Panoramas and Calibration

15-463: Rendering and Image Processing
Alexei Efros

…with a lot of slides stolen from Steve Seitz and Rick Szeliski

Why Mosaic?

Are you getting the whole picture?

- Compact Camera FOV = 50 x 35°

Slide from Brown & Lowe

Why Mosaic?

Are you getting the whole picture?

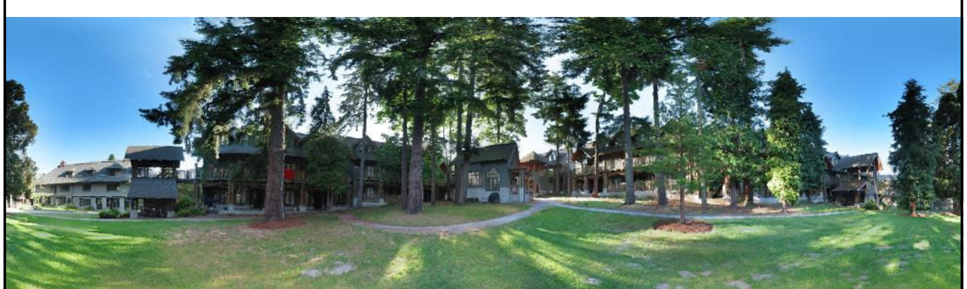
- Compact Camera FOV = 50 x 35°
- Human FOV = 200 x 135°

Slide from Brown & Lowe

Why Mosaic?

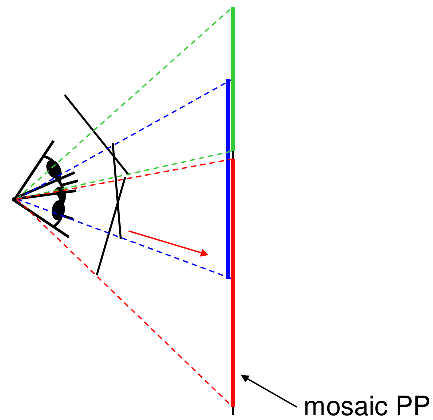
Are you getting the whole picture?

- Compact Camera FOV = 50 x 35°
- Human FOV = 200 x 135°
- Panoramic Mosaic = 360 x 180°



Slide from Brown & Lowe

Mosaic as Image Reprojection

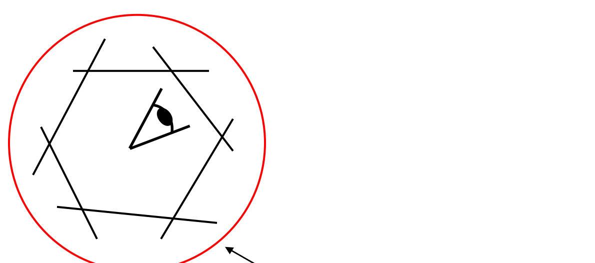


The mosaic has a natural interpretation in 3D

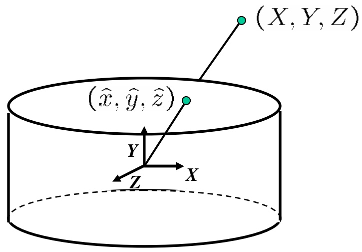
- The images are reprojected onto a common plane
- The mosaic is formed on this plane
- Mosaic is a *synthetic wide-angle camera*
- *Max FOV?*

Panoramas

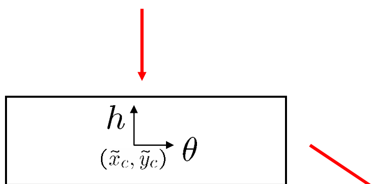
What if you want a 360° field of view?



