ECE5554 – Computer Vision
Lecture 9b – Image Stitching

Creed Jones, PhD

Today’s Objectives

Image Stitching

- Geometric transformation models
  - Homographic transformation
  - Rotational model
- Correspondences and outliers
- Gaps and blending
- A walkthrough
- Spherical Projection

Image Stitching takes a set of separate images of a scene and combines them into a larger, panoramic image

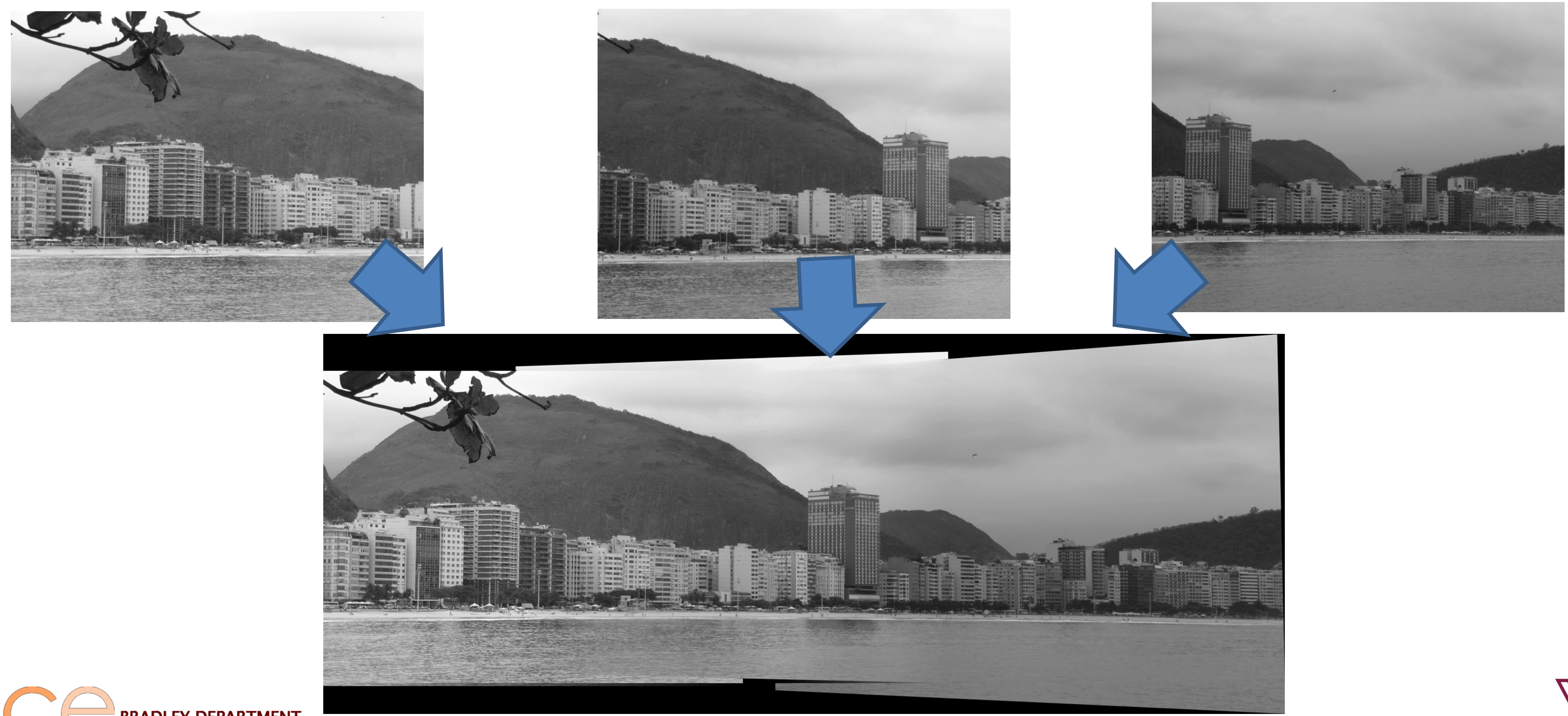


Image stitching (to form a *mosaic*) is not simply pasting images together; some geometric transformation is needed

