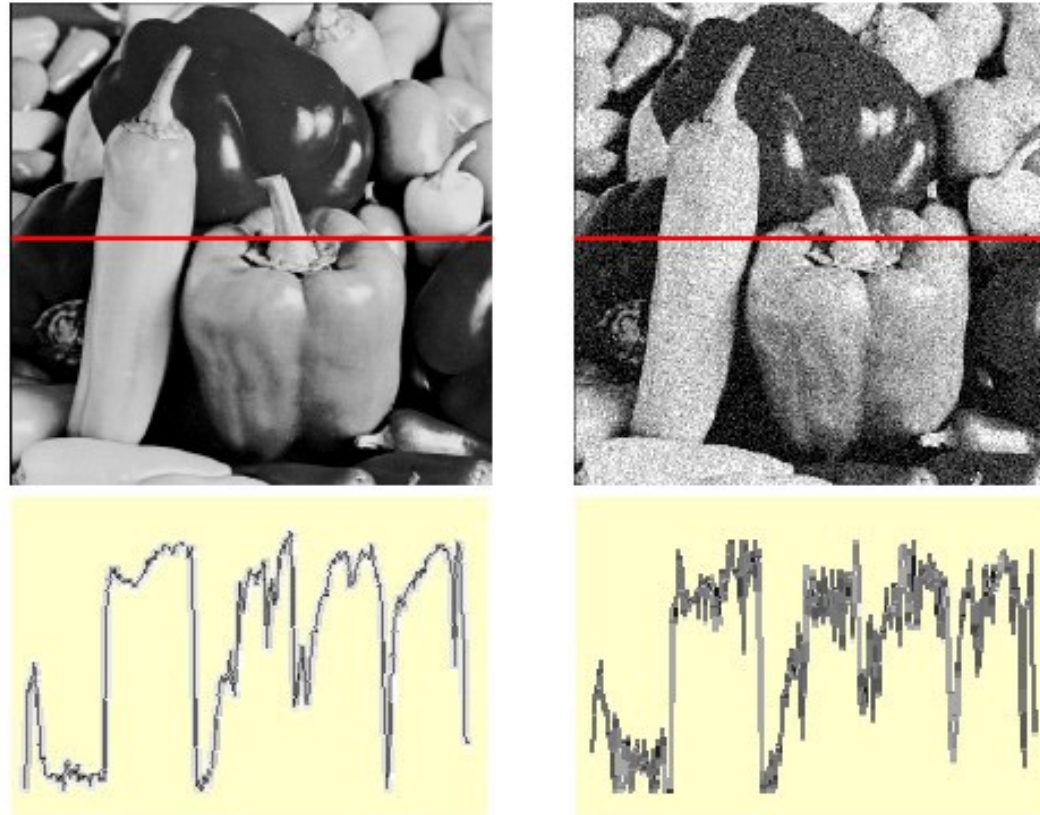


Impulse noise



Gaussian noise

Gaussian noise



$$f(x, y) = \underbrace{\bar{f}(x, y)}_{\text{Ideal Image}} + \underbrace{\eta(x, y)}_{\text{Noise process}}$$

Gaussian i.i.d. ("white") noise:
 $\eta(x, y) \sim \mathcal{N}(\mu, \sigma)$

```
>> noise = np.random.normal(mean, std_dev, shape)
>> output = im + noise;
```

What is the impact of the sigma?

