



Blending and Compositing



Computational Photography
Derek Hoiem, University of Illinois



hybridImage.m

```
function [im12, im1_low, im2_high] = hybridImage(im1, im2, cutoff1, cutoff2)
% function [im12, im1_low, im2_high] = hybridImage(im1, im2, cutoff1, cutoff2)
% Inputs:
%   im1: the image to be low passed with sigma = image_size / cutoff1
%   im2: the image to be high-passed with sigma = image_size / cutoff2
% Outputs:
%   im12: the hybrid image
%   im1_low: the low-pass first image
%   im2_high: the high-pass second image

len = sqrt(size(im1, 1).^2+size(im2, 1).^2);
sigma1 = (len / cutoff1);
sigma2 = (len / cutoff2);
disp(['sigma_low = ' num2str(sigma1) '   sigma_high = ' num2str(sigma2)]);
filter_low = fspecial('gaussian', ceil(sigma1)*6+1, sigma1);
filter_high = fspecial('gaussian', ceil(sigma2)*6+1, sigma2);
im1_low = imfilter(im1, filter_low, 'same');
im2_high = im2 - imfilter(im2, filter_high, sigma2, 'same');
im12 = (im1_low + im2_high)/2; % or im12=(im1_low + im2_high)
im12 = min(max(im12, 0), 1);

figure(1), hold off, imshow(im1_low)
figure(2), hold off, imagesc(im2_high), axis image, colormap gray
figure(3), hold off, imagesc(im12); axis image, colormap gray
```

pyramids.m

```
function pyramids(im, N)

if size(im, 3)==3
    im = rgb2gray(im);
end
figure, clf
gim{1} = im;
for k = 1:N
    gim{k+1} = imfilter(gim{k}, fspecial('gaussian', 13, 2));
    lim{k} = gim{k}-gim{k+1};
    gim{k+1} = gim{k+1}(1:2:end, 1:2:end);
    subplot(2, 7, k), imagesc(lim{k}), colormap gray, axis off, axis image
    subplot(2, 7, k+7), imagesc(gim{k+1}), colormap gray, axis image, axis off
end
```

Project 1: issues

- Basic tips
 - Display/save Laplacian images using `mat2gray` or `imagesc` (`imshow` is only for normal images)
 - Downsampling: use $\sigma \approx 2$ for factor of 2

Color shift

Result from Sihao Chen

More red (increase positive a channel values)



Original Image

L*a*b* space



More Red



Less Yellow

Less yellow (decrease positive b channel values)

Project 1 Results

Incomplete list of great project pages

<http://xiaoyan5.web.engr.illinois.edu/cs445/proj1/> (Xiaoyan Wang: most likes)

<http://tgurtler.web.engr.illinois.edu/cs445/proj1/>

<http://schen149.web.engr.illinois.edu/cs445/proj1/>

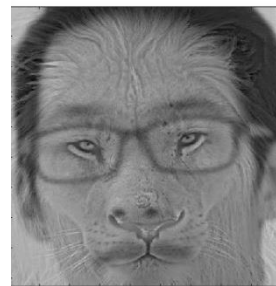
<http://kurtovc2.web.engr.illinois.edu/cs445/proj1/>

<http://jdrynld2.web.engr.illinois.edu/cs445/proj1/>

<http://mscraft2.web.engr.illinois.edu/cs445/proj1/>

<http://trrscl2.web.engr.illinois.edu/cs445/proj1/>

Hybrid results





Last class: finding boundaries

- Intelligent scissors
 - Good boundary has a low-cost path from seed to cursor
 - Low cost = edge, high gradient, right orientation
- GrabCut
 - Good region is similar to foreground color model and dissimilar from background color
 - Good boundaries have a high gradient
 - Optimize over both

Take-home questions

1. What would be the result in “Intelligent Scissors” if all of the edge costs were set to 1?
2. How could you change boundary costs for graph cuts to work better for objects with many thin parts?

Last Class: cutting out objects



This Class

How do I put an object from one image into another?

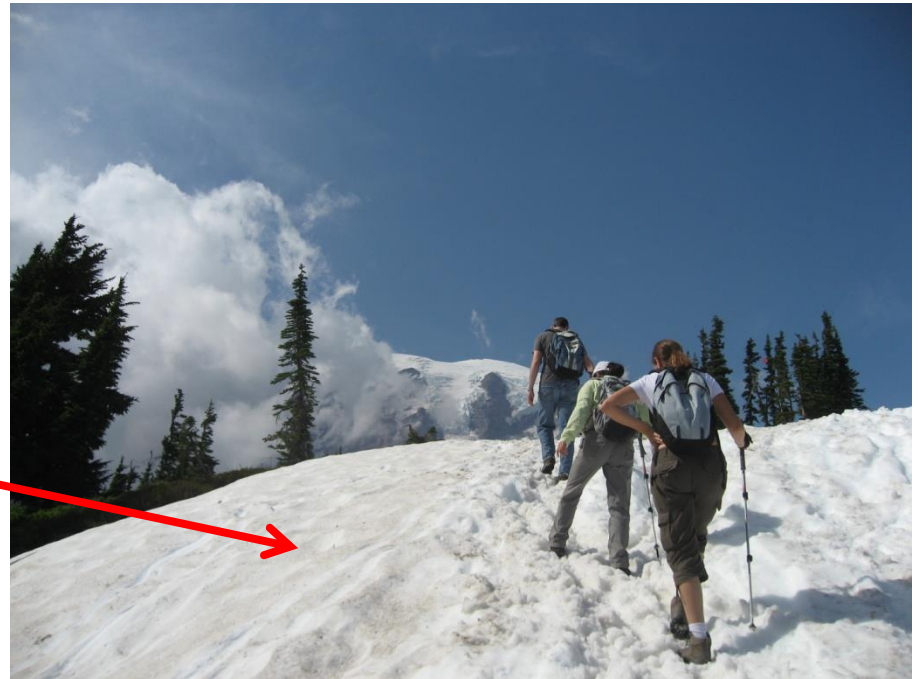
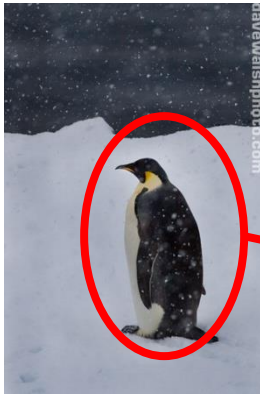


Image Compositing



News Composites

Original



“Enhanced”
Version



News Composites

Original



“Enhanced”
Version



Three methods

1. Cut and paste
2. Laplacian pyramid blending
3. Poisson blending

Method 1: Cut and Paste



Method 1: Cut and Paste

Method:

- Segment using intelligent scissors
- Paste foreground pixels onto target region



Method 1: Cut and Paste

Problems:

- Small segmentation errors noticeable
- Pixels are too blocky
- Won't work for semi-transparent materials



