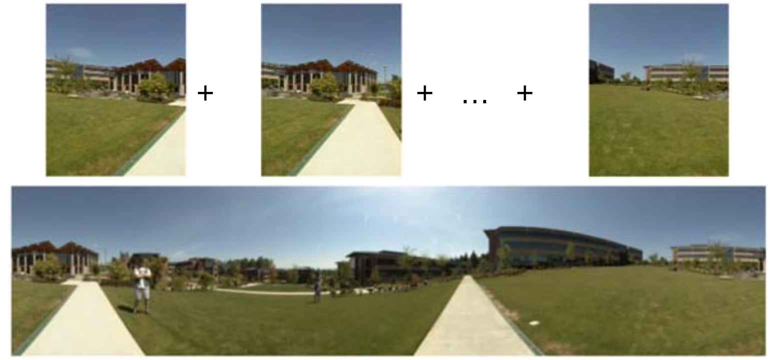
Mosaics

StreetView

Today's Readings

- Szeliski, Ch 5.1, 8.1

Image Mosaics



Goal

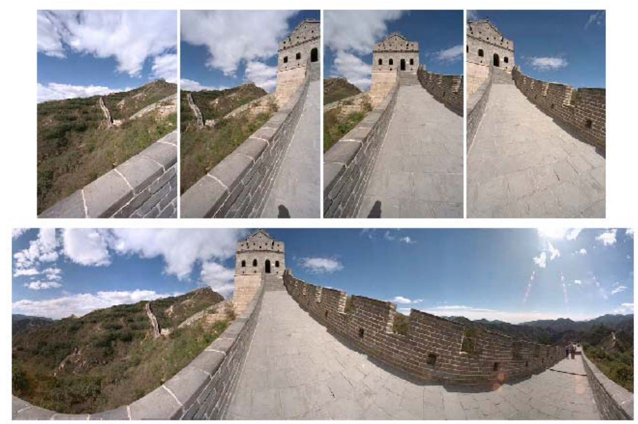
- Stitch together several images into a seamless composite

How to do it?

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
- Shift the second image to overlap with the first
- Blend the two together to create a mosaic
- If there are more images, repeat

Aligning images

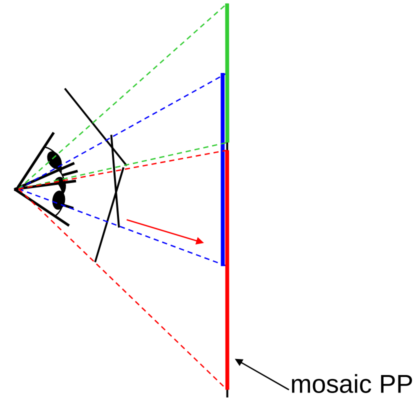


How to account for warping?

- Translations are not enough to align the images

Alignment Demo

Image reprojection



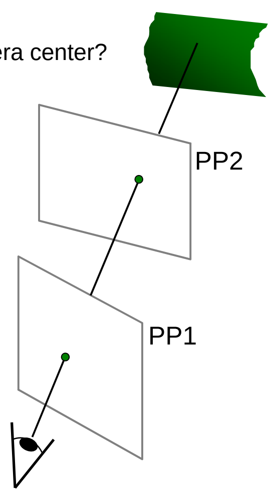
The mosaic has a natural interpretation in 3D

- The images are reprojected onto a common plane
- The mosaic is formed on this plane

Image reprojection

Basic question

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

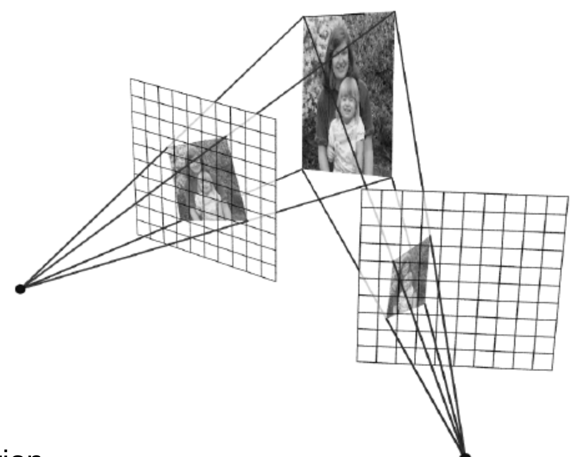


Answer

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

Don’t need to know what’s in the scene!