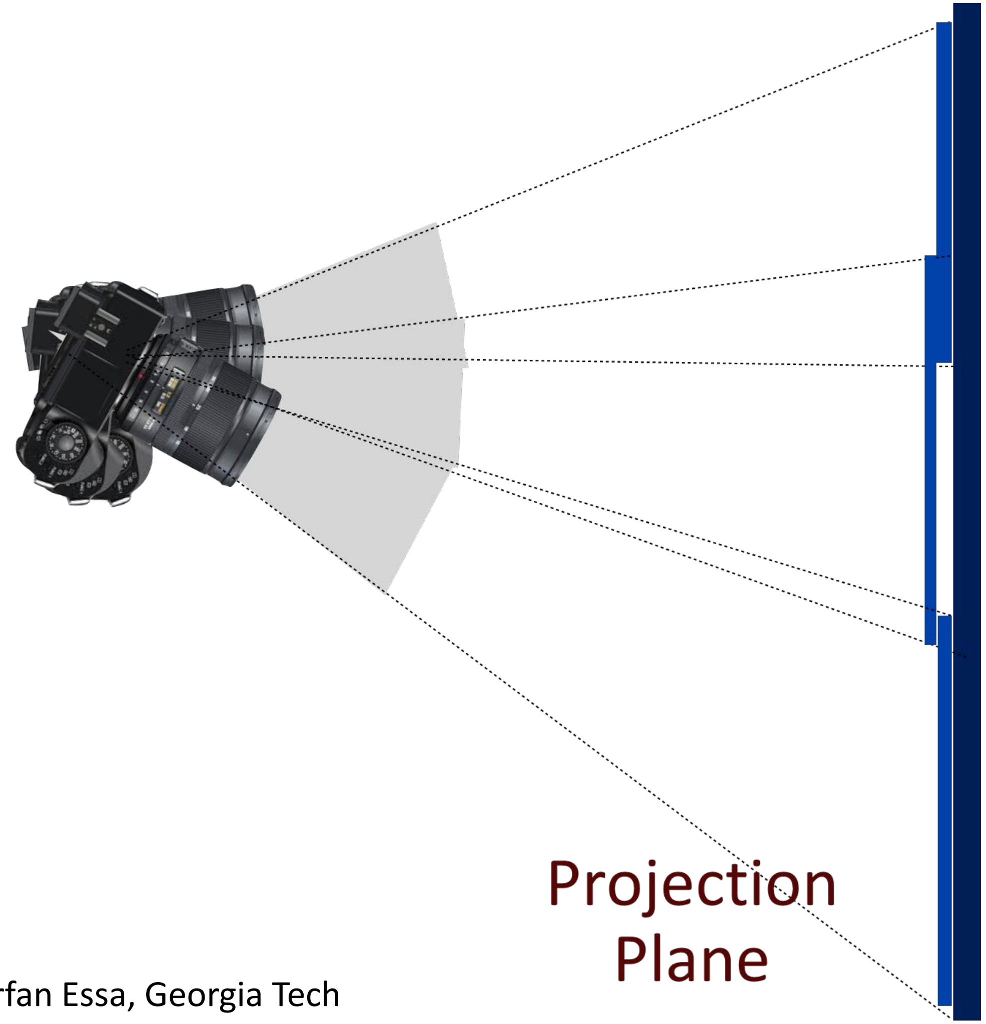


Image Re-projection to Panorama Projection Plane

- The panorama mosaic has a natural interpretation in 3D
- Images are reprojected onto a common plane
- The mosaic is formed on this plane
- Mosaic is a synthetic wide-angle camera

Thanks to Irfan Essa, Georgia Tech

What model should we assume for the geometric transformation between the two images? Is an affine transformation sufficient?

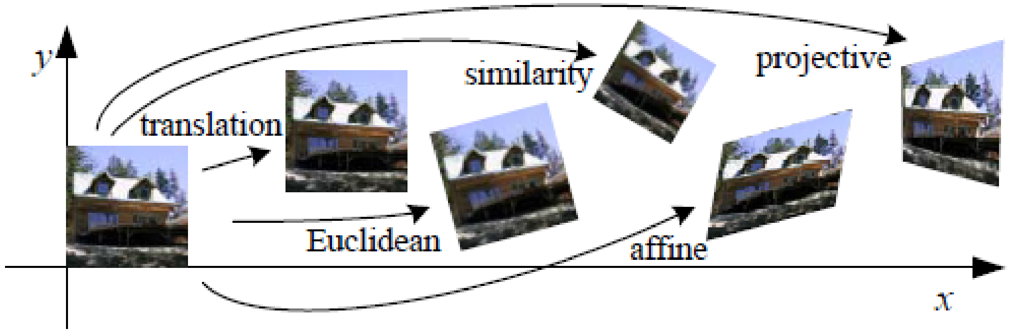


Figure 3.45 Basic set of 2D geometric image transformations.

