CSC 433/533
Computer Graphics

Joshua Levine
josh@email.arizona.edu

Lecture 05
Signal Processing

Sept. 11, 2019

Today's Agenda

- Reminders:
  - A01 due!
  - A02 posted, will discuss at end of class.
- Goals for today:
  - Introduce images as samples and signal processing

Recall:
Images are Functions

Domains and Ranges

- All functions have two components, the **domain** and **range**. For the case of images, I: R → V

Domains and Ranges

- All functions have two components, the **domain** and **range**. For the case of images, I: R → V

- The domain is:

Domains and Ranges

- All functions have two components, the **domain** and **range**. For the case of images, I: R → V
- The domain is:
  - R, is some rectangular area ($R \subseteq \mathbb{R}^2$)

Domains and Ranges

- All functions have two components, the **domain** and **range**. For the case of images, I: R → V
- The domain is:
  - R, is some rectangular area ($R \subseteq \mathbb{R}^2$)
- The range is:

Domains and Ranges

- All functions have two components, the **domain** and **range**. For the case of images, $I: R \rightarrow V$
- The domain is:
  - R, is some rectangular area ($R \subseteq \mathbb{R}^2$)
- The range is:
  - A set of possible values.

Domains and Ranges

- All functions have two components, the **domain** and **range**. For the case of images, I: R → V
- The domain is:
  - R, is some rectangular area ($R \subseteq \mathbb{R}^2$)
- The range is:
  - A set of possible values.
  - …in the space of color values we're encoding


Boo!


Boo!
Hi Everyone!

Operations on Images

Slides inspired from Fredo Durand

- Point (Range) Operations:

- Affect only the range of the image (e.g. brightness)
- Each pixel is processed separately, only depending on the color

Operations on Images

Slides inspired from Fredo Durand

- Point (Range) Operations:

- Affect only the range of the image (e.g. brightness)
- Each pixel is processed separately, only depending on the color

Operations on Images

Slides inspired from Fredo Durand

- Domain Operations:

- Only move the pixels around

Operations on Images

Slides inspired from Fredo Durand

- Domain Operations:

- Only move the pixels around

Operations on Images

Slides inspired from Fredo Durand

- Neighborhood operations:

- Combine domain and range
- Each pixel evaluated by working with other pixels nearby

Operations on Images

Slides inspired from Fredo Durand

- Neighborhood operations:

- Combine domain and range
- Each pixel evaluated by working with other pixels nearby

Concept for the Day: Pixels are Samples of Image Functions

Jim Blinn's Corner

http://www.research.microsoft.com/~blinn/

What Is a Pixel?

James F. Blinn

Microsoft Research

I am not a major sports fan. But there is one story from the folklore of football that has always intrigued me. The story goes that legendary football coach Vince Lombardi was observing his new players, recruited from the best college teams and all presumably excellent players. He was, however, not pleased with their performance. So he called them all to a meeting, which he began by holding up the essential tool of their trade and saying, "This is a football." I was sufficiently impressed by this back-to-basics attitude that, when I taught computer graphics rendering classes, I used to start the first lecture of the term by going to the blackboard (boards were black then, not white) and drawing a little dot and saying, "This is a pixel."

But was I right? Most computer graphicists would agree that the pixel is the fundamental atomic element of imaging. But what is a pixel really? As I have played with various aspects of pixel mashing, it has occurred to me that the concept of the pixel is really multifaceted (to use a weird metaphor.) Perhaps a better metaphor that spirit I am going to list some possible meanings for a pixel that I will expand on in some later columns.

A pixel is a little square

Early 2D windowing systems considered a pixel a little square. This is perhaps the simplest possible definition, but it only works if you are mostly drawing horizontal and vertical lines and rectangles that are integral numbers of pixels in size. Anything at fractional pixel size or at an angle yields jaggies and other forms of aliasing.

A pixel is a point sample of a continuous function

A more enlightened signal processing approach thinks of a pixel as a point sample of a continuous function (see the dots in Figure 1). (This approach was actually taken by the rendering community long before pixel displays were used for user interfaces and windowing systems.) Applying linear filtering theory to pixel manip-

Image Samples

- Each pixel is a sample of what?
  - One interpretation: a pixel represents the intensity of light at a single (infinitely small point in space)
- The sample is displayed in such a way as to spread the point out across some spatial area (drawing a square of color)

Continuous vs. Discrete

- Key Idea: An image represents data in either (both?) of
  - Continuous domain: where light intensity is defined at every (infinitesimally small) point in some projection
  - Discrete domain, where intensity is defined only at a discretely sampled set of points.

Converting Between Image Domains

- When an image is acquired, an image is **sampled** from some continuous domain to a discrete domain.
- **Reconstruction** converts digital back to continuous.
- The reconstructed image can then be **resampled** and **quantized** back to the discrete domain.

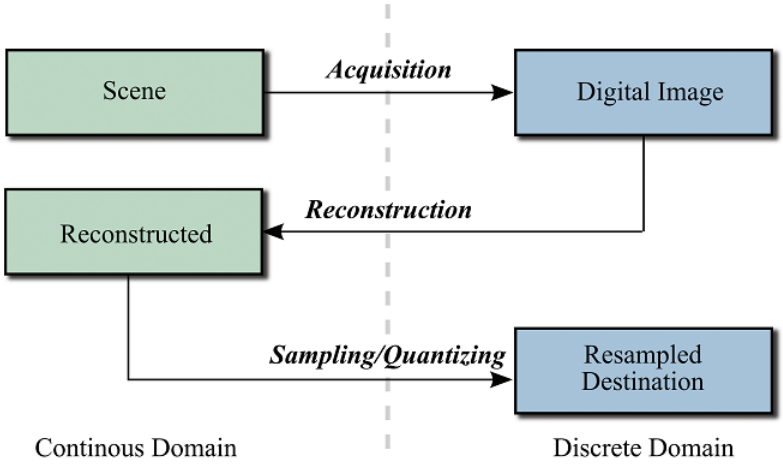


Figure 7.7. Resampling.

Naive Image Rescaling Code

```
//scale factor
let k = 4;

//create an output greyscale image that is both
//k times as wide and k times as tall
Uint8Array output = new Uint8Array((k*W)*(k*H));

//copy the pixels over
for (let row = 0, row < H; row++) {
   for (let col = 0; col < W; col++) {
      let index = row*W + col;
      let index2 = (k*row)*W + (k*col);
      output[index2] = input[index];
   }
}
```

Naive Image Rescaling

- Consider resizing an image to a large resolution
- Simple approach: Take all the pixels in input and place them in an output location.

100x100 image

N g

- C
- S
  t

What’s the Problem?

What’s the Problem?

- The output image has gaps!

What’s the Problem?

- The output image has gaps!
- Why: we skip a many of the pixels in the output.

What's the Problem?

- The output image has gaps!
- Why: we skip a many of the pixels in the output.
- Why don't we fix this by changing the code to at least put some color at each pixel of the output?

Naive Image Rescaling Code

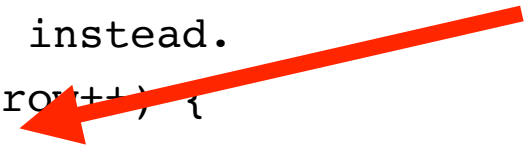
```
//scale factor
let k = 4;

//create an output greyscale image that is both
//k times as wide and k times as tall
Uint8Array output = new Uint8Array((k*W)*(k*H));

//copy the pixels over
for (let row = 0, row < H; row++) {
   for (let col = 0; col < W; col++) {
      let index = row*W + col;
      let index2 = (k*row)*W + (k*col);
      output[index2] = input[index];
   }
}
```

“Inverse” Image Rescaling Code

```
//scale factor
let k = 4;

//create an output greyscale image that is both
//k times as wide and k times as tall
Uint8Array output = new Uint8Array((k*W)*(k*H));

//Loop over each output pixel instead.
for (let row = 0, row < k*H; row++) {
   for (let col = 0; col < k*W; col++) {
      let index = (row/k)*W + (col/k);
      let index2 = row*k*W + col;
      output[index2] = input[index];
   }
}
```

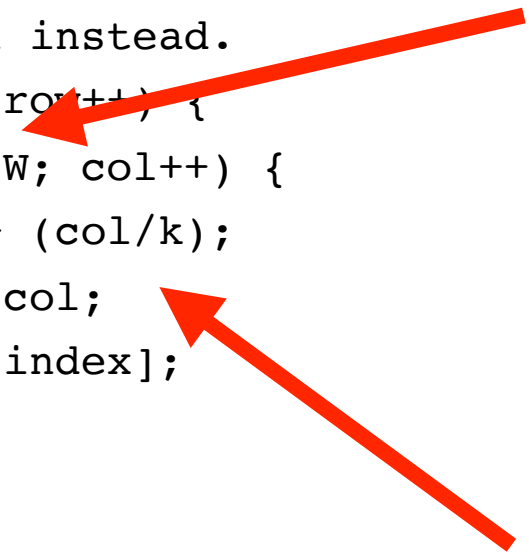
“Inverse” Image Rescaling Code

```
//scale factor
let k = 4;

//create an output greyscale image that is both
//k times as wide and k times as tall
Uint8Array output = new Uint8Array((k*W)*(k*H));

//Loop over each output pixel instead.
for (let row = 0, row < k*H; row++) {
   for (let col = 0; col < k*W; col++) {
      let index = (row/k)*W + (col/k);
      let index2 = row*k*W + col;
      output[index2] = input[index];
   }
}
```



“Inverse” Image Rescaling Code

```
//scale factor
let k = 4;

//create an output greyscale image that is both
//k times as wide and k times as tall
Uint8Array output = new Uint8Array((k*W)*(k*H));

//Loop over each output pixel instead.
for (let row = 0, row < k*H; row++) {
   for (let col = 0; col < k*W; col++) {
      let index = (row/k)*W + (col/k);
      let index2 = row*k*W + col;
      output[index2] = input[index];
   }
}
```



Inverse Image Rescaling

100x100 image

Ir g

What’s the Problem?

What’s the Problem?

- The output image is too “blocky”

What’s the Problem?

- The output image is too “blocky”
- Why: because our image reconstruction rounds the index to the nearest integer pixel coordinates

What's the Problem?

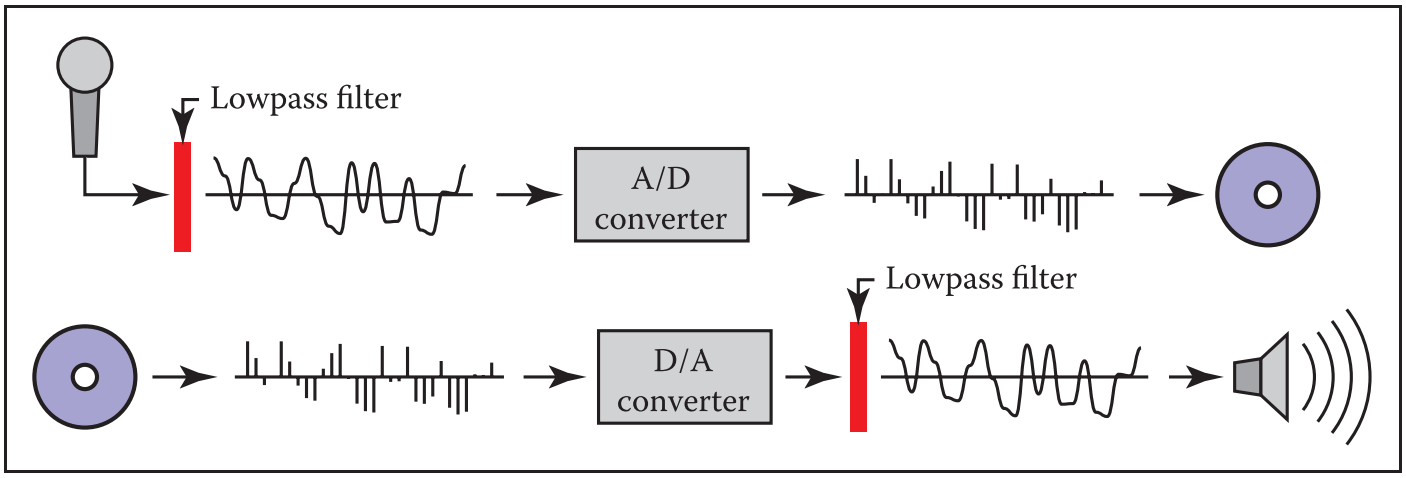
- The output image is too "blocky"
- Why: because our image reconstruction rounds the index to the nearest integer pixel coordinates
    - Rounding to the "nearest" is why this type of interpolation is called **nearest neighbor interpolation**

Sampling Artifacts / Aliasing

Motivation: Digital Audio

- Acquisition of images takes a continuous object and converts this signal to something digital

- Two types of artifacts:
  - **Undersampling** artifacts: on acquisition side
  - **Reconstruction** artifacts: when the samples are interpreted



