

# More Mosaic Madness

---



© Jeffrey Martin ([jeffrey-martin.com](http://jeffrey-martin.com))

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

15-463: Computational Photography  
Alexei Efros, CMU, Fall 2008

# Homography

---

A: Projective – mapping between any two PPs with the same center of projection

- rectangle should map to arbitrary quadrilateral
- parallel lines aren't
- but must preserve straight lines
- same as: project, rotate, reproject

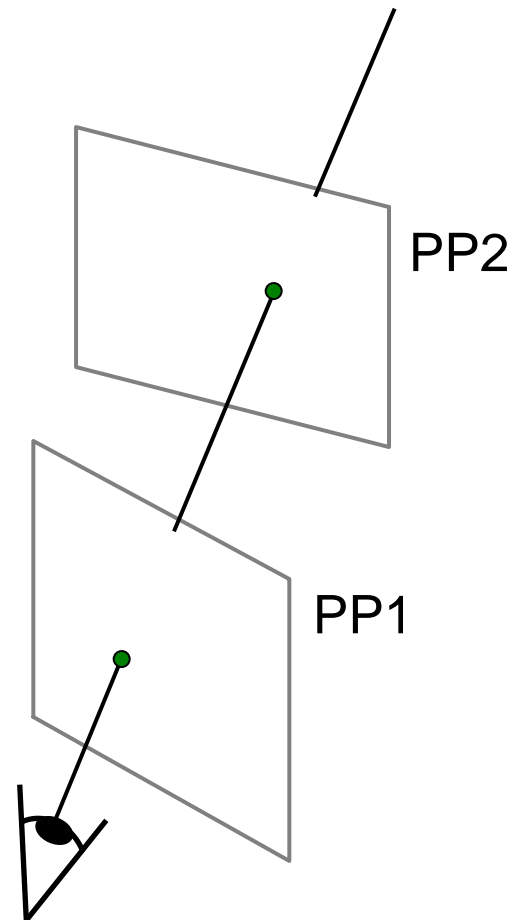
called Homography

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

$\mathbf{p}' \qquad \mathbf{H} \qquad \mathbf{p}$

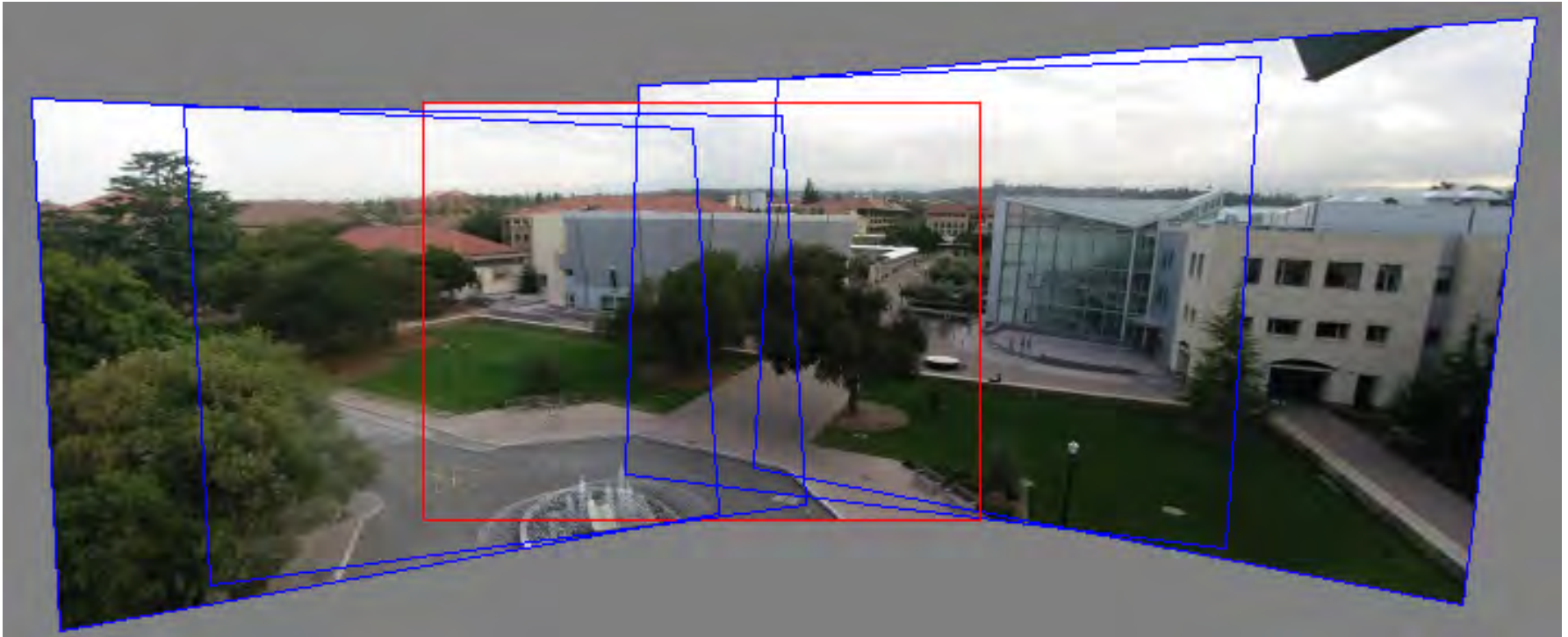
To apply a homography  $\mathbf{H}$

- Compute  $\mathbf{p}' = \mathbf{H}\mathbf{p}$  (regular matrix multiply)
- Convert  $\mathbf{p}'$  from homogeneous to image coordinates



# Rotational Mosaics

---

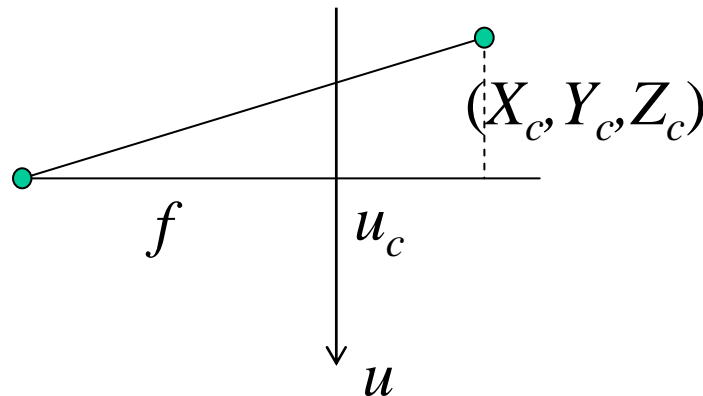


Can we say something more about rotational mosaics?  
i.e. can we further constrain our  $H$ ?