$\sigma=1$

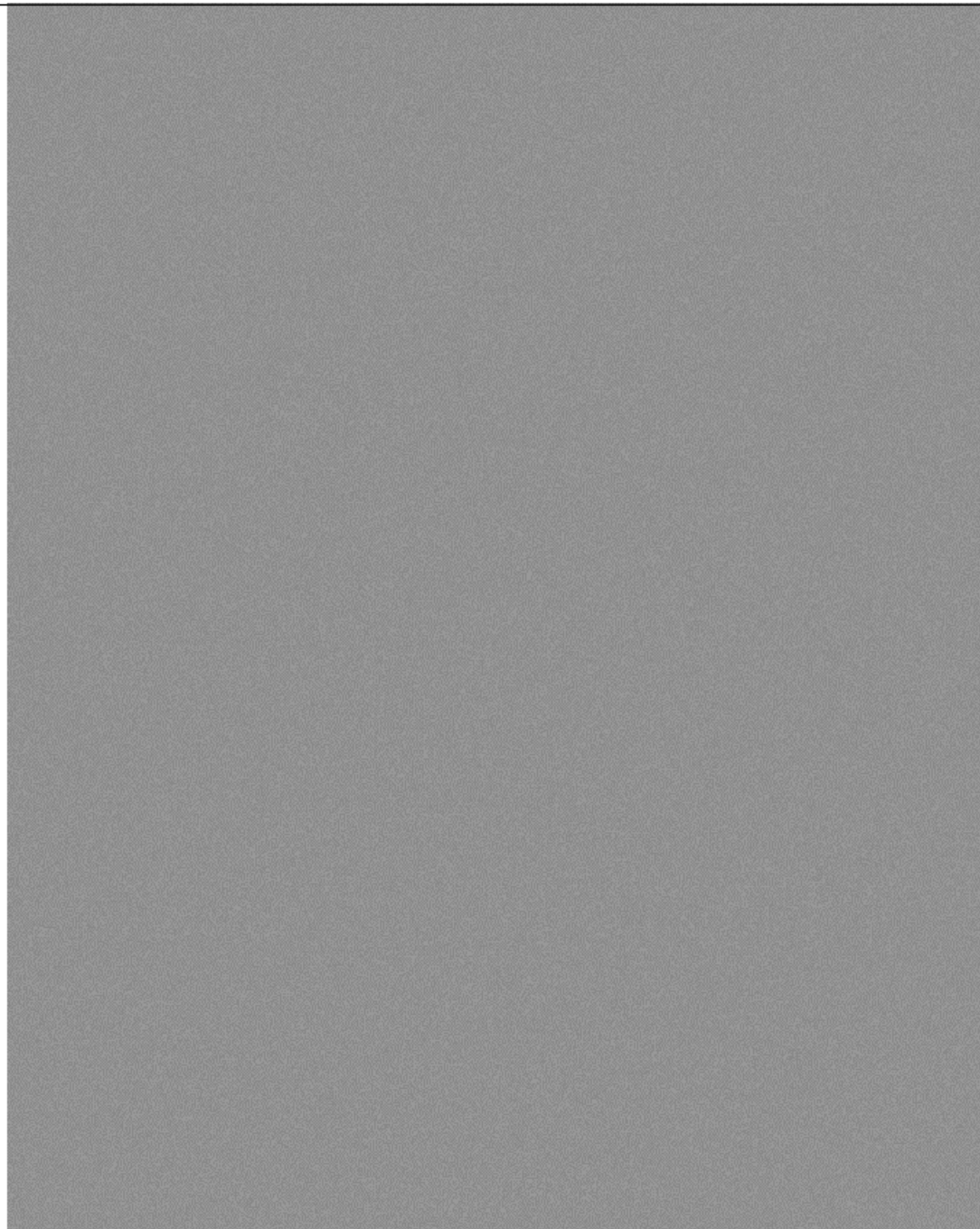
Effect of
sigma on
Gaussian
noise:

Image
shows the
noise values
themselves.

$\sigma=4$

Effect of
sigma on
Gaussian
noise:

Image
shows the
noise values
themselves.