Undersampling Artifacts

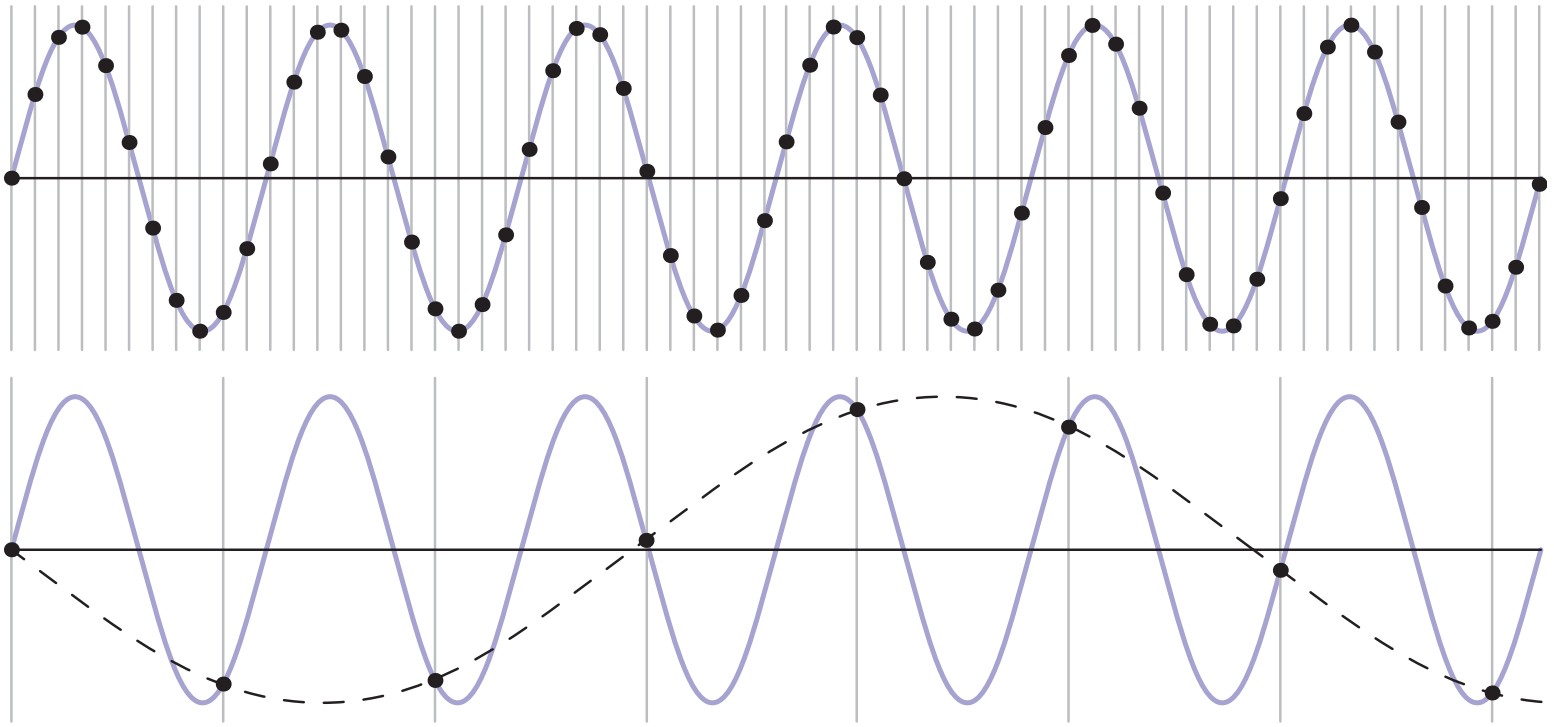


Image Reduction

- Consider reducing the high resolution image:


Boo!

Image Reduction

- Consider reducing the high resolution image:

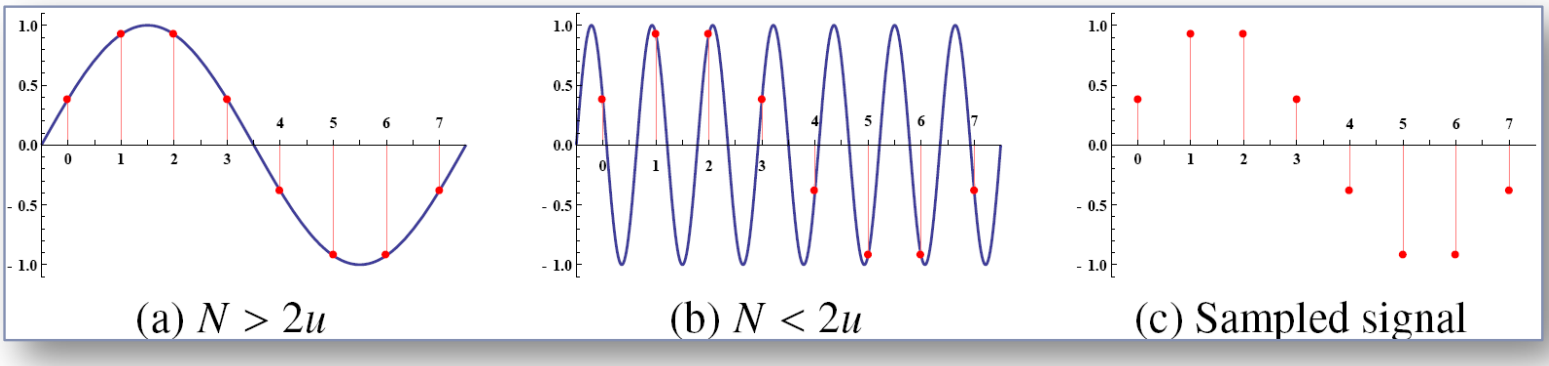
Image Reduction

- Consider reducing the high resolution image:

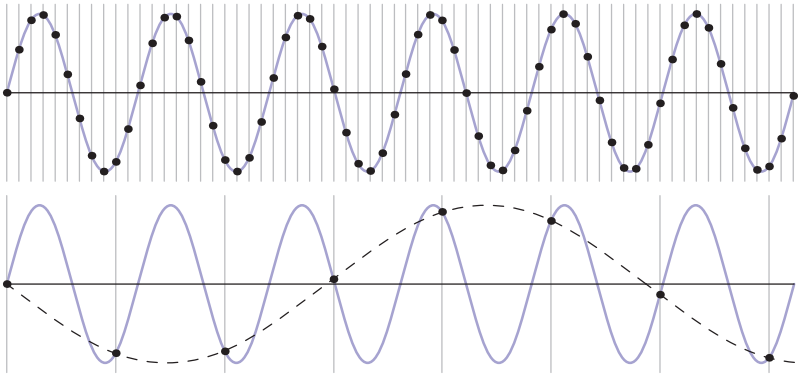
n

Aliasing Example

- (image a) Consider a sinusoid such that u=1 over a span of N = 8 units.
- (image b) Consider a sinusoid such that u=7 over a span of N = 8 units.
- The resulting samples are identical. The two different signals are indistinguishable after sampling and hence are aliases for each other.



Shannon-Nyquist Theorem

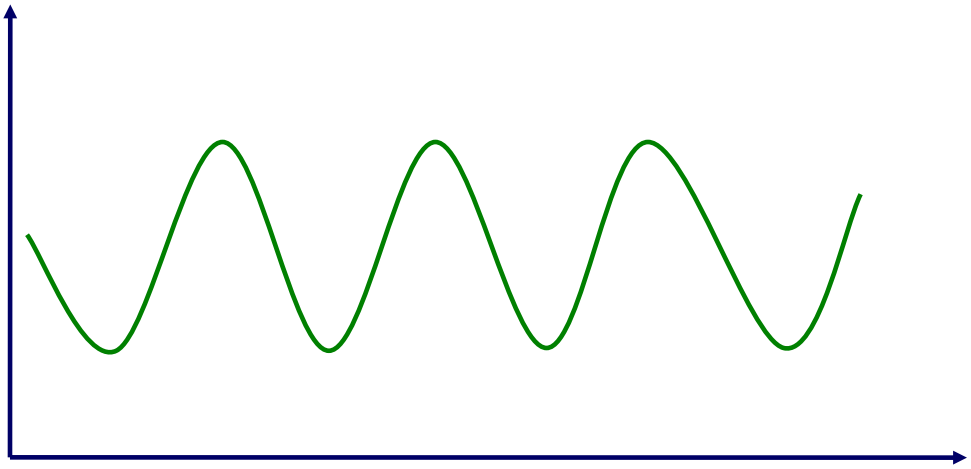


- The sampling frequency must be **<u>double</u>** the highest frequency of the content.
- If there are any higher frequencies in the data, or the sampling rate is too low, **aliasing**, happens
  - Named this because the discrete signal “pretends” to be something lower frequency

S-N Theorem Illustrated

How many samples are enough to avoid aliasing?

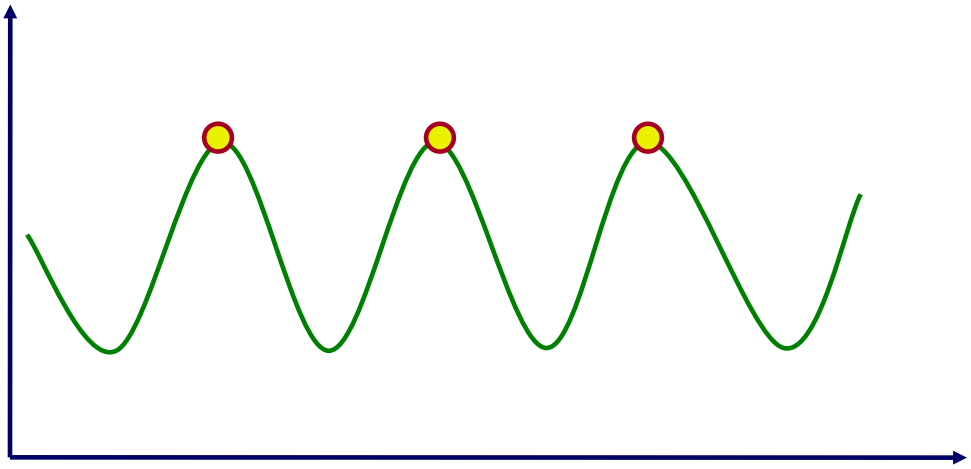
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



S-N Theorem Illustrated

How many samples are enough to avoid aliasing?

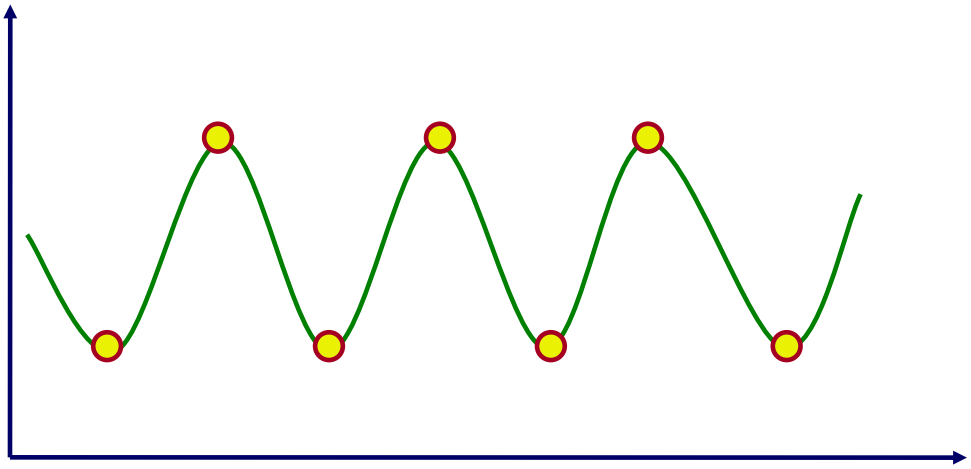
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



S-N Theorem Illustrated

How many samples are enough to avoid aliasing?

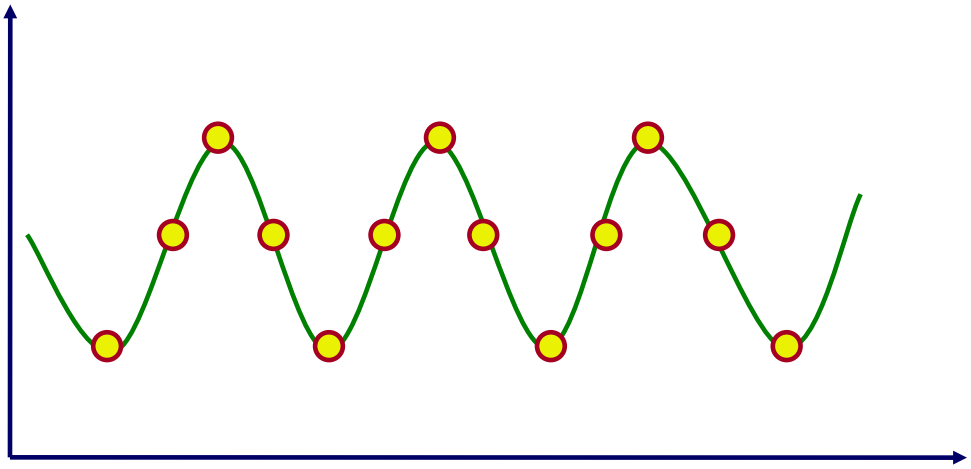
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



S-N Theorem Illustrated

How many samples are enough to avoid aliasing?

