

# Recitation 7

Blending and Stitching  
Hough Transform

# Agenda

## 1. Blending and Stitching

- a. Seam Location – Naïve
- b. Blend the Transition - Feathering (Alpha Blending) and Pyramid Blending
- c. Optimal Seam –
  - I. Dynamic Programming
  - II. Graph Cut

## 2. Hough Transform

# Blending and Stitching

Hough Transform



# Why Mosaic ?

Are you getting the whole picture?

- Compact Camera FOV =  $50 \times 35^\circ$



# Why Mosaic ?

Are you getting the whole picture?

- Compact Camera FOV =  $50 \times 35^\circ$
- Human FOV =  $200 \times 135^\circ$





# Why Mosaic ?

Are you getting the whole picture?

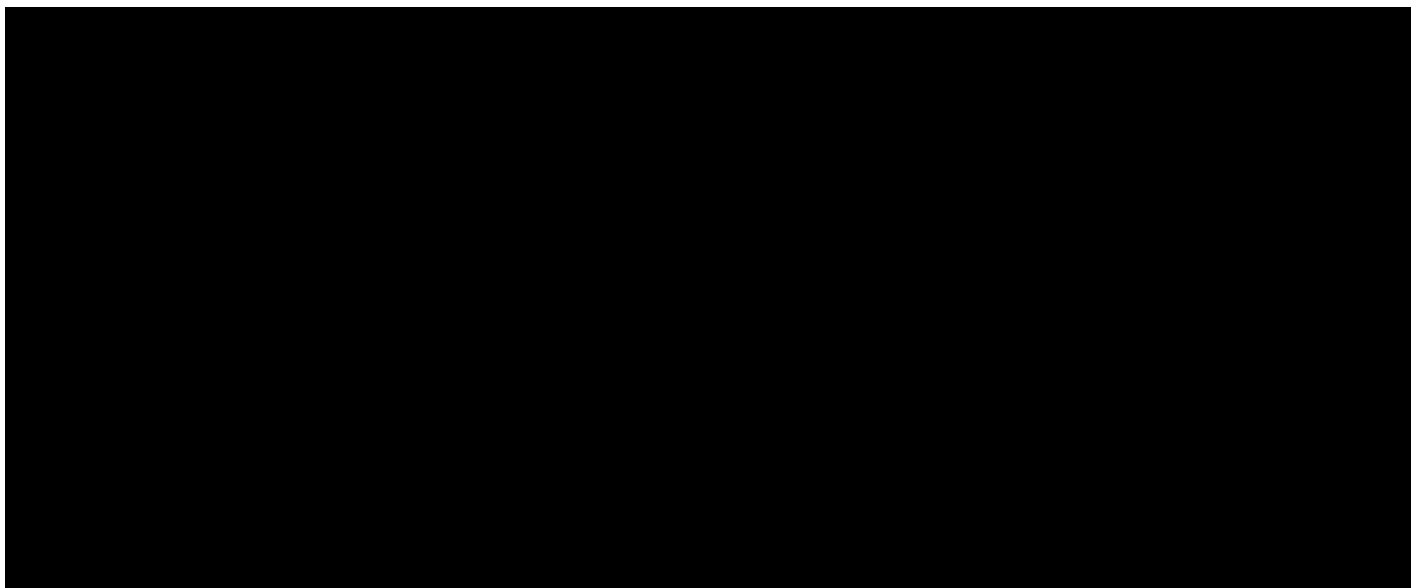
- Compact Camera FOV =  $50 \times 35^\circ$
- Human FOV =  $200 \times 135^\circ$
- Panoramic Mosaic =  $360 \times 180^\circ$



# Stages in Building Panoramas

- ✓ 1. Find alignment between overlapping images
  - Choose motion transformation between images
  - (translation, translation + rotation, affine, homography)
- ✓ 2. Choose a compositing surface for warping
3. Warp
4. Seamlessly blend images









# Problems



# Stitching the Images Together

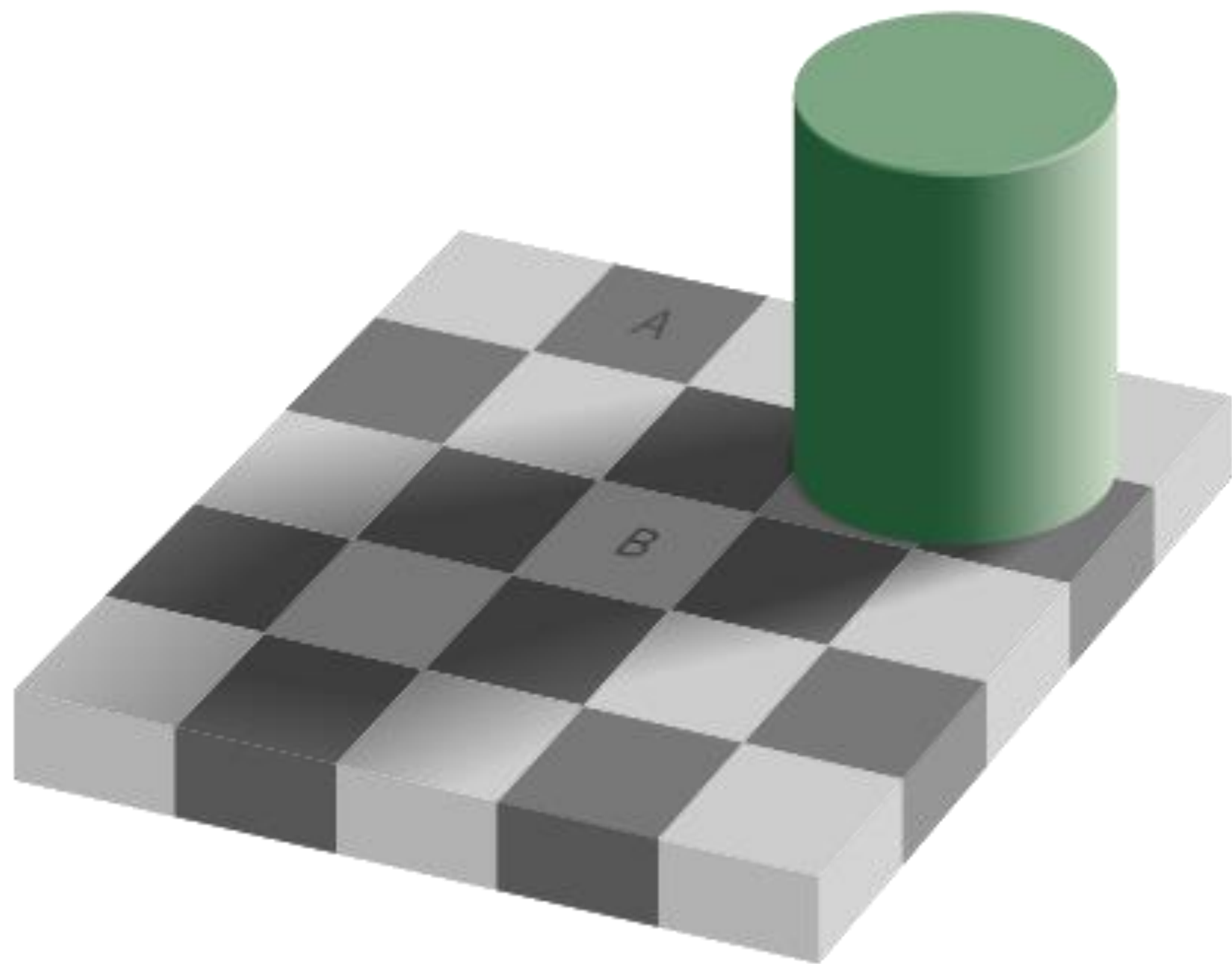
## Why is this a challenge?

- Exposure differences
- Vignetting
- Blurring (due to miss-registration)
- Ghosting (due to moving objects)



## Goal: invisible seams between images

- Minimal amount of seam artifacts: avoid the creation of edges that did not appear in the original images





A

B

# Approaches

Assuming that the images have already been aligned

- Simple (naïve) seam location

- Stitching

- Seam location + smoothing

Blend the transition between images

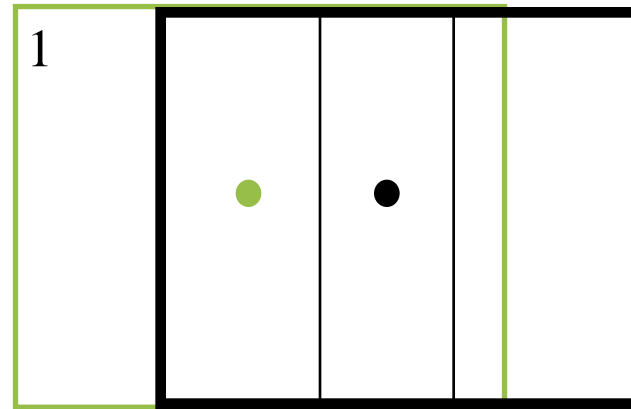
- Feathering (Alpha blending); Pyramid blending;

- Search for optimal seam

- Dynamic programming; Graph cuts;

