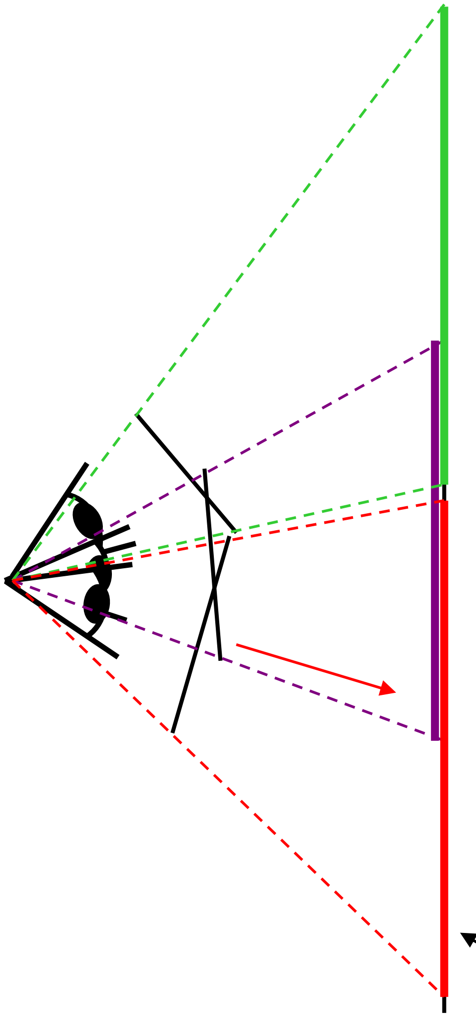
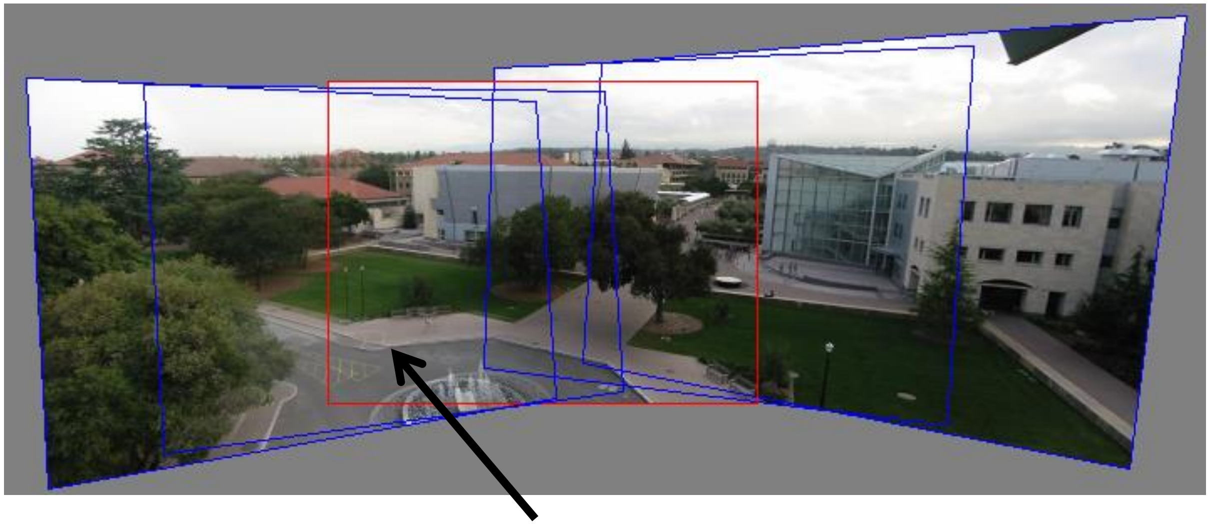
Idea: projecting images onto a common plane





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

We'll see what this homography means later.

