Geometry in Computer Vision

Perspective Model, Calibration, and Stereo

CSE 6367 – Computer Vision
Vassilis Athitsos
University of Texas at Arlington

Part 1

Perspective Camera Model

3D information

- Ideally (but rarely in practice), we would like to know for every pixel:
  - How far the location depicted in that pixel is from the camera.
- What other types of 3D information would we want to know about objects and surfaces visible in the image?

3D information

- Ideally (but rarely in practice), we would like to know for every pixel:
  - How far the location depicted in that pixel is from the camera.
- For the objects and surfaces that are visible in the image, we would like to know:
  - what their 3D shape is
  - where they are located in 3D
  - how big they are
  - how far they are from the camera and from each other.

The Need for 3D Information

- What kind of applications would benefit from estimating 3D information?

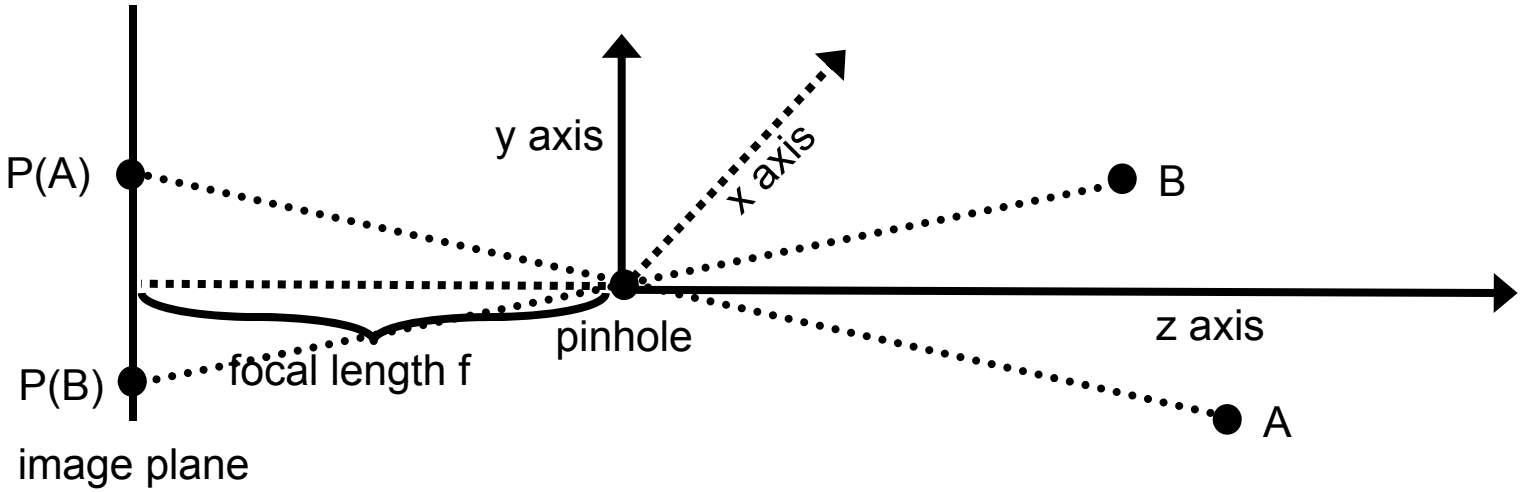
The Need for 3D Information

- What kind of applications would benefit from estimating 3D information?
  - A robot that wants to grasp an object must know how far its hand is from the object.
  - An unmanned vehicle needs to know how far obstacles are, in order to determine if it is safe to continue moving or not.
  - 3D information can tell us, for a person viewed from the side, whether the left leg or the right leg is at the front.
  - 3D information can help determine the object where someone is pointing.

From 2D to 3D and Vice Versa

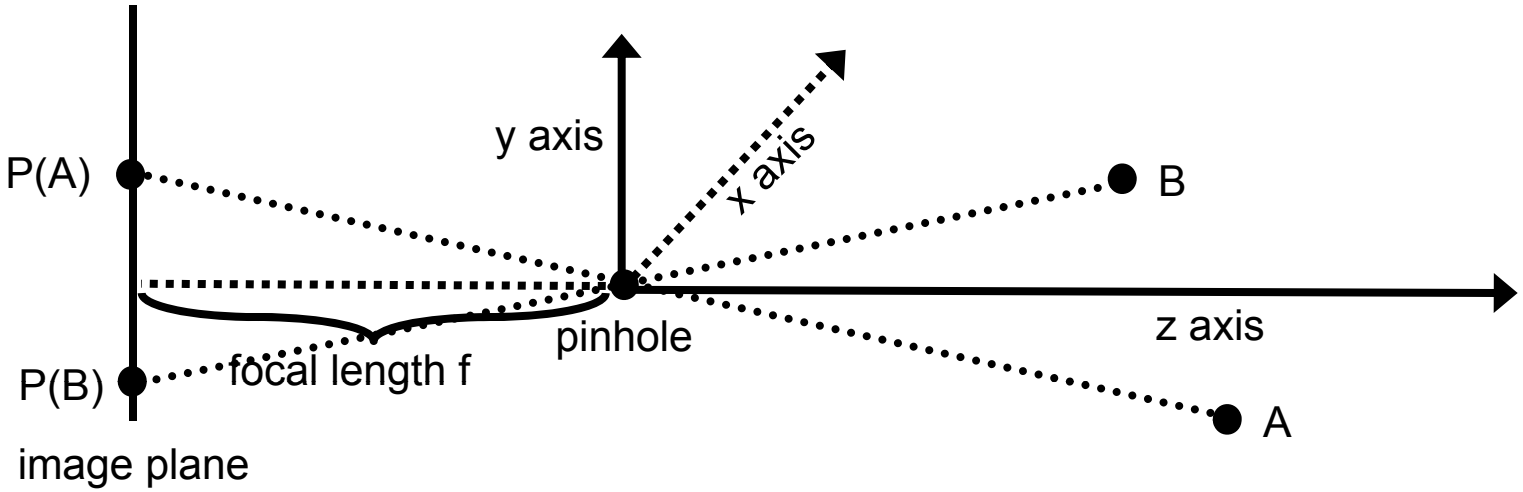
- To estimate 3D information, we ask the question:
  - Given a pixel (u, v), what 3D point (x, y, z) is seen at that pixel?
- That is a hard problem (one-to-many).
  - Can be solved if we have additional constraints.
  - For example, if we have two cameras (stereo vision).
- We start by solving the inverse problem, which is easier:
  - Given a 3D point (x, y, z), what pixel (u, v) does that 3D point map to?
  - This can be easily solved, as long as we know some camera parameters.

Pinhole Model



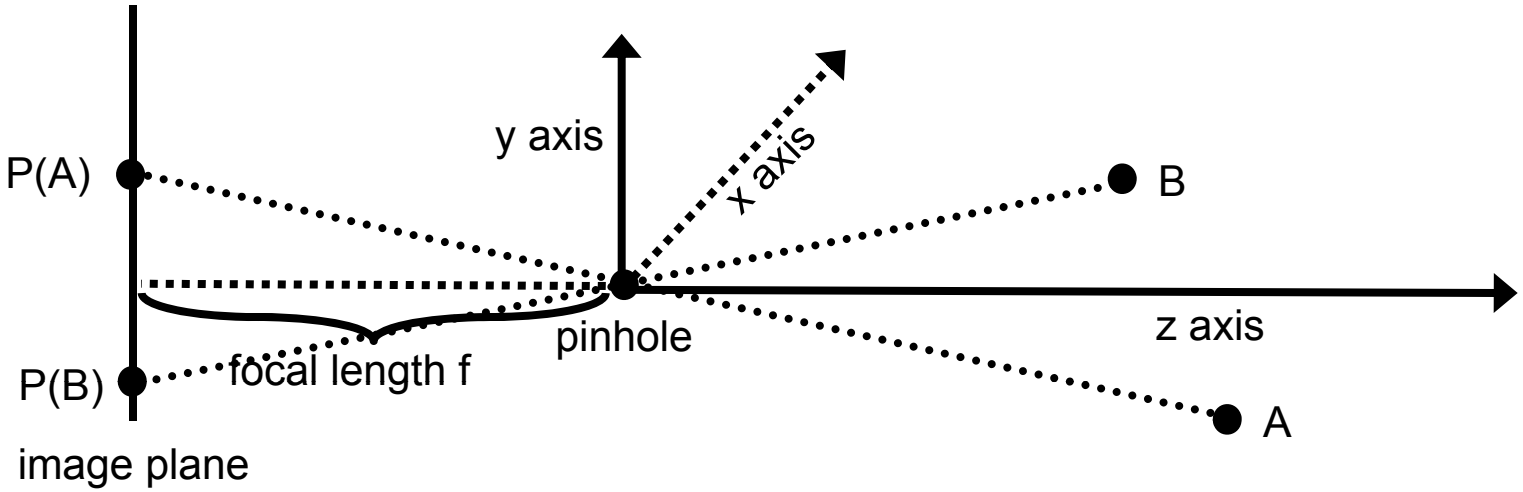
- Terminology:
  - *image plane* is a planar surface of sensors. The response of those sensors to light forms the image.
  - The *focal length f* is the distance between the image plane and the pinhole.
  - A set of points is *collinear* if there exists a straight line going through all points in the set.