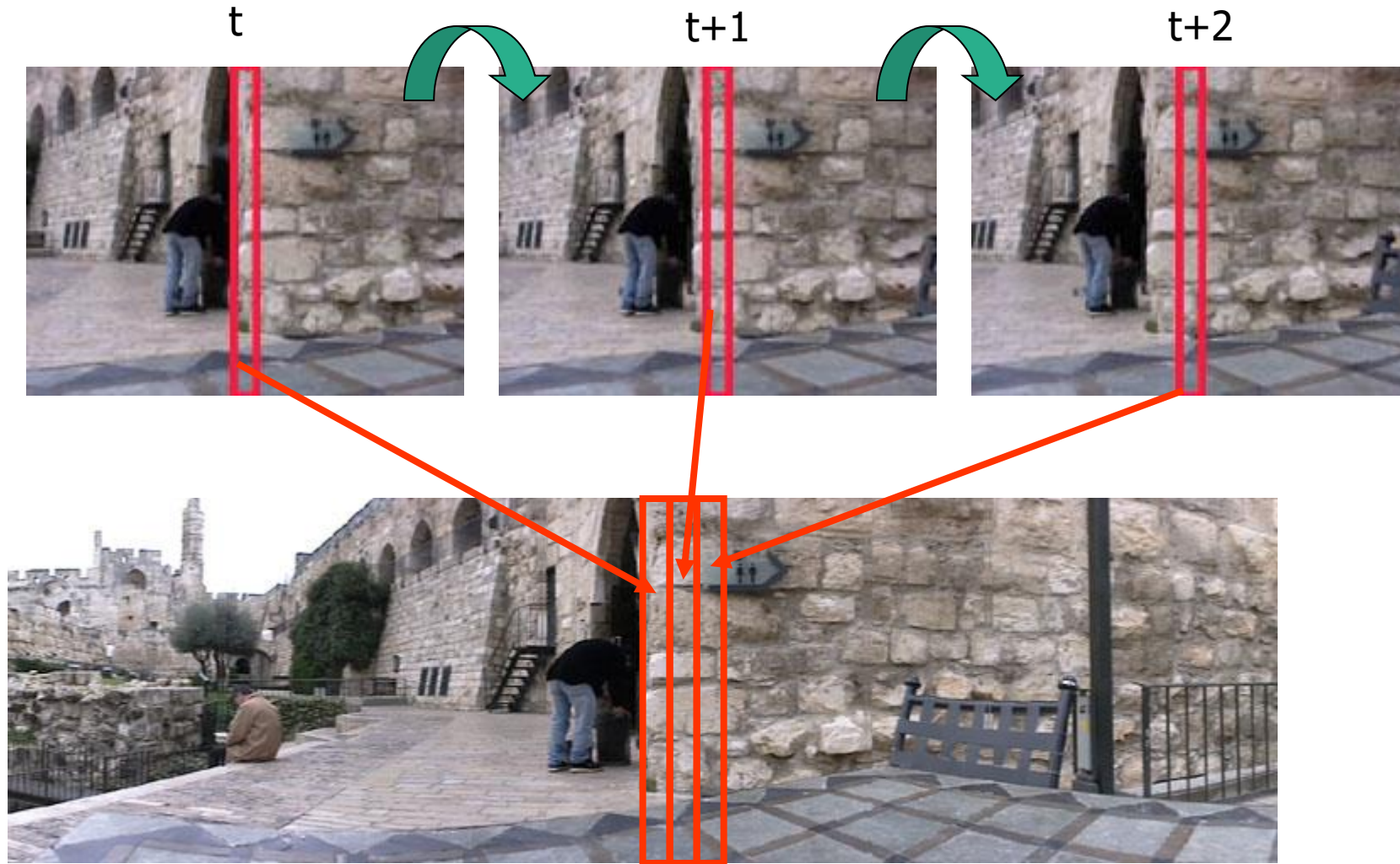
# Simple Seam Location

Cut & Paste using Center Strips

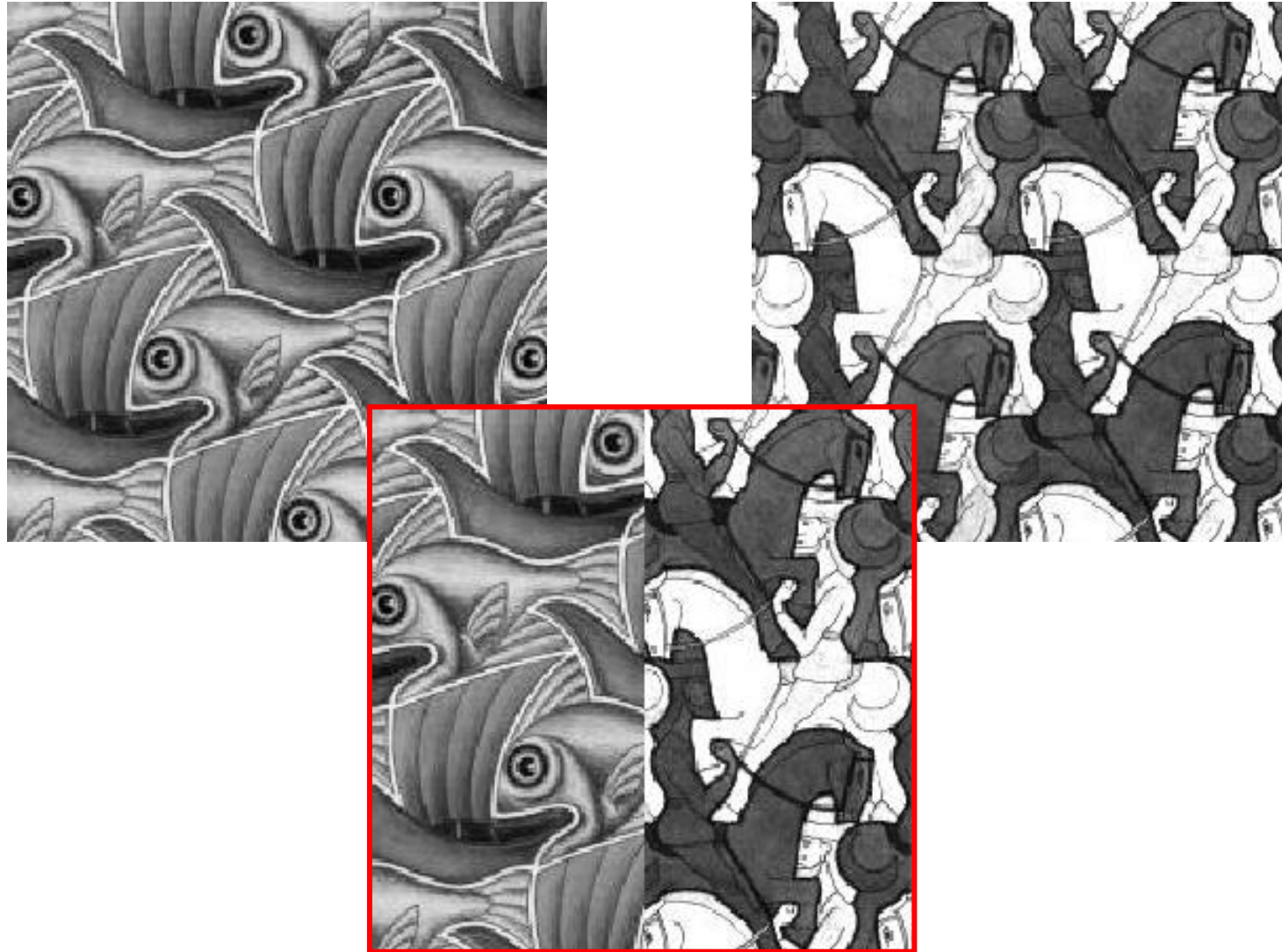


Strip taken from “2”

# Simple Seam Location



# Image Blending





# Approaches

Assuming that the images have already been aligned

- Simple (naïve) seam location

- Stitching

- Seam location + smoothing

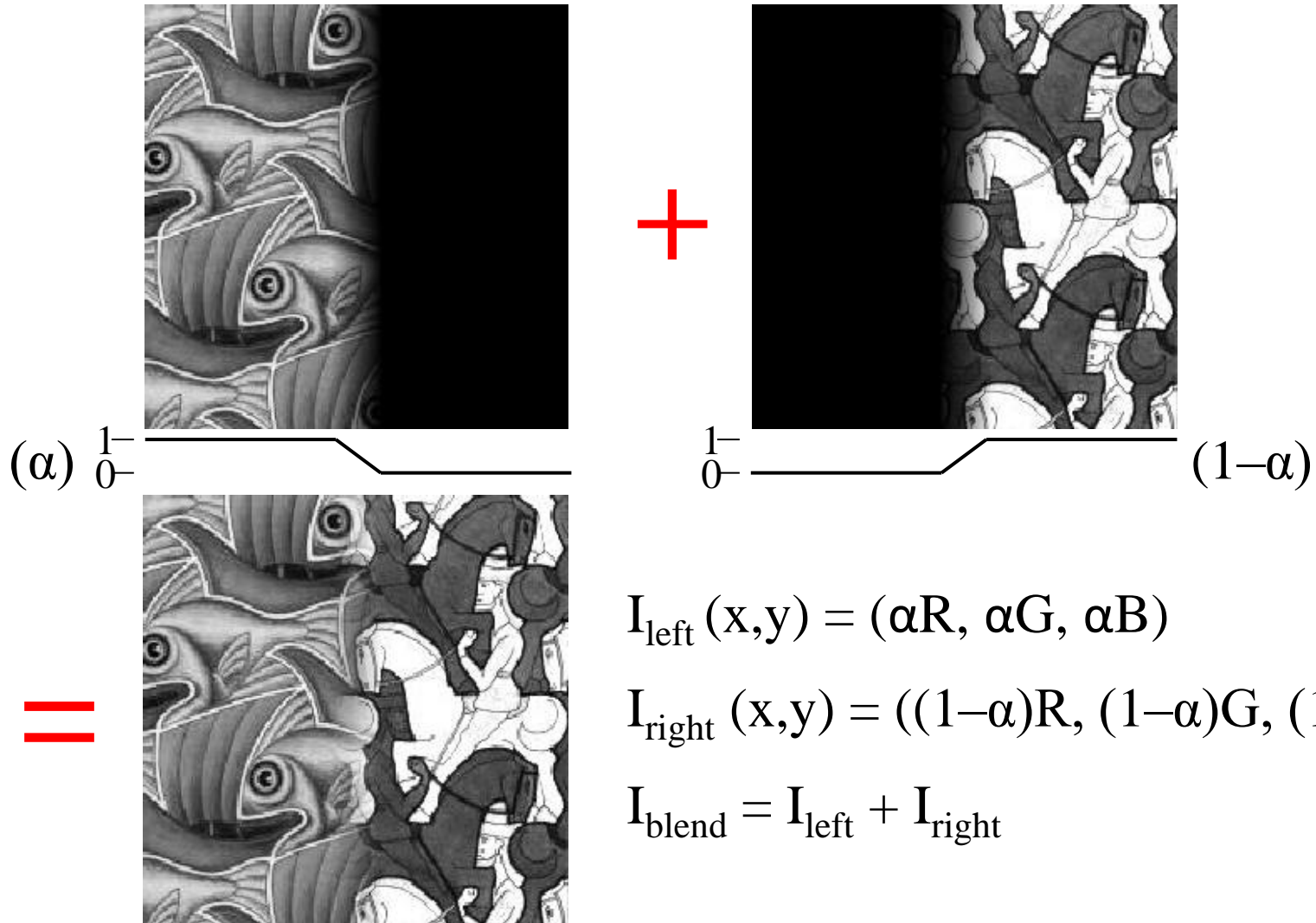
Blend the transition between images

- Feathering (Alpha blending); Pyramid blending;

