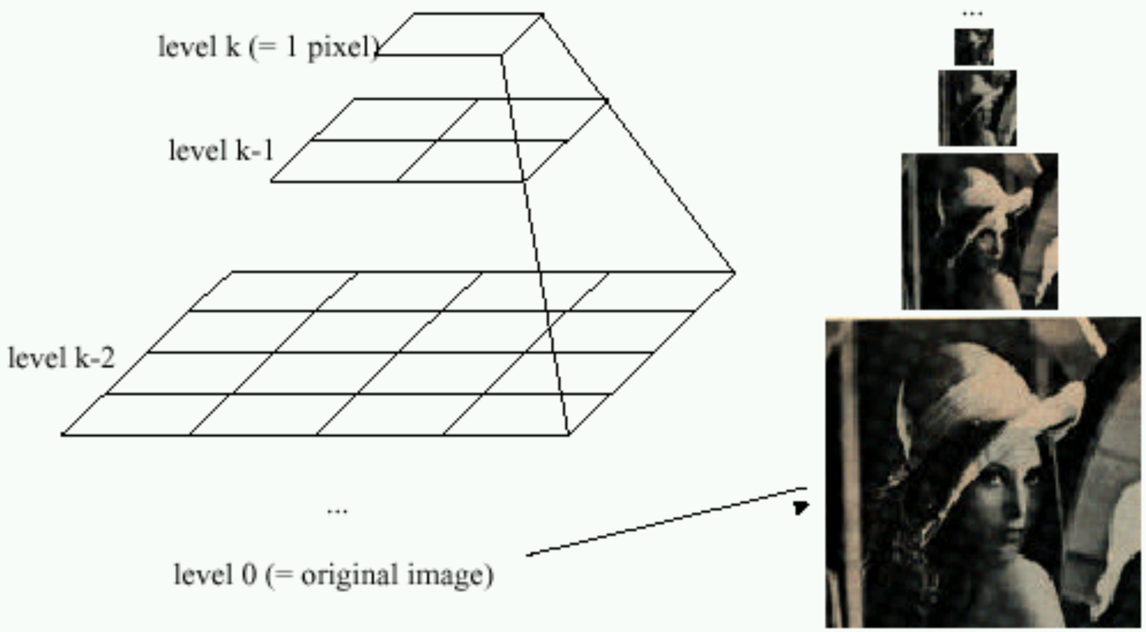
CSC589 Lecture 11

Pyramid Image Blending

Bei Xiao

Gaussian pyramids
[Burt and Adelson, 1983]

Idea: Represent NxN image as a “pyramid” of 1×1, 2×2, 4×4,…, $2^k \times 2^k$ images (assuming $N=2^k$)

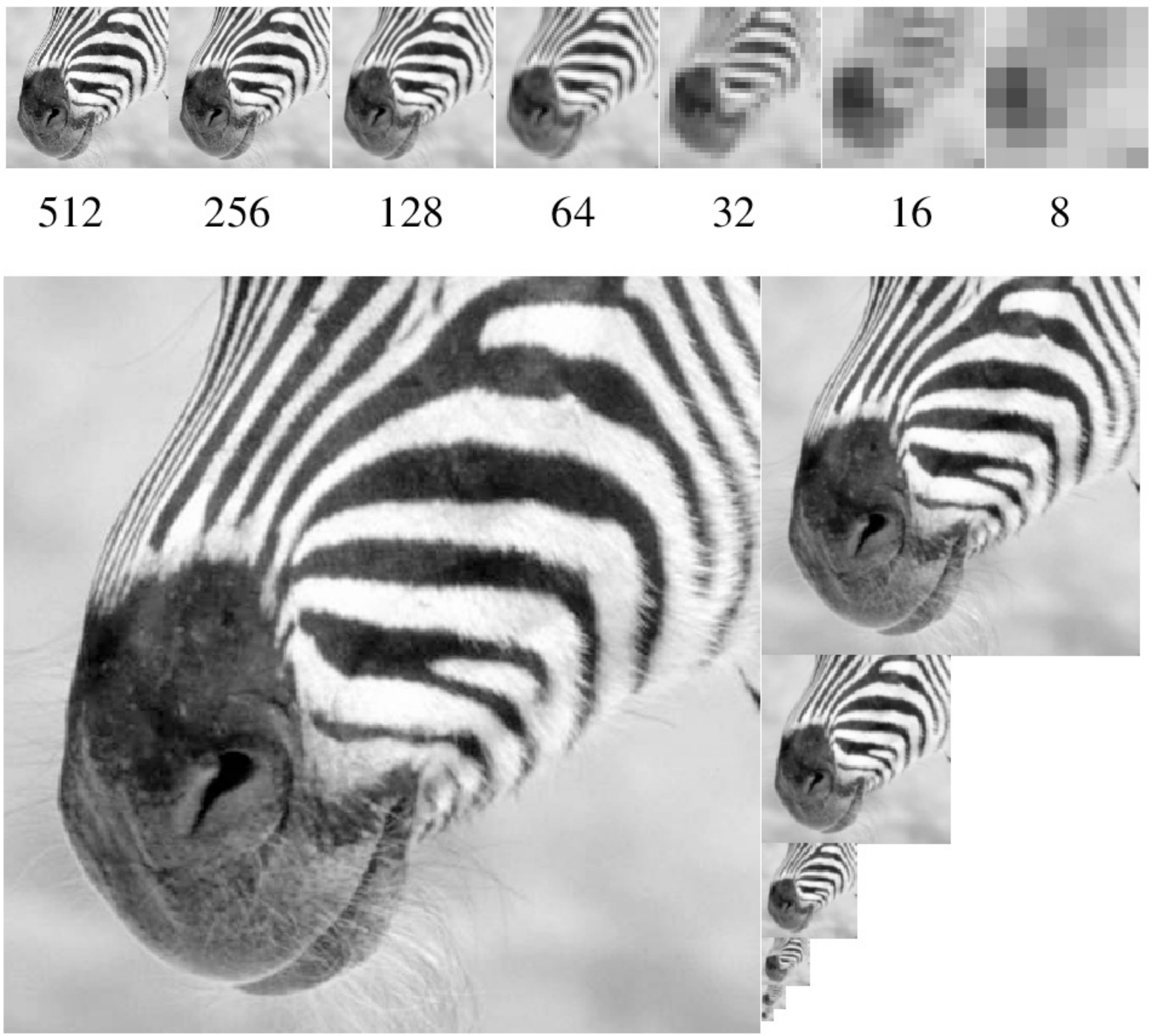


- In computer graphics, a *mip map* [Williams, 1983]
- A precursor to *wavelet transform*

Gaussian Pyramids have all sorts of applications in computer vision

Source: S. Seitz

Gaussian pyramid



A bar in the big image is a hair on the zebera’s nose; in smaller images, a stripe; in the smallest, the animal’s nose.

Source: Forsyth

What are Pyramids good for?

Improve Search

- Search over translations
- Search over scale
    - Template matching
    - E.g. find face at different scales

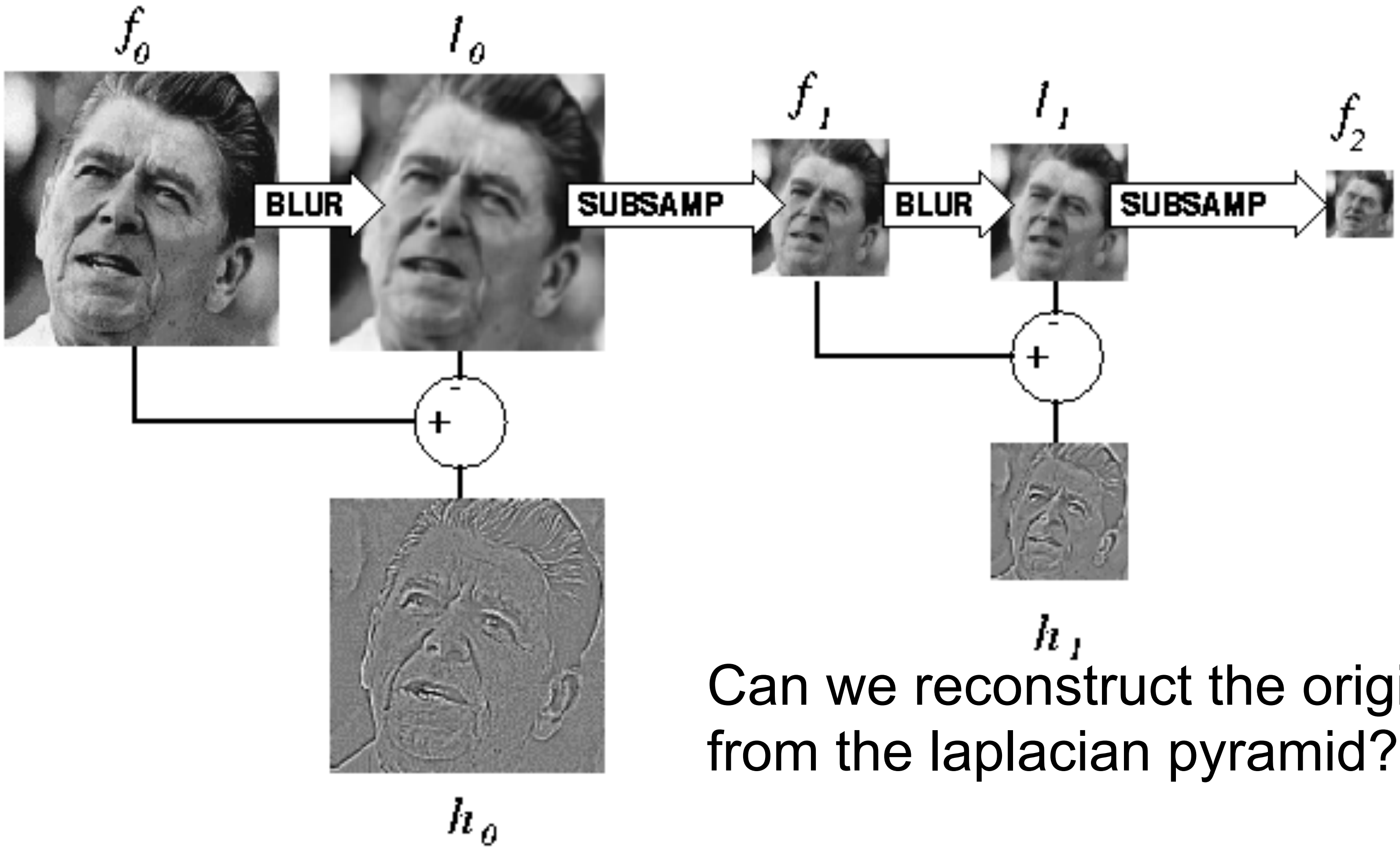
Precomputation

- Need to access image at different blur levels
- Useful for texture mapping at different resolutions (called mip-mapping)