Pinhole Model



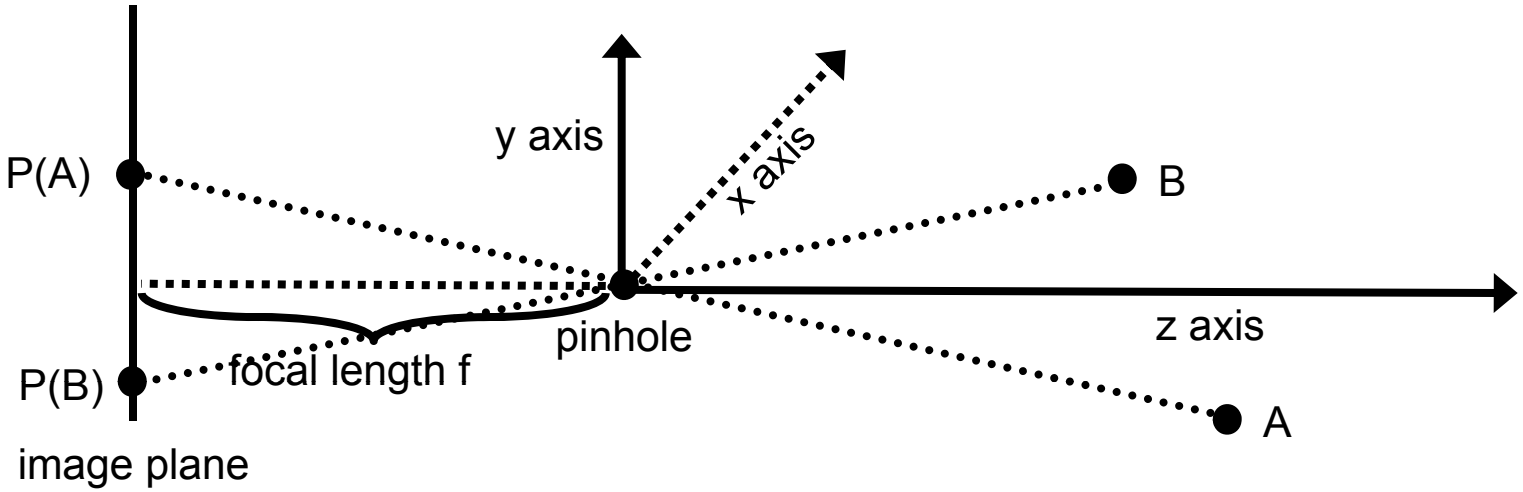
- Pinhole model:
  - light from all points enters the camera through an infinitesimal hole, and then reaches the *image plane*.
  - The *focal length f* is the distance between the image plane and the pinhole.
  - the light from point A reaches image location P(A), such that A, the pinhole, and P(A) are *collinear*.

Different Coordinate Systems



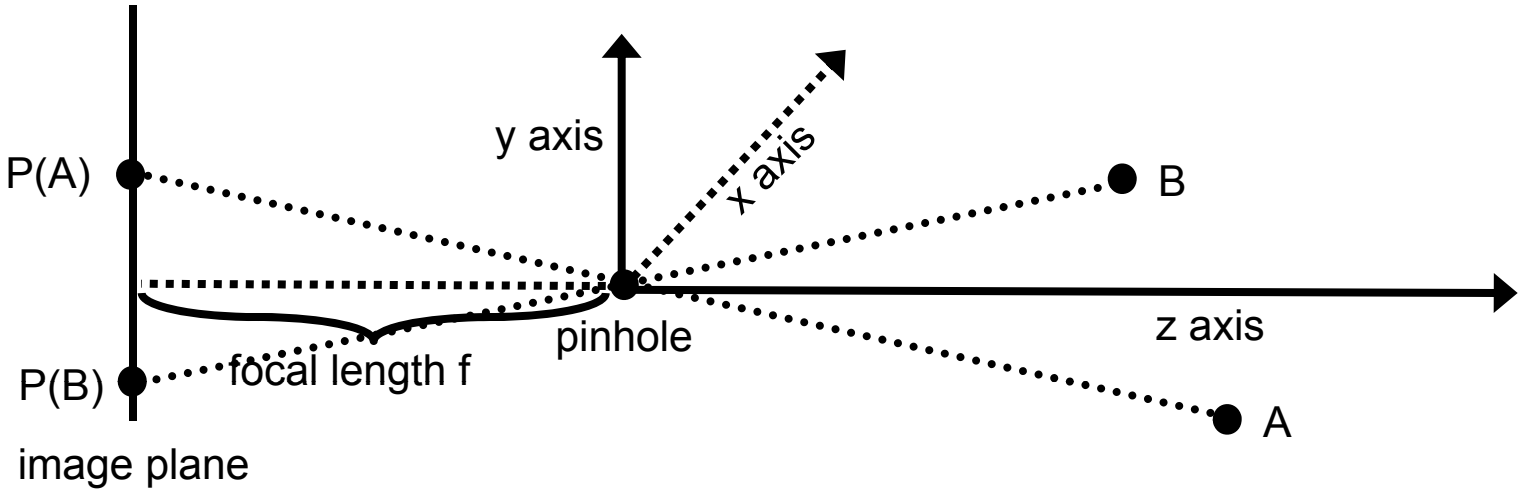
- World coordinate system (3D):
  - Pinhole is at location t, and at orientation R.
- Camera coordinate system (3D):
  - Pinhole is at the origin.
  - The camera faces towards the positive side of the z axis.

Different Coordinate Systems



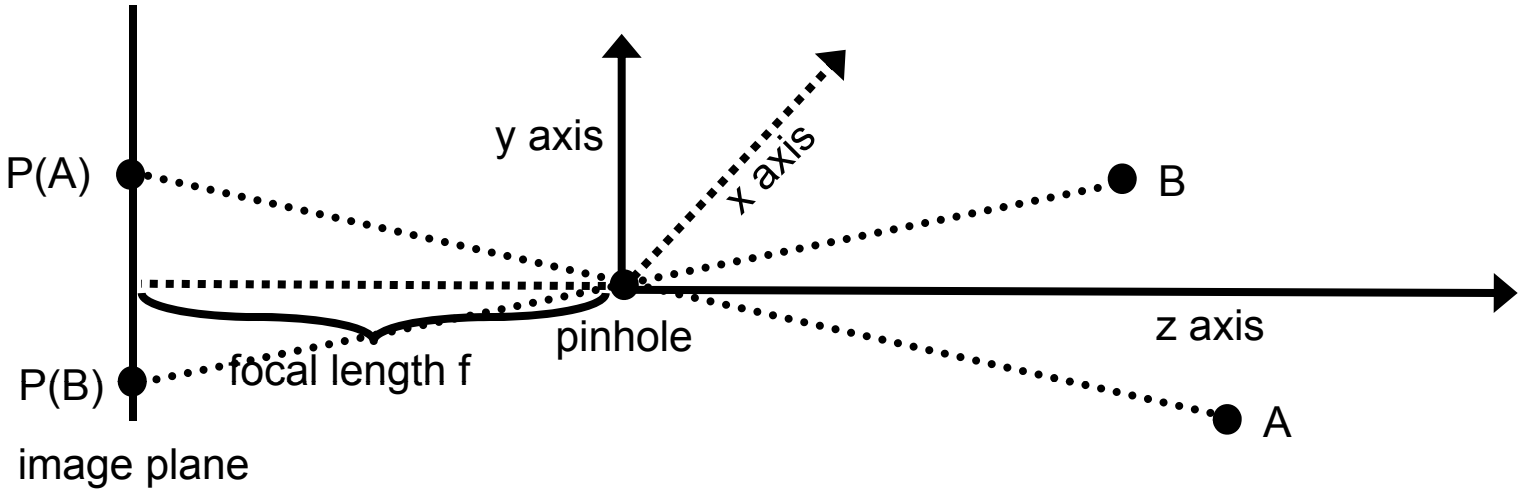
- Normalized image coordinate system (2D):
  - Coordinates on the image plane.
    - The (x, y) values of the camera coordinate system.
    - We drop the z value (always equal to f, not of interest).
  - Center of image is (0, 0).
- Image (pixel) coordinate system (2D):
  - pixel coordinates.

Pinhole Model



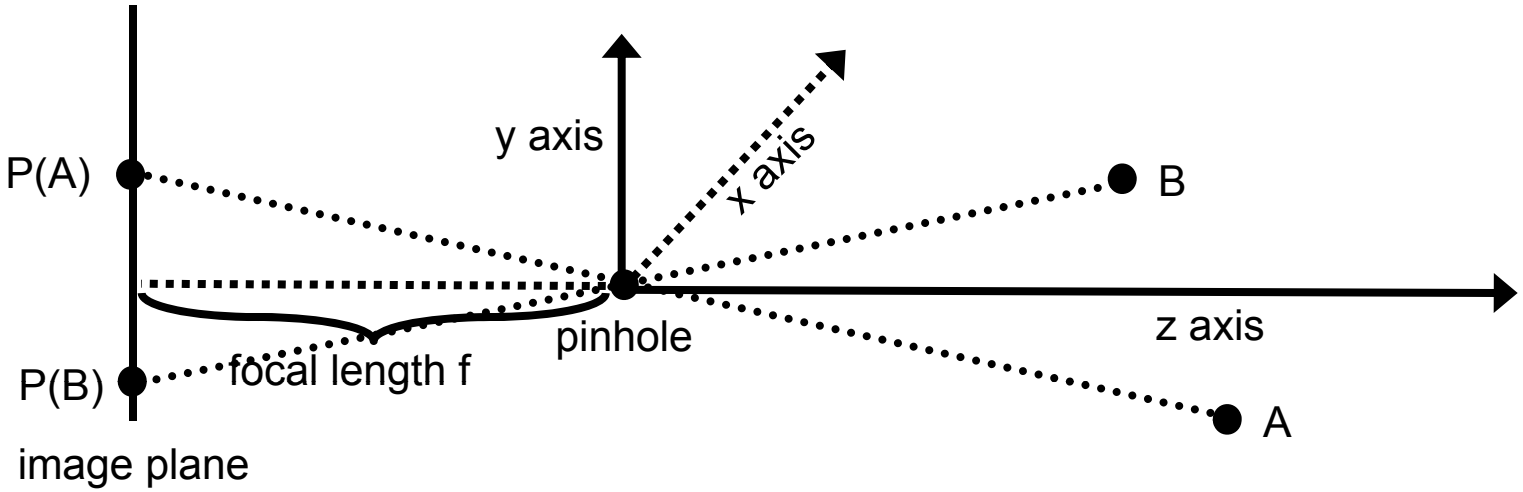
- A simple example:
  - Assume that world coordinates = camera coordinates.
  - Assume that the z axis points right, the y axis points up.
    - The x axis points away from us.
- If A is at position (A_x, A_y, A_z), what is P(A)?
  - Note: A is in world coordinates, P(A) is in normalized image coordinates.