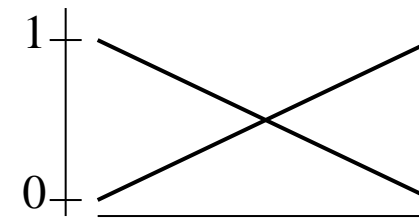
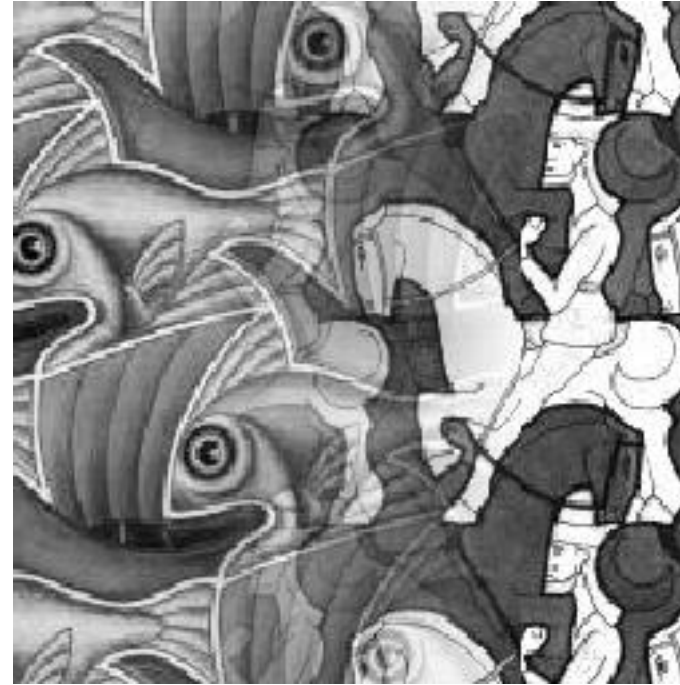
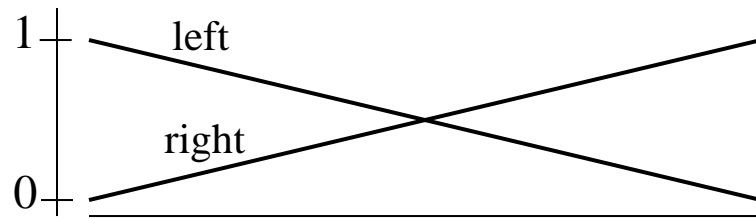
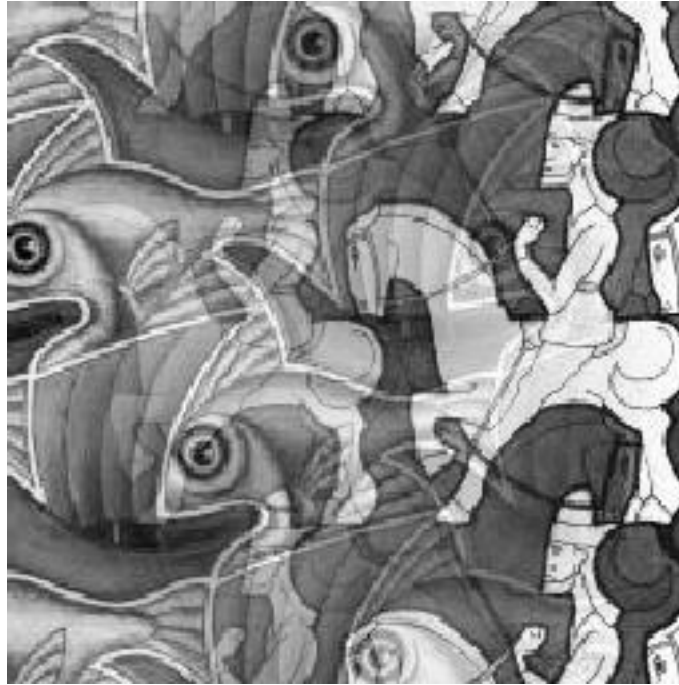
- Search for optimal seam

- Dynamic programming; Graph cuts;

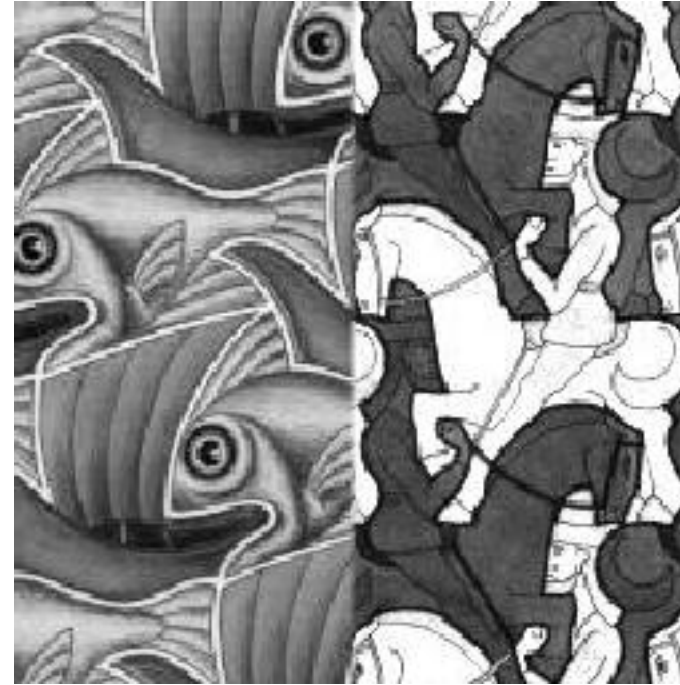
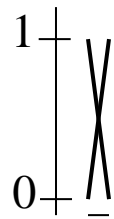
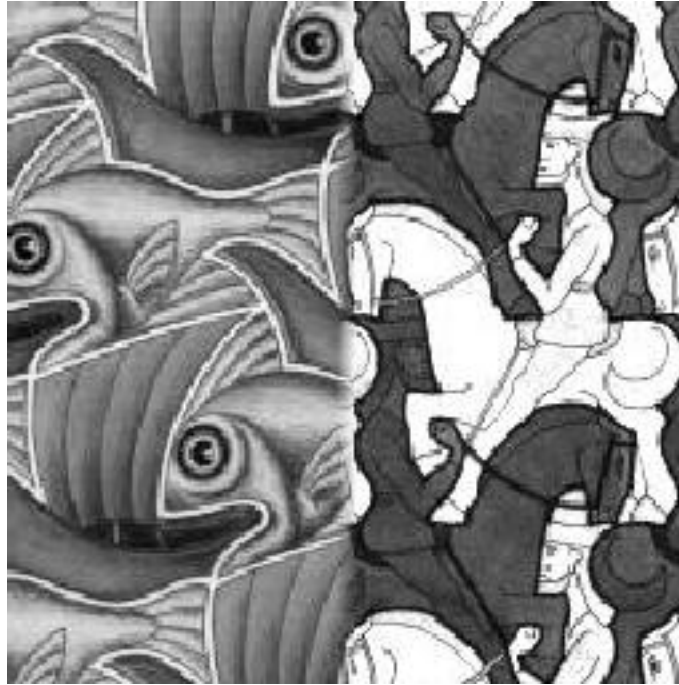
# Feathering



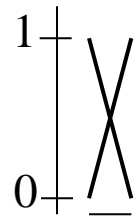
# Effect of Window Size



# Effect of Window Size



# Good Window Size

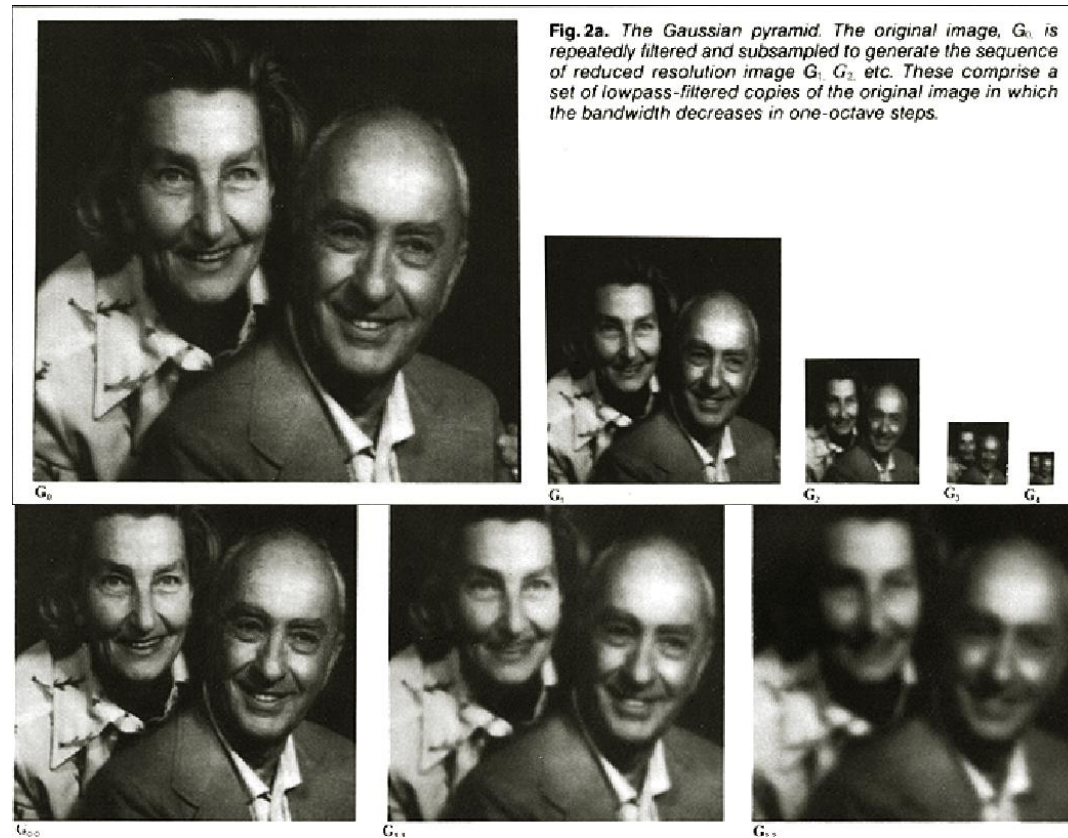


“Optimal” Window: Is smooth but not ghosted



# Band-pass filtering

# Gaussian Pyramid (low-pass images)



## Laplacian Pyramid (subband images)

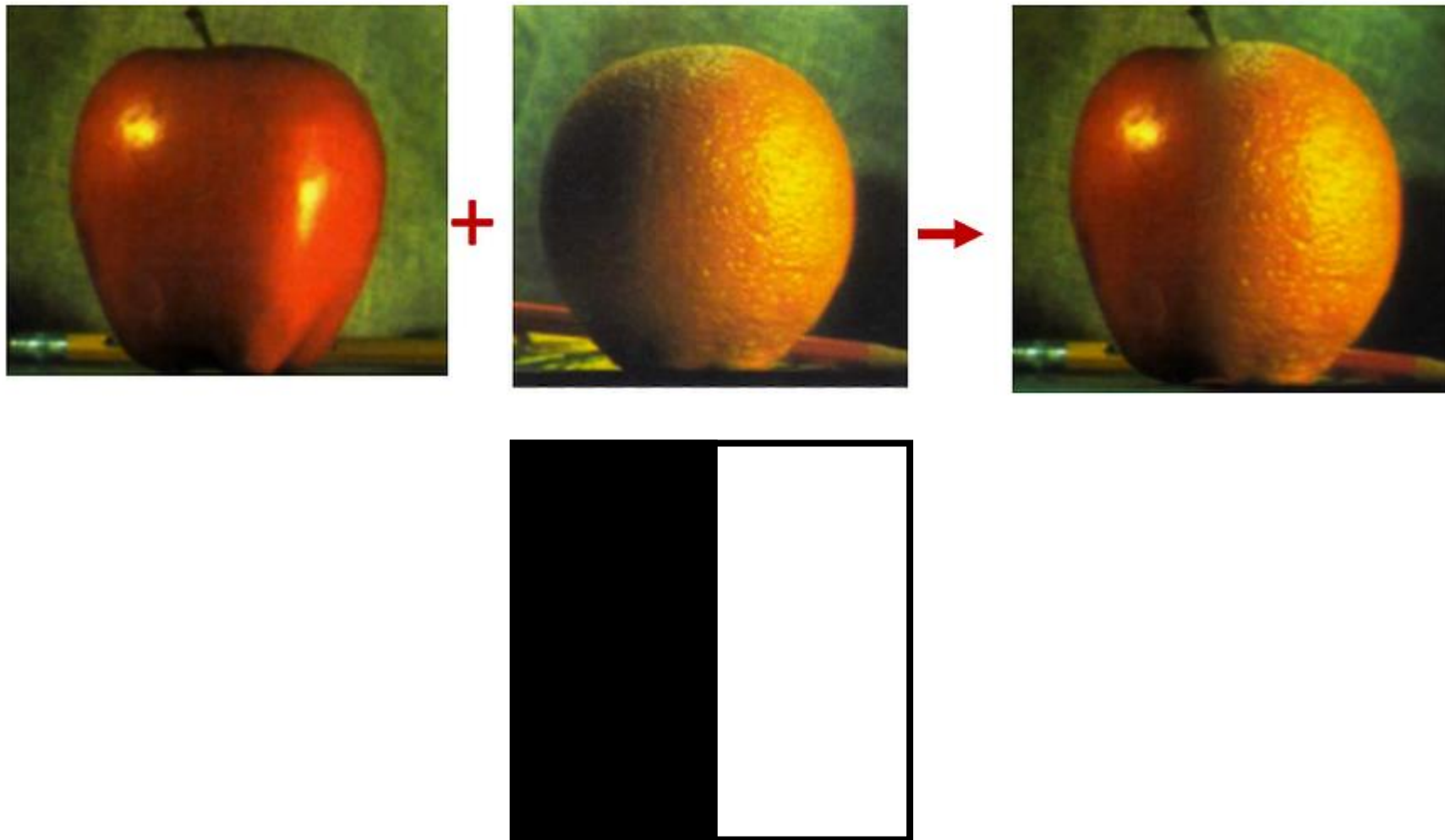
Created from Gaussian pyramid by subtraction

