

3D Reconstruction with Computer Vision

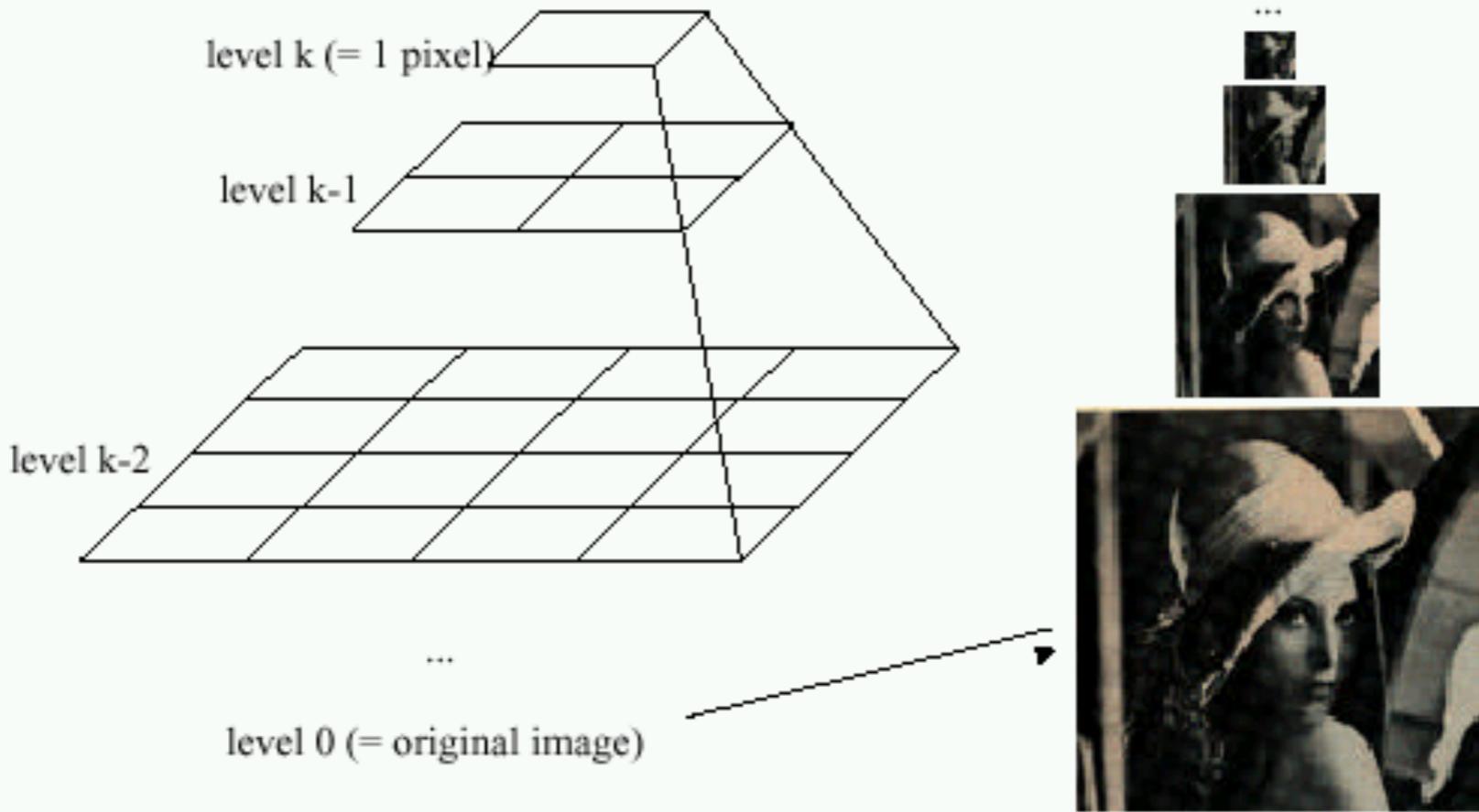
Meeting 6: Image Blending



Slides by Alexei Efros and others
CS 378 Fall 2014, UT Austin, Bryan Klingner, 16 September

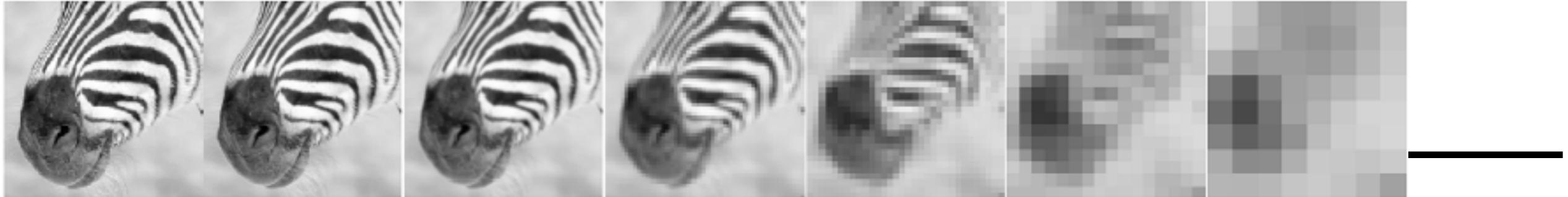
Image Pyramids

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



Known as a **Gaussian Pyramid** [Burt and Adelson, 1983]

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



512

256

128

64

32

16

8



A bar in the big images is a hair on the zebra's nose; in smaller images, a stripe; in the smallest, the animal's nose

What are they good for?

Improve Search

- Search over translations
 - Classic coarse-to-fine strategy
- Search over scale
 - Template matching
 - E.g. find a face at different scales

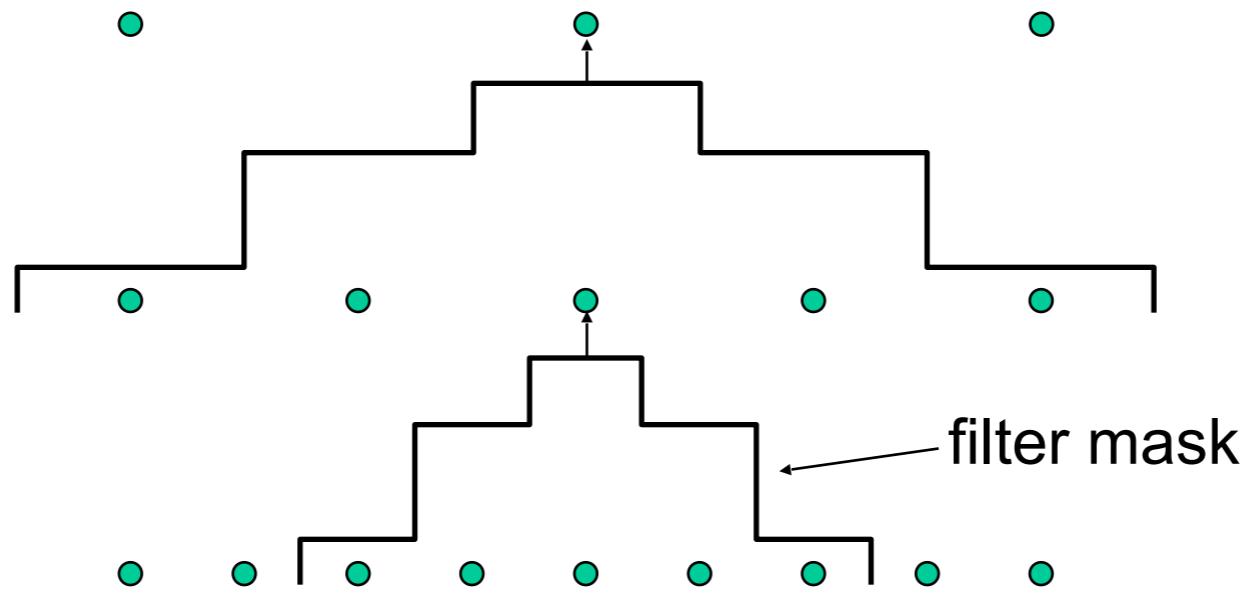
Pre computation

- 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...

Gaussian pyramid construction



Repeat

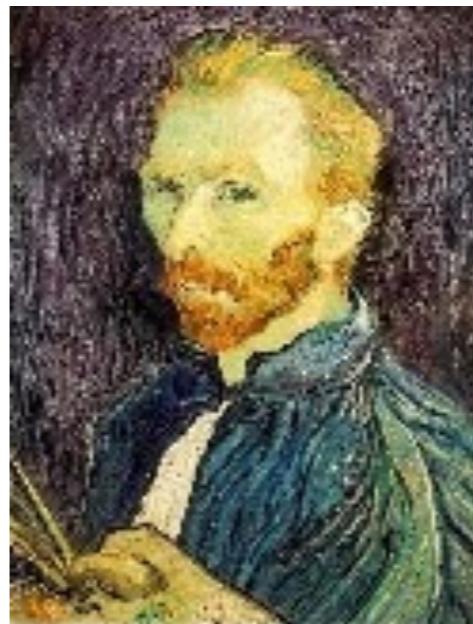
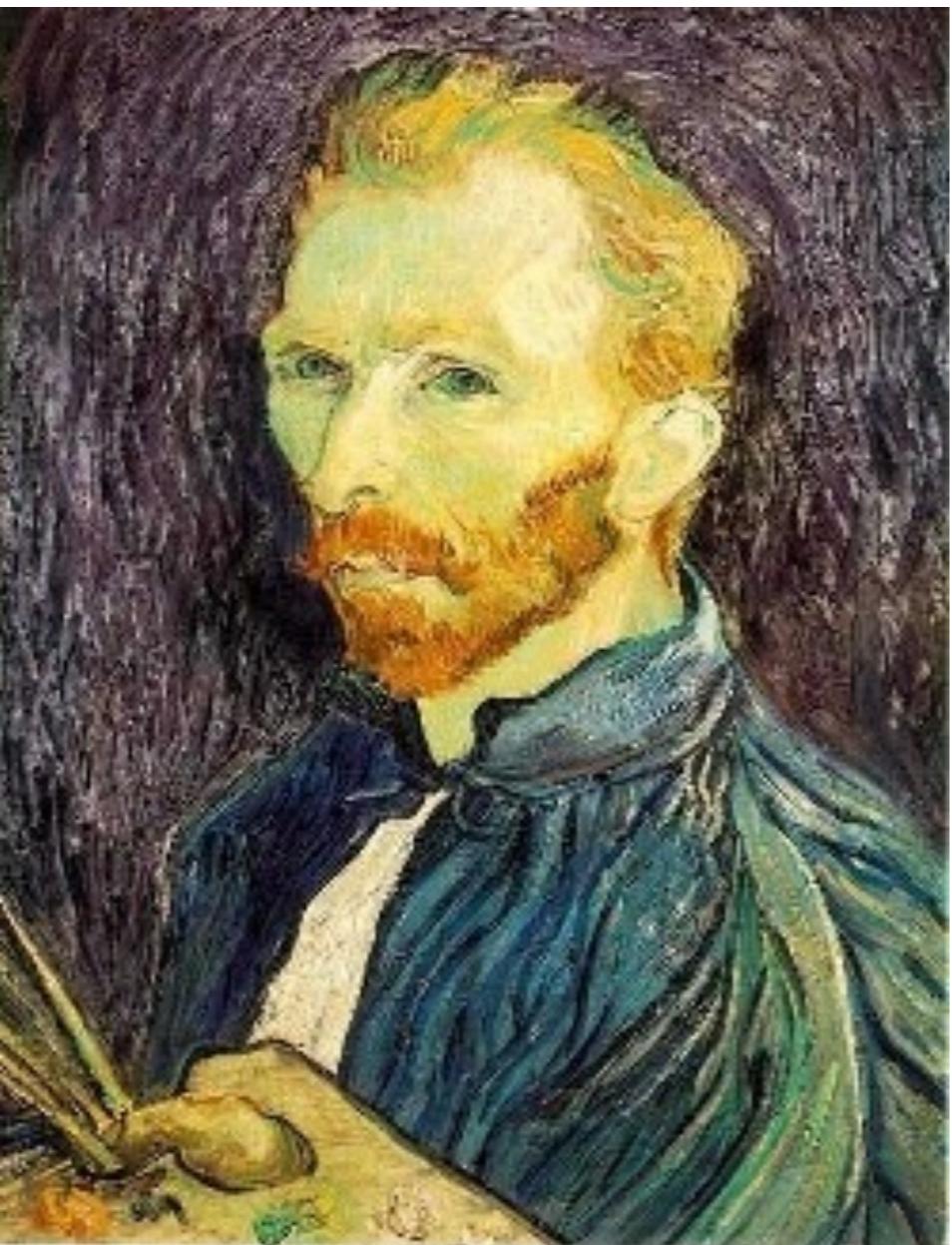
- Filter
- Subsample

Until minimum resolution reached

- can specify desired number of levels (e.g., 3-level pyramid)

The whole pyramid is only $4/3$ the size of the original image!

Image sub-sampling



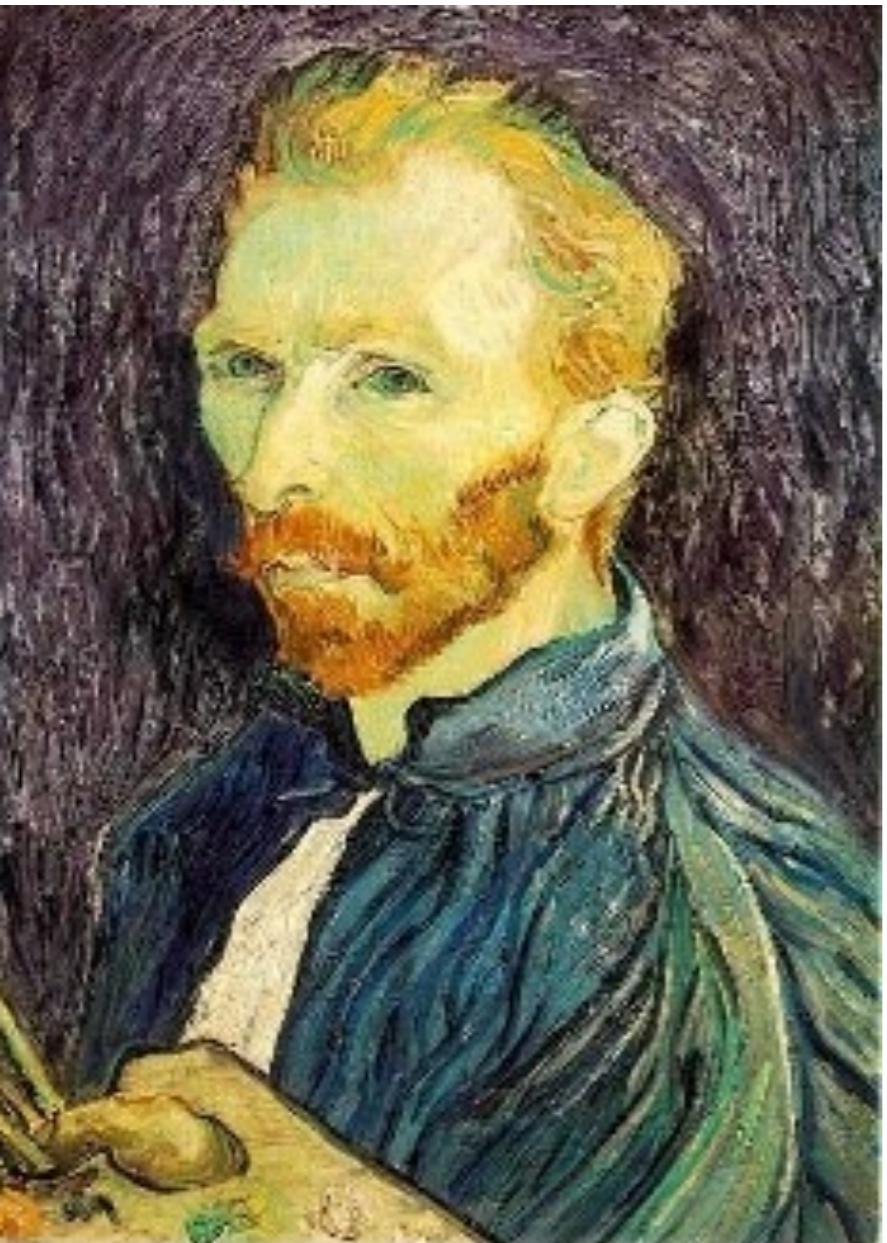
1/4



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



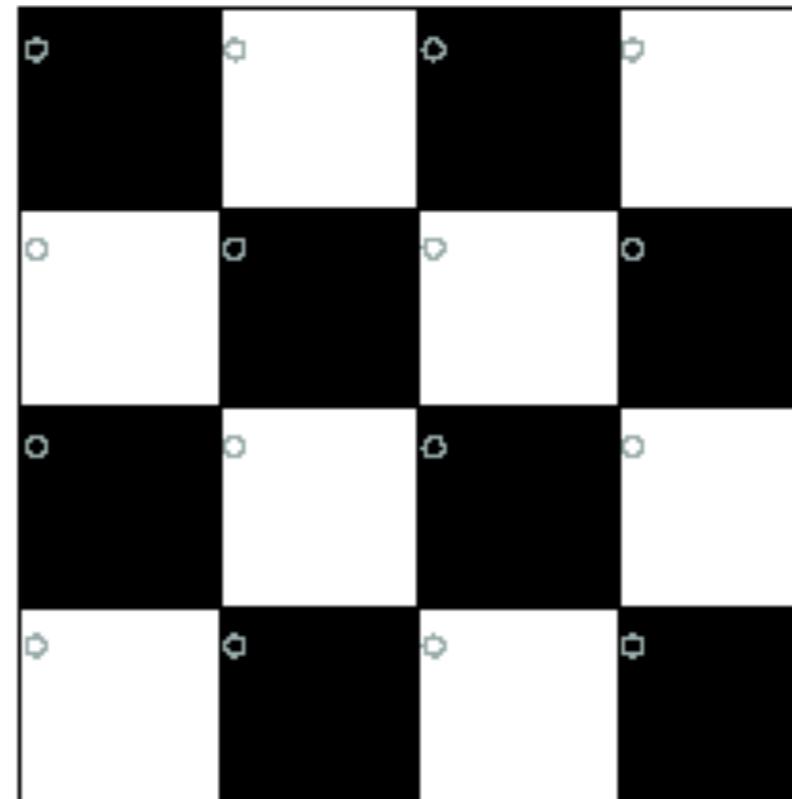
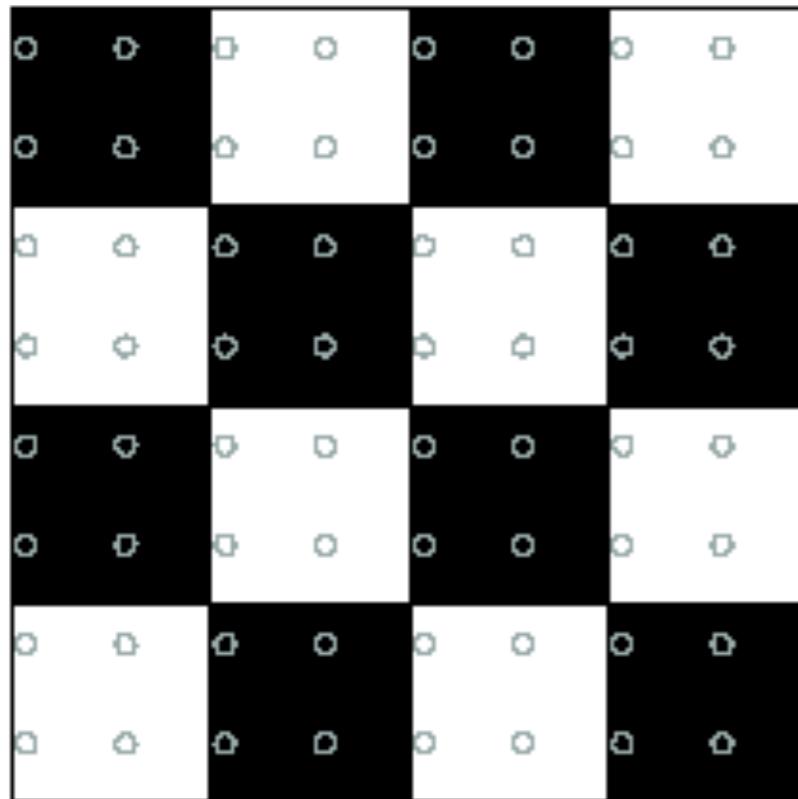
1/4 (2x zoom)