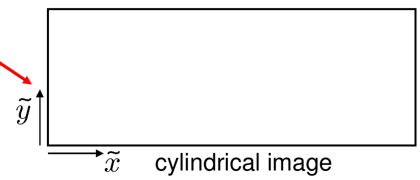
Cylindrical projection



unit cylinder



unwrapped cylinder



cylindrical image

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

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

- Convert to cylindrical coordinates

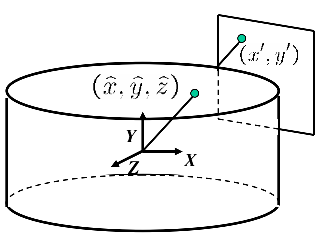
$$(sin\theta, h, cos\theta) = (\widehat{x},\widehat{y},\widehat{z})$$

- Convert to cylindrical image coordinates

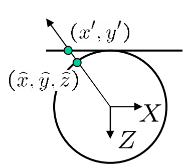
$$(\tilde{x},\tilde{y}) = (f\theta, fh) + (\tilde{x}_c,\tilde{y}_c)$$

Cylindrical reprojection

How to map from a cylinder to a planar image?



side view



top-down view

- Apply camera projection matrix
  - *w* = image width, *h* = image height

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} -f & 0 & w/2 & 0 \\ 0 & -f & h/2 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \widehat{x} \\ \widehat{y} \\ \widehat{z} \\ 1 \end{bmatrix}$$

- Convert to image coordinates
  - divide by third coordinate (w)

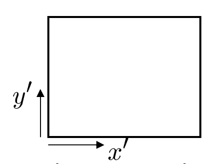


image coords

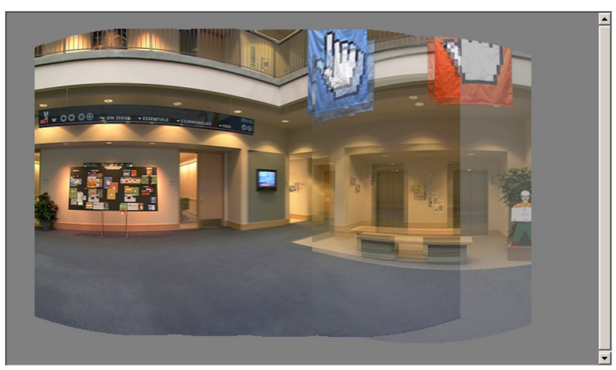
Cylindrical panoramas

Steps

- Reproject each image onto a cylinder
- Blend
- Output the resulting mosaic

What are the assumptions here?

Cylindrical image stitching



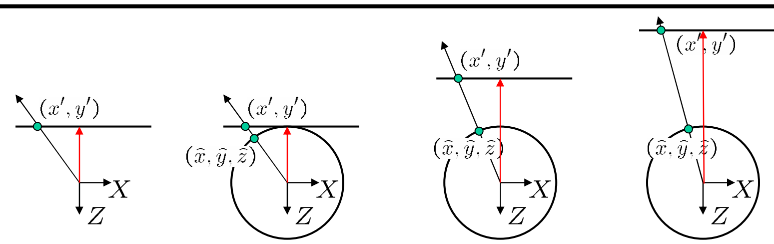
What if you don't know the camera rotation?

- Solve for the camera rotations
  - Note that a rotation of the camera is a **translation** of the cylinder!

Full-view Panorama

Different projections are possible

Cylindrical reprojection



top-down view

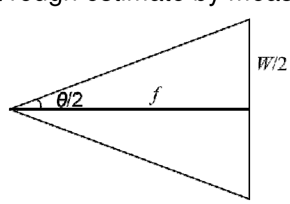
Focal length – the dirty secret…

Image 384x300 **f = 180 (pixels)** **f = 280** **f = 380**

What's your focal length, buddy?

Focal length is (highly!) camera dependant

- Can get a rough estimate by measuring FOV:



- Can use the EXIF data tag (might not give the right thing)
- Can use several images together and try to find f that would make them match
- Can use a known 3D object and its projection to solve for f
- Etc.

There are other camera parameters too:

- Optical center, non-square pixels, lens distortion, etc.