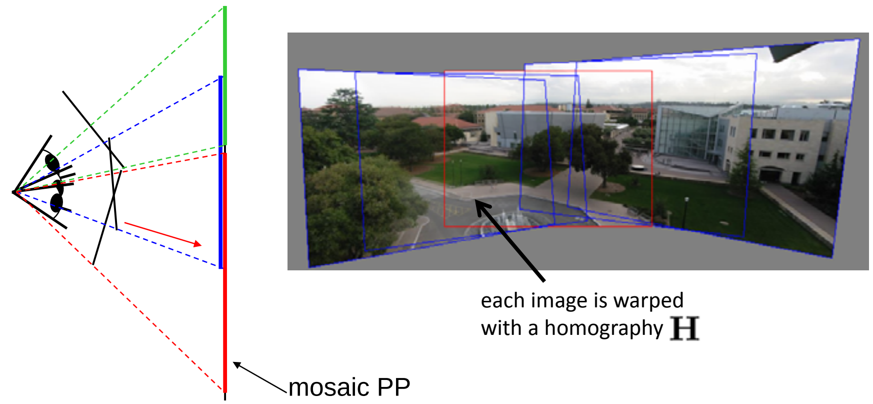
Image reprojection



Observation

- Rather than thinking of this as a 3D reprojection, think of it as a 2D image warp from one image to another

Homographies



Homographies

Perspective projection of a plane

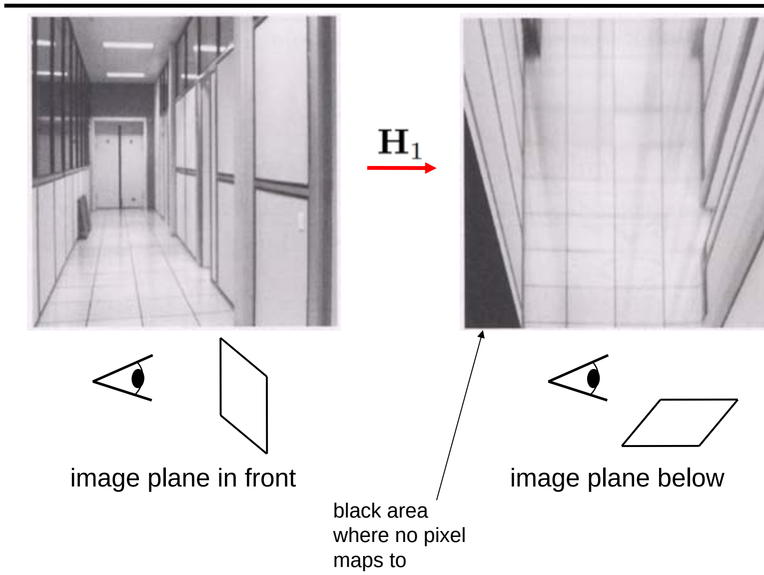
- Lots of names for this:
  - **homography**, texture-map, colineation, planar projective map
- Modeled as a 2D warp using homogeneous coordinates

$$\underset{\mathbf{p'}}{\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix}} = \underset{\mathbf{H}}{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}} \underset{\mathbf{p}}{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}$$

To apply a homography **H**

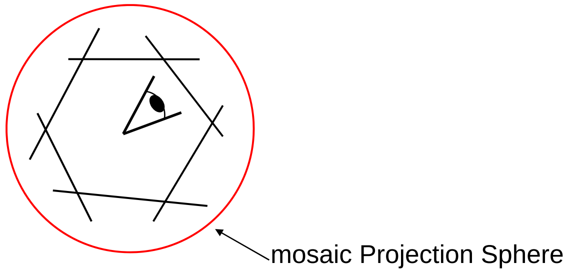
- Compute **p'** = **Hp** (regular matrix multiply)
- Convert **p'** from homogeneous to image coordinates
  - divide by w (third) coordinate

Image warping with homographies



Panoramas

What if you want a 360° field of view?