1/8 (4x zoom)

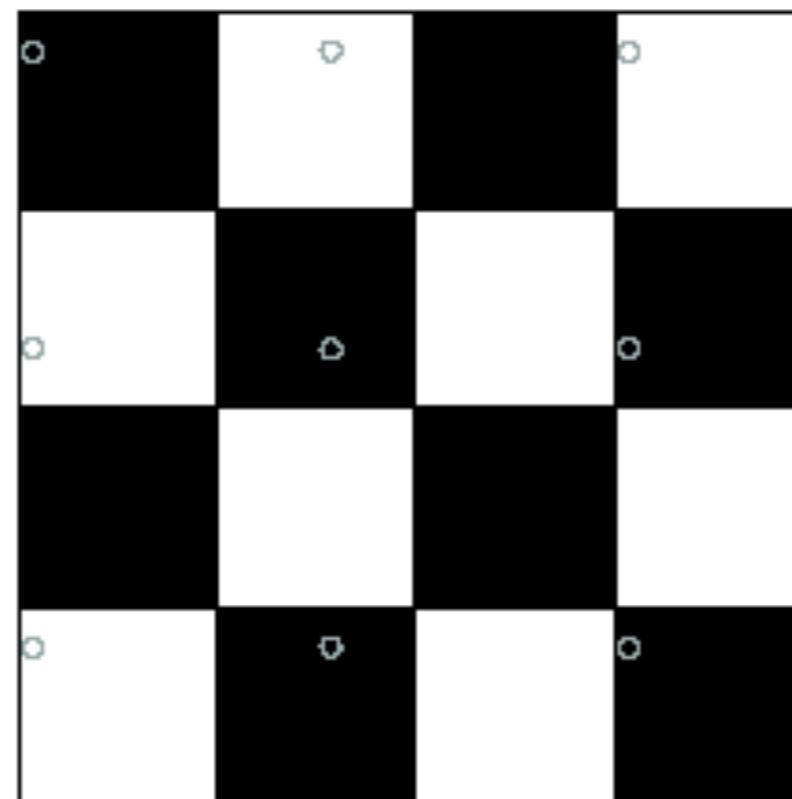
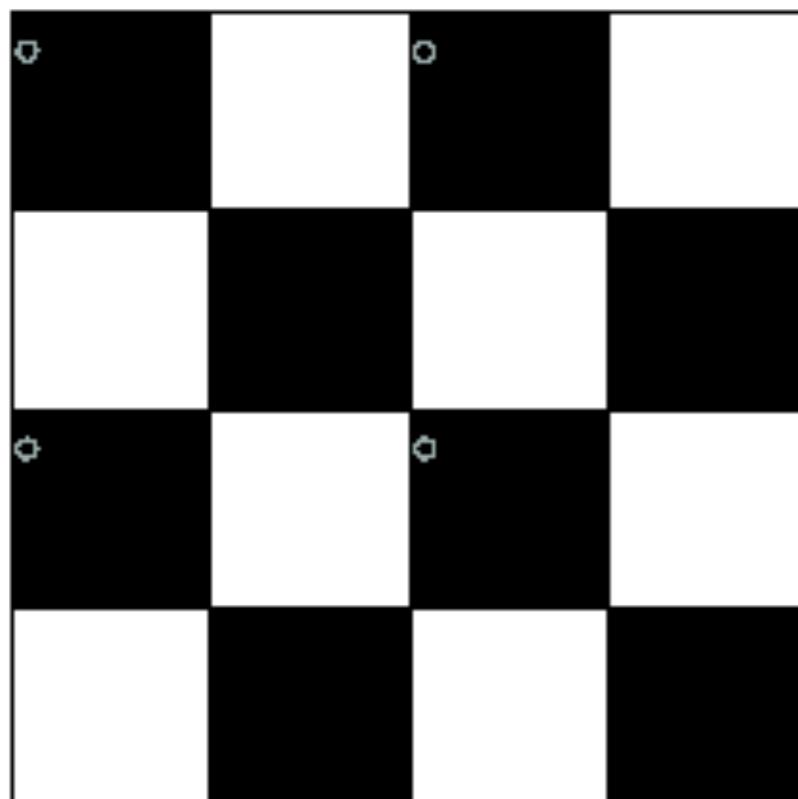
Why does this look so bad?

Sampling



Good sampling:

- Sample often or,
- Sample wisely



Bad sampling:

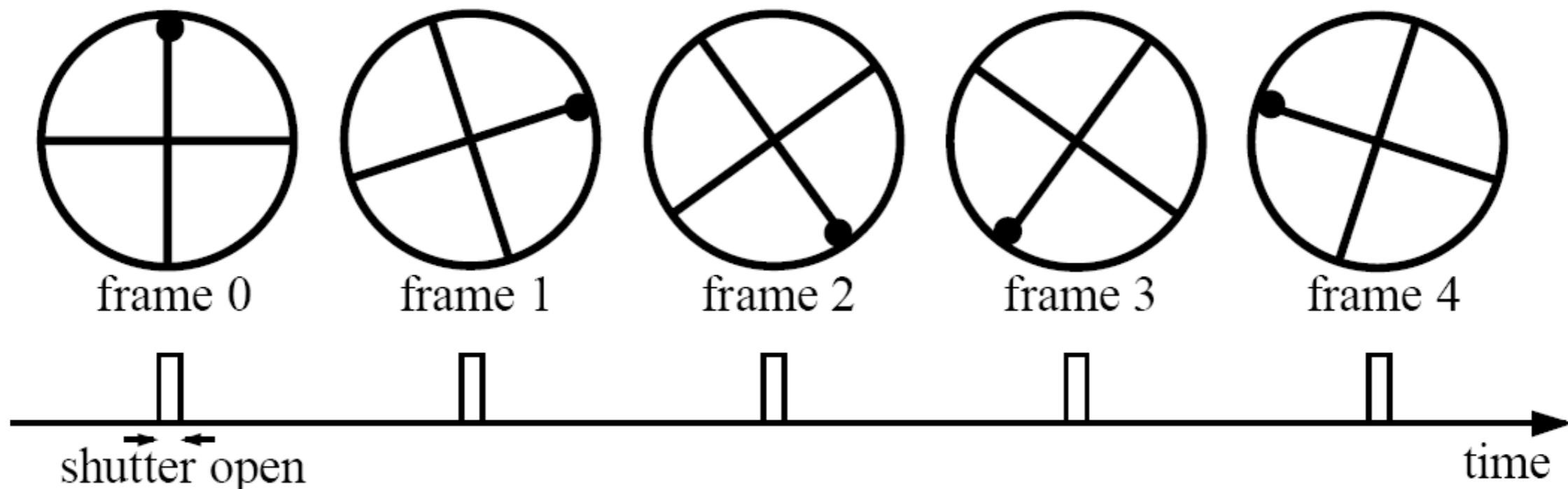
- see aliasing in action!

Really bad in video

Imagine a spoked wheel moving to the right (rotating clockwise).

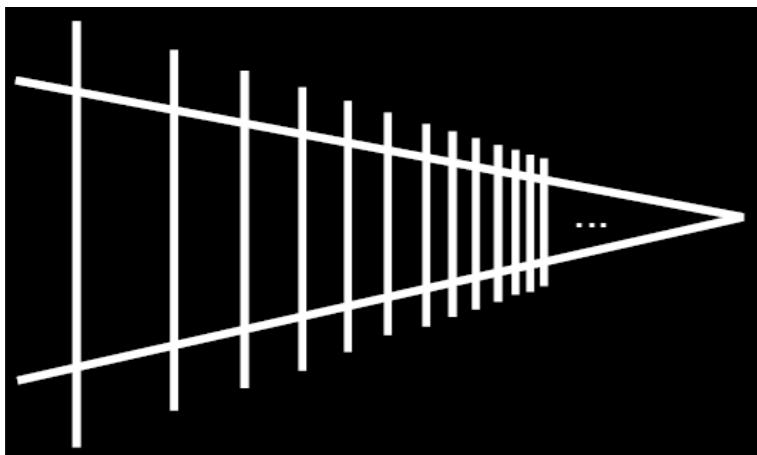
Mark wheel with dot so we can see what's happening.

If camera shutter is only open for a fraction of a frame time (frame time = 1/30 sec. for video, 1/24 sec. for film):



Without dot, wheel appears to be rotating slowly backwards!
(counterclockwise)

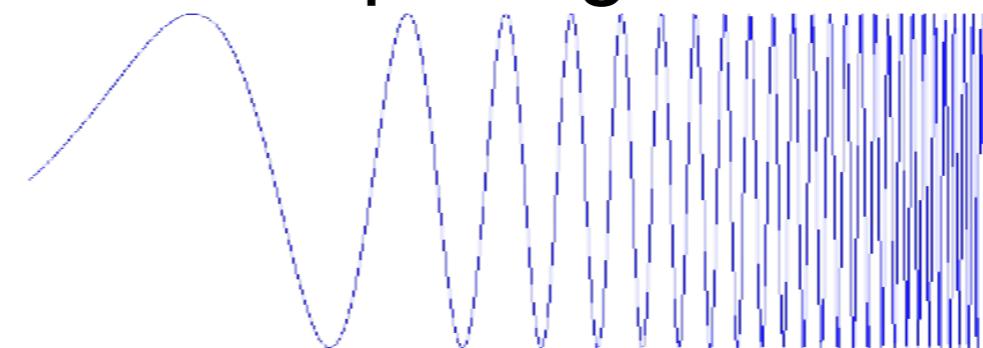
Alias: n., an assumed name



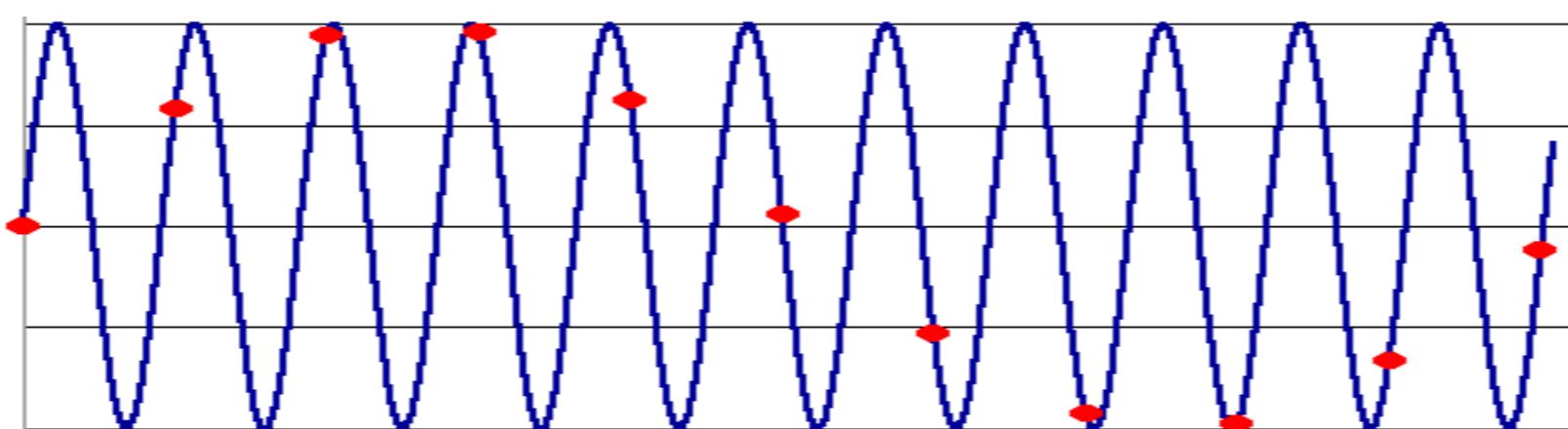
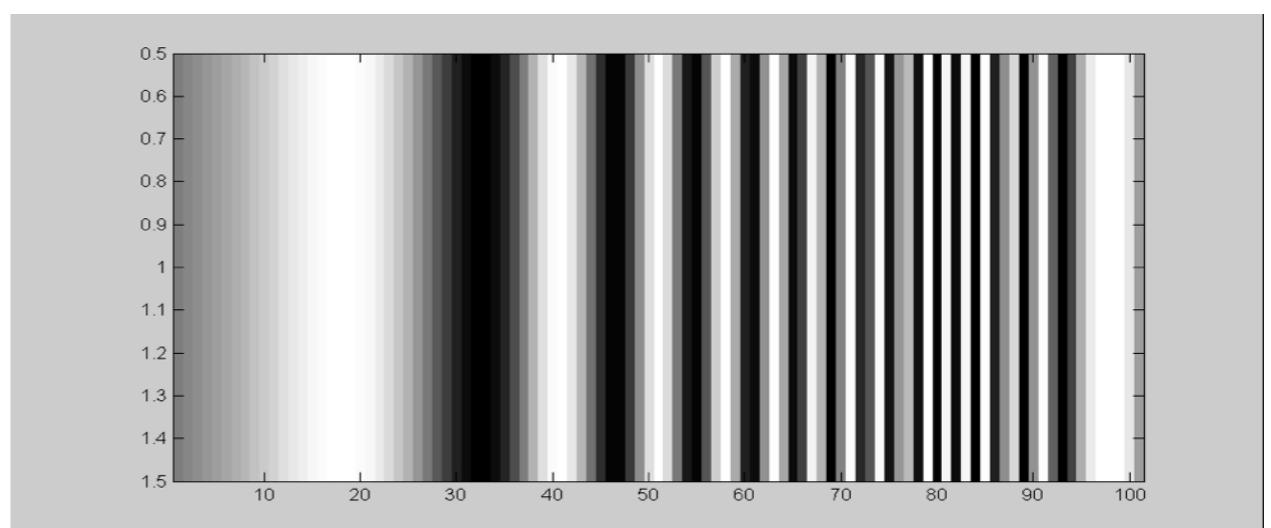
Picket fence receding
Into the distance will
produce aliasing...

WHY?

Input signal:



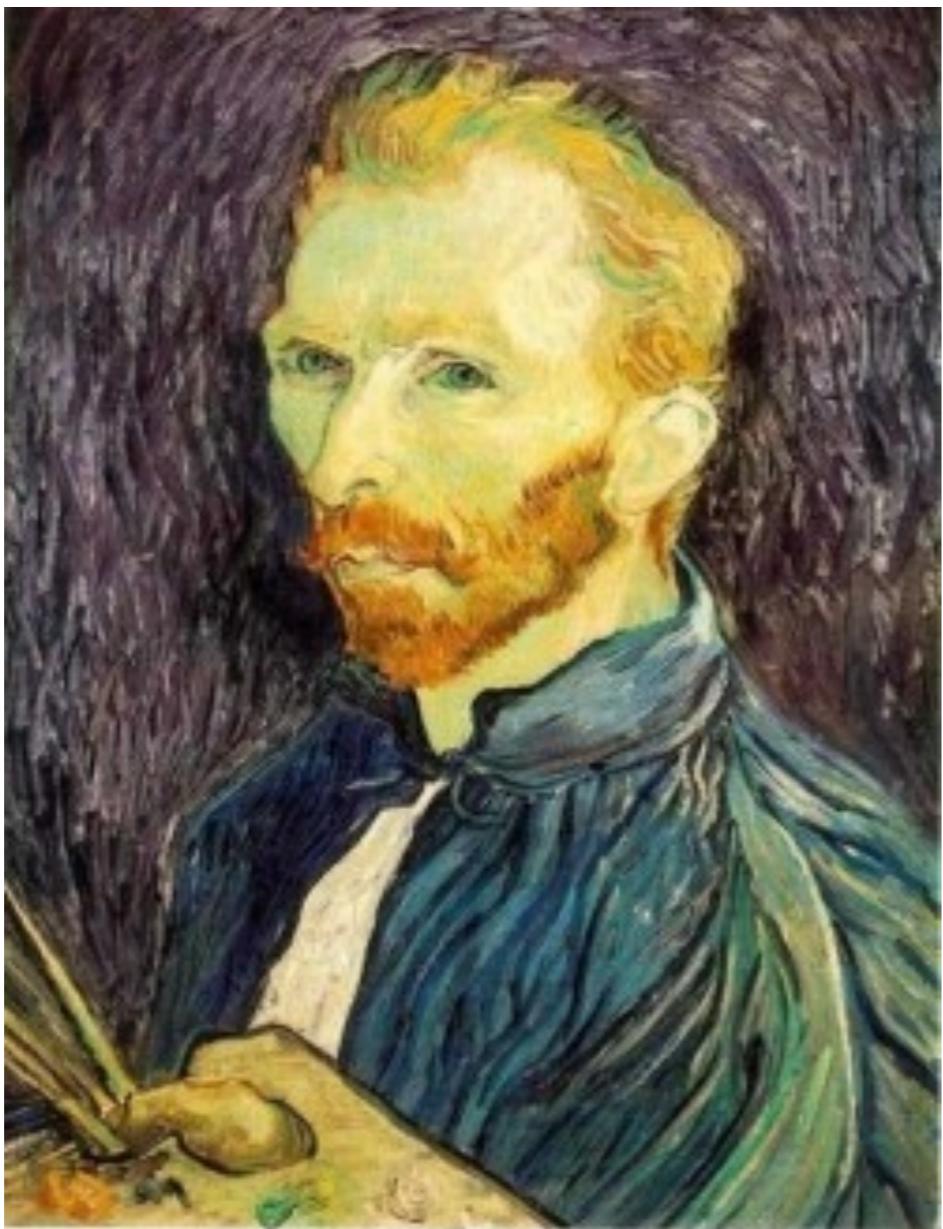
Matlab output:



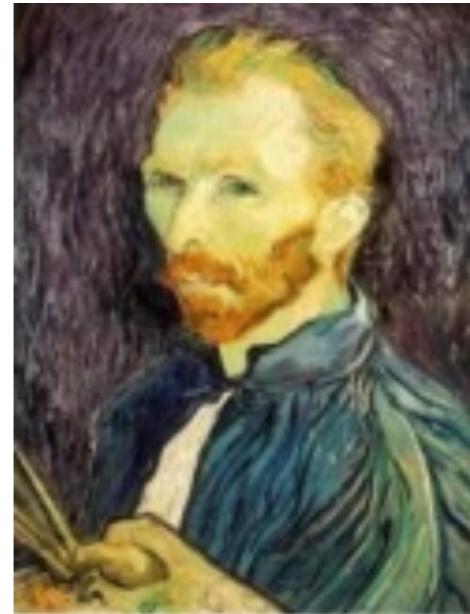
Aj-aj-aj:
Alias!