Feathering

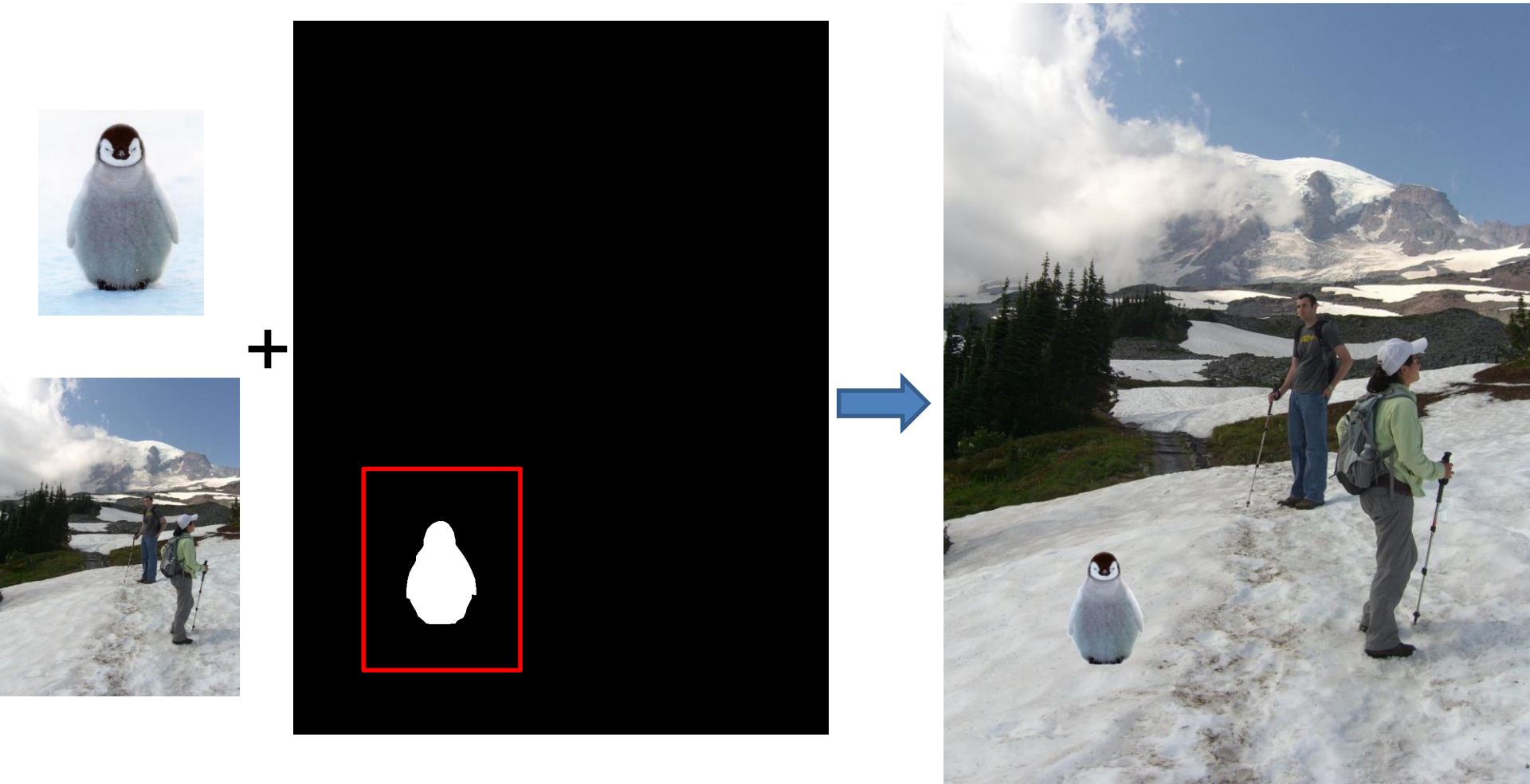
Near object boundary pixel values come partly from foreground and partly from background



Method 1: Cut and Paste (with feathering)

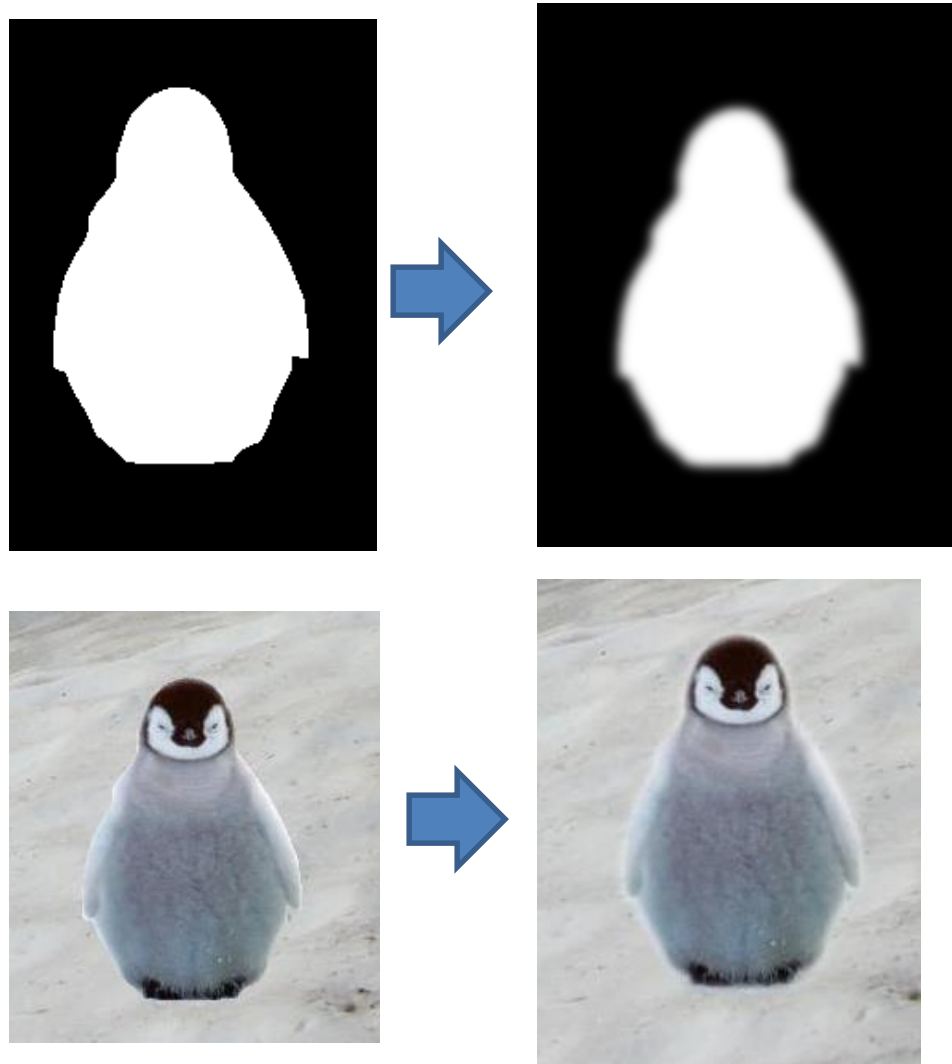


Alpha compositing



$$\text{Output} = \text{foreground} * \text{mask} + \text{background} * (1 - \text{mask})$$

Alpha compositing with feathering



Output = foreground*mask + background*(1-mask)

Another example (without feathering)

Mattes

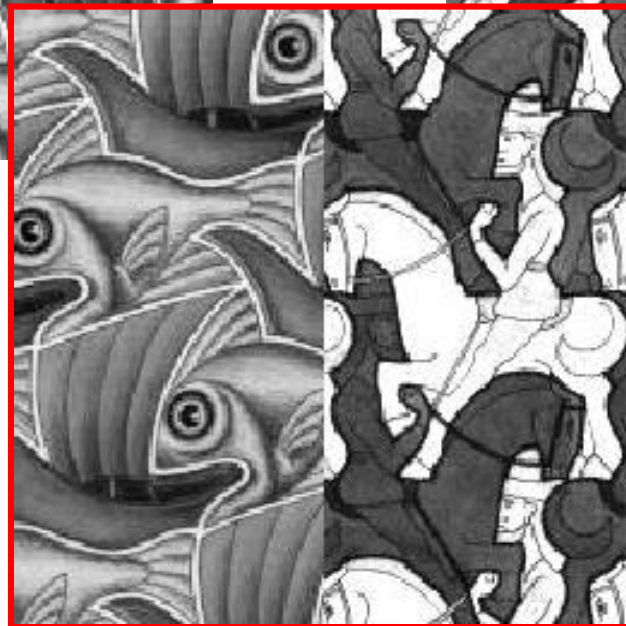
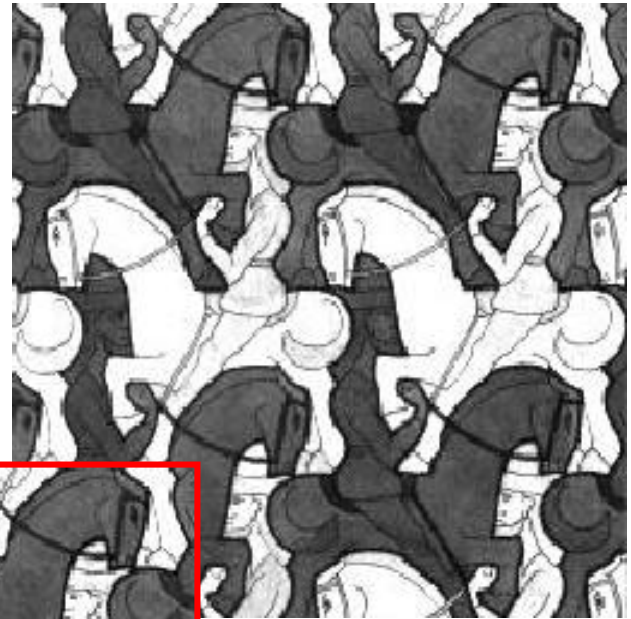
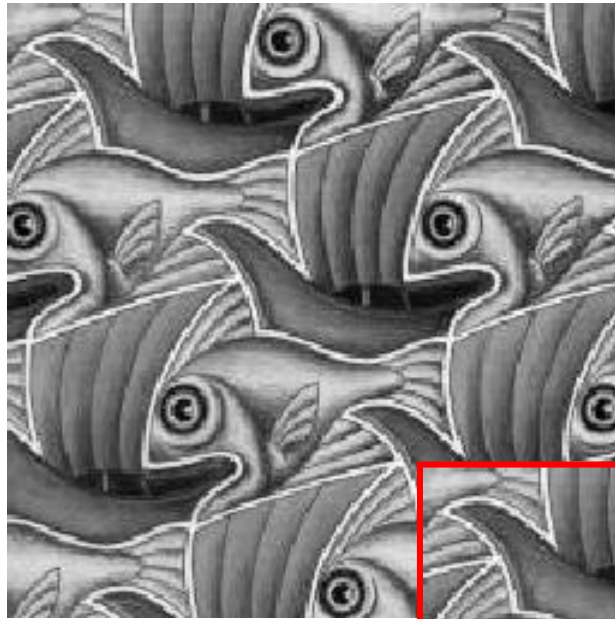