Pinhole Model



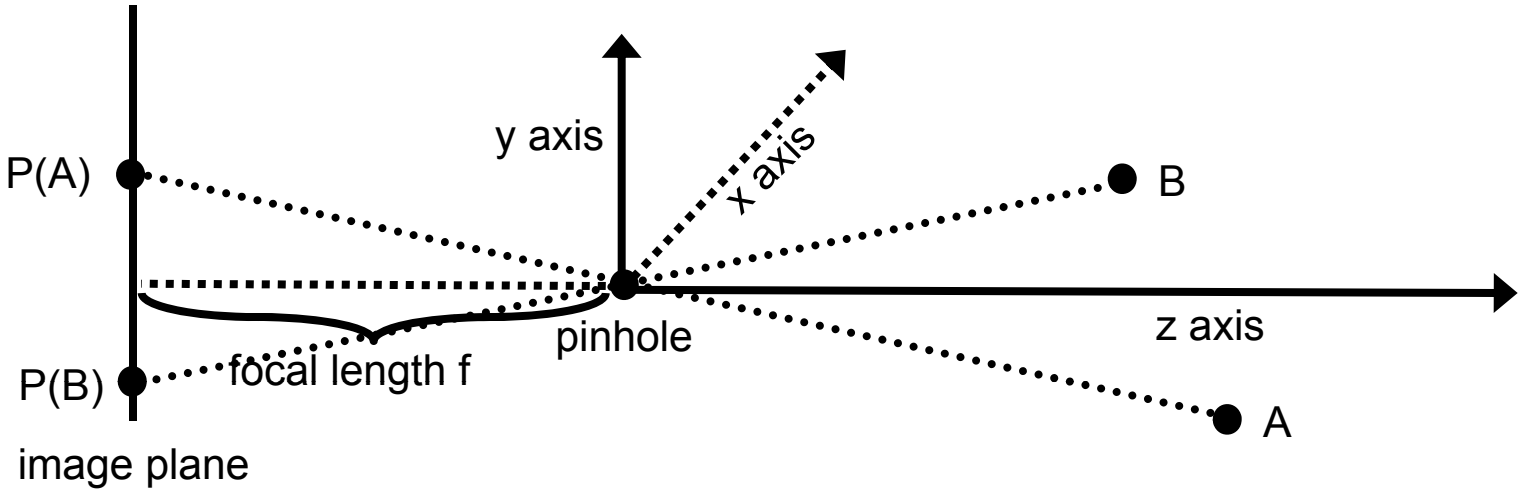
- $P(A) = (-A_x/A_z * f, -A_y/A_z * f)$.
  - P(A) is **two-dimensional** (normalized image coordinates).
- This is a simple formula, because we chose a convenient coordinate system (world coordinates = camera coordinates).
- What happens if the pinhole is at (C_x, C_y, C_z)?

Handling Camera Translation



- If the pinhole is at (C_x, C_y, C_z)?
- We define a change-of-coordinates transformation T.
  - In new coordinates, the hole is at $T(C_x, C_y, C_z) = (0, 0, 0)$.
  - If V is a point, $T(V) = V - (C_x, C_y, C_z)$.
  - $T(A) = T(A_x, A_y, A_z) = (A_x – C_x, A_y – C_y, A_z – C_z)$
- $P(A) = (-(A_x-C_x)/(A_z-C_z) * f, -(A_y-C_y)/(A_z-C_z) * f)$.
  - Remember, P(A) is in *normalized image coordinates.*

Handling Camera Translation

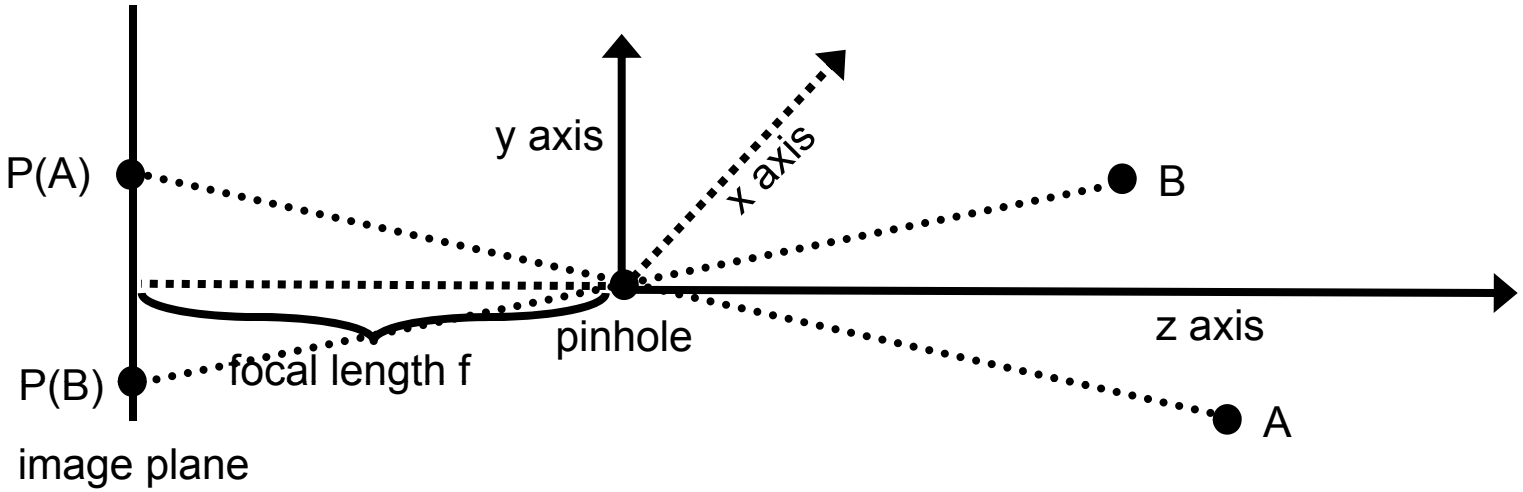


- If the pinhole is at (C_x, C_y, C_z):
  - $P(A) = (-(A_x-C_x)/(A_z-C_z) * f, -(A_y-C_y)/(A_z-C_z) * f)$.
- The concept is simple, but the formulas are messy.
- Formulas get a lot more messy in order to describe arbitrary camera placements.
  - We also need to allow for rotations.
- We simplify notation using *homogeneous coordinates.*

Homogeneous Coordinates

- Homogeneous coordinates are used to simplify formulas, so that camera projection can be modeled as matrix multiplication.
- For a 3D point:
  - instead of writing $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ we write $\begin{pmatrix} cx \\ cy \\ cz \\ c \end{pmatrix}$ where c can be any constant.
  - How many ways are there to write $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ in homogeneous coordinates?
  - INFINITE (one for each real number c).
- For a 2D point $\begin{pmatrix} u \\ v \end{pmatrix}$: we write it as $\begin{pmatrix} cu \\ cv \\ c \end{pmatrix}$.

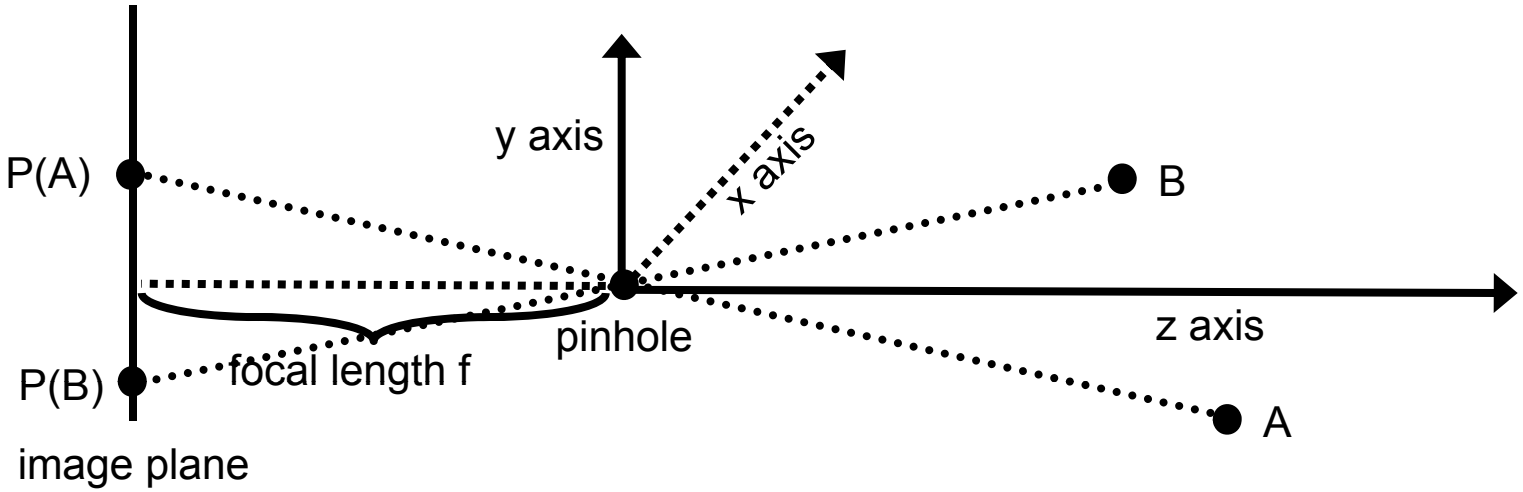
Revisiting Simple Case



- World coordinates = camera coordinates.
- Let A= $\begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, P(A) = $\begin{pmatrix} (-A_x/A_z) * f \\ (-A_y/A_z) * f \\ 1 \end{pmatrix}$. Then:

How do we write P(A) as a matrix multiplication?

Revisiting Simple Case



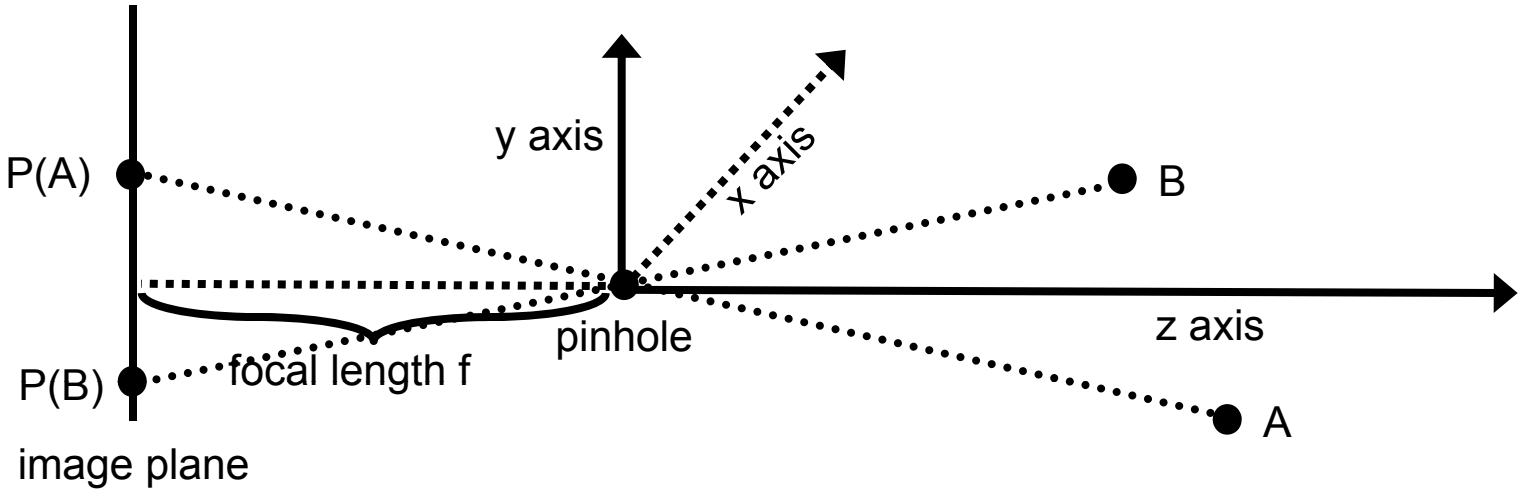
- World coordinates = camera coordinates.
- Let A= $\begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, P(A) = $\begin{pmatrix} (-A_x/A_z) * f \\ (-A_y/A_z) * f \\ 1 \end{pmatrix}$. Then:

$$\begin{pmatrix} (-A_x/A_z) * f \\ (-A_y/A_z) * f \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix} \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$$

Why?