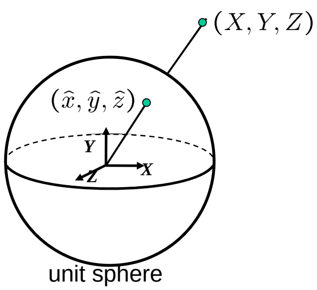


Spherical projection systems

Omnimax

CAVE (UI Chicago)

Spherical projection



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

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

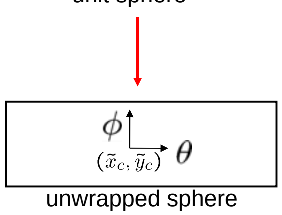
- Convert to spherical coordinates

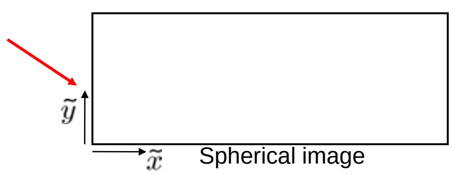
$$(sin\theta cos\phi, sin\phi, cos\theta cos\phi) = (\hat{x},\hat{y},\hat{z})$$

- Convert to spherical image coordinates

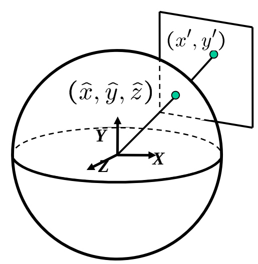
$$(\tilde{x},\tilde{y}) = (s\theta, s\phi) + (\tilde{x}_c,\tilde{y}_c)$$

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

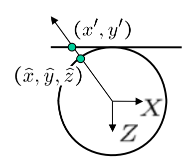




Spherical reprojection



side view

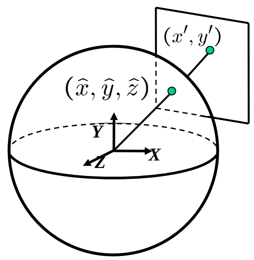


top-down view

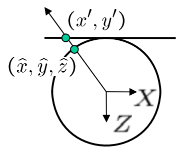
How to map sphere onto a flat image?

- $(\hat{x},\hat{y},\hat{z})$ to (x',y')

Spherical reprojection



side view



top-down view

How to map sphere onto a flat image?

- $(\hat{x}, \hat{y}, \hat{z})$ to (x', y')
  – Use image projection matrix!
  – or use the version of projection that properly accounts for radial distortion, as discussed in projection slides. This is what you'll do for project 2.

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 we can align spherical images by translating them

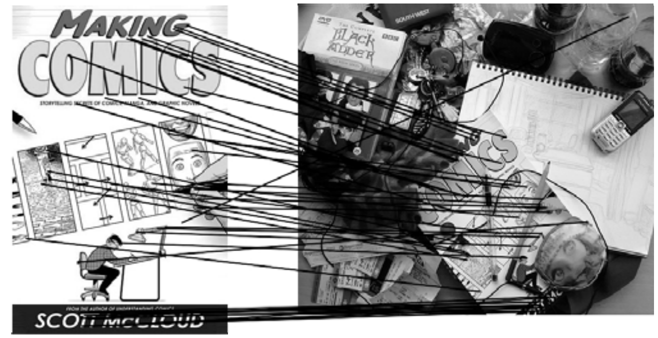
Spherical image stitching

What if you don't know the camera rotation?

- Solve for the camera rotations
  - Note that a pan (rotation) of the camera is a **translation** of the sphere!
  - **Use feature matching to solve for translations of spherical-warped images**

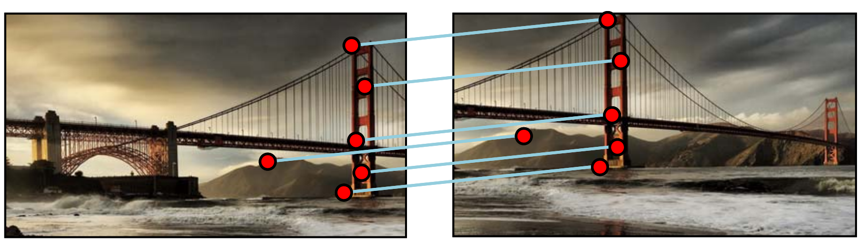
Computing transformations

- Given a set of matches between images A and B
  - How can we compute the transform T from A to B?