Composite

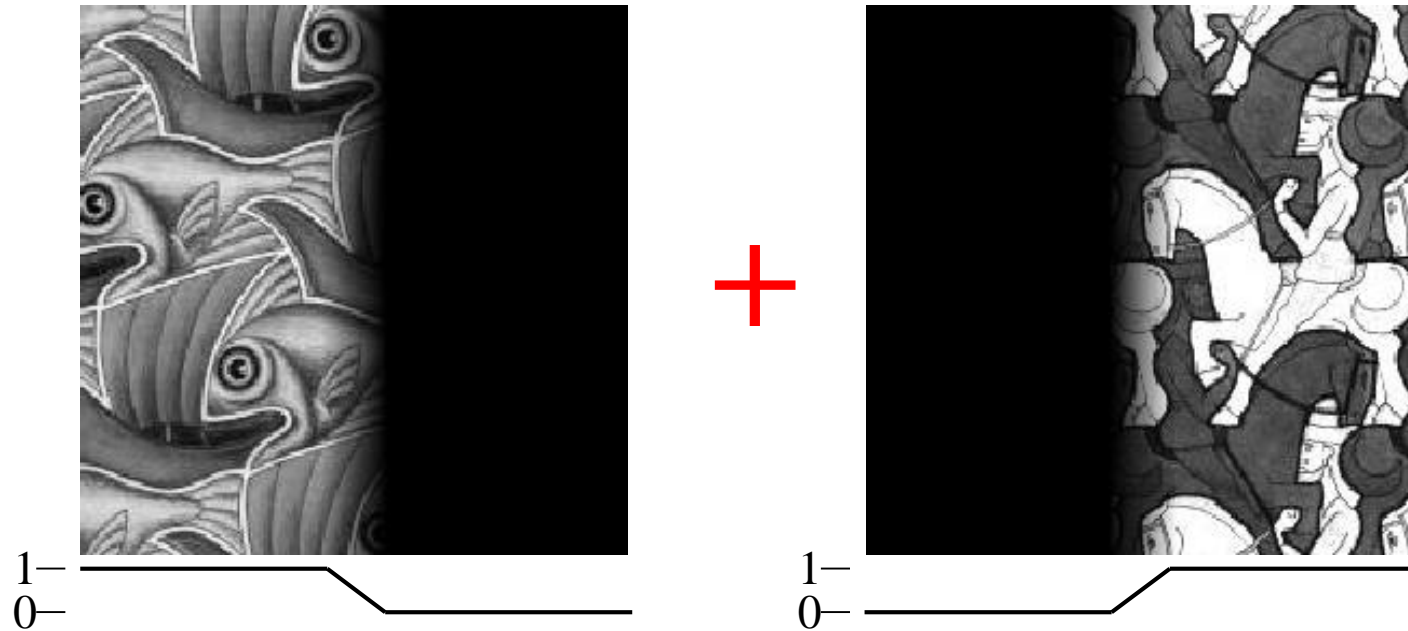


Composite by
David Dewey

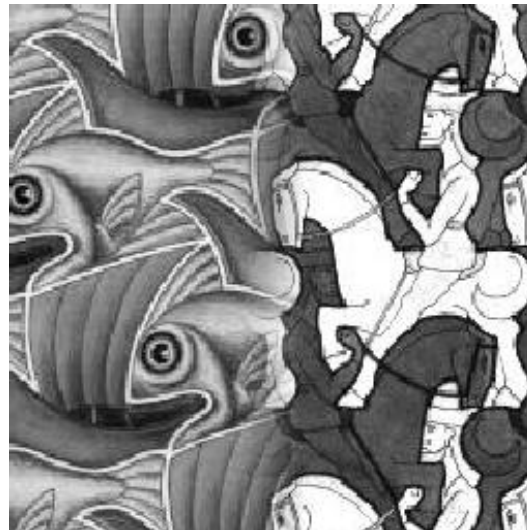
Proper blending is key



Alpha Blending / Feathering

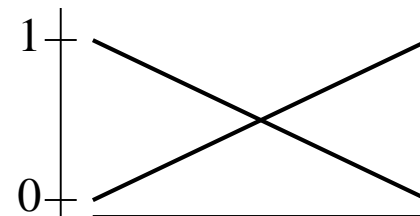
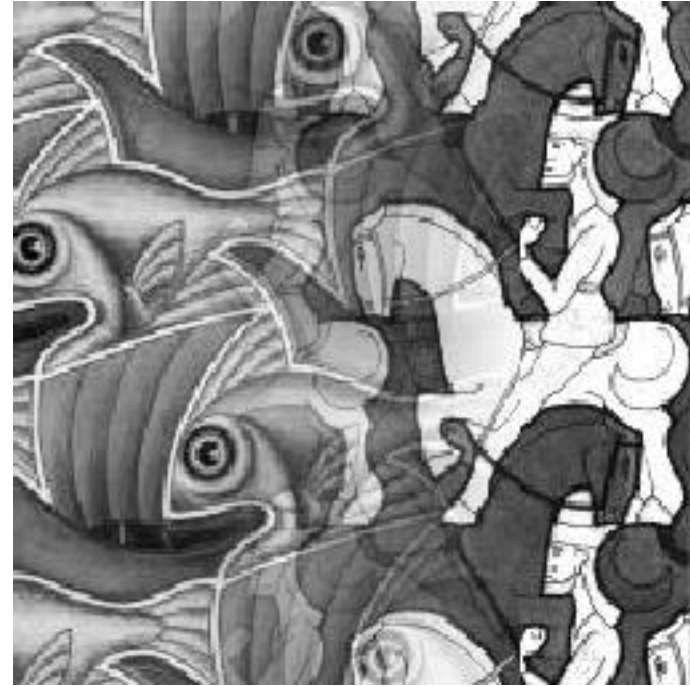
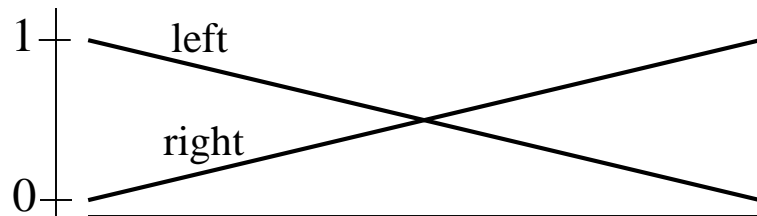


