CS4670 / 5670: Computer Vision

Kavita Bala

Lecture 20: Panoramas

Announcements

- **Prelim on Thu**
  - Everything till Lecture 17
  - Closed book
  - Bring your calculator
  - **7:30 pm, Location**
    - **Kennedy Hall, 116**
- **Review on Tuesday**
  - **5-7pm, Olin 255**

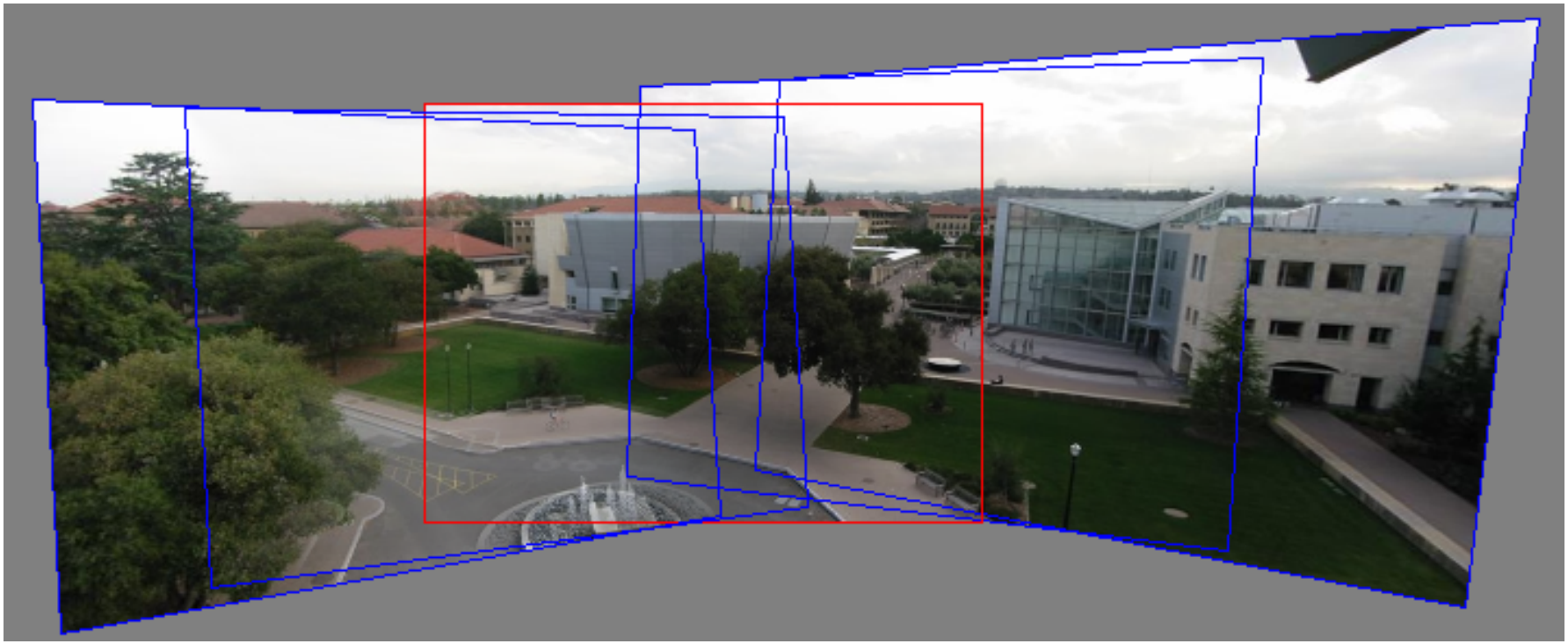
Mosaics

- How do we align the images?