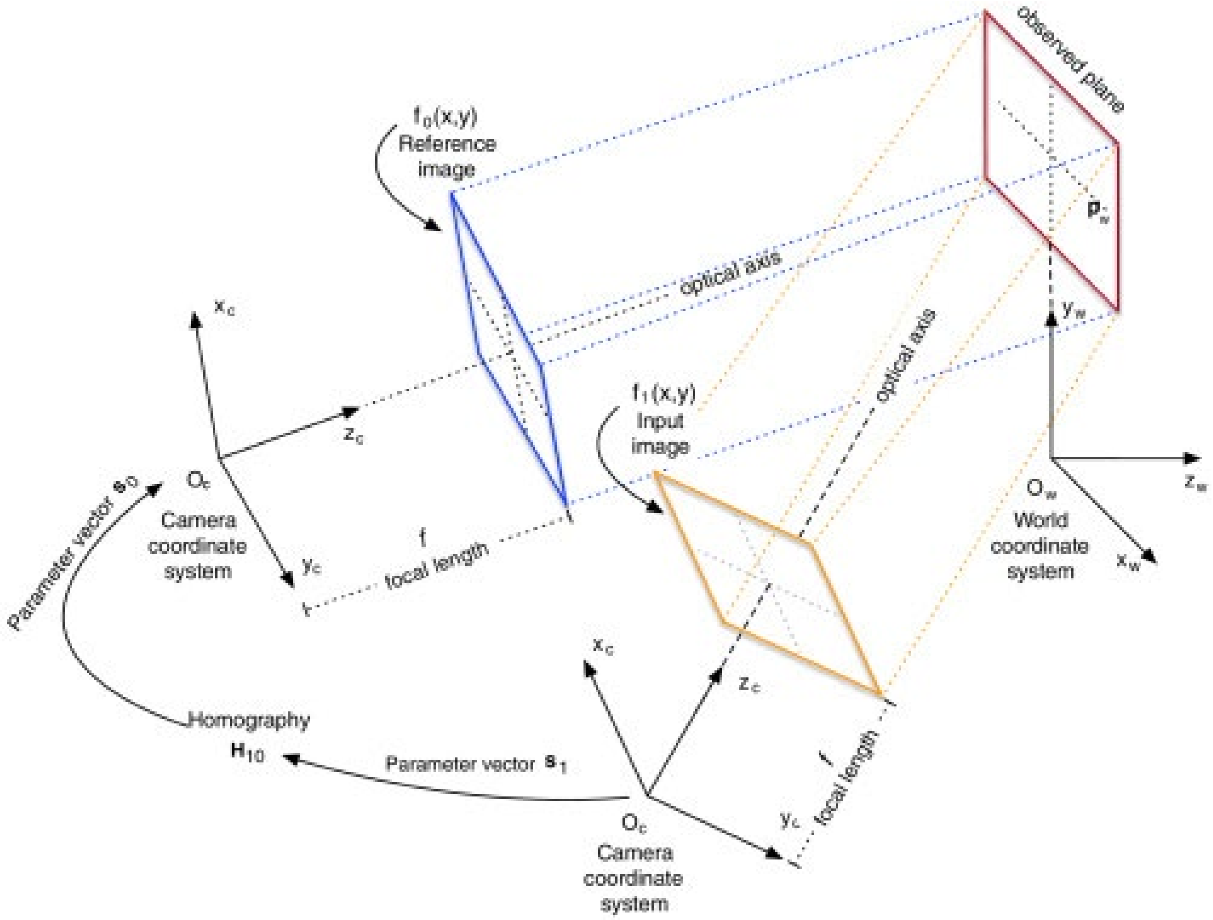
Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} I \mid t \end{bmatrix}_{2\times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} R \mid t \end{bmatrix}_{2\times 3}$	3	lengths	
similarity	$\begin{bmatrix} sR \mid t \end{bmatrix}_{2\times 3}$	4	angles	
affine	$\begin{bmatrix} A \end{bmatrix}_{2\times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{H} \end{bmatrix}_{3\times 3}$	8	straight lines	

We often use a 4-point *homography*, to model the deformation of a planar surface (or nearly flat scene) viewed from different positions