=

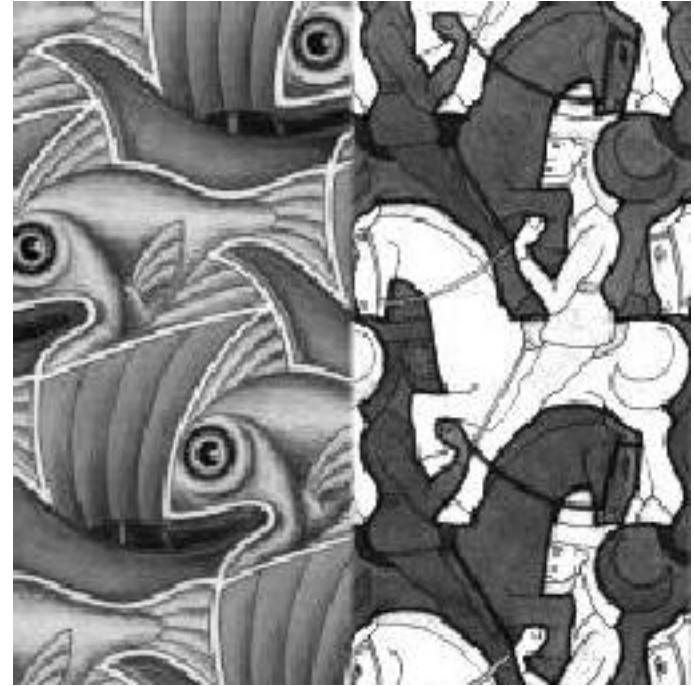
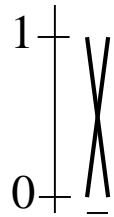


$$I_{\text{blend}} = \alpha I_{\text{left}} + (1-\alpha) I_{\text{right}}$$

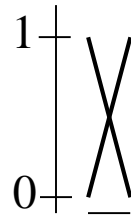
Effect of Window Size



Effect of Window Size

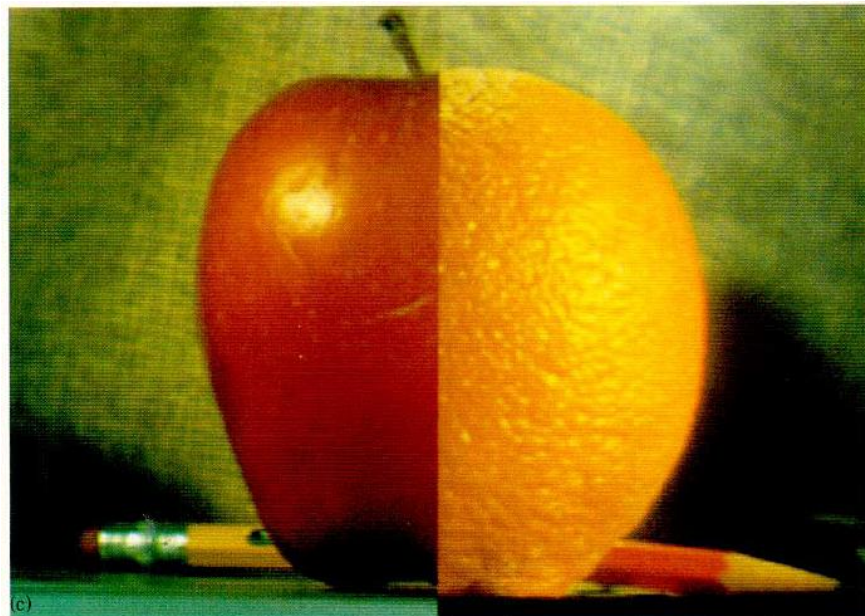
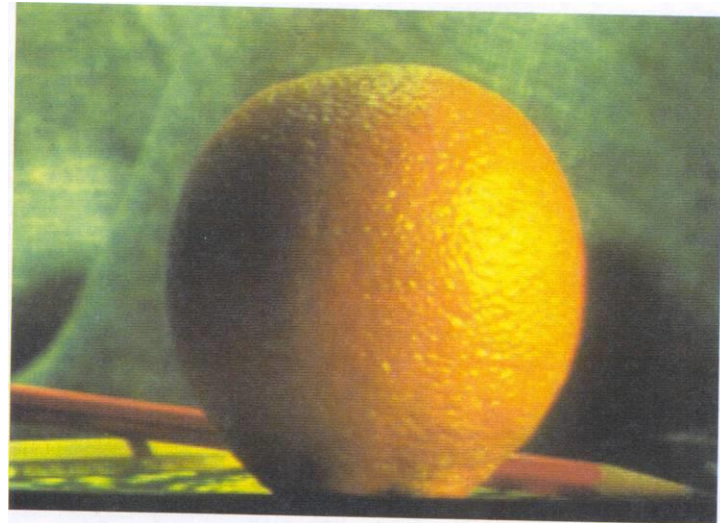
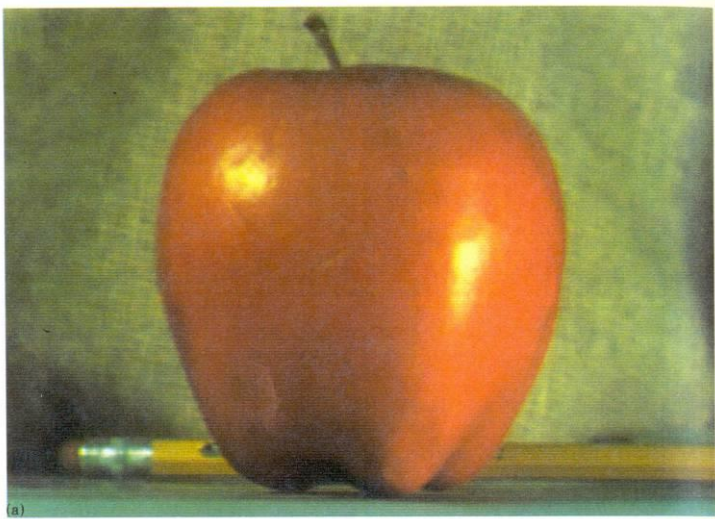


Good Window Size



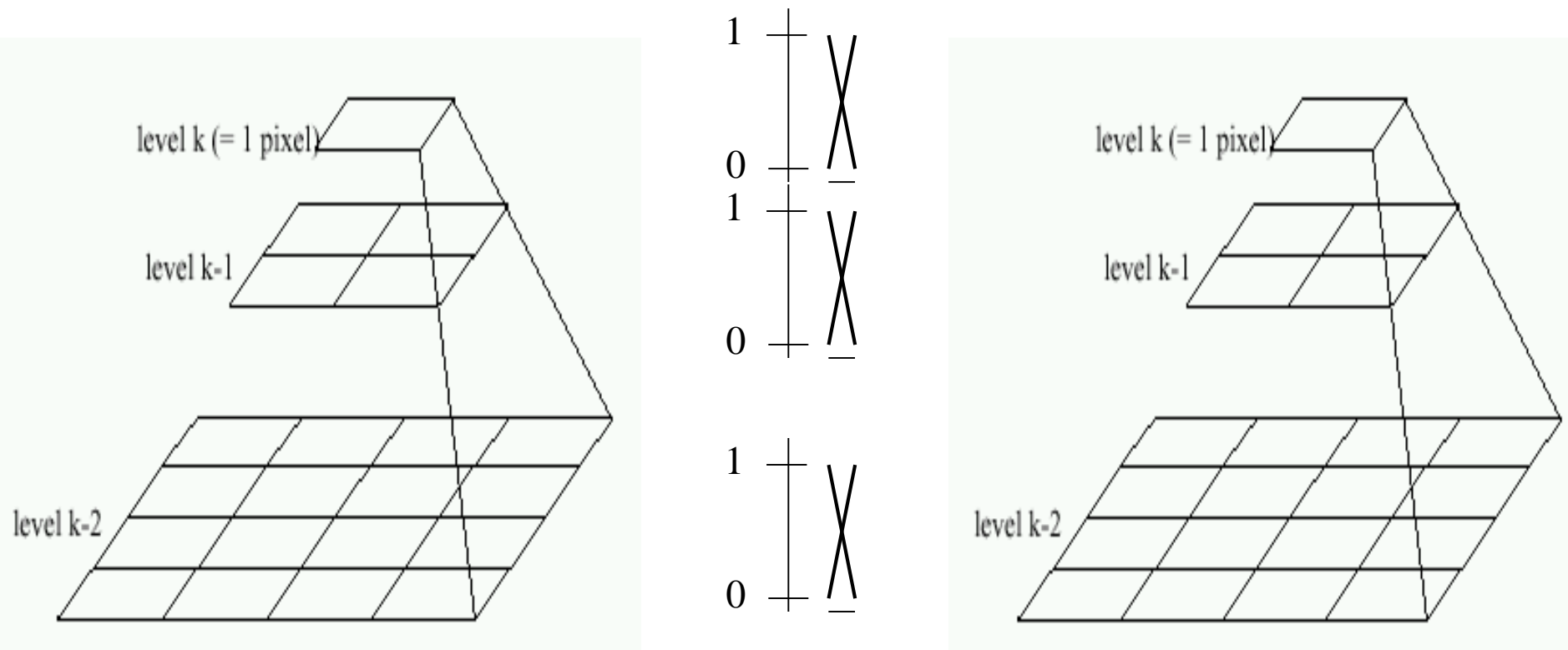
“Optimal” Window: smooth but not ghosted

How much should we blend?