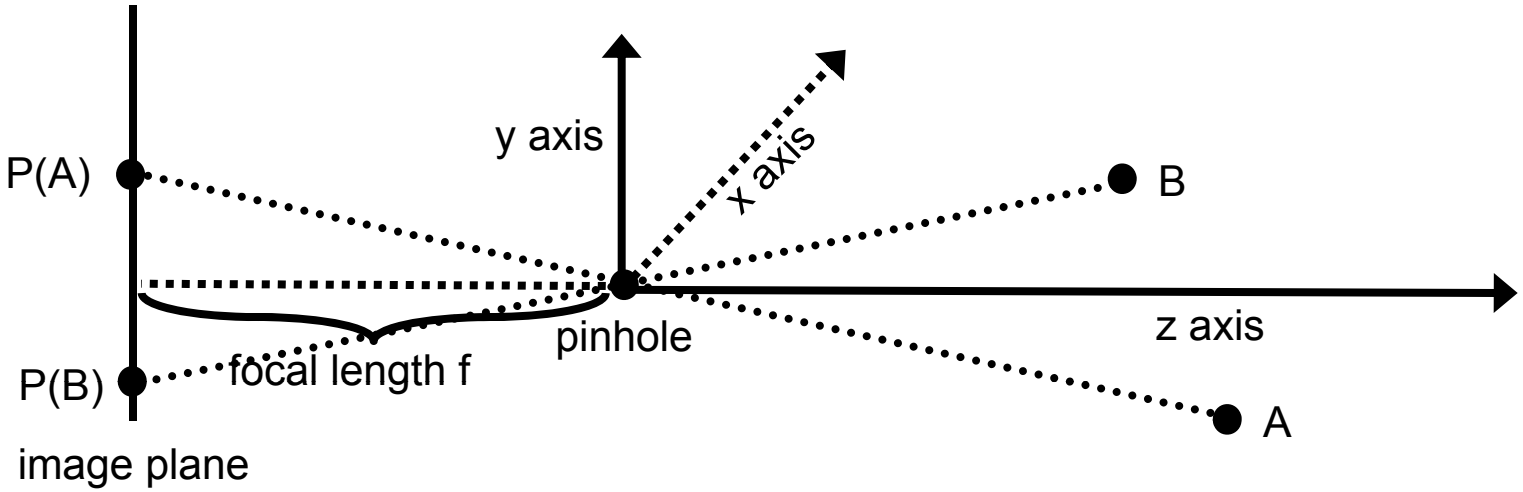
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix} \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix} = \begin{pmatrix} A_x \\ A_y \\ -A_z/f \end{pmatrix} = \begin{pmatrix} (-A_x/A_z) * f \\ (-A_y/A_z) * f \\ 1 \end{pmatrix}$$

Revisiting Simple Case



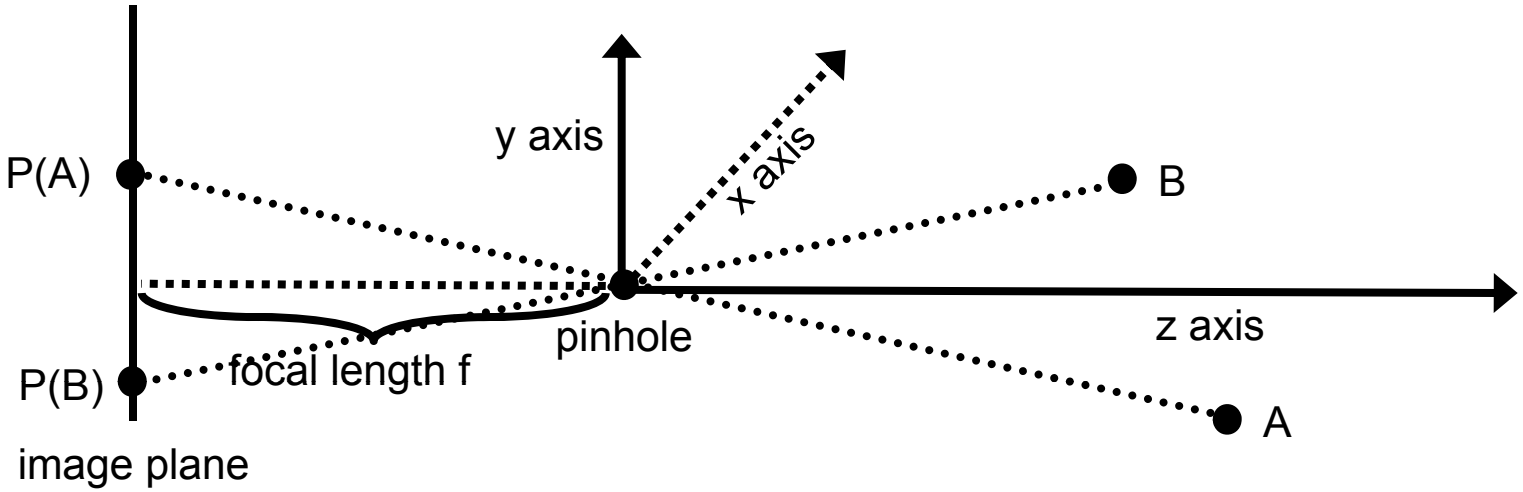
- World coordinates = camera coordinates.
- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$. $P(A) = \begin{pmatrix} (-A_x/A_z) * f \\ (-A_y/A_z) * f \\ 1 \end{pmatrix}$. Define $C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}$.
- Then: $P(A) = C_1 * A$.
  - We map world coordinates to normalized camera coordinates using a simple matrix multiplication.

Handling Camera Translation



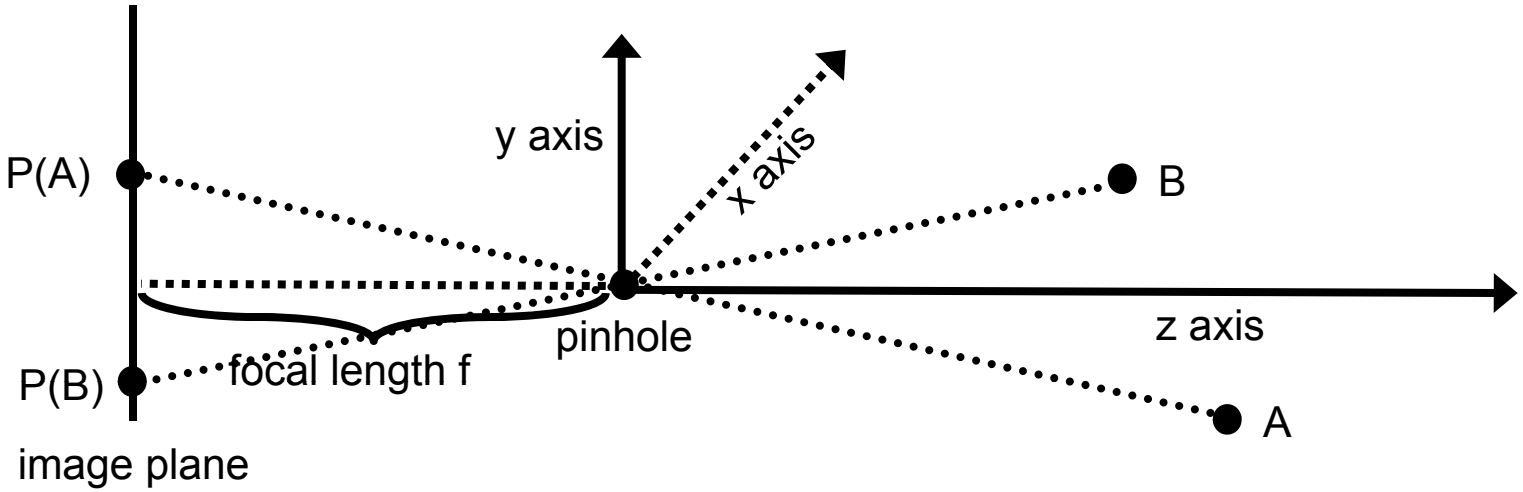
- Suppose camera is at (C_x, C_y, C_z).
  - Camera coordinates and world coordinates are different.
- Define T(A) to be the transformation from world coordinates to camera coordinates.
- If we know T(A), what is P(A)?

Handling Camera Translation



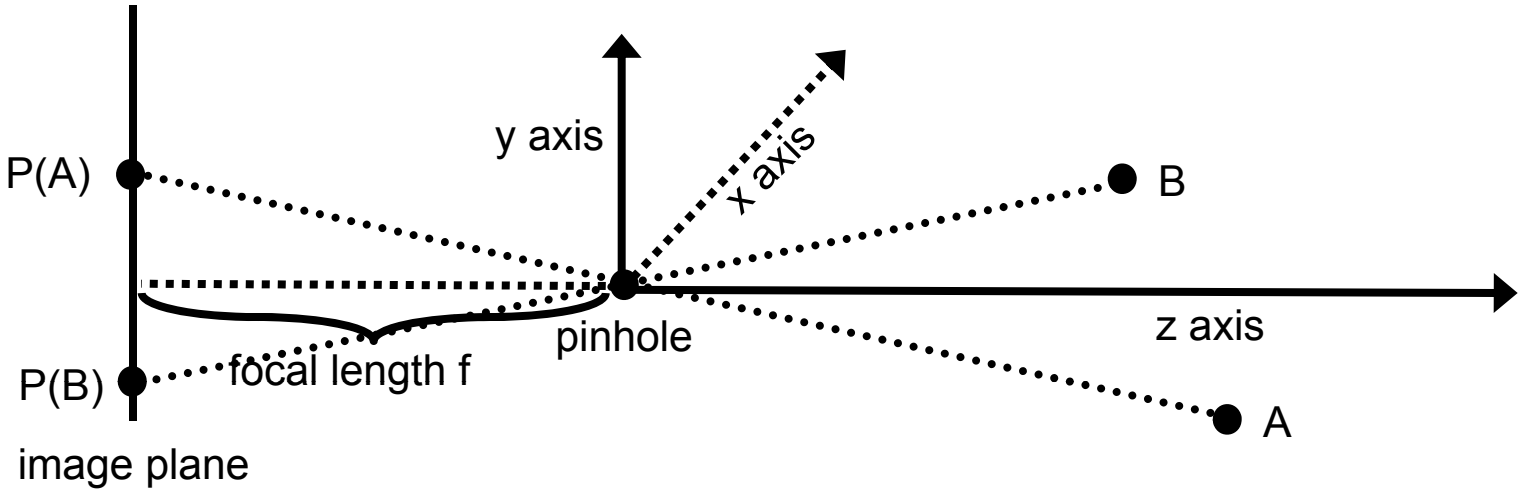
- Suppose camera is at (C_x, C_y, C_z).
  - Camera coordinates and world coordinates are different.
- Define T(A) to be the transformation from world coordinates to camera coordinates.
- If we know T(A), what is P(A)?
- $P(A) = C_1 * T(A)$.