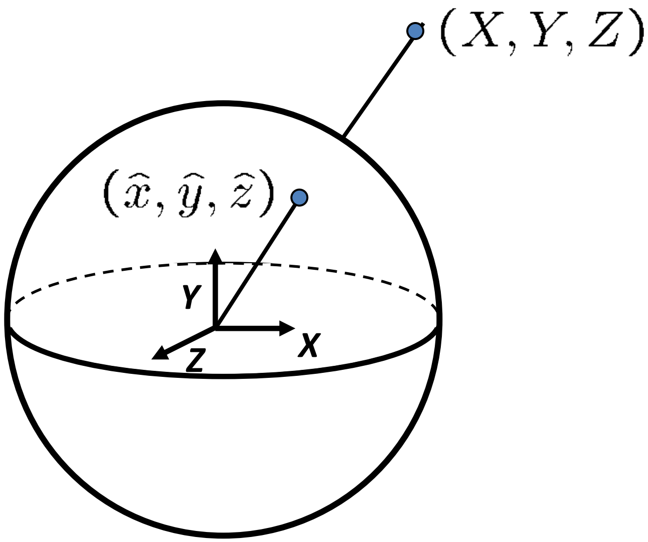
First -- Can't create a 360 panorama this way…

mosaic PP

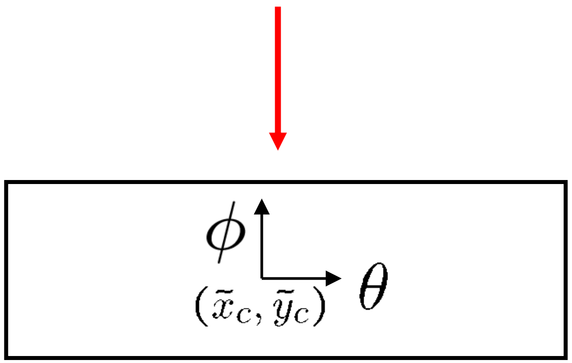
Project 3

- Take pictures on a tripod (or handheld)
- Warp to spherical coordinates (optional if using homographies to align images)
- Extract features
- Align neighboring pairs using RANSAC
- Write out list of neighboring translations
- Blend the images
- Correct for drift
- Now enjoy your masterpiece!

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

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

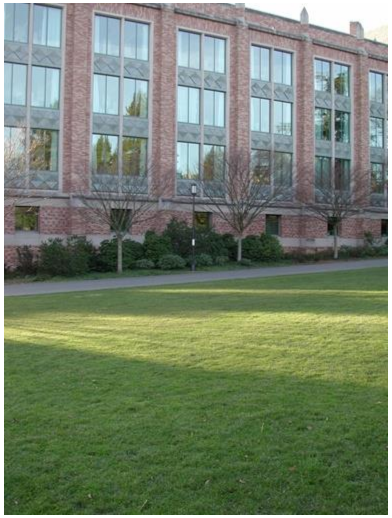
- Convert to spherical image coordinates

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

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

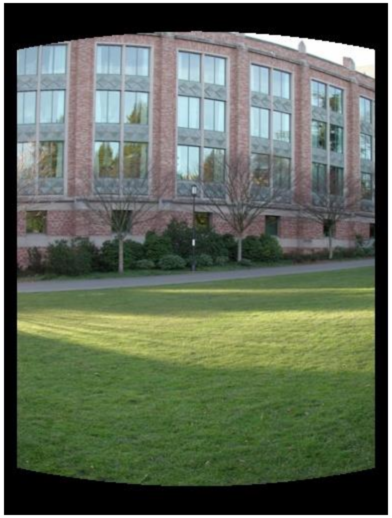
Spherical image

Spherical reprojection

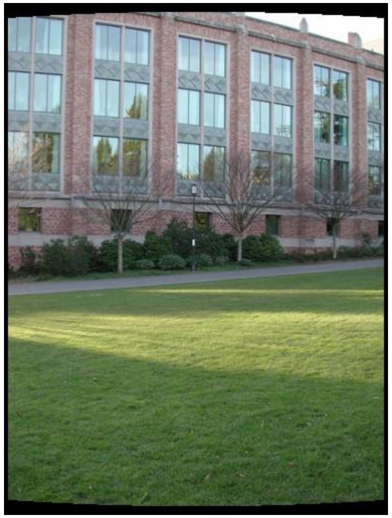


input

f = 200 (pixels)



f = 400



f = 800

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

Modeling distortion

Project $(\widehat{x}, \widehat{y}, \widehat{z})$ to "normalized" image coordinates