Method 2: Pyramid Blending

- At low frequencies, blend slowly
- At high frequencies, blend quickly

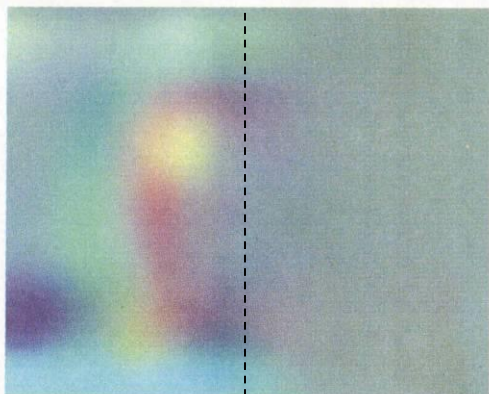


Left pyramid

blend

Right pyramid

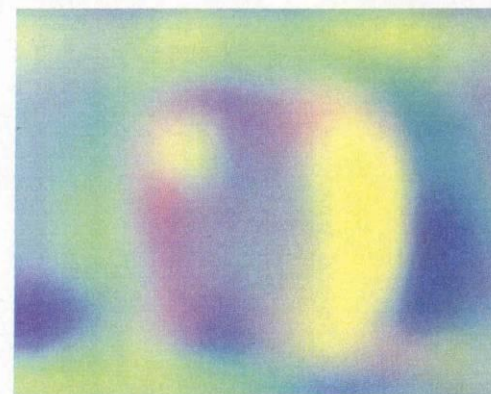
laplacian
level
4



(c)

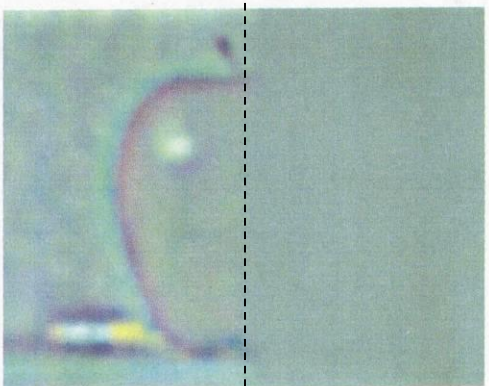


(g)

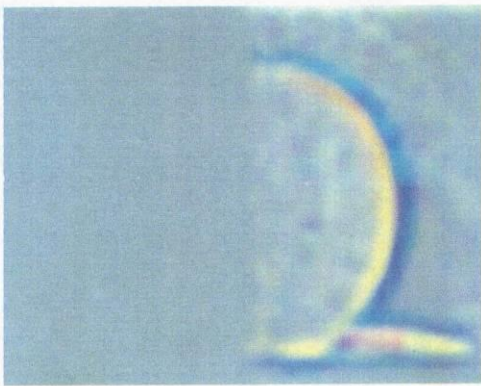


(k)

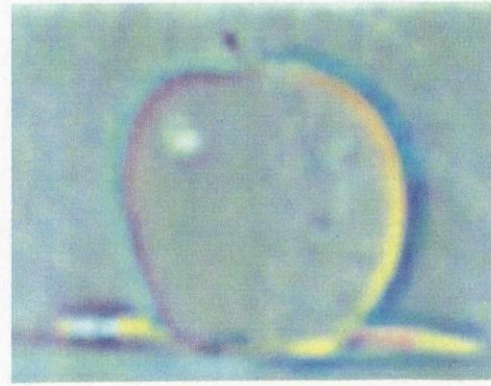
laplacian
level
2



(b)

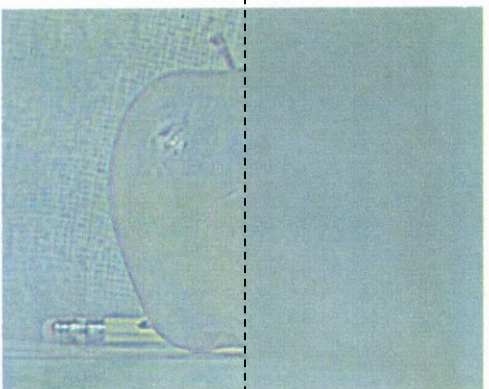


(f)

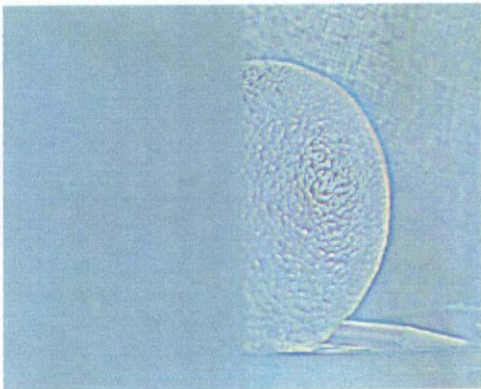


(j)

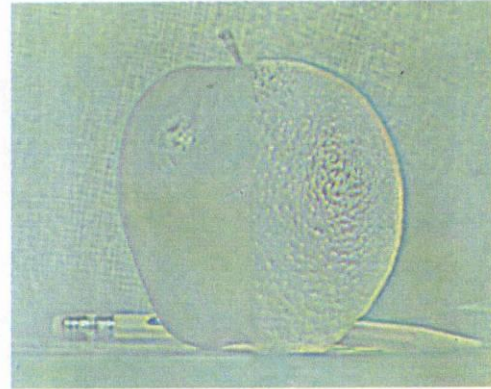
laplacian
level
0



(a)



(e)



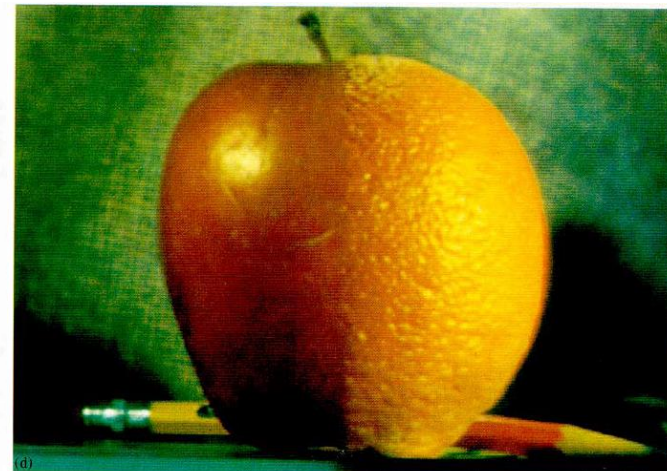
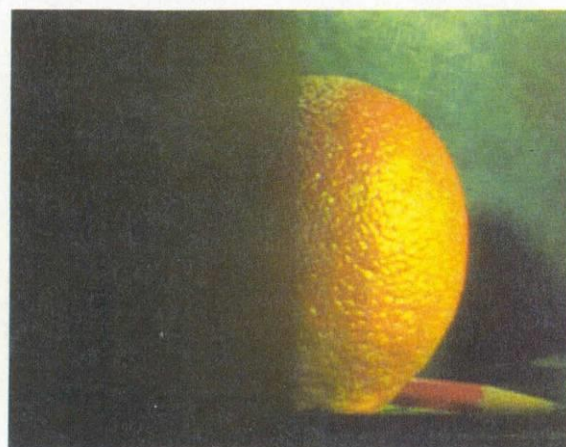
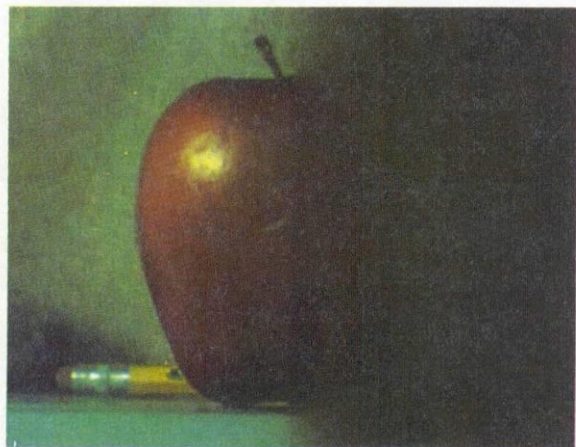
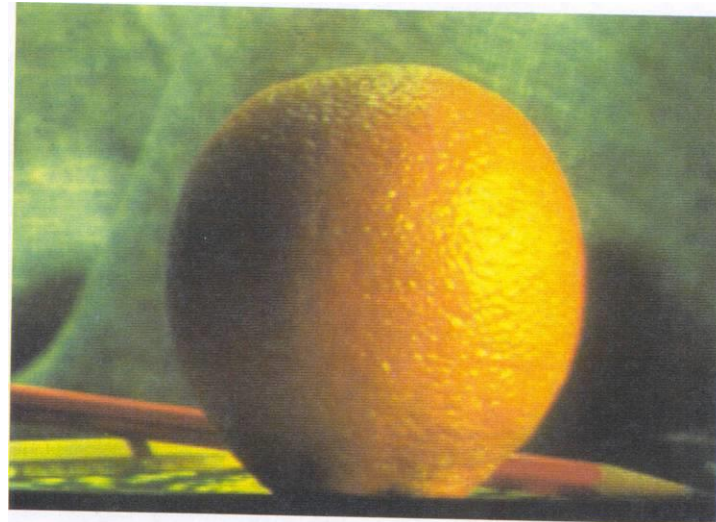
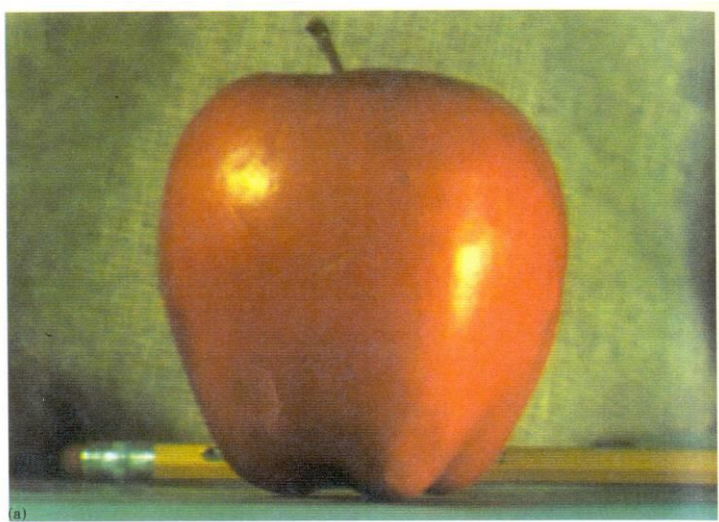
(i)