Image Processing

- Editing frequency bands separately
- E.g. image blending

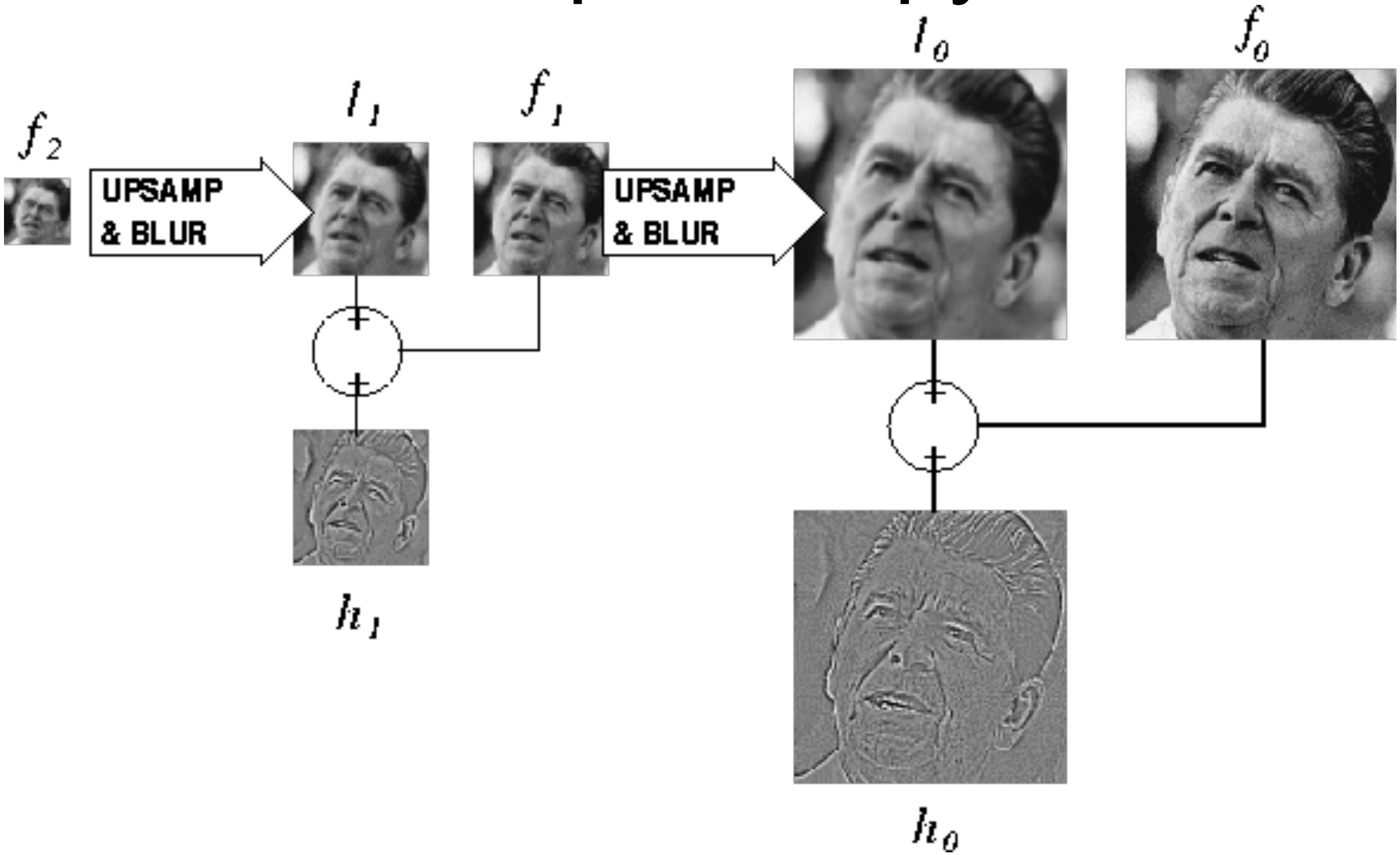
Computing Gaussian/Laplacian Pyramid



Can we reconstruct the original from the laplacian pyramid?

http://sepwww.stanford.edu/~morgan/texturematch/paper_html/node3.html

Can we reconstruct the original from the laplacian pyramid?

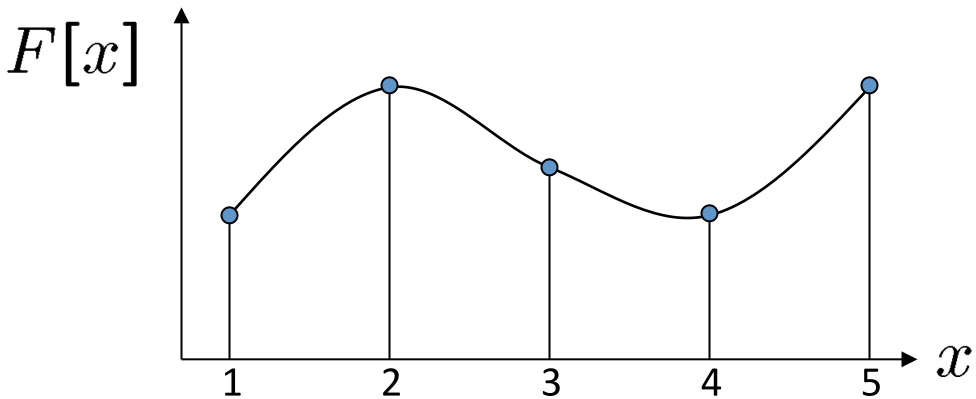


http://sepwww.stanford.edu/~morgan/texturematch/paper_html/node3.html

Upsampling

- This image is too small for this screen: 
- How can we make it 10 times as big?
- Simplest approach: repeat each row and column 10 times
- ("Nearest neighbor interpolation")

Image interpolation



d = 1 in this example

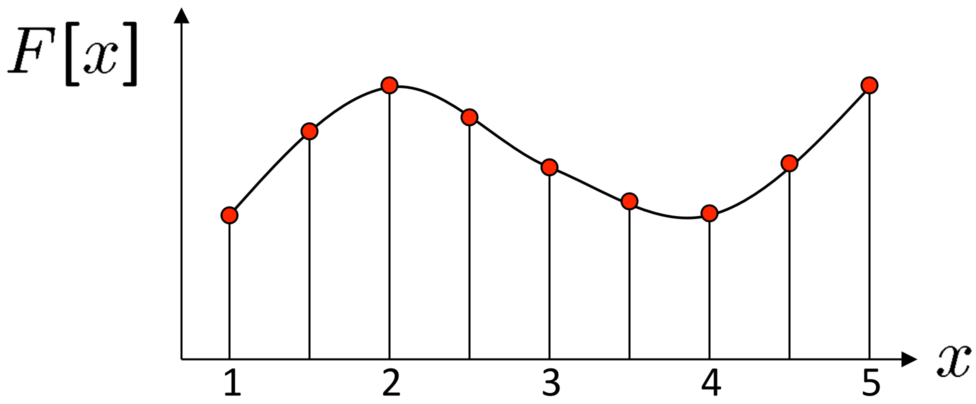
Recall how a digital image is formed

$$F[x, y] = \text{quantize}\{f(xd, yd)\}$$

- It is a discrete point-sampling of a continuous function
- If we could somehow reconstruct the original function, any new image could be generated, at any resolution and scale

Adapted from: S. Seitz

Image interpolation



d = 1 in this example

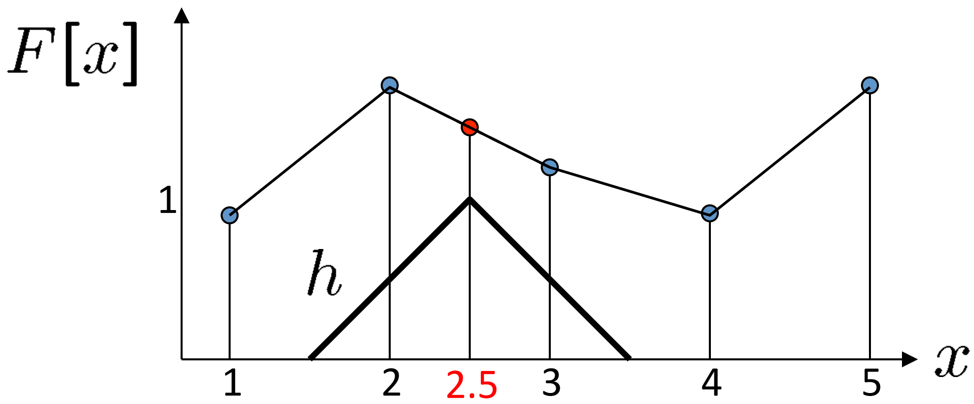
Recall how a digital image is formed

$$F[x,y] = \text{quantize}\{f(xd, yd)\}$$

- It is a discrete point-sampling of a continuous function
- If we could somehow reconstruct the original function, any new image could be generated, at any resolution and scale

Adapted from: S. Seitz

Image interpolation



d = 1 in this example

- What if we don't know f ?
  - Guess an approximation: $\tilde{f}$
  - Can be done in a principled way: filtering
  - Convert F to a continuous function:

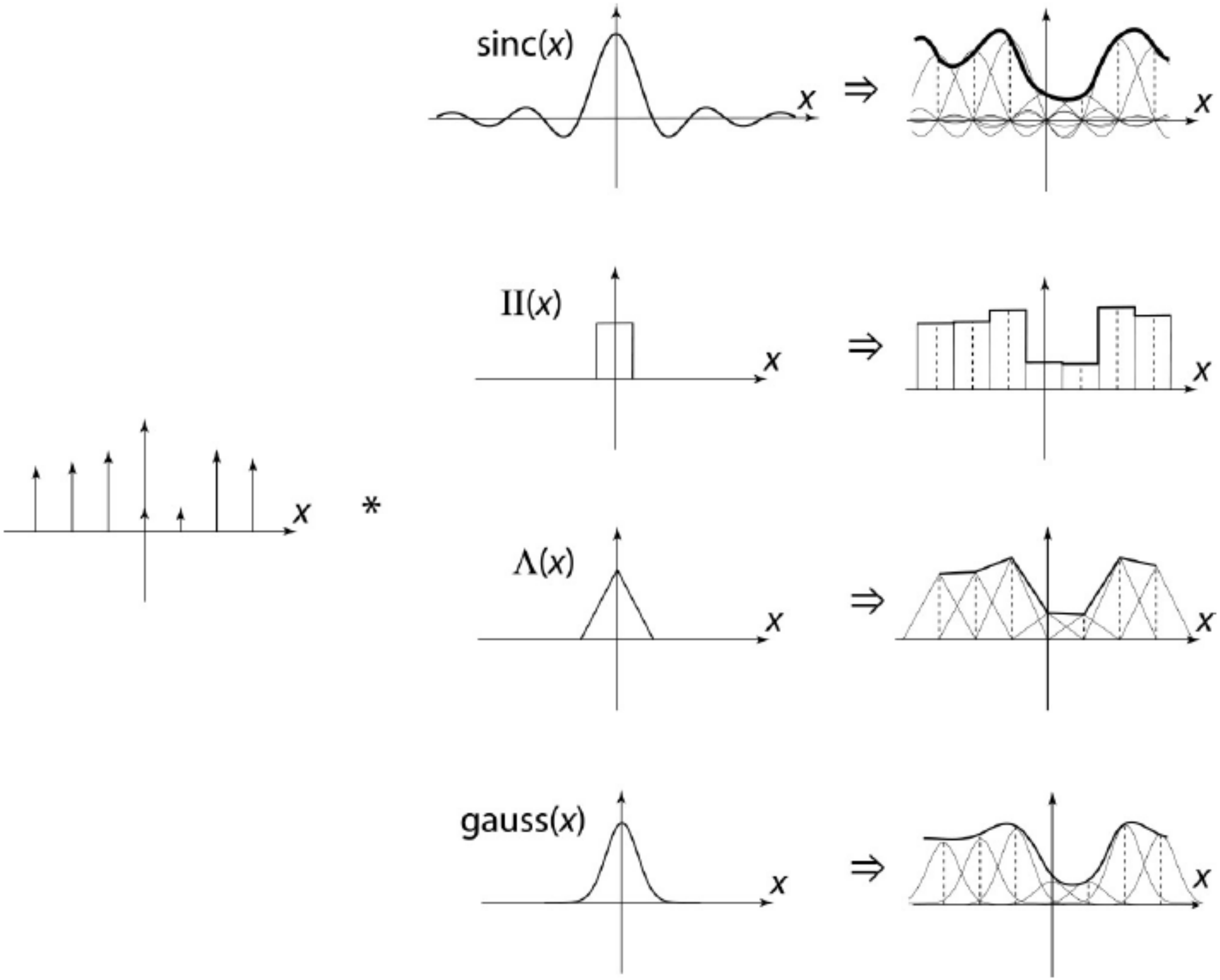
$$f_F(x) = F(\tfrac{x}{d}) \text{ when } \tfrac{x}{d} \text{ is an integer, 0 otherwise}$$

  - Reconstruct by convolution with a *reconstruction filter, h*

$$\tilde{f} = h * f_F$$

Adapted from: S. Seitz

Image interpolation



"Ideal" reconstruction

Nearest-neighbor interpolation

Linear interpolation

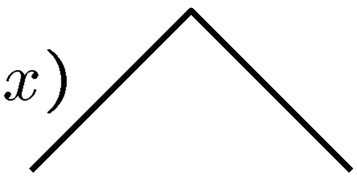
Gaussian reconstruction

Source: B. Curless

Reconstruction filters

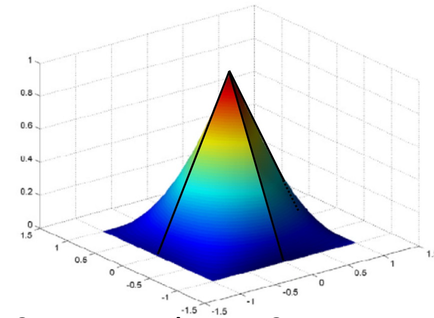
- What does the 2D version of this hat function look like?

$h(x)$



performs
linear interpolation

$h(x, y)$



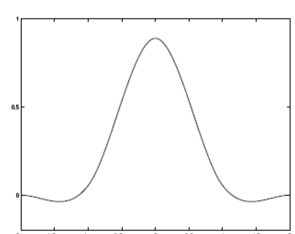
(tent function) performs
bilinear interpolation

Often implemented without cross-correlation

- E.g., http://en.wikipedia.org/wiki/Bilinear_interpolation

Better filters give better resampled images

- **Bicubic** is common choice



Cubic reconstruction filter

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

Image interpolation

Original image:  x 10

Nearest-neighbor interpolation

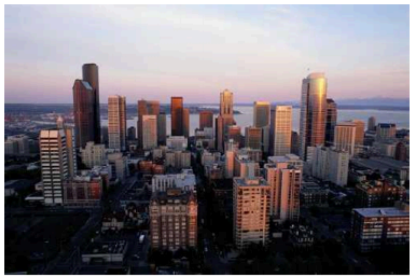
Bilinear interpolation

Bicubic interpolation

Image Composition

Image Composition

1. Extract Sprites (e.g using *Intelligent Scissors* in Photoshop)




2. Blend them into the composite (in the right order)

Composite by
David Dewey

Image Composition

What can you do?

1. Copy and paste. Will generate artifacts at the border.

2. Feathering, using alpha channel and use a weighted sum over a window of two images (Feathering).

3. Combine the two images at different frequency bands (Pyramid Image Blending)

Alpha Channel

- Add one more channel:

-Image (R,G,B, alpha)

- Encodes transparency (or pixel coverage)

-Alpha = 1: Opaque object (complete coverage)

-Alpha = 0: transparent object (no coverage)

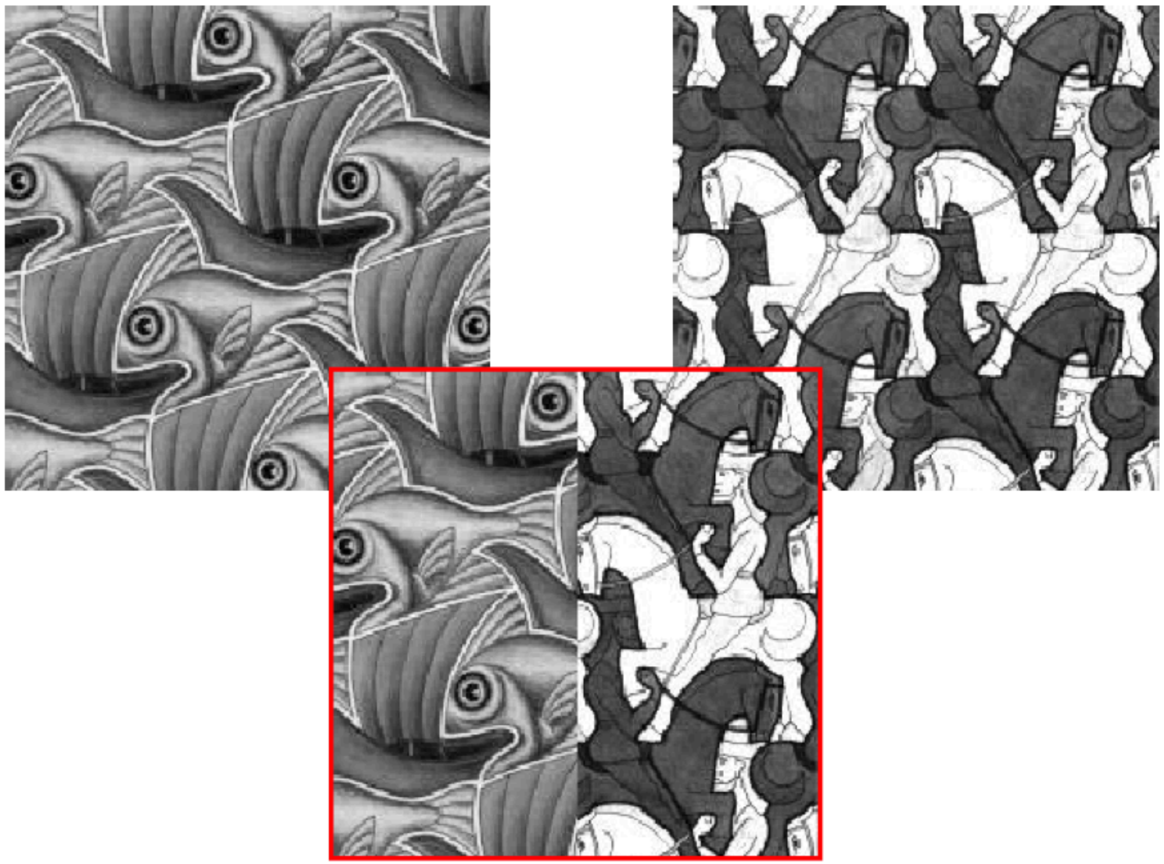
-$0<$Alpha < 1: semi-transparent (partial coverage)

- Example: alpha = 0.3

- Read in alpha channel in Python:

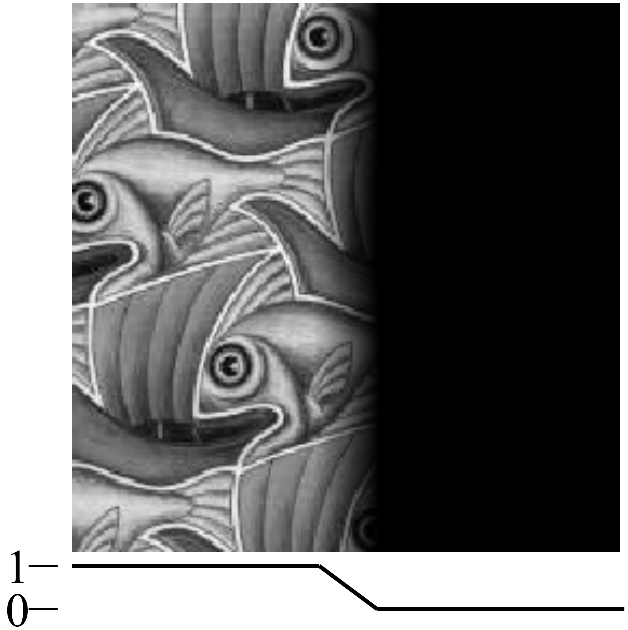
- `alpha_img = cv2.imread(path, cv2.IMREAD_UNCHANGED)`

