CS559: Computer Graphics

Lecture 4: Compositing and Resampling

Li Zhang

Spring 2008

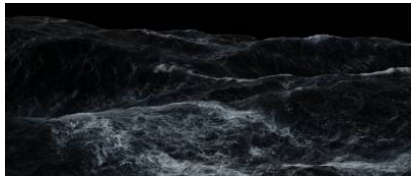
Compositing

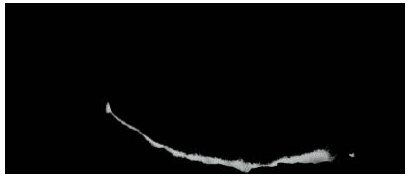
- Compositing combines components from two or more images to make a new image
  - Special effects are easier to control when done in isolation
  - Even many all live-action sequences are more safely shot in different layers

Compositing

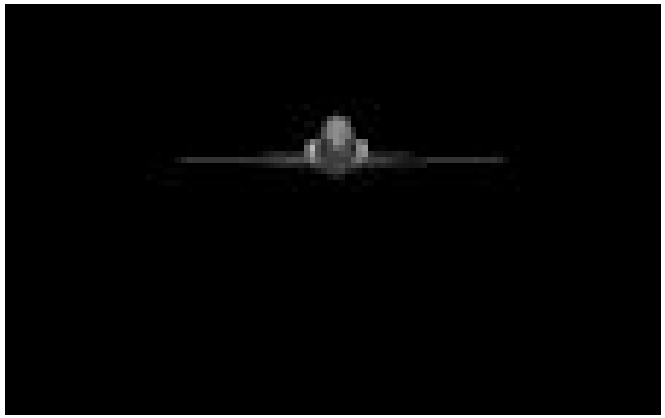
- Compositing combines components from two or more images to make a new image
  - Special effects are easier to control when done in isolation
  - Even many all live-action sequences are more safely shot in different layers

Perfect Storm

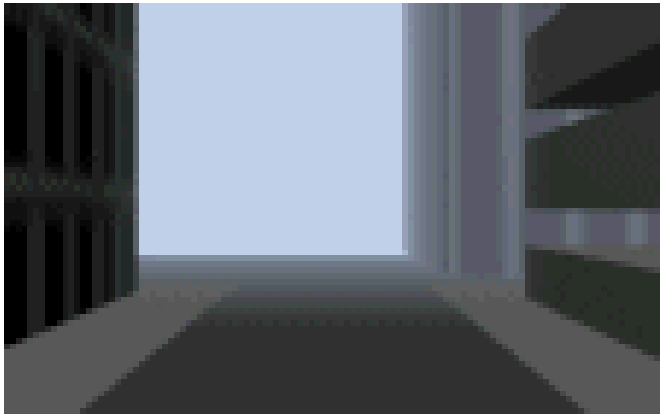




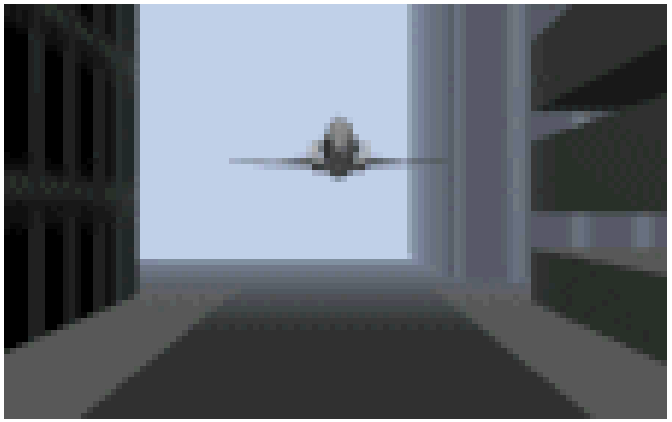
Animated Example



over



=



Mattes

- A *matte* is an image that shows which parts of another image are foreground objects
- Term dates from film editing and cartoon production
- How would I use a matte to insert an object into a background?
- How are mattes usually generated for television?

Working with Mattes

- To insert an object into a background
  - Call the image of the object the source
  - Put the background into the destination
  - For all the source pixels, if the matte is white, copy the pixel, otherwise leave it unchanged
- To generate mattes:
  - Use smart selection tools in Photoshop or similar
    - They outline the object and convert the outline to a matte
  - **Blue Screen:** Photograph/film the object in front of a blue background, then consider all the blue pixels in the image to be the background

Compositing

- Compositing is the term for combining images, one over the other
  - Used to put special effects into live action
    - Or live action into special effects

Alpha

- Basic idea: Encode opacity information in the image
- Add an extra channel, the *alpha* channel, to each image
  - For each pixel, store R, G, B and Alpha
  - alpha = 1 implies full opacity at a pixel
  - alpha = 0 implies completely clear pixels
- Images are now in RGBA format, and typically 32 bits per pixel (8 bits for alpha)
- All images in the project are in this format

Pre-Multiplied Alpha

- Instead of storing (R,G,B,α), store (αR,αG,αB,α)
- The compositing operations in the next several slides are easier with pre-multiplied alpha
- **To display and do color conversions, must extract RGB by dividing out α**
  - α=0 is always black
  - Some loss of precision as α gets small, but generally not a big problem

Basic Compositing Operation

- At each pixel, combine the pixel data from *f* and the pixel data from *g* with the equation:

$$c_f = [\alpha_f R_f, \alpha_f G_f, \alpha_f B_f, \alpha_f]$$

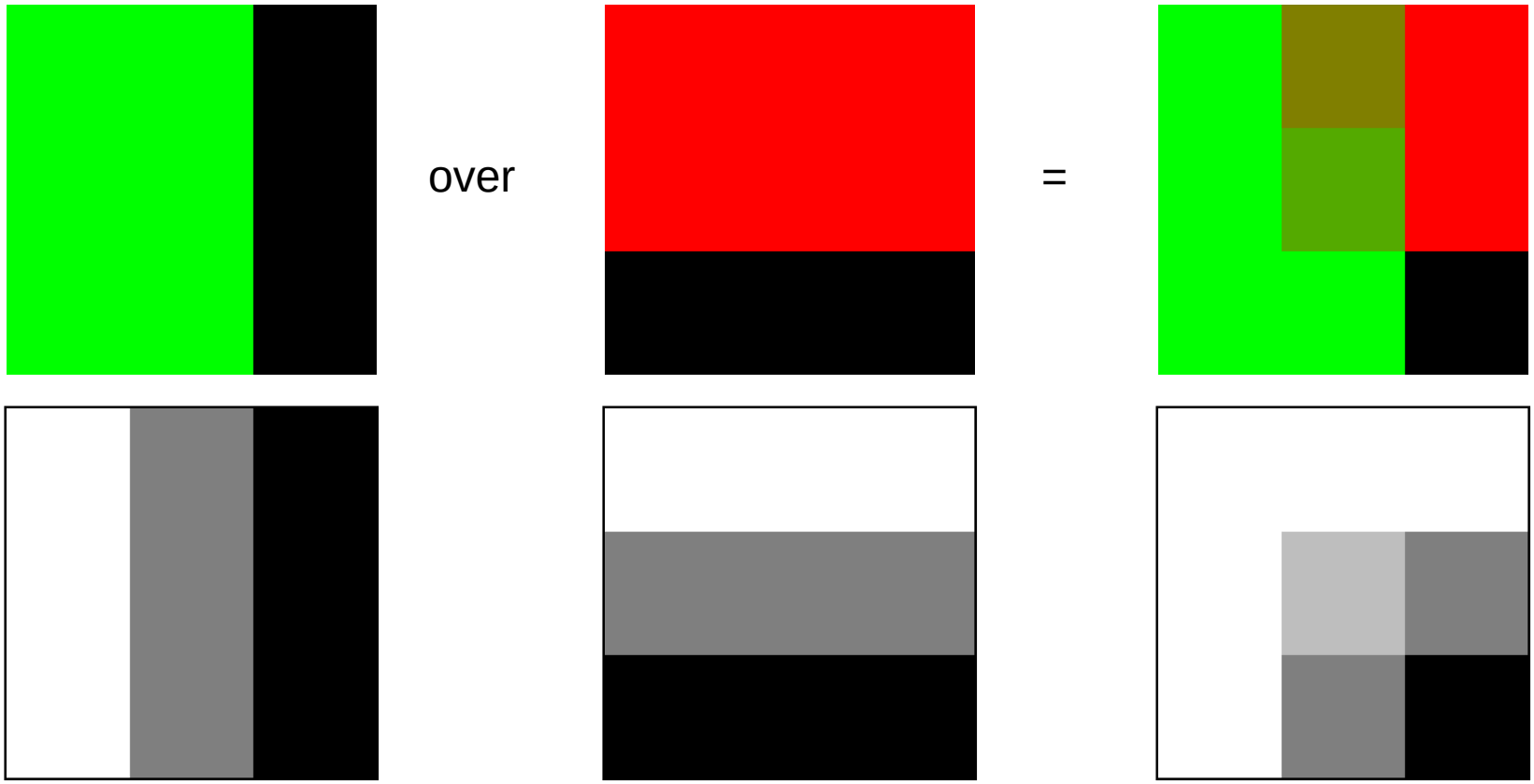
$$c_g = [\alpha_g R_g, \alpha_g G_g, \alpha_g B_g, \alpha_g]$$