$$x'_n = \widehat{x}/\widehat{z}$$
$$y'_n = \widehat{y}/\widehat{z}$$

Apply radial distortion

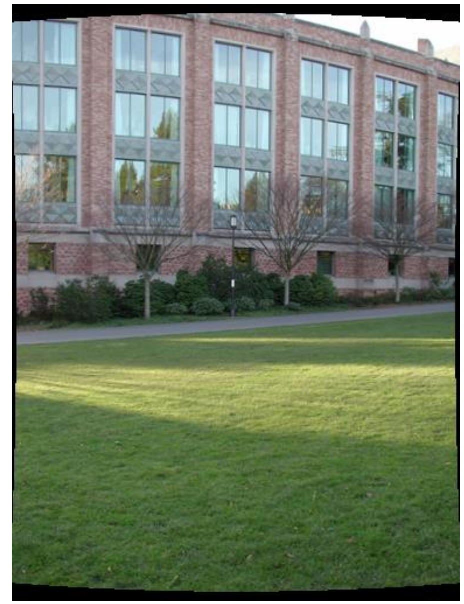
$$r^2 = {x'_n}^2 + {y'_n}^2$$
$$x'_d = x'_n(1 + \kappa_1 r^2 + \kappa_2 r^4)$$
$$y'_d = y'_n(1 + \kappa_1 r^2 + \kappa_2 r^4)$$

Apply focal length translate image center

$$x' = f x'_d + x_c$$
$$y' = f y'_d + y_c$$

- To model lens distortion with panoramas
  - Use above projection operation after projecting onto a sphere

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

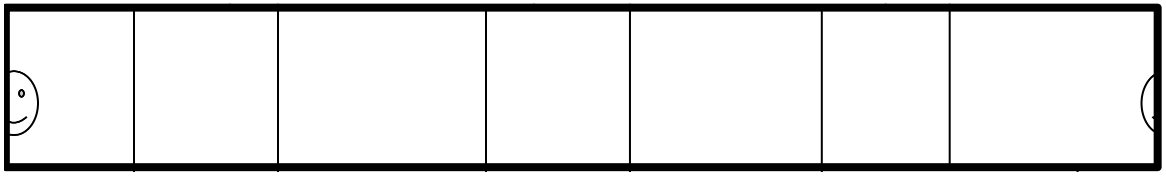
Solving for homographies

$$
\underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y'_1 \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_n x_n & -x'_n y_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_n x_n & -y'_n y_n & -y'_n \end{bmatrix}}_{\substack{\mathbf{A} \\ 2n \times 9}}
\underbrace{\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix}}_{\substack{\mathbf{h} \\ 9}}
=
\underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}}_{\substack{\mathbf{0} \\ 2n}}
$$

Defines a least squares problem: minimize $\|\mathbf{A}\mathbf{h} - \mathbf{0}\|^2$

- Since $\mathbf{h}$ is only defined up to scale, solve for unit vector $\hat{\mathbf{h}}$
- Solution: $\hat{\mathbf{h}}$ = eigenvector of $\mathbf{A}^T\mathbf{A}$ with smallest eigenvalue
- Works with 4 or more matches (8 rows in A). How do you find these points?