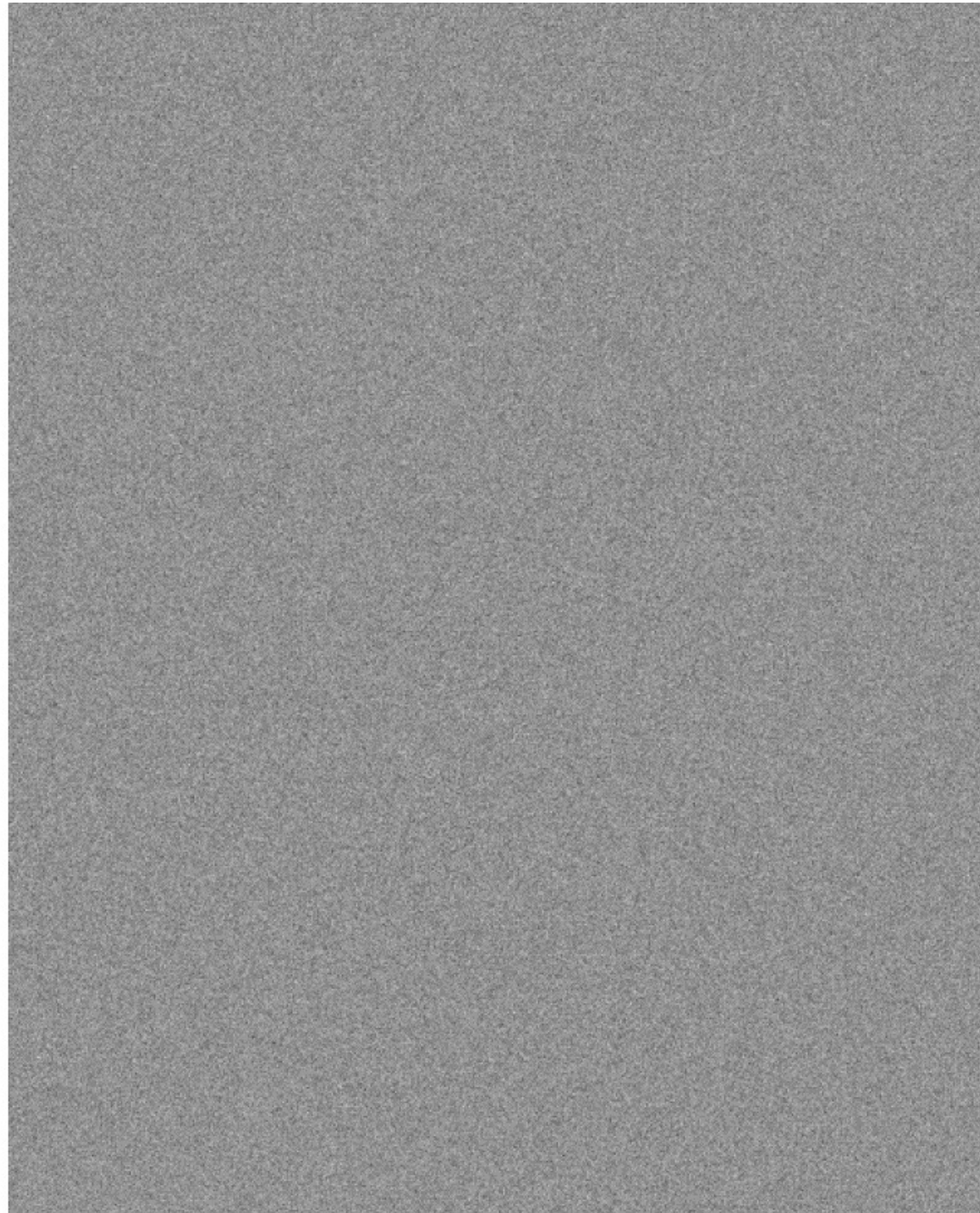
$\sigma=1$



Effect of
sigma on
Gaussian
noise:

This shows
the noise
values
added to the
raw
intensities of
an image.

sigma=
16



Effect of
sigma on
Gaussian
noise:

Image
shows the
noise values
themselves.

$\sigma=16$



Effect of
sigma on
Gaussian
noise

This shows
the noise
values
added to the
raw
intensities of
an image.

Noise Reduction

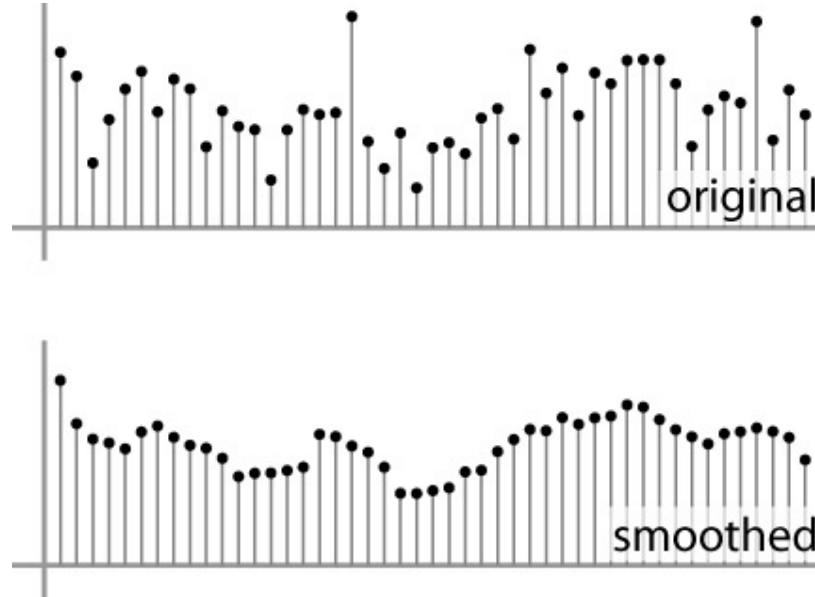
- How should we reduce the noise?

First attempt at a solution

- Let's replace each pixel with an average of all the values in its neighborhood (neighborhood operation)
- Assumptions:
 - Expect pixel to be like their neighbors
 - Expect noise processes to be independent from pixel to pixel

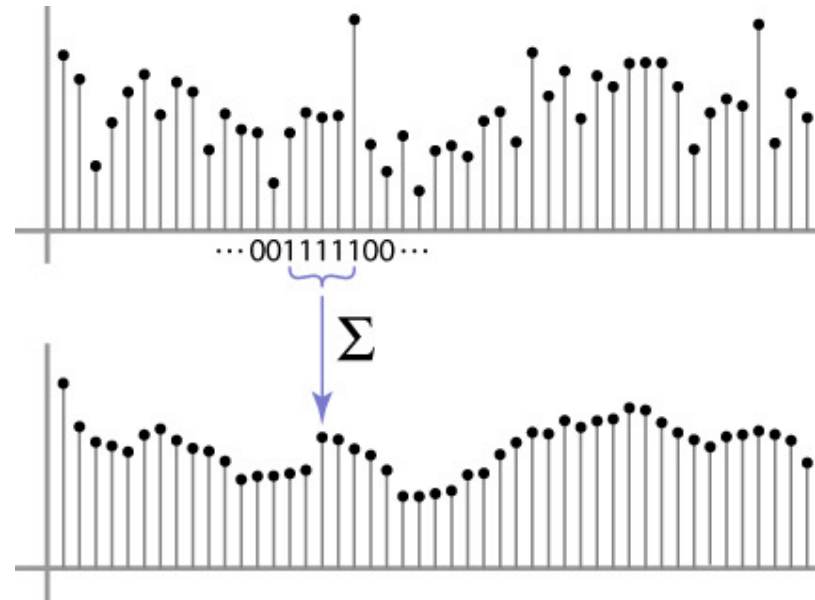
First attempt at a solution

- Let's replace each pixel with an average of all the values in its neighborhood
- Moving average in 1D:



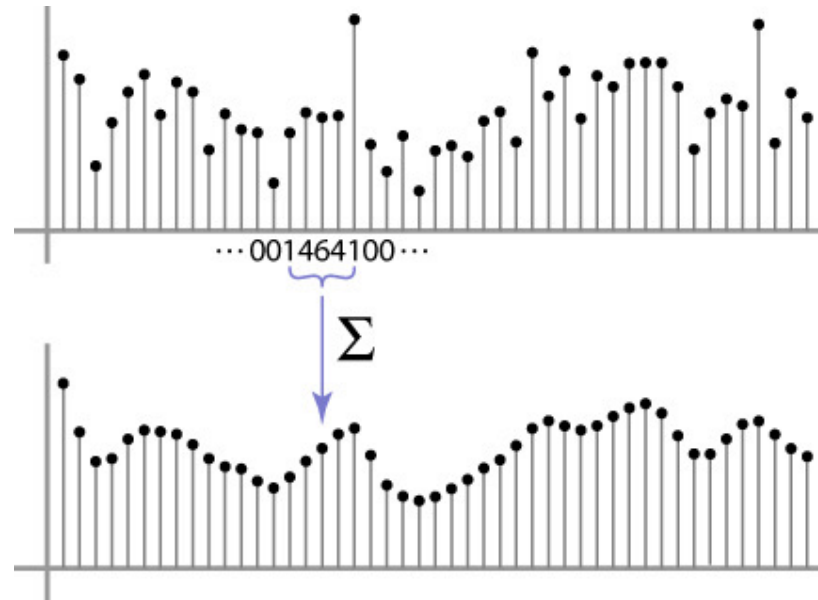
Weighted Moving Average

- Can add weights to our moving average
- *Weights* $[1, 1, 1, 1, 1] / 5$



Weighted Moving Average

- Non-uniform weights [1, 4, 6, 4, 1] / 16



Moving Average In 2D

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$G[x, y]$$

	0								

Moving Average In 2D

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$G[x, y]$$

	0	10							

Moving Average In 2D

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$G[x, y]$$

	0	10	20						

Moving Average In 2D

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$G[x, y]$$

	0	10	20	30					

Moving Average In 2D

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$G[x, y]$$

	0	10	20	30	30				

Moving Average In 2D

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$G[x, y]$$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Correlation filtering

Say the **averaging window size** is $(2k+1) \times (2k+1)$:

$$G[x, y] = \underbrace{\frac{1}{(2k+1)^2}}_{\text{Attribute uniform weight to each pixel}} \underbrace{\sum_{u=-k}^k \sum_{v=-k}^k F[x+u, y+v]}_{\text{Loop over all pixels in neighborhood around image pixel } F[x,y]} \quad \text{average over the window}$$

Now we can generalize to allow **different weights** depending on **neighboring pixel's relative position**:

$$G[x, y] = \sum_{u=-k}^k \sum_{v=-k}^k \underbrace{H[u, v]}_{\text{Non-uniform weights}} F[x+u, y+v]$$

Correlation filtering

after

$$G[x, y] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[x + u, y + v]$$

before

u, v define the size of window

kernel/mask

moving across the pixels in the window

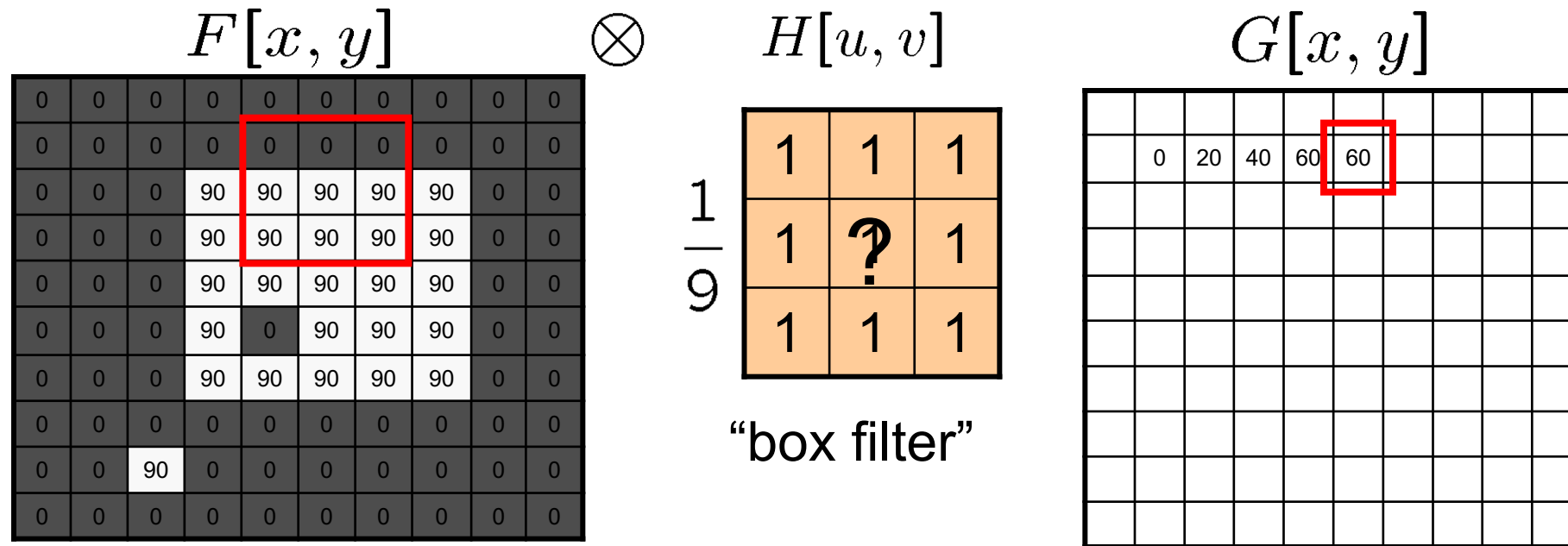
that depends on the coordinates in the window