Image Blending

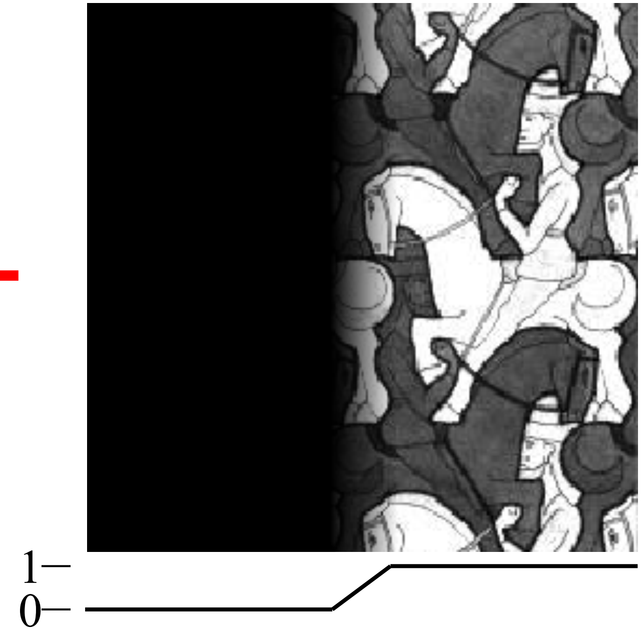


Slide: Efros

Alpha Blending/Feathering



+

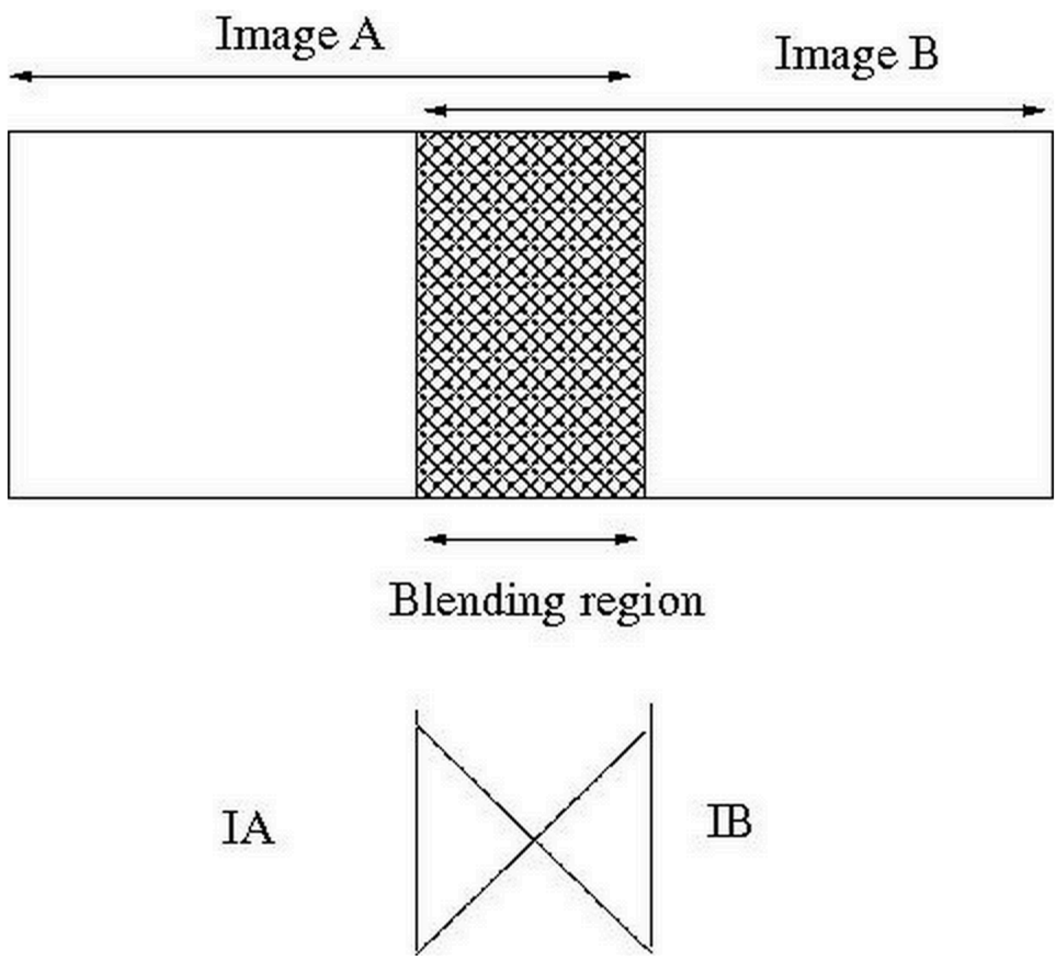


=

$$I_{blend} = \alpha I_{left} + (1\text{-}\alpha) I_{right}$$

Alpha Blending/Feathering

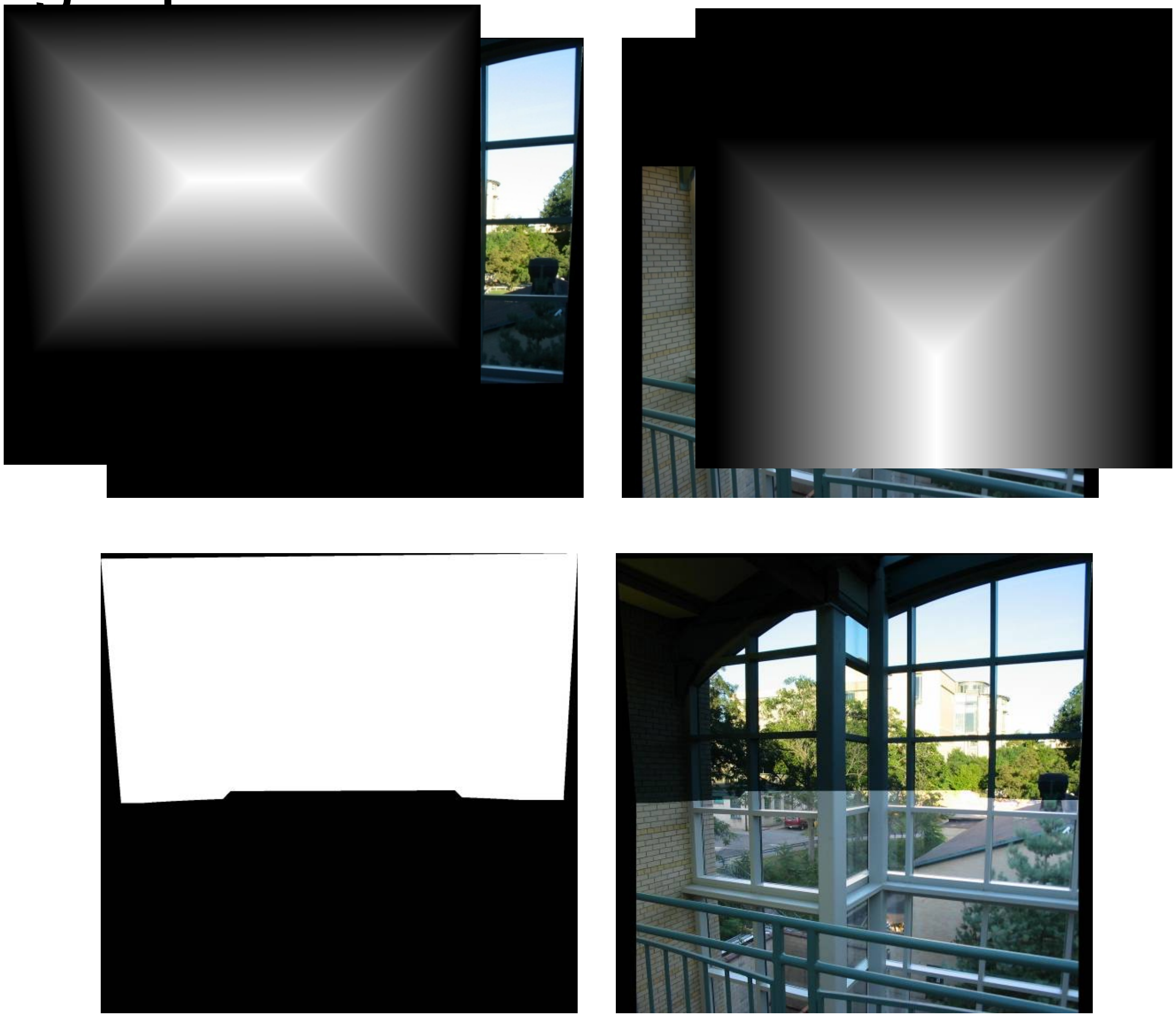
PB(i,j) = (1-w)*PA(i,j) + w*PB(i,j)