$$c_o = 1 \cdot c_f + (1 - \alpha_f) \cdot c_g$$

"Over" Operator

“Over” Operator

- If there’s some *f*, get *f*, otherwise get *g*



$$c_o = 1 \cdot c_f + (1 - \alpha_f) \cdot c_g$$

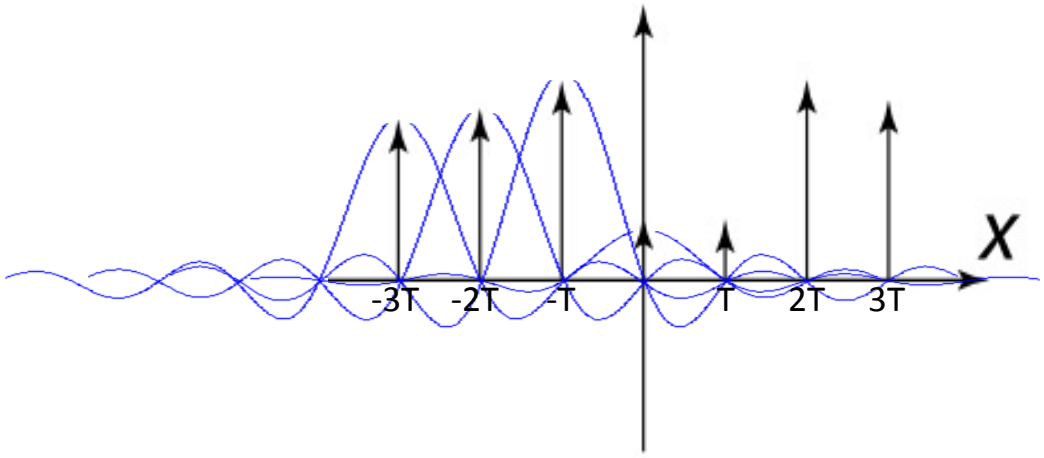
Reconstruction theorem

Let $g[n] = f(nT)$ be the sampling sequence for $f(x)$.

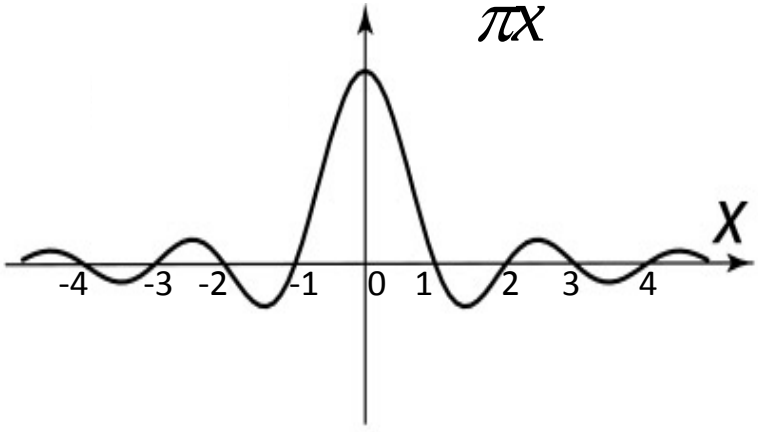
If f(x) has no freq above $1/2$ the sampling freq,

then $f(x) = \sum_{n=-\infty}^{\infty} g[n] \cdot \mathrm{sinc}\left(\frac{x - nT}{T}\right)$

$g[n]$



$$\mathrm{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

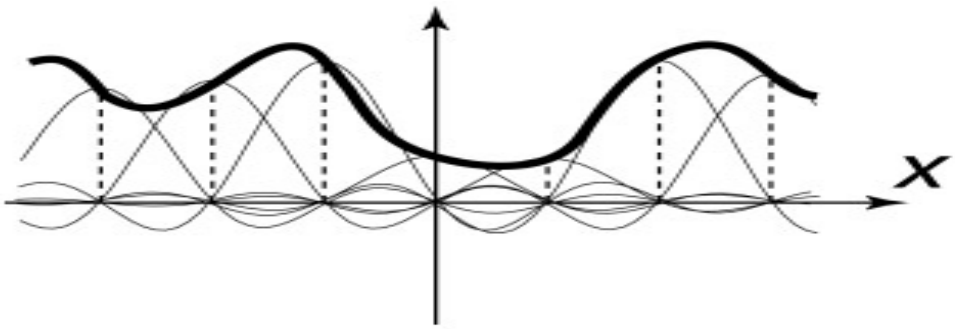


Reconstruction theorem

Let $g[n] = f(nT)$ be the sampling sequence.