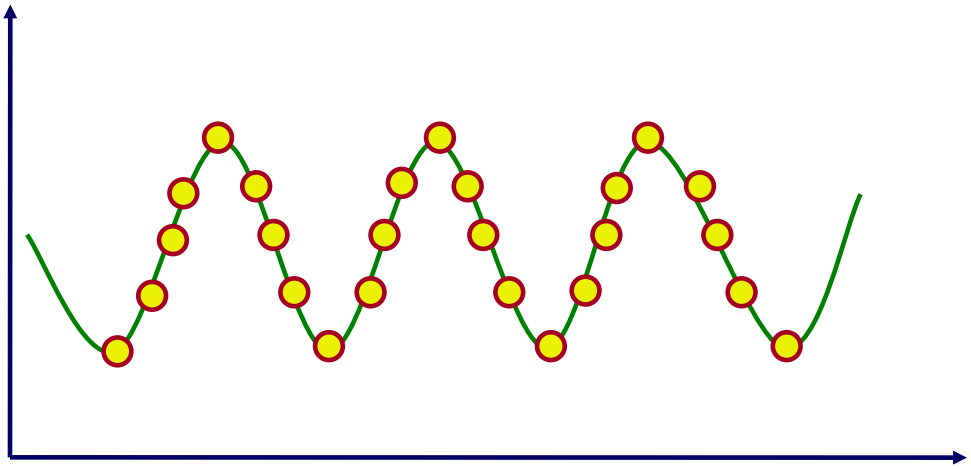
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



S-N Theorem Illustrated

How many samples are enough to avoid aliasing?

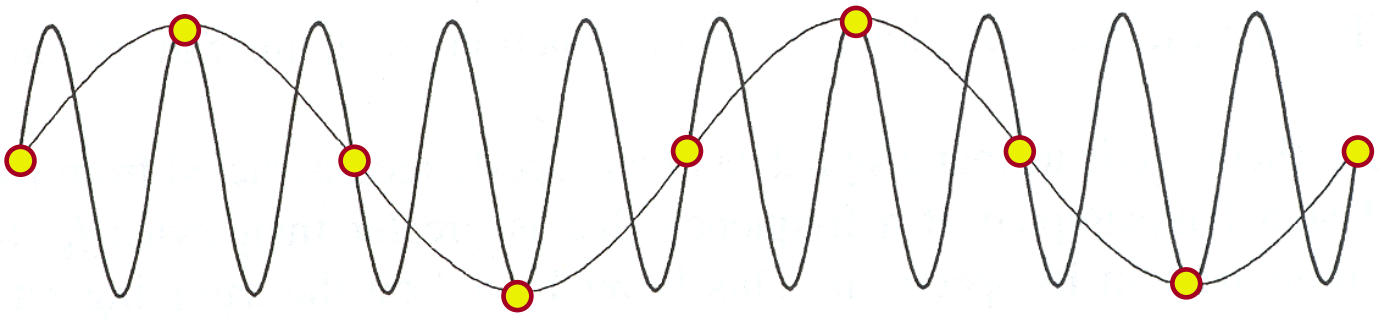
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



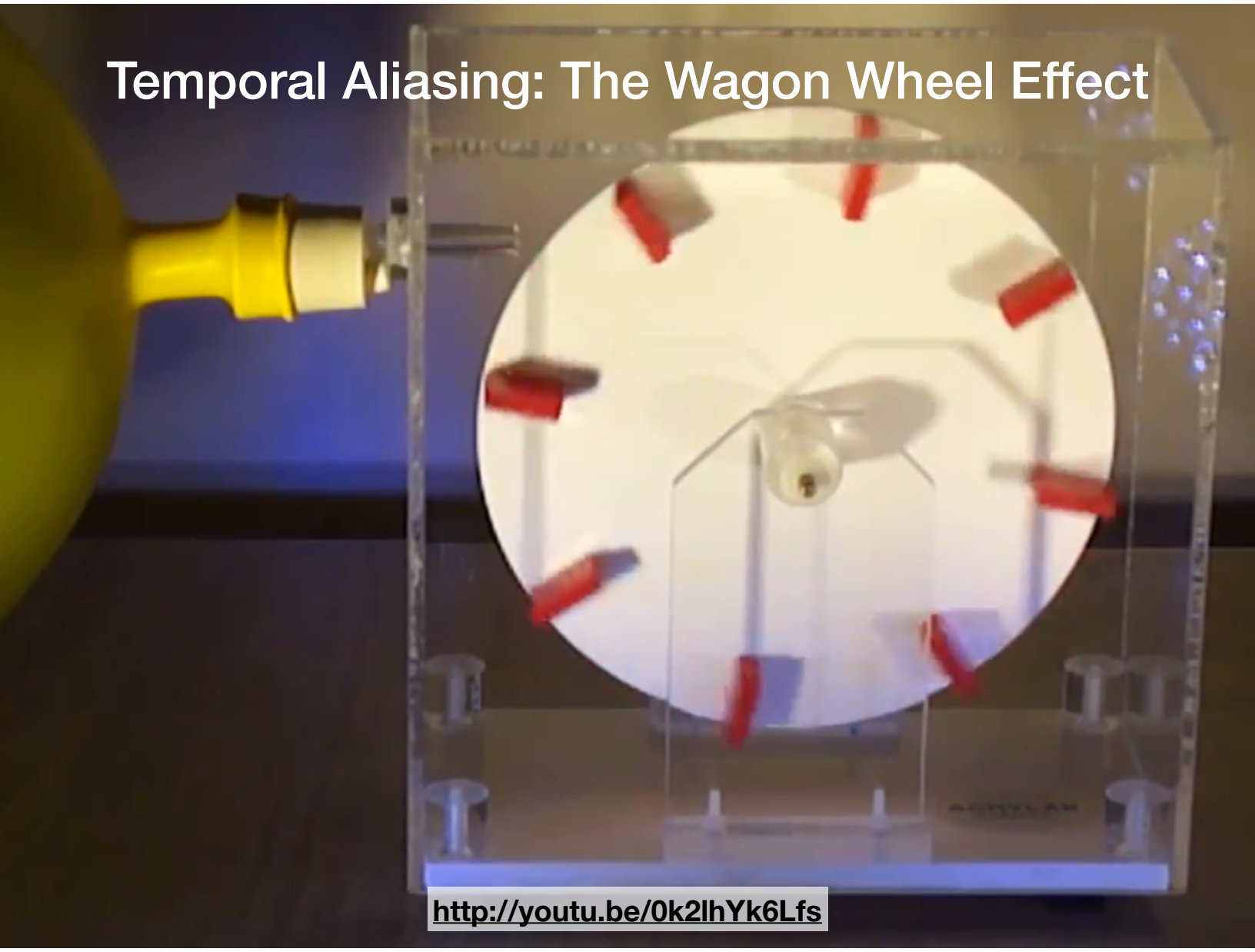
Shannon-Nyquist Theorem

A signal can be reconstructed from its samples, iff the original signal has no content >= 1/2 the sampling frequency - Shannon

Aliasing will occur if the signal is under-sampled



Temporal Aliasing: The Wagon Wheel Effect



http://youtu.be/0k2lhYk6Lfs

Convolution

An Example: Mean Filtering

- Mean filters sum all of the pixels in a local neighborhood N_i and divide by the total number, computing the average pixel.
- Said another way, we replace each pixel as a linear combination of its neighbors (with equal weights!)
- $f(N_i) = 1 / |N_i| \Sigma C_j$, for pixel j in N_i
- Where the N_i is a square, we call these **box** filters

Box Filtering

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix}$$

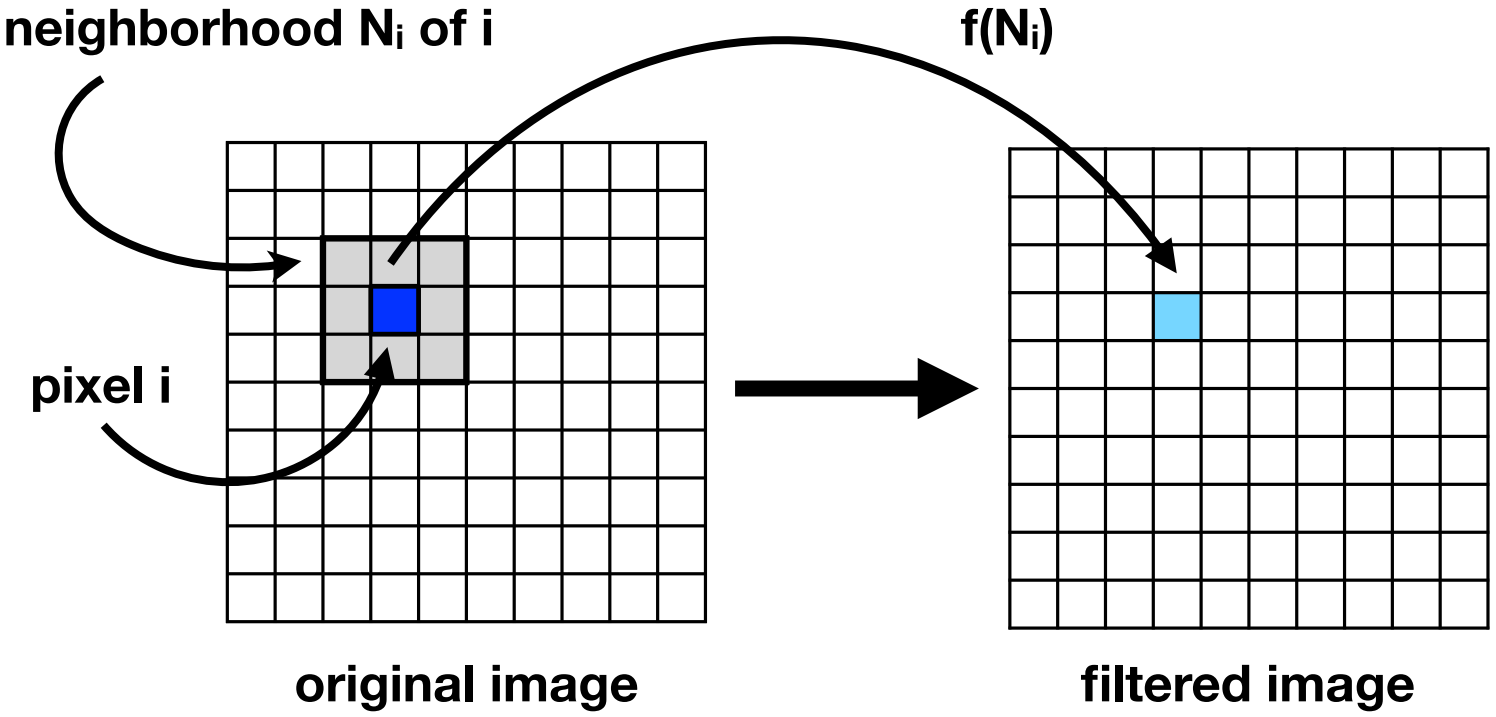
$$1/9 * \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Box Filtering

$$\frac{1}{9} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

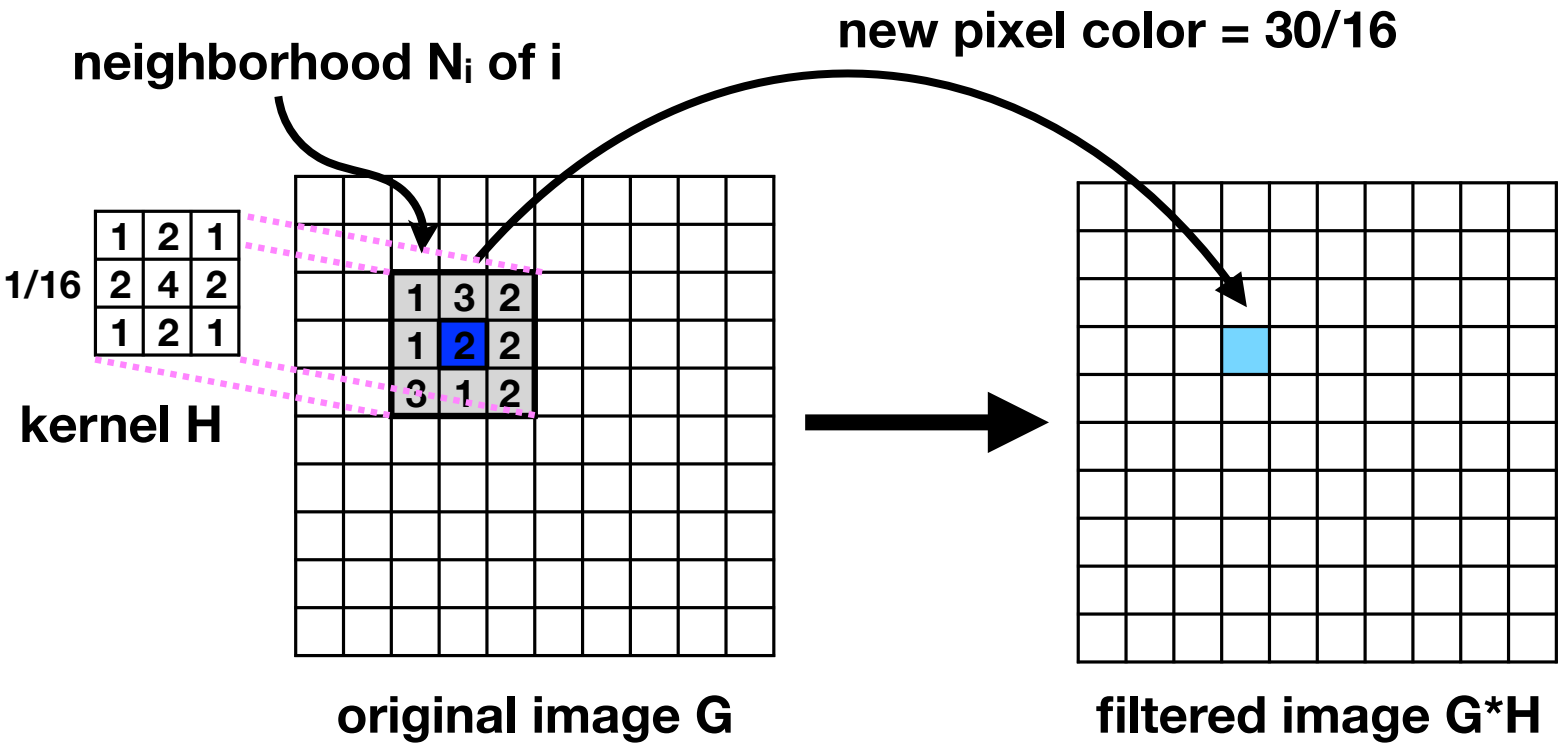
$$\frac{1}{25} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Neighborhood Filtering (Schematic)