This is called **cross-correlation**, denoted $G = H \otimes F$

- Filtering an image: replace each pixel with a **linear combination of its neighbors**.
- The filter **“kernel” or “mask”** $H[u, v]$ defines the weights in the linear combination.

Averaging filter

- What values belong to the kernel H for the moving average example?



$$G = H \otimes F$$

Smoothing by averaging



depicts box filter:
white = high value, black = low value



original



filtered

We can see the 'Block' effect on the filtered image.

as each neighbour
pixel contributes the
same

Image Filtering

- Image filtering: compute the function value of **local neighbourhood** at each pixel.
- Very useful
 - **Enhance images**
 - De-noise, resize, increase contrast, etc.
 - **Extract information from images**
 - Texture, edges, distinctive points, etc.
 - **Detect patterns**
 - Template matching

Gaussian filter

- What if we want nearest neighboring pixels to have the most influence on the output?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$F[x, y]$

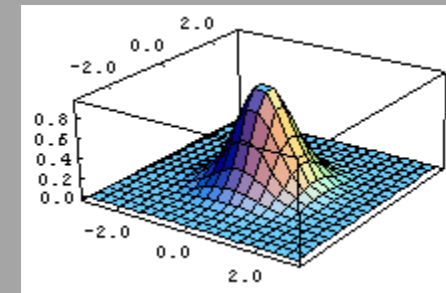
1	2	1
2	4	2
1	2	1

$H[u, v]$

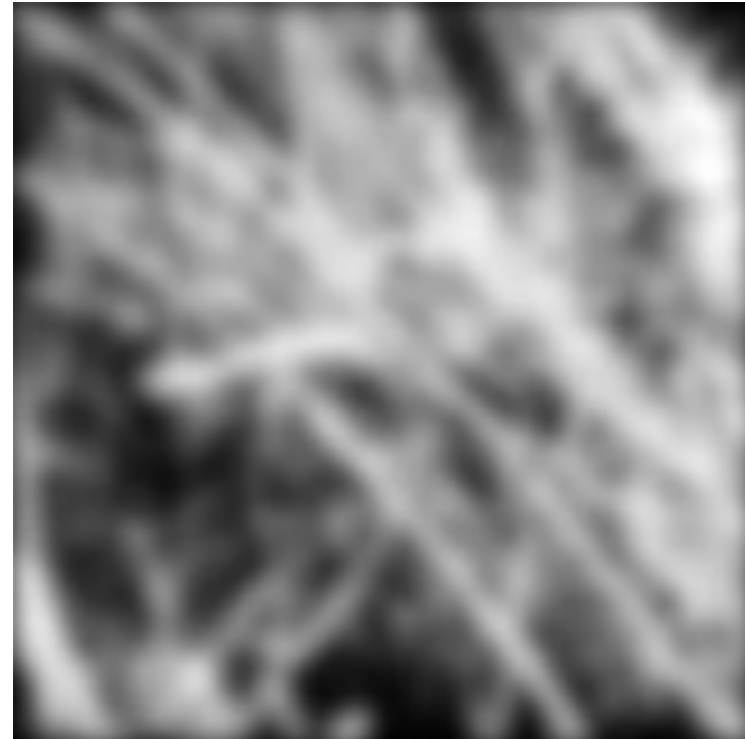
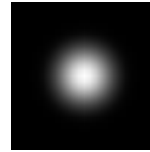
$\frac{1}{16}$

- This kernel is an approximation of a 2d Gaussian function:

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



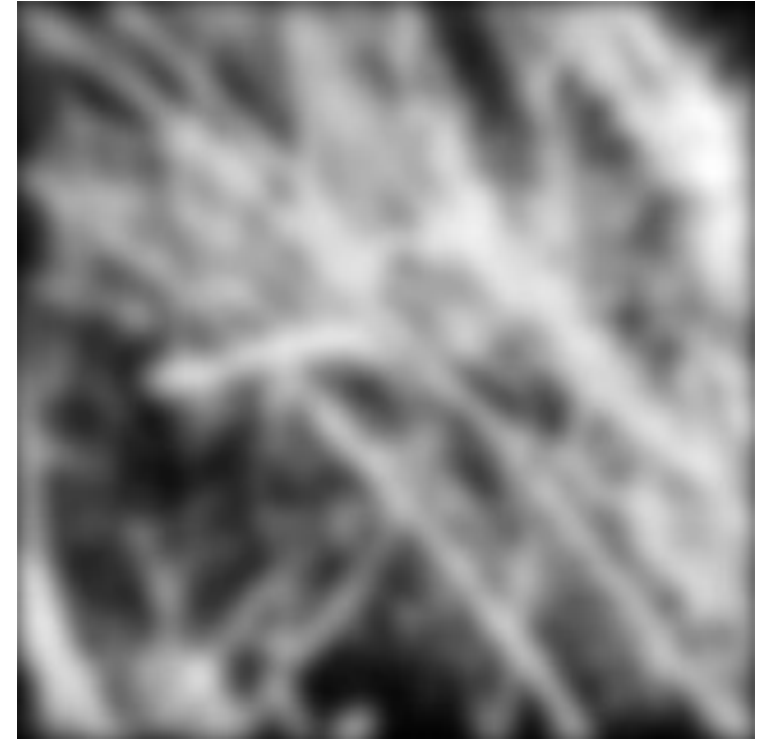
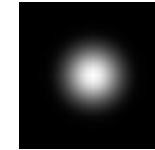
Smoothing with a Gaussian



Compare box filter with Gaussian filter effects



Smoothed image by box filter

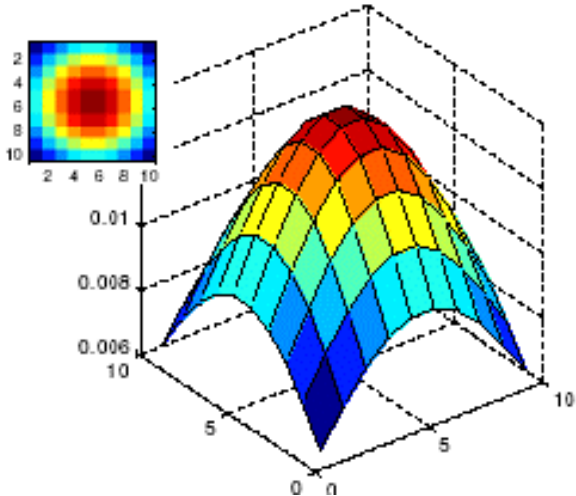
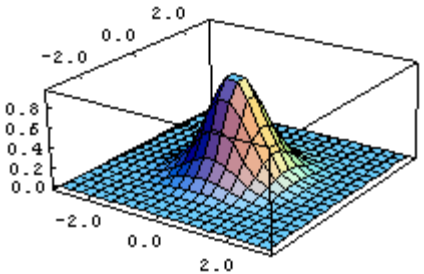


Smoothed image by Gaussian filter

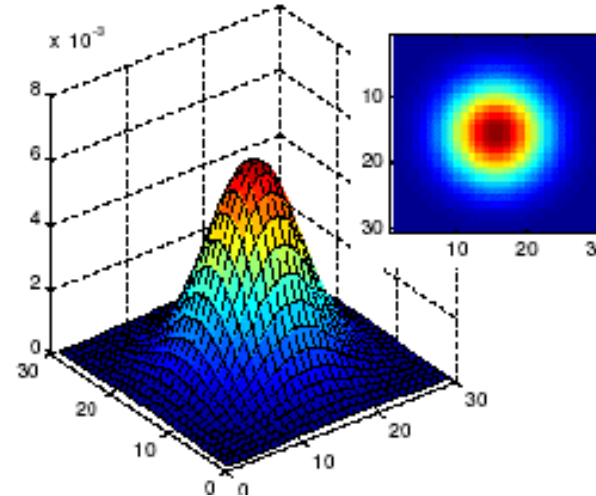
Gaussian filters

- What parameters matter here?
- **Size of kernel or mask**
 - Note, Gaussian function has infinite support, but discrete filters use finite kernels

$$h(u, v) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{u^2+v^2}{2\sigma^2}\right)$$