Not enough samples

Gaussian pre-filtering



Gaussian 1/2



G 1/4

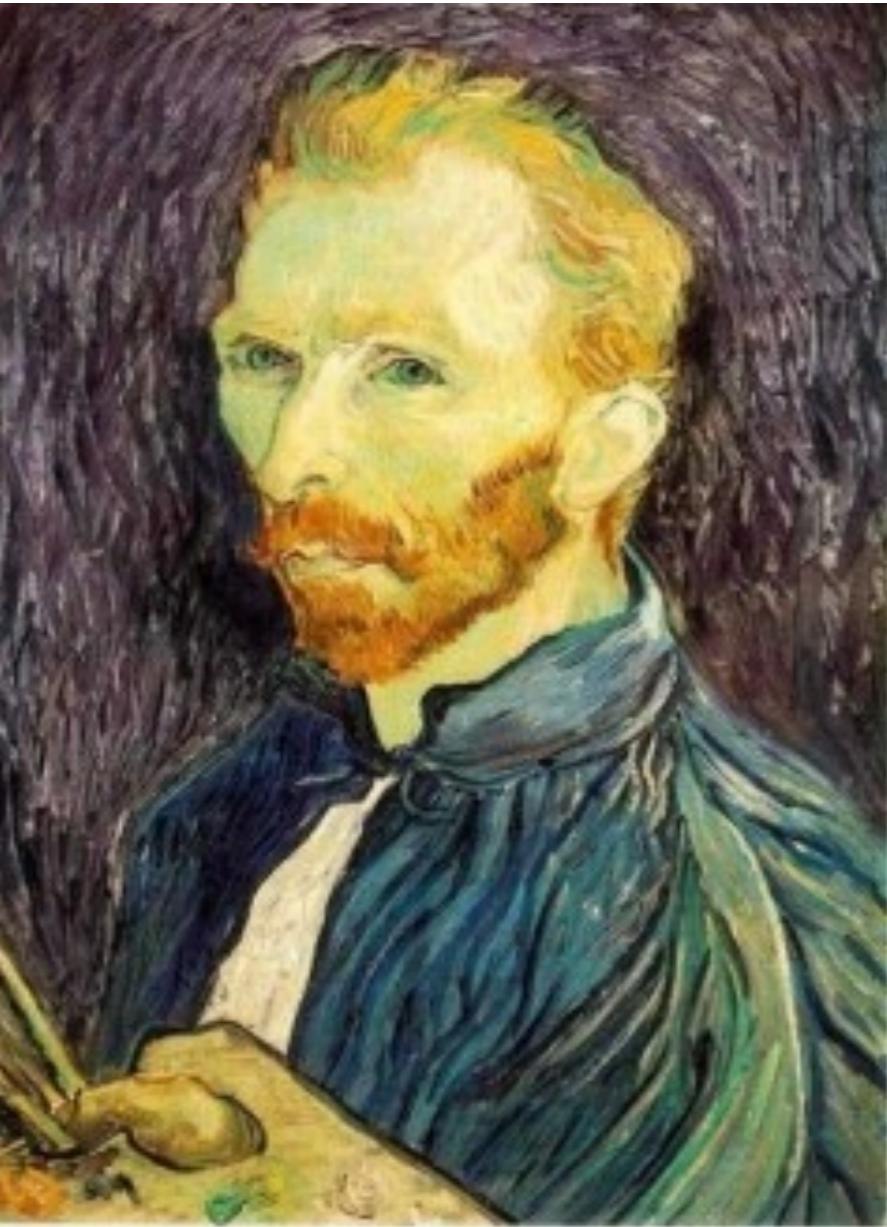


G 1/8

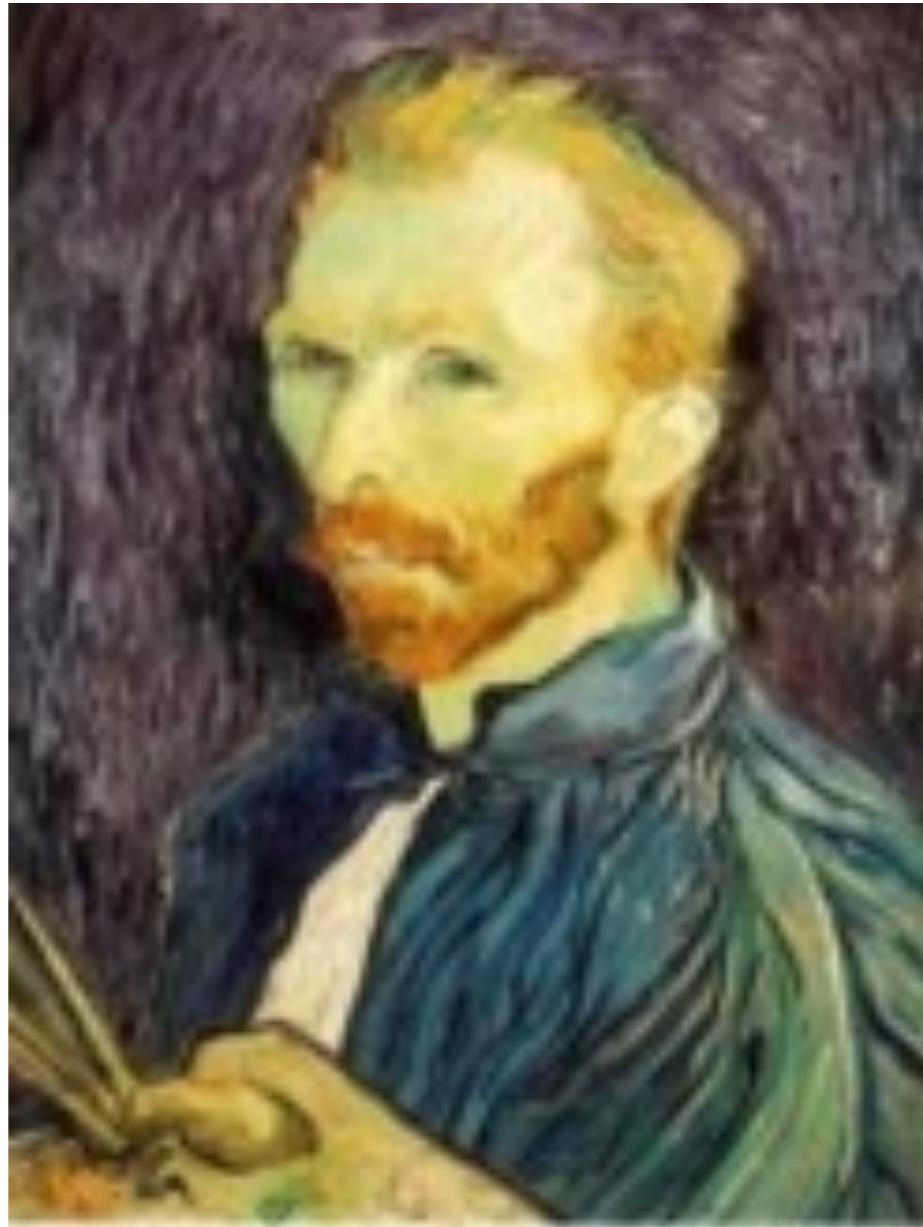
Solution: filter the image, *then* subsample

- Filter size should double for each $\frac{1}{2}$ size reduction. Why?

Subsampling with Gaussian pre-filtering



Gaussian 1/2



G 1/4

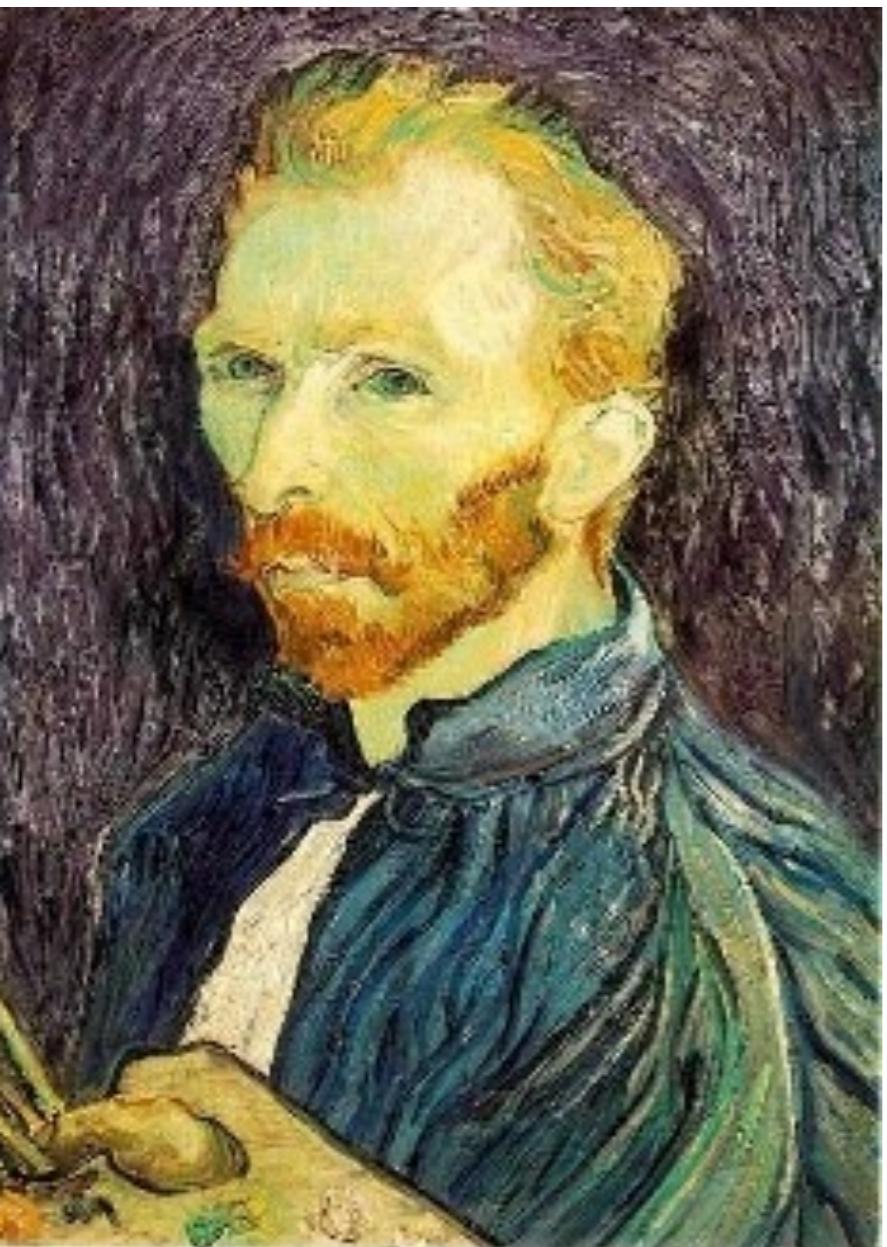


G 1/8

Solution: filter the image, *then* subsample

- Filter size should double for each $\frac{1}{2}$ size reduction. Why?
- How can we speed this up?

Compare with...



1/2

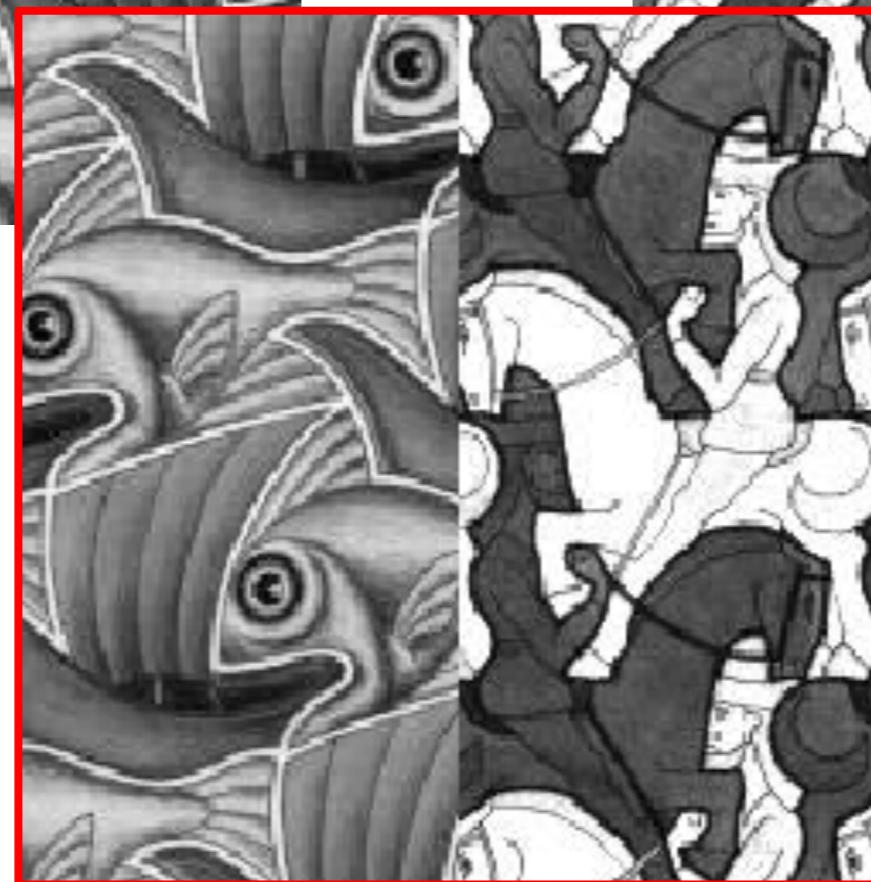
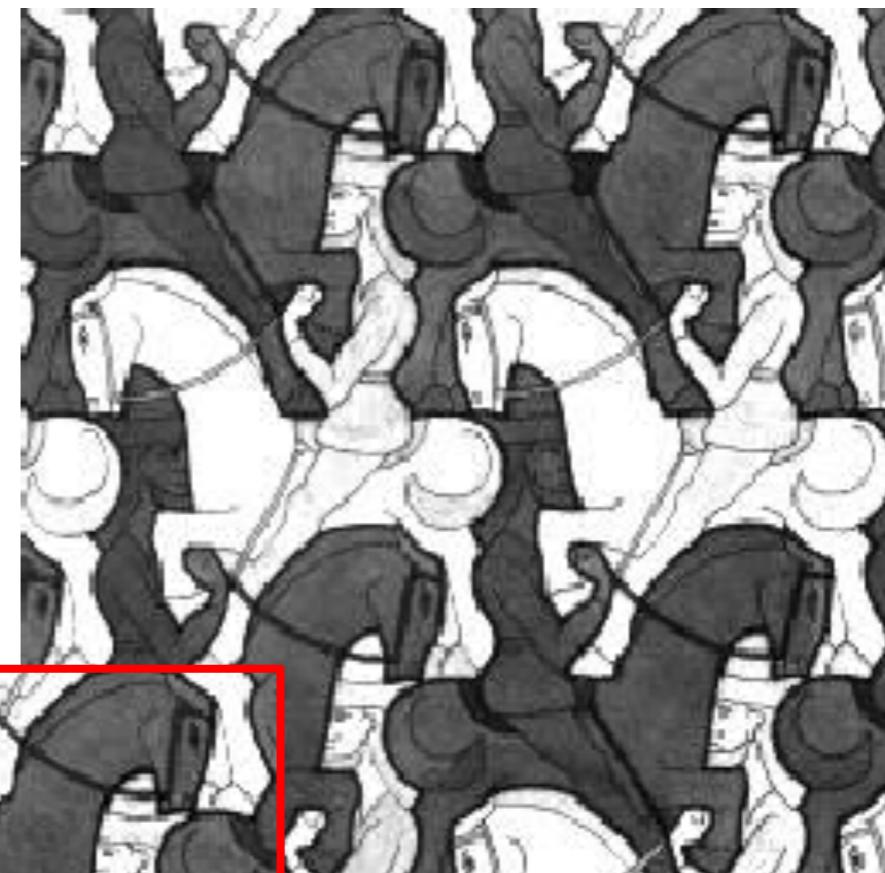
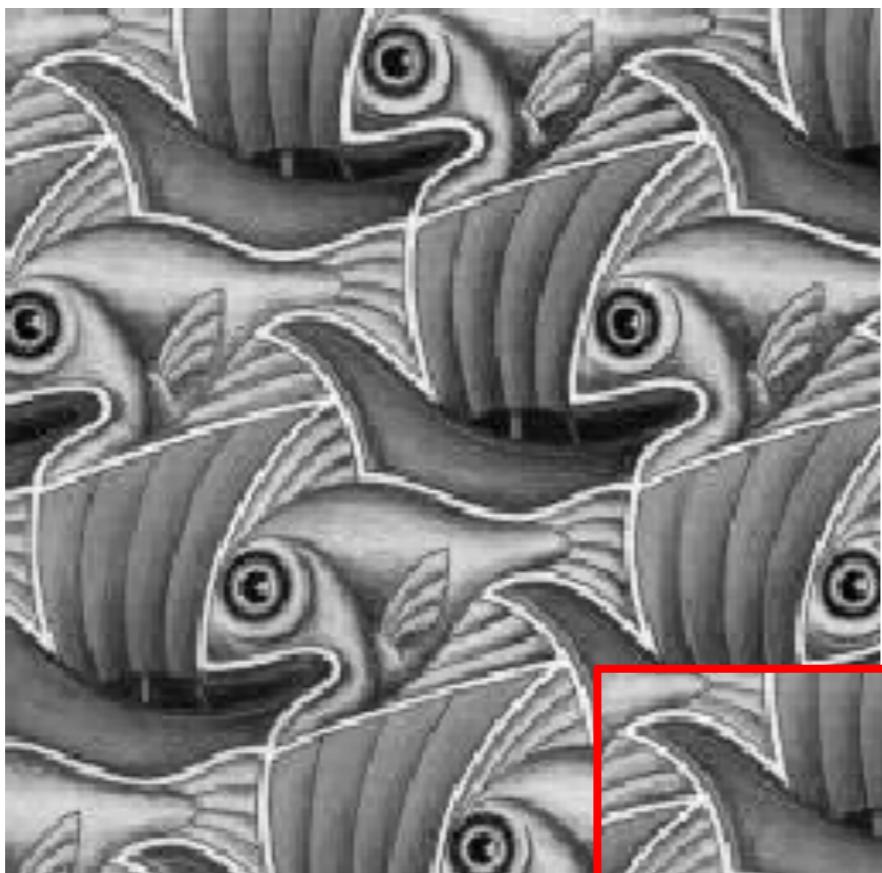


1/4 (2x zoom)



1/8 (4x zoom)