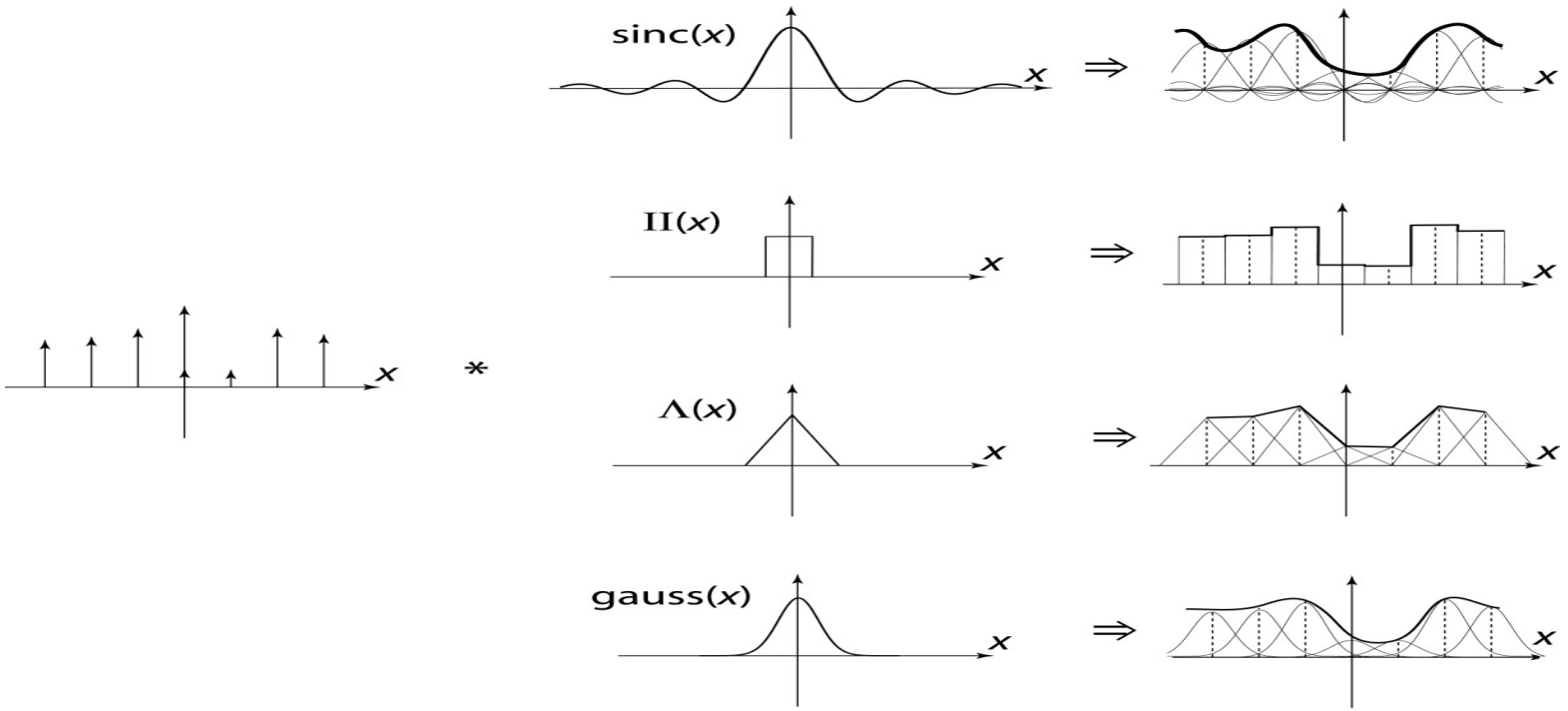
If f(x) has no freq above $1/2$ the sampling freq,

then $f(x) = \sum_{n=-\infty}^{\infty} g[n] \cdot \mathrm{sinc}(\frac{x - nT}{T})$



Reconstruction filters

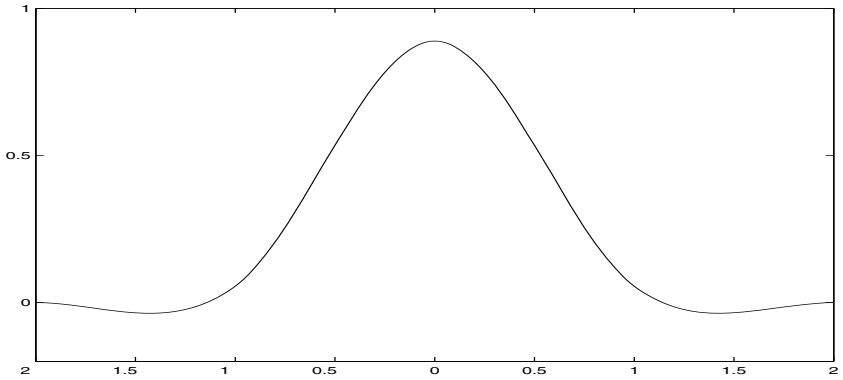
- The sinc filter, while “ideal”, has two drawbacks:
  - It has large support (slow to compute)
  - It introduces ringing in practice
- We can choose from many other filters…



Cubic filters

- Mitchell and Netravali (1988) experimented with cubic filters of the following form:

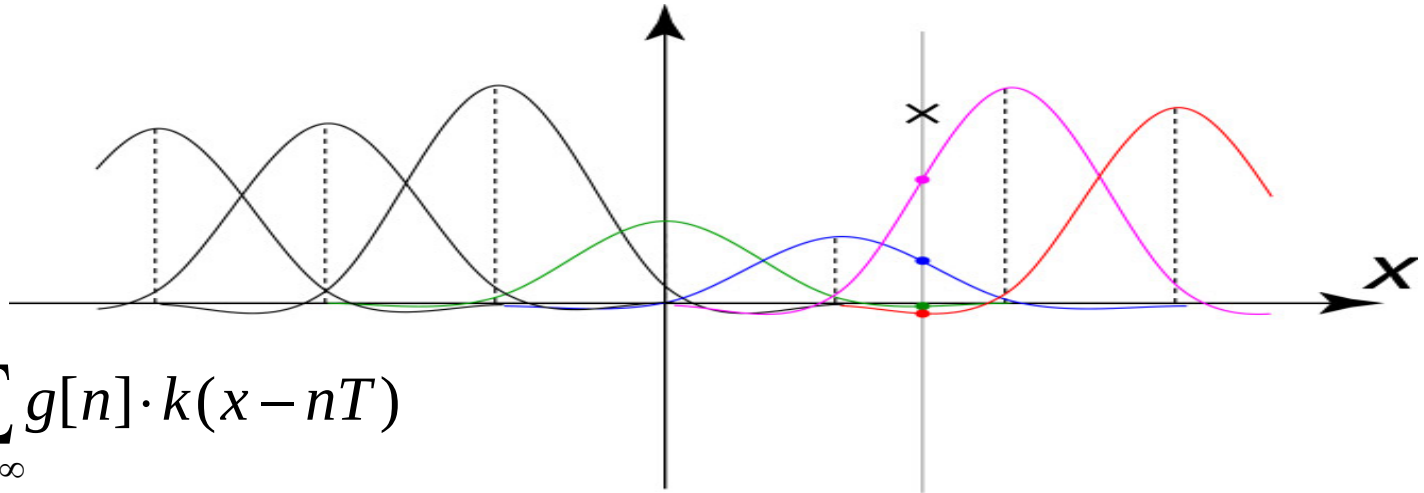
$$r(x) = \frac{1}{6}\begin{cases} (12 - 9B - 6C)|x|^3 + (-18 + 12B + 6C)|x|^2 + (6 - 2B) & |x| < 1 \\ ((-B - 6C)|x|^3 + (6B + 30C)|x|^2 + (-12B - 48C)|x| + (8B + 24C) & 1 \le |x| < 2 \\ 0 & otherwise \end{cases}$$



- The choice of B or C trades off between being too blurry or having too much ringing. B=C=1/3 was their "visually best" choice.
- The resulting reconstruction filter is often called the "Mitchell filter."