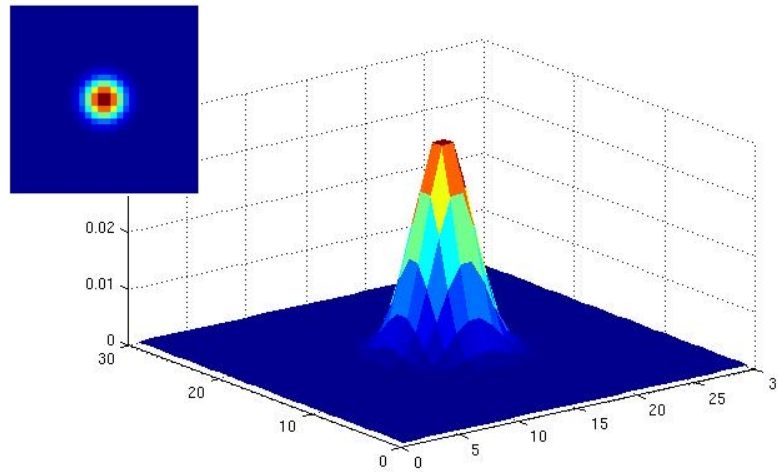
$\sigma = 5$ with
10x 10
kernel



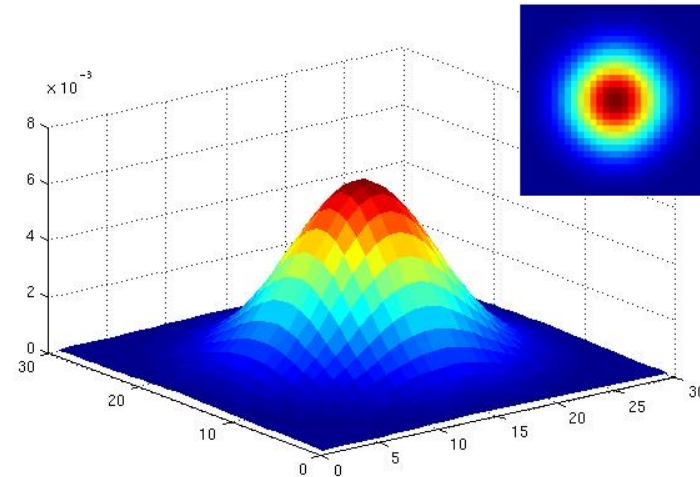
$\sigma = 5$ with
30 x 30
kernel

Gaussian filters

- What parameters matter here?
- **Variance** of the Gaussian function : **determines extent of smoothing**



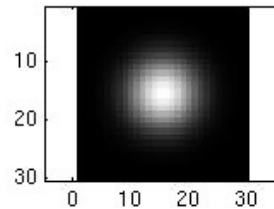
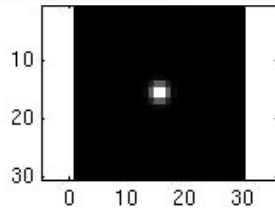
$\sigma = 2$ with
 30×30
kernel



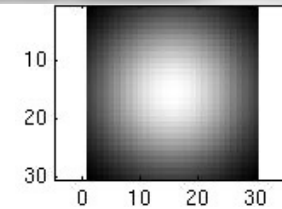
$\sigma = 5$ with
 30×30
kernel

Smoothing with a Gaussian

Parameter σ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.



...



Convolution

- Convolution:
 - Flip the filter in both dimensions (bottom to top, right to left)
 - Then apply cross-correlation

$$G[x, y] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[x - u, y - v]$$

$$G = H \star F$$

↑
*Notation for
convolution
operator*

Convolution

- Convolution:
 - Flip the filter in both dimensions (bottom to top, right to left)
 - Then apply cross-correlation

$$G(x, y) = \sum_{u=0}^M \sum_{v=0}^N F(u, v) H(x - u, y - v)$$

$$G = F \star H$$

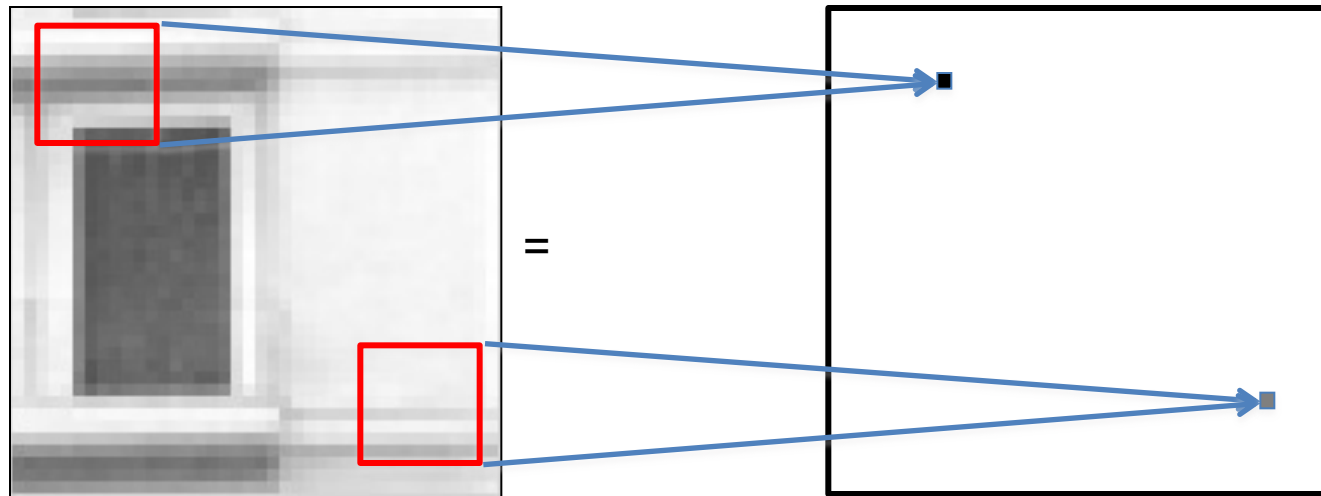
↑
*Notation for
convolution
operator*

Discrete linear system



In vision, many times, we are interested in operations that are spatially invariant.
For a linear spatially invariant system:

$$g(m, n) = h \star f = \sum_{u=-k}^{u=k} \sum_{v=-k}^{v=k} f(m-u, n-v) h(u, v)$$



Properties of convolution

- **Shift invariant:**

- Operator behaves the same everywhere, i.e. the value of the output depends on the pattern in the **image neighborhood**, not the position of the neighborhood.