  - Find transform T that best "agrees" with the matches

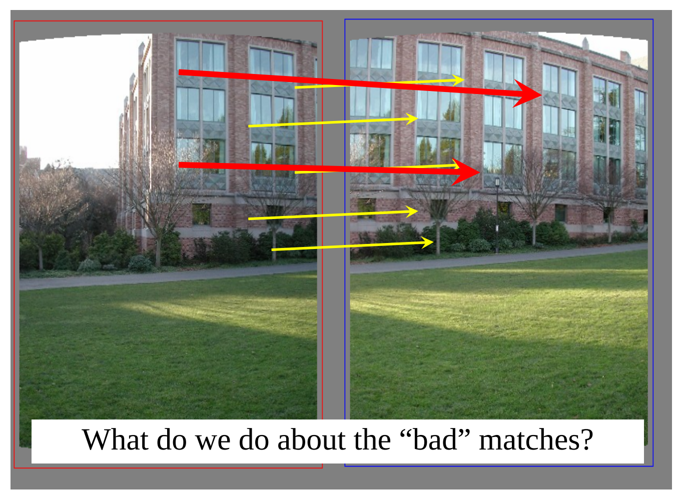
Simple case: translations



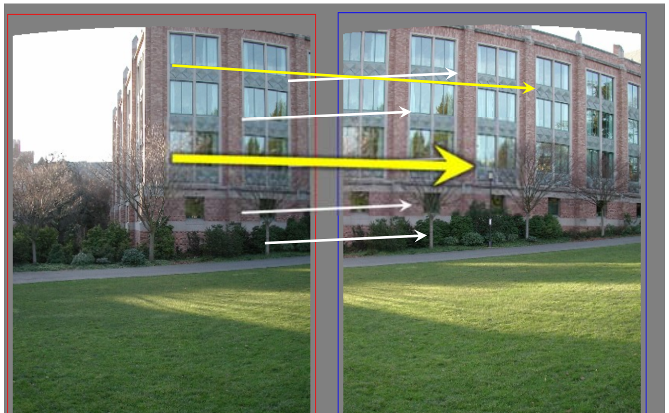
$(\mathbf{x}_t, \mathbf{y}_t)$

How do we solve for $(\mathbf{x}_t, \mathbf{y}_t)$**?**

But not all matches are good

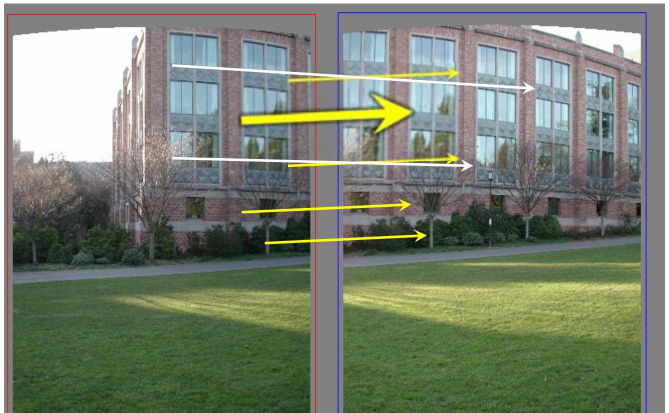


RAndom SAmple Consensus



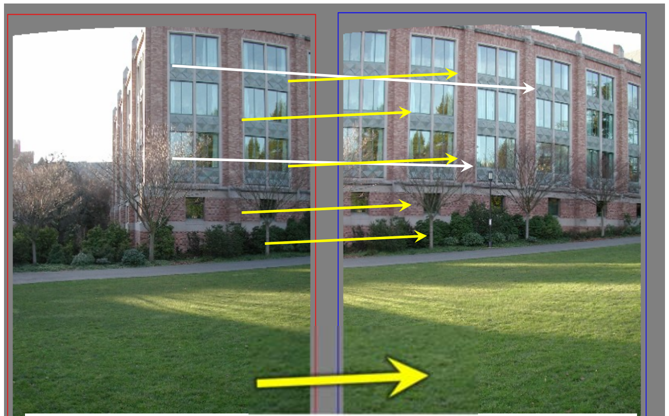
Select one match, count inliers
(in this case, only one)