# Laplacian Pyramid

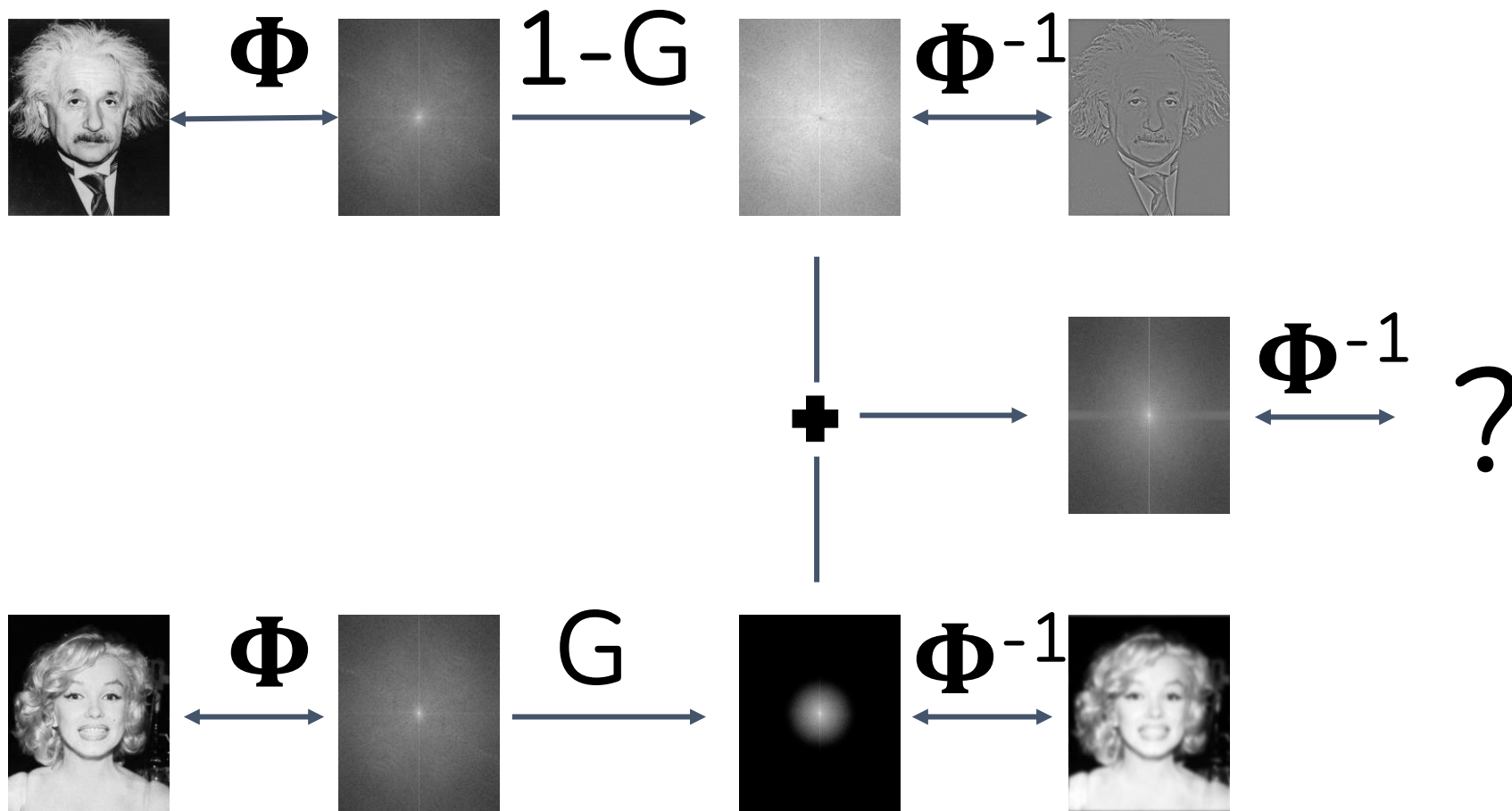


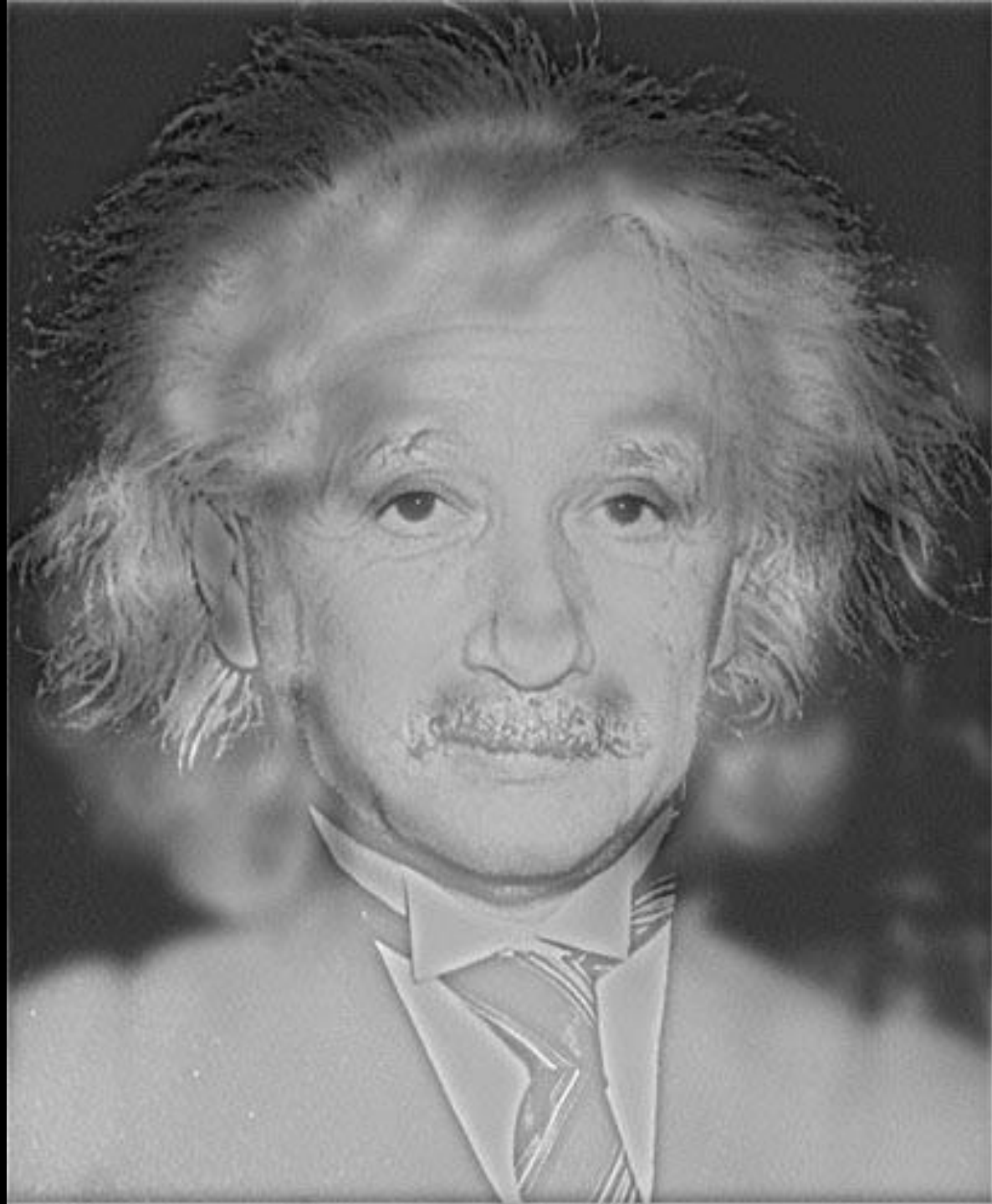
Blends **low frequencies** over a large spatial range  
and **high frequencies** over a short range

# Result



# Fourier Domain Blending?







# Approaches

Assuming that the images have already been aligned

- Simple (naïve) seam location

- Stitching

- Seam location + smoothing

Blend the transition between images

- Feathering (Alpha blending); Pyramid blending;

- Search for optimal seam

- Dynamic programming; Graph cuts;

# How to Stitch Moving Objects?



# Don't Blend, CUT!

(Search for optimal seam)



Moving objects become ghosts

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

# Where Should the Cut Pass?



# Davis, 1998

Segment the mosaic

- Single source image per segment
- Avoid artifacts along boundaries





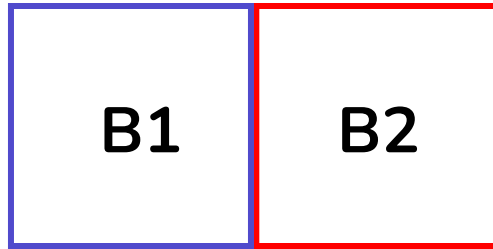
# Problems



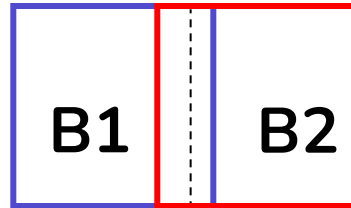


# Another Application - Texture Synthesis

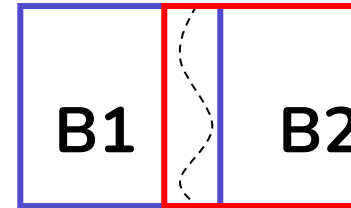
(Efros & Freeman, 2001)