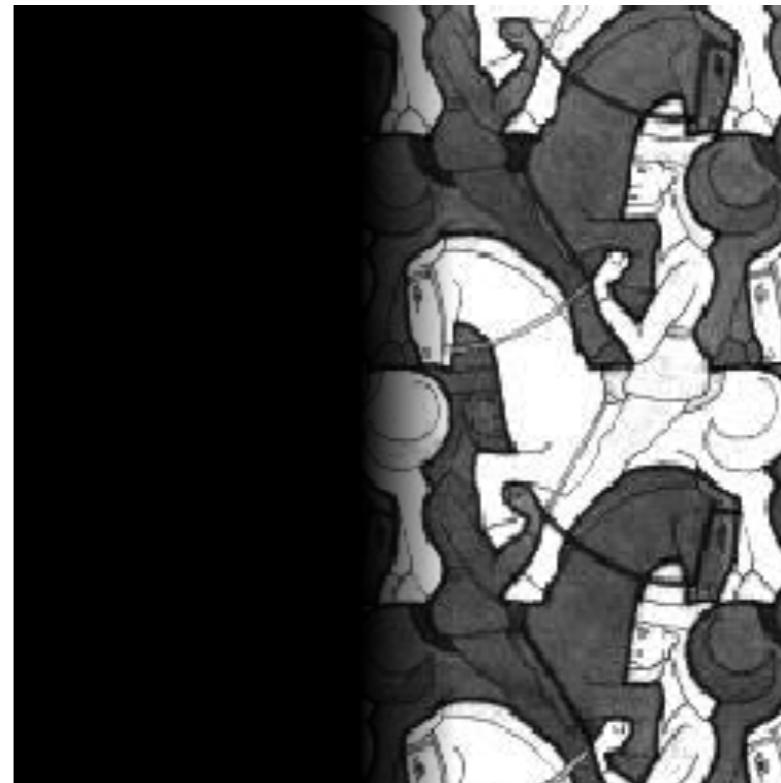
Image Blending



Feathering



+



—

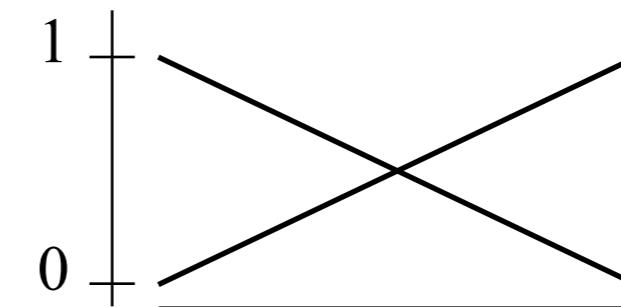
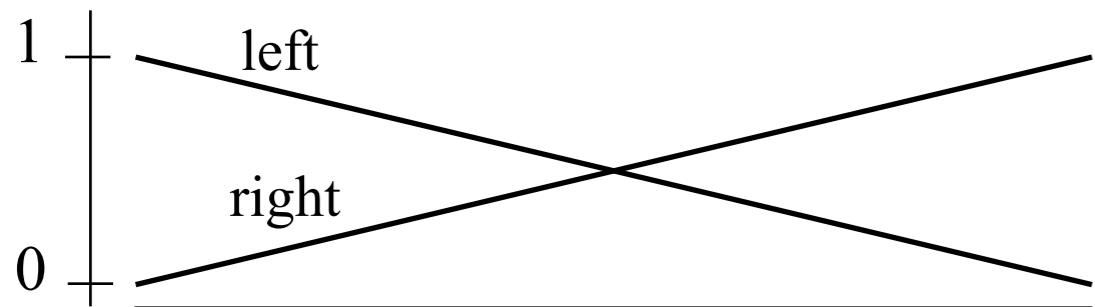
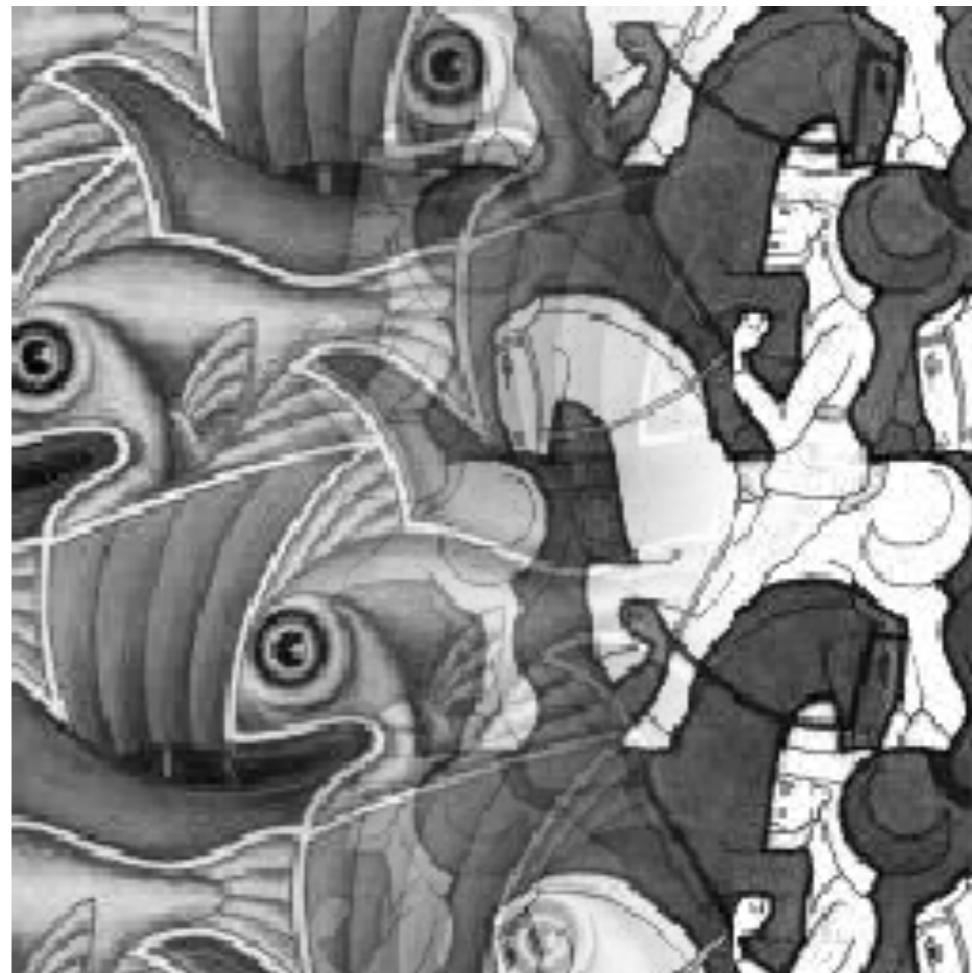


Encoding transparency

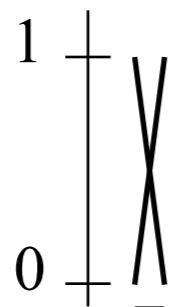
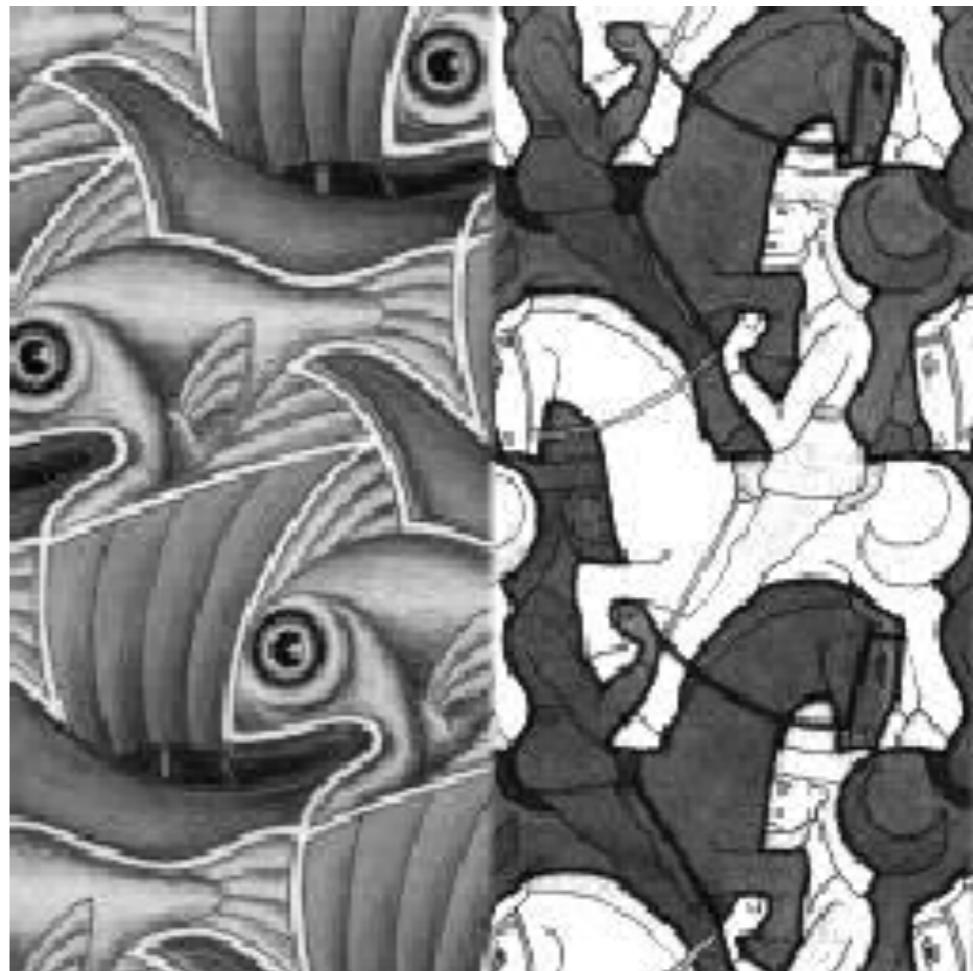
$$I(x,y) = (\alpha R, \alpha G, \alpha B, \alpha)$$

$$I_{\text{blend}} = I_{\text{left}} + I_{\text{right}}$$

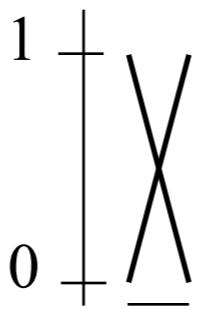
Affect of Window Size



Affect of Window Size



Good Window Size



“Optimal” Window: smooth but not ghosted

What is the Optimal Window?

To avoid seams

- window \geq size of largest prominent feature

To avoid ghosting

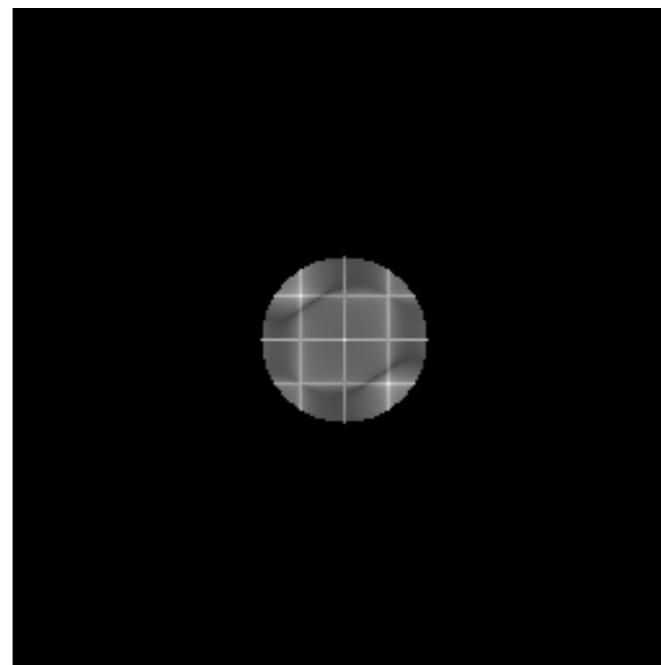
- window $\leq 2 \times$ size of smallest prominent feature

Natural to cast this in the *Fourier domain*

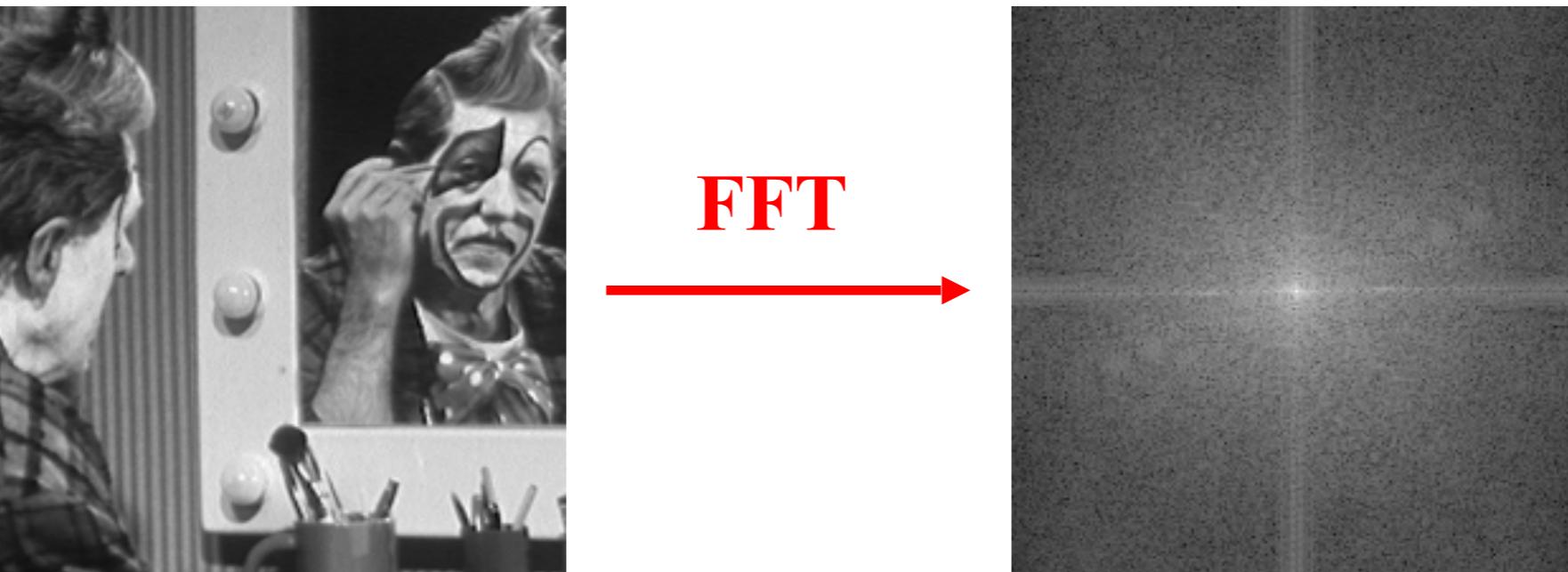
- largest frequency $\leq 2 \times$ size of smallest frequency
- image frequency content should occupy one “octave” (power of two)



FFT
→

A red arrow pointing from the input image to the output spectrum, labeled with the text "FFT" above it.

What if the Frequency Spread is Wide



Idea (Burt and Adelson)

- Compute $F_{\text{left}} = \text{FFT}(I_{\text{left}})$, $F_{\text{right}} = \text{FFT}(I_{\text{right}})$
- Decompose Fourier image into octaves (bands)
 - $F_{\text{left}} = F_{\text{left}}^1 + F_{\text{left}}^2 + \dots$
- 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*

What does blurring take away?



original

What does blurring take away?



smoothed (5x5 Gaussian)

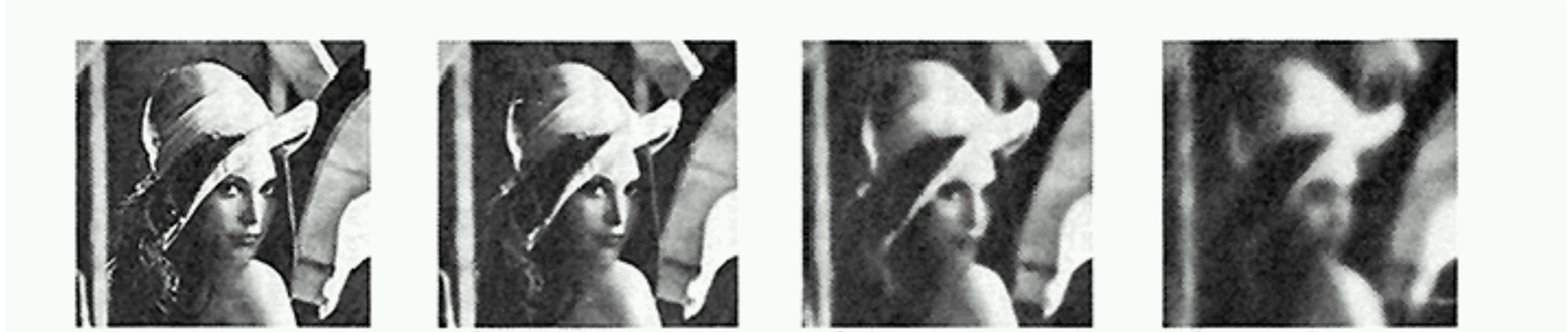
High-Pass filter



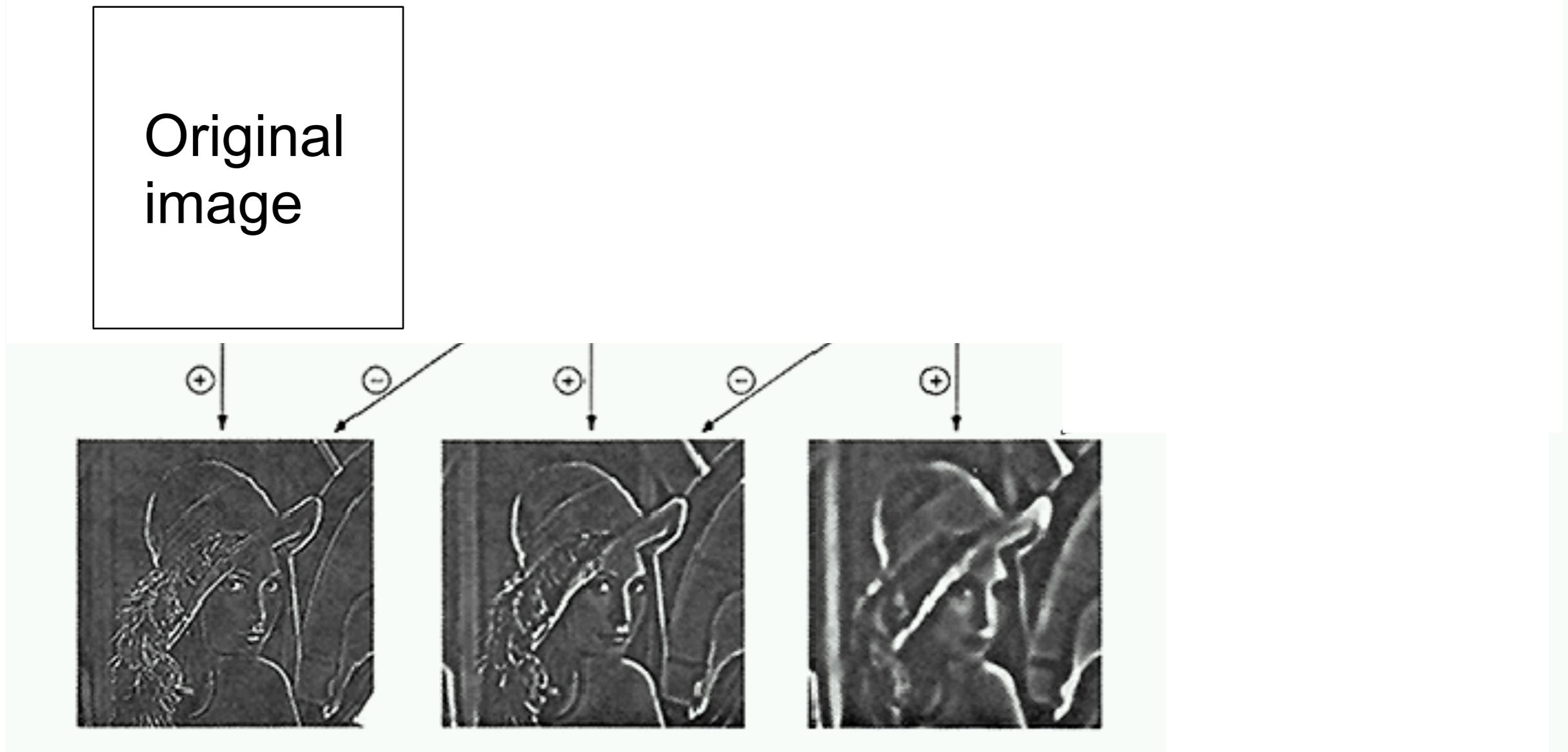
smoothed – original

Band-pass filtering

Gaussian Pyramid (low-pass images)