Practical upsampling

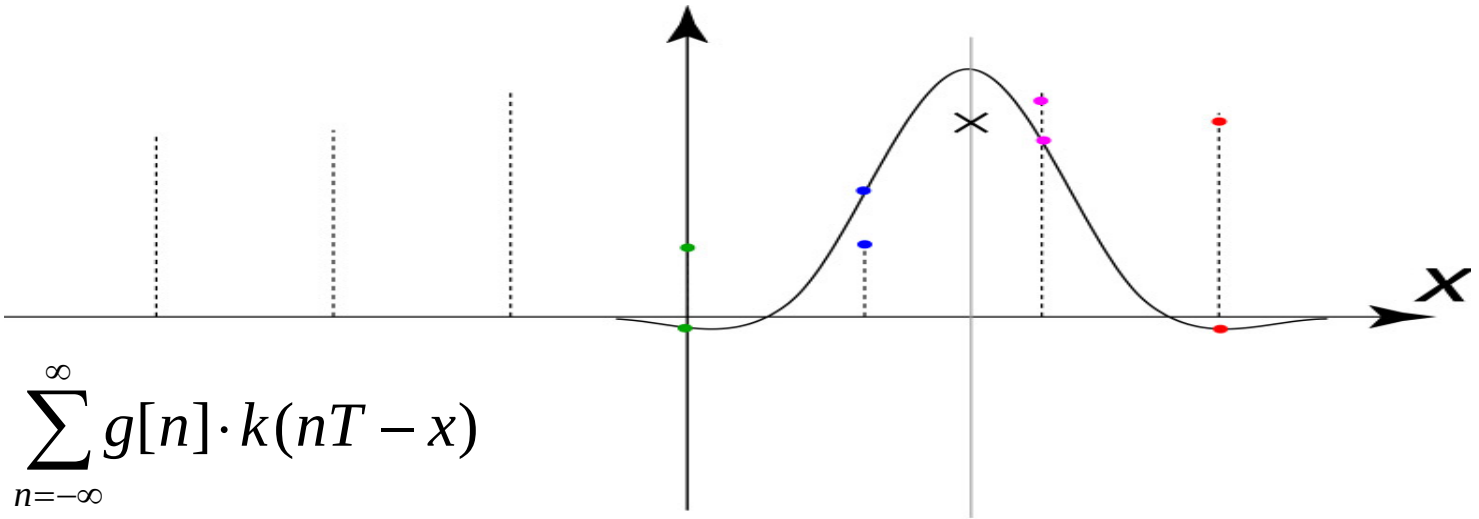
- When resampling a function (e.g., when resizing an image), you do not need to reconstruct the complete continuous function.
- For zooming in on a function, you need only use a reconstruction filter and evaluate as needed for each new sample.
- Here's an example using a cubic filter:



$$f(x) = \sum_{n=-\infty}^{\infty} g[n] \cdot k(x - nT)$$

Practical upsampling

- This can also be viewed as:
  1. putting the reconstruction filter at the desired location
  2. evaluating at the original sample positions
  3. taking products with the sample values themselves
  4. summing it up



$$f(x) = \sum_{n=-\infty}^{\infty} g[n] \cdot k(nT - x)$$

2D Fourier transform

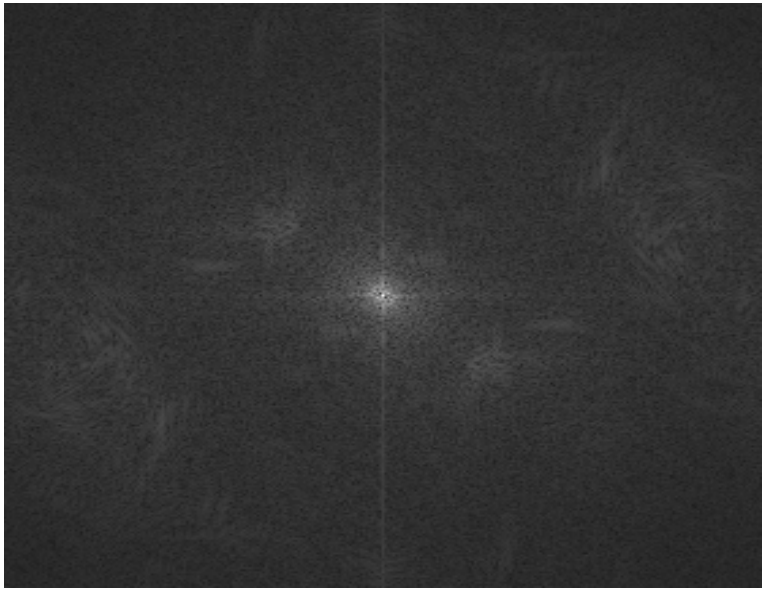
$$F(s_x, s_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-i 2\pi (s_x x + s_y y)} dx\, dy$$

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(s_x, s_y) e^{i 2\pi (s_x x + s_y y)} ds_x\, ds_y$$

Spatial domain

$f(x, y)$

Frequency domain

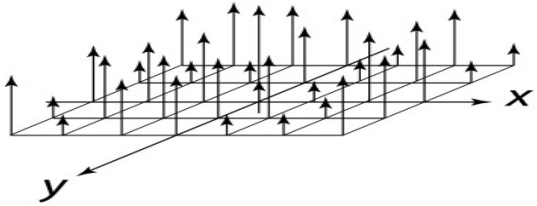


$|F(s_x, s_y)|$

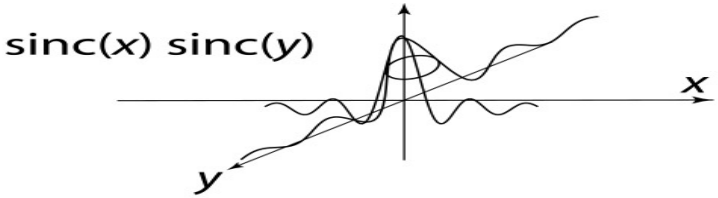
High frequency coefficients are small.

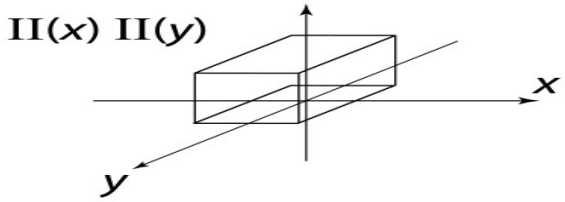
Reconstruction filters in 2D

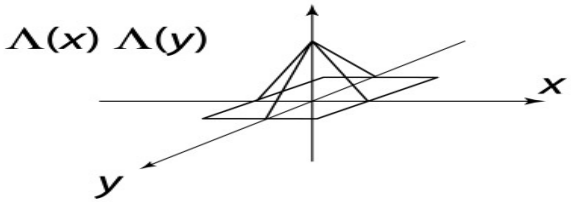
- We can also perform reconstruction in 2D…

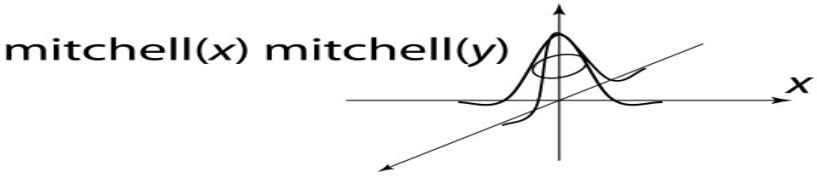


*

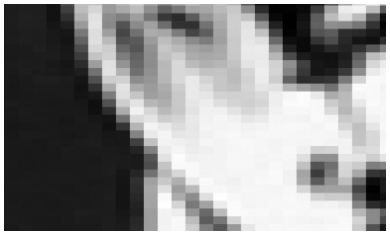








⇒

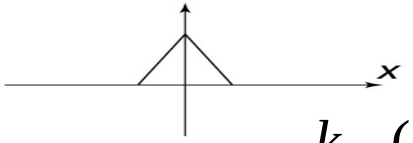


⇒

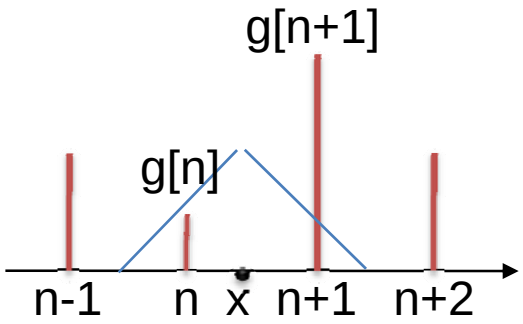
⇒

Reconstruction filters in 2D

1D example:



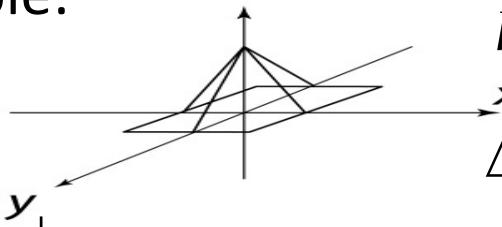
$$k_{1D}(x) = \begin{cases} 1-|x| & |x|<1 \\ 0 & \text{otherwise} \end{cases}$$



$$\Delta x = x - n$$

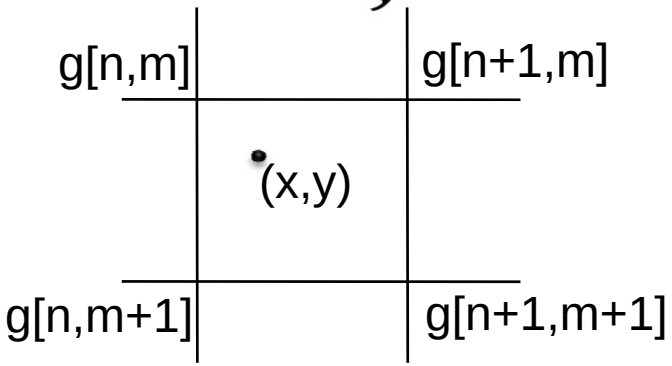
$$f(x) = g[n]\cdot(1-\Delta x) + g[n+1]\cdot\Delta x$$

2D example:



$$k_{2D}(x,y) = \begin{cases} (1-|x|)(1-|y|) & |x|<1, |y|<1 \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta x = x - n,\ \Delta y = y - m$$



$$\begin{aligned} f(x,y) = {} & g[n,m]\cdot(1-\Delta x)\cdot(1-\Delta y) \\ & + g[n+1,m]\cdot\Delta x\cdot(1-\Delta y) \\ & + g[n,m+1]\cdot(1-\Delta x)\cdot\Delta y \\ & + g[n+1,m+1]\cdot\Delta x\cdot\Delta y \end{aligned}$$

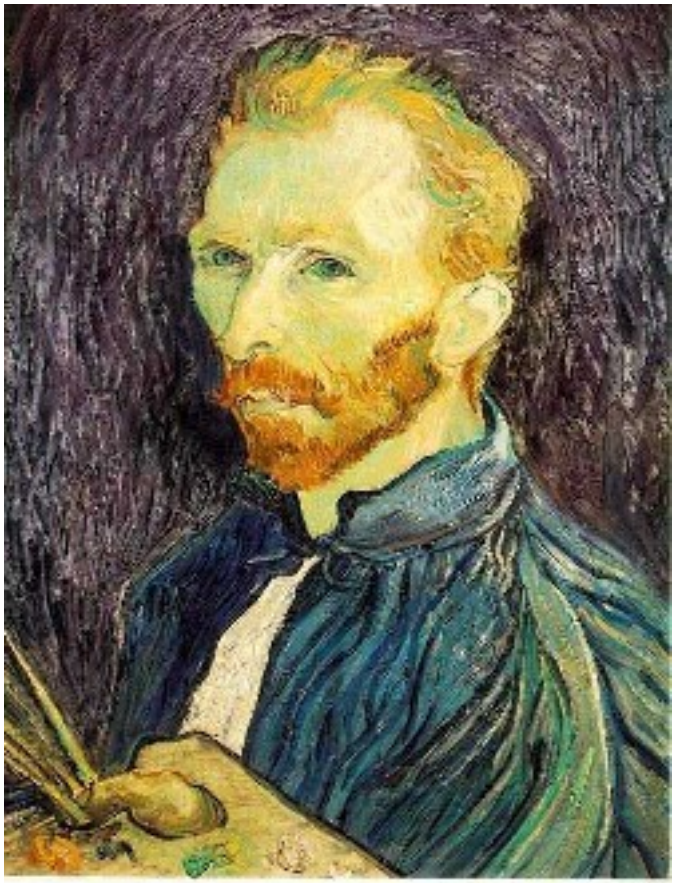
Reconstruction filters in 2D

We've been looking at **separable** filters:

$$k_{2D}(x, y) = k(x)_{1D} k_{1D}(y)$$

How might you use this fact for efficient resampling in 2D?