RAndom SAmple Consensus



Select one match, count inliers
(4 inliers)

Least squares fit



Find "average" translation vector
for largest set of inliers

RANSAC

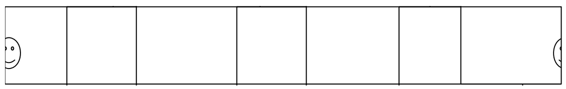
Same basic approach works for any transformation

- Translation, rotation, homographies, etc.
- Very useful tool

General version

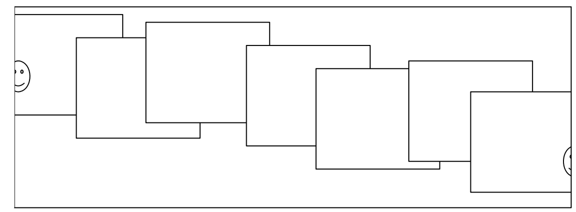
- Randomly choose a set of K correspondences
  - Typically K is the minimum size that lets you fit a model
- Fit a model (e.g., homography) to those correspondences
- Count the number of inliers that "approximately" fit the model
  - Need a threshold on the error
- Repeat as many times as you can
- Choose the model that has the largest set of inliers
- Refine the model by doing a least squares fit using ALL of the inliers

Assembling the panorama



Stitch pairs together, blend, then crop

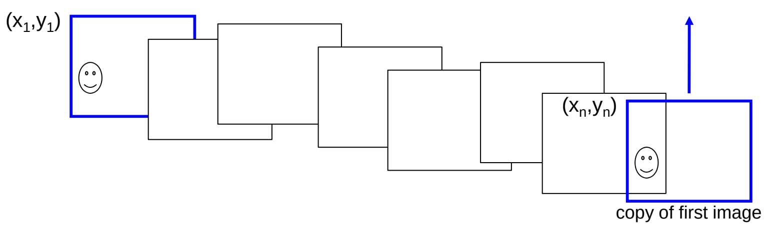
Problem: Drift



Error accumulation

- small errors accumulate over time

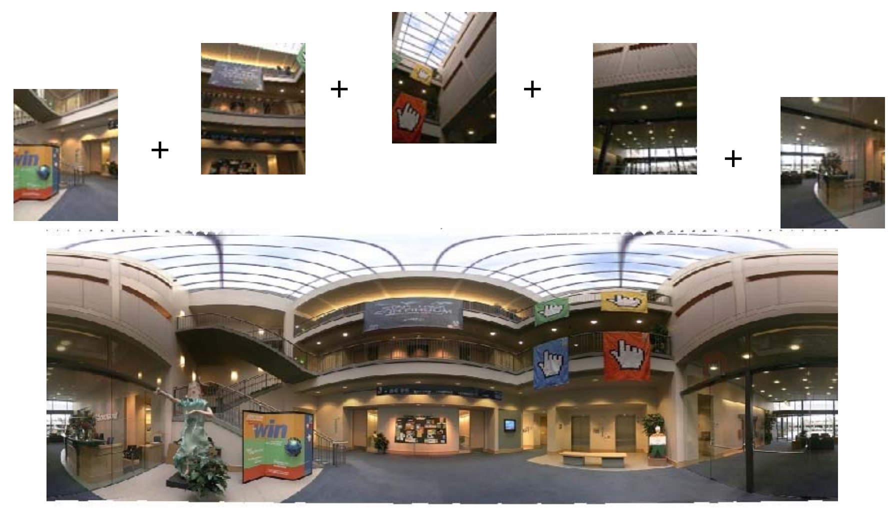
Problem: Drift



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
  - compute a global warp: $y' = y + ax$
  - run a big optimization problem, incorporating this constraint
    - » best solution, but more complicated
    - » known as “bundle adjustment”

Full-view Panorama



Different projections are possible

Project 2

Take pictures with your phone (or on a tripod)