- Earlier, we called this a projective transformation

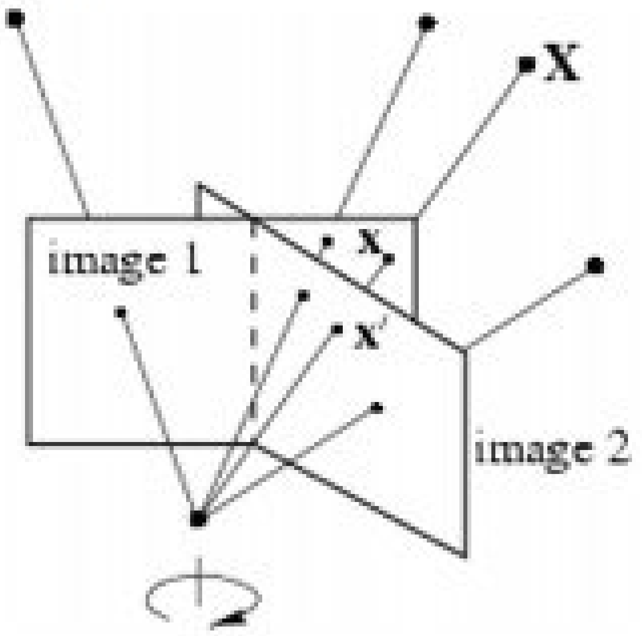
- If the scene is nearly planar as compared to the distance from the camera(s), then a homographic transformation is used
- To transform the input image into the coordinate system of the reference image,

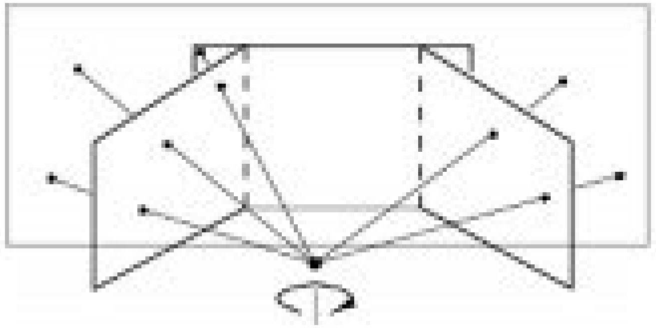
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \boldsymbol{H} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



We can use a rotational model if the scene is not necessarily planar but the camera only rotates (doesn't translate)