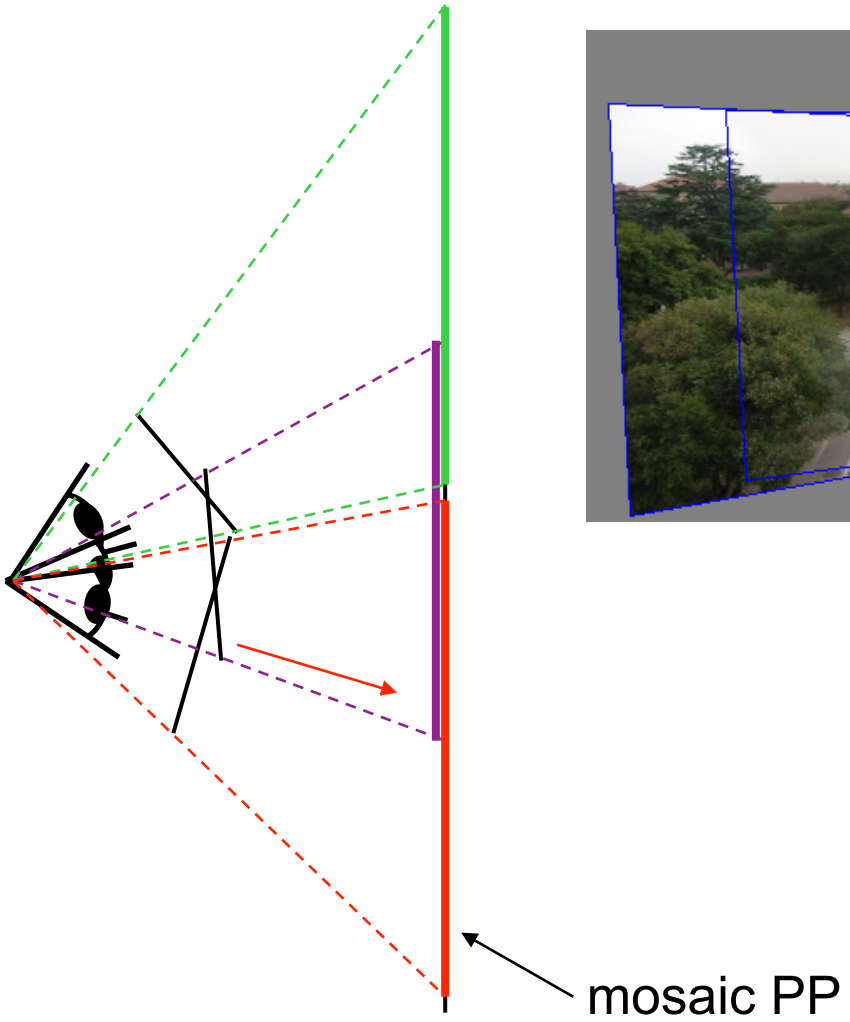
Creating a panorama

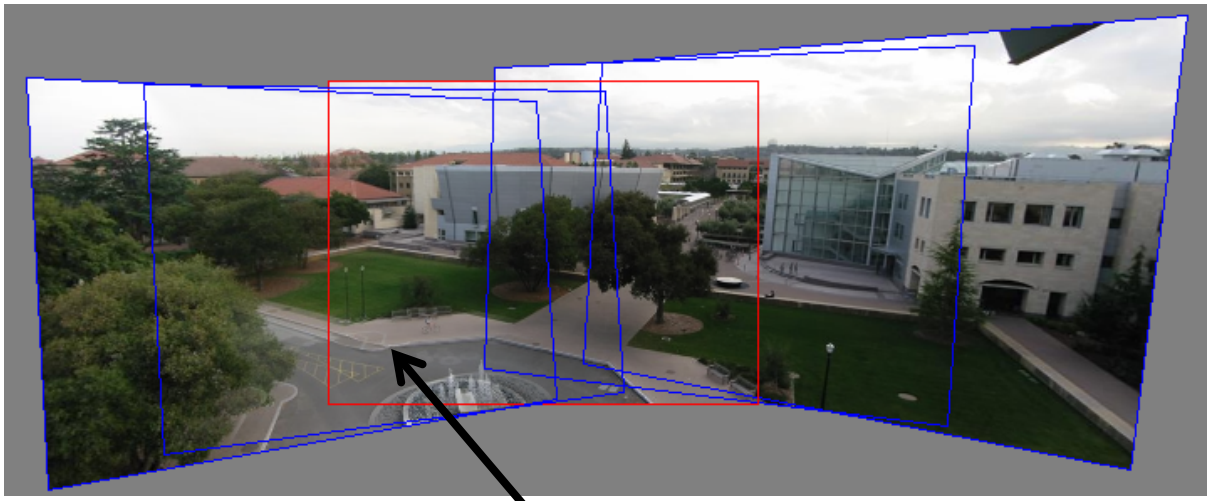
- Basic Procedure
  - Take a sequence of images from the same position
    - Rotate the camera about its optical center
  - Compute transformation between second image and first
  - Transform the second image to overlap with the first
  - Blend the two together to create a mosaic
  - If there are more images, repeat

Can we use homography to create a 360 panorama?



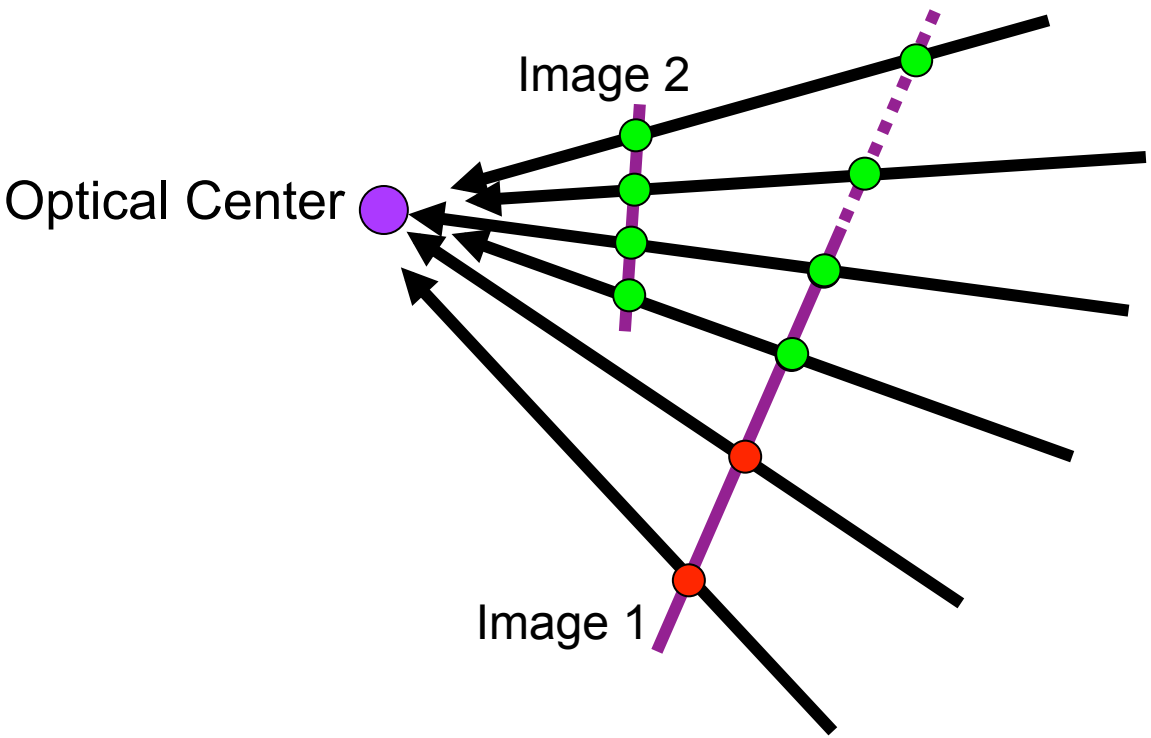
Idea: projecting images onto a common plane