Camera calibration

Determine camera parameters from *known* 3D points or calibration object(s)

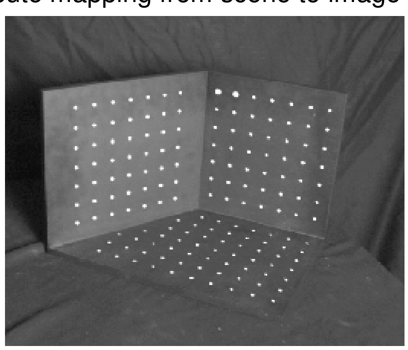
1. *internal* or *intrinsic* parameters such as focal length, optical center, aspect ratio:
   what kind of camera?
2. *external* or *extrinsic* (pose) parameters:
   where is the camera in the world coordinates?
   - World coordinates make sense for multiple cameras / multiple images

How can we do this?

Approach 1: solve for projection matrix

Place a known object in the scene

- identify correspondence between image and scene
- compute mapping from scene to image



$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \cong \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

Direct linear calibration

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \cong \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

Solve for Projection Matrix Π using least-squares (just like in homework)

Advantages:

- All specifics of the camera summarized in one matrix
- Can predict where any world point will map to in the image

Disadvantages:

- Doesn't tell us about particular parameters
- Mixes up internal and external parameters
  - pose specific: move the camera and everything breaks

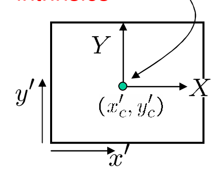
Approach 2: solve for parameters

A camera is described by several parameters

- Translation T of the optical center from the origin of world coords
- Rotation R of the image plane
- focal length f, principle point (x'_c, y'_c), pixel size (s_x, s_y)
- blue parameters are called "extrinsics," red are "intrinsics"

Projection equation

$$\mathbf{X} = \begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \mathbf{\Pi X}$$



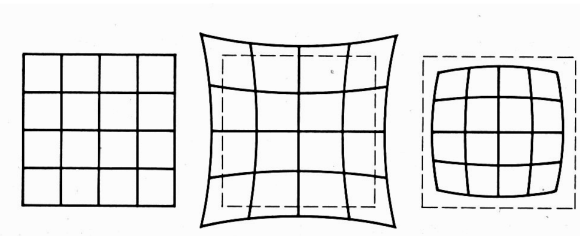
- The projection matrix models the cumulative effect of all parameters
- Useful to decompose into a series of operations

identity matrix

$$\mathbf{\Pi} = \underbrace{\begin{bmatrix} -fs_x & 0 & x'_c \\ 0 & -fs_y & y'_c \\ 0 & 0 & 1 \end{bmatrix}}_{\text{intrinsics}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{projection}} \underbrace{\begin{bmatrix} \mathbf{R}_{3x3} & \mathbf{0}_{3x1} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix}}_{\text{rotation}} \underbrace{\begin{bmatrix} \mathbf{I}_{3x3} & \mathbf{T}_{3x1} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix}}_{\text{translation}}$$

- Solve using non-linear optimization

Distortion

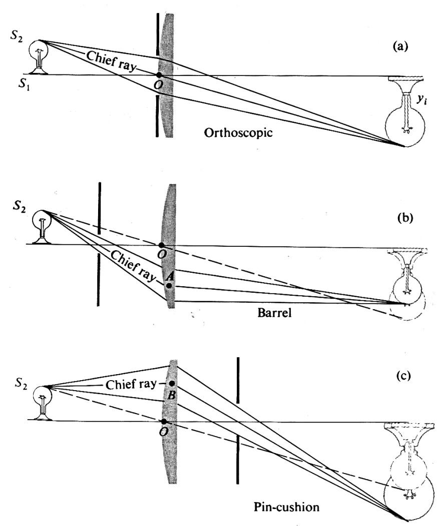


No distortion Pin cushion Barrel

Radial distortion of the image

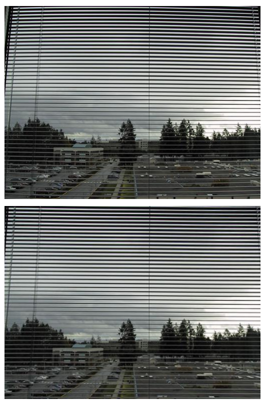
- Caused by imperfect lenses
- Deviations are most noticeable for rays that pass through the edge of the lens