# 3D $\rightarrow$ 2D Perspective Projection

---

$$(x, y, z) \rightarrow \left(-d\frac{x}{z}, -d\frac{y}{z}\right)$$



$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} f & 0 & u_c \\ 0 & f & v_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

K

# 3D Rotation Model

---

Projection equations

1. Project from image to 3D ray

$$(x_0, y_0, z_0) = (u_0 - u_c, v_0 - v_c, f)$$

2. Rotate the ray by camera motion

$$(x_1, y_1, z_1) = \mathbf{R}_{01} (x_0, y_0, z_0)$$

3. Project back into new (source) image

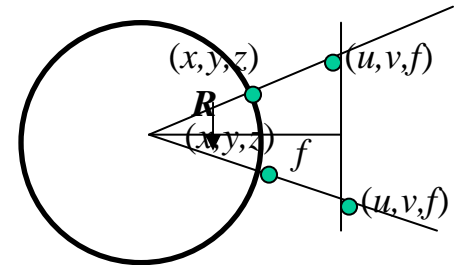
$$(u_1, v_1) = (fx_1/z_1 + u_c, fy_1/z_1 + v_c)$$

Therefore:

$$\mathbf{H} = \mathbf{K}_0 \mathbf{R}_{01} \mathbf{K}_1^{-1}$$

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



# Pairwise alignment

---



Procrustes Algorithm [Golub & VanLoan]

Given two sets of matching points, compute  $R$

$$p_i' = \mathbf{R} p_i \quad \text{with 3D rays}$$

$$p_i = N(x_i, y_i, z_i) = N(u_i - u_c, v_i - v_c, f)$$

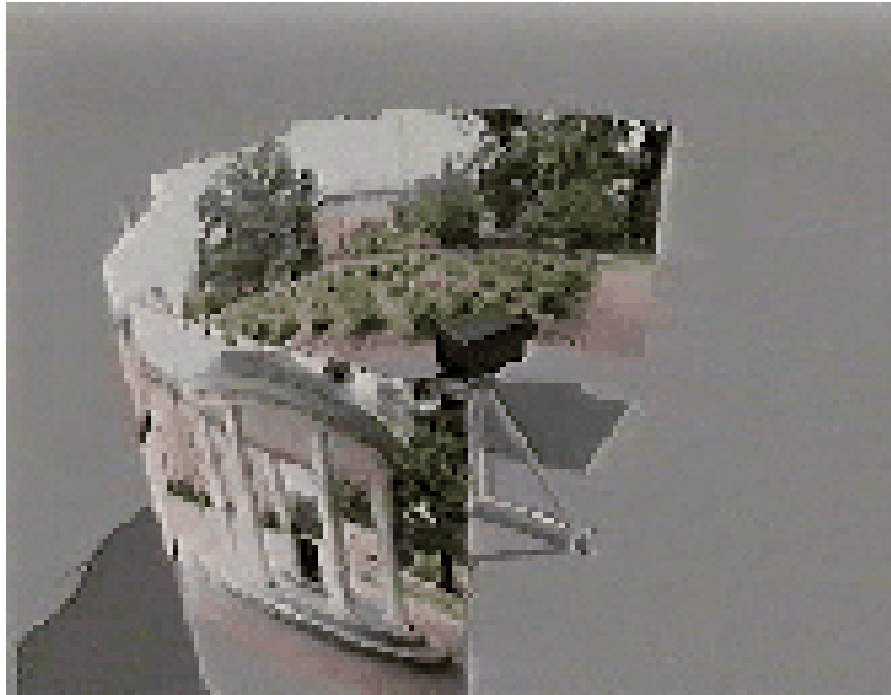
$$\mathbf{A} = \sum_i p_i p_i'^T = \sum_i p_i p_i^T \mathbf{R}^T = \mathbf{U} \mathbf{S} \mathbf{V}^T = (\mathbf{U} \mathbf{S} \mathbf{U}^T) \mathbf{R}^T$$

$$\mathbf{V}^T = \mathbf{U}^T \mathbf{R}^T$$

$$\mathbf{R} = \mathbf{V} \mathbf{U}^T$$

# Rotation about vertical axis

---

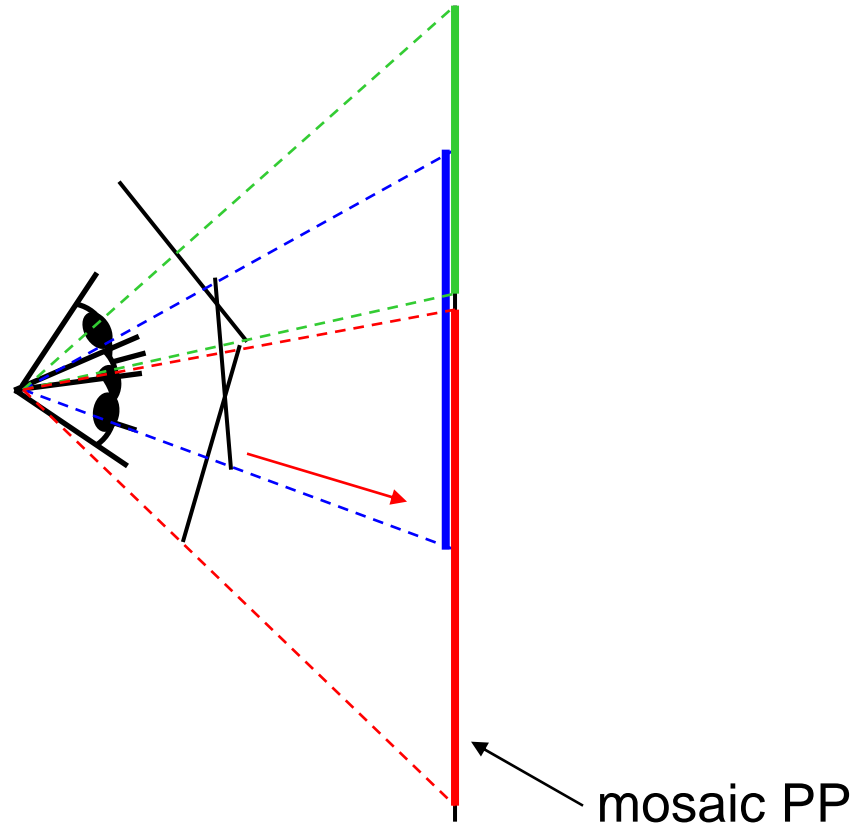


What if our camera rotates on a tripod?

What's the structure of  $H$ ?