left pyramid

right pyramid

blended pyramid

Method 2: Pyramid Blending



Laplacian Pyramid Blending

Implementation:

1. Build Laplacian pyramids for each image
2. Build a Gaussian pyramid of region mask
3. Blend each level of pyramid using region mask from the same level

$$L_{12}^i = L_1^i \cdot R^i + L_2^i \cdot (1 - R^i)$$

Image 1 at level i of
Laplacian pyramid

Pointwise multiply

Region mask at level i
of Gaussian pyramid

4. Collapse the pyramid to get the final blended image

Simplification: Two-band Blending

- Brown & Lowe, 2003
 - Only use two bands: high freq. and low freq.
 - Blends low freq. smoothly



2-band Blending



Low frequency



High frequency

Linear Blending



2-band Blending



Blending Regions





© Chris Cameron

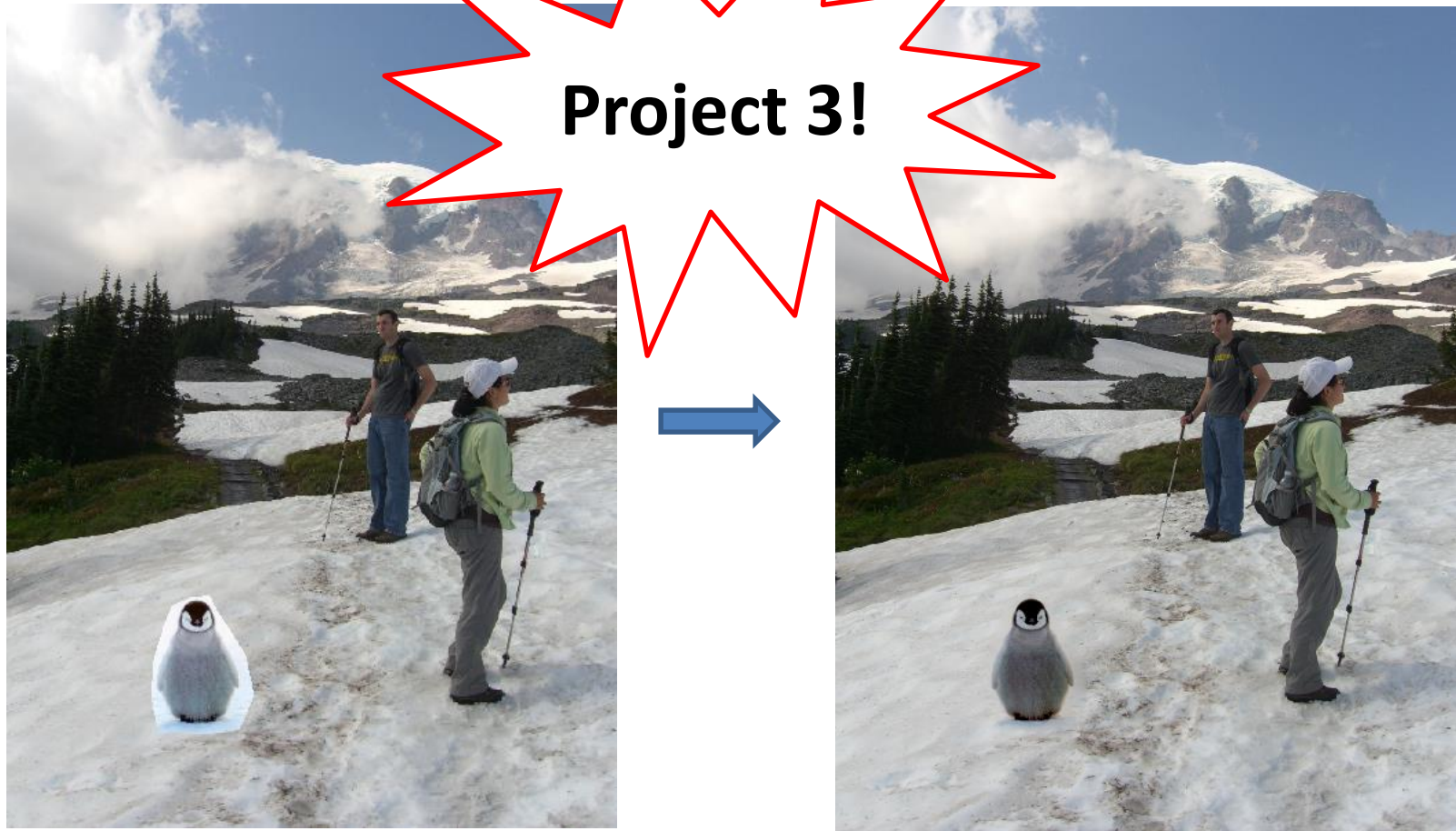
Related idea: Poisson Blending

A good blend should preserve gradients of source region without changing the background



Related idea: Poisson Blending

A good blend should preserve gradients of source region without changing the background



Method 3: Poisson Blending

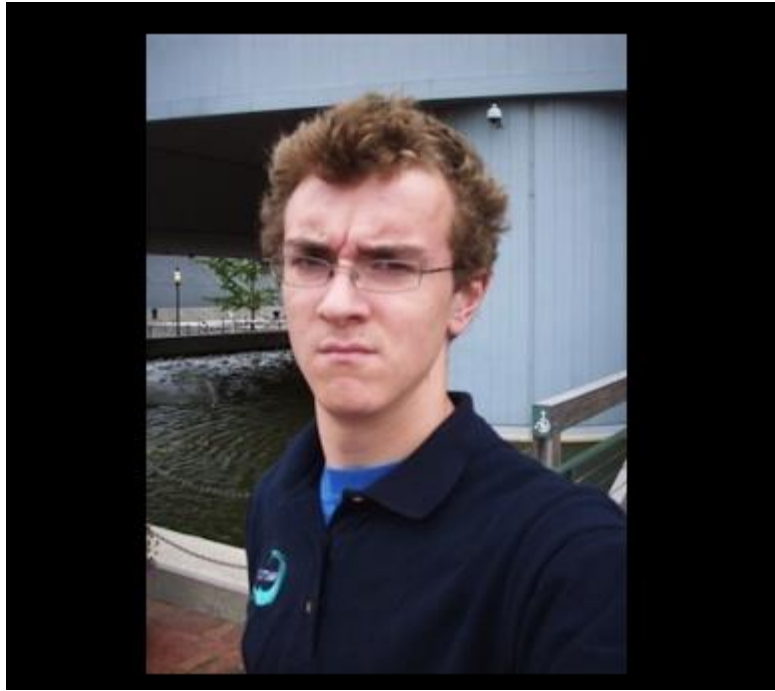
A good blend should preserve gradients of source region without changing the background