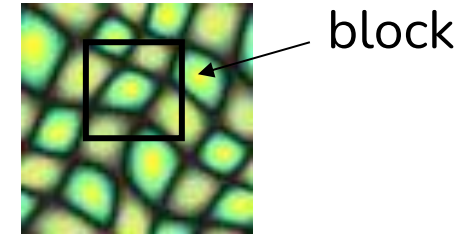
Random placement  
of blocks



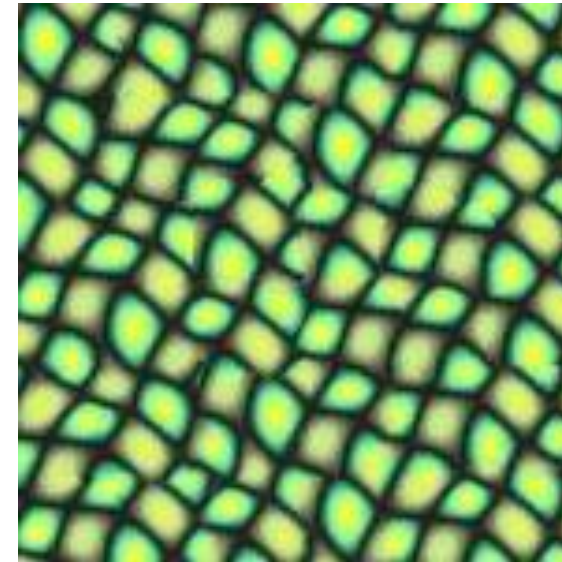
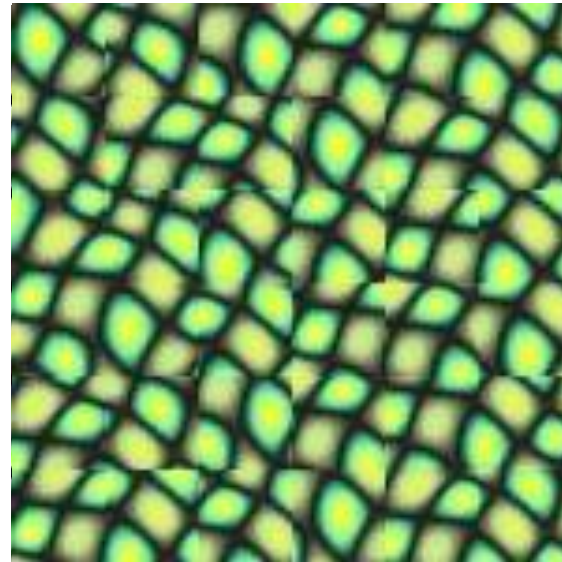
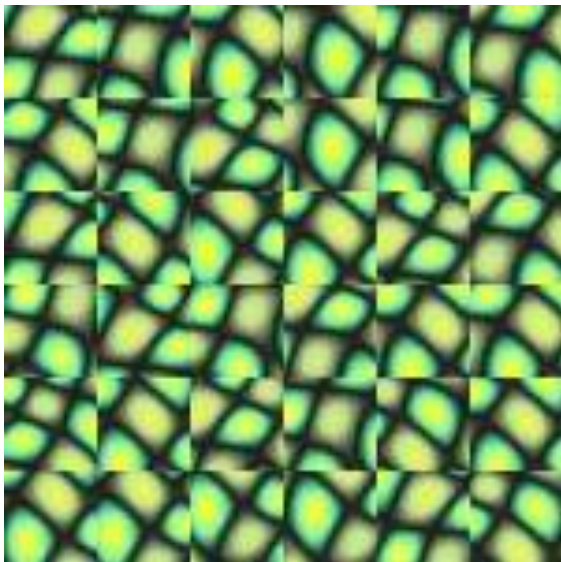
Neighboring blocks  
constrained by overlap



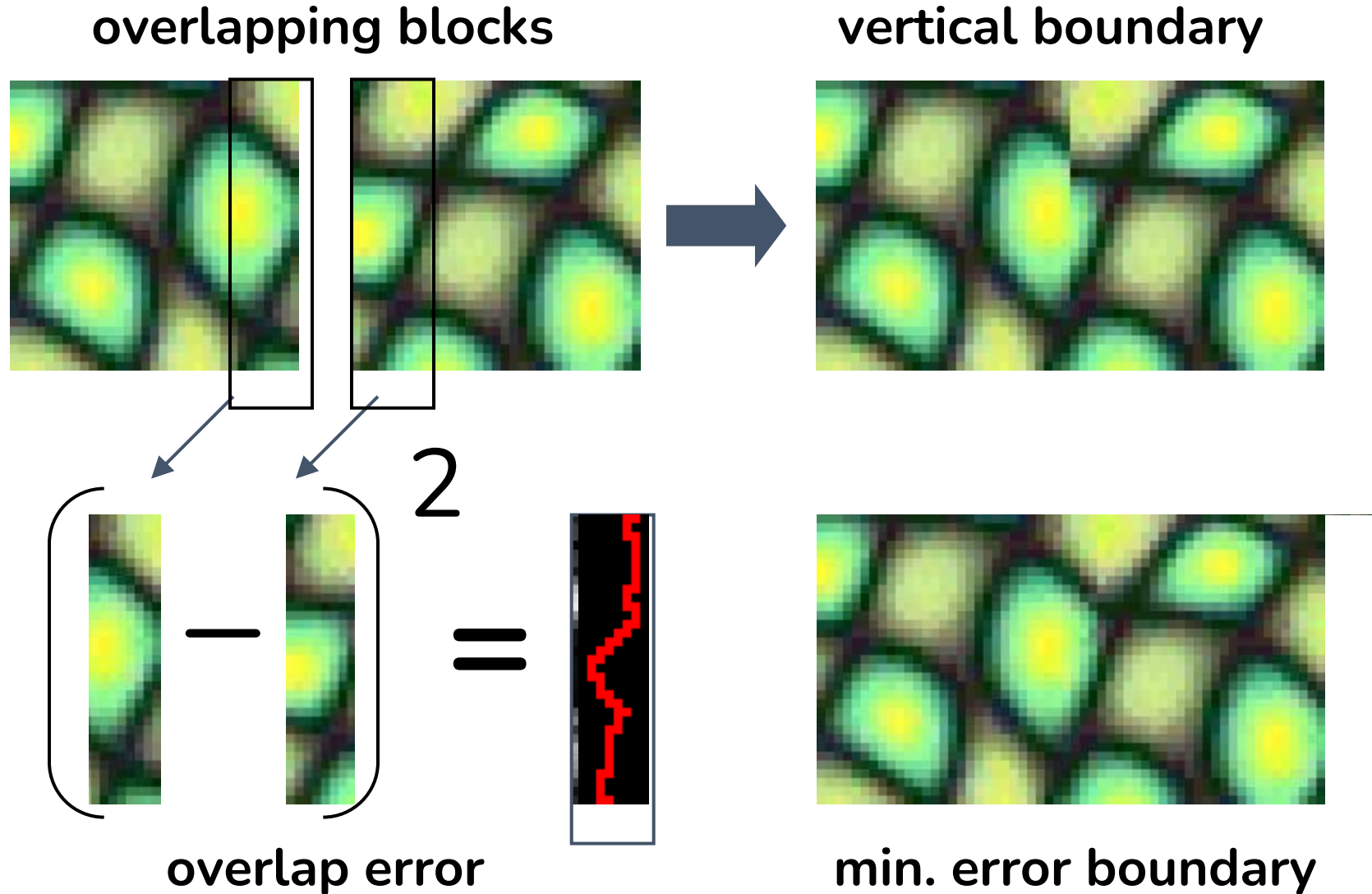
Minimal error  
boundary cut



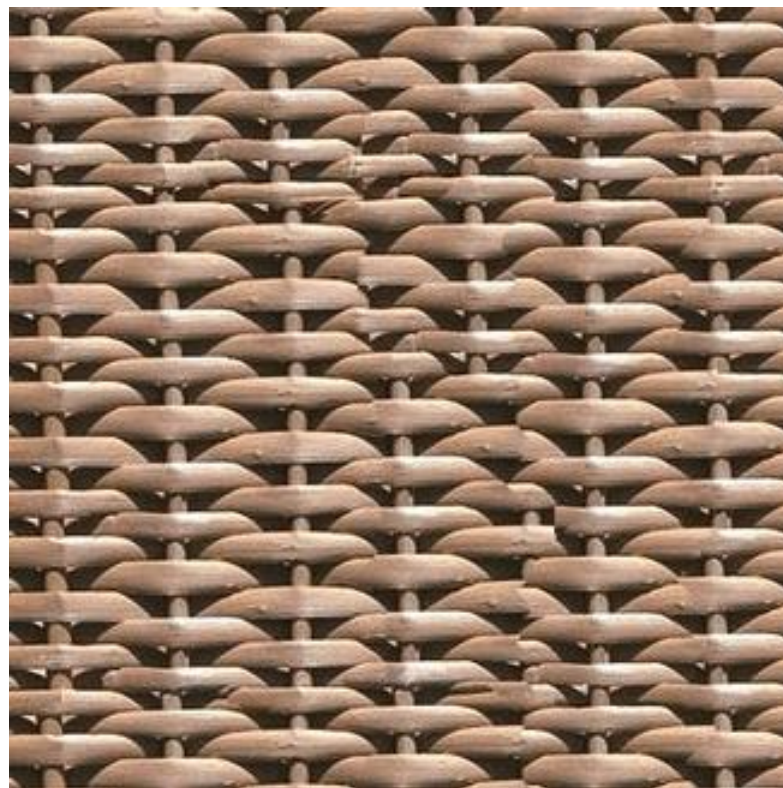
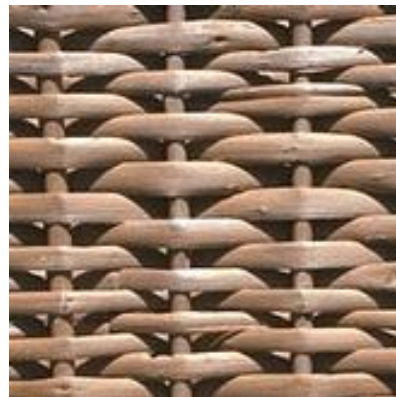
Input texture