Image Downsampling



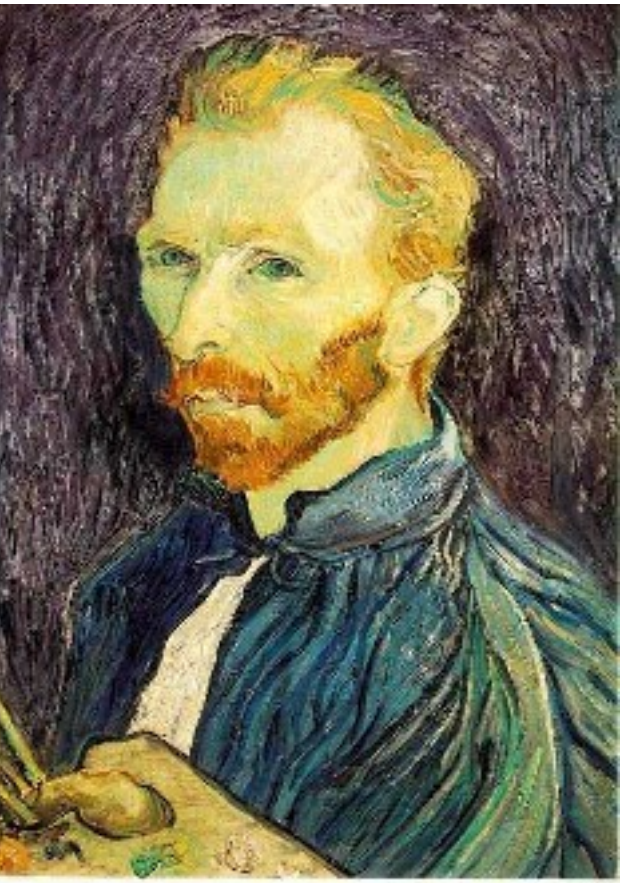
1/4

1/8

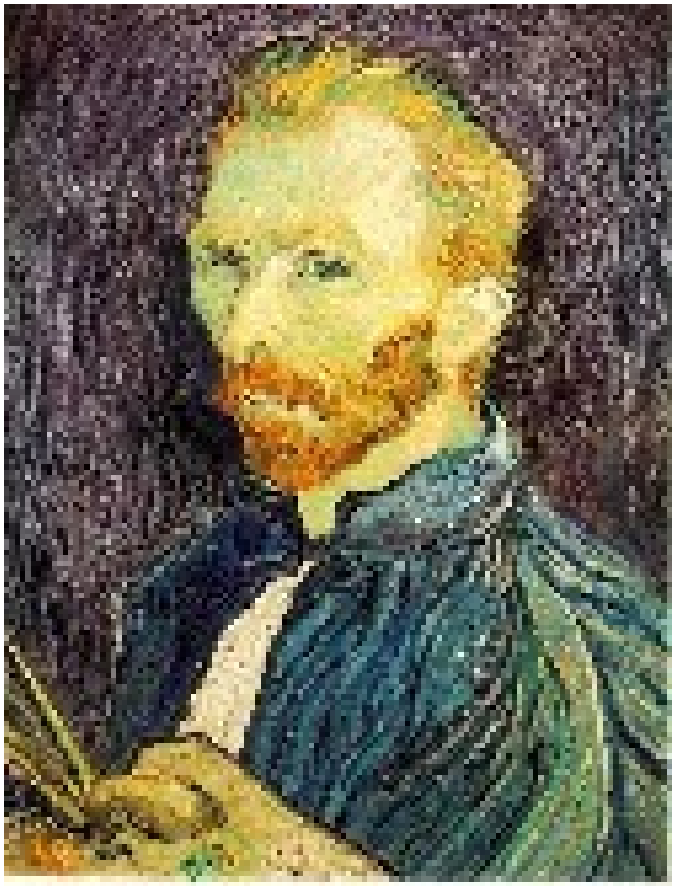
Throw away every other row and column to create a 1/2 size image

- called *image sub-sampling*

Image sub-sampling



1/2



1/4 (2x zoom)

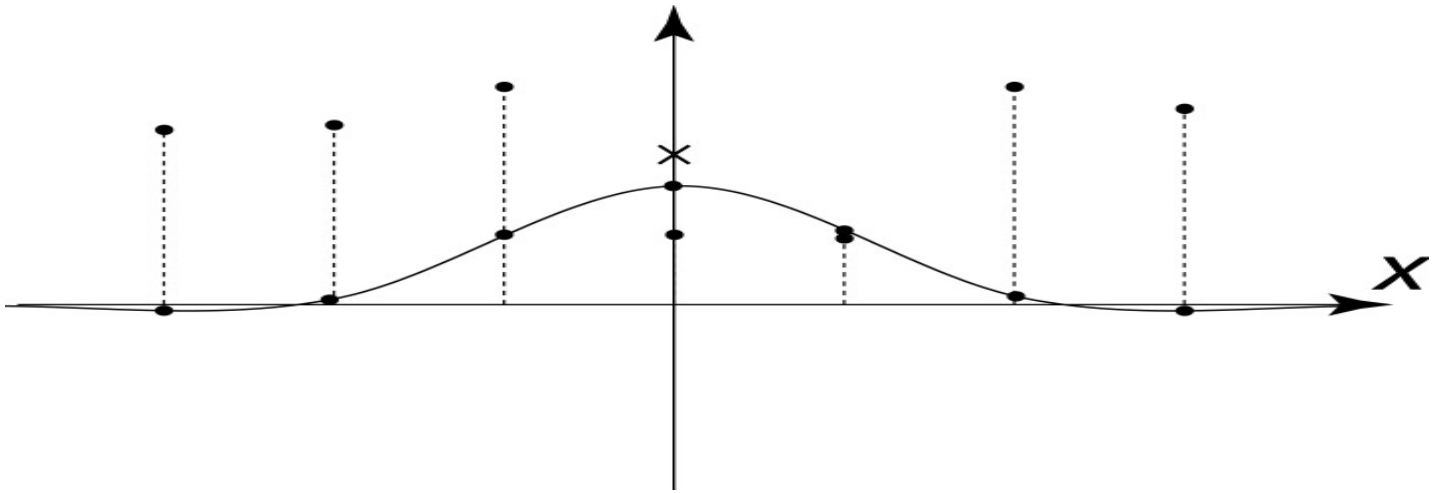
1/8 (4x zoom)

Why does this look so crufty?

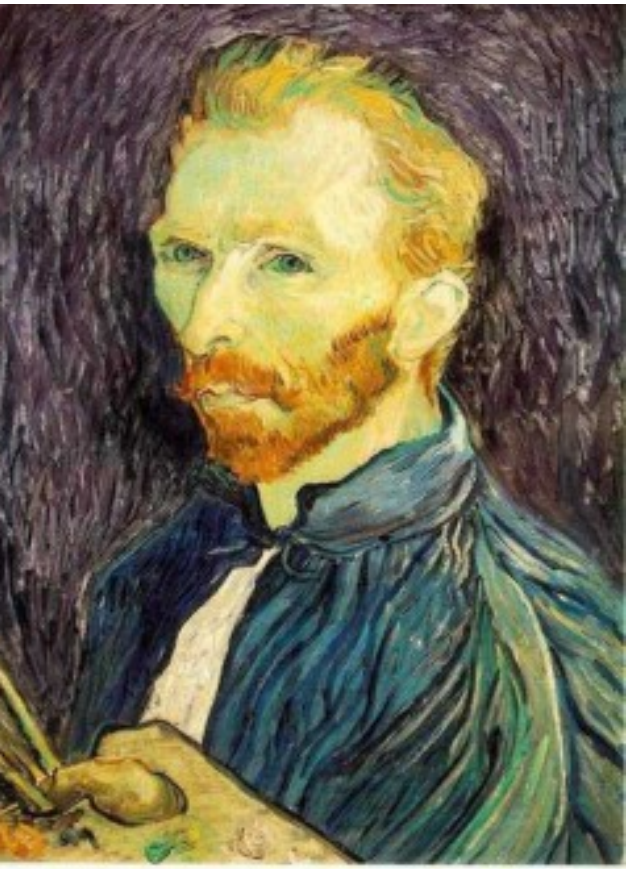
Minimum Sampling requirement is not satisfied – resulting in **Aliasing effect**

Practical downsampling

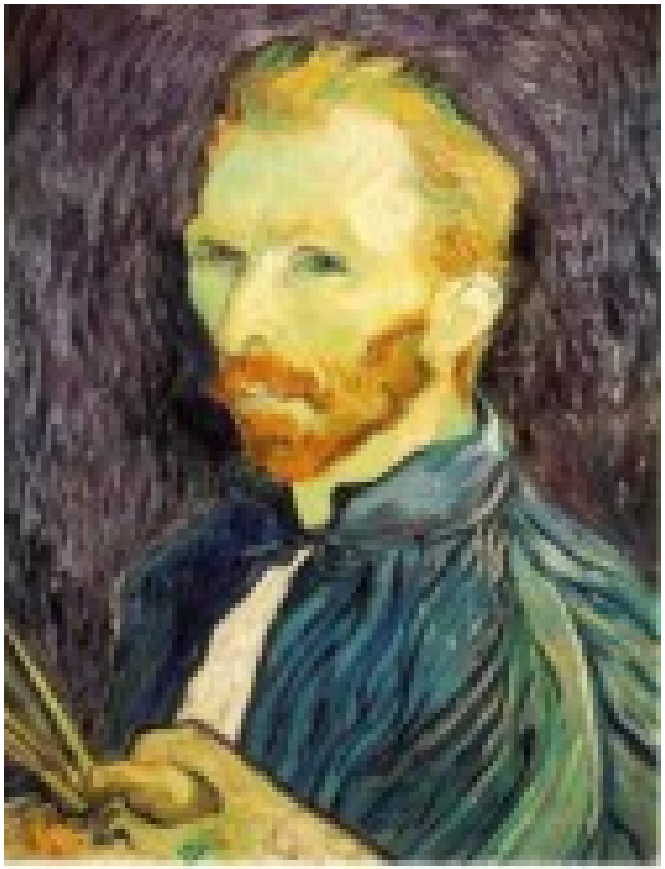
- Downsampling is similar, but filter has larger support and smaller amplitude.
- Operationally:
  - given the downsampling rate, *d*, ratio of new sampling rate to old sampling rate
  1. Choose reconstruction filter
  2. Stretch the filter by 1/*d* and scale it down by *d*
  3. Follow upsampling procedure (previous slides) to compute new values