Convolution

- This process of adding up pixels multiplied by various weights is called **convolution**



Kernels

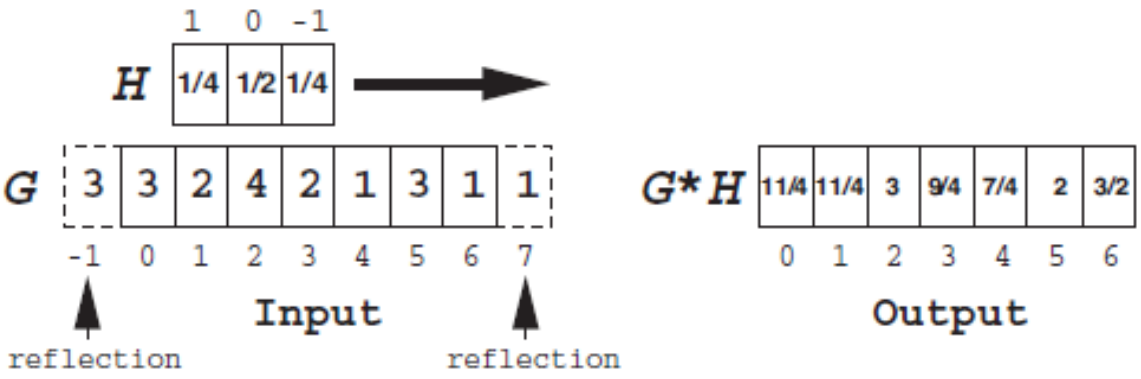
- Convolution employs a rectangular grid of coefficients, known as a **kernel**
- Kernels are like a neighborhood mask, they specify which elements of the image are in the neighborhood and their relative weights.
- A kernel is a set of weights that is applied to corresponding input samples that are summed to produce the output sample.

One-dimensional Convolution

- Can be expressed by the following equation, which takes a filter H and convolves it with G:

$$\hat{G}[i] = (G * H)[i] = \sum_{j=i-n}^{i+n} G[j]H[i-j],\ i \in [0, N-1]$$

- Equivalent to sliding a window

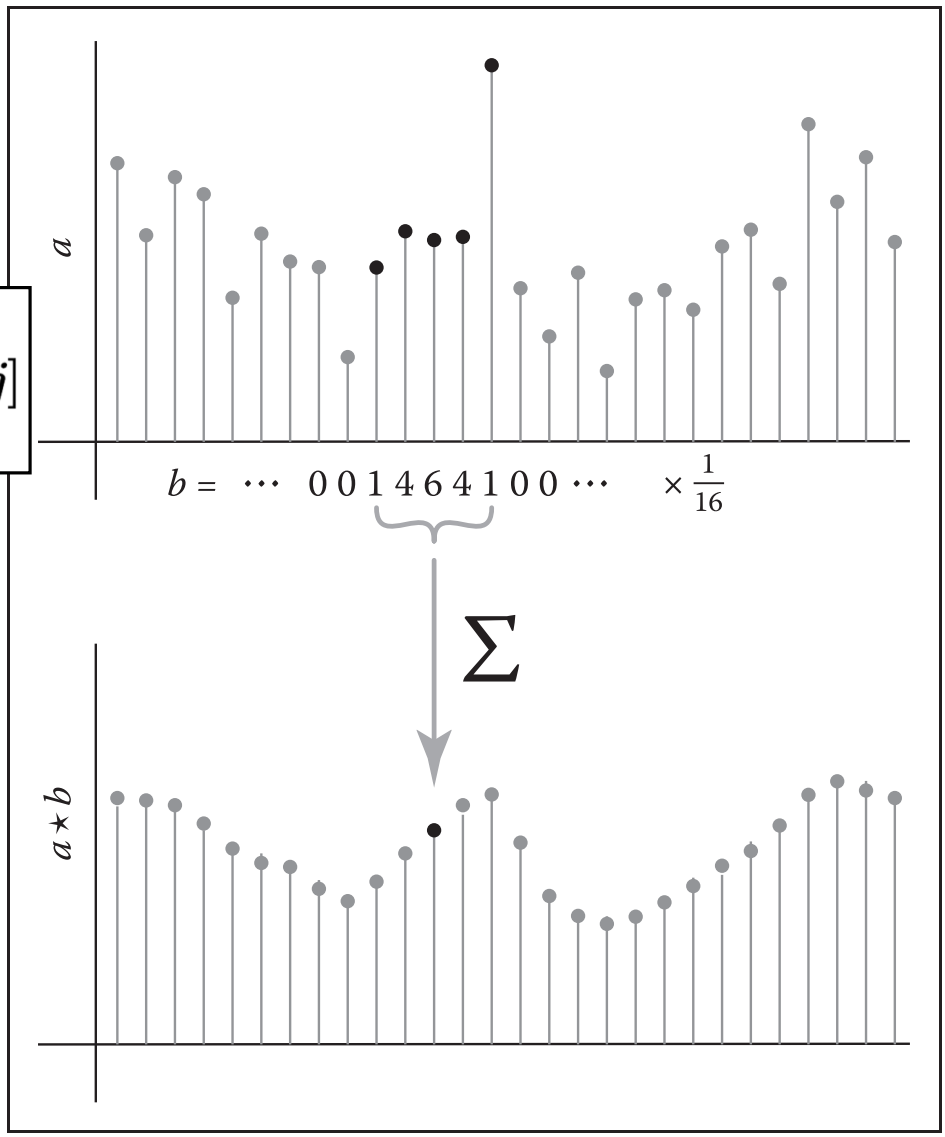


Convolution is a Moving, Weighted Average

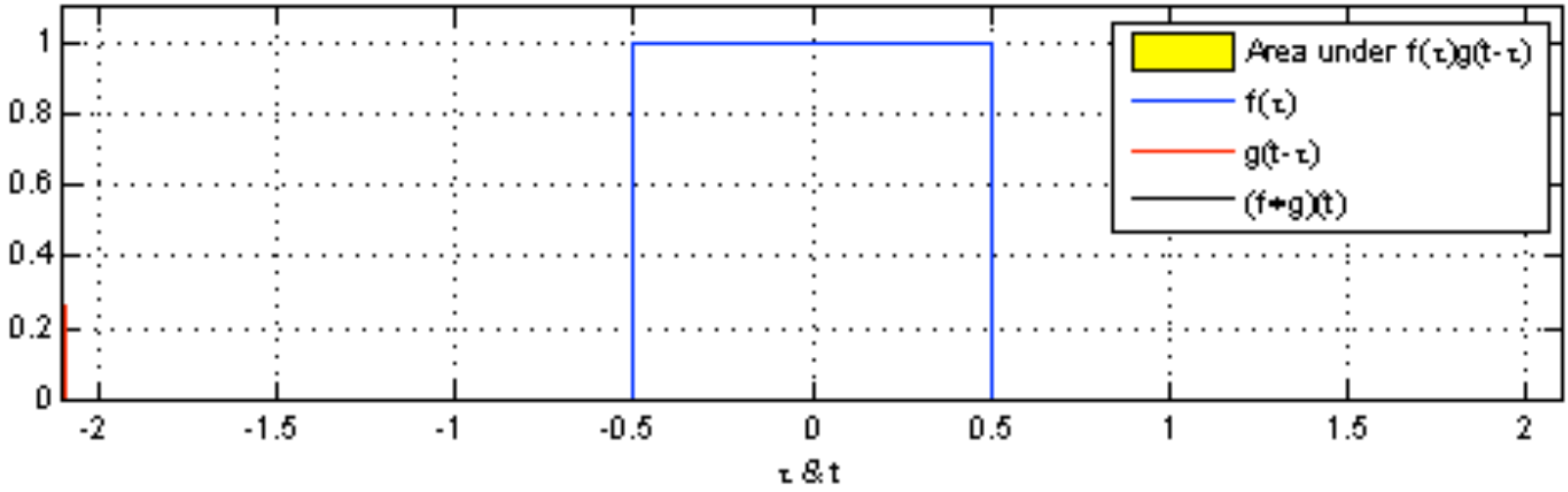
$$(a \star b)[i] = \sum_{j=i-r}^{i+r} a[j]b[i-j]$$

- Mathematically, this is equivalent to integrating the product of a and b with a shift in the domain

- Compare a to a*b on the right



Box Filters (Animated)



- The above is continuous, but a discrete version could be imagined as:

H = 111

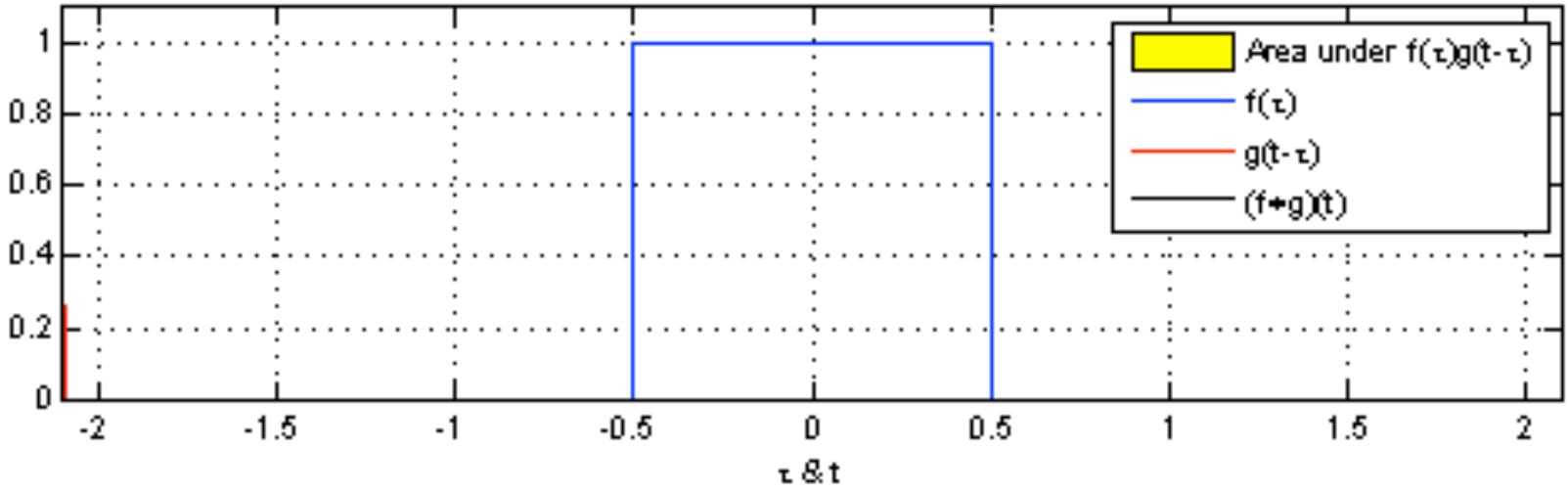
G = ...0000011100000...

G*H = ...0000012100000...

http://en.wikipedia.org/wiki/Convolution

More examples: http://math.mit.edu/daimp/ConvFlipDrag.html

Box Filters (Animated)



- The above is continuous, but a discrete version could be imagined as:

H = 111

G = ...0000011100000...

G*H = ...0000012100000...

http://en.wikipedia.org/wiki/Convolution

More examples: http://math.mit.edu/daimp/ConvFlipDrag.html

2-Dimensional Version

- Given an image a and a kernel b with $(2r+1)^2$ values, the convolution of a with b is given below as a*b:

$$(a \star b)[i,j] = \sum_{i'=i-r}^{i+r} \sum_{j'=j-r}^{j+r} a[i',j']\, b[i-i', j-j']$$

- The (i-i') and (j-j') terms can be understood as reflections of the kernel about the central vertical and horizontal axes.