Assembling the panorama

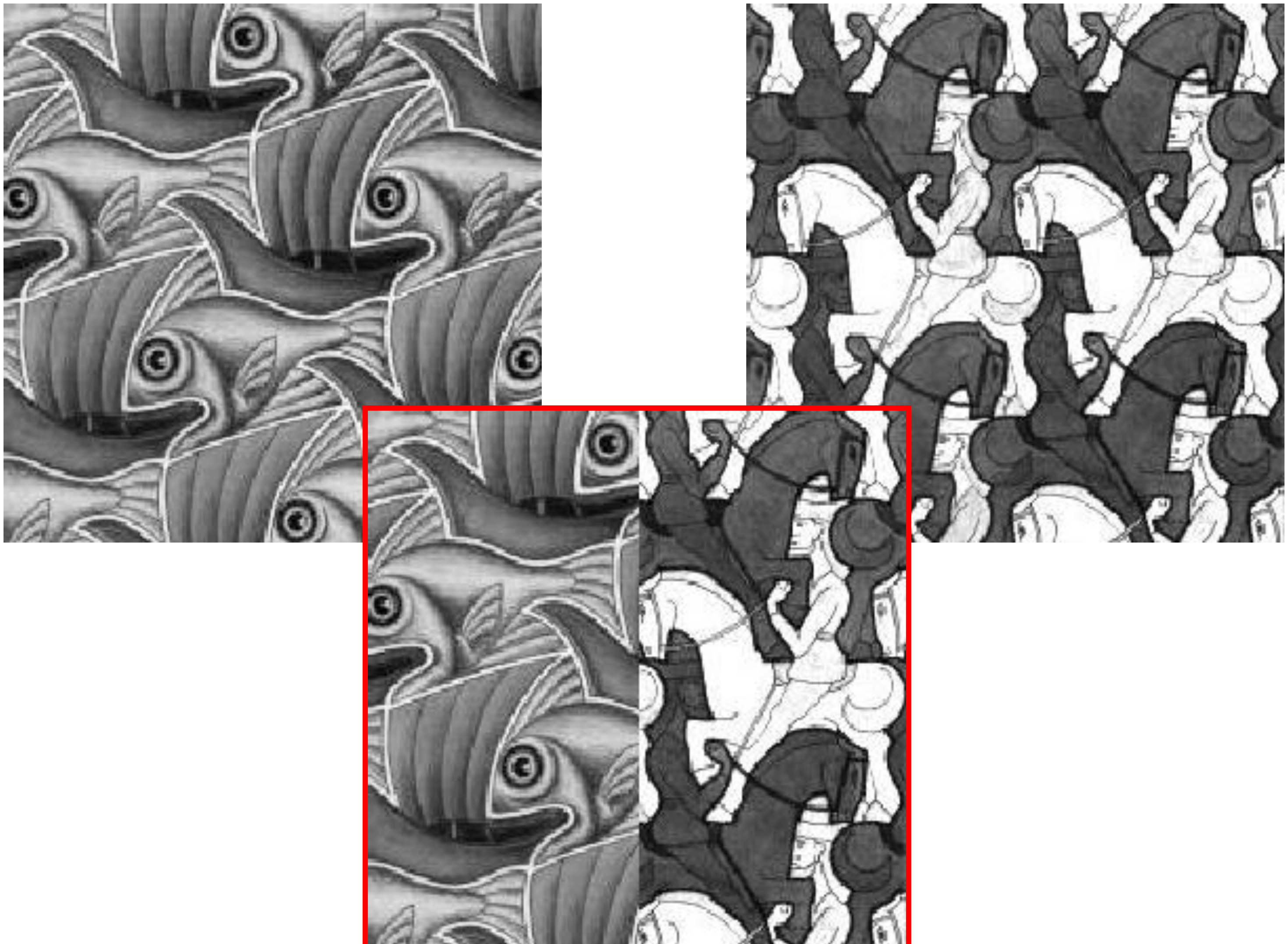


- Stitch pairs together, blend, then crop