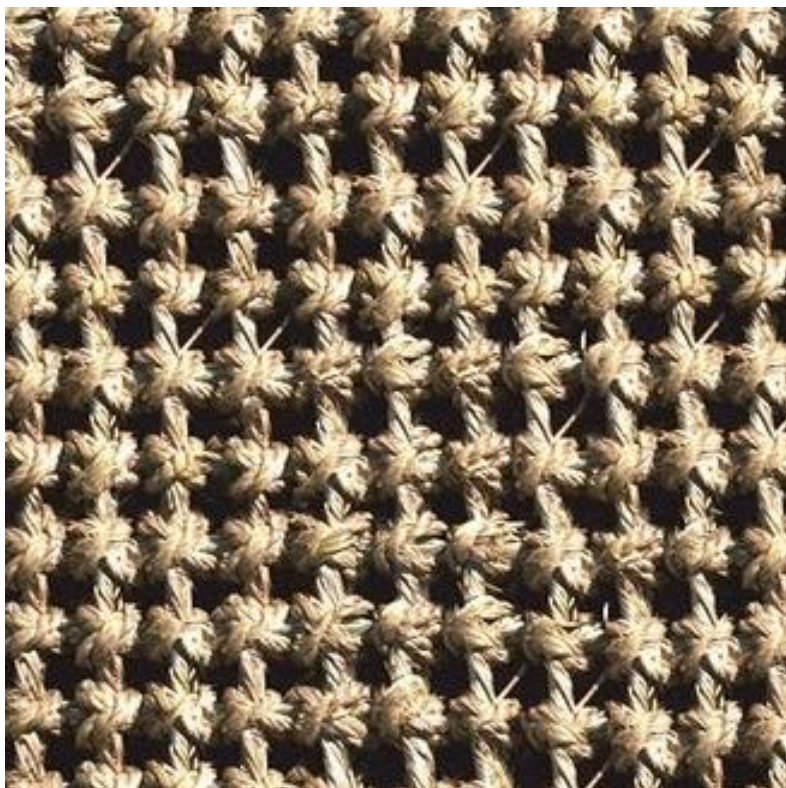
# Minimal Error Boundary









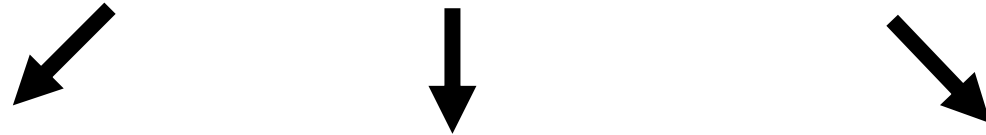


# Stitching Using Dynamic Programming


Scan the image row by row and compute a cumulative minimum squared difference  $E$  for all paths:

$$E[i, j] = e(i, j) + \min(E[i-1, j-1], E[i-1, j], E[i-1, j+1])$$

S.T



$$e = (I_1 - I_2)^2 \quad (\text{Overlapping areas in images})$$



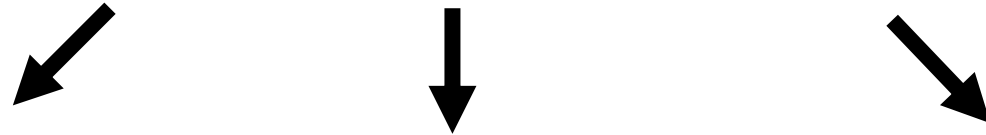
overlap error  $e$

# Stitching Using Dynamic Programming

Scan the image row by row and compute a cumulative minimum squared difference  $E$  for all paths:

$$E[i, j] = e(i, j) + \min(E[i-1, j-1], E[i-1, j], E[i-1, j+1])$$

S.T



$$e = (I_1 - I_2)^2 \quad (\text{Overlapping areas in images})$$

The optimal path can be obtained by tracing back with a minimal cost from bottom to top.

Dynamic programming can be computed in  $O(n)$  where  $n$  is the number of pixels in the overlap area (two passes...)

$e$

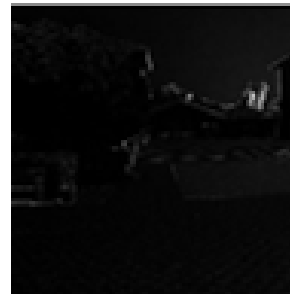


# Stitching Using Dynamic Programming

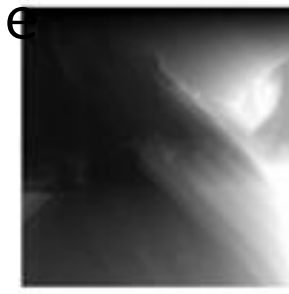
Sources images    Overlapping area



Error



Cumulative

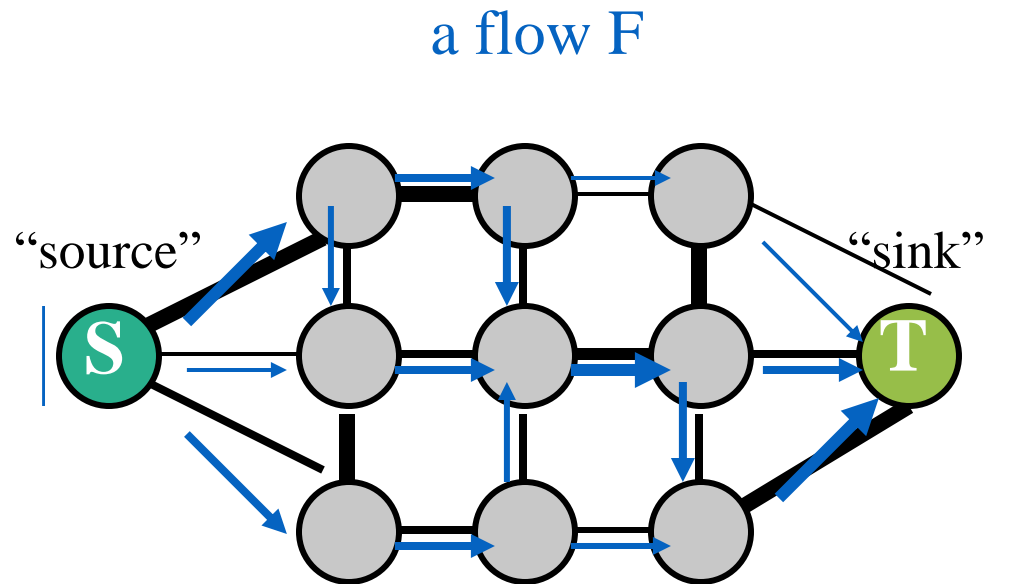


Result



dynamic programming algorithm is unable to take complex cuts

# Maximum Flow Problem

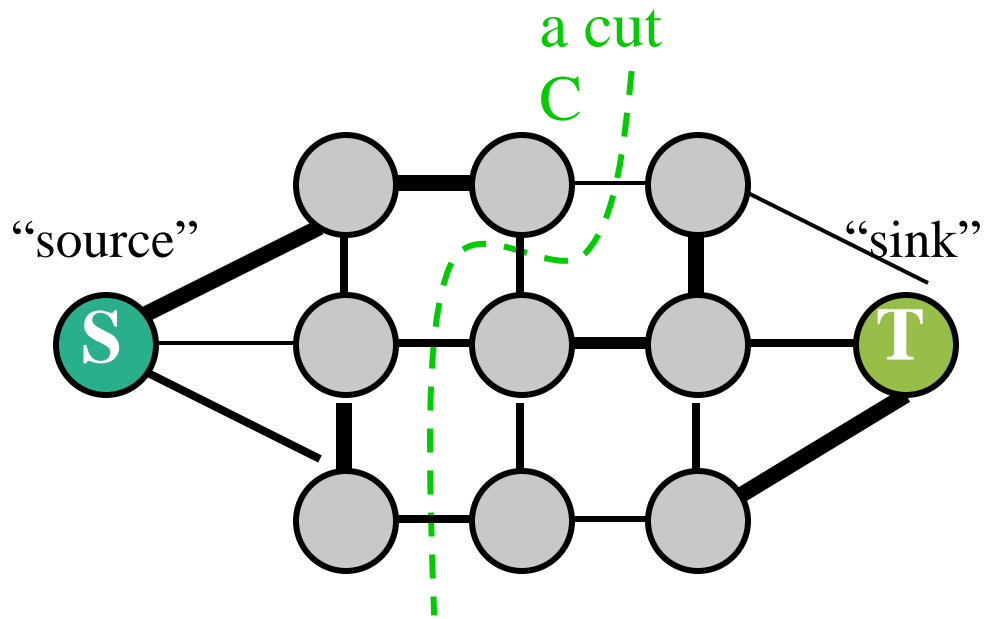


A graph with two terminals

## Max flow problem:

- Each edge is a “pipe”
- Edge weights give the pipe’s capacity
- Find the largest flow  $F$  of “water” that can be sent from the “source” to the “sink” along the pipes

# Minimum cut problem

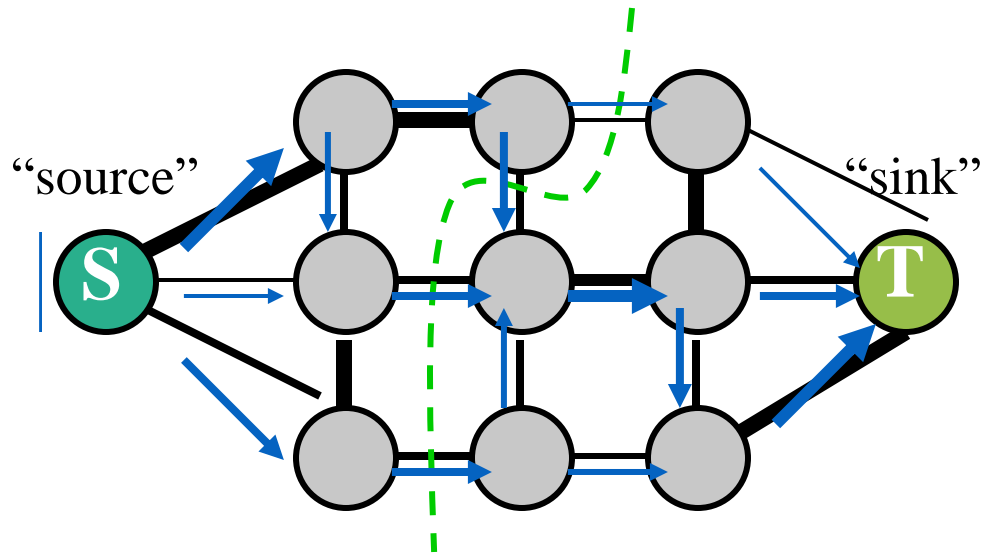


A graph with two terminals

## Min cut problem:

- Find the cheapest way to cut the edges so that the “source” is completely separated from the “sink”
- Edge weights now represent cutting “costs”

# Max Flow/Min Cut Theorem



A graph with two terminals

## Max Flow = Min Cut:

- Maximum flow saturates the edges along the minimum cut.
- Ford and Fulkerson, 1962
- Problem reduction!

Ford and Fulkerson gave first polynomial time algorithm for globally optimal solution

# How Does Min Cut Relate to Our Problem?

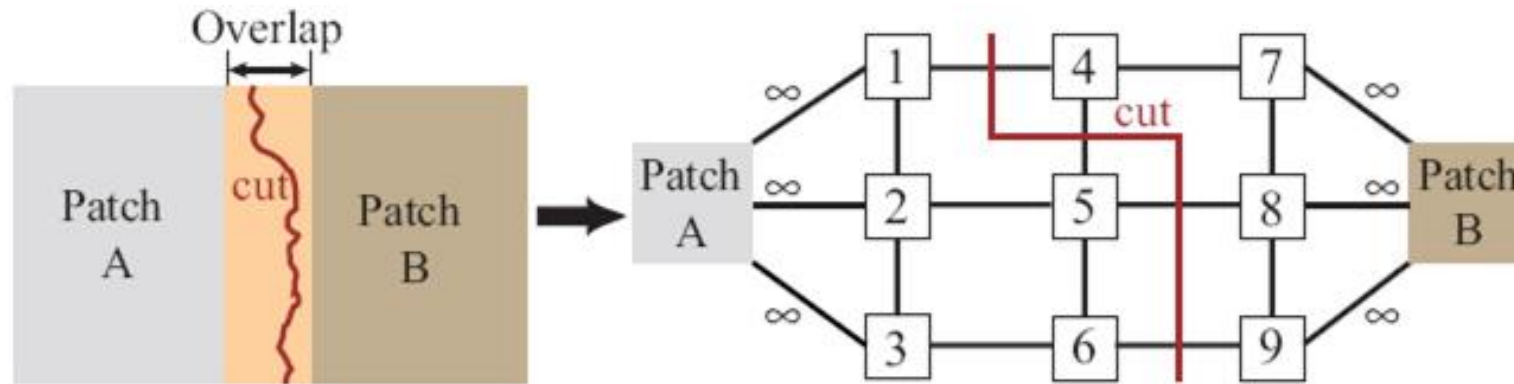


Figure 2: (Left) Schematic showing the overlapping region between two patches. (Right) Graph formulation of the seam finding problem, with the red line showing the minimum cost cut.

the selected path will run **between** pairs of pixels.