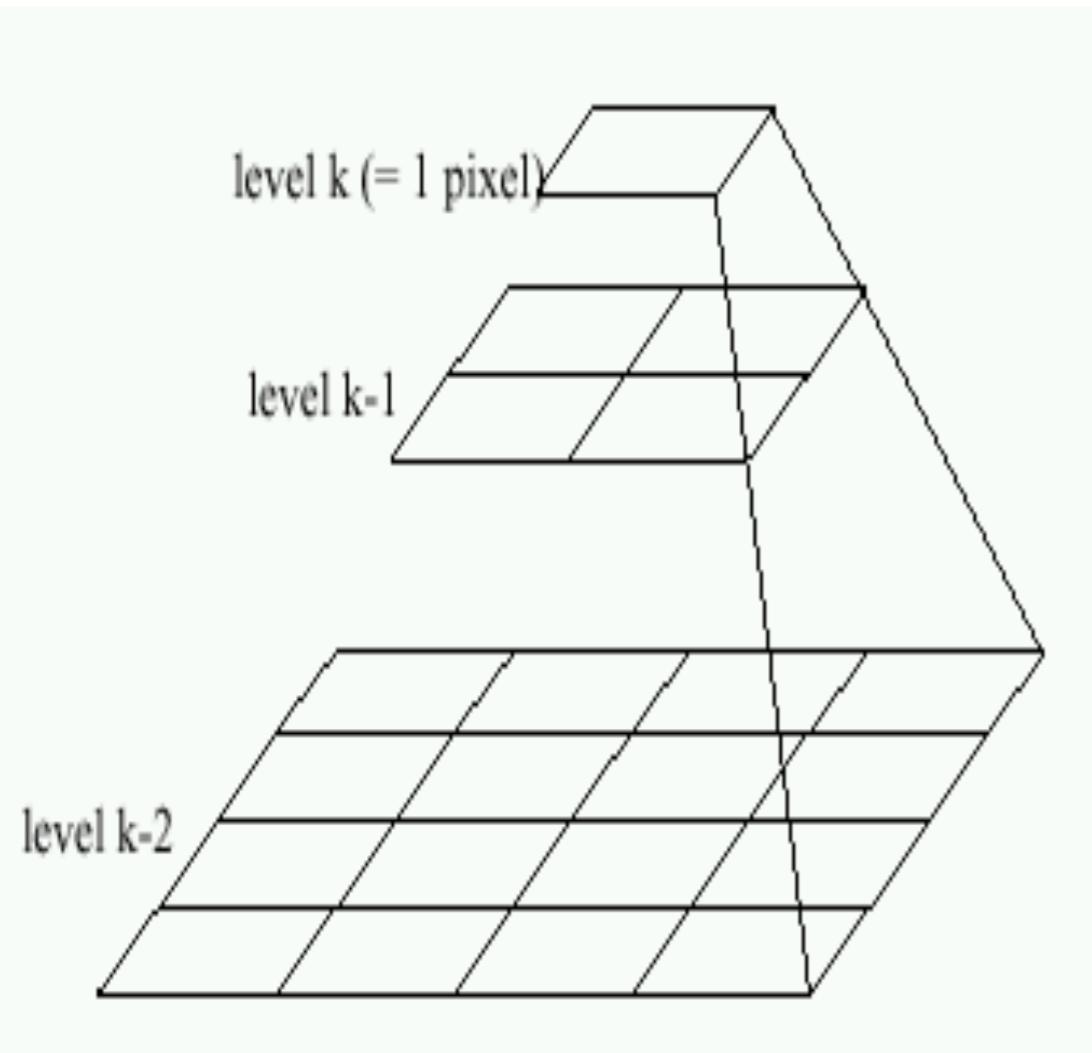


Laplacian Pyramid

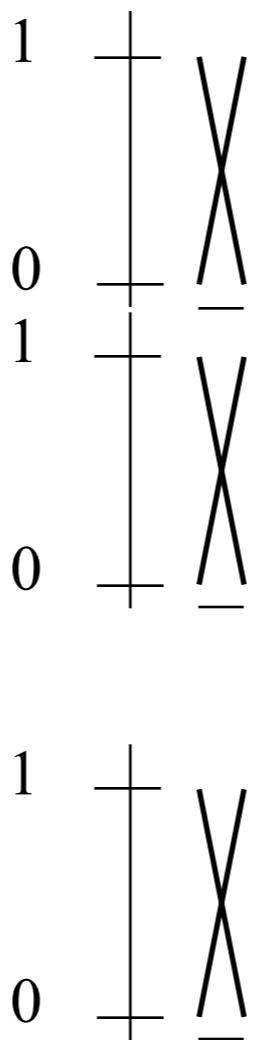


How can we reconstruct (collapse) this pyramid into the original image?

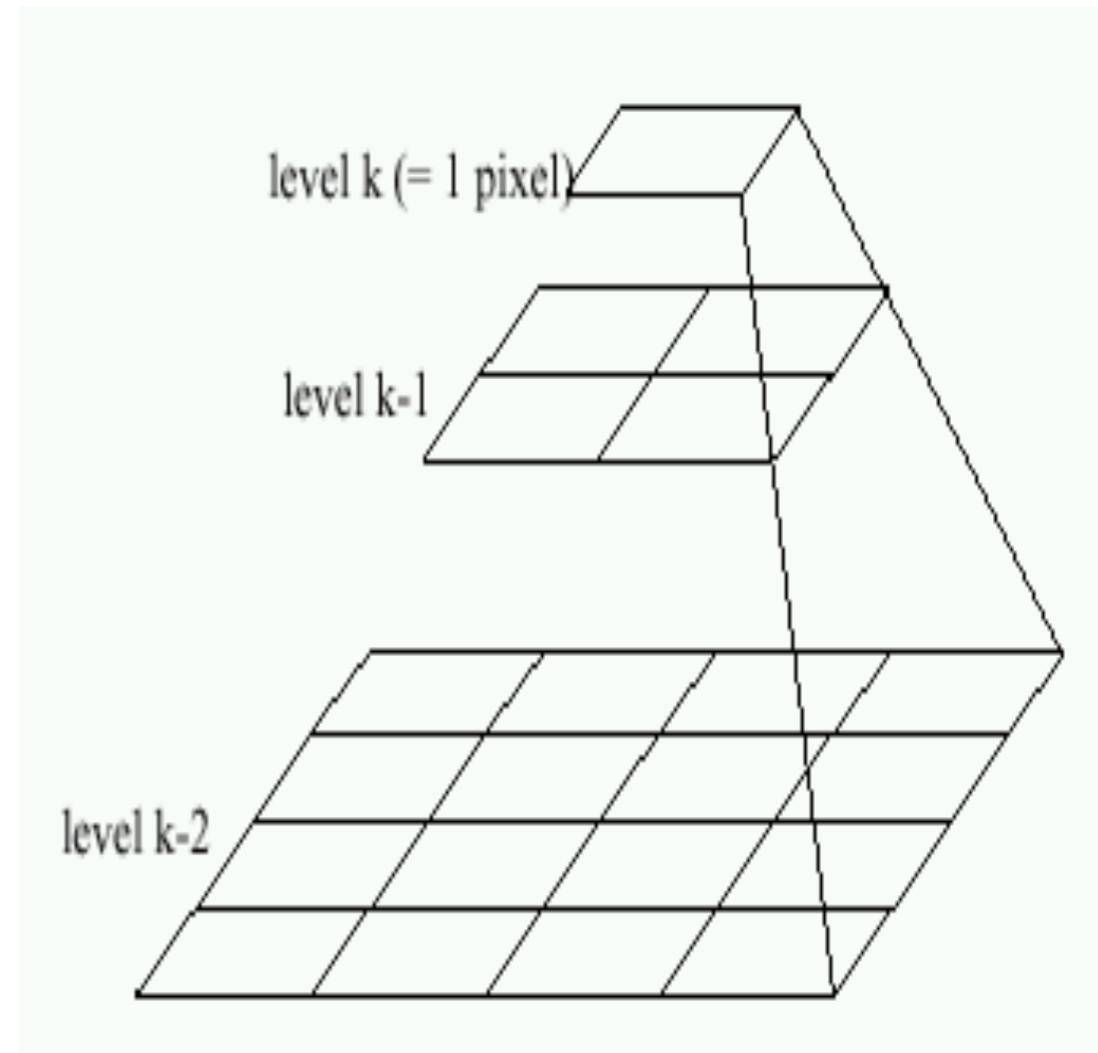
Pyramid Blending



Left pyramid

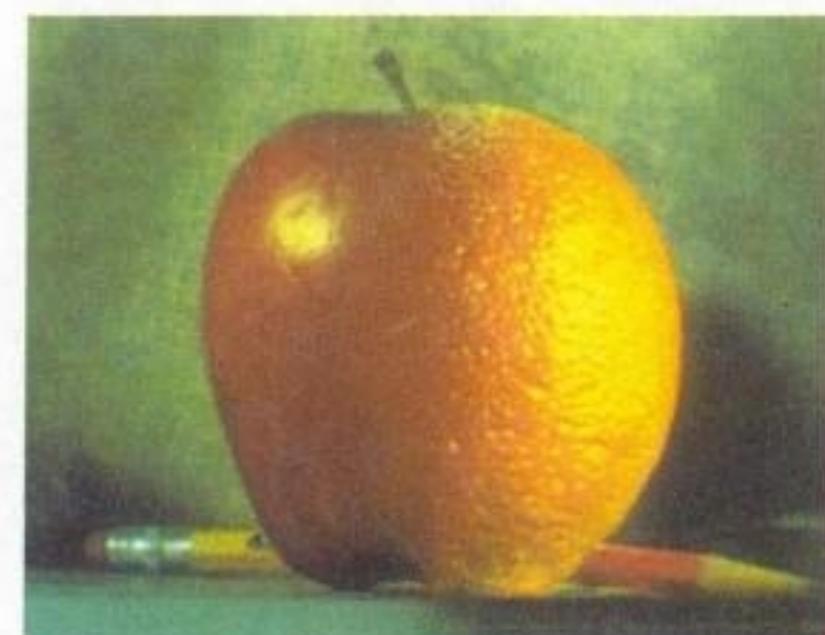
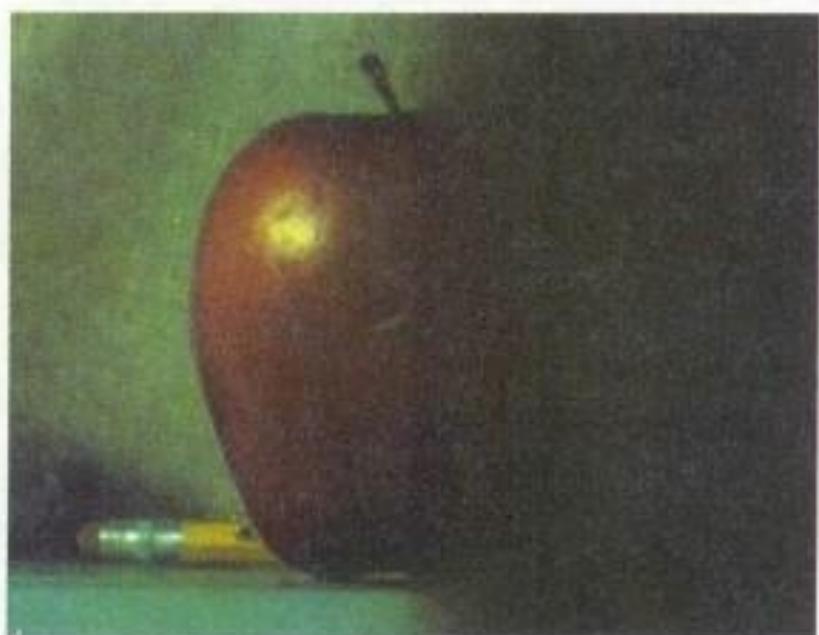
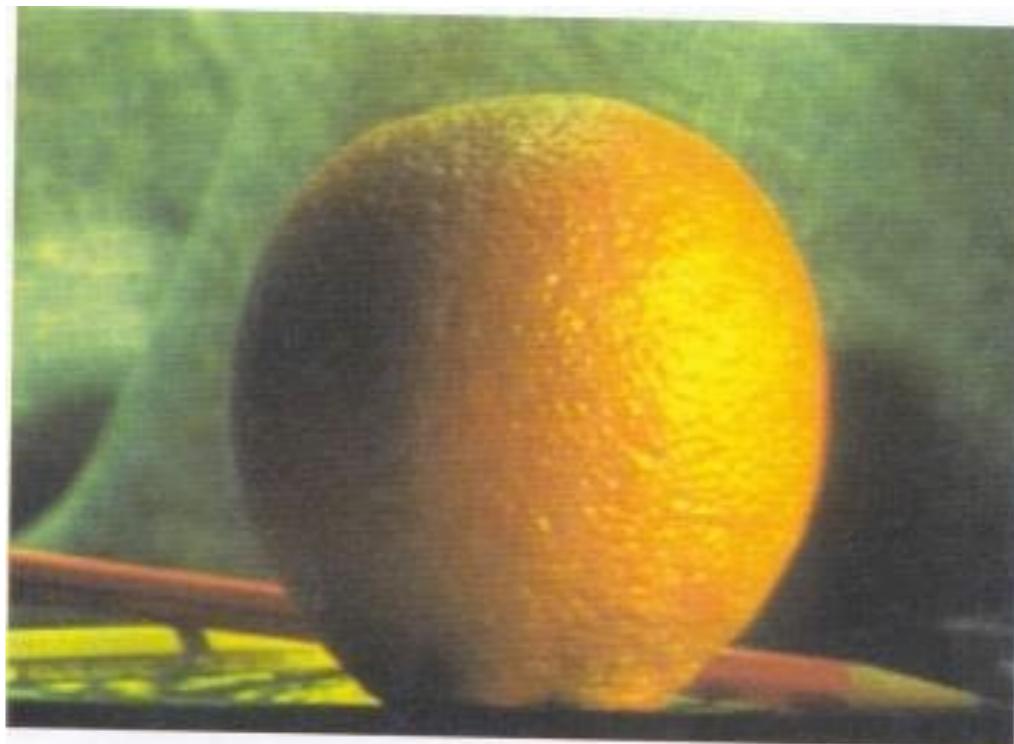
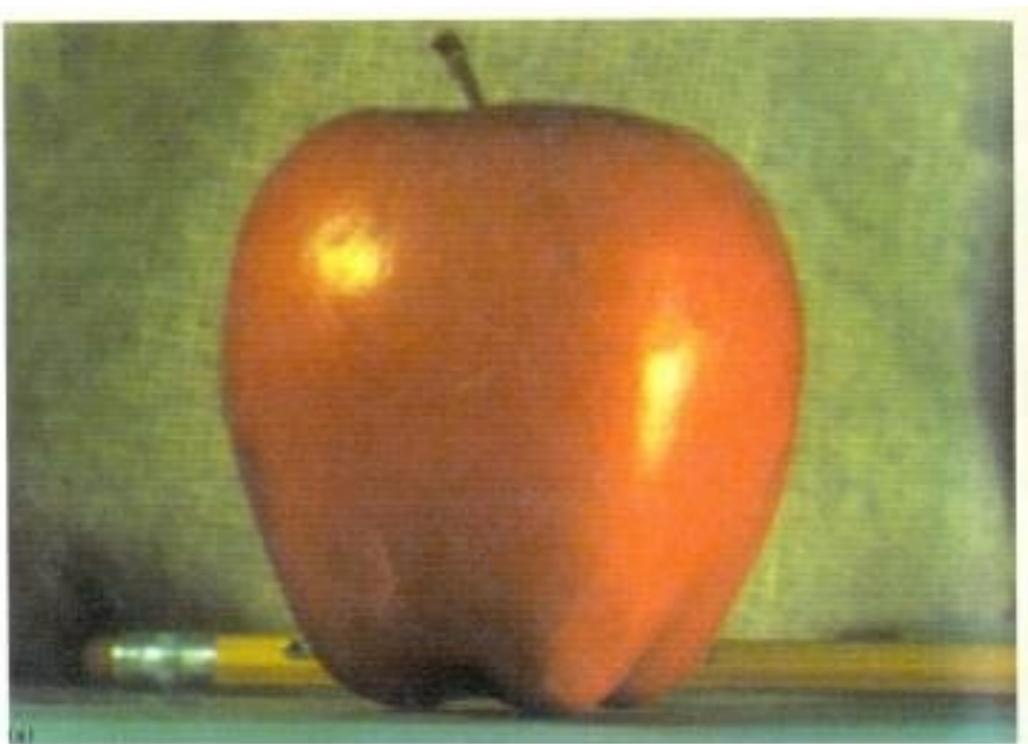


blend



Right pyramid

Pyramid Blending

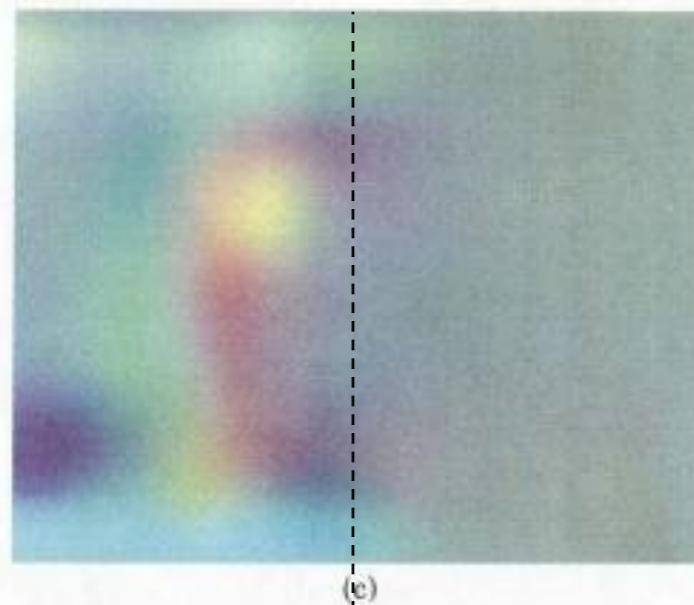


(d)

(h)

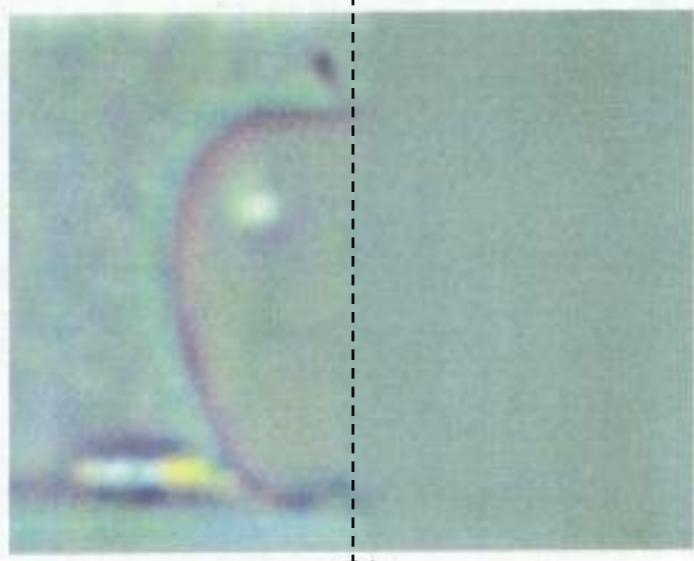
(l)

laplacian
level
4



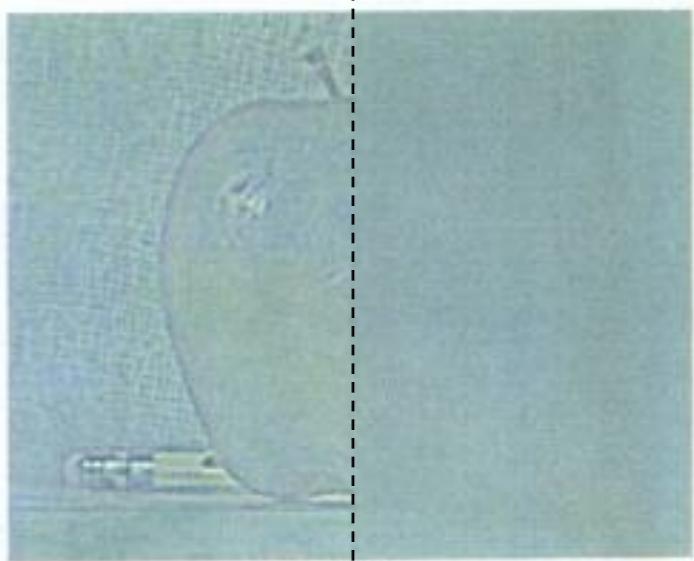
(c)

laplacian
level
2



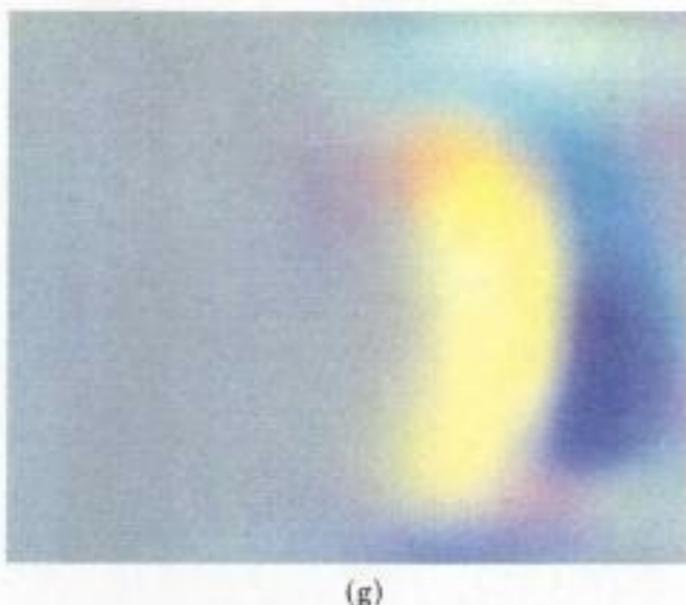
(b)

laplacian
level
0

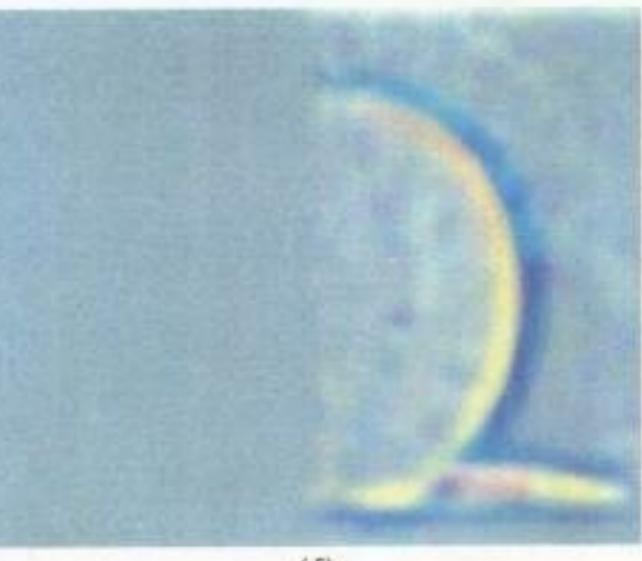


(a)