Treat pixels as variables to be solved

- Minimize squared difference between gradients of foreground region and gradients of target region
- Keep background pixels constant

$$\mathbf{v} = \underset{\mathbf{v}}{\operatorname{argmin}} \sum_{i \in S, j \in N_i \cap S} ((v_i - v_j) - (s_i - s_j))^2 + \sum_{i \in S, j \in N_i \cap \neg S} ((v_i - t_j) - (s_i - s_j))^2$$

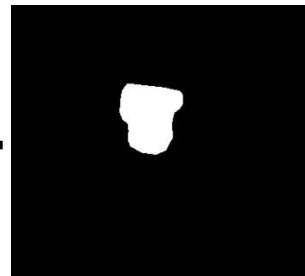
Example



Gradient Visualization



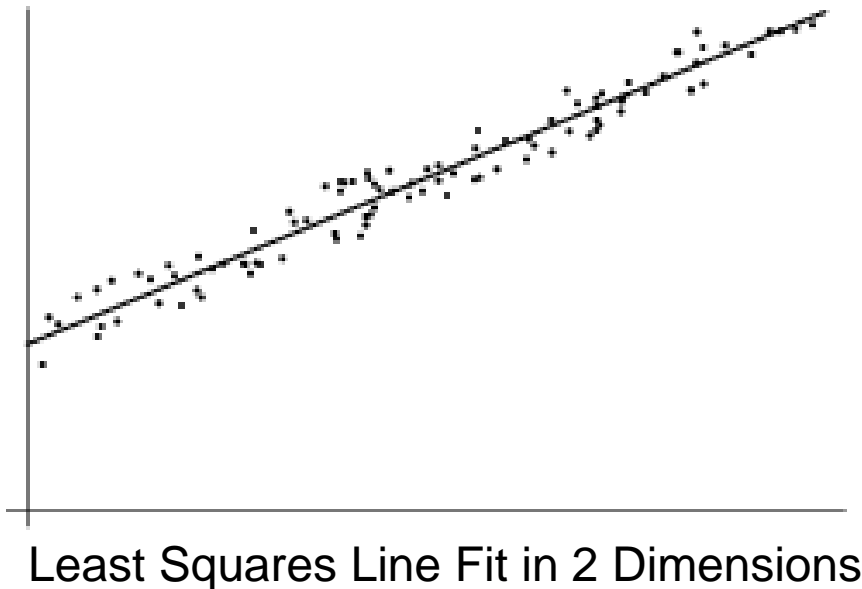
+



Source: Evan Wallace

Gradient-domain editing

Creation of image = least squares problem in terms of: 1) pixel intensities; 2) differences of pixel intensities



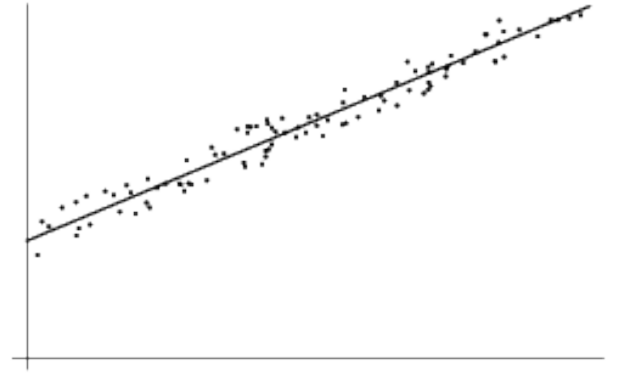
$$\hat{\mathbf{v}} = \arg \min_{\mathbf{v}} \sum_i \left(\mathbf{a}_i^T \mathbf{v} - b_i \right)^2$$

$$\hat{\mathbf{v}} = \arg \min_{\mathbf{v}} \left(\mathbf{A} \mathbf{v} - \mathbf{b} \right)^2$$

Use Matlab least-squares solvers for numerically stable solution with sparse \mathbf{A}

Examples

1. Line-fitting: $y=mx+b$



Examples

2. Gradient domain processing

$$\mathbf{v} = \operatorname{argmin}_{\mathbf{v}} \sum_{i \in S, j \in N_i \cap S} ((v_i - v_j) - (s_i - s_j))^2 + \sum_{i \in S, j \in N_i \cap \bar{S}} ((v_i - t_j) - (s_i - s_j))^2$$

source image

1	20	5	20	9	20	13	20
2	20	6	80	10	20	14	20
3	20	7	20	11	80	15	20
4	20	8	20	12	20	16	20

background image

1	10	5	10	9	10	13	10
2	10	6	10	10	10	14	10
3	10	7	10	11	10	15	10
4	10	8	10	12	10	16	10

target image

1	10	5	10	9	10	13	10
2	10	6	\mathbf{v}_1	10	\mathbf{v}_3	14	10
3	10	7	\mathbf{v}_2	11	\mathbf{v}_4	15	10
4	10	8	10	12	10	16	10

Other results

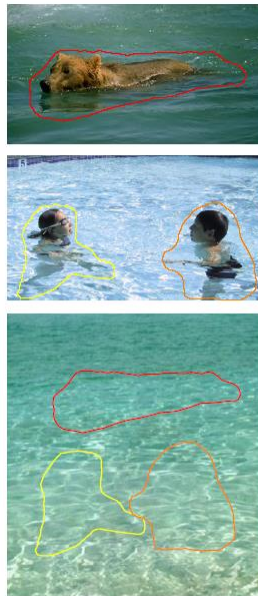


sources

destinations

cloning

seamless cloning



sources/destinations



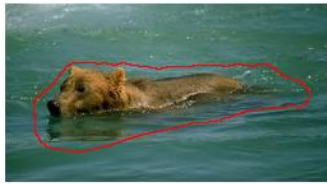
cloning



seamless cloning

What do we lose?

- Foreground color changes
- Background pixels in target region are replaced



sources/destinations



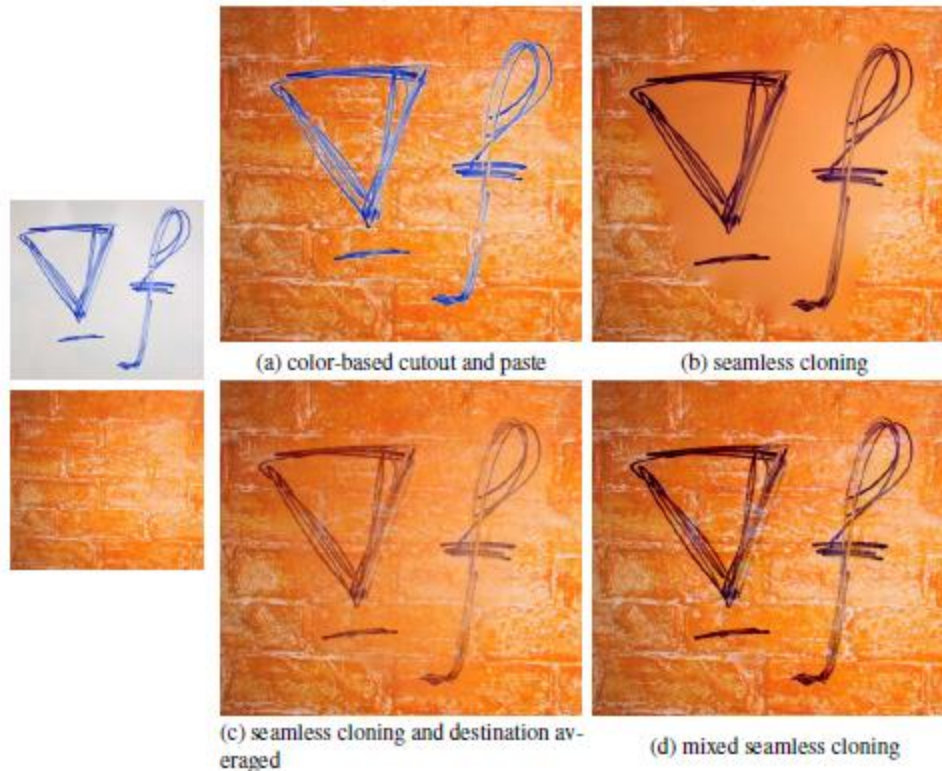
cloning



seamless cloning

Blending with Mixed Gradients

- Use foreground or background gradient with larger magnitude as the guiding gradient