each image is warped
with a homography $\mathbf{H}$

Geometric Interpretation of Mosaics

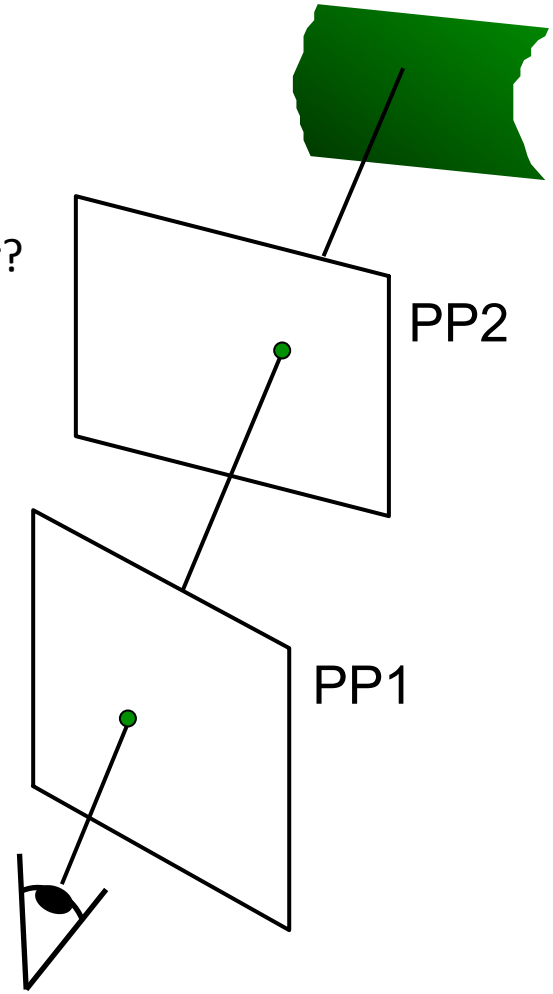


- If we capture all 360º of rays, we can create a 360º panorama
- The basic operation is *projecting* an image from one plane to another
- The projective transformation is scene-INDEPENDENT
  - This depends on all the images having the same optical center

Image reprojection

- Basic question
  - How to relate two images from the same camera center?
    - how to map a pixel from PP1 to PP2

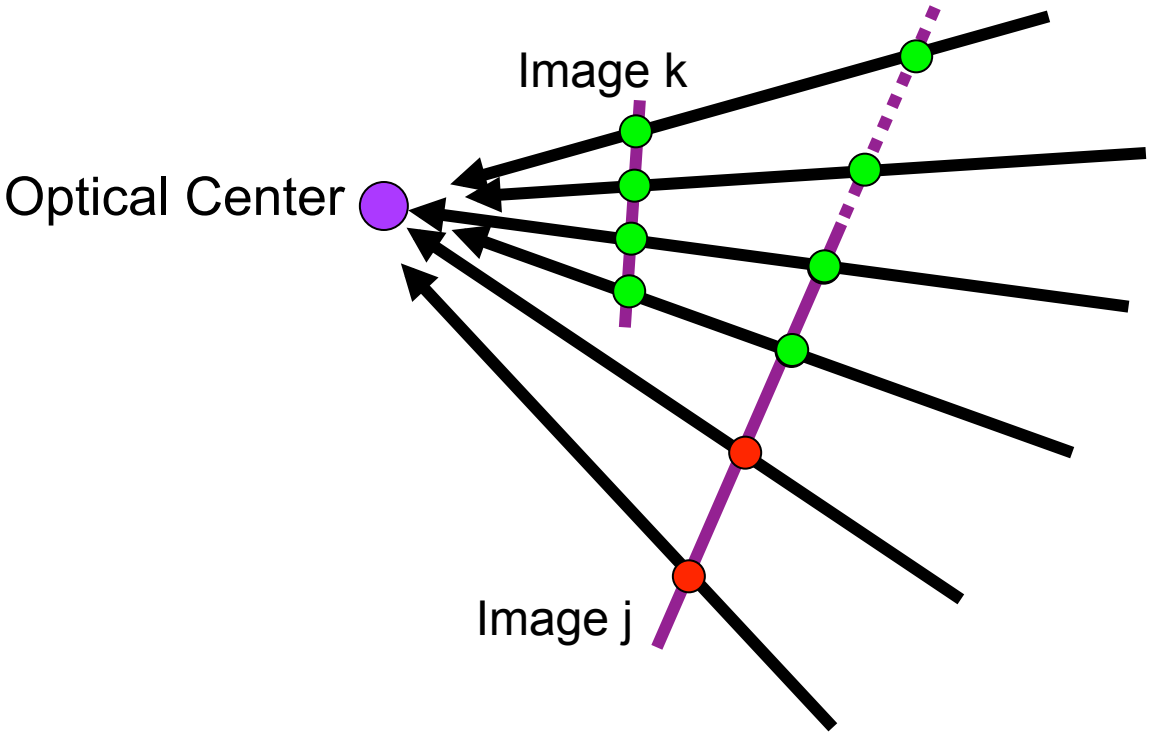
  Answer
    - Cast a ray through each pixel in PP1
    - Draw the pixel where that ray intersects PP2



Projection matrix

$$\mathbf{\Pi} = \underbrace{\mathbf{K}}_{\text{intrinsics}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{projection}} \underbrace{\begin{bmatrix} & \mathbf{R} & & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{rotation}} \underbrace{\begin{bmatrix} & \mathbf{I}_{3\times 3} & & -\mathbf{c} \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{translation}}$$