# Do we have to project onto a plane?

---

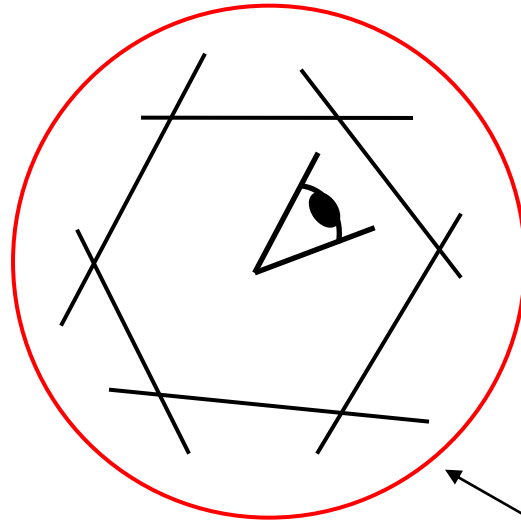




# Full Panoramas

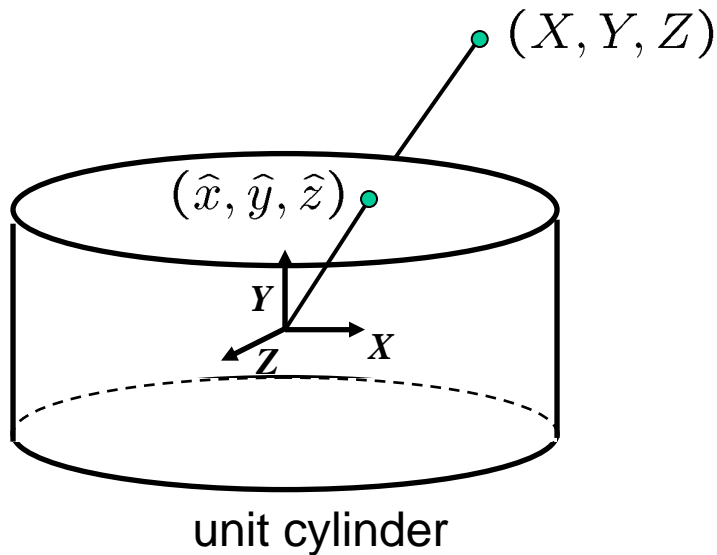
---

What if you want a 360° field of view?



mosaic Projection Cylinder

# Cylindrical projection



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

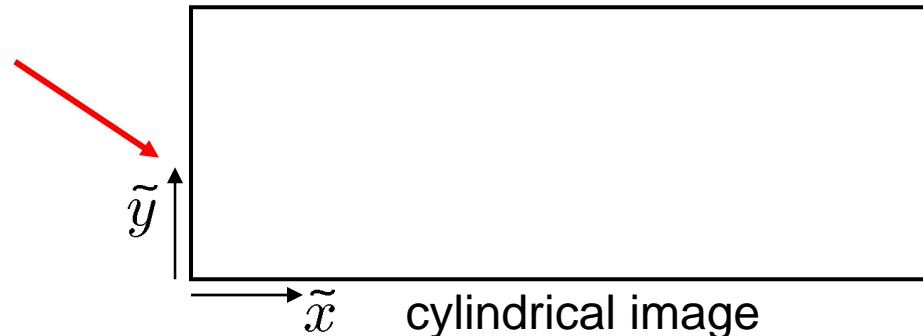
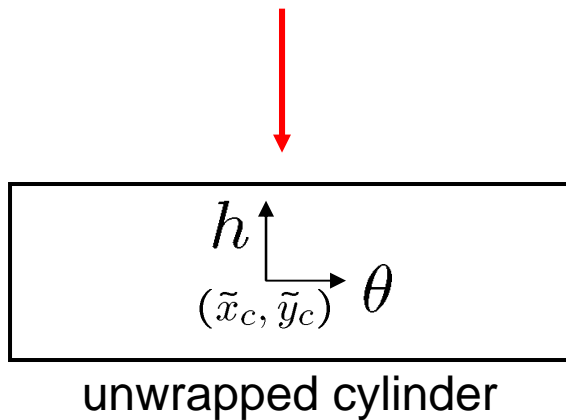
$$(\hat{x}, \hat{y}, \hat{z}) = \frac{1}{\sqrt{X^2 + Z^2}}(X, Y, Z)$$

- Convert to cylindrical coordinates

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

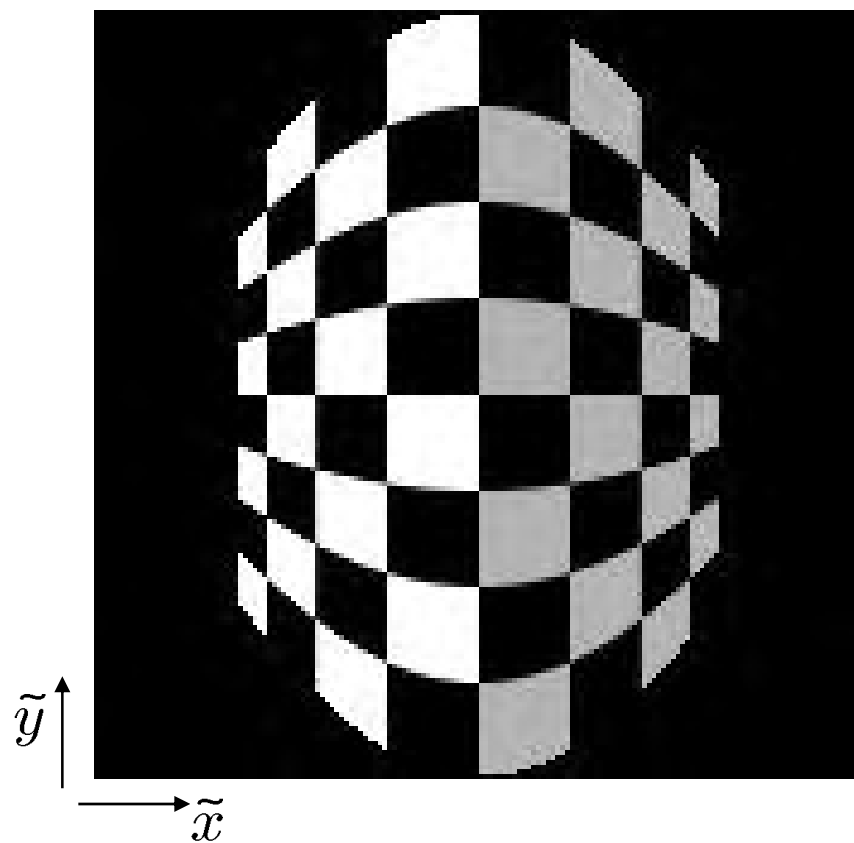
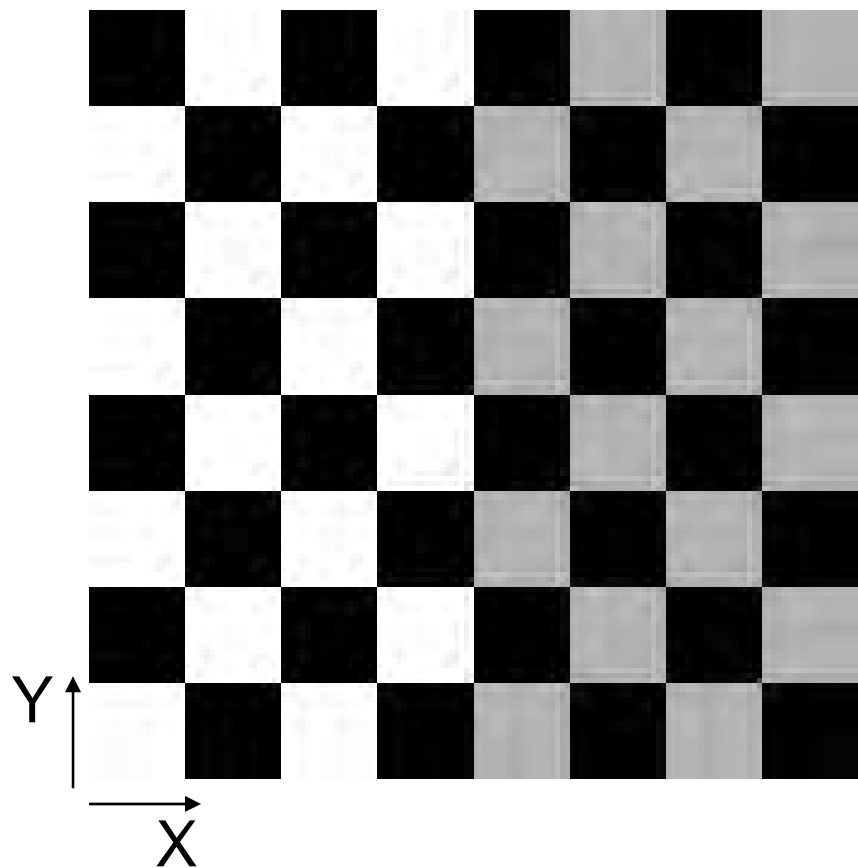
- Convert to cylindrical image coordinates