Warp to spherical coordinates

Extract features

Align neighboring pairs using RANSAC

Write out list of neighboring translations

Correct for drift

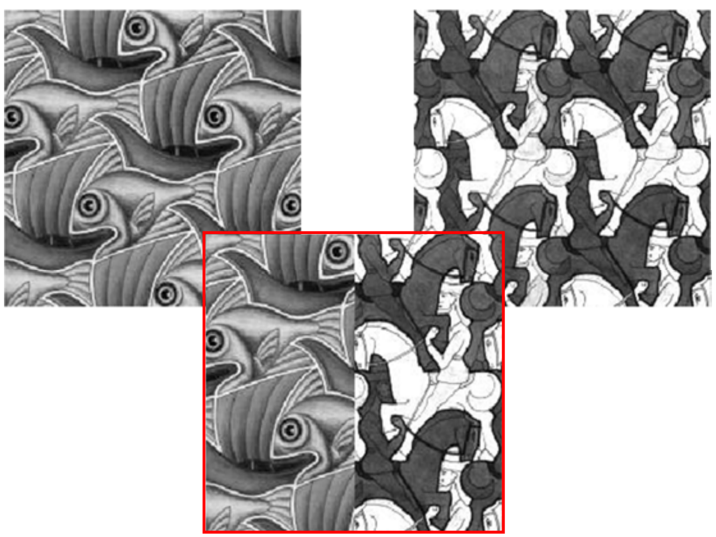
Read in warped images and blend them

Crop the result and import into a viewer

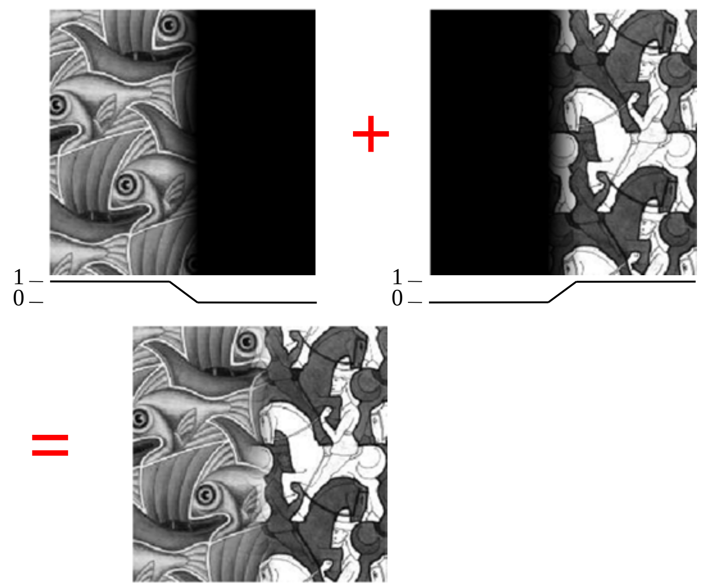
Roughly based on **Autostitch**

- By Matthew Brown and David Lowe
- http://www.cs.ubc.ca/~mbrown/autostitch/autostitch.html