Handling Camera Translation



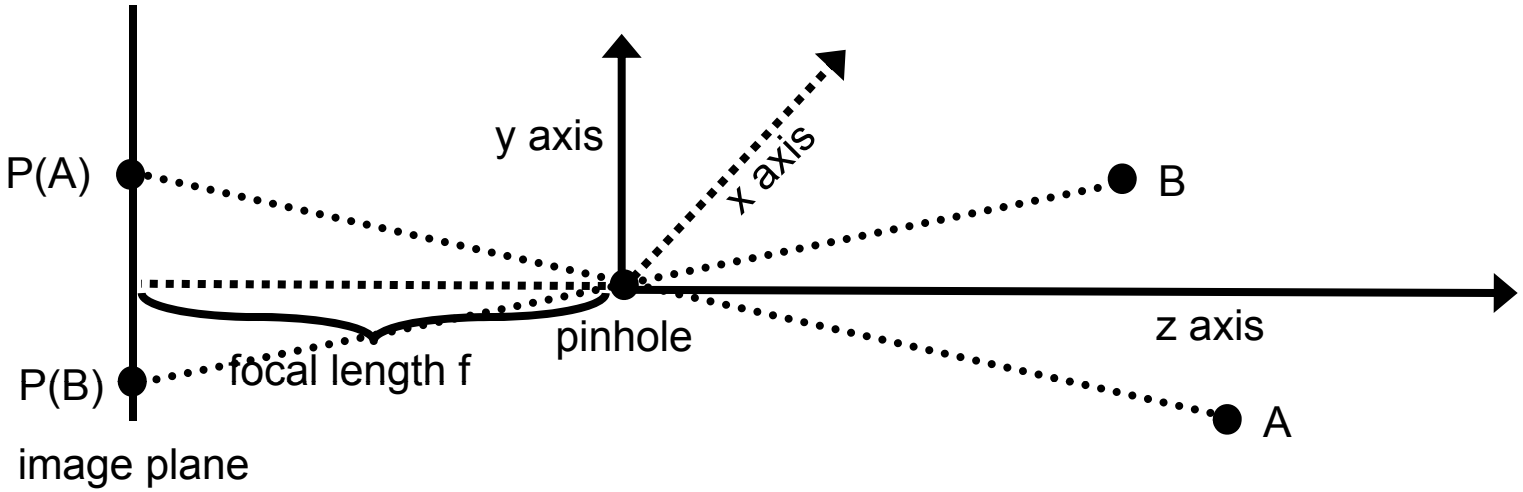
- Suppose camera is at (C_x, C_y, C_z).
- Define T(A) to be the transformation from world coordinates to camera coordinates.
- If we know T(A), $P(A) = C_1 * T(A)$.
- How can we write T(A) as a matrix multiplication?

Handling Camera Translation



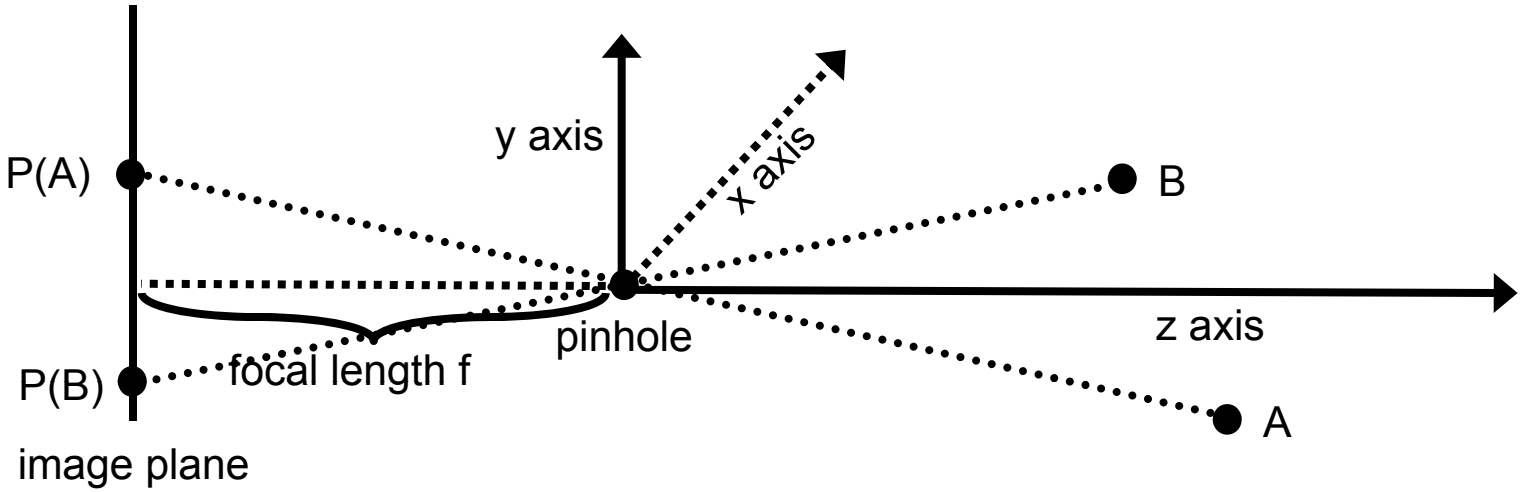
- First of all, how can we write T(A) in the most simple form, in non-homogeneous coordinates? (Forget about matrix multiplication for a second).

Handling Camera Translation



- First of all, how can we write T(A) in the most simple form, in non-homogeneous coordinates?
- $T(A) = (A_x, A_y, A_z) – (C_x, C_y, C_z)$.
- How can we represent that as a matrix multiplication?

Handling Camera Translation

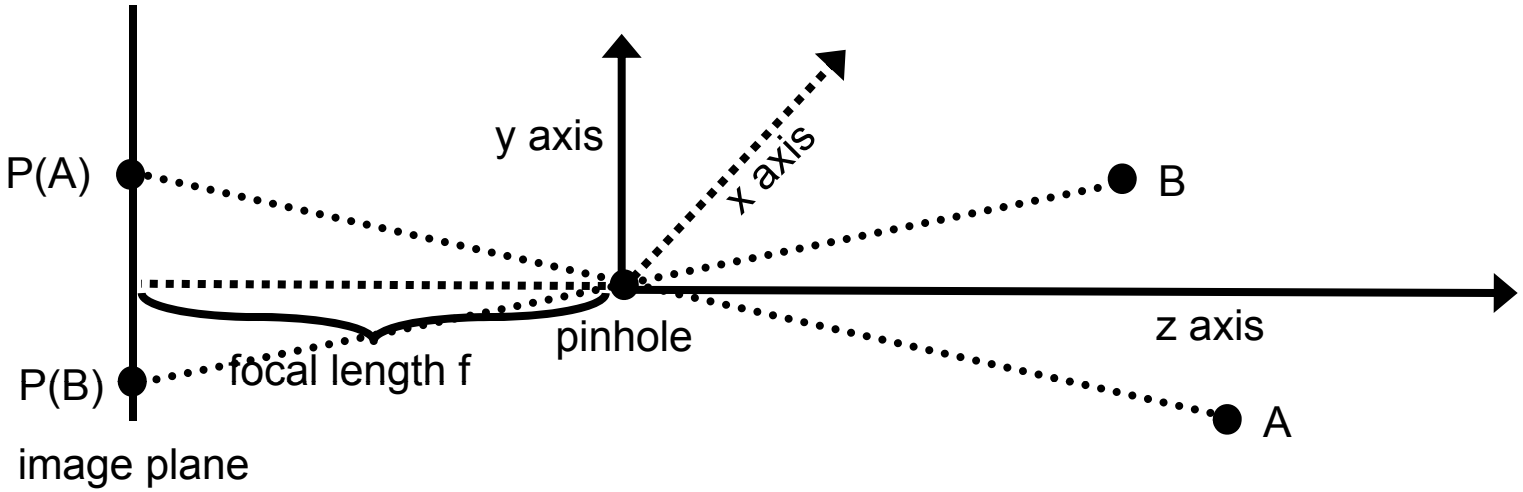


- $T(A) = (A_x, A_y, A_z) – (C_x, C_y, C_z)$.
- In homogeneous coordinates:

$$\begin{pmatrix} A_x - C_x \\ A_y - C_y \\ A_z - C_z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$$

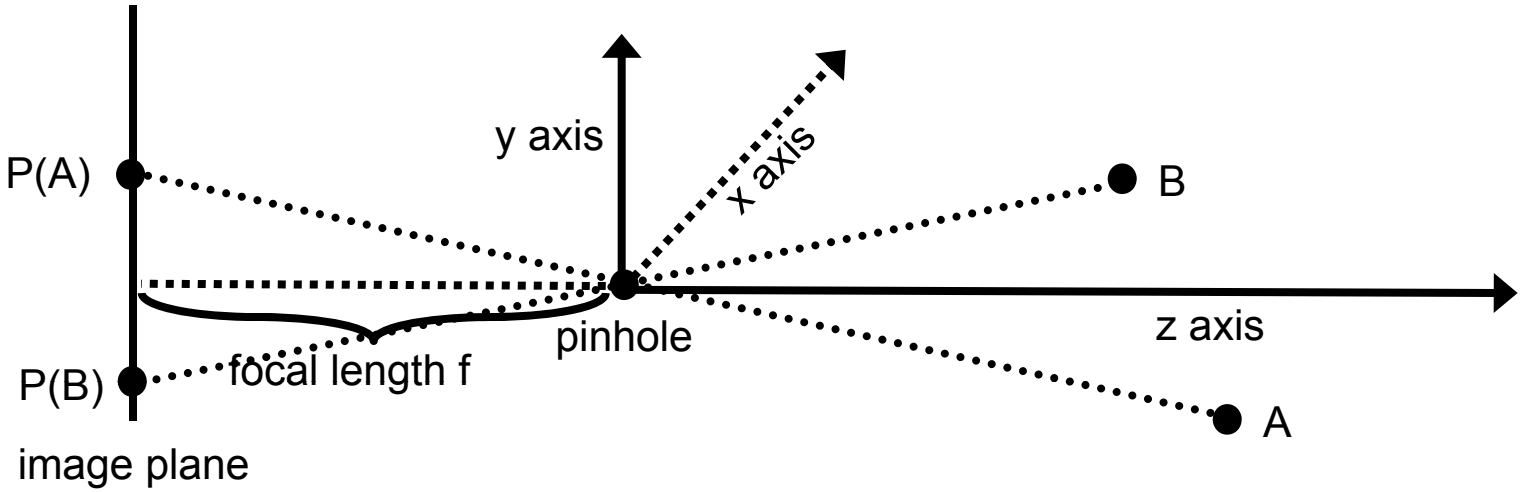
- Homogeneous coordinates allow us to represent translation as matrix multiplication.