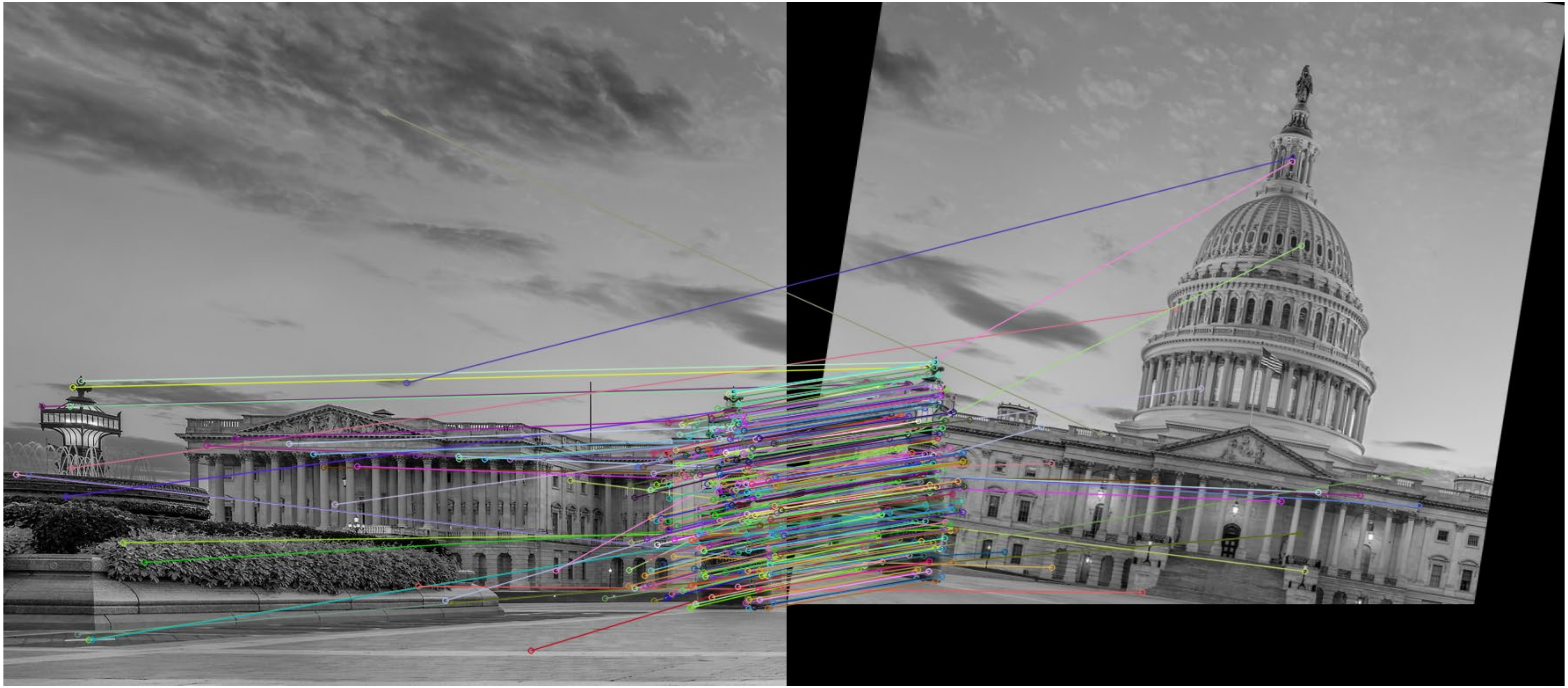
Rotating camera, arbitrary world





Consider two images of the same (nearly planar) scene – with camera movement (displacement and rotation); we use a homographic mapping to relate the two

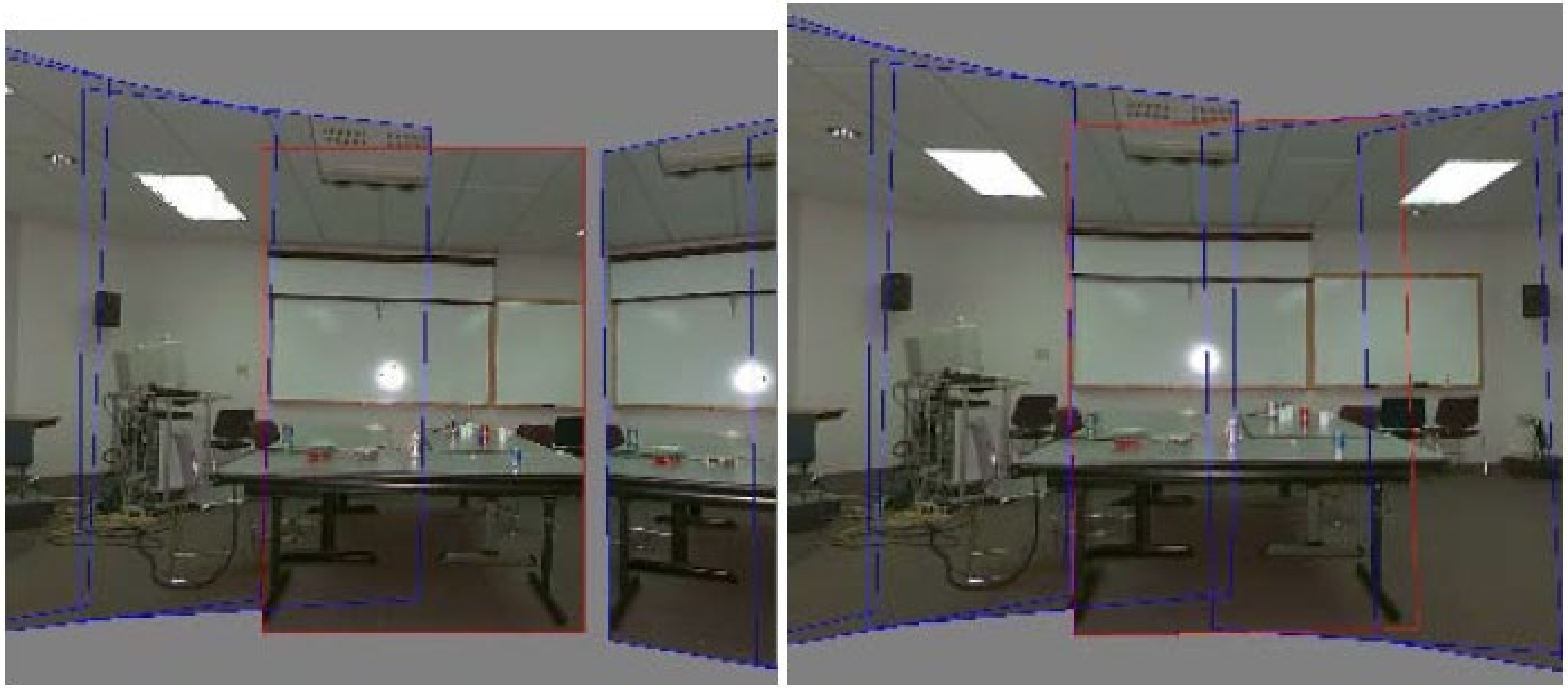
Find corresponding points in the two images (in this case I used SIFT); note the false correspondences!