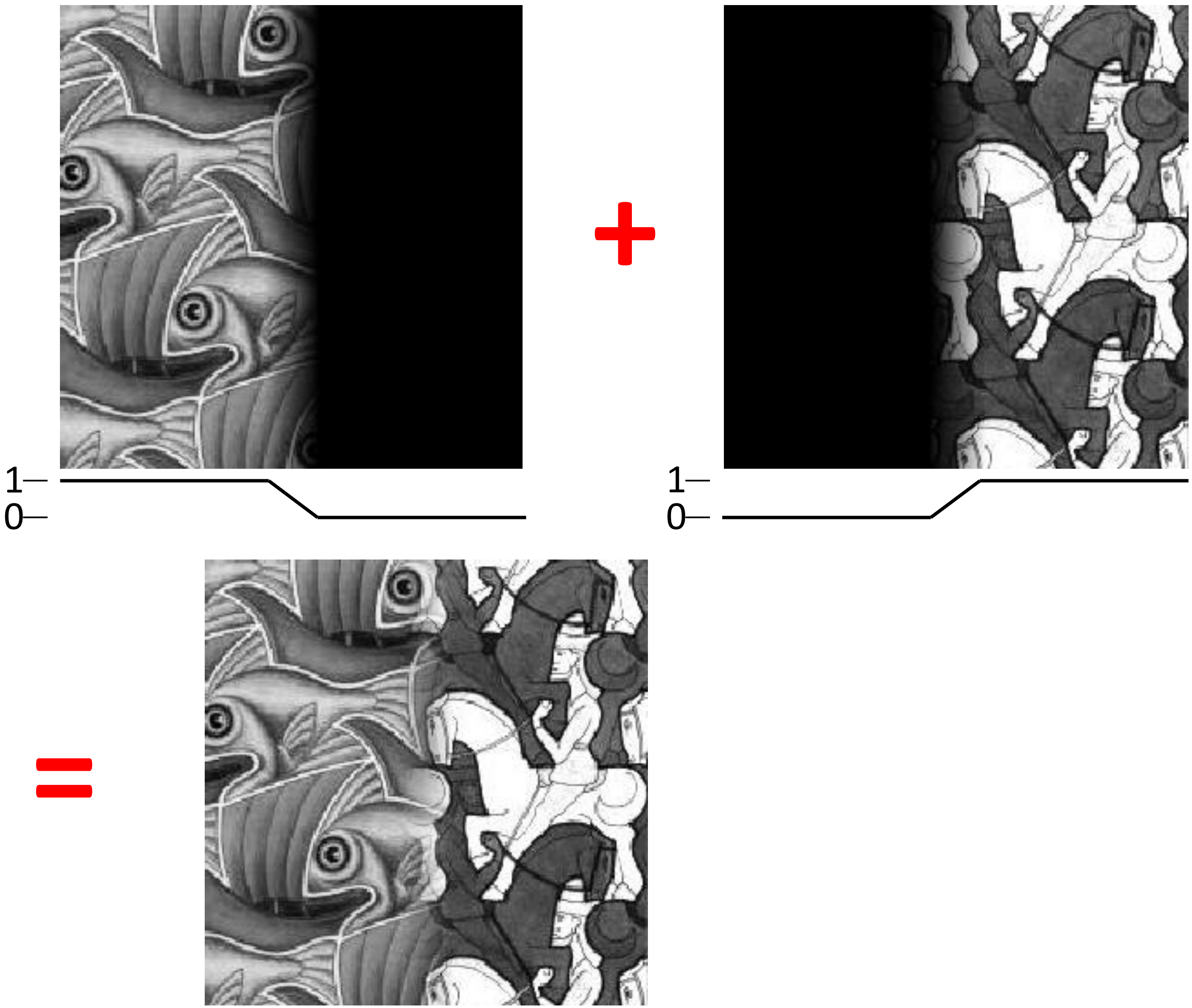
Blending

- We’ve aligned the images – now what?