Handling Camera Translation



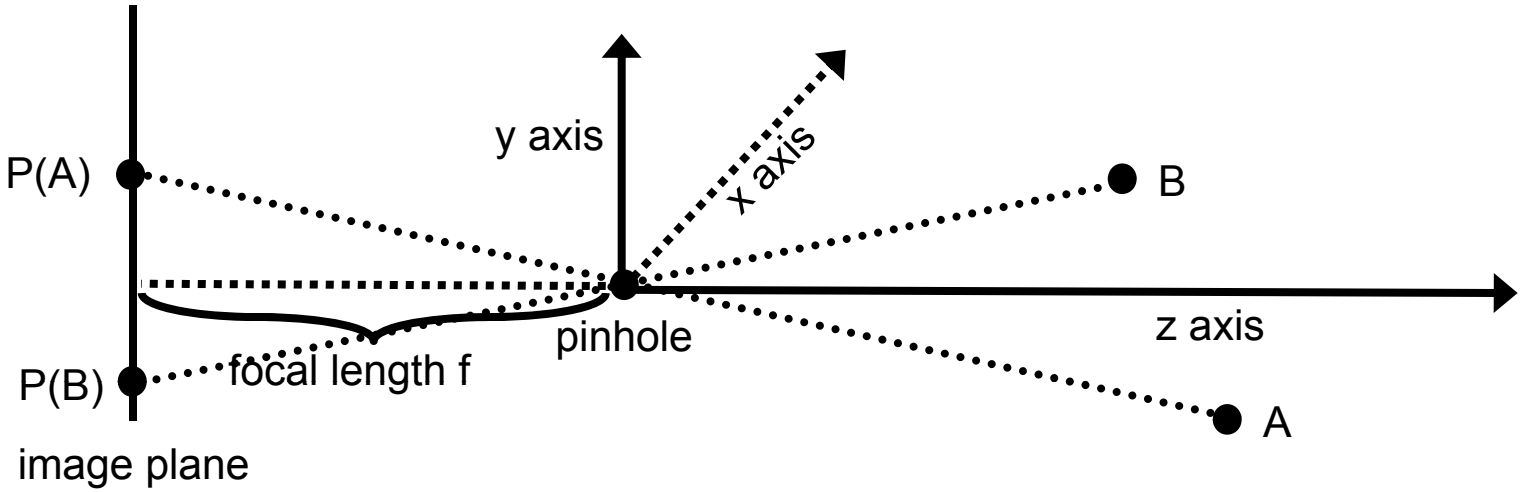
- Let A= $\begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$. Define $C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}$, T = $\begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- Then: P(A) = C_1 * T * A.
- P(A) is still a matrix multiplication:
  - We multiply A by (C_1 * T).

Handling Camera Translation



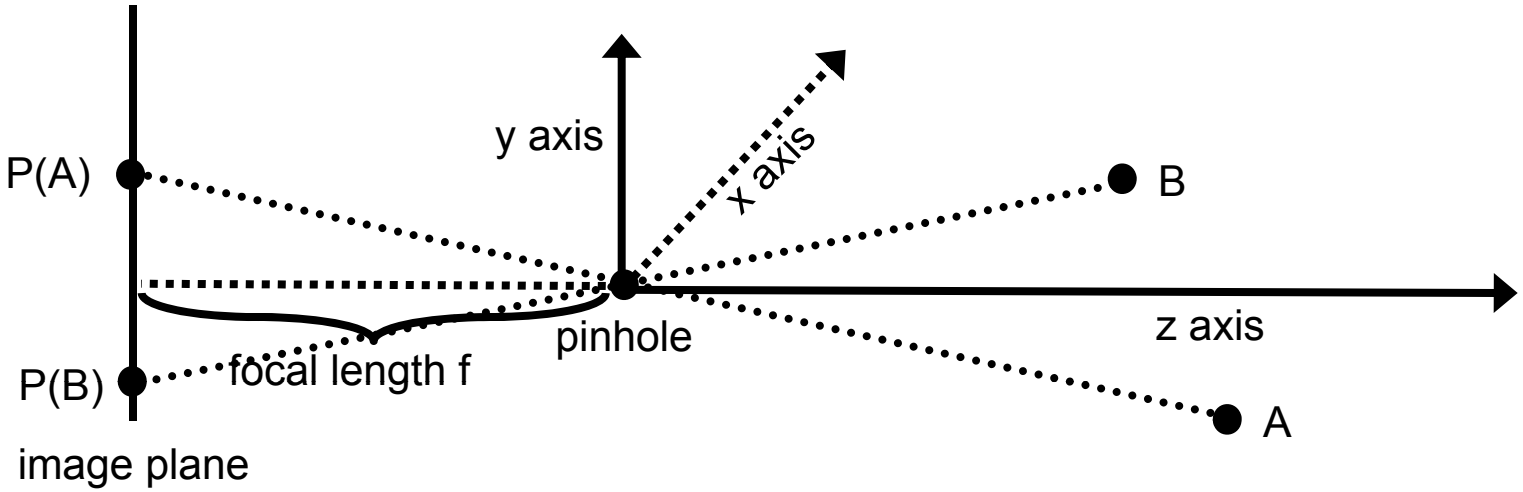
- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$. Define $C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}$, $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- Then: $P(A) = C_1 * T * A$.
- Why is C_1 3x4 and T 4x4?

Handling Camera Translation



- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$. Define $C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}$, $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$.
- Then: $P(A) = C_1 * T * A$.
- Why is C_1 3x4 and T 4x4?
  - T maps 3D coordinates to 3D coordinates.
  - C1 maps 3D coordinates to normalized image coordinates.

Handling Camera Rotation



- The camera can be rotated around the x axis, around the y axis, and/or around the z axis.
- Rotation transformation R: rotates the world coordinates, so that the x, y, and z axis of the world coordinate system match the x, y, and z axis of the camera coordinate system.

Handling Camera Rotation

- In non-homogeneous coordinates, rotation of A around the origin can be represented as R*A.
  - R: 3x3 rotation matrix.
- How does camera rotation affect the image?

Handling Camera Rotation

- In non-homogeneous coordinates, rotation of A around the origin can be represented as R*A.
  - R: 3x3 rotation matrix.
- How does camera rotation affect the image?
  - It changes the viewing direction.
    - Determines what is visible.
  - It changes the image orientation.
    - Determines what the “up” direction in the image corresponds to in the 3D world.
- Rotating the camera by R_c has the same affect as rotating the world by the inverse of R_c.
  - That is, rotating every point in the world, around the origin, the opposite way of what is specified in R_c.

Handling Camera Rotation

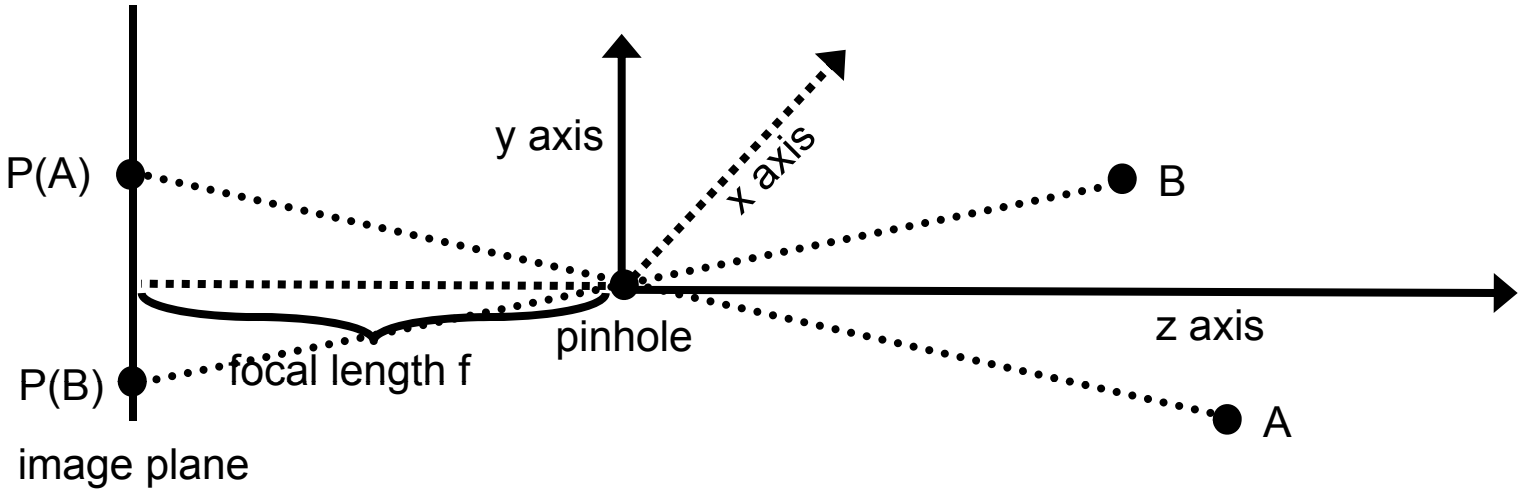
- Any rotation R can be decomposed into three rotations:
  - a rotation R_x by θ_x around the x axis.
  - a rotation R_y by θ_y around the y axis.
  - a rotation R_z by θ_z around the z axis.
- Rotation of point A = R * A = $R_z * R_y * R_x * A$.
- ORDER MATTERS.
  - $R_z * R_y * R_x * A$ is not the same as $R_x * R_y * R_z * A$.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{pmatrix} \quad R_x$$