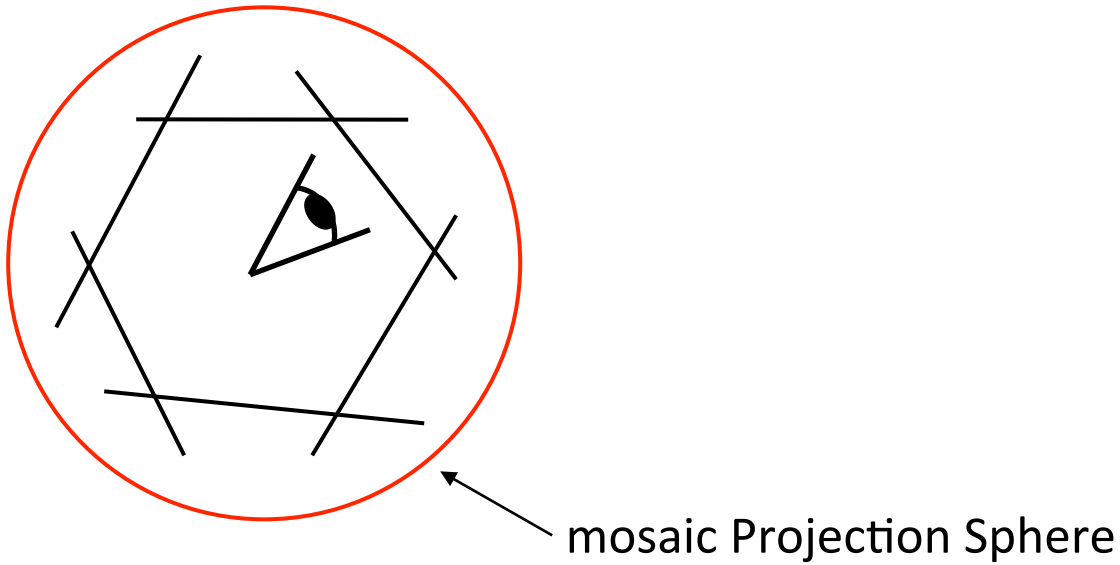
What is the transformation?



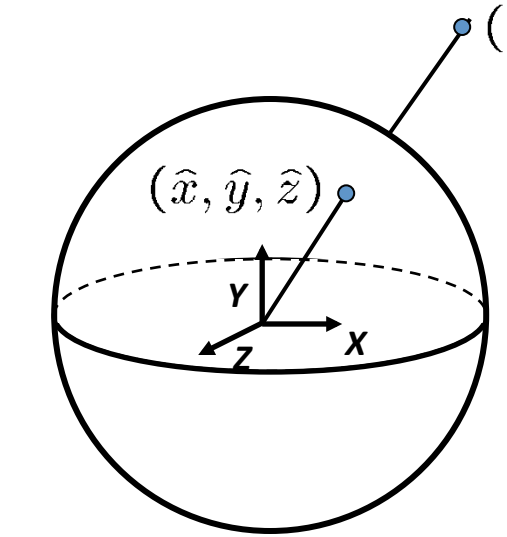
$$\tilde{\boldsymbol{x}}_{ik} \sim \tilde{\boldsymbol{H}}_{kj}\tilde{\boldsymbol{x}}_{ij} = \boldsymbol{K}_k\boldsymbol{R}_k\boldsymbol{R}_j^{-1}\boldsymbol{K}_j^{-1}\tilde{\boldsymbol{x}}_{ij}.$$

Panoramas

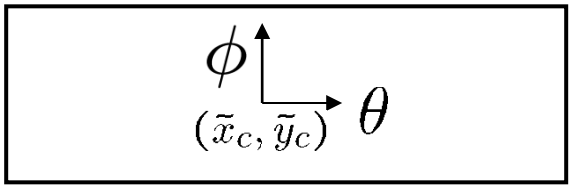
- What if you want a 360° field of view?



Spherical projection



unit sphere



unwrapped sphere

- Map 3D point (X,Y,Z) onto sphere

$$(\hat{x}, \hat{y}, \hat{z}) = \frac{1}{\sqrt{X^2+Y^2+Z^2}}(X, Y, Z)$$

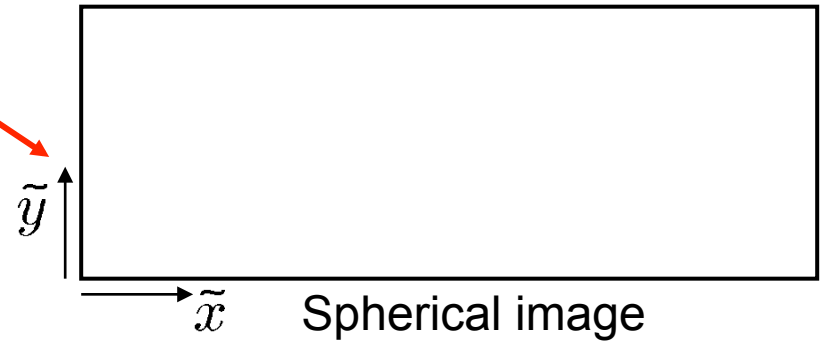
- Convert to spherical coordinates

$$(sin\theta cos\phi, sin\phi, cos\theta cos\phi) = (\hat{x}, \hat{y}, \hat{z})$$

- Convert to spherical image coordinates

$$(\tilde{x}, \tilde{y}) = (s\theta, s\phi) + (\tilde{x}_c, \tilde{y}_c)$$

  - s defines size of the final image
    - » often convenient to set s = camera focal length in pixels