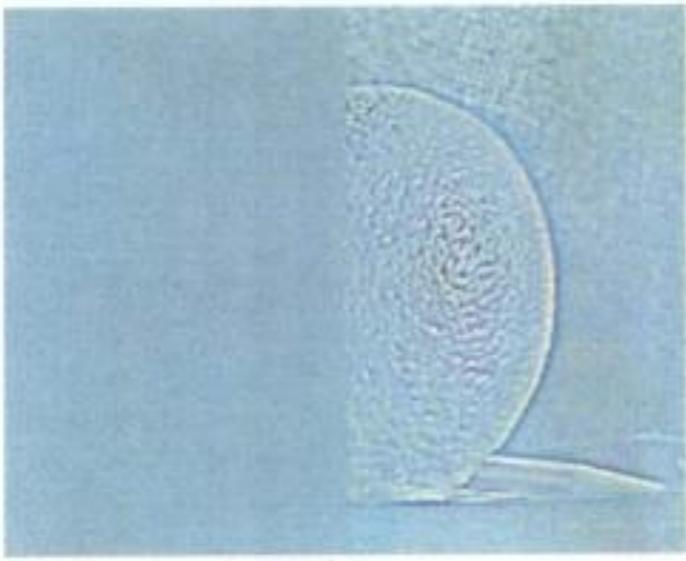
left pyramid



(g)

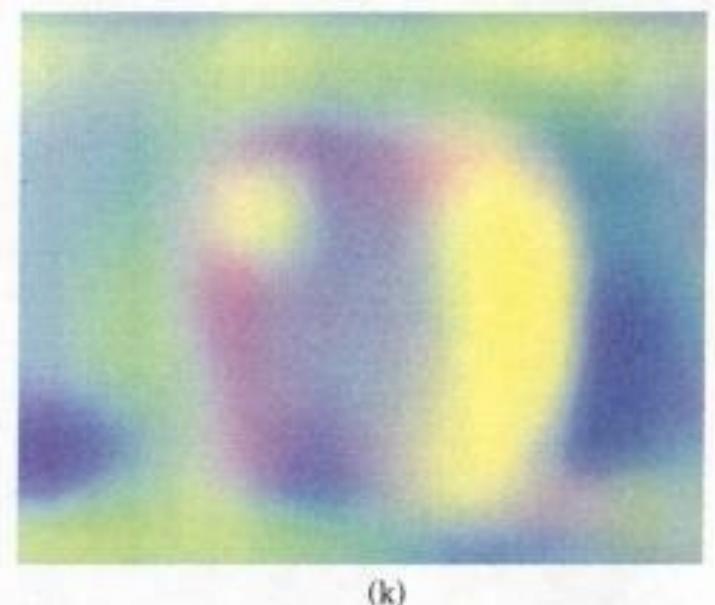


(f)

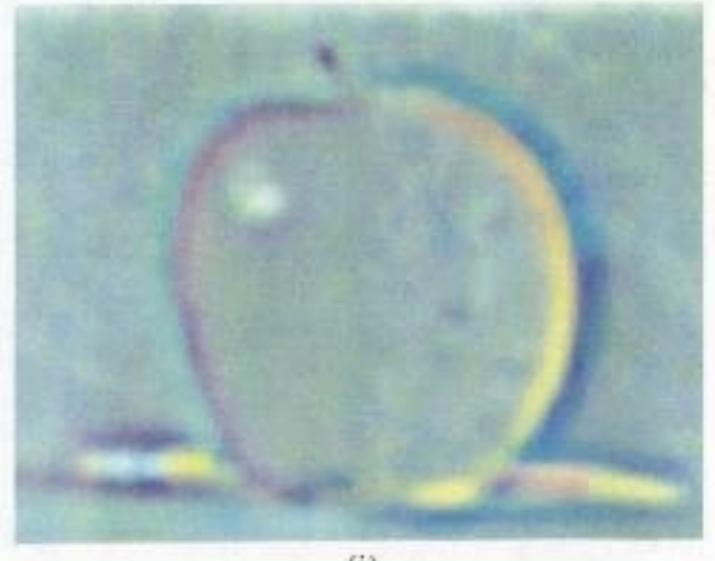


(e)

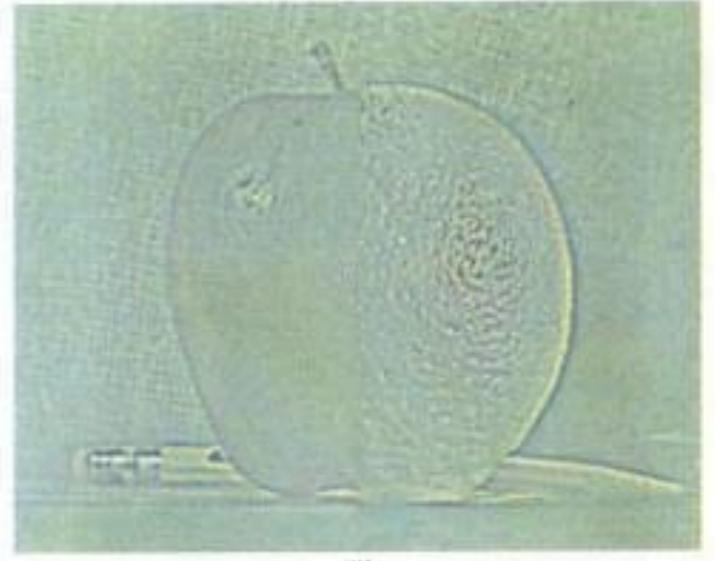
right pyramid



(k)



(j)



(i)

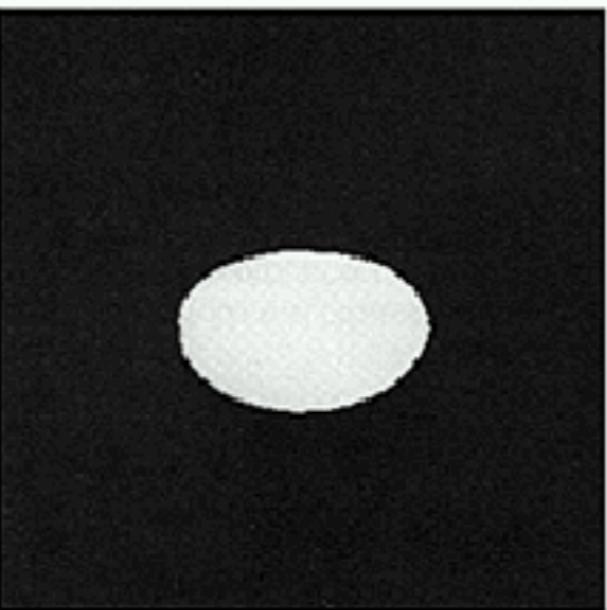
blended pyramid

Laplacian Pyramid: Blending

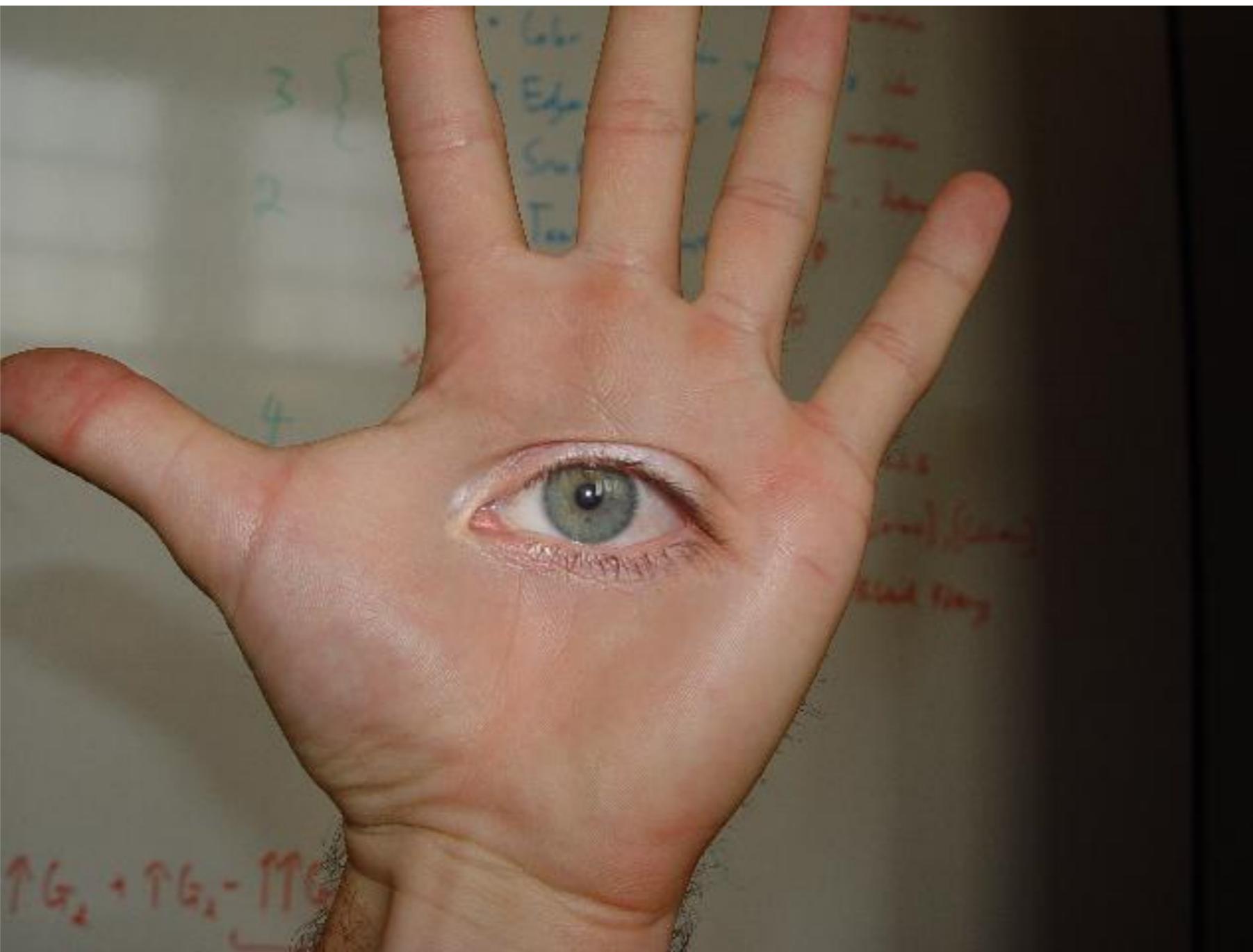
General Approach:

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

Blending Regions



Horror Photo



© prof. dmartin

Season Blending (St. Petersburg)



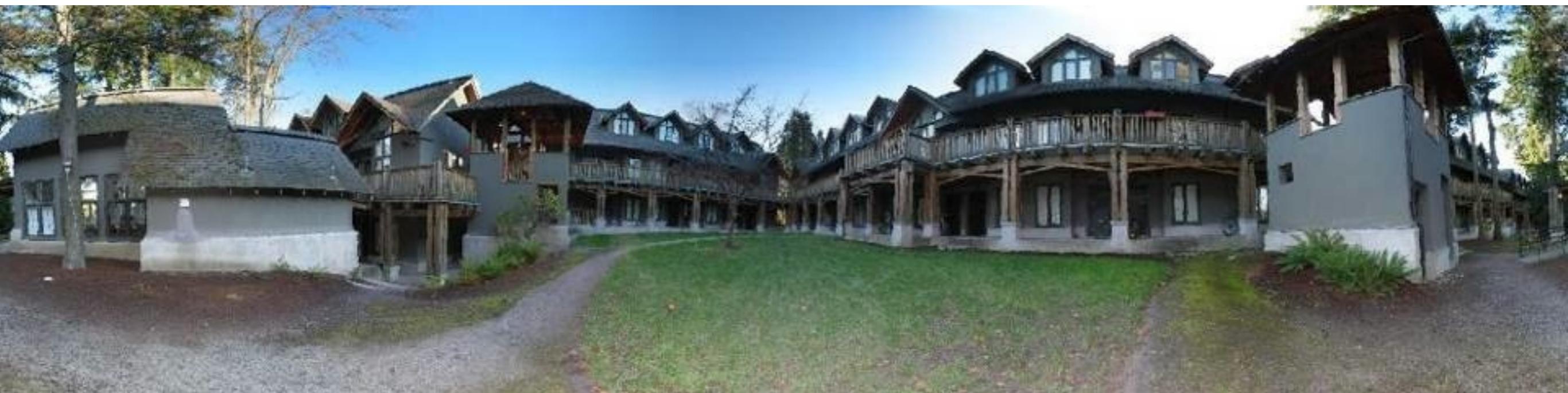
Season Blending (St. Petersburg)



Simplification: Two-band Blending

Brown & Lowe, 2003

- Only use two bands: high freq. and low freq.
- Blends low freq. smoothly
- Blend high freq. with no smoothing: use binary mask



2-band Blending



Low frequency ($\lambda > 2$ pixels)



High frequency ($\lambda < 2$ pixels)

Linear Blending



2-band Blending



Gradient Domain

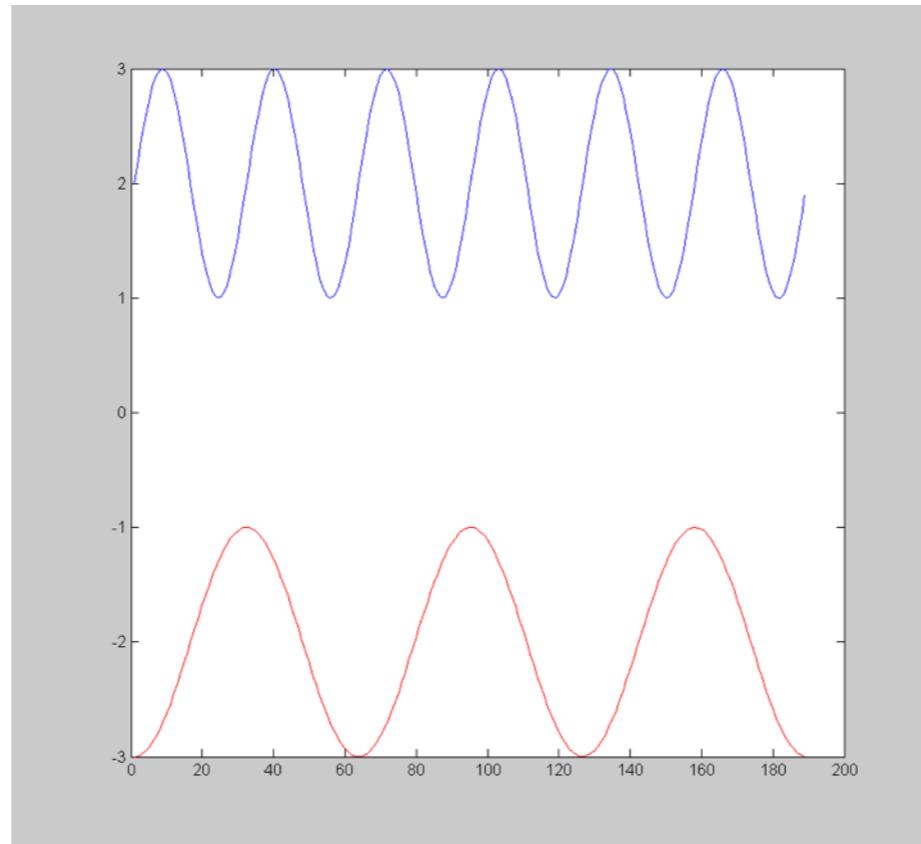
In Pyramid Blending, we decomposed our image into 2nd derivatives (Laplacian) and a low-res image

Let us now look at 1st derivatives (gradients):

- No need for low-res image
 - captures everything (up to a constant)
- Idea:
 - Differentiate
 - Blend
 - Reintegrate

Gradient Domain blending (1D)