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



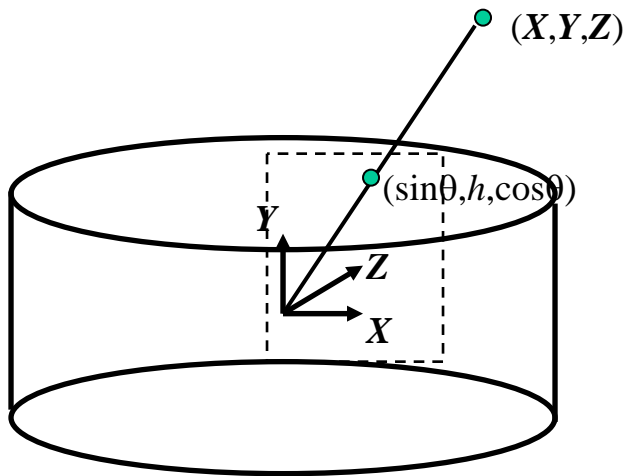
# Cylindrical Projection

---



# Inverse Cylindrical projection

---



$$\theta = (x_{cyl} - x_c) / f$$

$$h = (y_{cyl} - y_c) / f$$

$$\hat{x} = \sin \theta$$

$$\hat{y} = h$$

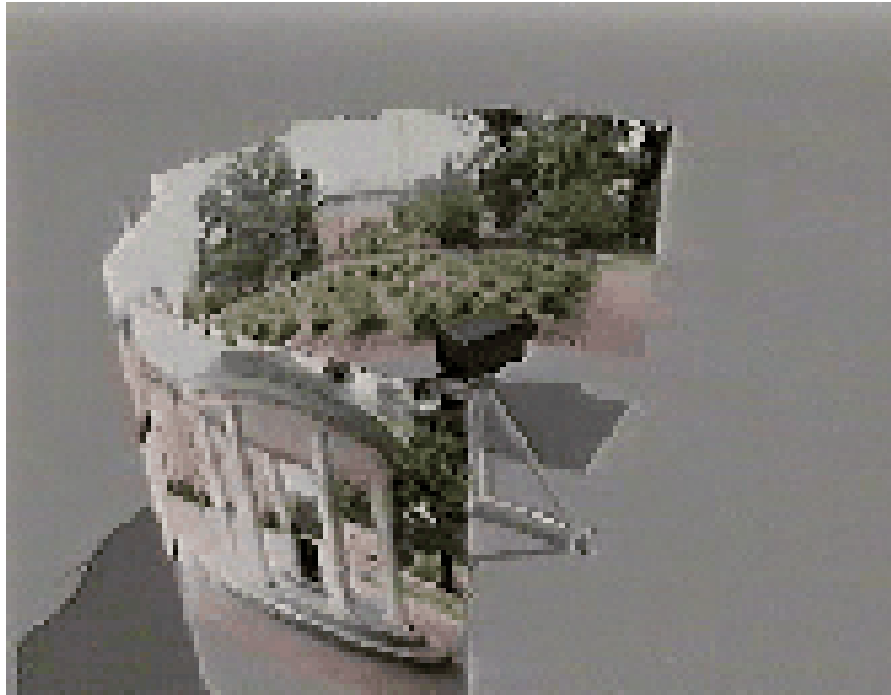
$$\hat{z} = \cos \theta$$

$$x = f \hat{x} / \hat{z} + x_c$$

$$y = f \hat{y} / \hat{z} + y_c$$

# Cylindrical panoramas

---



## Steps

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

# Cylindrical image stitching

---

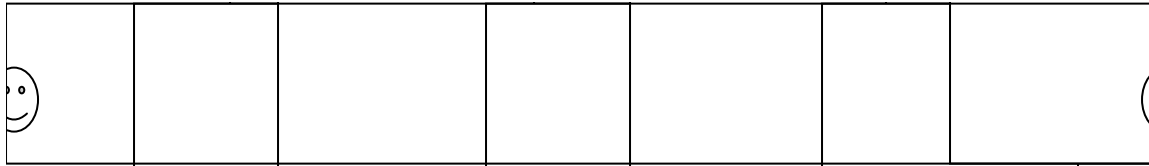


What if you don't know the camera rotation?

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

# Assembling the panorama

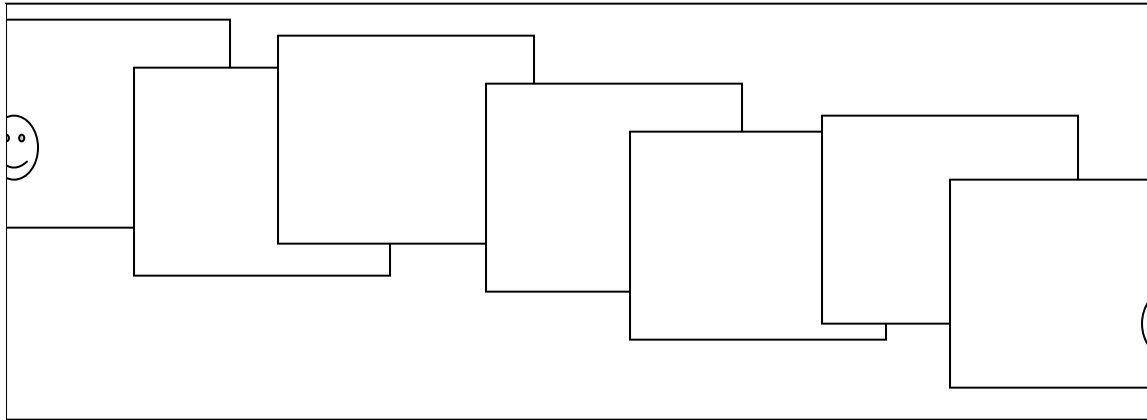
---



Stitch pairs together, blend, then crop

# Problem: Drift

---



## Vertical Error accumulation

- small (vertical) errors accumulate over time
- apply correction so that sum = 0 (for 360° pan.)