Spherical image

Spherical reprojection

input **f = 200 (pixels)** **f = 400** **f = 800**

- Map image to spherical coordinates
  - need to know the focal length

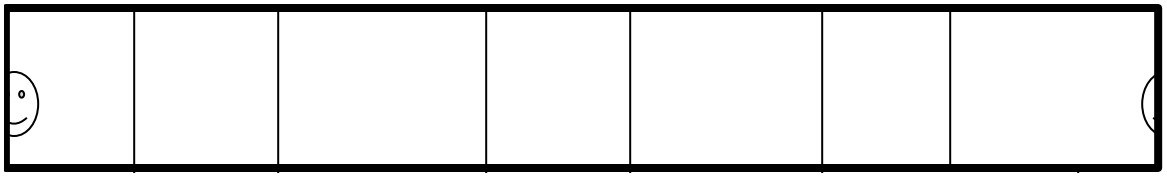
Aligning spherical images

- Suppose we rotate the camera by θ about the vertical axis
  - How does this change the spherical image?

Aligning spherical images

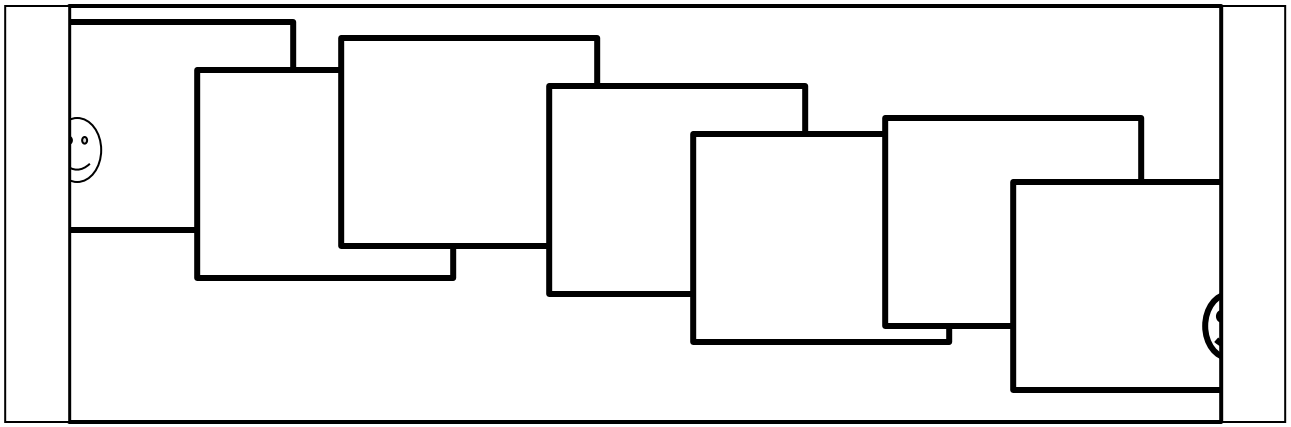
- Suppose we rotate the camera by θ about the vertical axis
  - How does this change the spherical image?
    - Translation by θ
  - This means that we can align spherical images by translation

Assembling the panorama



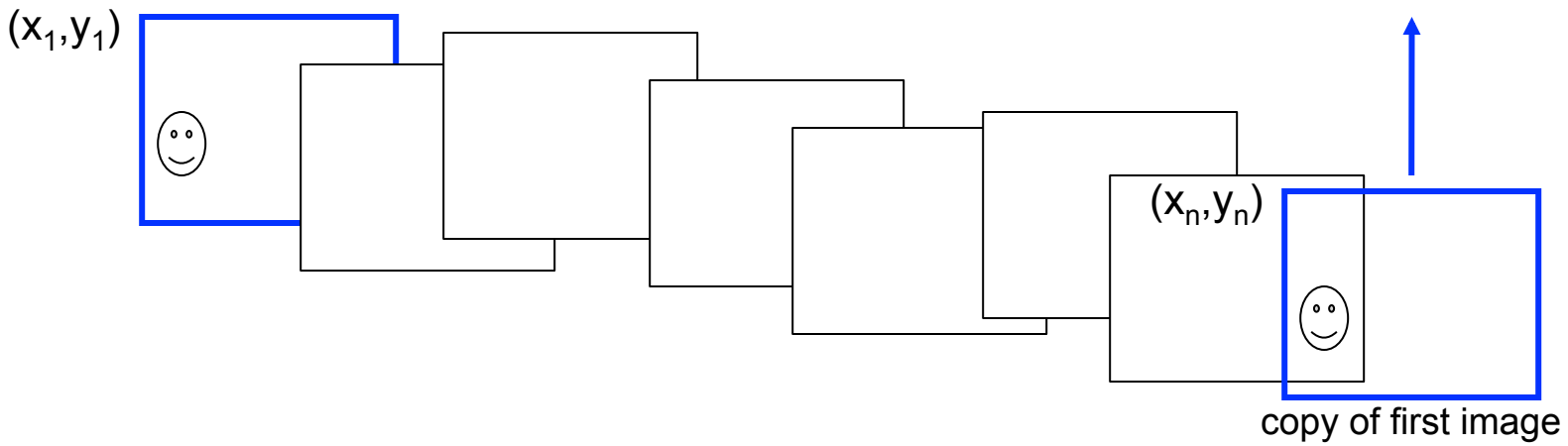
- Stitch pairs together, blend, then crop

Problem: Drift



- Error accumulation
  - small errors accumulate over time

Problem: Drift



- Solution
  - add another copy of first image at the end
  - this gives a constraint: $y_n = y_1$
  - there are a bunch of ways to solve this problem
    - add displacement of $(y_1 - y_n)/(n-1)$ to each image after the first
    - **apply an affine warp: $y' = y + ax$ [you will implement this for P3]**
    - run a big optimization problem, incorporating this constraint
      - best solution, but more complicated
      - known as "bundle adjustment"