Calculate the proper perspective projection – the homographic translation matrix to put image 2 into image 1’s coordinate system (false correspondences are *outliers* and are removed using RANSAC)

I have just laid one image (properly registered) on top of the other, but there will usually be gaps

Gap closing can be simple in the case of small camera movement <u>and friendly optics</u>, but lens distortions can make this more complex



(a) (b)

Figure 9.5 Gap closing (Szeliski and Shum 1997) © 1997 ACM: (a) A gap is visible when the focal length is wrong ($f = 510$). (b) No gap is visible for the correct focal length ($f = 468$).

Rather than overlaying images, we combine the pixels in overlapping regions – but there are difficulties

- Do we just average the pixels in the overlap?
  - Geometric distortions
  - Difference in exposure
- How do we find the overlap?
  - I often use gray level 0 as the “no data” value, so I only need to blend in areas where both images are nonzero
  - Of course, you must avoid any actual image pixels at gray level zero
- What about image size?
  - I usually increase size of images dramatically and trim when combining is done

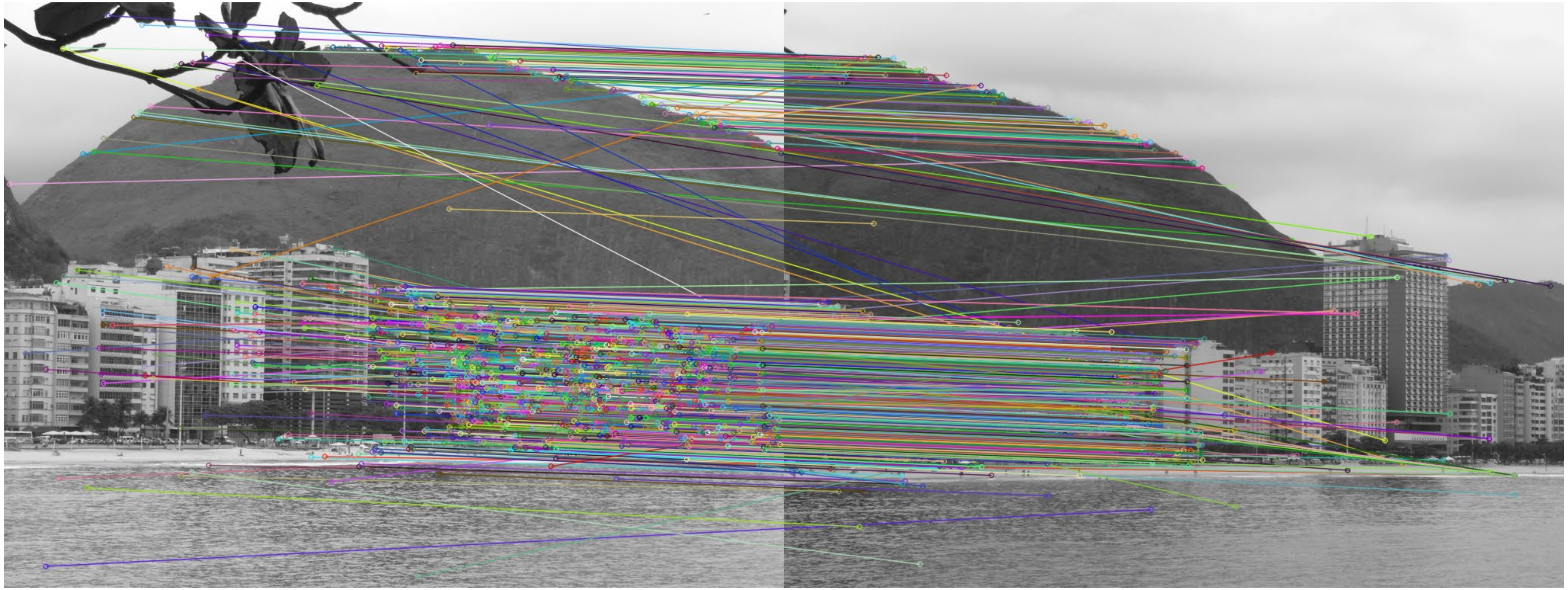
A walkthrough of a simple image stitching flow using OpenCV