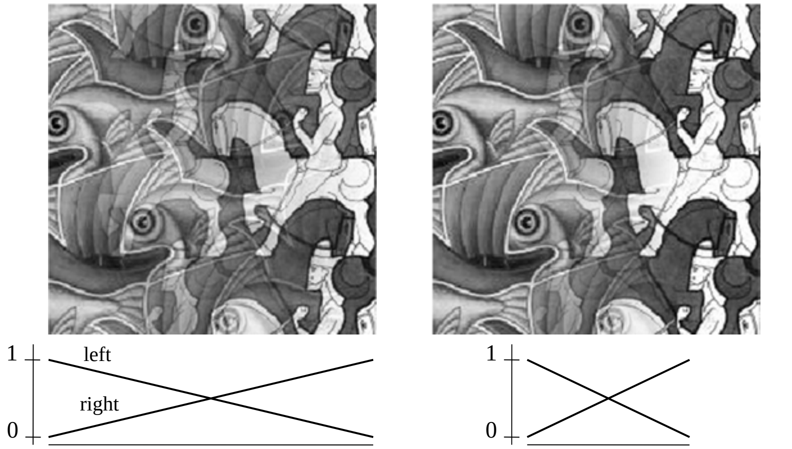
Image Blending



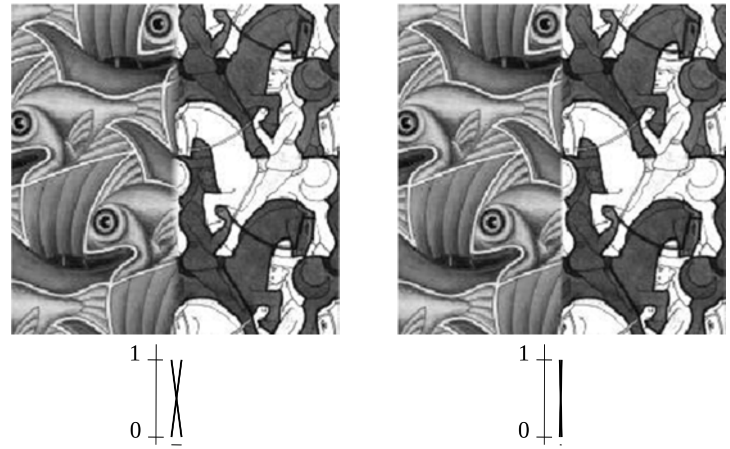
Feathering



Effect of window (ramp-width) size



Effect of window size

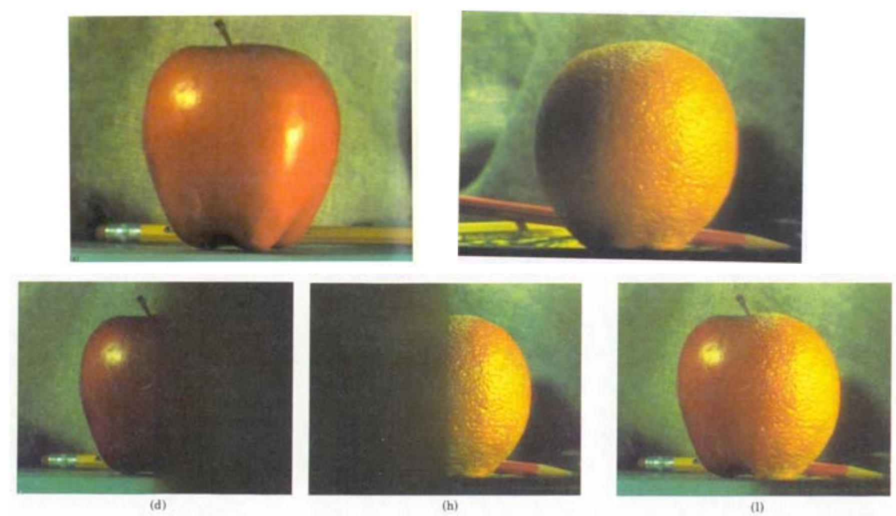


Good window size

"Optimal" window: smooth but not ghosted

- Doesn't always work...

Pyramid blending



Create a Laplacian pyramid, blend each level

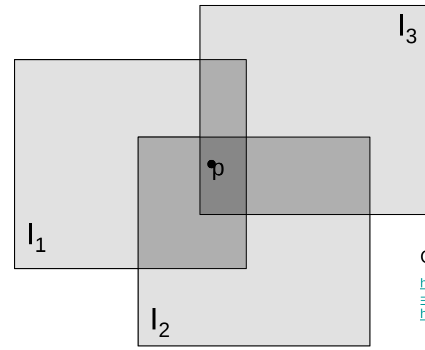
- Burt, P. J. and Adelson, E. H., A multiresolution spline with applications to image mosaics, ACM Transactions on Graphics, 42(4), October 1983, 217-236.