Subsampling with Gaussian pre-filtering



Gaussian 1/2

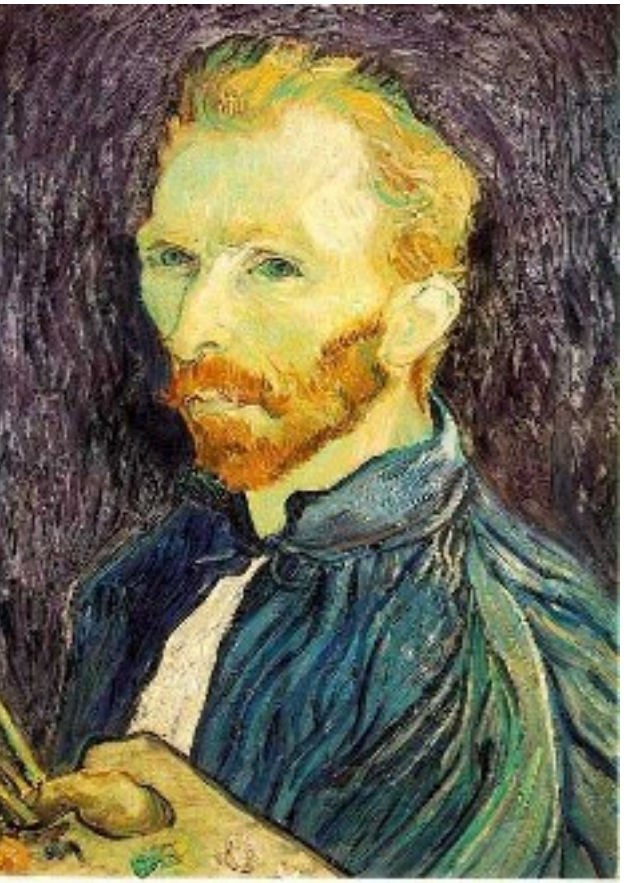


G 1/4

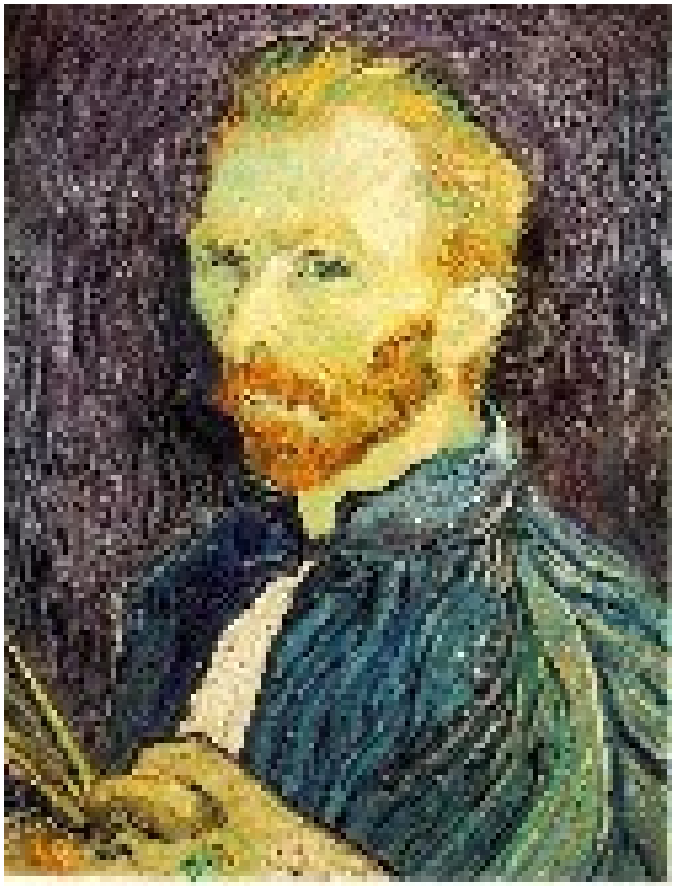
G 1/8

- Solution: filter the image, *then* subsample

Compare with...



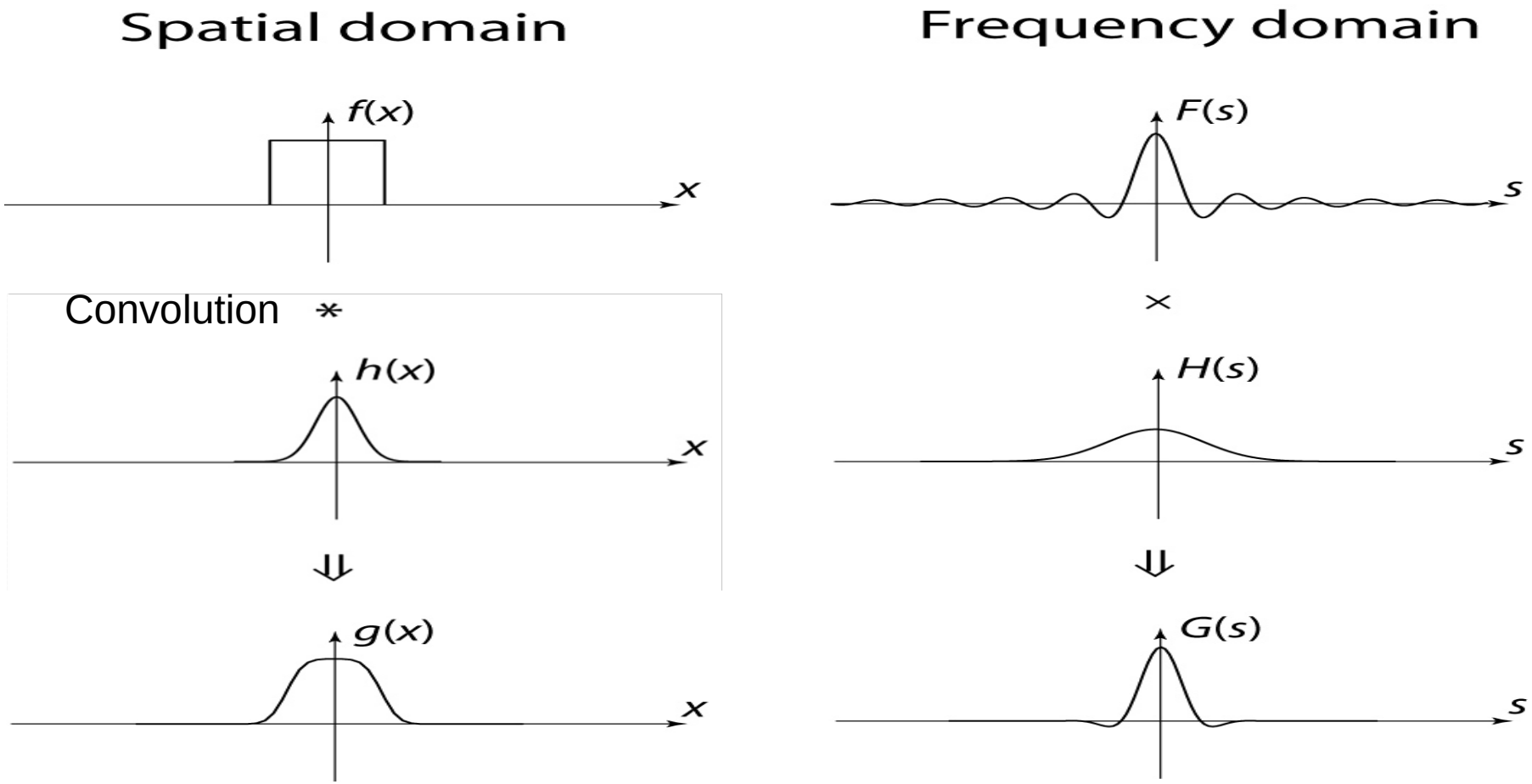
1/2



1/4 (2x zoom)