1. Detect keypoints in the image and characterize them
    1. use cv2.SIFT_create() and sift.detectAndCompute(), or other methods
2. Match the keypoints in the two images
    1. use cv2.BFMatcher() and bf.knnMatch(), or other methods
3. Sort the matches to keep only the best one (most distinct, least confused)
4. Find the geometric mapping from image 1 to image 2 from the matches
    1. use cv2.estimateAffine2D() if the only allowed transformation is affine
    2. for perspective transformation, use cv2.findHomograpy()
5. Transform the second image to lie in the coordinates of the first
    1. use cv2.warpAffine() or cv2.warpPerspective()
6. Combine the images
7. Blend the images

Here are three images of the Rio de Janeiro waterfront, taken from a moving camera (the scene is far away, so virtually planar)

- I will use a homographic mapping

Find the correspondences between the first two images and use these to calculate the homography matrix H



The second image is transformed to be in the coordinate system of the first image

These points correspond!

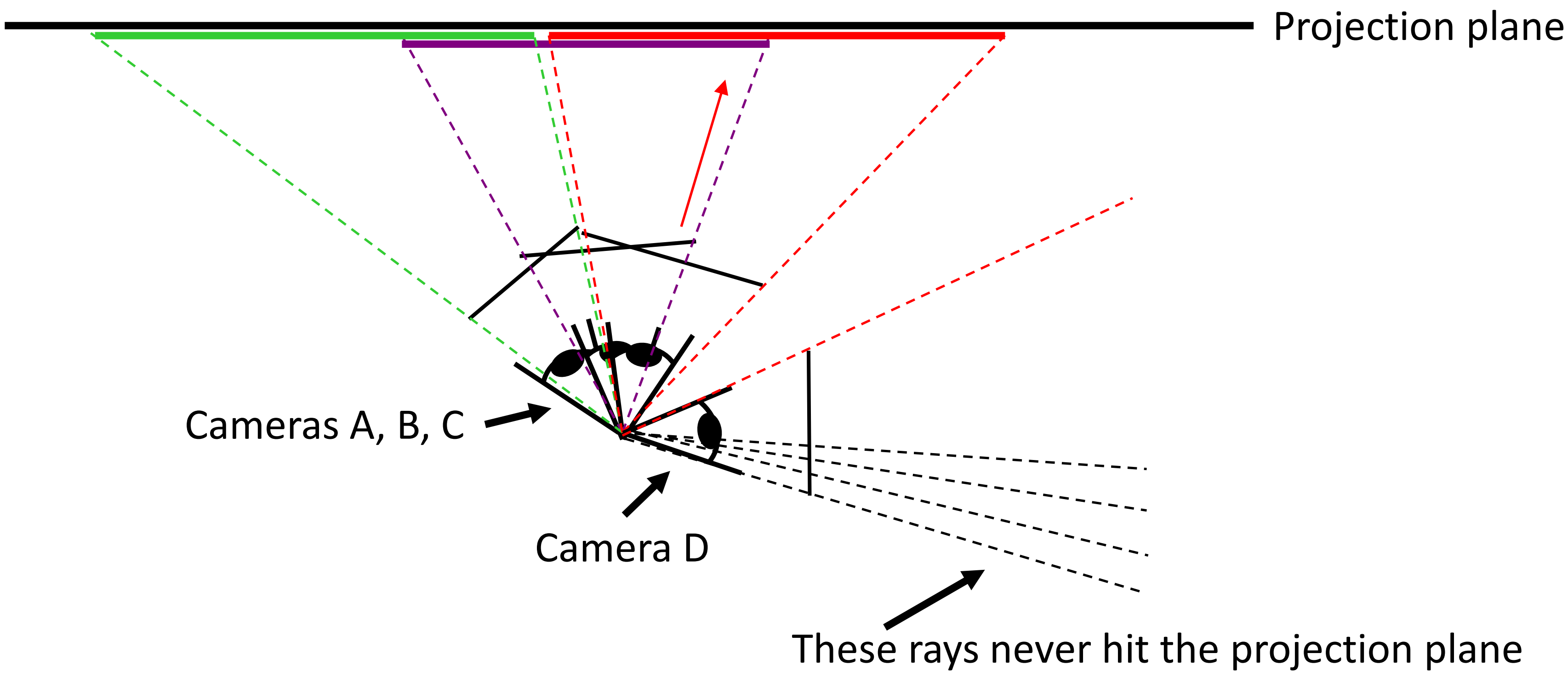
The same operation is used to combine image 3 with the result of combining images 1 and 2 – here, overlapping pixels are averaged