## Horizontal Error accumulation

- can reuse first/last image to find the right panorama radius

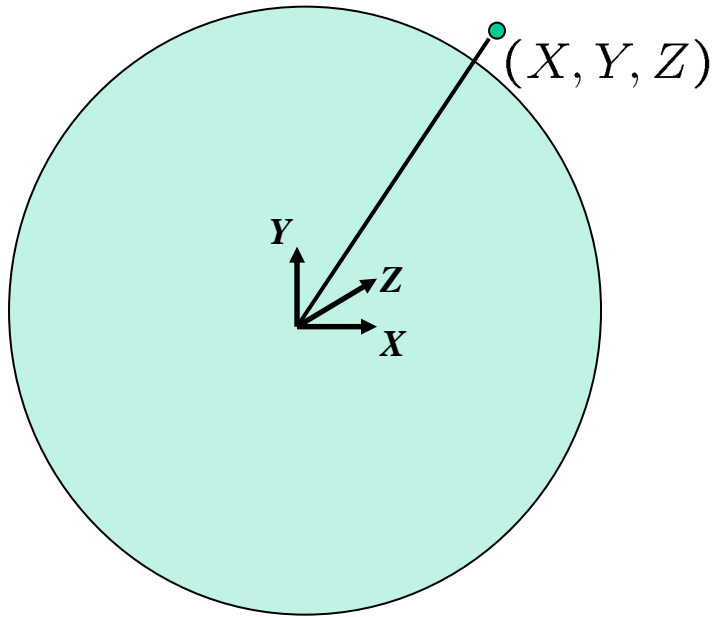


# Full-view (360°) panoramas

---



# Spherical projection

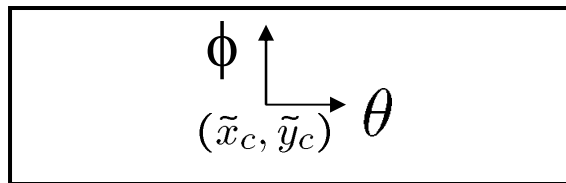


- Map 3D point (X,Y,Z) onto sphere

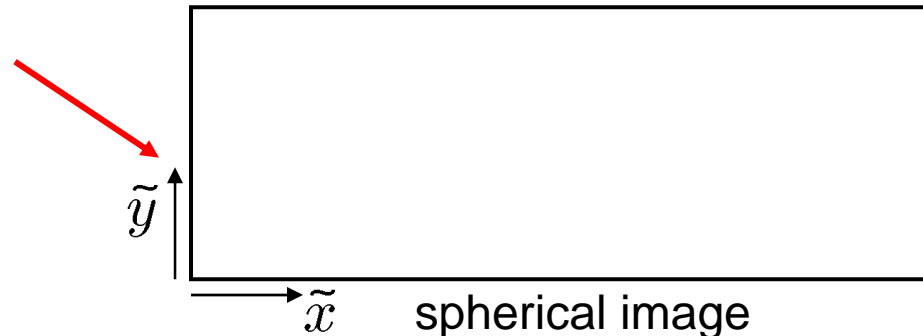
$$(\hat{x}, \hat{y}, \hat{z}) = \frac{1}{\sqrt{X^2 + Y^2 + Z^2}}(X, Y, Z)$$

- 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

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



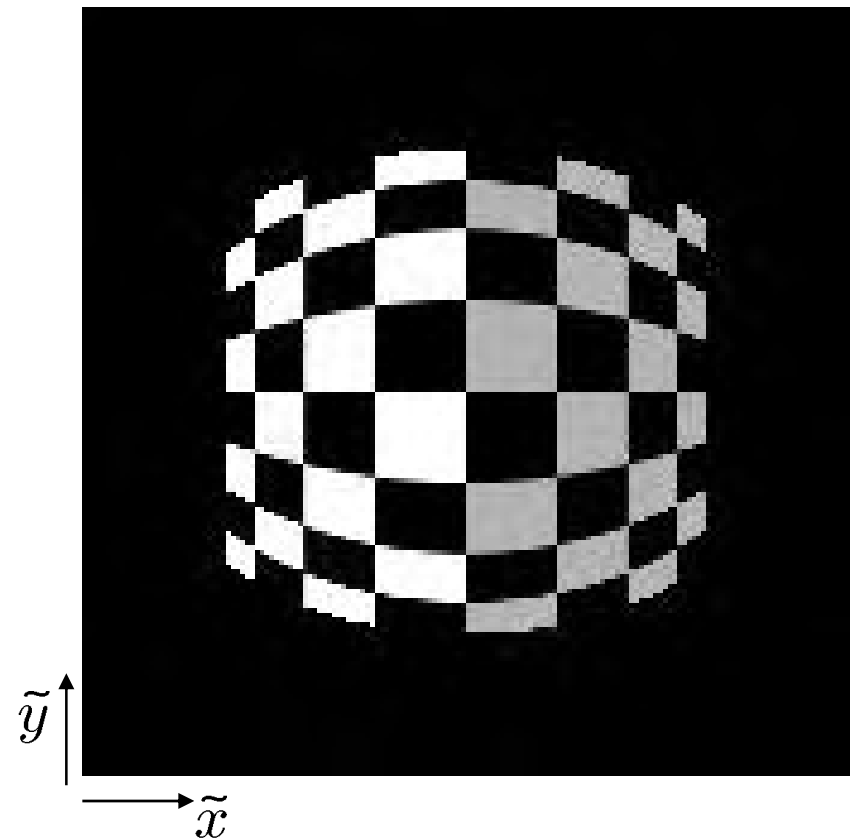
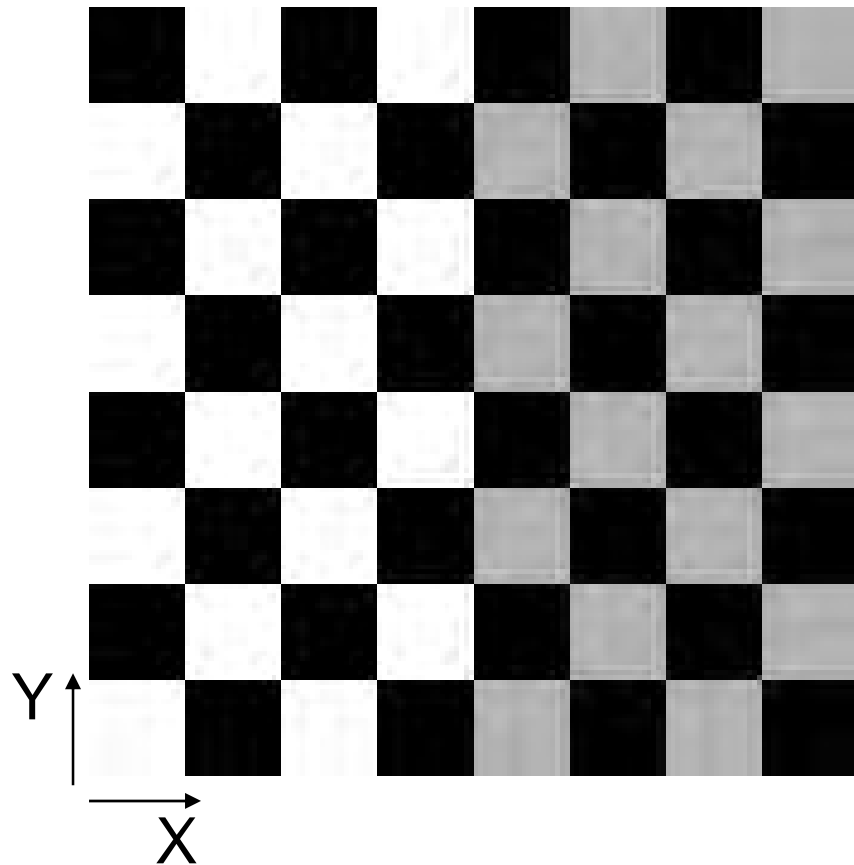
unwrapped sphere