Properties of convolution

- Commutative:

$$f * g = g * f$$

- Associative

$$(f * g) * h = f * (g * h)$$

- Distributes over addition

$$f * (g + h) = (f * g) + (f * h)$$

- Scalars factor out

$$kf * g = f * kg = k(f * g)$$

- Identity:

$$\text{unit impulse } e = [\dots, 0, 0, 1, 0, 0, \dots]. \quad f * e = f$$

Separability

- In some cases, the filter is **separable**, and we can factor into two steps:
 - Convolve all rows with a 1D filter
 - Convolve all columns with a 1D filter

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 0 \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 \end{bmatrix}$$

element-wise
multiplication

Separability example

- 2D convolution (center location only)(65)

1	2	1
2	4	2
1	2	1

 \ast

2	3	3
3	5	5
4	4	6

$$\begin{aligned} &= 2 + 6 + 3 = 11 \\ &= 6 + 20 + 10 = 36 \\ &= 4 + 8 + 6 = 18 \\ &\hline &65 \end{aligned}$$

h

f

Separability example

1	2	1
2	4	2
1	2	1

 \ast

2	3	3
3	5	5
4	4	6

$= 2 + 6 + 3 = 11$
 $= 6 + 20 + 10 = 36$
 $= 4 + 8 + 6 = 18$

65

- The filter is factored into a product of 1D filters.

1	2	1
2	4	2
1	2	1

 $=$

1
2
1

 \times

1	2	1
---	---	---

Separability example

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array} = \begin{array}{l} = 2 + 6 + 3 = 11 \\ = 6 + 20 + 10 = 36 \\ = 4 + 8 + 6 = 18 \\ \hline 65 \end{array}$$

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} = \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array}$$

- Perform convolutions along rows.

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline & 11 & \\ \hline & 18 & \\ \hline & 18 & \\ \hline \end{array}$$

Separability example

1	2	1
2	4	2
1	2	1

 \ast

2	3	3
3	5	5
4	4	6

$$\begin{aligned} &= 2 + 6 + 3 = 11 \\ &= 6 + 20 + 10 = 36 \\ &= 4 + 8 + 6 = 18 \end{aligned}$$

65

1	2	1
2	4	2
1	2	1

 $=$

1
2
1

 \times

1	2	1
---	---	---

1	2	1
---	---	---

 \ast

2	3	3
3	5	5
4	4	6

 $=$

	11	
	18	
	18	

- Perform convolutions along columns.

1
2
1

 \ast

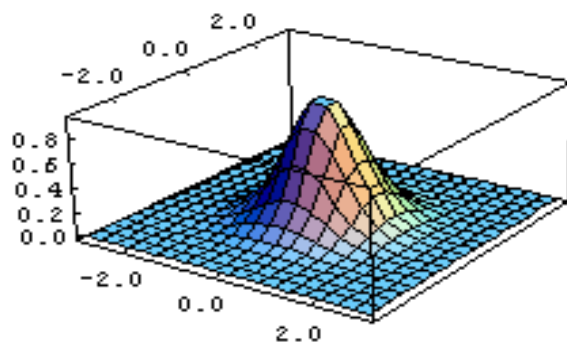
	11	
	18	
	18	

 $=$

	65	

This kernel is an approximation of
a 2d Gaussian function:

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



Properties of Gaussian filter

- Rotational symmetry treats features of all orientations equally (isotropy).
- Convolution with self gives another Gaussian
 - So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
 - Convolving two times with Gaussian kernel of width σ is same as convolving once with kernel of width $\sigma\sqrt{2}$
- *Separable* kernel
 - Factors into the product of two 1D Gaussians

Separability of the Gaussian filter

$$\begin{aligned} G_{\sigma}(x, y) &= \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2 + y^2}{2\sigma^2}} \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{x^2}{2\sigma^2}} \right) \left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{y^2}{2\sigma^2}} \right) \end{aligned}$$

The 2D Gaussian can be expressed as the product of two functions, one a function of x and the other a function of y

In this case, the two functions are the (identical) 1D Gaussian

- What is the complexity of filtering an $n \times n$ image with an $m \times m$ kernel?

- $O(n^2 m^2)$

- What if the kernel is separable?

- $O(n^2 m)$

1. 1D Convolution along Rows:

- Convolve each row of the image ($n \times n$) with a 1D kernel of size m .
- The complexity is $O(n \cdot n \cdot m)$ for the row-wise convolution.

$$\text{RowConv} = O(n \cdot n \cdot m)$$

2. 1D Convolution along Columns:

- Convolve each column of the intermediate result ($n \times n$) with a 1D kernel of size m .
- The complexity is $O(n \cdot n \cdot m)$ for the column-wise convolution.

$$\text{ColConv} = O(n \cdot n \cdot m)$$

3. Total Complexity:

- The total complexity is the sum of the row-wise and column-wise convolutions.

$$\text{Total Complexity} = \text{RowConv} + \text{ColConv} = O(n \cdot n \cdot m)$$

Readings

- Computer Vision: Algorithms and Applications, Chapter 3.1, 3.2 and 3.3