Setting alpha: simple averaging

Alpha = .5 in overlap region

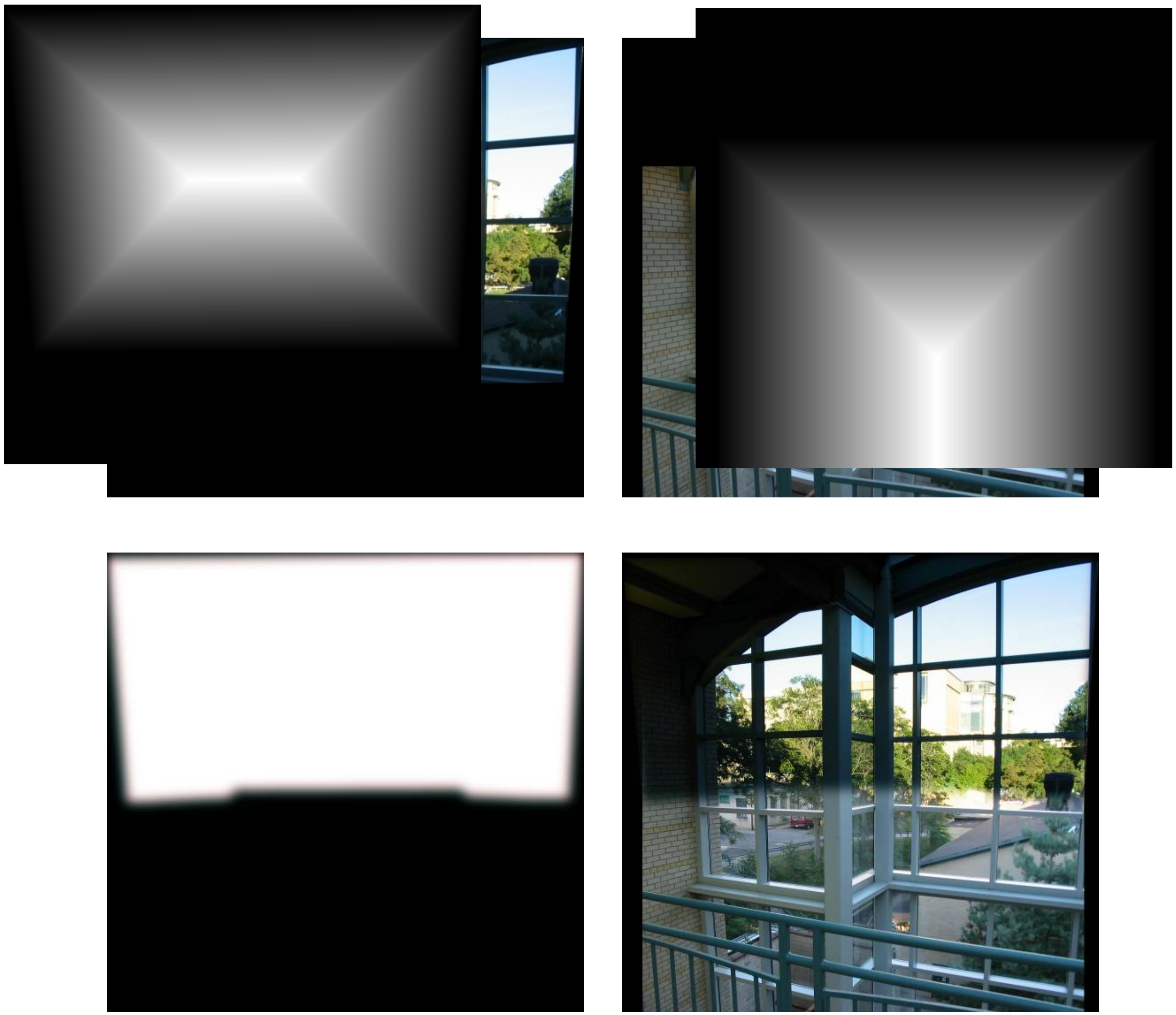
Setting alpha: center seam



Compute Distance Transform Between binary Images using `bwdist`

Alpha = logical(dtrans1>dtrans2)

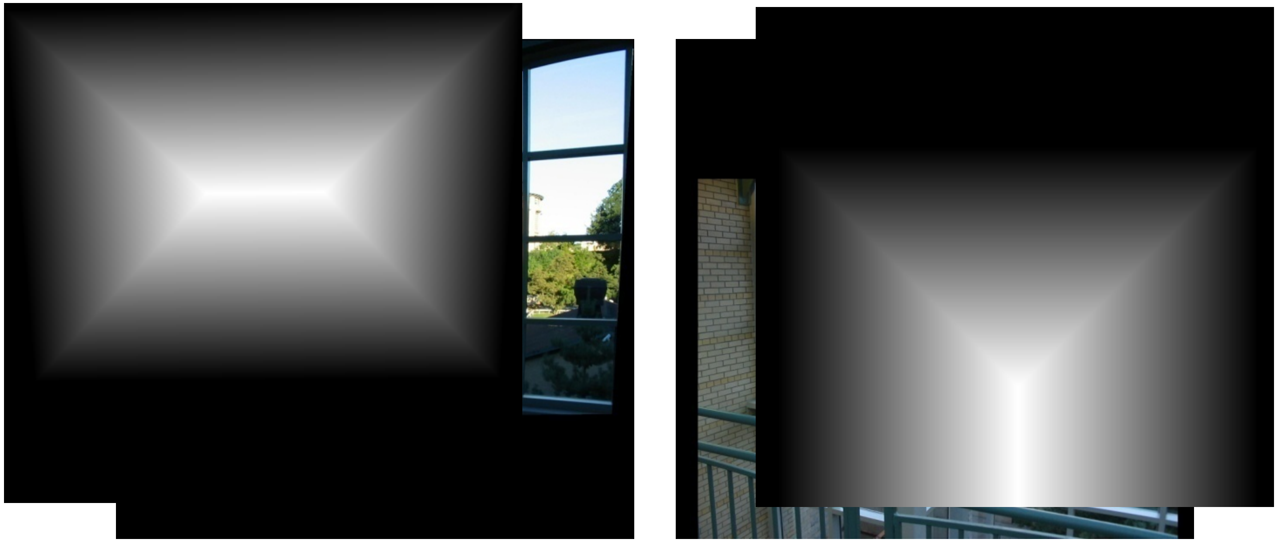
Setting alpha: blurred seam



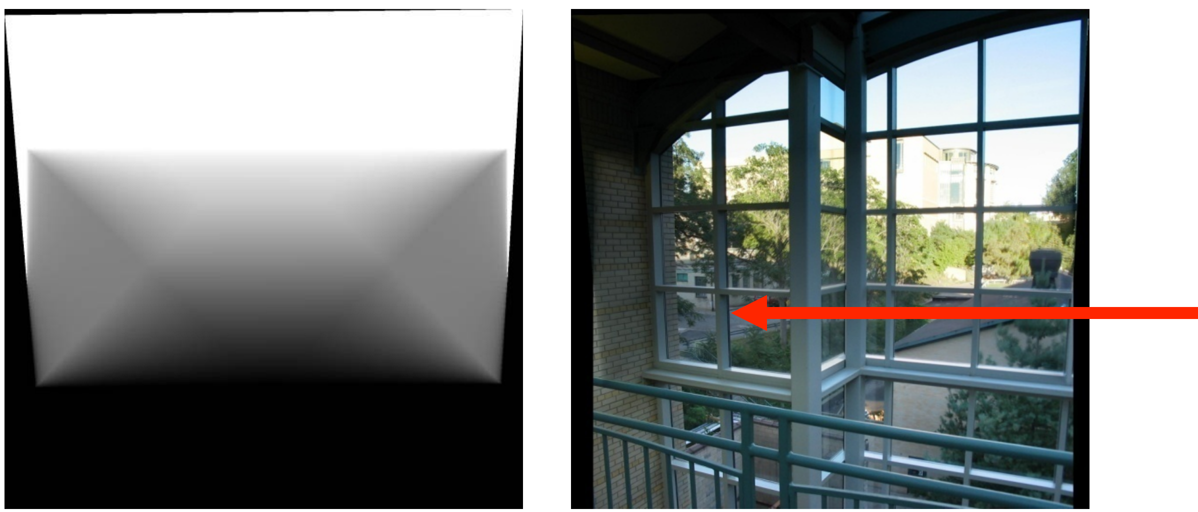
Distance transform

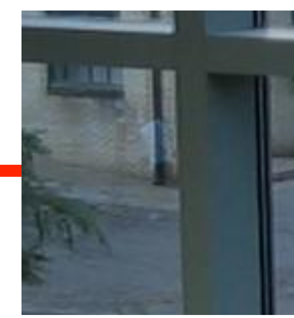
Alpha = blurred

Setting alpha: center weighting



Distance transform

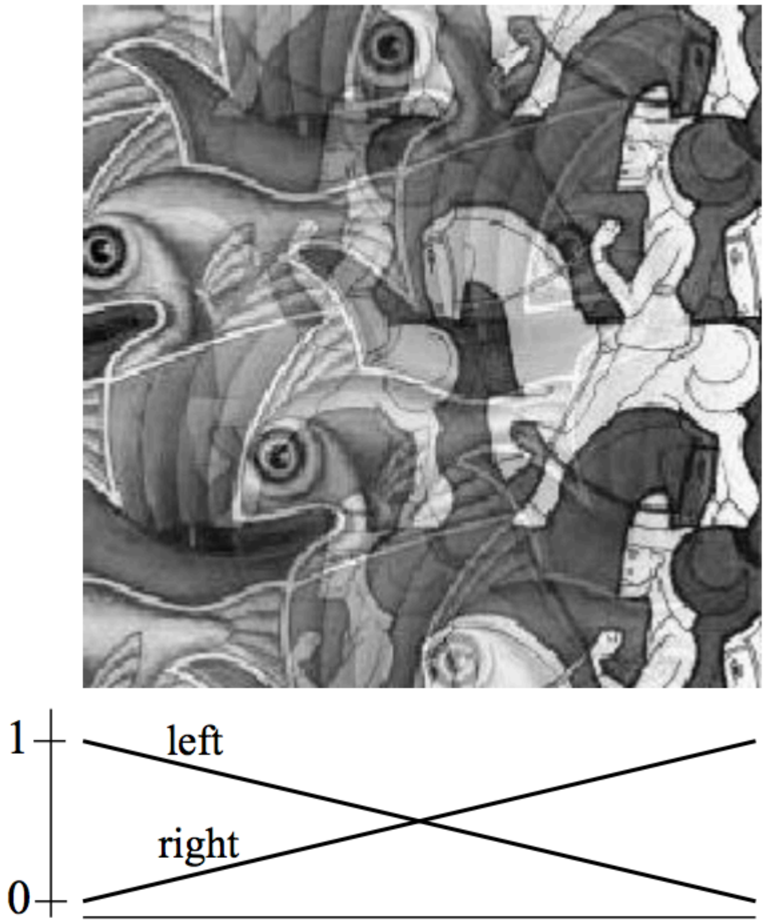


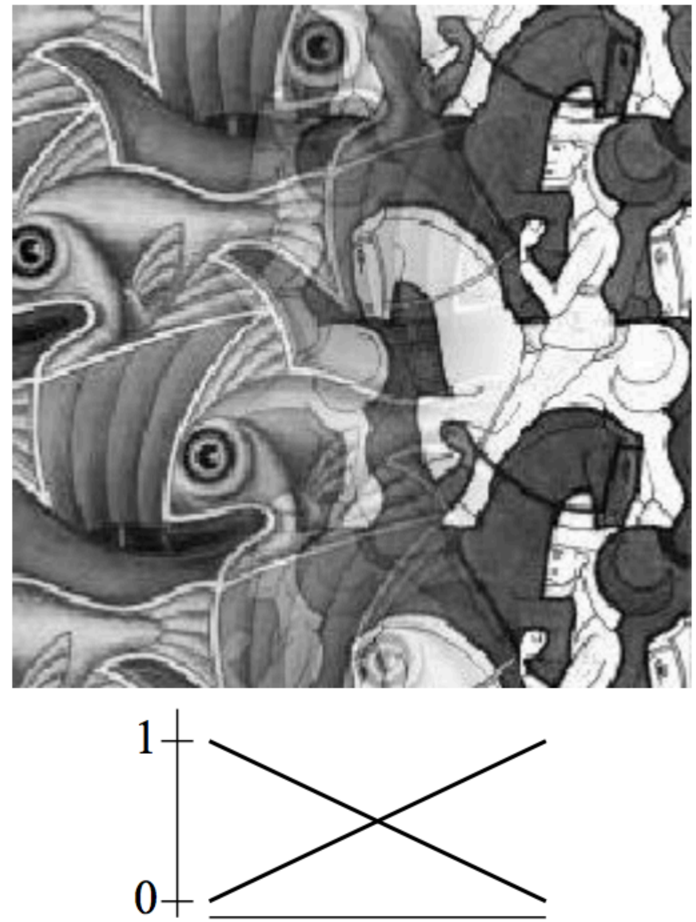


Ghost!