Distortion



Radial distortion

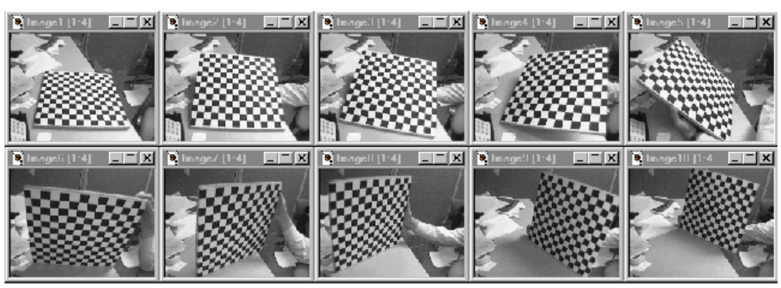
Correct for "bending" in wide field of view lenses



$$
\begin{aligned}
\widehat{r}^2 &= \widehat{x}^2 + \widehat{y}^2 \\
\widehat{x}' &= \widehat{x}/(1 + \kappa_1 \widehat{r}^2 + \kappa_2 \widehat{r}^4) \\
\widehat{y}' &= \widehat{y}/(1 + \kappa_1 \widehat{r}^2 + \kappa_2 \widehat{r}^4) \\
x &= f\widehat{x}'/\widehat{z} + x_c \\
y &= f\widehat{y}'/\widehat{z} + y_c
\end{aligned}
$$

Use this instead of normal projection

Multi-plane calibration



Images courtesy Jean-Yves Bouguet, Intel Corp.

Advantage

- Only requires a plane
- Don't have to know positions/orientations
- Good code available online!
  - Intel's OpenCV library: http://www.intel.com/research/mrl/research/opencv/
  - Matlab version by Jean-Yves Bouget: http://www.vision.caltech.edu/bouguetj/calib_doc/index.html
  - Zhengyou Zhang's web site: http://research.microsoft.com/~zhang/Calib/

Homography revisited

$$x = \Pi X \qquad \mathbf{\Pi} = \underbrace{\begin{bmatrix} -fs_x & 0 & x'_c \\ 0 & -fs_y & y'_c \\ 0 & 0 & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{P} \underbrace{\begin{bmatrix} \mathbf{R}_{3x3} & \mathbf{0}_{3x1} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix}}_{R} \underbrace{\begin{bmatrix} \mathbf{I}_{3x3} & \mathbf{T}_{3x1} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix}}_{T}$$

x = PRTX $\qquad$ $X \sim T^{-1}R^{-1}P^{-1}x$

$x_1 = P_1R_1T_1T_2^{-1}R_2^{-1}P_2^{-1}x_2 = Mx_2$

M is 4x4 but if all points X are on a plane, we can drop the last row and get our homography matrix H:

$x_1 \sim Hx_2$

Now, if the camera only rotates (no translation):

$H = K_1R_1R_2^{-1}K_2^{-1}$

Therefore, our homography has only 3,4 or 5 DOF, depending if focal length is known, same, or different.

- This makes image registration much better behaved

Image registration

How do we determine alignment between images?

- **Direct (pixel-based) alignment**
- One possibility: *block matching* (correlation), i.e., find minimum squared error

$$E(u,v) = \sum_{(x,y)} [I_1(x+u, y+v) - I_0(x,y)]^2$$


- Another possiblility: Fourier-domain correlation [Brown'92]
- But have to be more clever when more DOF are needed

Image registration

How do we determine alignment between images?

- Feature-based Alignment
- Match features between images and use as correspondences
- But matching is tricky:
  - Features look like each other
  - Features don't look like themselves when transformed