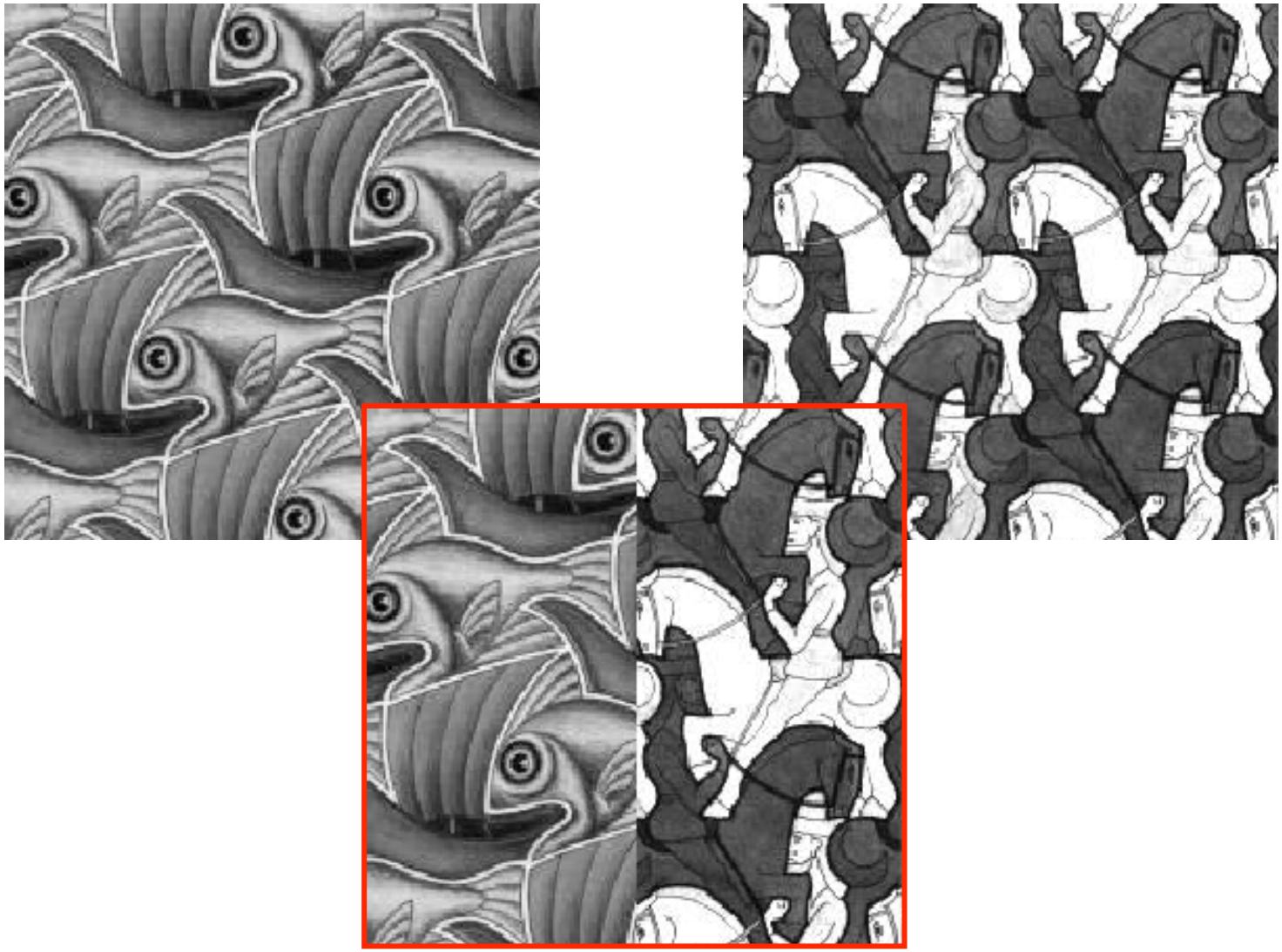
Blending

- We’ve aligned the images – now what?