Alpha = dtrans1 / (dtrans1+dtrans2)

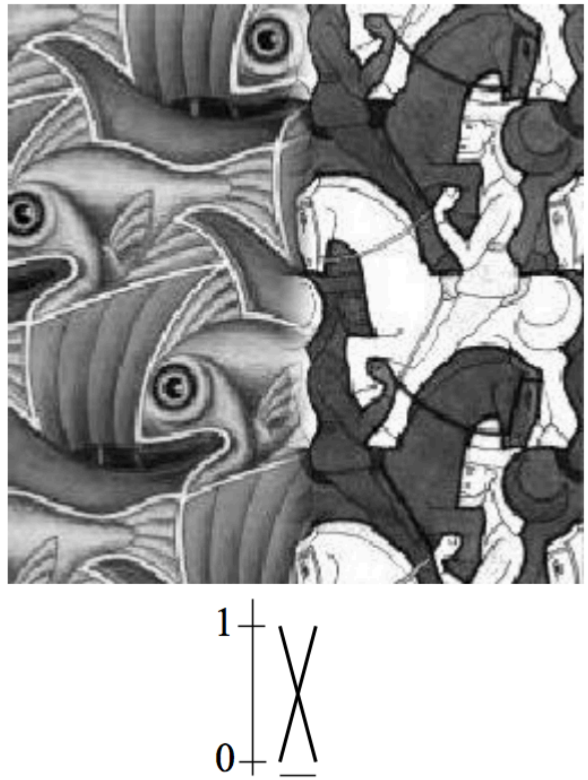
Affect of Window Size





Slide: Efros

Affect of Window Size



"Optimal" Window: smooth but not ghosted

Slide: Efros

What is the optimal window size?

To avoid seams

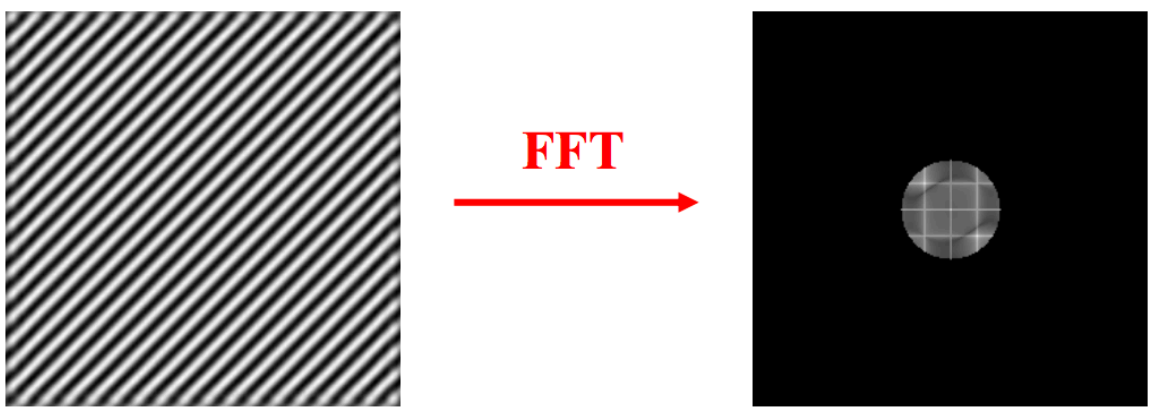
- window = size of largest prominent feature

To avoid ghosting

- window <= 2*size of smallest prominent feature

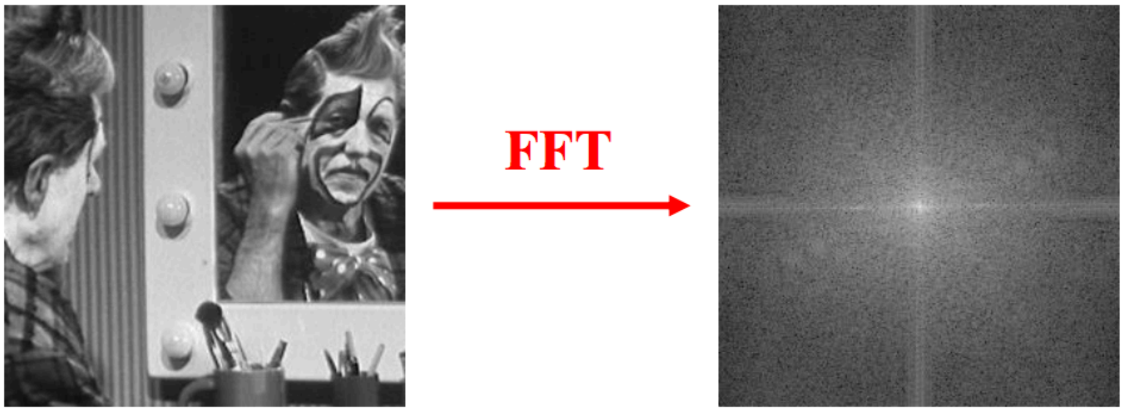
Natural to cast this in the *Fourier domain*

- largest frequency <= 2*size of smallest frequency
- image frequency content should occupy one “octave” (power of two)



Slide: Efros

What if the Frequency Spread is Wide?



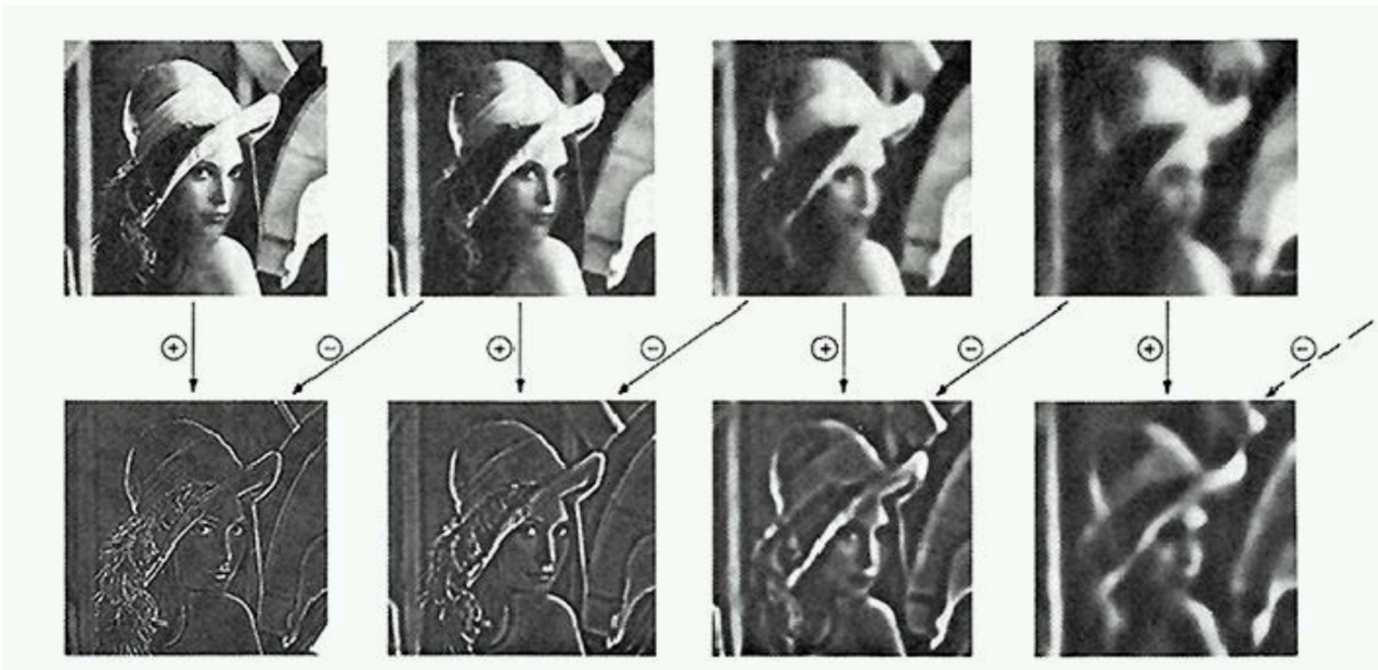
Idea (Burt and Adelson)

- Compute $F_{left} = FFT(I_{left})$, $F_{right} = FFT(I_{right})$
- Decompose Fourier image into octaves (bands)
  - $F_{left} = F_{left}^{1} + F_{left}^{2} + \ldots$
- Feather corresponding octaves F_{left}^{i} with F_{right}^{i}
  - Can compute inverse FFT and feather in spatial domain
- Sum feathered octave images in frequency domain