- The kernel weights are multiplied by the corresponding image samples and then summed together.

A Note on Indexing

- Convolution **reflects** the filter to preserve orientation.
- **Correlation** does **not** have this reflection.
  - But we often use them interchangeably since most kernels are symmetric!!

Convolution reflects and shifts the kernel

Given kernel H =

1	2	3
4	5	6
7	8	9

An Illustration

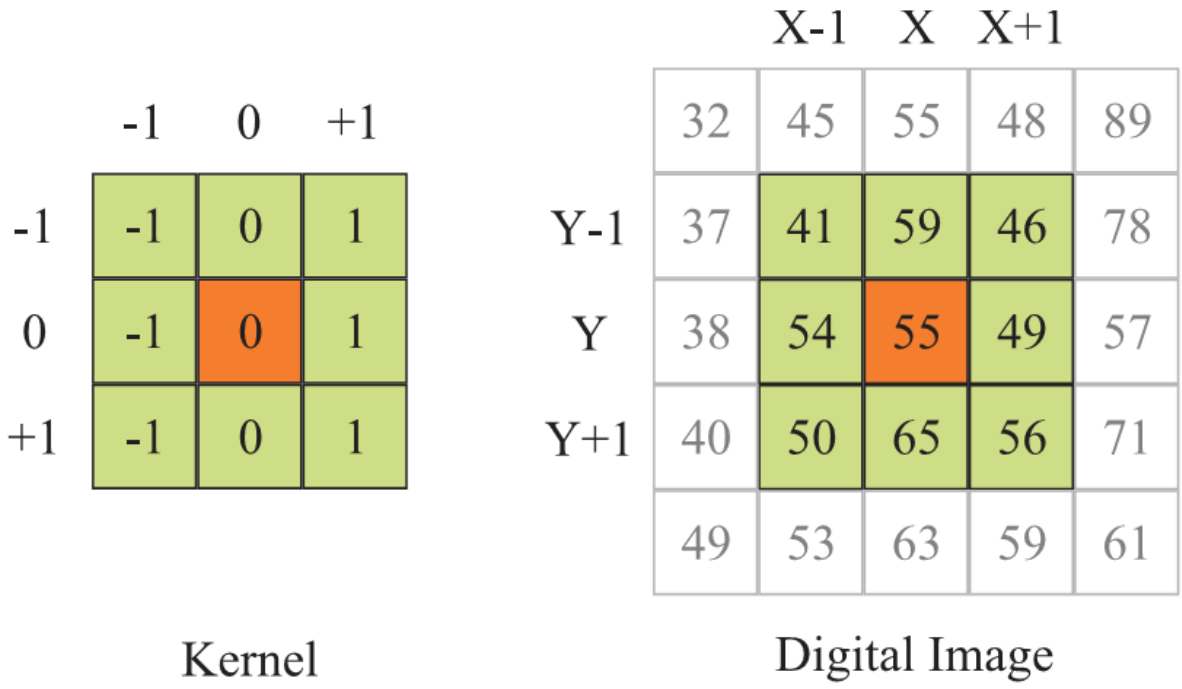


Figure 6.2. A 3×3 kernel is centered over sample $I(x, y)$.

An Illustration

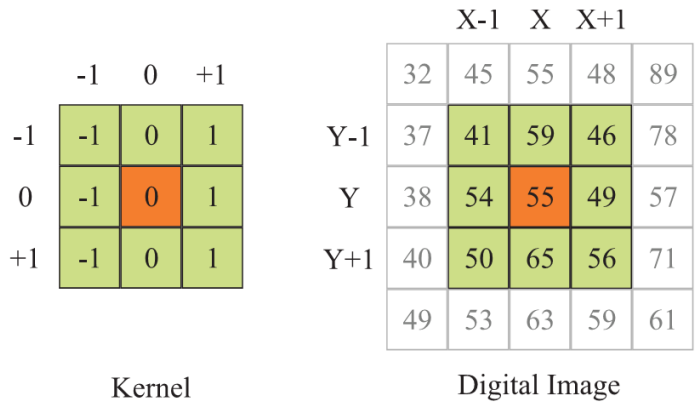


Figure 6.2. A 3×3 kernel is centered over sample $I(x, y)$.

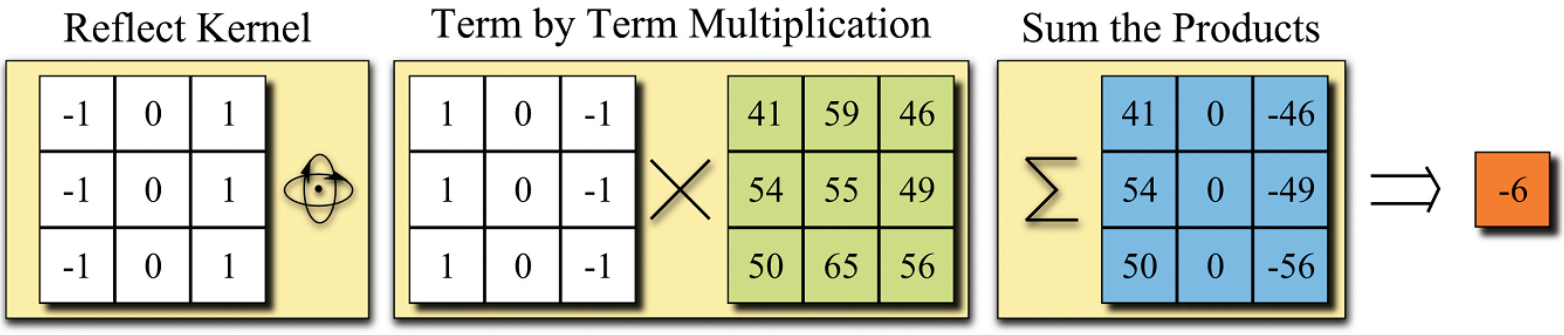
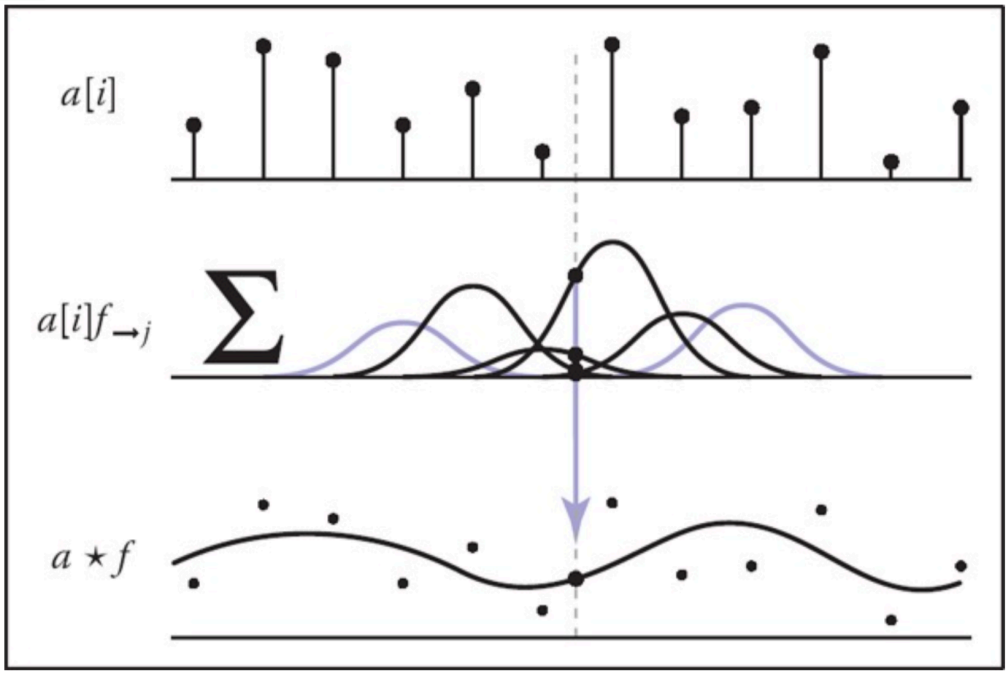


Figure 6.3. Convolution steps.

Convolution Can Also Convert from Discrete to Continuous

- Discrete signal a
- Continuous filter f
- Output a*f defined on positions x as opposed to discrete pixels i



$$(a \star f)(x) = \sum_{i=\lceil x-r \rceil}^{\lfloor x+r \rfloor} a[i] f(x-i)$$

Back to Image Rescaling

100x100 image

B

g

B

g

Filtering helps to reconstruct the signal better when rescaling

Inverse Rescaling

Reconstructed w/ Discrete-to-Continuous

Discrete-Continuous Image Rescaling Code

```
//scale factor
let k = 4;

//create an output greyscale image that is both
//k times as wide and k times as tall
Uint8Array output = new Uint8Array((k*W)*(k*H));

//Loop over each output pixel instead.
for (let row = 0, row < k*H; row++) {
   for (let col = 0; col < k*W; col++) {
      let x = col/k;
      let y = row/k;
      let index = row*k*W + col;
      output[index] = reconstruct(input,x,y);
   }
}
```

Types of Filters: Smoothing

Smoothing Spatial Filters

- Any weighted filter with positive values will smooth in some way, examples:

$$\frac{1}{9}\times\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \qquad \frac{1}{16}\times\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

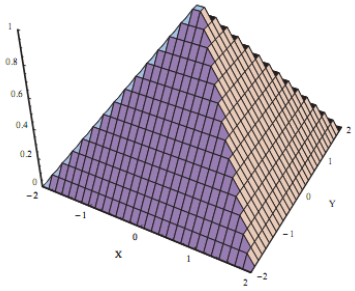
- Normally, we use integers in the filter, and then divide by the sum (computationally more efficient)

- These are also called **blurring** or **low-pass** filters

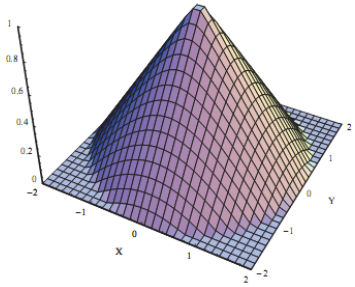
Smoothing Kernels

$$f(x,y) = -\alpha \cdot \max(|x|,|y|)$$

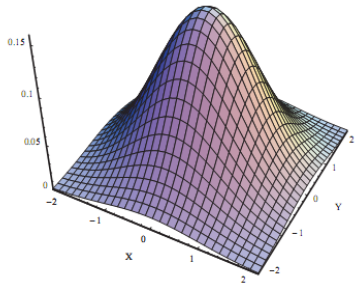
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$



(a) Pyramid.



(b) Cone.



(c) Gaussian.

$$f(x,y) = -\alpha \cdot \sqrt{x^2+y^2}$$

1	2	3	2	1
2	4	6	4	2
3	6	9	6	3
2	4	6	4	2
1	2	3	2	1

(a) Pyramid.

0	0	1	0	0
0	2	2	2	0
1	2	5	2	1
0	2	2	2	0
0	0	1	0	0

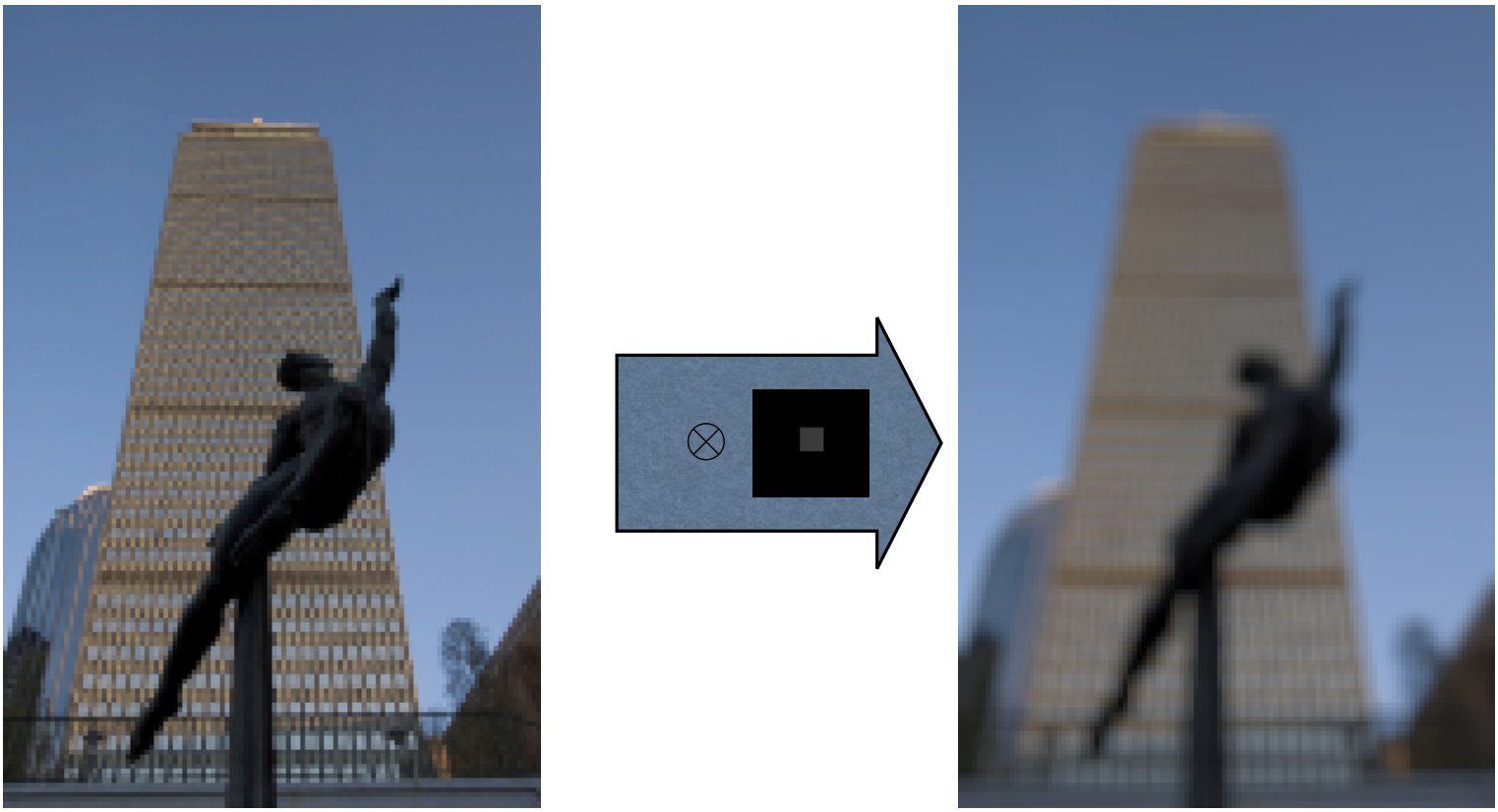
(b) Cone.