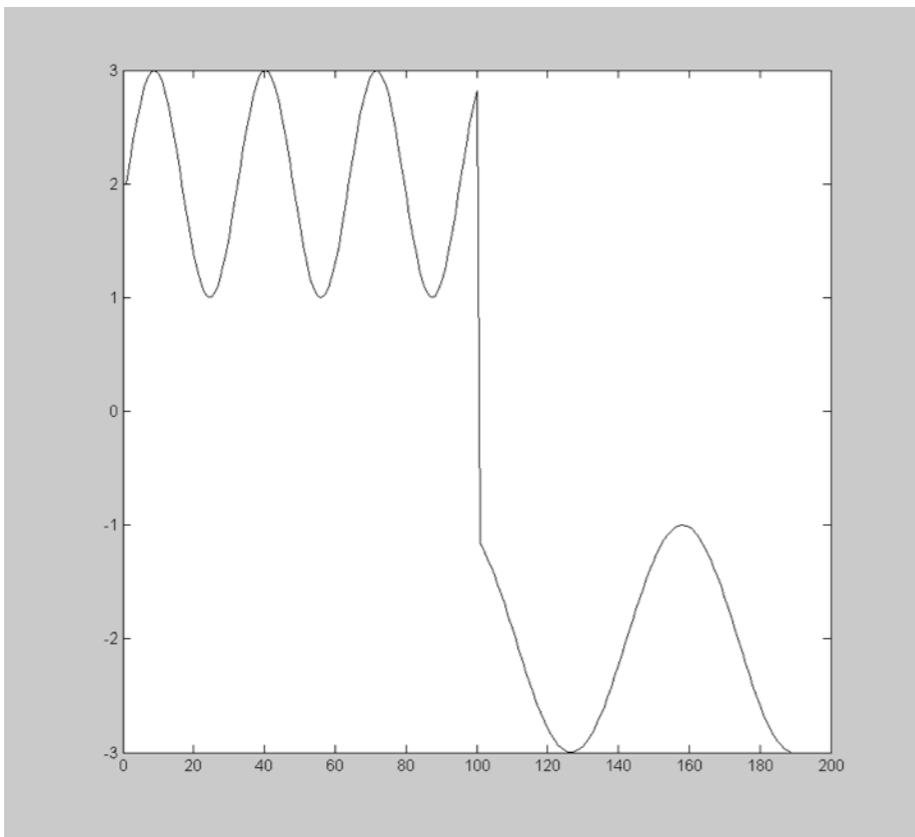
Two signals



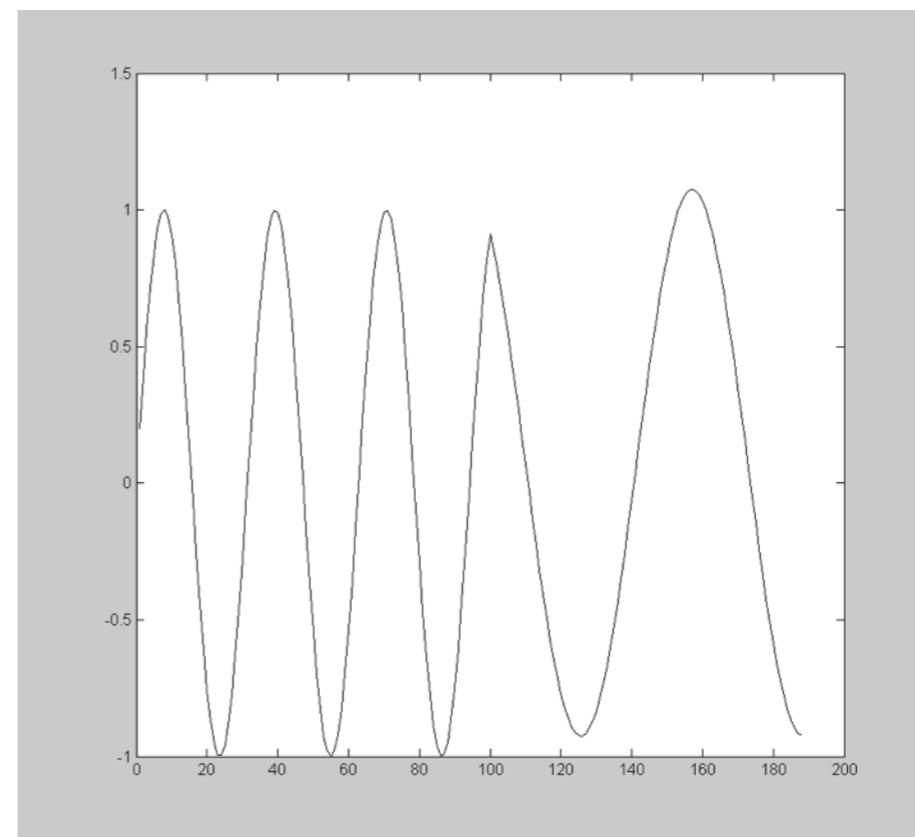
bright

dark

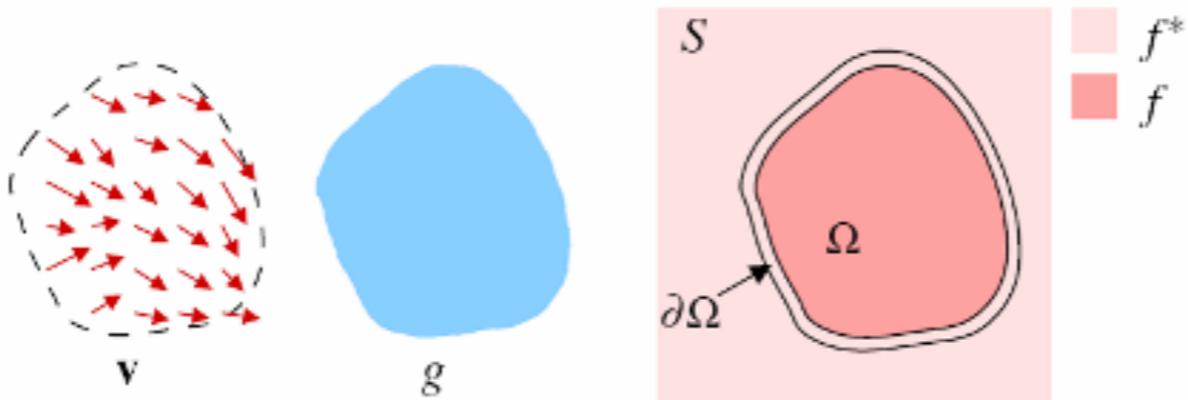
Regular blending



Blending derivatives



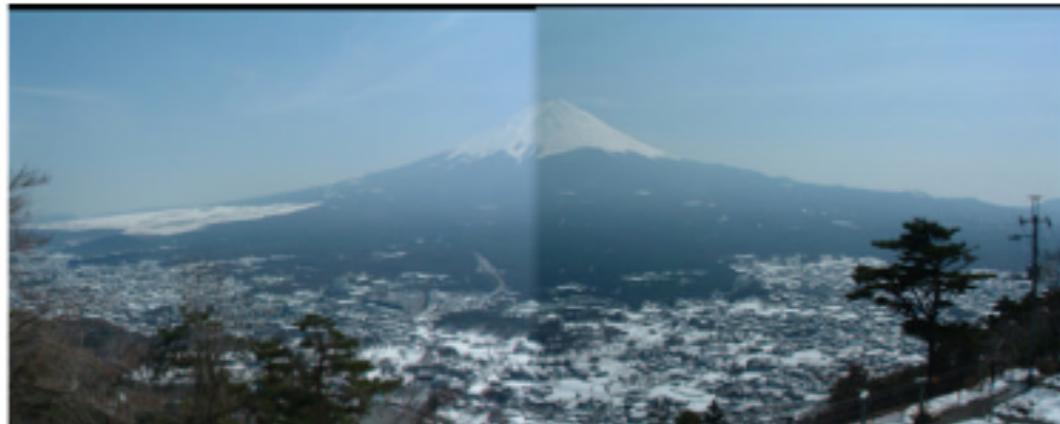
Gradient Domain Blending (2D)



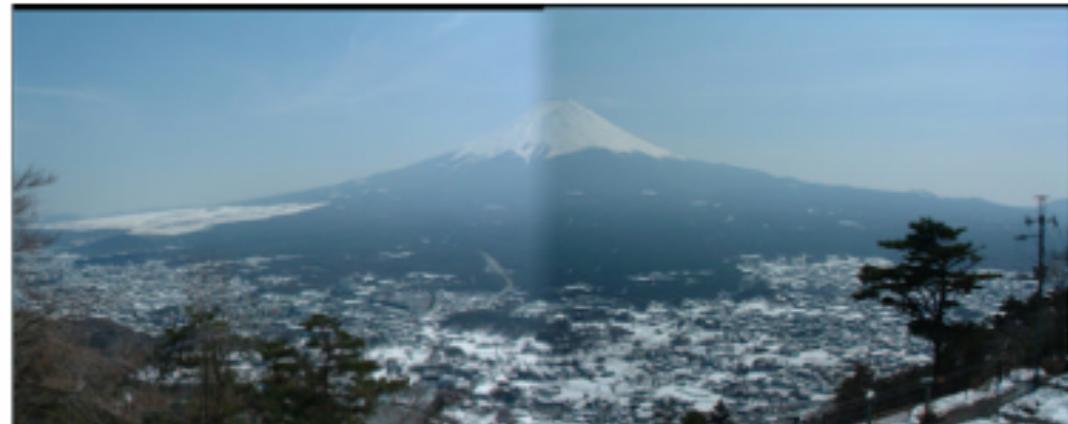
Trickier in 2D:

- Take partial derivatives dx and dy (the gradient field)
- Fiddle around with them (smooth, blend, feather, etc)
- Reintegrate
 - But now $\int(dx)$ might not equal $\int(dy)$
- Find the most agreeable solution
 - Equivalent to solving Poisson equation
 - Can use FFT, deconvolution, multigrid solvers, etc.

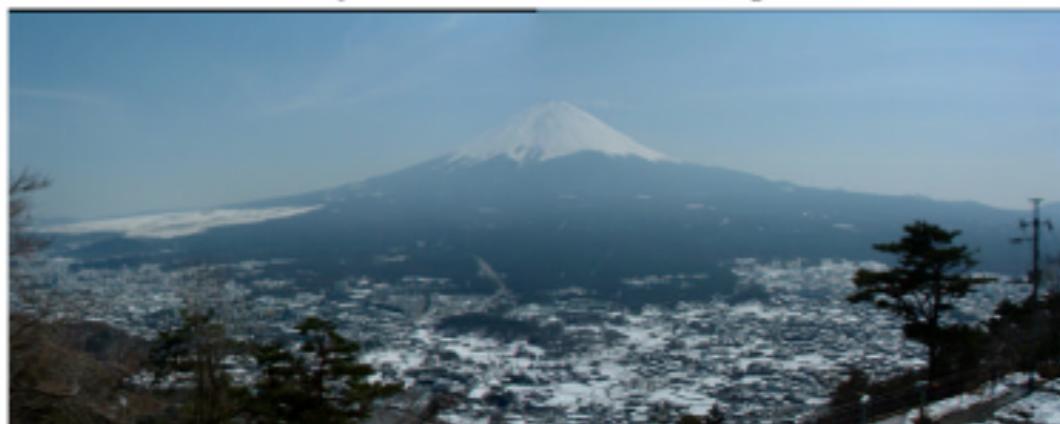
Comparisons: Levin et al, 2004



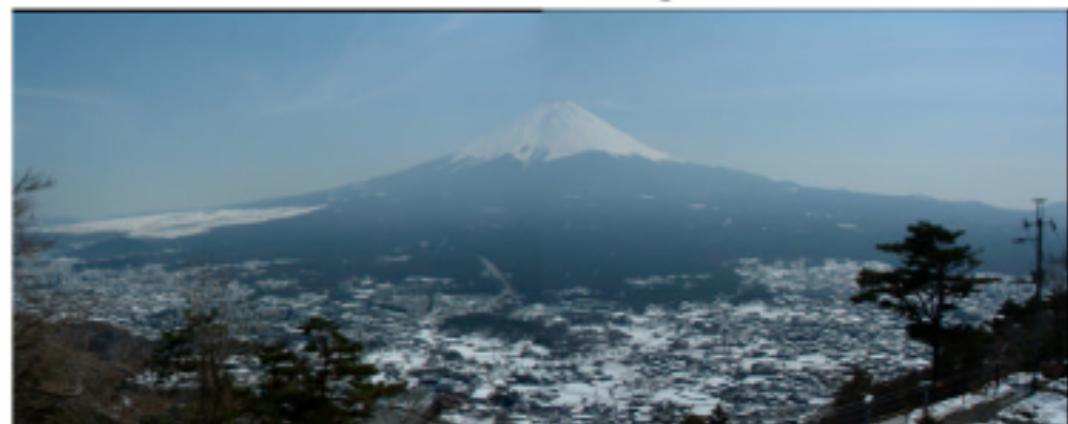
Pyramid blending



Feathering

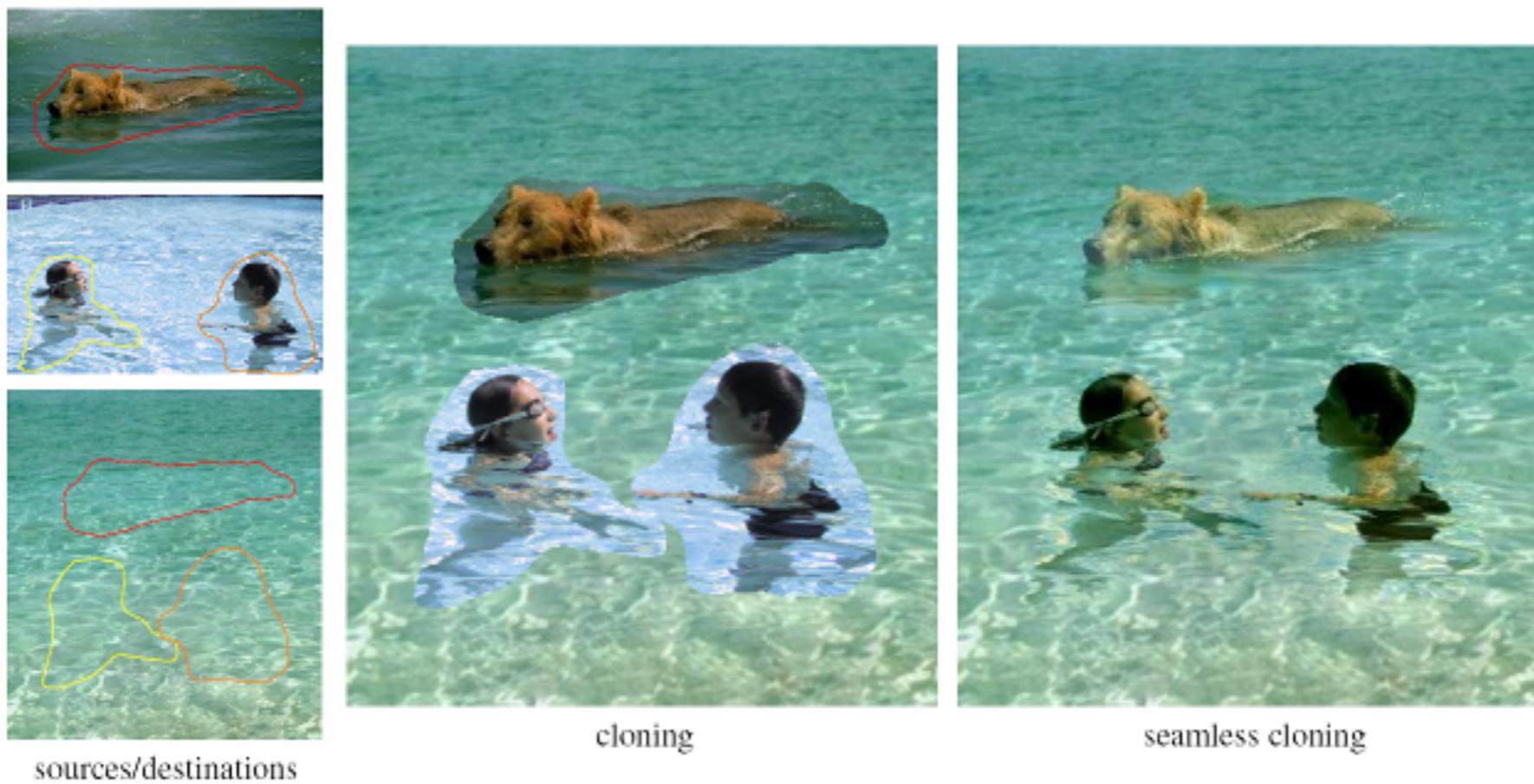
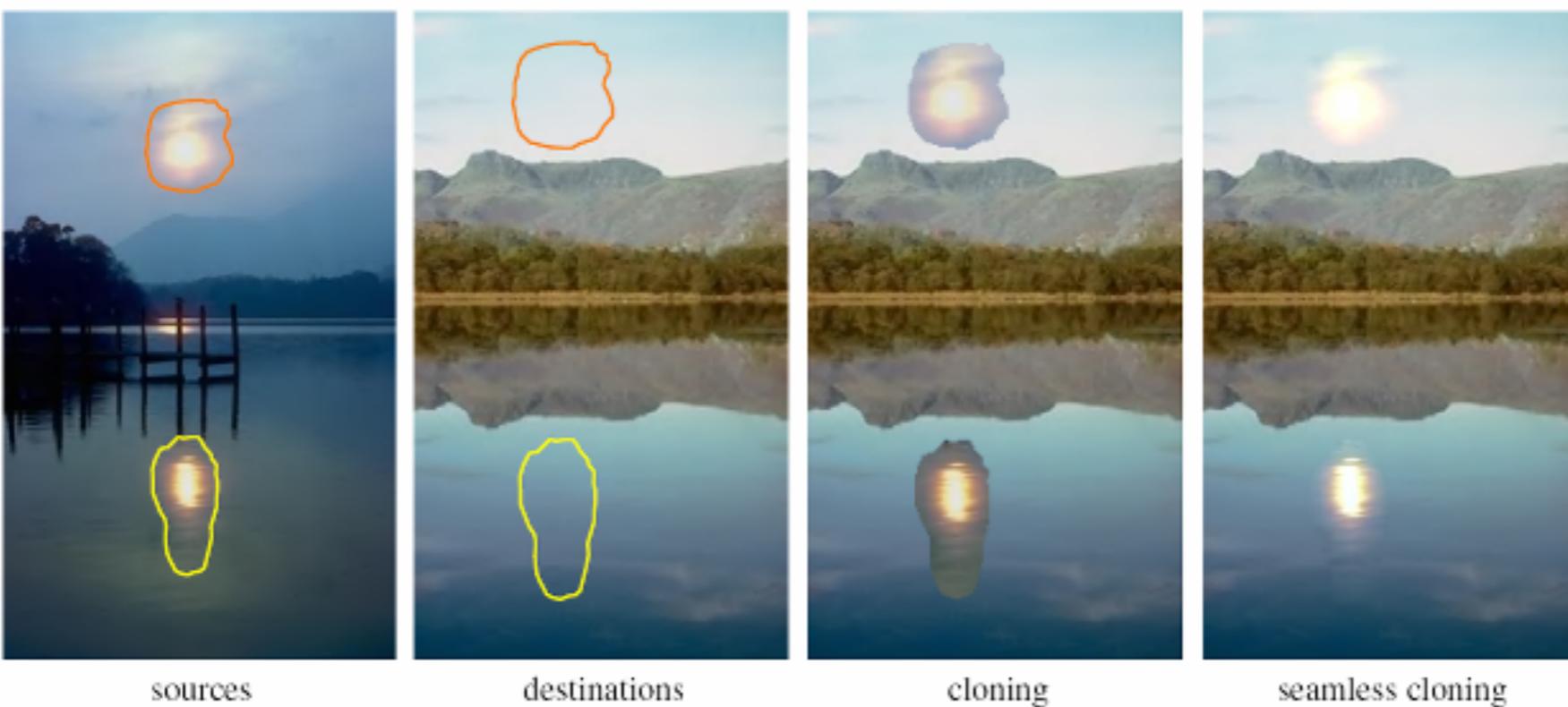


Pyramid blending on the gradients

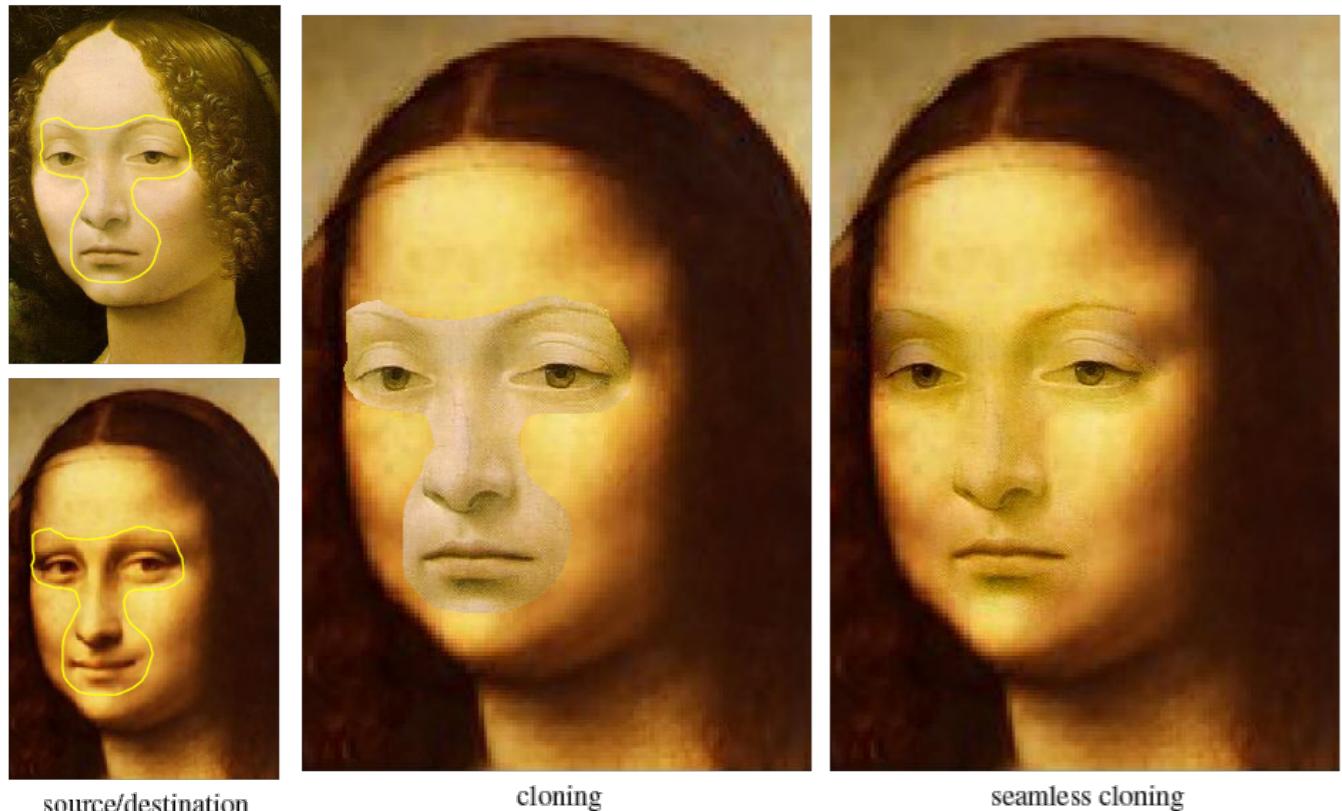


GIST1

Perez et al., 2003



Perez et al, 2003



source/destination

cloning

seamless cloning



editing

Limitations:

- Can't do contrast reversal (gray on black -> gray on white)
- Colored backgrounds “bleed through”
- Images need to be very well aligned

Don't blend, CUT!