Better implemented in *spatial domain*

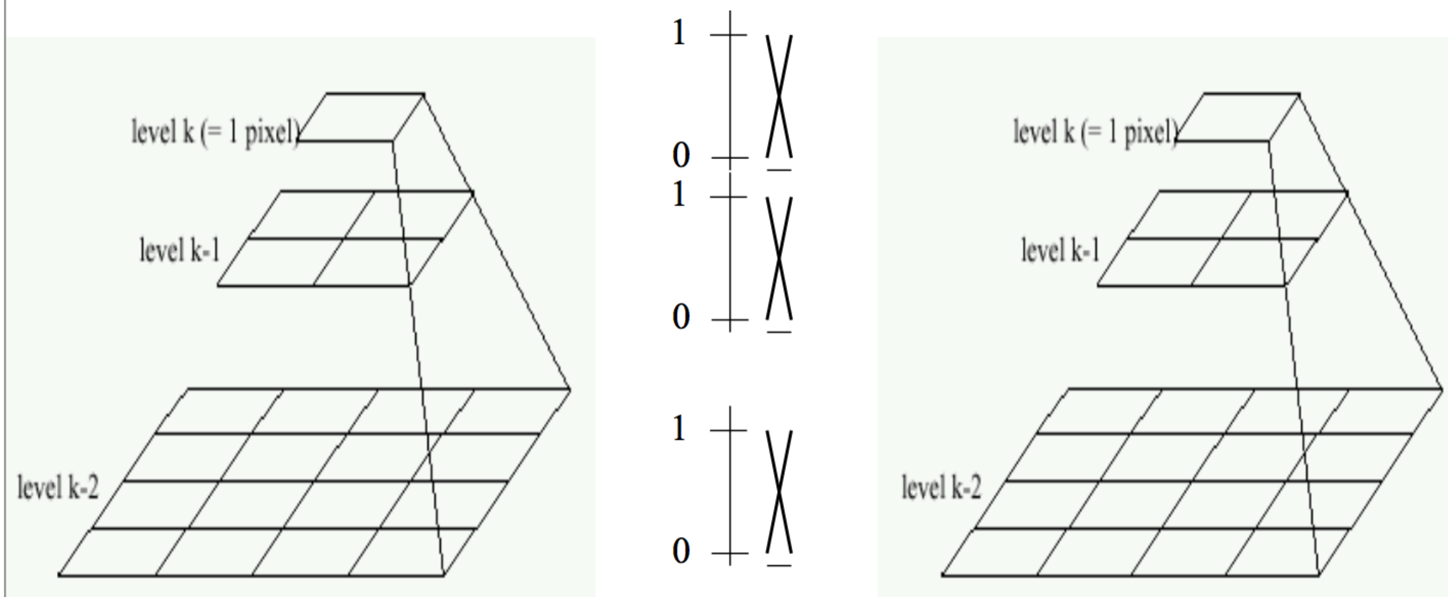
Octaves in the Spatial Domain

Lowpass Images



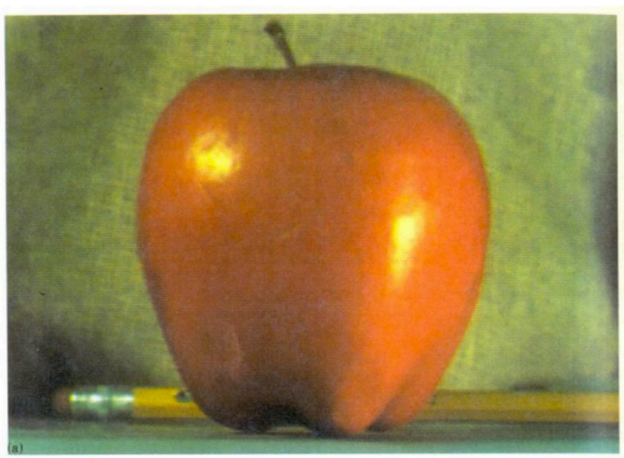
Bandpass Images

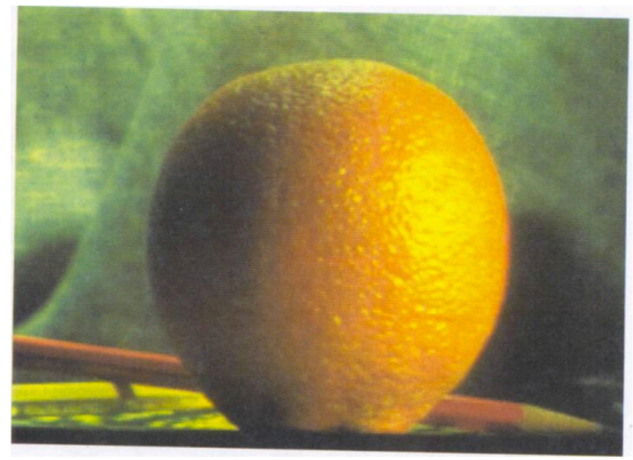
Pyramid Blending

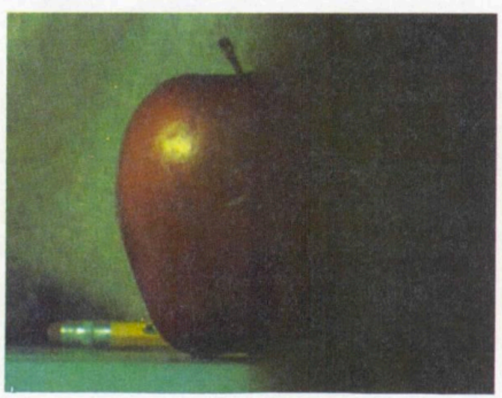


Left pyramid　　　blend　　　Right pyramid

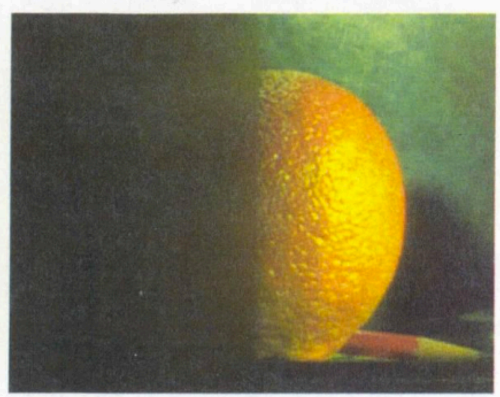
Pyramid Blending



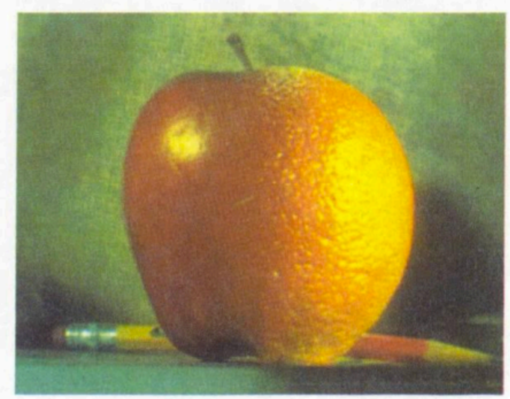




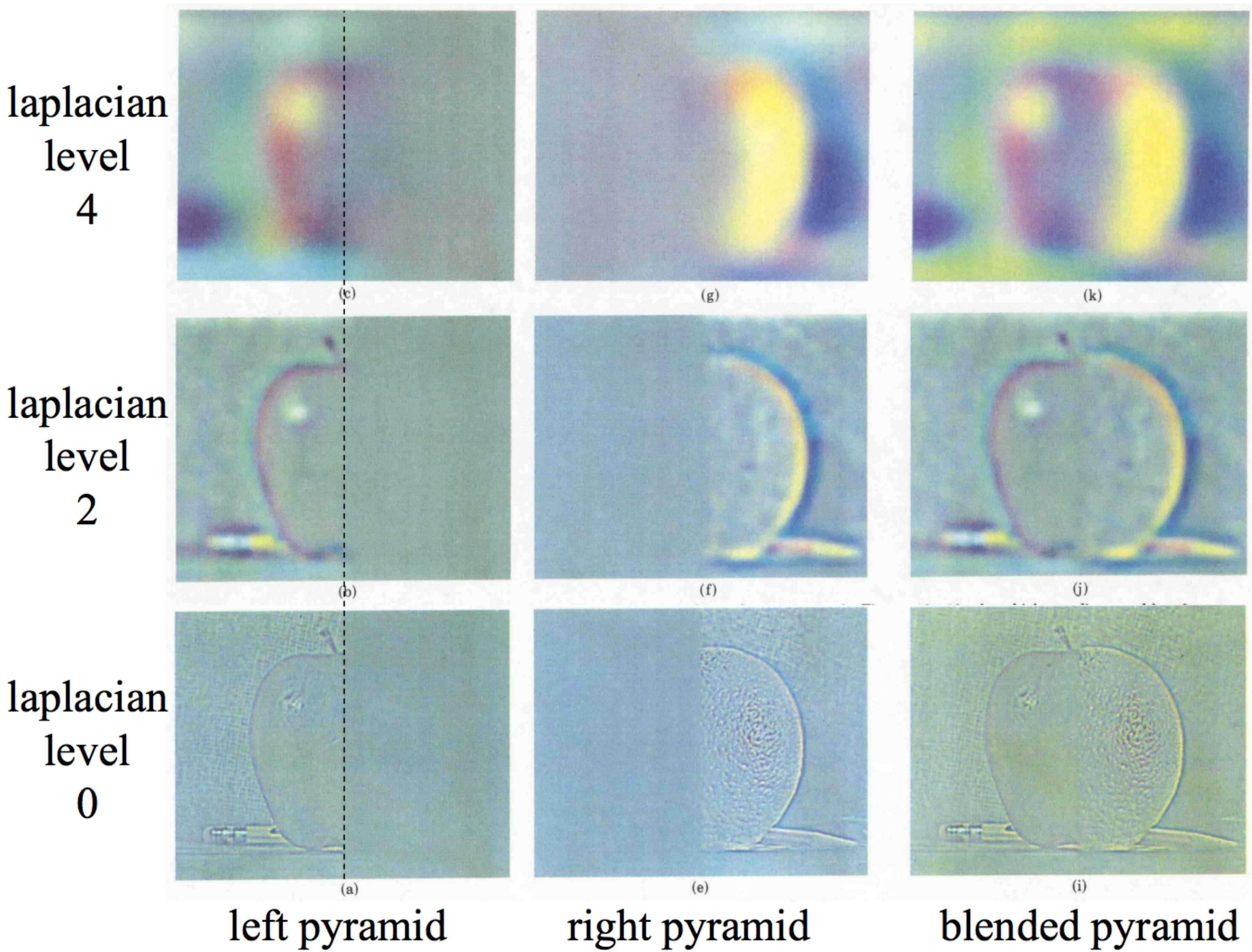
(d)



(h)