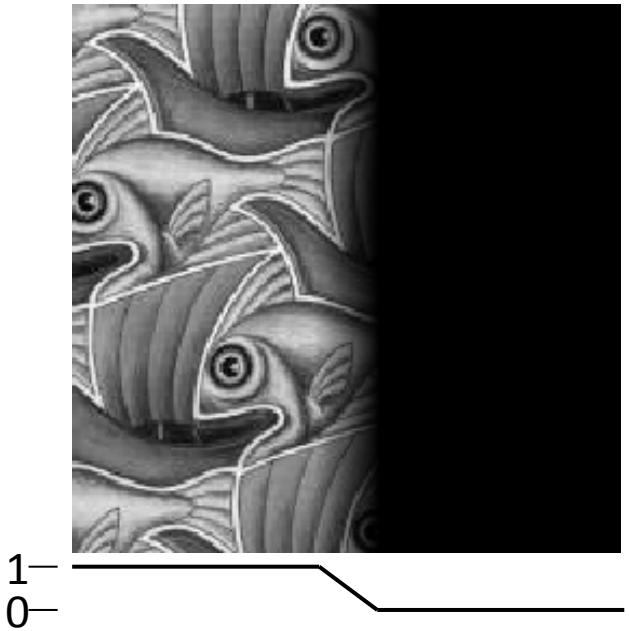
Blending

- Want to seamlessly blend them together

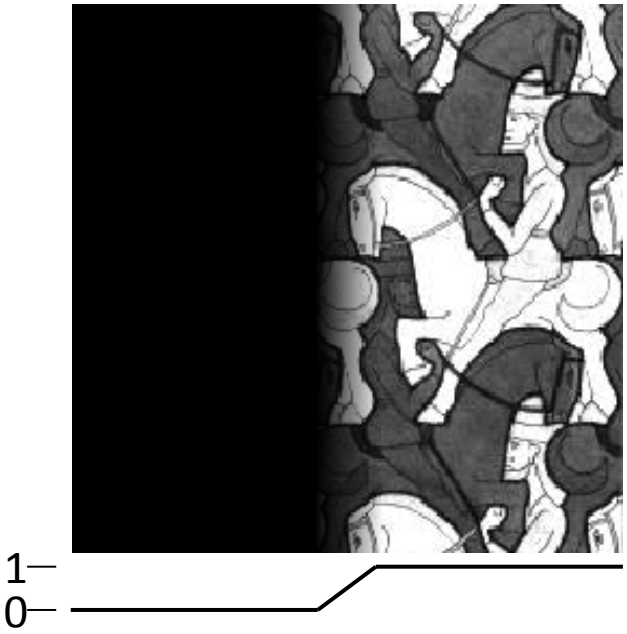
Image Blending



Feathering

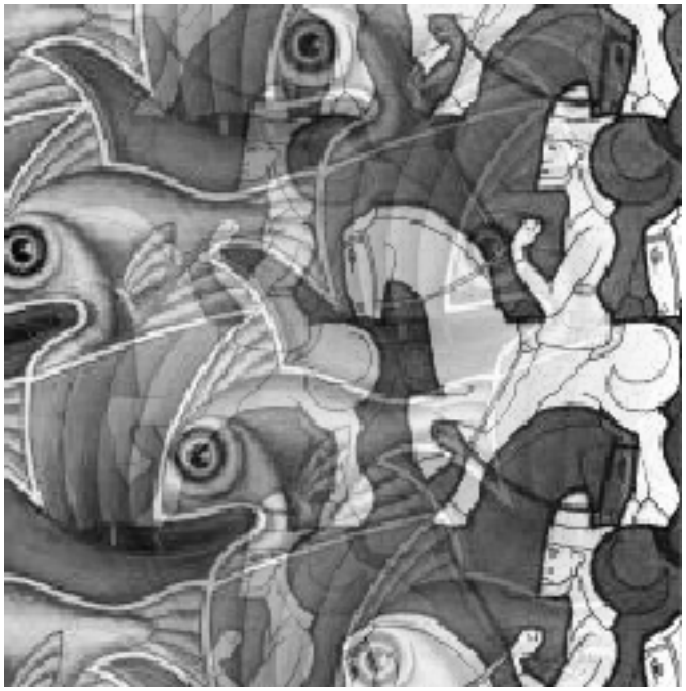


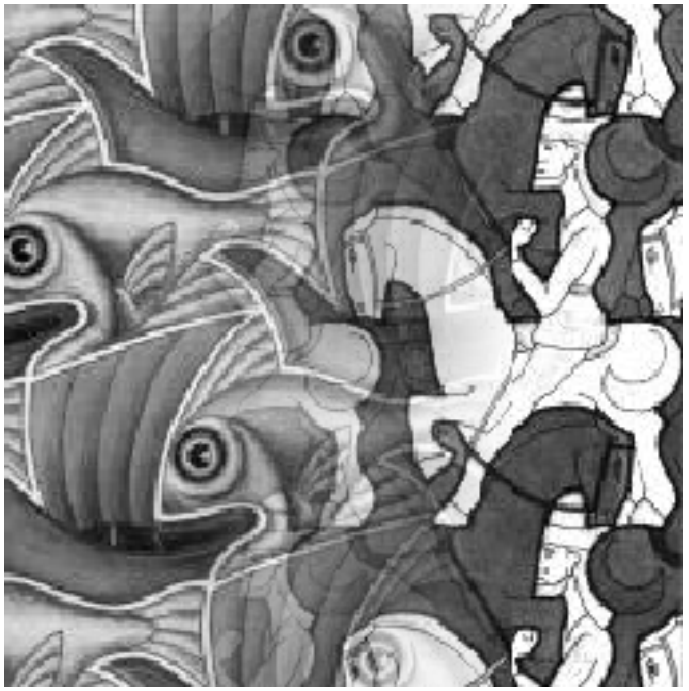
+

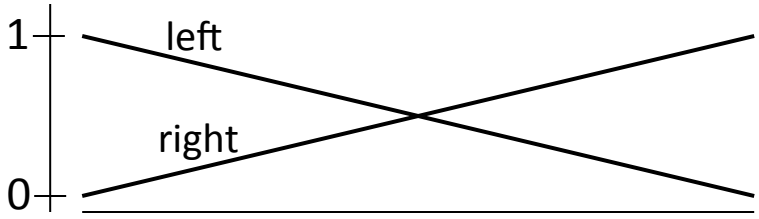


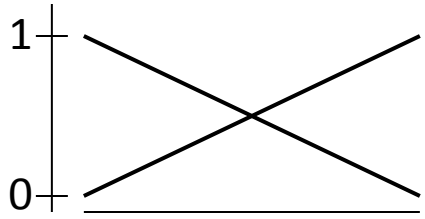
=

Effect of window size

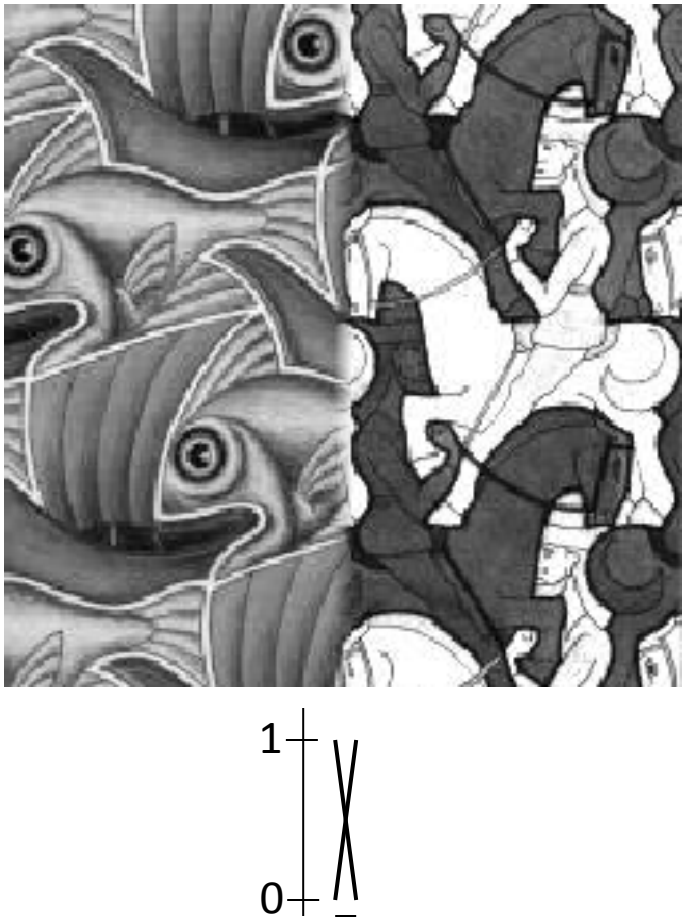