1	4	7	4	1
4	16	28	16	4
7	28	49	28	7
4	16	28	16	4
1	4	7	4	1

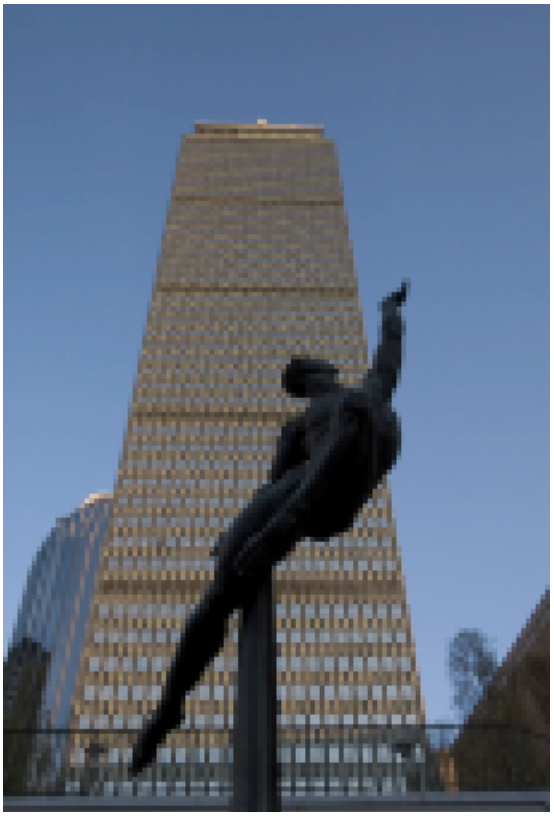
(c) Gaussian.

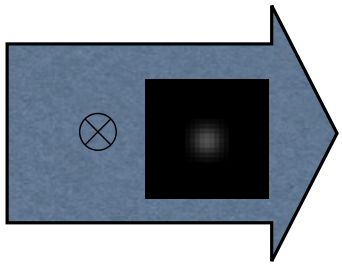
Table 6.1. Discretized kernels.

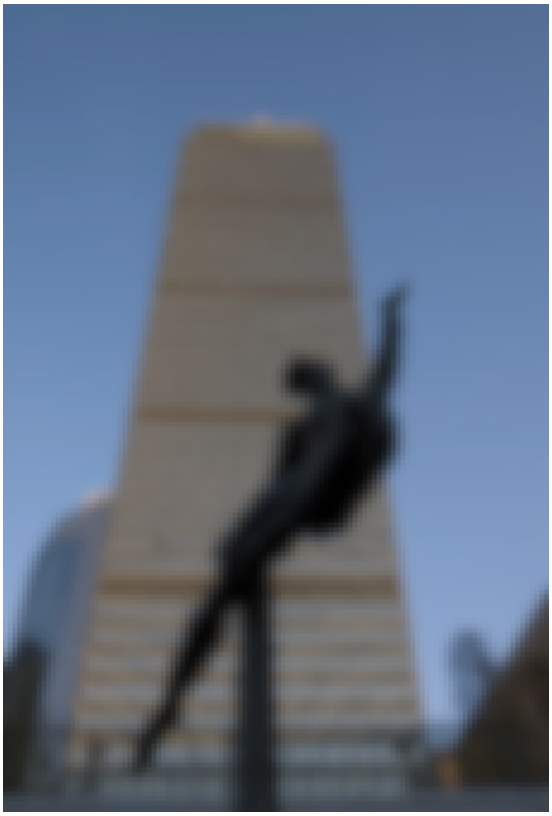
Box Filter



Gaussian Filter



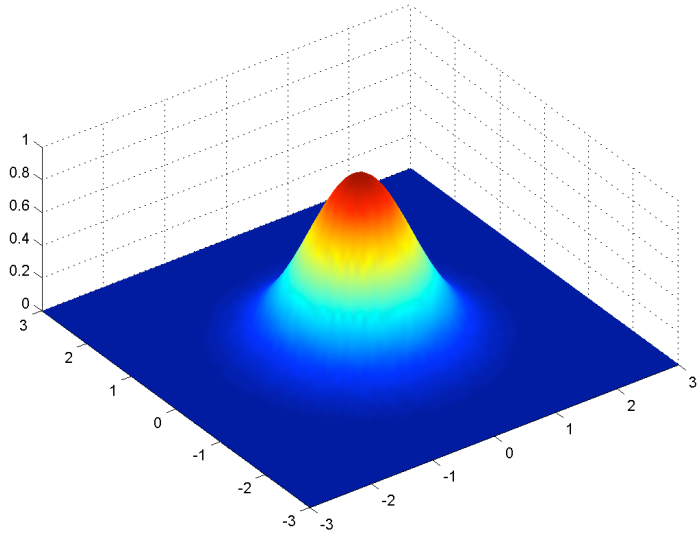




Gaussians

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

- Gaussian kernel is parameterized on the standard deviation σ
- Large σ's reduce the center peak and spread the information across a larger area
- Smaller σ's create a thinner and taller peak
- Gaussians are smooth everywhere.
- Gaussians have infinite **support**
  - >0 everywhere
- But often truncate to 2σ or 3σ



http://en.wikipedia.org/wiki/Gaussian_function

Smoothing Comparison

(a) Source image.

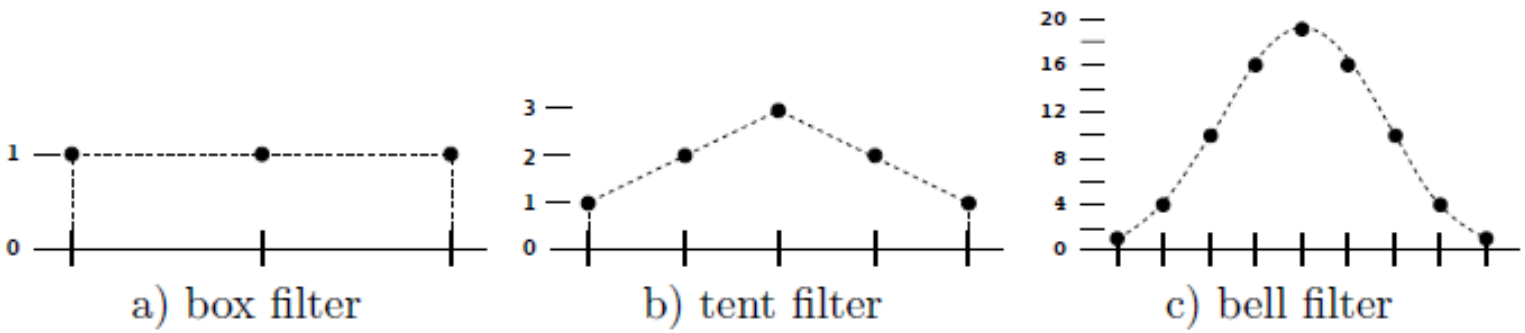
(b) 17 × 17 Box.

(c) 17 × 17 Gaussian.

Figure 6.10. Smoothing examples.

Smoothing the Smoothing Filters

- Box * Box = Tent (Pyramid)
- Tent * Tent = Bell



a) box filter b) tent filter c) bell filter

Types of Filters: Sharpening

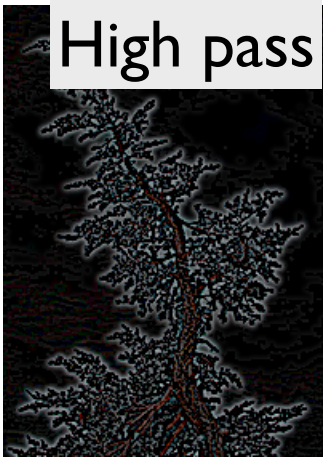
Sharpening (Idea)

-

=

High pass

+k*



=

Sharpened image

Sharpening is a Convolution

- This procedure can then expressed as a single kernel
- Assume that $I = I*d$ and $I_{low} = I*f_{g,\sigma}$.
  - d is the discrete identify function (kernel with 1 in center, 0 elsewhere)
  - $f_{g,\sigma}$ is a smoothing filter (e.g. Gaussian of width σ).
- This leads to:

$$
\begin{aligned}
I_{\text{sharp}} &= (1+\alpha)I - \alpha\big(I \bigstar f_{g,\sigma}\big) \\
&= I \bigstar \big((1+\alpha)d - \alpha f_{g,\sigma}\big) \\
&= I \bigstar f_{\text{sharp}}(\sigma,\alpha),
\end{aligned}
$$

Sharpening is a Convolution

$$
\begin{aligned}
I_{\text{sharp}} &= (1+\alpha)I - \alpha\big(I \;\bigstar\; f_{g,\sigma}\big) \\
&= I \;\bigstar\; \big((1+\alpha)d - \alpha f_{g,\sigma}\big) \\
&= I \;\bigstar\; f_{\text{sharp}}(\sigma,\alpha),
\end{aligned}
$$

Note: could also define d as $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

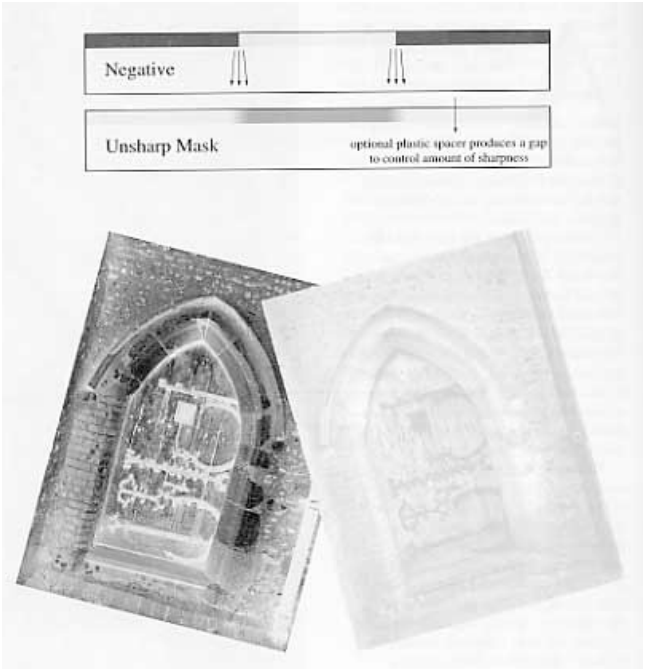
$$
\boldsymbol{d} = \frac{1}{9} \times \begin{bmatrix} 0 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 0 \end{bmatrix},
$$

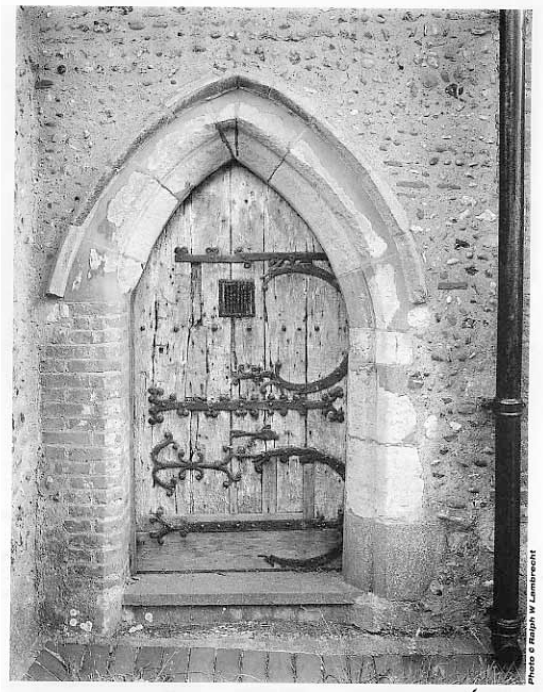
$$
\boldsymbol{f_{g,\sigma}} = \frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix},
$$

$$
\big((1+\alpha)d - \alpha f_{g,\sigma}\big) = \frac{1}{9} \times \begin{bmatrix} -\alpha & -\alpha & -\alpha \\ -\alpha & (9+8\alpha) & -\alpha \\ -\alpha & -\alpha & -\alpha \end{bmatrix}
$$