$$\begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{pmatrix} \quad R_y$$

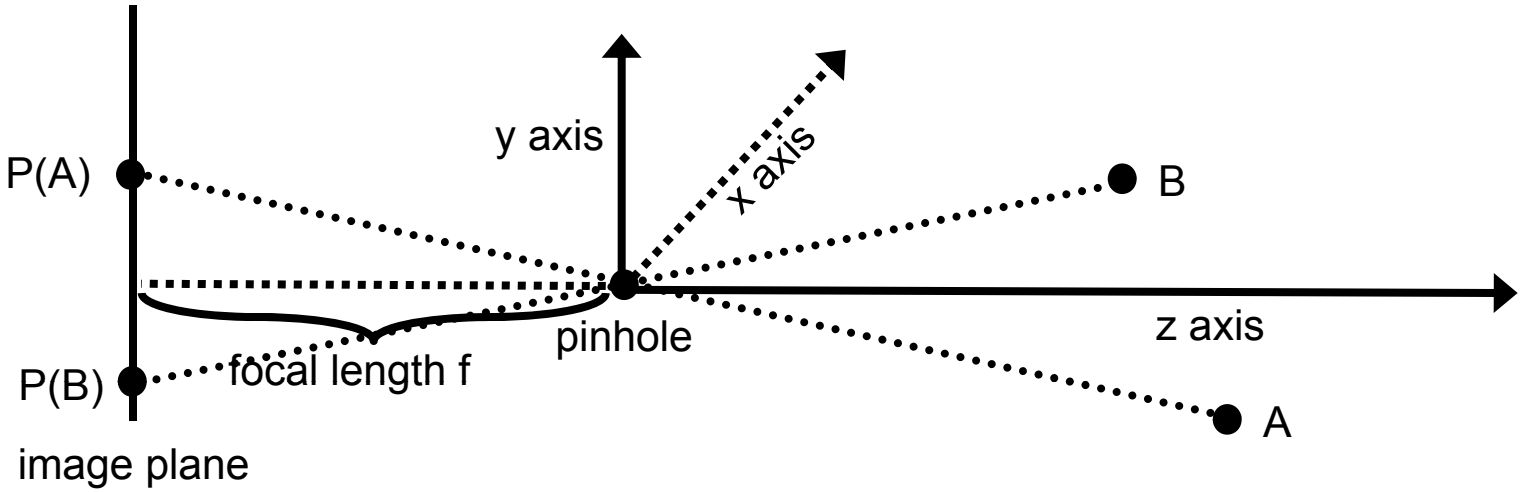
$$\begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad R_z$$

Handling Camera Rotation



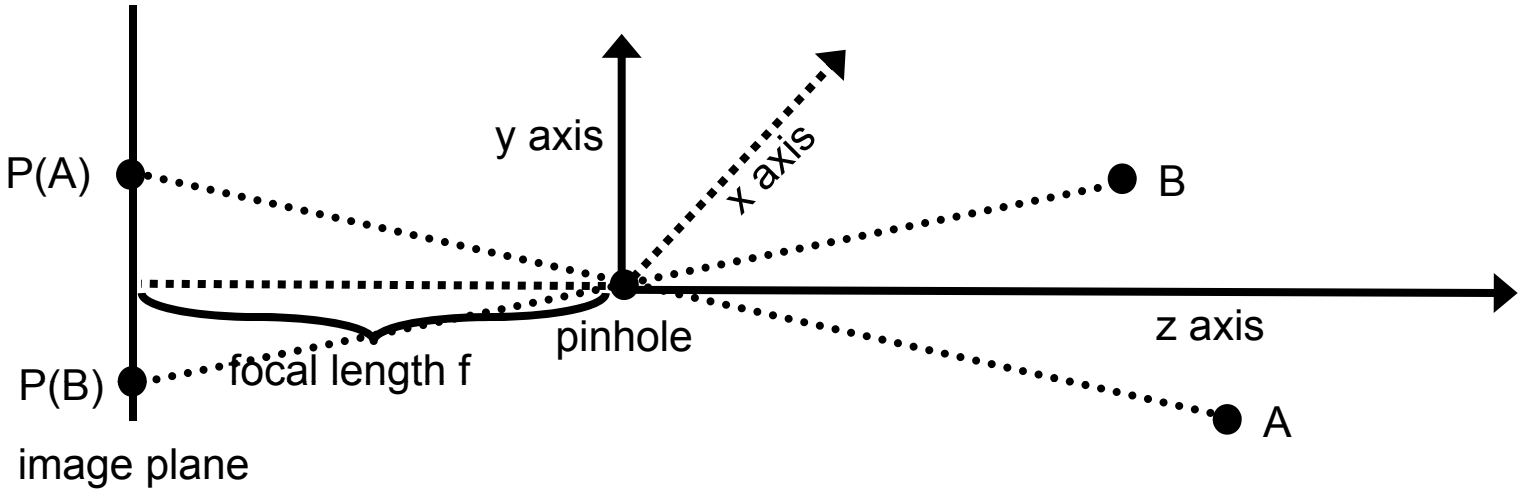
- In homogeneous coordinates, rotation of A around the origin can be represented as R*A.
  - R: 4x4 rotation matrix.
- Let R' = $\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$. Then, R = $\begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.

Handling Camera Rotation



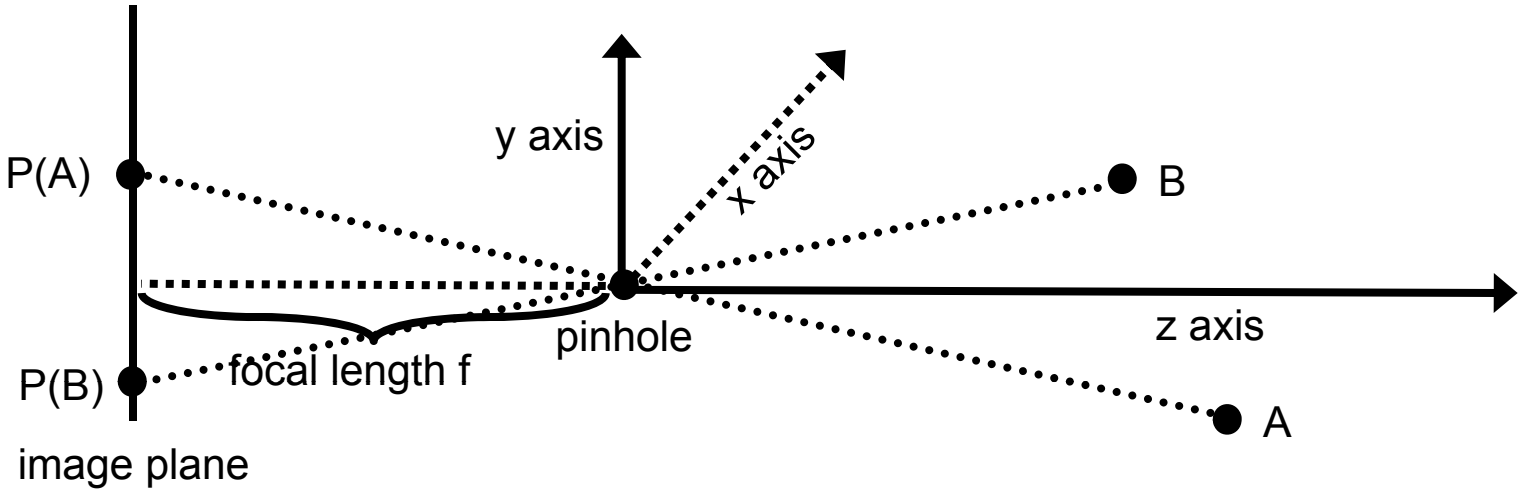
- Let R' = $\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$. Then, R = $\begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.
- What is the right way to write P(A) so that we include translation and rotation?

Handling Camera Rotation



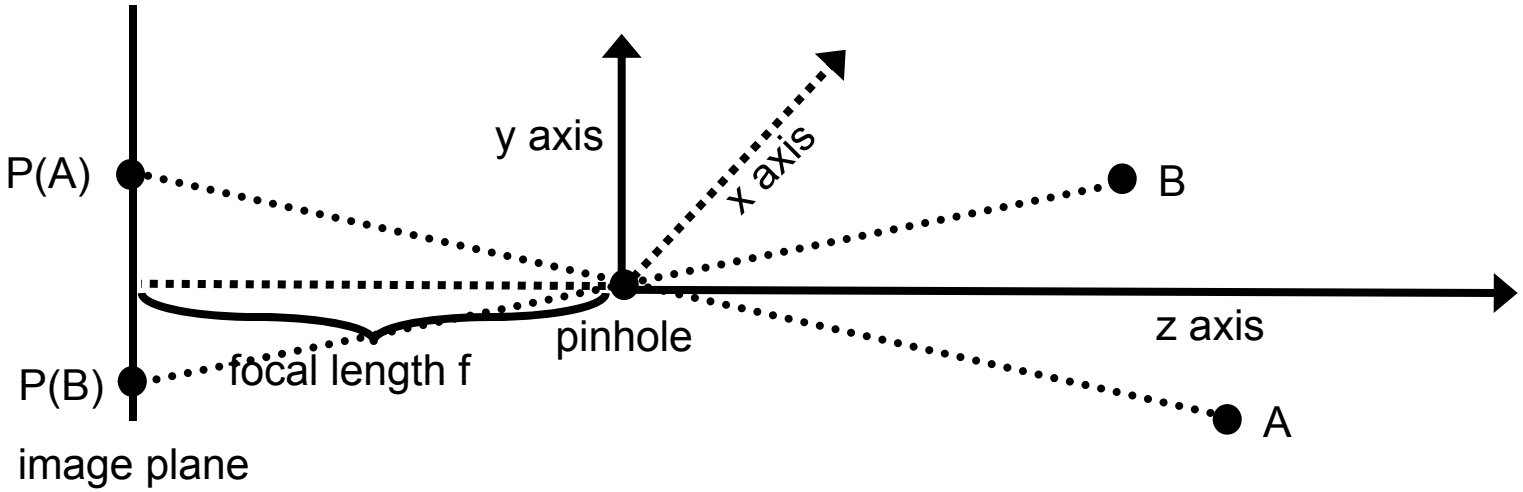
- Let R' = $\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$. Then, R = $\begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- What is the right way to right P(A) so that we include translation and rotation?
- Would it be P(A) = C_1 * T * R *A?