1/8 (4x zoom)

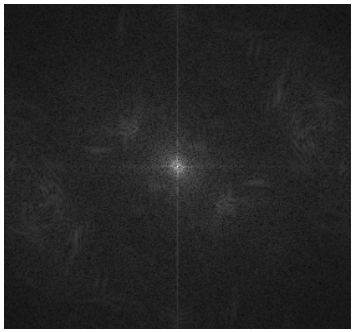
Explanation using Fourier Transform



Explanation using Fourier Transform

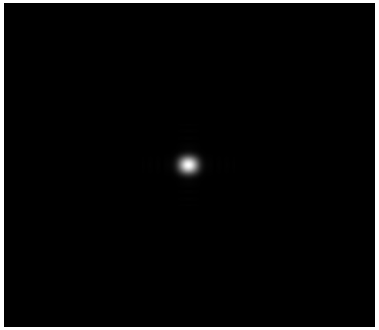
$f(x,y)$

$|F(s_x,s_y)|$

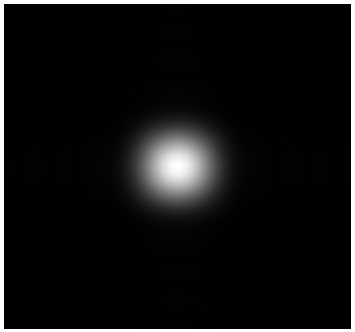


$*$ $\asymp$

$h(x,y)$



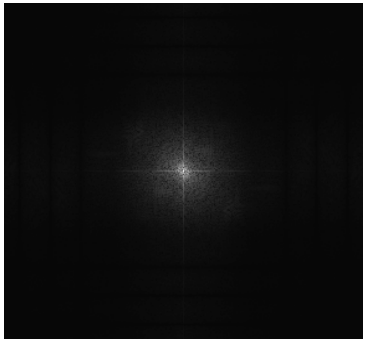
$|H(s_x,s_y)|$



$\Downarrow$ $\Downarrow$

$g(x,y)$

$|G(s_x,s_y)|$

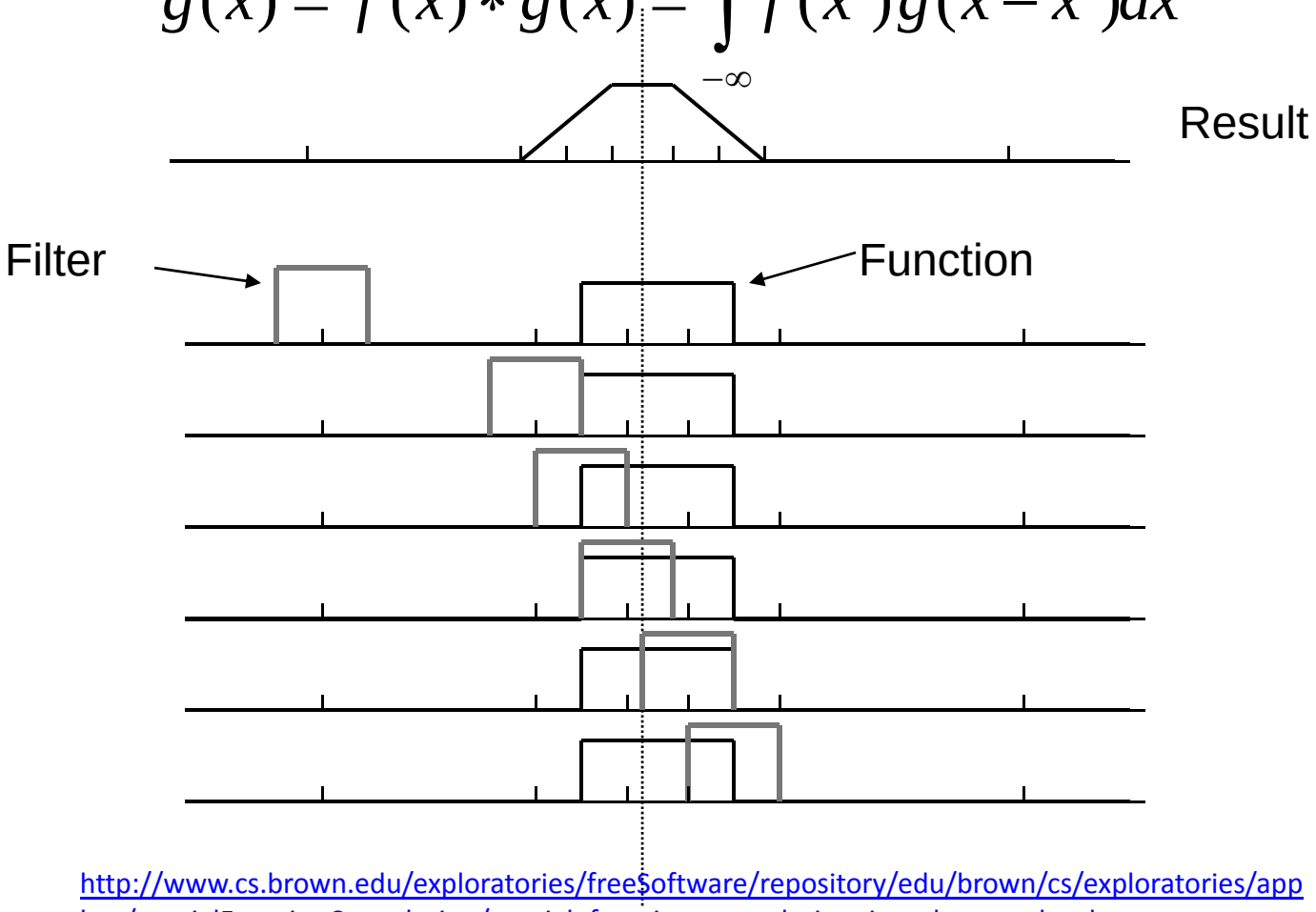


Convolution Definition

$$g(x) = f(x) * g(x) = \int_{-\infty}^{\infty} f(x')g(x - x')dx'$$

Convolution Example

$$g(x) = f(x) * g(x) = \int_{-\infty}^{\infty} f(x')g(x-x')dx'$$



http://www.cs.brown.edu/exploratories/freeSoftware/repository/edu/brown/cs/exploratories/applets/specialFunctionConvolution/special_function_convolution_java_browser.html

Convolution theorems

- **Convolution theorem**: *Convolution* in the *spatial* domain is equivalent to *multiplication* in the *frequency* domain.

$$f * h \longleftrightarrow F \cdot H$$

- **Symmetric theorem**: *Convolution* in the *frequency* domain is equivalent to *multiplication* in the *spatial* domain.

$$f \cdot h \longleftrightarrow F * H$$

Explanation using Fourier Transform