spherical image

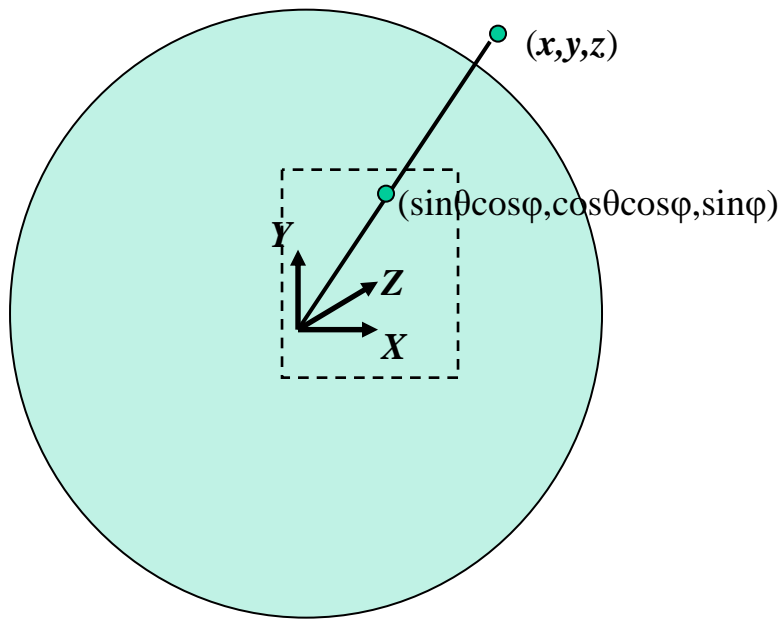
# Spherical Projection

---



# Inverse Spherical projection

---



$$\theta = (x_{sph} - x_c) / f$$

$$\varphi = (y_{sph} - y_c) / f$$

$$\hat{x} = \sin \theta \cos \varphi$$

$$\hat{y} = \sin \varphi$$

$$\hat{z} = \cos \theta \cos \varphi$$

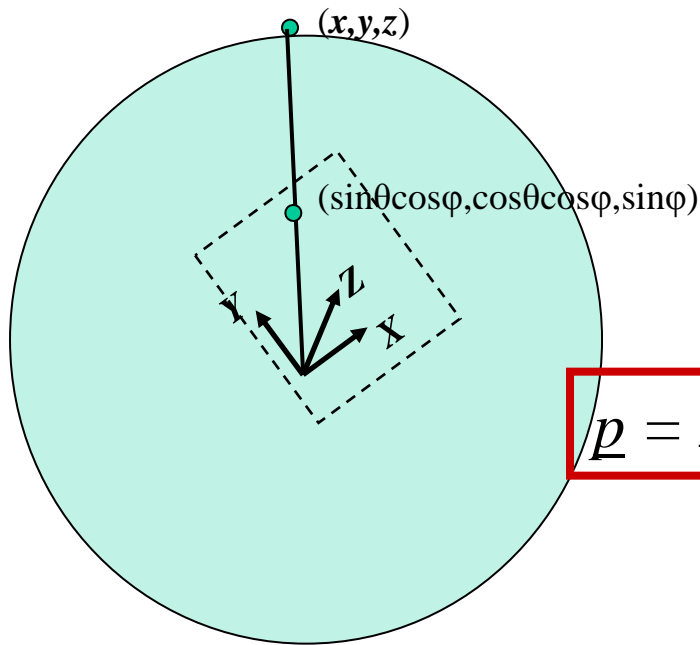
$$x = f \hat{x} / \hat{z} + x_c$$

$$y = f \hat{y} / \hat{z} + y_c$$

# 3D rotation

---

Rotate image before placing on unrolled sphere



$$\theta = (x_{sph} - x_c) / f$$

$$\varphi = (y_{sph} - y_c) / f$$

$$\hat{x} = \sin \theta \cos \varphi$$

$$\hat{y} = \sin \varphi$$

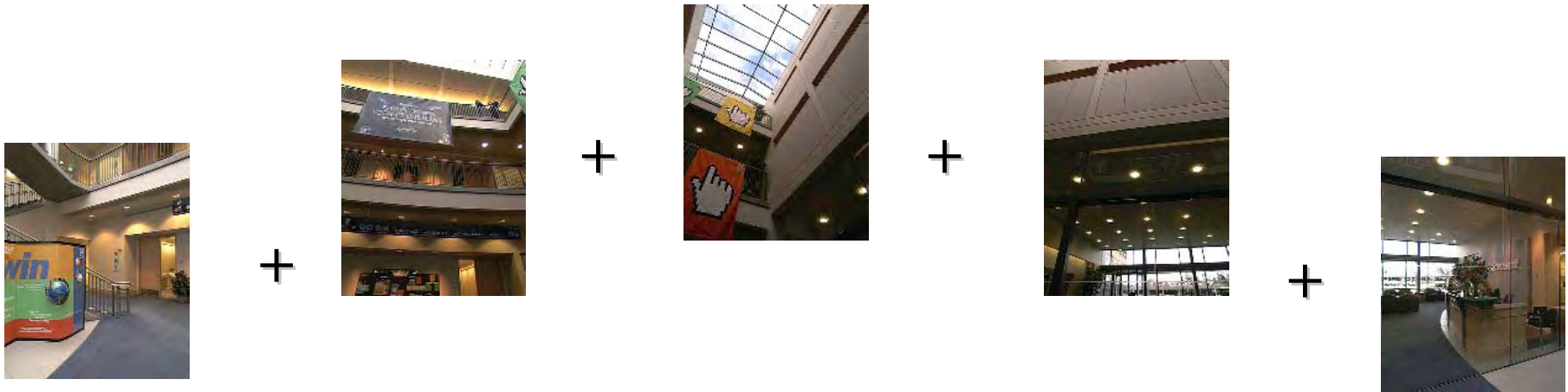
$$\hat{z} = \cos \theta \cos \varphi$$

$$x = f \hat{x} / \hat{z} + x_c$$

$$y = f \hat{y} / \hat{z} + y_c$$

# Full-view Panorama

---



# Other projections are possible

---



You can stitch on the plane and then warp the resulting panorama

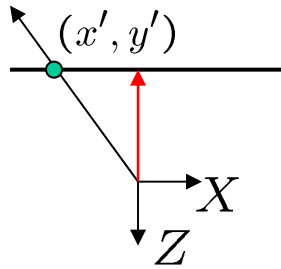
- What's the limitation here?