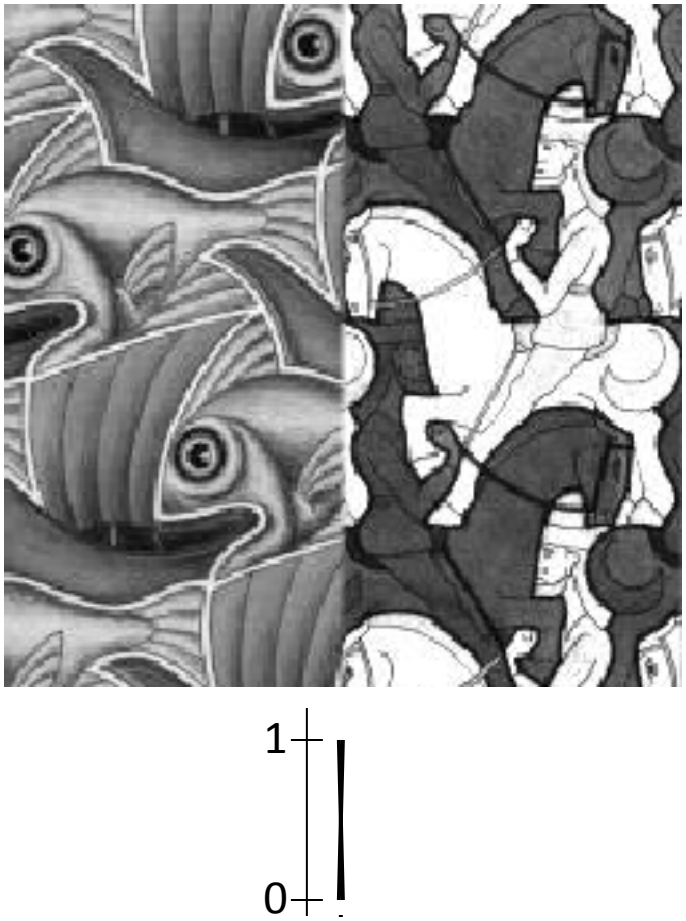




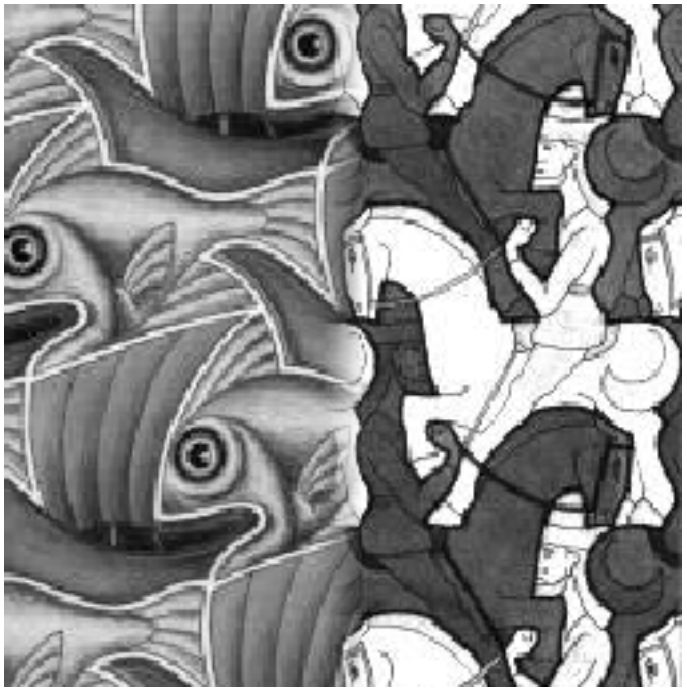


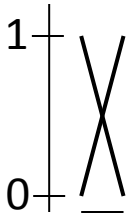
Effect of window size





Good window size





"Optimal" window: smooth but not ghosted

- Doesn't always work...

What is the optimal size?

- To avoid seams
  - Window >= size of largest prominent feature
- To avoid ghosting
  - Window <= 2 * size of smallest prominent feature