# How Does Min Cut Relate to Our Problem?

**The graph nodes:** Pixels  $p=(x,y)$ .

**The graph edges:** Pixel 4-adjacency relations.

The edge weights (flow capacities) are defined to be the color differences between the edge pixels in both images:

$$W(p_1, p_2, A, B) = \|A(p_1) - B(p_1)\| + \|A(p_2) - B(p_2)\|$$

- $p_1, p_2$  are two adjacent pixel locations
- $A(p_1)$  and  $B(p_1)$  are the pixel colors at location  $p_1$  in A and B respectively

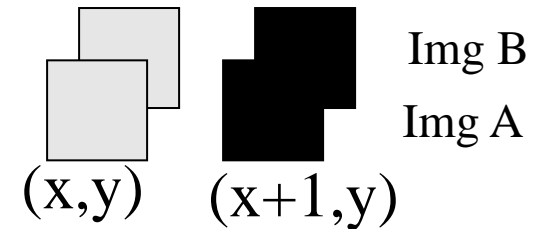
**Source and Target:** Pixels that we define explicitly to be taken from either A or B.

**The cut location determines the seam between the images**

# Edge Weights

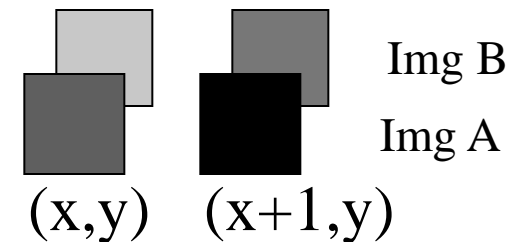
Case 1: The pair of pixels in image B has similar values to the pair in image A:

- The weight is very small and the cut “prefers” this edge.



Case 2: The pair of pixels in image B has different values than the pair in image A:

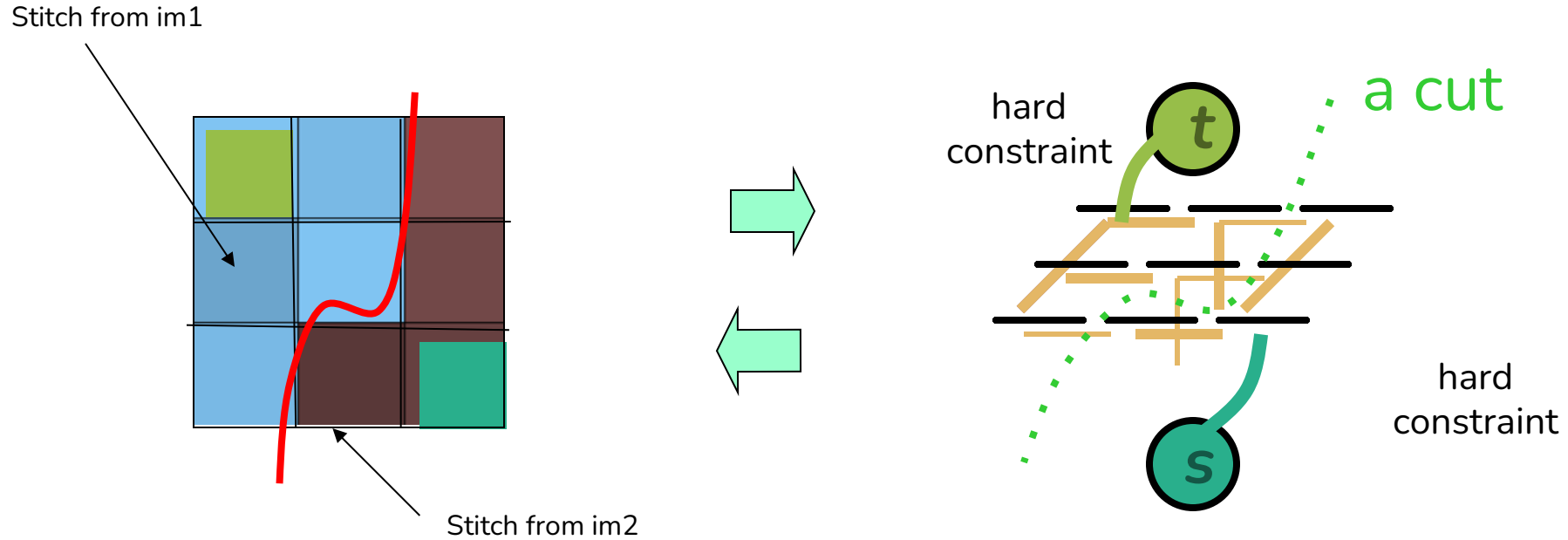
- The weight is large and the cut “avoids” this edge



$$W(p_1, p_2, A, B) = \|A(p_1) - B(p_1)\| + \|A(p_2) - B(p_2)\|$$

# Applying Min Cut on Images

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



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

# Putting it all together

Compositing images / mosaics:

Simple seam location + smoothing transitions (blending)

- Feathering (Alpha blending)
- Pyramid blending
  - Less suitable when there is miss-alignment or moving objects

Search for optimal seam

- Dynamic programming
- Min cut
  - Less suitable in case of global intensity differences

# Dynamic Panorama – Min Cut Stitching

ICCV 2005, Beijing





# Why do We Need the Min Cut?

Simple Stitching



min-cut

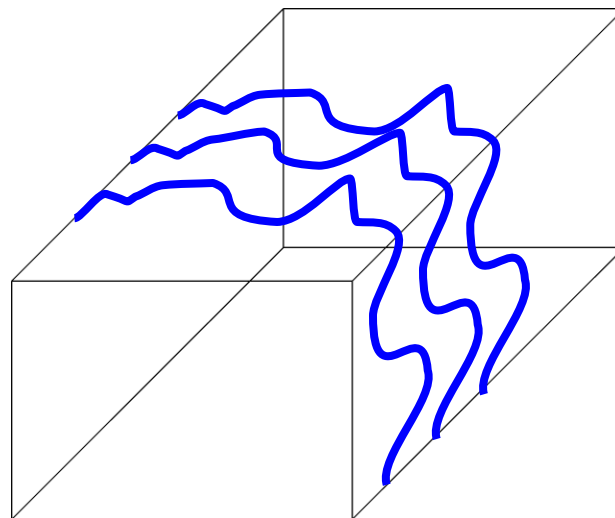


# Mosaic Stitching Examples

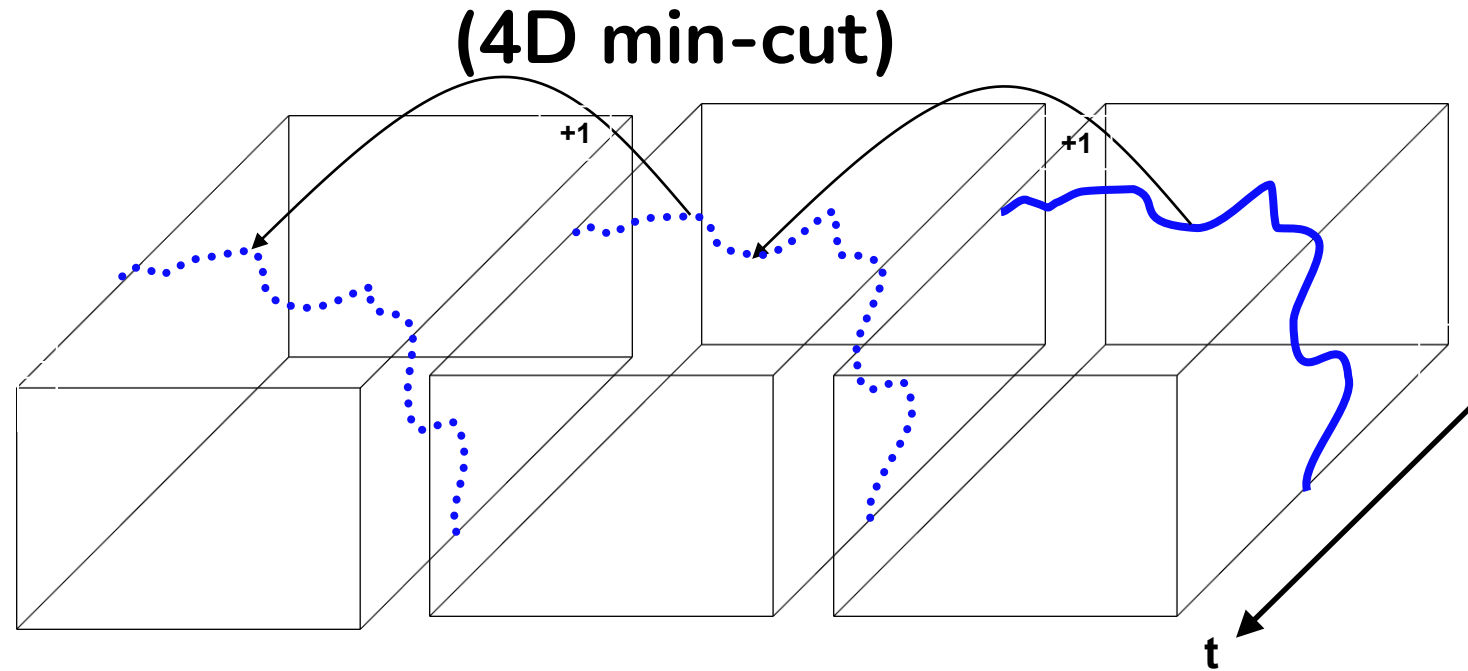


# Movie Output: Multiple Time Fronts

(4D min-cut)



# Movie Output: Multiple Time Fronts



$\chi(x, y, t) \leftarrow S((x, y, t) + e_i) - I((x, y, M(x, y, t)) + e_i)$

# Dynamic Panorama – Min Cut Stitching

ICCV 2005, Beijing



Bill Trigs   Xiaou Tang   Matthew Turk   Michael Cohen   Irfan Essa   Gerard Medioni   Jim Rehg   Shmuel Peleg   David Fleet

David Fleet left out...



Blending and Stitching

# Hough Transform

# Hough Transform

*(proposed by Hough in 1959 for the analysis of bubble chamber photographs)*

## Finding Shapes

- *Straight Lines*
- *Rectangles*
- *Circles*
- *etc.*

## More generally

The Hough Transform is a voting scheme on shape parameters.