Moving objects become ghosts

So far we only tried to blend between two images.
What about finding an optimal seam?

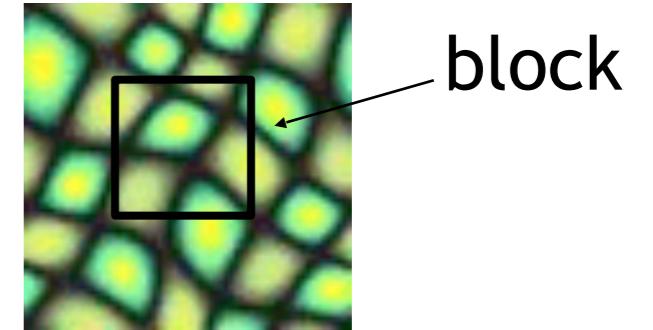
Davis, 1998

Segment the mosaic

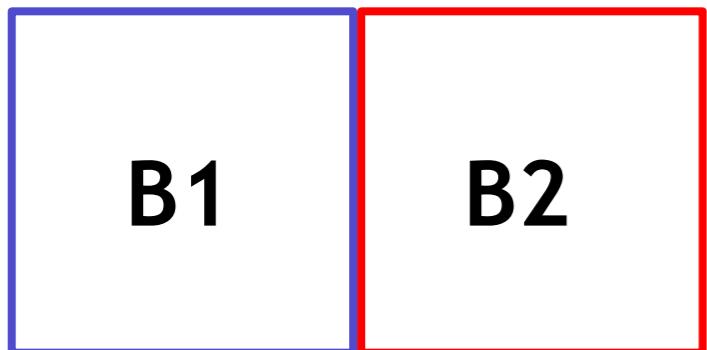
- Single source image per segment
- Avoid artifacts along boundaries
 - Dijkstra's algorithm



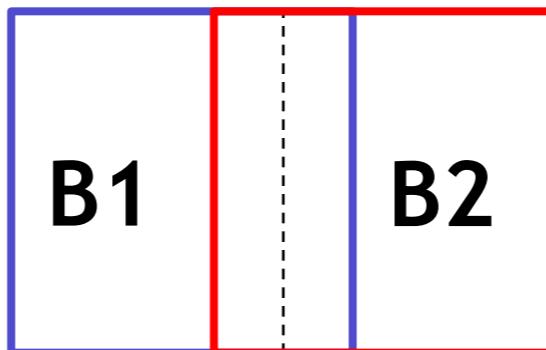
Efros & Freeman, 2001



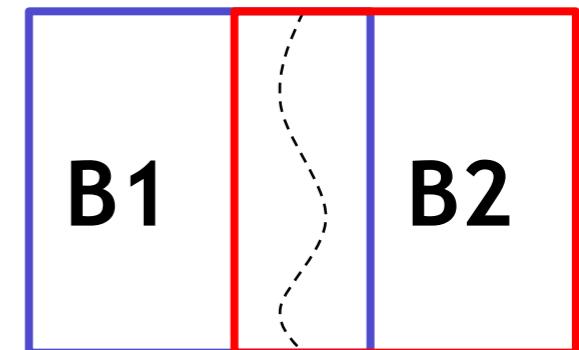
Input texture



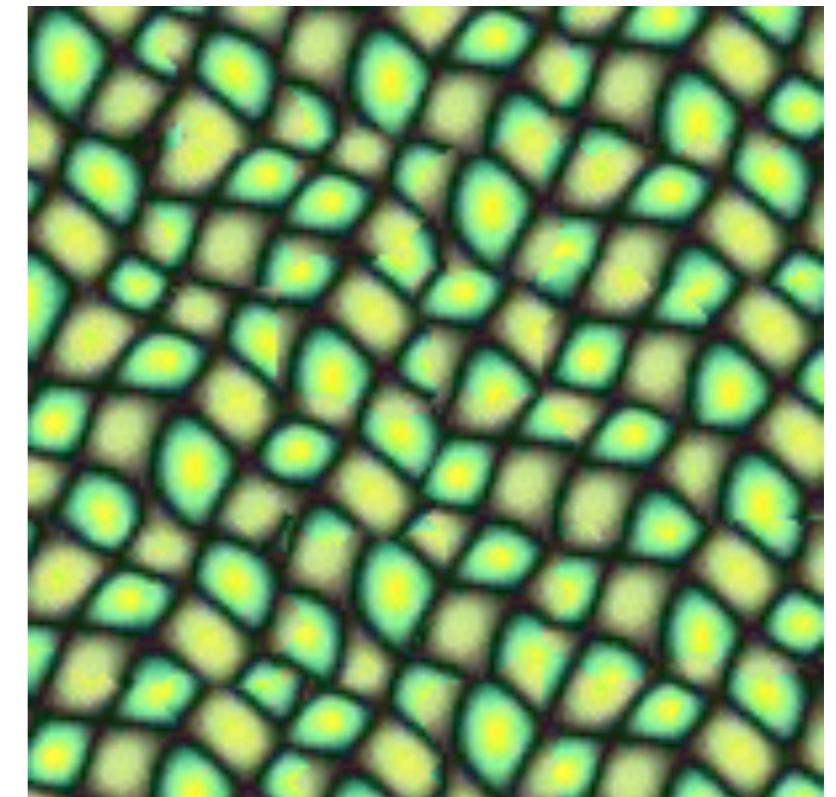
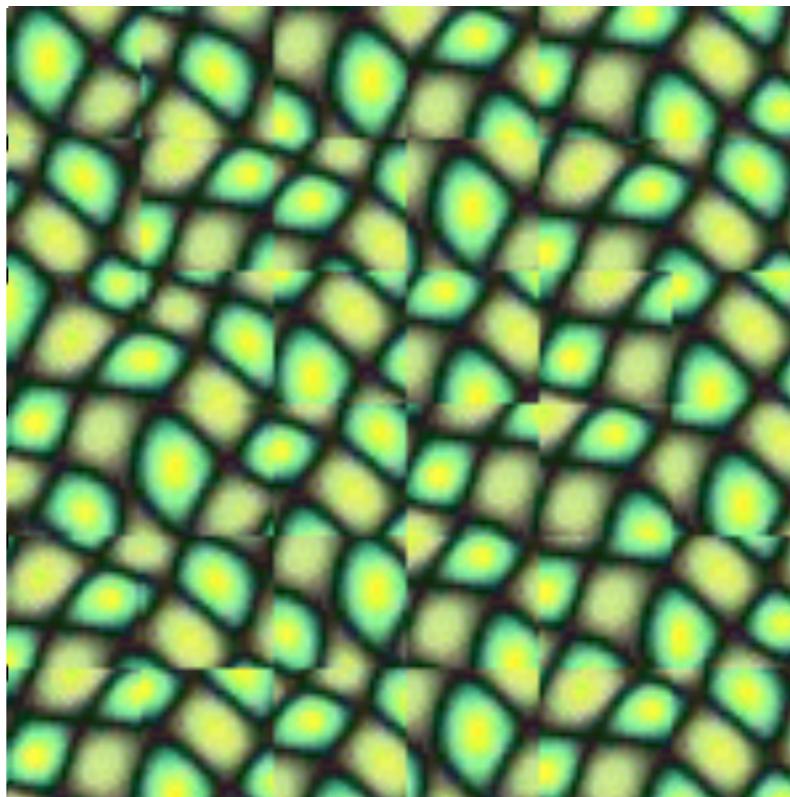
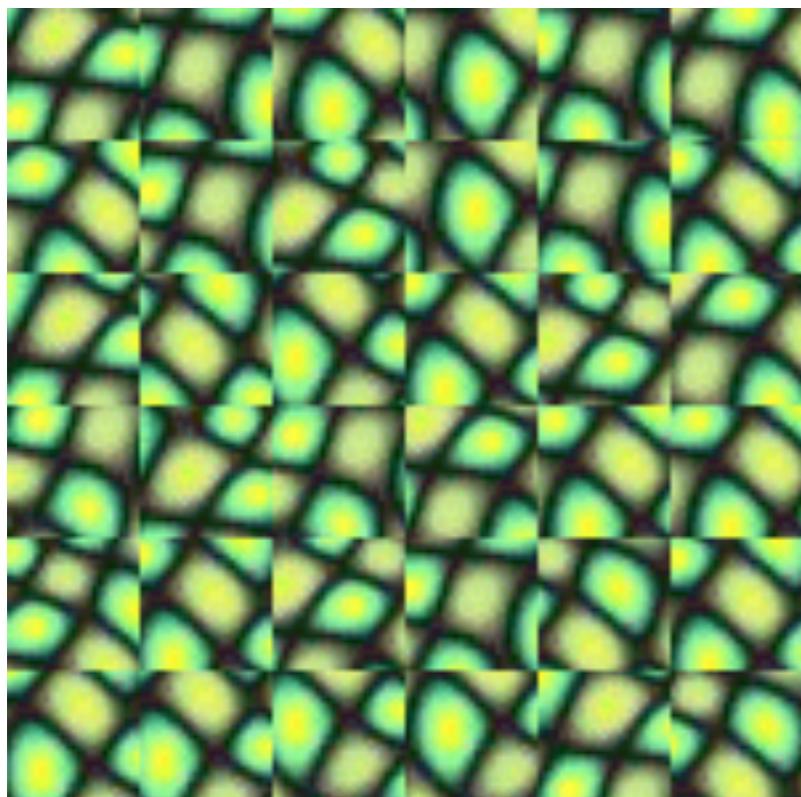
Random placement
of blocks



Neighboring blocks
constrained by overlap

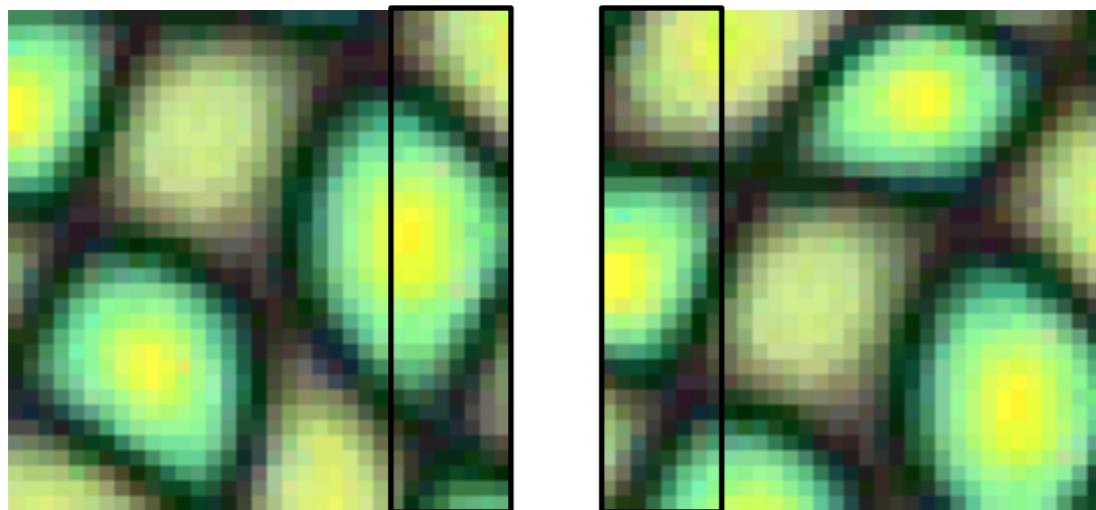


Minimal error
boundary cut

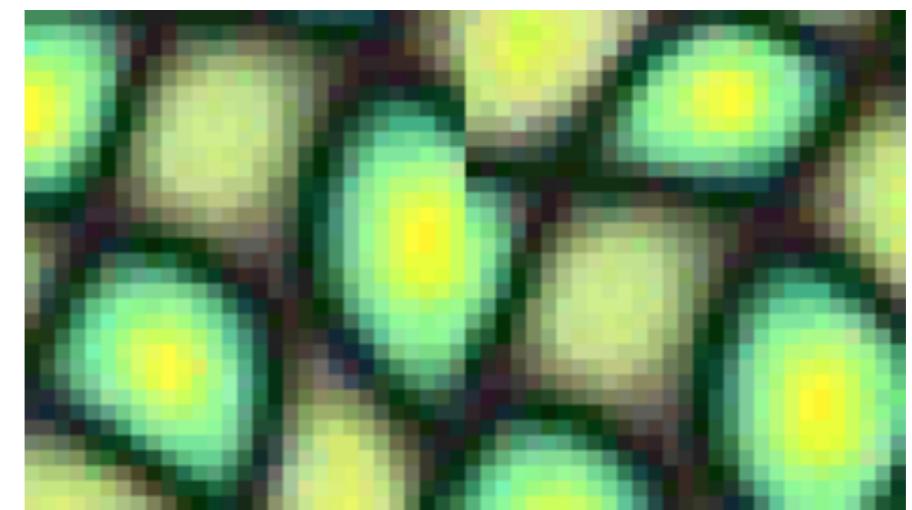


Minimal error boundary

overlapping blocks



vertical boundary



A diagram showing the calculation of overlap error. It consists of two small overlapping blocks from the top image, separated by a minus sign, enclosed in curly braces. To the right of the minus sign is an equals sign followed by a small image of a vertical boundary with red highlights at the edges. Arrows point from the two small blocks to the corresponding parts of the equation.

$$\left[\begin{array}{c} \text{block 1} \\ - \\ \text{block 2} \end{array} \right] = \text{boundary}$$

overlap error



min. error boundary

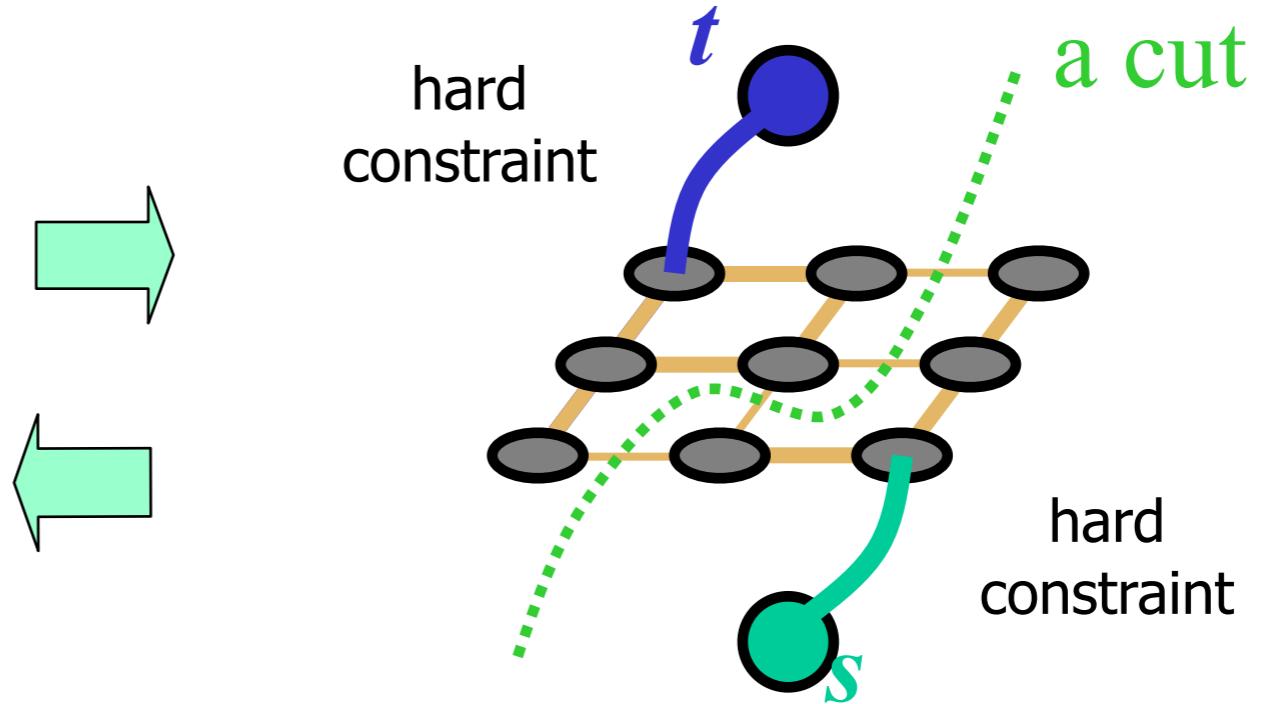
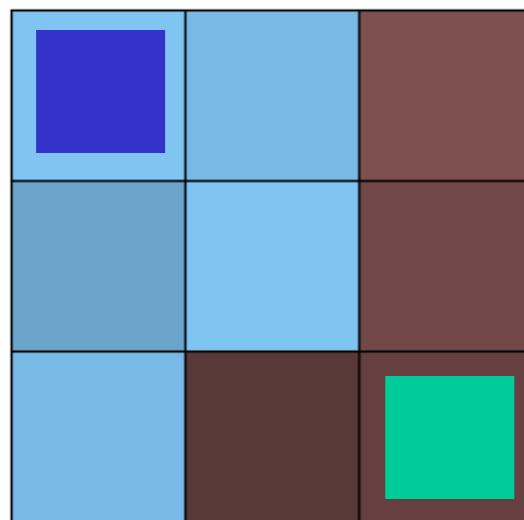
Graphcuts

What if we want similar “cut-where-things-agree” idea, but for closed regions?

- Dynamic programming can't handle loops

Graph cuts

(simple example à la Boykov&Jolly, ICCV'01)



Minimum cost cut can be computed in polynomial time
(max-flow/min-cut algorithms)

Kwatra et al, 2003



Actually, for this example, DP will work just as well...

Putting it all together

Compositing images

- Have a clever blending function
 - Feathering
 - blend different frequencies differently
 - Gradient based blending
- Choose the right pixels from each image
 - Dynamic programming – optimal seams
 - Graph-cuts

Worktime: Project 1