Feathering is a method of blending; in overlapping regions, varying weights for the separate images are calculated based on the distance from the edge of each

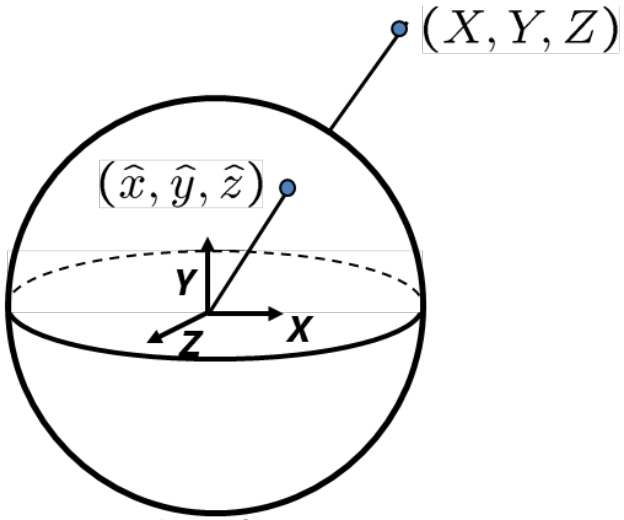
- We can find the distance of each pixel from the edge of its valid region and calculate weights appropriately
  - the distance transform can be used to find out how “far inside” the regions pixels are
- More sophisticated methods determine weightings from image contents
  - Blending and feathering inevitably create “seams”
  - Why don’t we put these seams at convenient spots in the scene?

I calculated variable pixel weights for the overlap regions – the result is pretty good!

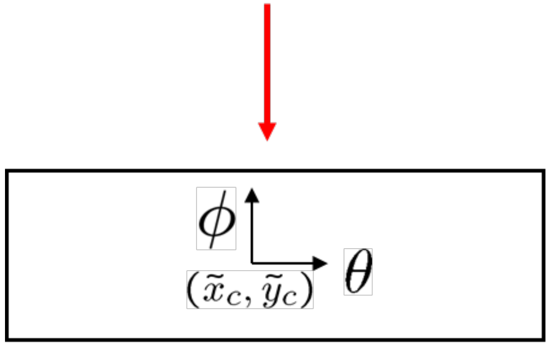
We have to use other geometric models to create wider results – such as a 360° panorama



Spherical Projection models image relationships for images taken at all angles, from a single point



unit sphere



unwrapped sphere

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

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

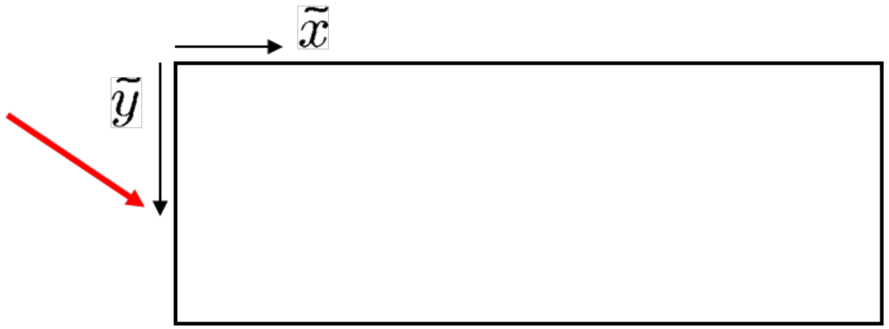
- Convert to spherical coordinates

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

- 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



Spherical image

Spherical Reprojection

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

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

Today's Objectives

Image Stitching

- Geometric transformation models
  - Homographic transformation
  - Rotational model
- Correspondences and outliers
- Gaps and blending
- A walkthrough
- Spherical Projection