Or, you can use these as stitching surfaces

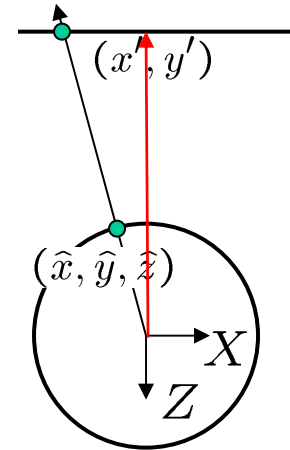
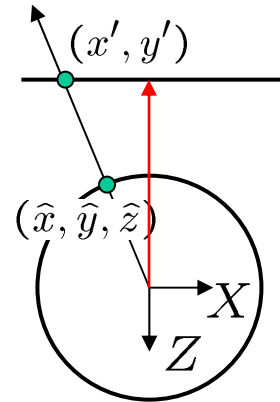
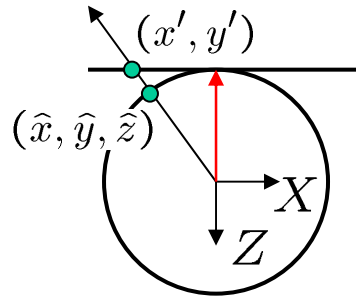
- But there is a catch...



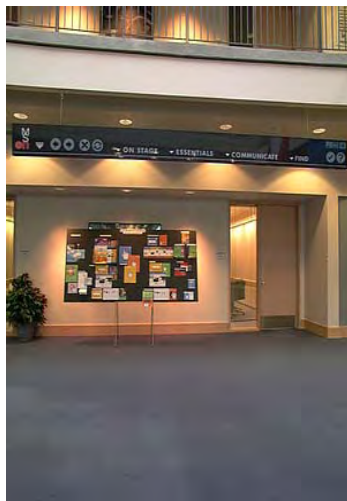
# Cylindrical reprojection



top-down view



**Focal length** – the dirty secret...



**Image 384x300**



**f = 180 (pixels)**



**f = 280**



**f = 380**

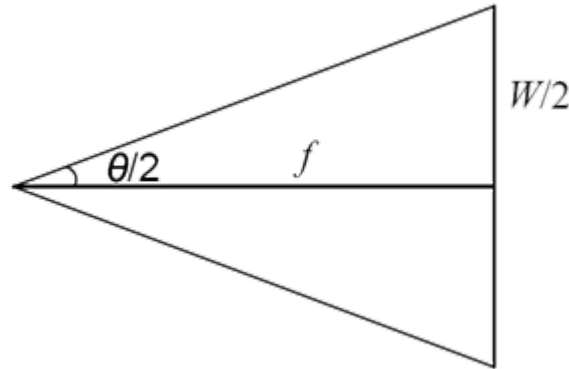


# What's your focal length, buddy?

---

Focal length is (highly!) camera dependant

- Can get a rough estimate by measuring FOV:



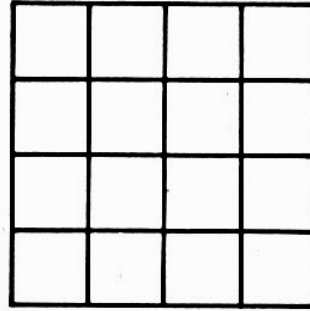
- Can use the EXIF data tag (might not give the right thing)
- Can use several images together and try to find  $f$  that would make them match
- Can use a known 3D object and its projection to solve for  $f$
- Etc.

There are other camera parameters too:

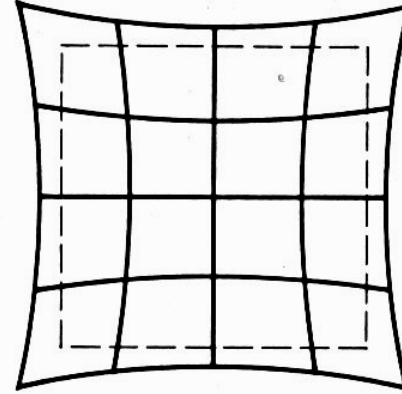
- Optical center, non-square pixels, lens distortion, etc.