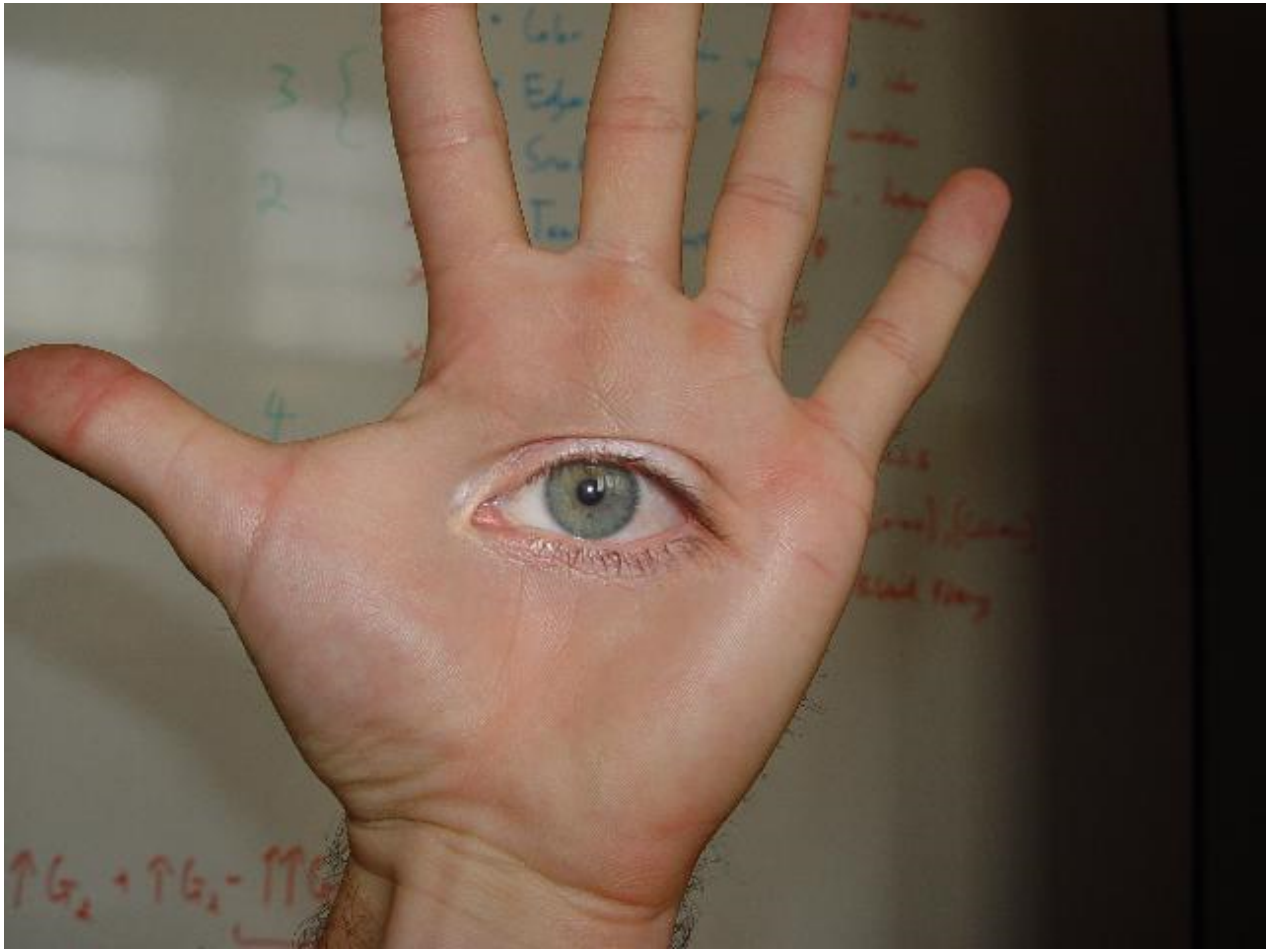
(l)


laplacian level 4
laplacian level 2
laplacian level 0
(c)
(g)
(k)
(b)
(f)
(j)
(a)
(e)
(i)
left pyramid
right pyramid
blended pyramid

Laplacian Pyramid: Blending

1. Load the two images of apple and orange
2. Find the Gaussian Pyramids for apple and orange (in this particular example, number of levels is 6)
2. From Gaussian Pyramids, find their Laplacian Pyramids
4. Now join the left half of apple and right half of orange in each levels of Laplacian Pyramids
4. Finally from this joint image pyramids, reconstruct the original image.

Horror Photo! (Homework 3)



david dmartin (Boston College)

Blending Regions

Laplacian Pyramid: Blending

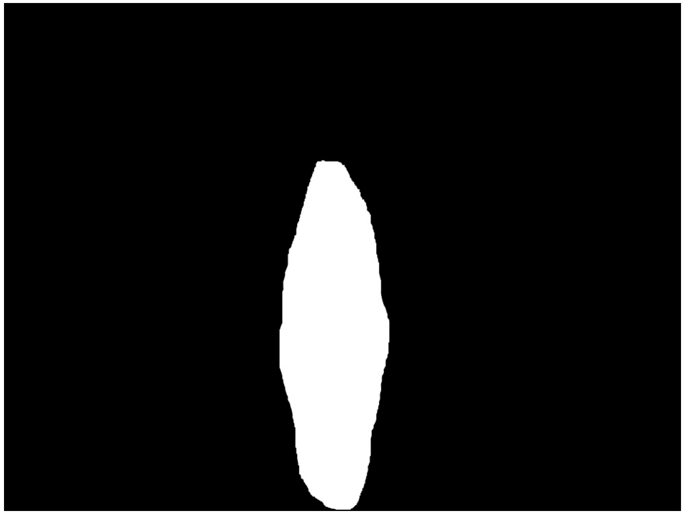
General Approach:

1. Build Laplacian pyramids *LA* and *LB* from images *A* and *B*
2. Build a Gaussian pyramid *GR* from selected region *R*
3. Form a combined pyramid *LS* from *LA* and *LB* using nodes of *GR* as weights:
   - *LS(i,j) = GR(I,j,)*LA(I,j) + (1-GR(I,j))*LB(I,j)*
4. Collapse the *LS* pyramid to get the final blended image

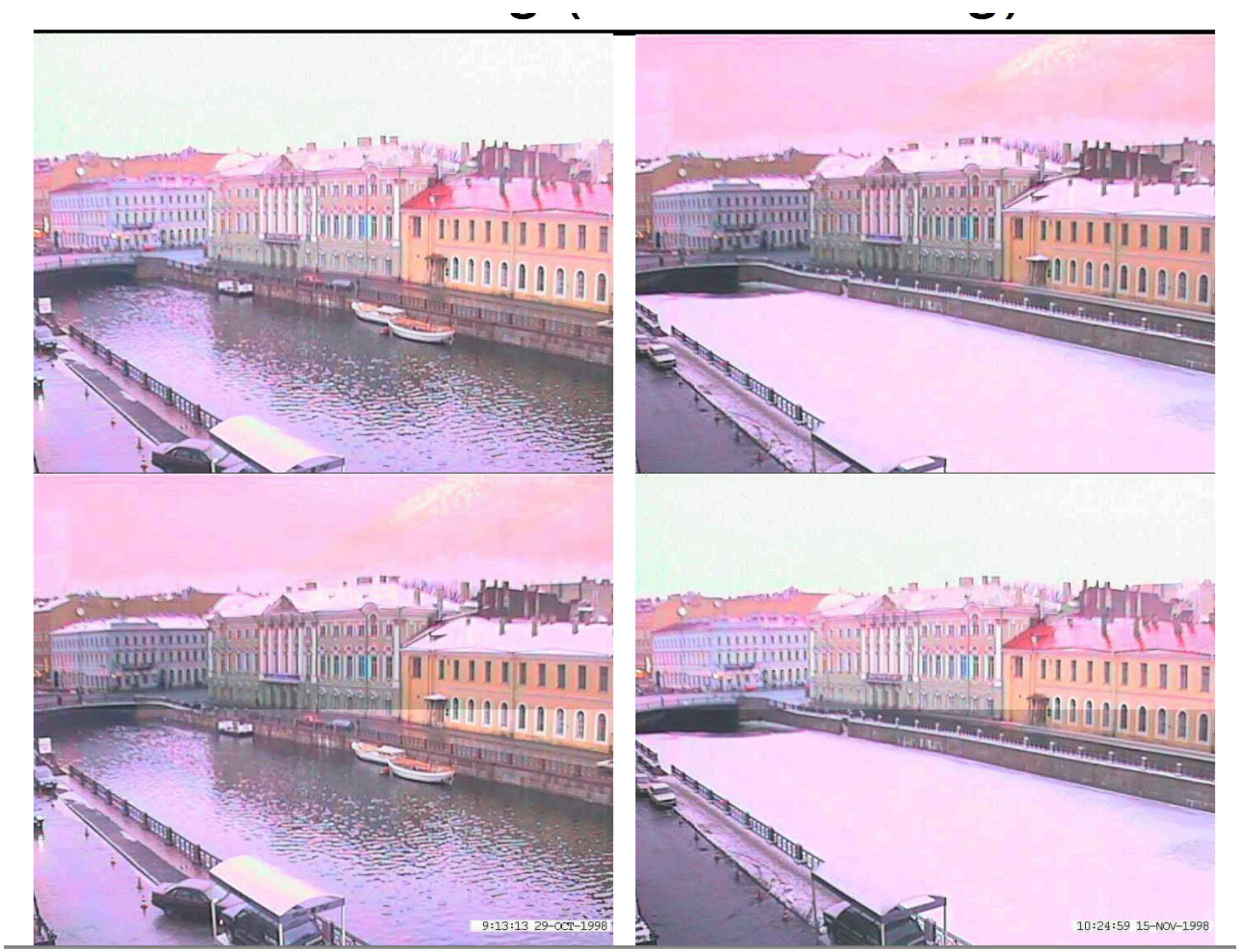
In Python:

```
def blend(lpr_white,lpr_black,gauss_py_mask):
Blend_pyr = []
k = len(gauss_pyr_mask)
for I in range(0,k):
  p1= gauss_pyr_mask[i]*lapl_pyr_white[i]
  p2=(1 - gauss_pyr_mask[i])*lapl_pyr_black[i]
  blended_pyr.append(p1 + p2)
Return blended_pyr
```

Application: seamless scene blending



Season Blending (St. Petersberg)



Season Blending (St. Petersberg)

Take-home reading

- Image Blending:
- http://pages.cs.wisc.edu/~csverma/CS766_09/ImageMosaic/imagemosaic.html

- Pyramids and wavelets: Chapter 3.5 Szeliski
- Chapter 9.3, Szeliski (Gradient domain image blending)