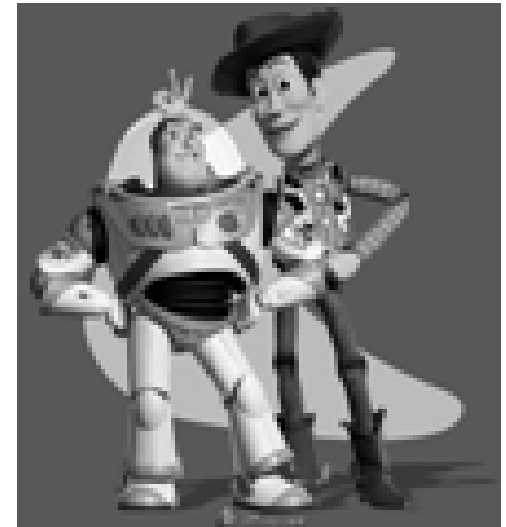
Project 3: Gradient Domain Editing

General concept: Solve for pixels of new image that satisfy constraints on the gradient and the intensity

- Constraints can be from one image (for filtering) or more (for blending)

Project 3: Reconstruction from Gradients

1. Preserve x-y gradients
2. Preserve intensity of one pixel



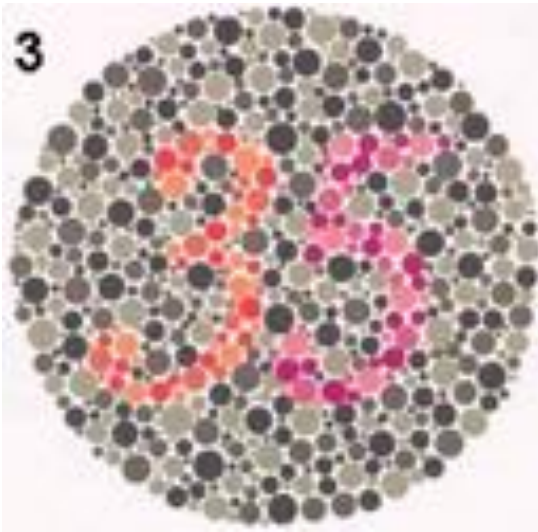
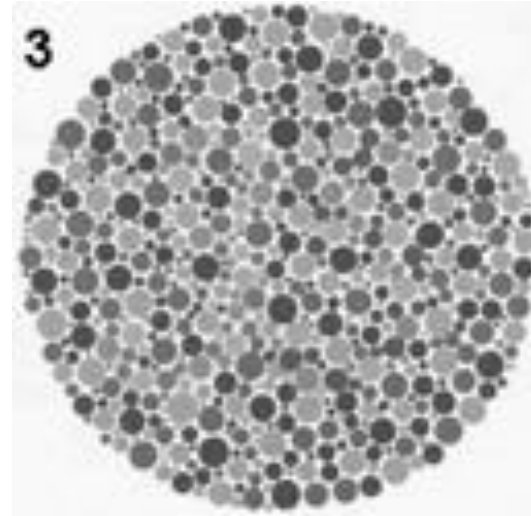
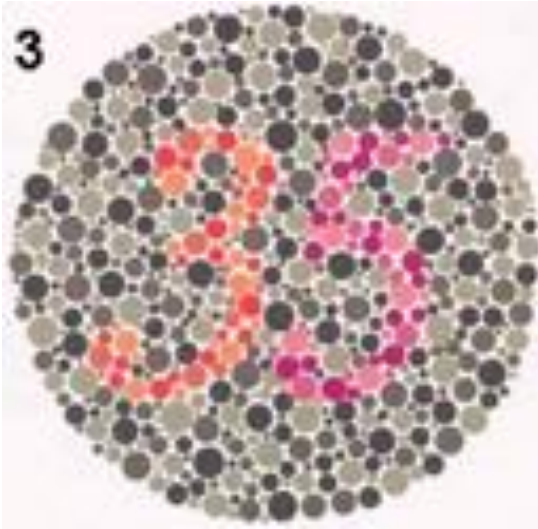
Source pixels: s

Variable pixels: v

1. minimize $(v(x+1,y)-v(x,y) - (s(x+1,y)-s(x,y)))^2$
2. minimize $(v(x,y+1)-v(x,y) - (s(x,y+1)-s(x,y)))^2$
3. minimize $(v(1,1)-s(1,1))^2$

Project 3 (extra): Color2Gray

rgb2gray



Gradient-domain
editing



?

Project 3 (extra): NPR

- Preserve gradients on edges
 - e.g., get canny edges with `edge(im, 'canny')`
- Reduce gradients not on edges
- Preserve original intensity



Colorization using optimization

- Solve for uv channels such that similar intensities have similar colors
- Minimize squared color difference, weighted by intensity similarity

$$J(U) = \sum_{\mathbf{r}} \left(U(\mathbf{r}) - \sum_{\mathbf{s} \in \mathcal{N}(\mathbf{r})} w_{\mathbf{r}\mathbf{s}} U(\mathbf{s}) \right)^2$$

- Solve with sparse linear system of equations



Things to remember

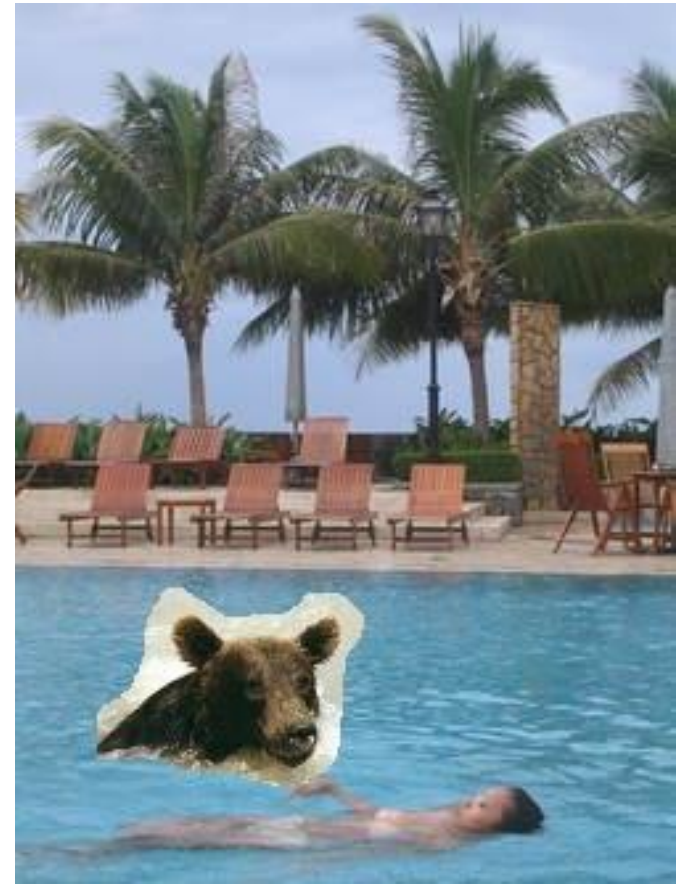
- Three ways to blend/composite
 1. Alpha compositing
 - Need nice cut (intelligent scissors)
 - Should **feather**
 2. Laplacian pyramid blending
 - **Smooth blending at low frequencies, sharp at high frequencies**
 - Usually used for stitching
 3. Gradient domain editing
 - Also called **Poisson Editing**
 - Explicit control over what to preserve
 - Changes foreground color (for better or worse)
 - Applicable for many things besides blending

Take-home questions

1) I am trying to blend this bear into this pool.

What problems will I have if I use:

- a) Alpha compositing with feathering
- b) Laplacian pyramid blending
- c) Poisson editing?



Take-home questions

- 2) How would you make a sharpening filter using gradient domain processing? What are the constraints on the gradients and the intensities?

Next class

- Image warping: affine, projective, rotation, etc.