# Distortion

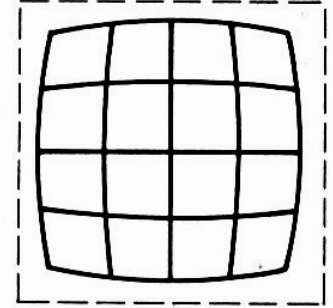
---



No distortion



Pin cushion



Barrel

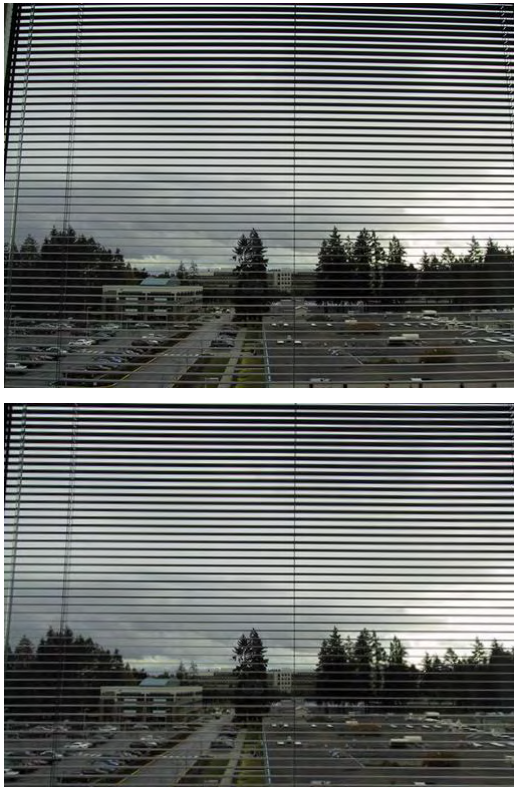
## Radial distortion of the image

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

# Radial distortion

---

Correct for “bending” in wide field of view lenses



$$\hat{r}^2 = \hat{x}^2 + \hat{y}^2$$

$$\hat{x}' = \hat{x} / (1 + \kappa_1 \hat{r}^2 + \kappa_2 \hat{r}^4)$$

$$\hat{y}' = \hat{y} / (1 + \kappa_1 \hat{r}^2 + \kappa_2 \hat{r}^4)$$

$$x = f \hat{x}' / \hat{z} + x_c$$

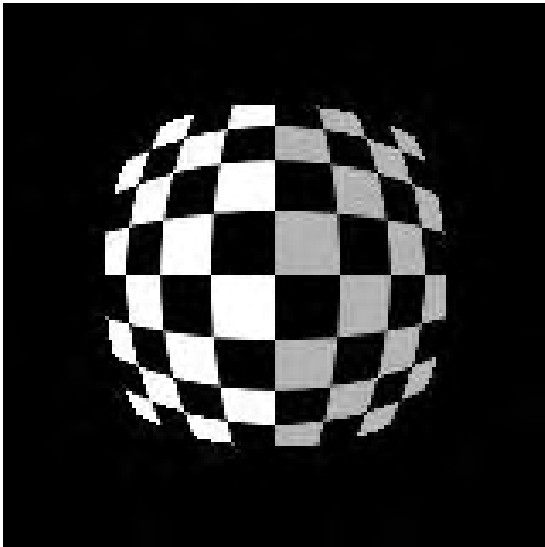
$$y = f \hat{y}' / \hat{z} + y_c$$

Use this instead of normal projection

# Polar Projection

---

Extreme “bending” in ultra-wide fields of view



$$\hat{r}^2 = \hat{x}^2 + \hat{y}^2$$

$$(\cos \theta \sin \phi, \sin \theta \sin \phi, \cos \phi) = s(x, y, z)$$

Equations become

$$x' = s\phi \cos \theta = s \frac{x}{r} \tan^{-1} \frac{r}{z},$$
$$y' = s\phi \sin \theta = s \frac{y}{r} \tan^{-1} \frac{r}{z},$$



# 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

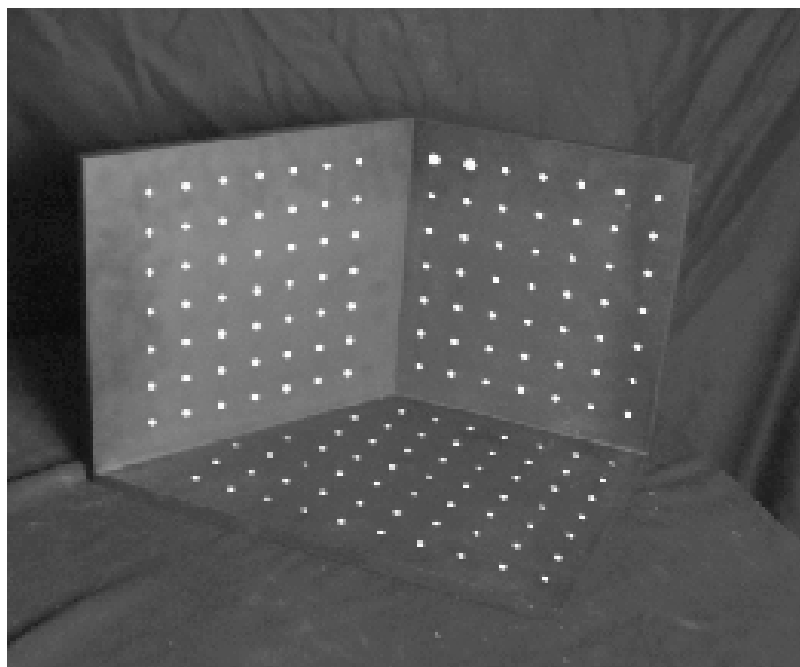
How can we do this?

# 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



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

# 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  $\Pi$  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

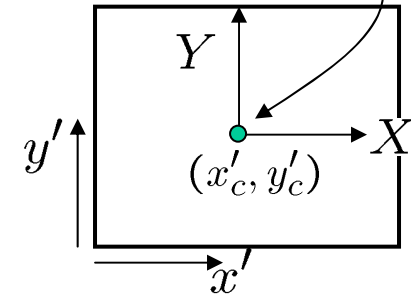
# Approach 2: solve for parameters

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 “intrinsic”

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

$$\mathbf{\Pi} = \begin{bmatrix} -fs_x & 0 & x'_c \\ 0 & -fs_y & y'_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{3 \times 3} & \mathbf{0}_{3 \times 1} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{T}_{3 \times 1} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix}$$

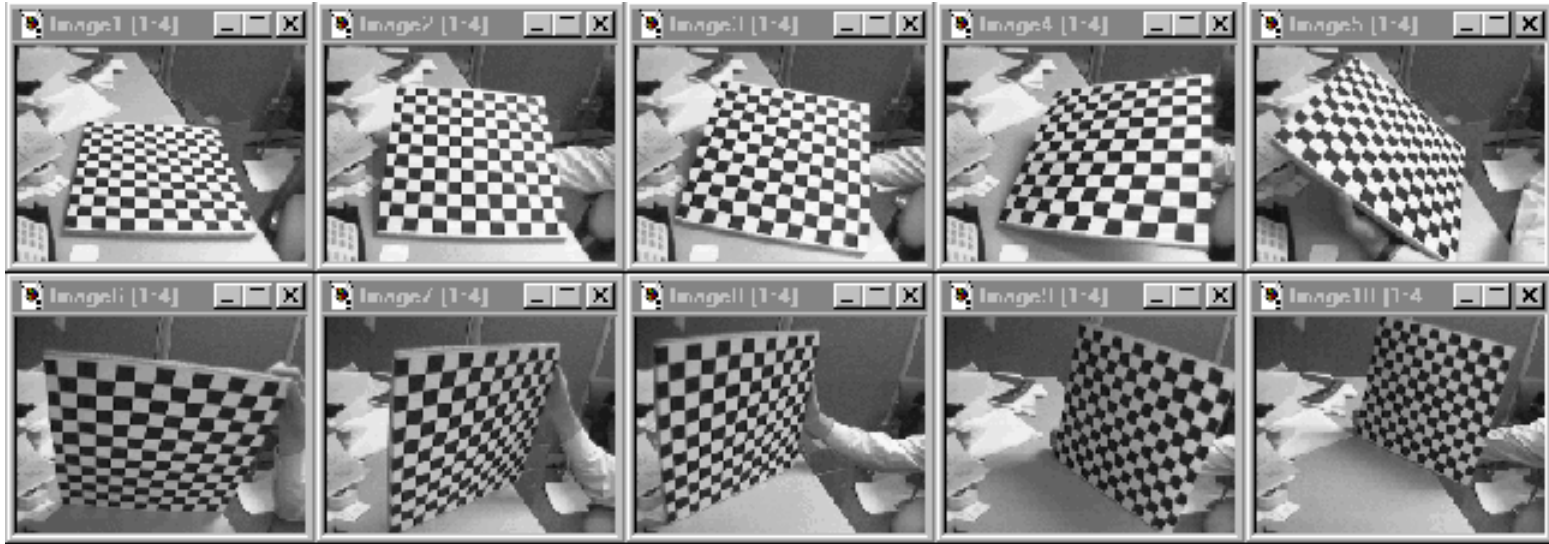
intrinsic                  projection                  rotation                  translation

identity matrix

- Solve using non-linear optimization



# 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 Bouguet: [http://www.vision.caltech.edu/bouguetj/calib\\_doc/index.html](http://www.vision.caltech.edu/bouguetj/calib_doc/index.html)
  - Zhengyou Zhang's web site: <http://research.microsoft.com/~zhang/Calib/>