Unsharp Masks

- Sharpening is often called “unsharp mask” because photographers used to sandwich a negative with a blurry positive film in order to sharpen

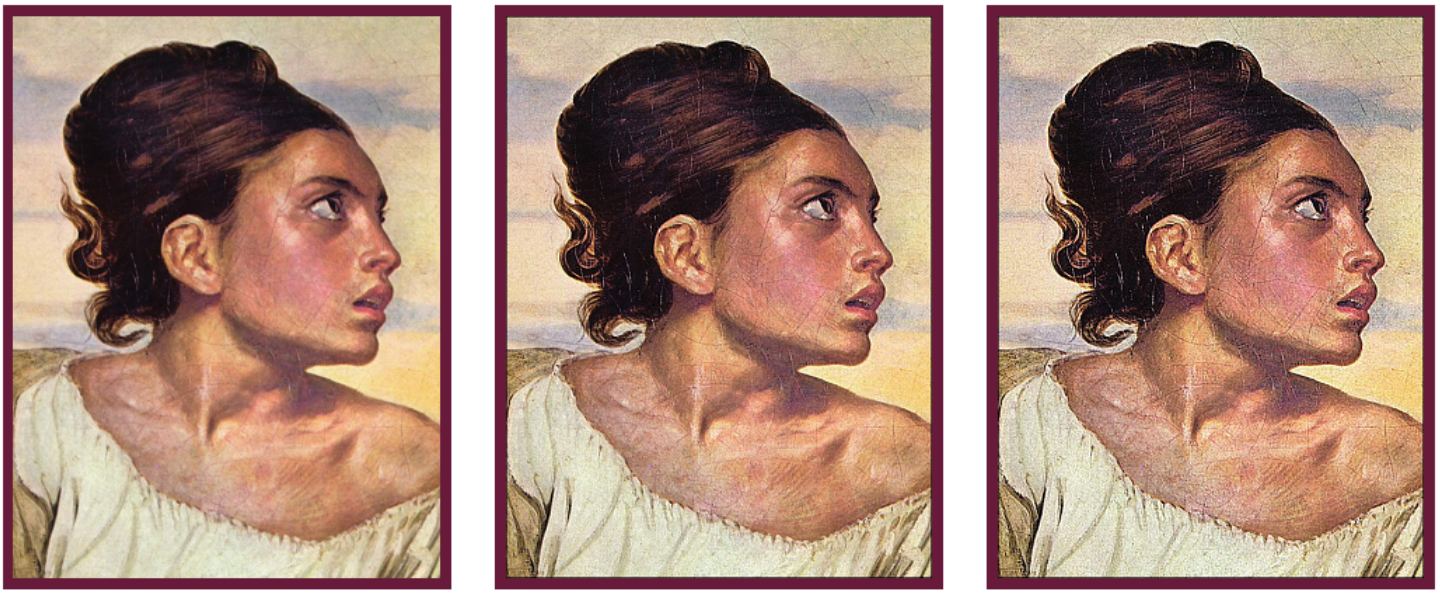




http://www.tech-diy.com/UnsharpMasks.htm

Edge Enhancement

- The parameter α controls how much of the source image is passed through to the sharpened image.



(a) Source image. (b) $\alpha = .5$. (c) $\alpha = 2.0$.

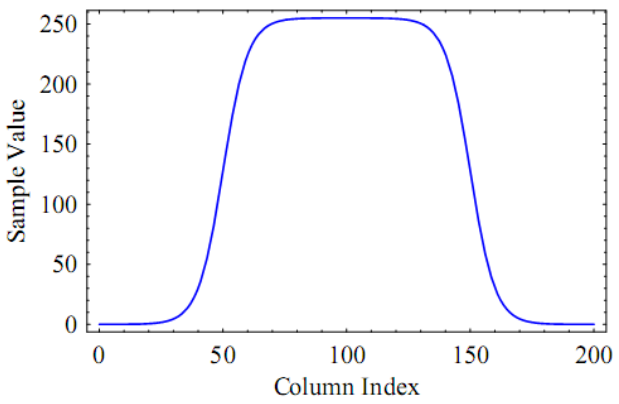
Figure 6.20. Image sharpening.

Defining Edges

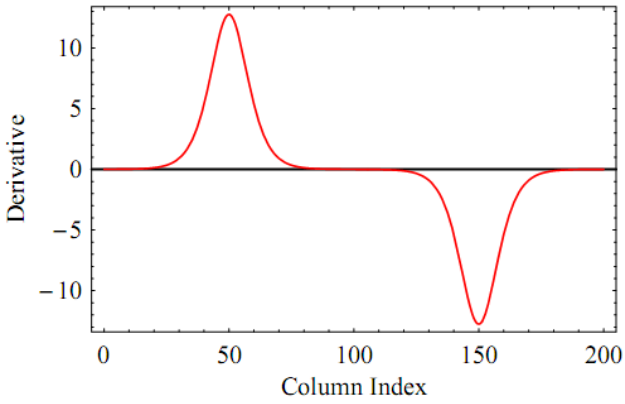
- Sharpening uses negative weights to enhance regions where the image is changing rapidly
    - These rapid transitions between light and dark regions are called **edges**
- Smoothing reduces the strength of edges, sharpening strengthens them.
    - Also called **high-pass** filters
- Idea: smoothing filters are weighted averages, or integrals. Sharpening filters are weighted differences, or derivatives!

Edges

(a)



(b)



(c)

Figure 6.11. (a) A grayscale image with two edges, (b) row profile, and (c) first derivative.

(Review?) Derivatives via Finite Differences

- We can approximate the derivative with a kernel w:

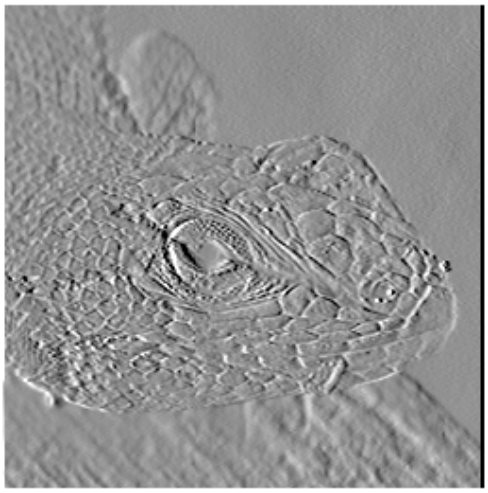
$$\frac{\partial f}{\partial x} \approx \frac{1}{2h}\left(f(x+1,y) - f(x-1,y)\right)$$

$$\frac{\partial f}{\partial x} \approx w_{dx} \circ f \qquad w_{dx} = \begin{array}{|c|c|c|} \hline -\frac{1}{2} & 0 & \frac{1}{2} \\ \hline \end{array}$$

$$\frac{\partial f}{\partial y} \approx w_{dy} \circ f \qquad w_{dy} = \begin{array}{|c|} \hline -\frac{1}{2} \\ \hline 0 \\ \hline \frac{1}{2} \\ \hline \end{array}$$

Taking Derivatives with Convolution

$$\begin{matrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{matrix}$$



$$\begin{matrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$$

Gradients with Finite Differences

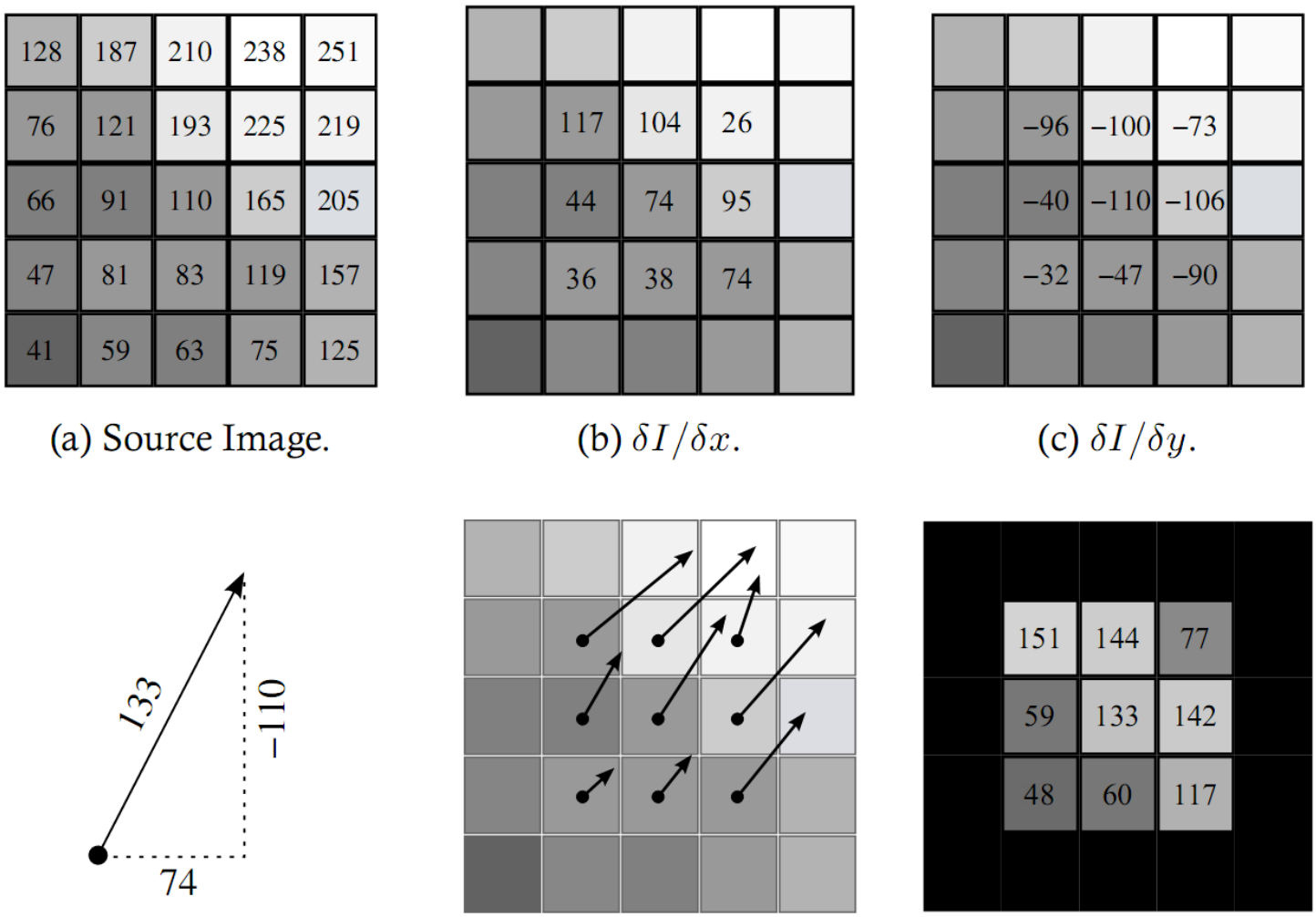
- These partial derivatives approximate the image gradient, ∇I.
- Gradients are the unique direction where the image is changing the most rapidly, like a slope in high dimensions
- We can separate them into components kernels G_x, G_y. $\nabla I = (G_x, G_y)$

$$\nabla I(x,y) = \begin{pmatrix} \delta I(x,y)/\delta x \\ \delta I(x,y)/\delta y \end{pmatrix}.$$

$$G_x = [1, 0, -1] \qquad G_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix};$$

$$\nabla I = \begin{pmatrix} \delta I/\delta x \\ \delta I/\delta y \end{pmatrix} \simeq \begin{pmatrix} I \otimes G_x \\ I \otimes G_y \end{pmatrix}.$$

Figure 6.12. Image gradient (partial).

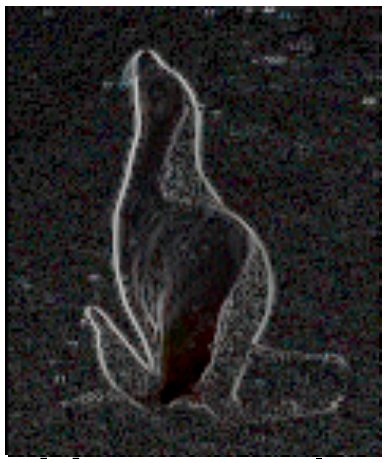


(a) Source Image. (b) $\delta I/\delta x$. (c) $\delta I/\delta y$.

(d) Center sample gradient. (e) Gradient. (f) Magnitude of gradient.