Poisson Image Editing

For more info: Perez et al, SIGGRAPH 2003

- http://research.microsoft.com/vision/cambridge/papers/perez_siggraph03.pdf

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

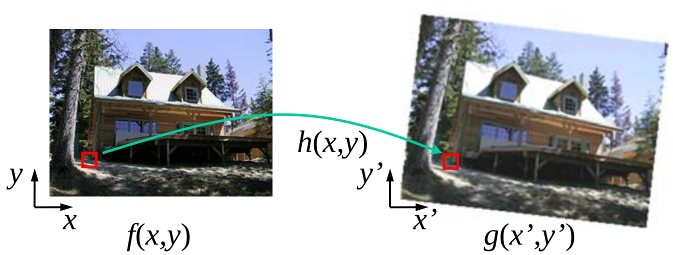
color at p = $\dfrac{(\alpha_1 R_1, \alpha_1 G_1, \alpha_1 B_1, \alpha_1) + (\alpha_2 R_2, \alpha_2 G_2, \alpha_2 B_2, \alpha_2) + (\quad)}{\alpha_1 + \alpha_2 + \alpha_3}$

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

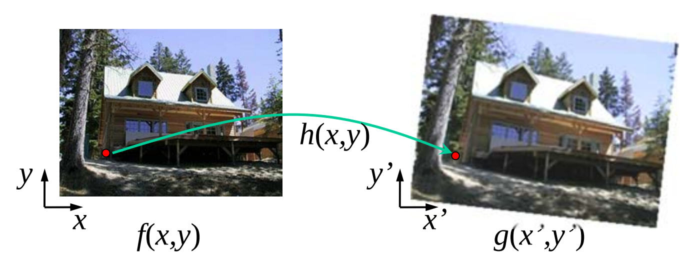
Q: what if $\alpha = 0$?

Image warping



Given a coordinate transform $(x',y') = h(x,y)$ and a source image $f(x,y)$, how do we compute a transformed image $g(x',y') = f(h(x,y))$?

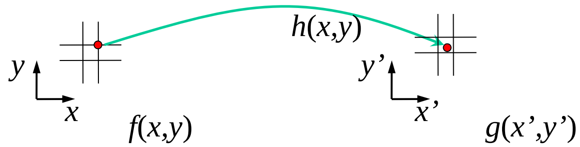
Forward warping



Send each pixel $f(x,y)$ to its corresponding location $(x',y') = h(x,y)$ in the second image

Q: what if pixel lands "between" two pixels?

Forward warping



Send each pixel *f*(*x,y*) to its corresponding location

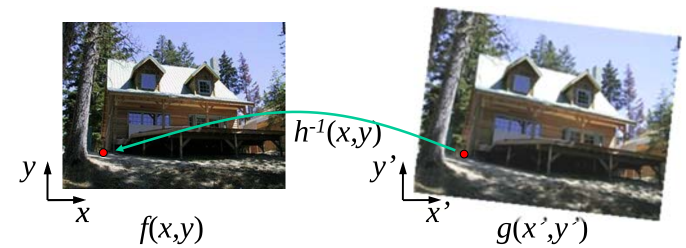
(*x',y'*) = *h*(*x,y*) in the second image

Q: what if pixel lands "between" two pixels?

A: distribute color among neighboring pixels (x',y')

– Known as "splatting"

Inverse warping

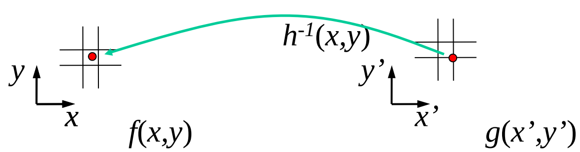


Get each pixel *g*(*x',y'*) from its corresponding location

(*x,y*) = h^{-1}(*x',y'*) in the first image

Q: what if pixel comes from "between" two pixels?

Inverse warping



Get each pixel *g*(*x',y'*) from its corresponding location

(*x,y*) = h^{-1}(*x',y'*) in the first image

Q: what if pixel comes from "between" two pixels?

A: *resample* color value

– We discussed resampling techniques before

• nearest neighbor, bilinear, Gaussian, bicubic

Forward vs. inverse warping

Q: which is better?

A: usually inverse—eliminates holes

- however, it requires an invertible warp function—not always possible...

Other types of mosaics

Can mosaic onto *any* surface if you know the geometry

- See NASA's Visible Earth project for some stunning earth mosaics
  - http://earthobservatory.nasa.gov/Newsroom/BlueMarble/
  - Click for images…