# Applications

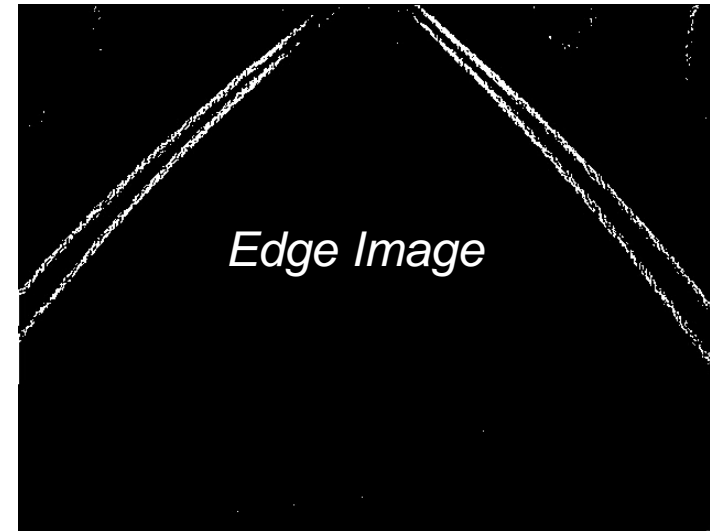
- Find signs in an image (detect rectangles, circles).
- 3D reconstruction of buildings (detect façade lines).
- Find the eye in a face image, a ball in a scene.
- More...

## **The challenge:**

- The image usually contains lots of irrelevant information (edges unrelated to the shape or due to noise).
- There are partial occlusions.



# Detecting Lines from Edge Images?





# Detecting Lines from Edge Images?

- Given an edge image, we would like to group sets of edge pixels to identify dominant lines in an image.
- One possibility is to perform **template matching** in the edge image
  - Generate a hypothetical line “kernel” and convolve with the image
  - Requires a large set of masks to accurately localize lines, which is computationally expensive
- We will instead perform an **edge image transformation** where edge pixels **vote** for possible lines.
  - Eventually reduces the problem to vote thresholding

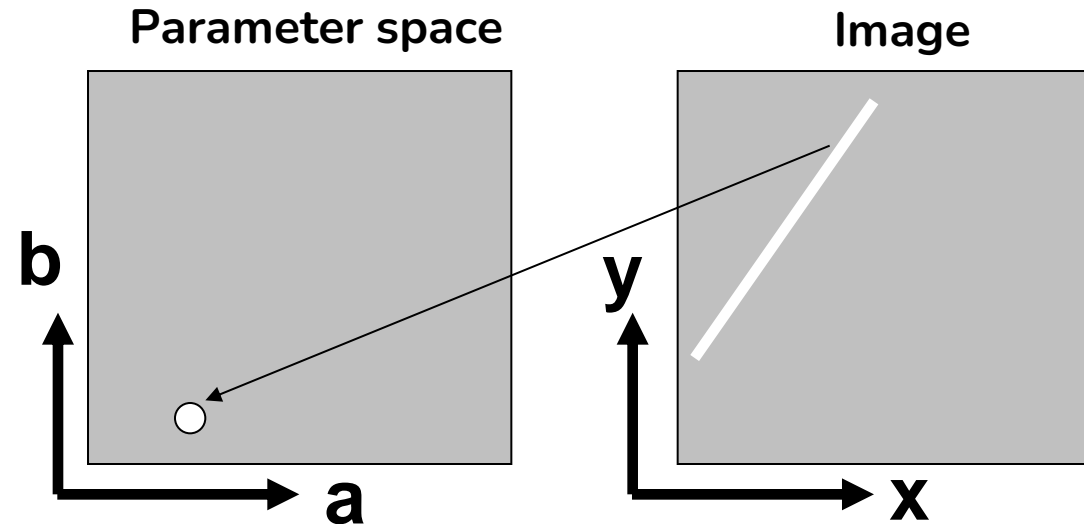
# The Parameter Space

- The line equation in slope-intercept representation:

$$y = ax + b$$

- Lines can be represented with the 2 parameters (a, b).

$$y = 2x + 3$$



# The Parameter Space

- The line equation in slope-intercept representation:  
 $y = ax + b$
- Lines can be represented with the 2 parameters (a, b).
- If we define a 2D parameter space, each **line** is mapped to a single **point** in this space.

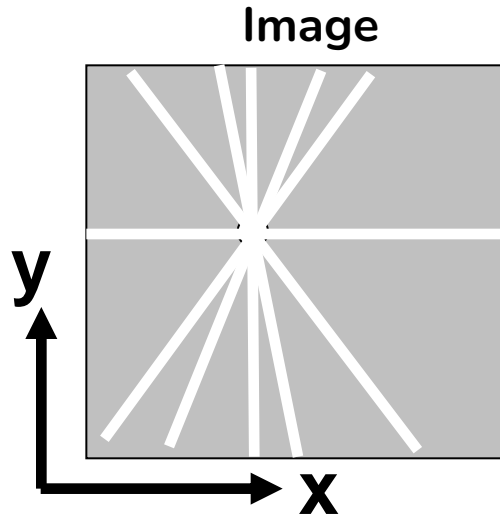
# The Parameter Space

- For each point  $(x, y)$  in the image there are infinite number of points  $(a, b)$  passing through:

$$y = ax + b$$



$$b = -xa + y$$



# The Parameter Space

- For each point  $(x, y)$  in the image there are infinite number of points  $(a, b)$  passing through:

$$b = -xa + y$$

*For Example:*

$$(x, y) = (6, 8)$$

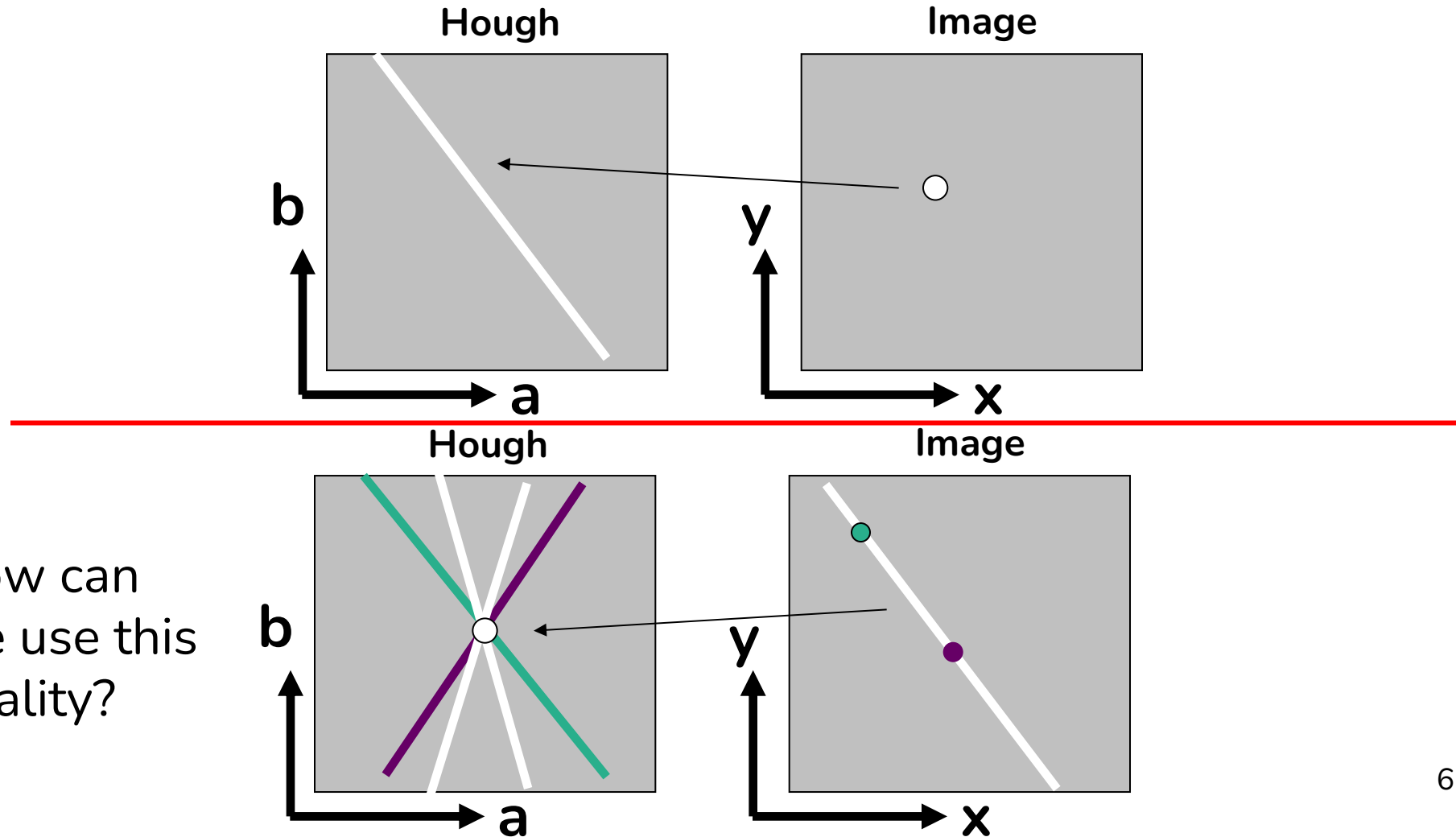
$$b = -6a + 8$$

*And this is a line*



# The duality property

*A point is mapped to a line, and a line is mapped to a point.*



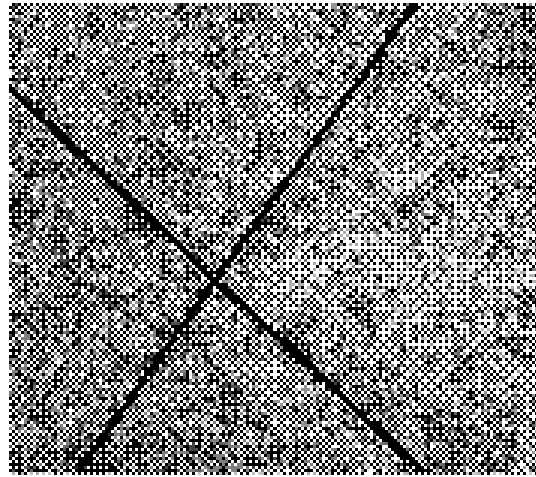
How can  
we use this  
duality?

# Hough Transform: Use a Voting Scheme

- Apply an **edge detector** on the image (gradient + threshold, or using Canny's method)
- Prepare a **table  $h(a,b)$**  for  $a,b$  in a certain range.
- Loop over the image: for each pixel  $(x,y)$  on an edge, **vote** for all the cells  $(a,b)$  which satisfy the equation:  
 $y = ax + b$  (meaning,  
increase the cell counter:  **$h(a,b)++$**  )
- Choose the cell having **maximal value** (or the cells which pass a given threshold)

# Example: Slope-Intercept Representation

*Original  
image*



(A)

*Edge Map*

*Hough  
(before  
threshold)*

*Take two  
strongest  
lines*

# Issues to Consider...

- **Representation:** Vertical lines are not covered by the representation  $y = ax+b$ .
- **Range:** Selecting the range of values for the table
- **Quantization:** choosing a grid.

# Next week:

Hough Transform  
Colors