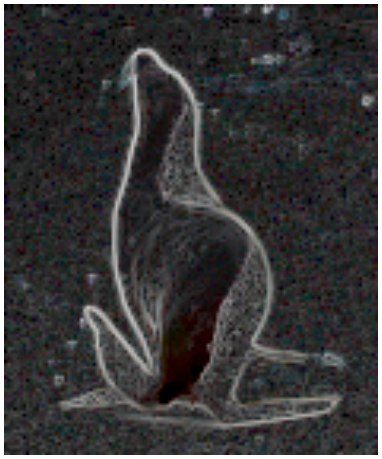
Figure 6.14. Numeric example of an image gradient.

Gradients G_x, G_y



$|G_x|$

$|G_y|$



$|G|$

$$|G| = \sqrt{(G_x^2 + G_y^2)}$$

Second Derivatives (Sharpening, almost)

- Partial derivatives in x and y lead to two kernels:

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

and, similarly, in the y-direction we have

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

0	1	0
1	−4	1
0	1	0

1	1	1
1	−8	1
1	1	1

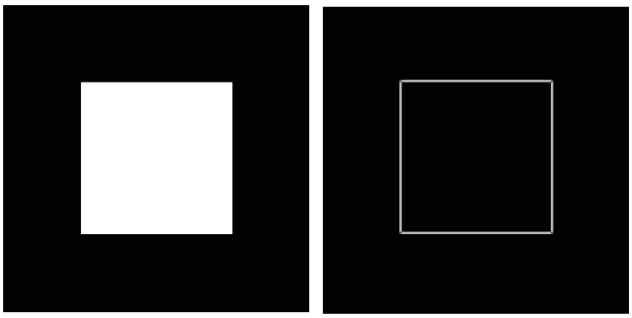
Compare with Sharpening filter: unbalanced counts!

$$\begin{bmatrix} -\alpha & -\alpha & -\alpha \\ -\alpha & (9+8\alpha) & -\alpha \\ -\alpha & -\alpha & -\alpha \end{bmatrix}$$

1	1	1
1	-9	1
1	1	1

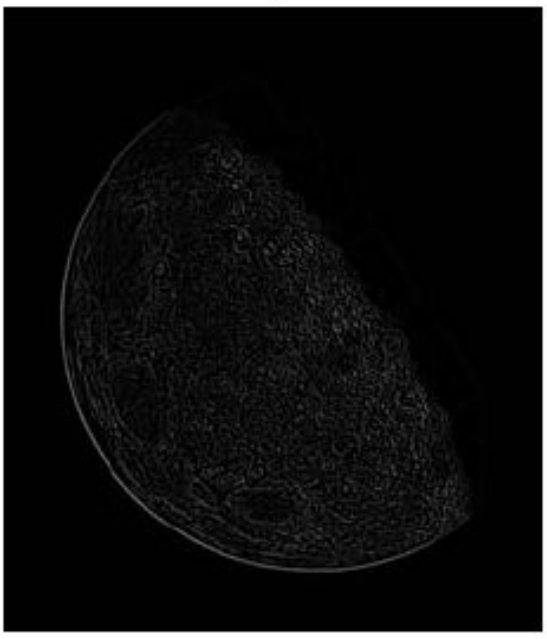
Examples

a)



b)

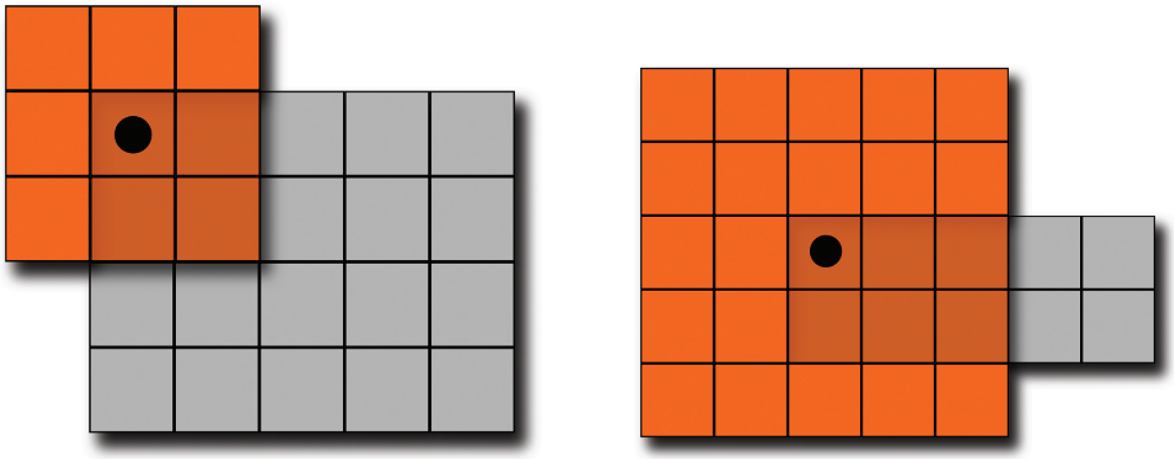
0	1	0
1	−4	1
0	1	0



Boundaries

Handling Image Boundaries

- What should be done if the kernel falls off of the boundary of the source image as shown in the illustrations below?



(a) Kernel at $I(0,0)$. (b) Kernel larger than the source.

Figure 6.4. Illustration of the edge handling problem.

Handling Image Boundaries

- When pixels are near the edge of the image, neighborhoods become tricky to define

- Choices:

    1. Shrink the output image (ignore pixels near the boundary)

    2. Expanding the input image (padding to create values near the boundary which are “meaningful”)

    3. Shrink the kernel (skip values that are outside the boundary, and reweigh accordingly)

Boundary Padding

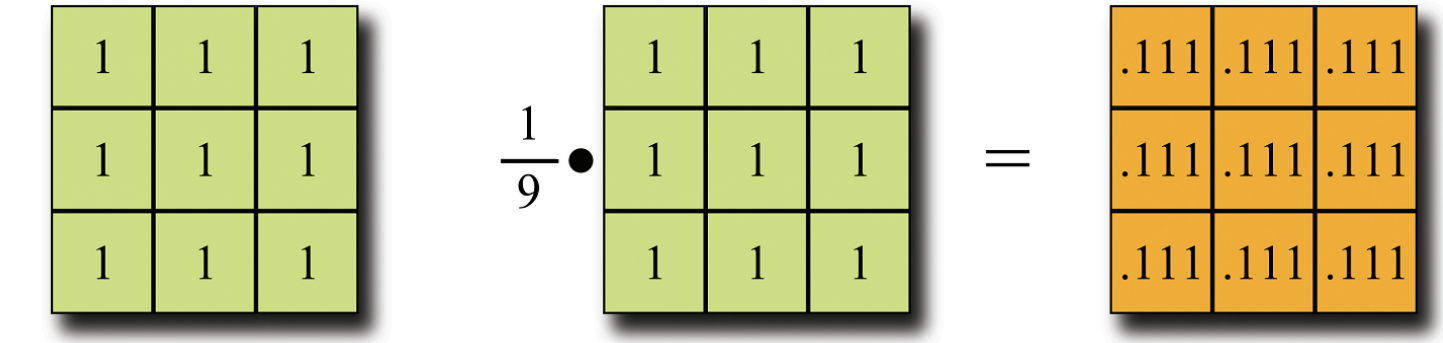
- When one pads, they pretend the image is large and either produce a constant (e.g. zero), or use circular / reflected indexing to tile the image:

(a) (b) (c)

Figure 6.5. (a) Zero padding, (b) circular indexing, and (c) reflected indexing.

Kernel Rescaling

- Important to always rescale the kernel by making the coefficients sum to 1.
  - The effect of the kernel is unchanged
  - Rescaling the kernel is equivalent to rescaling the convolved image.



(a) Non-normalized. (b) Normalization. (c) Normalized.

Figure 6.6. Normalizing a kernel.

Kernel Separability

Convolution Complexity

- What is the computational complexity of the “brute-force” approach to convolving a WxH image?

Convolution Complexity

- What is the computational complexity of the “brute-force” approach to convolving a WxH image?
    - Computing each destination sample requires M*N multiplications and additions

Convolution Complexity

- What is the computational complexity of the “brute-force” approach to convolving a WxH image?
  - Computing each destination sample requires M*N multiplications and additions
  - There are W*H samples.

Convolution Complexity

- What is the computational complexity of the “brute-force” approach to convolving a WxH image?
  - Computing each destination sample requires M*N multiplications and additions
  - There are W*H samples.
  - Convolution is on the order of W*H*M*N operations.

Convolution Complexity

- What is the computational complexity of the “brute-force” approach to convolving a WxH image?
  - Computing each destination sample requires M*N multiplications and additions
  - There are W*H samples.
  - Convolution is on the order of W*H*M*N operations.
- This is SLOW!

Convolution Complexity

- What is the computational complexity of the “brute-force” approach to convolving a WxH image?
  - Computing each destination sample requires M*N multiplications and additions
  - There are W*H samples.
  - Convolution is on the order of W*H*M*N operations.
- This is SLOW!
  - Large kernels are impractical

Convolution Can Often Be Perfectly Separated

- Computational efficiency can be dramatically improved if the kernel is separable.
- A MxN kernel is separable iff there exists an Mx1 column vector A and a 1xN row vector B such that K = AB.
- Example:

$$AB = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

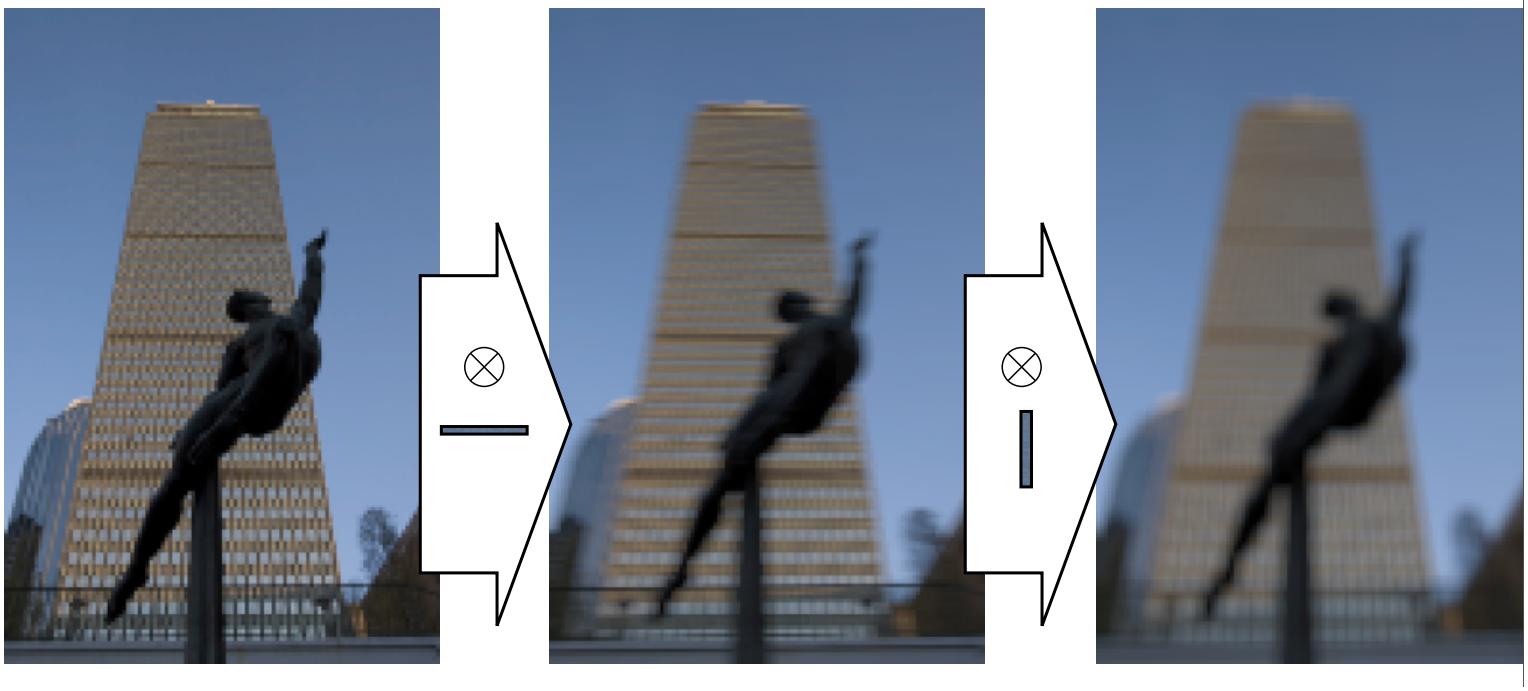
- The benefit derives from realizing that convolution can be performed by consecutive application of convolution of I with A followed by B.

$$I \otimes K = (I \otimes A) \otimes B \quad \text{where} \quad K = AB.$$

- This leads to M+N operations per pixel, or W*H*(M+N) total.

Separable Box Smoothing

- First blur horizontally using g(x)
- Then blur vertically using g(y)



Works for Gaussians too!

Lec06 Required Reading

- FOCG, Ch. 21 (particularly 21.1-21.3)
- See also optional readings (especially grads!)

Reminder: Assignment 02

Assigned: Wednesday, Sept. 11
Written Due: Monday, Sept. 23, 4:59:59 pm
Program Due: Wednesday, Sept. 25, 4:59:59 pm