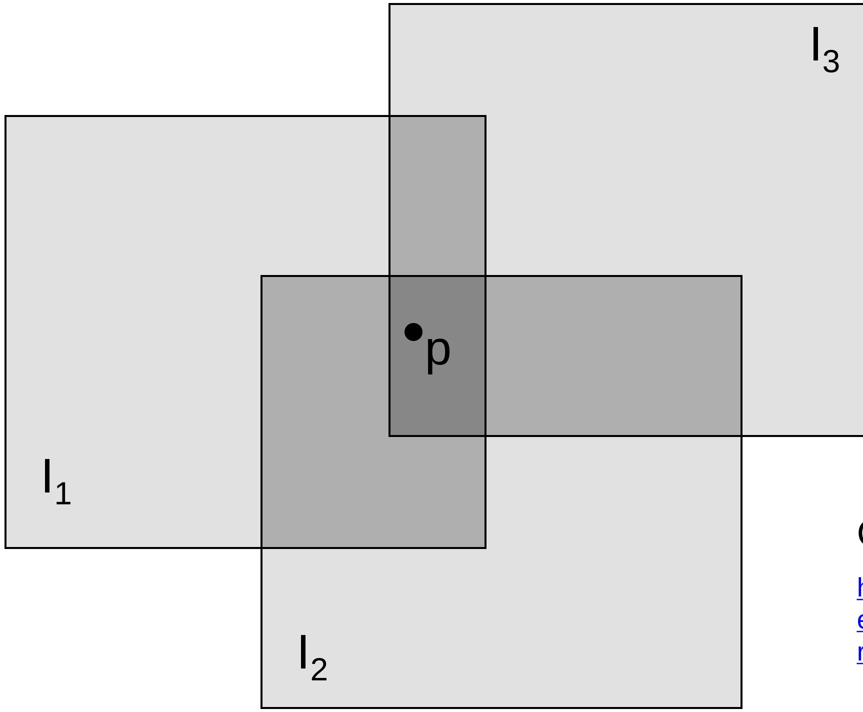
Image Blending



Feathering: Linear Interpolation



Alpha Blending



Optional: see Blinn (CGA, 1994) for details:

http://ieeexplore.ieee.org/iel1/38/7531/00310740.pdf?isNumber=7531&prod=JNL&arnumber=310740&arSt=83&ared=87&arAuthor=Blinn%2C+J.F.

Encoding blend weights: $I(x,y) = (\alpha R, \alpha G, \alpha B, \alpha)$

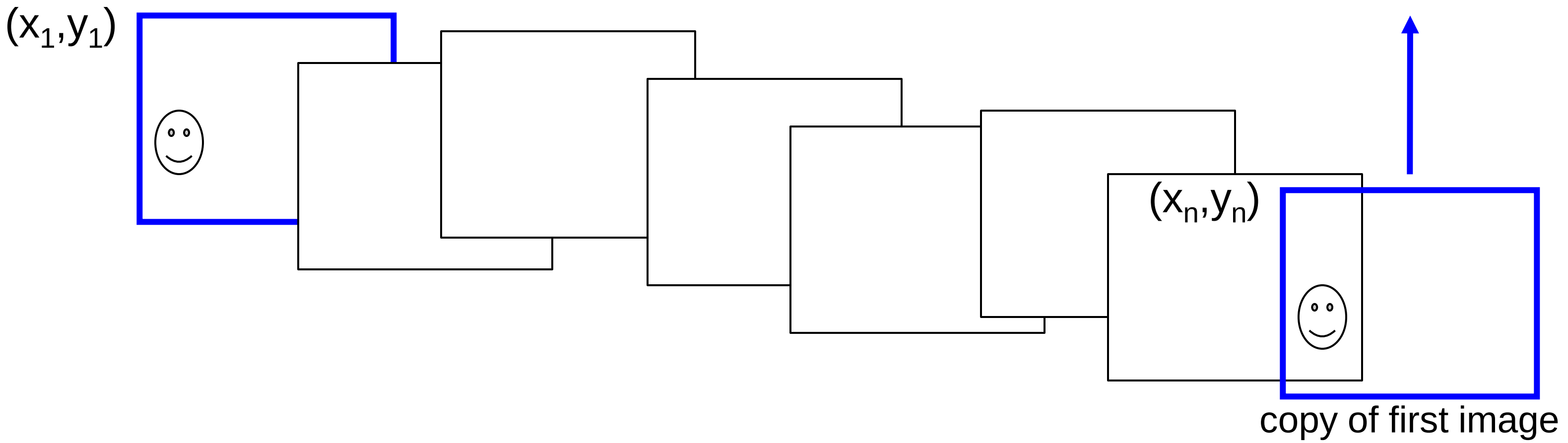
color at p = $$\frac{(\alpha_1 R_1,\ \alpha_1 G_1,\ \alpha_1 B_1) + (\alpha_2 R_2,\ \alpha_2 G_2,\ \alpha_2 B_2) + (\alpha_3 R_3,\ \alpha_3 G_3,\ \alpha_3 B_3)}{\alpha_1 + \alpha_2 + \alpha_3}$$

Implement this in two steps:

1. accumulate: add up the (α premultiplied) RGBα values at each pixel
2. normalize: divide each pixel's accumulated RGB by its α value

   Q: what if $\alpha = 0$?

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”

Demo