Handling Camera Rotation



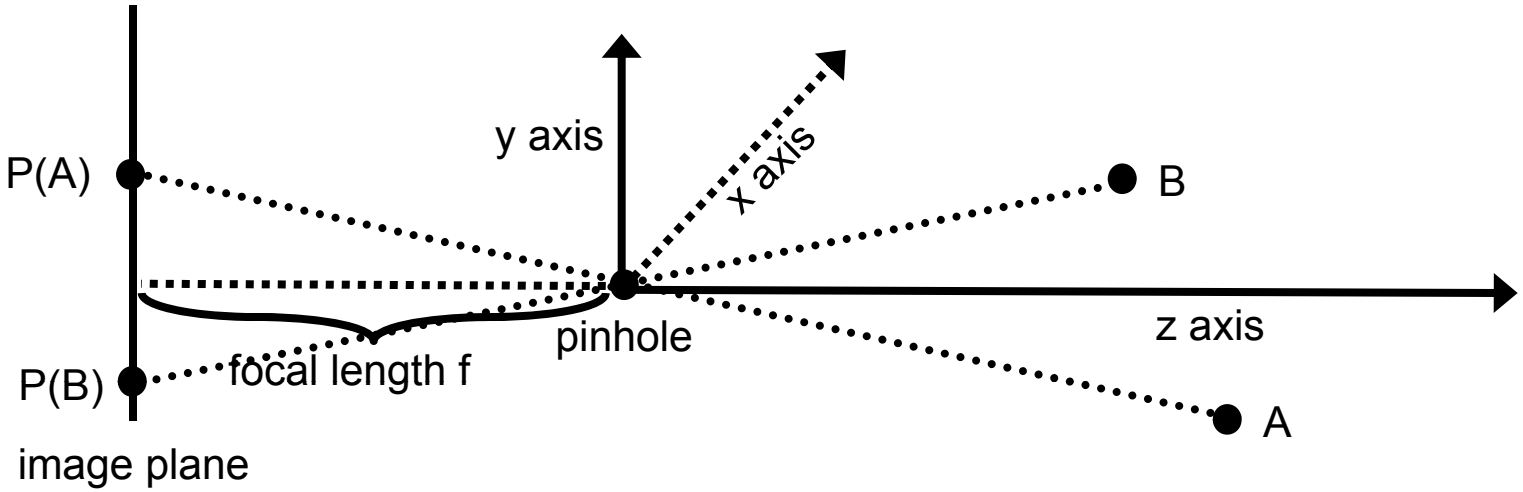
- Let R' = $\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$. Then, R = $\begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- Would it be P(A) = C_1 * T * R *A?
  - NO, we must first translate and then rotate.
  - Why?

Handling Camera Rotation



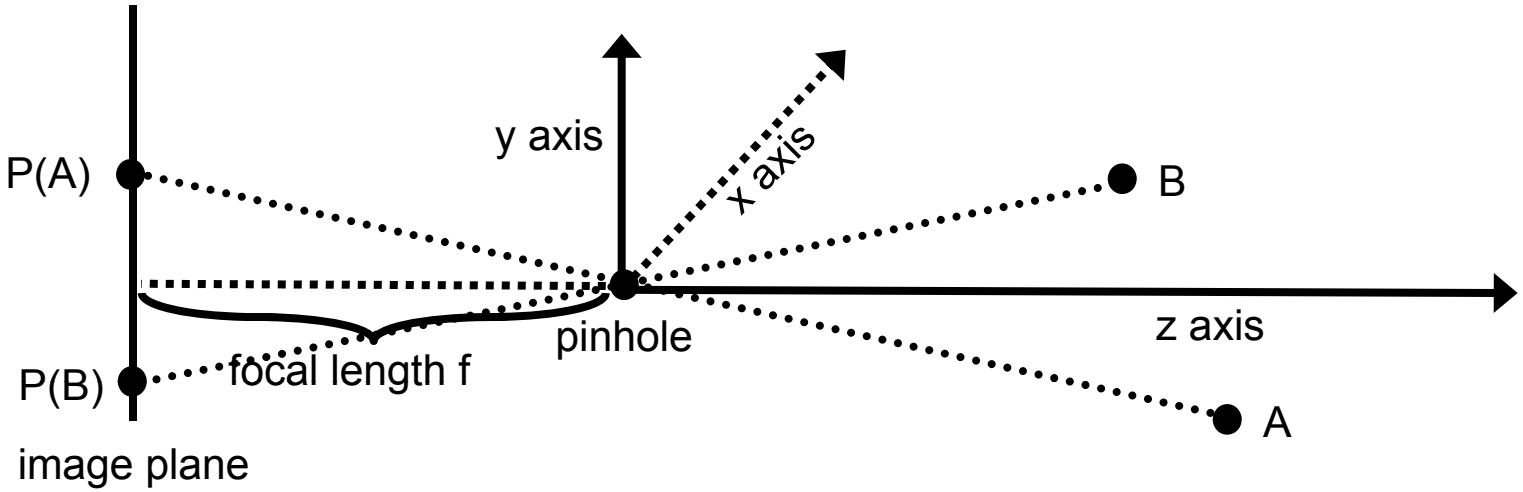
- Let R' = $\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$. Then, R = $\begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- Would it be P(A) = C_1 * T * R *A?
  - NO, we must first translate and then rotate.
  - Rotation is always around the origin. First we must apply T to move the pinhole to the origin, and then we can apply R.

Handling Camera Rotation



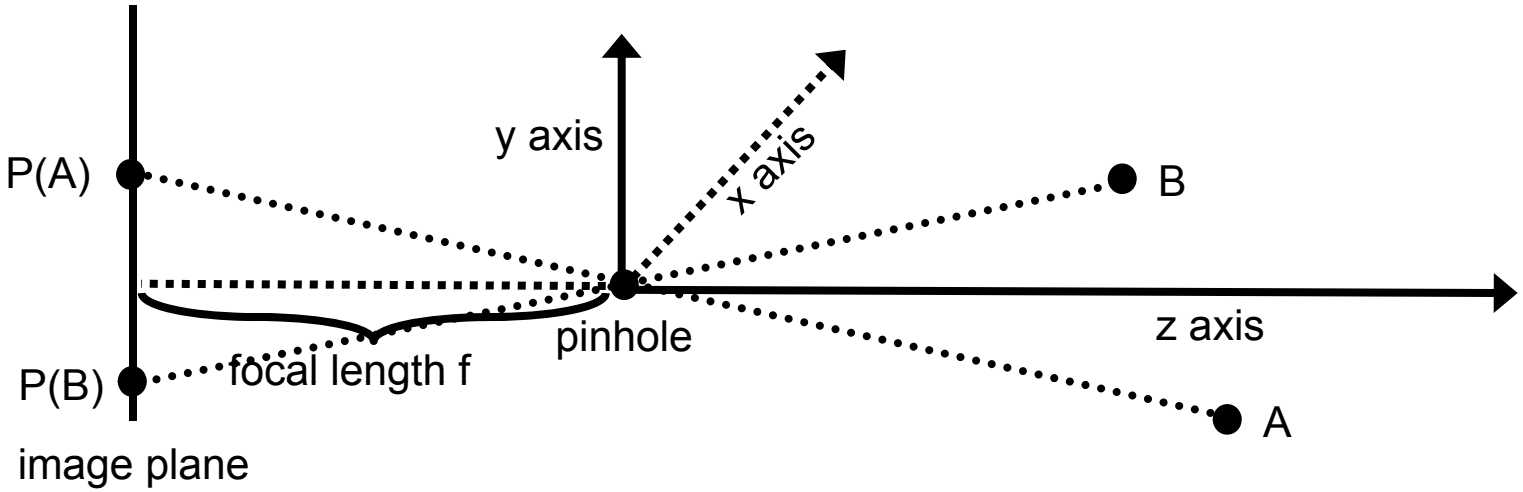
- Let R' = $\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$. Then, R = $\begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- $P(A) = C_1 * R * T * A$.
- P(A) is *still* modeled as matrix multiplication.
  - We multiply A with matrix ($C_1 * R * T$).

Handling Scale



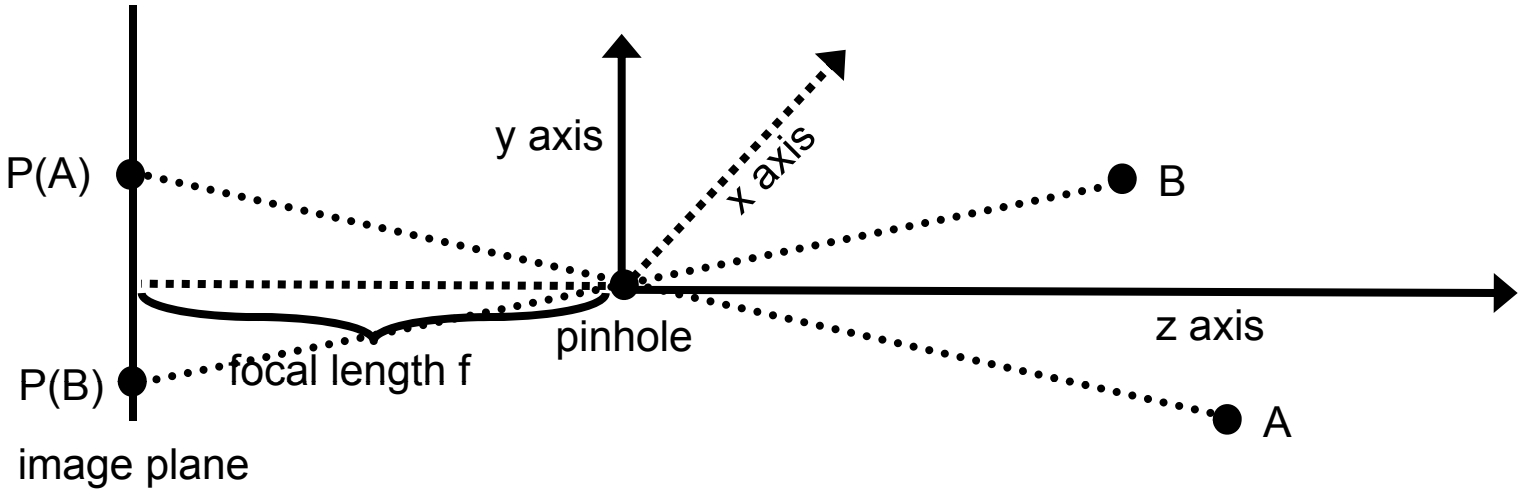
- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}$, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  - $P(A) = C_1 * R * T * A$ accounts for translation and rotation.
  - Translation: moving the camera.
  - Rotation: rotating the camera.
  - Scaling: what does it correspond to?

Handling Scale



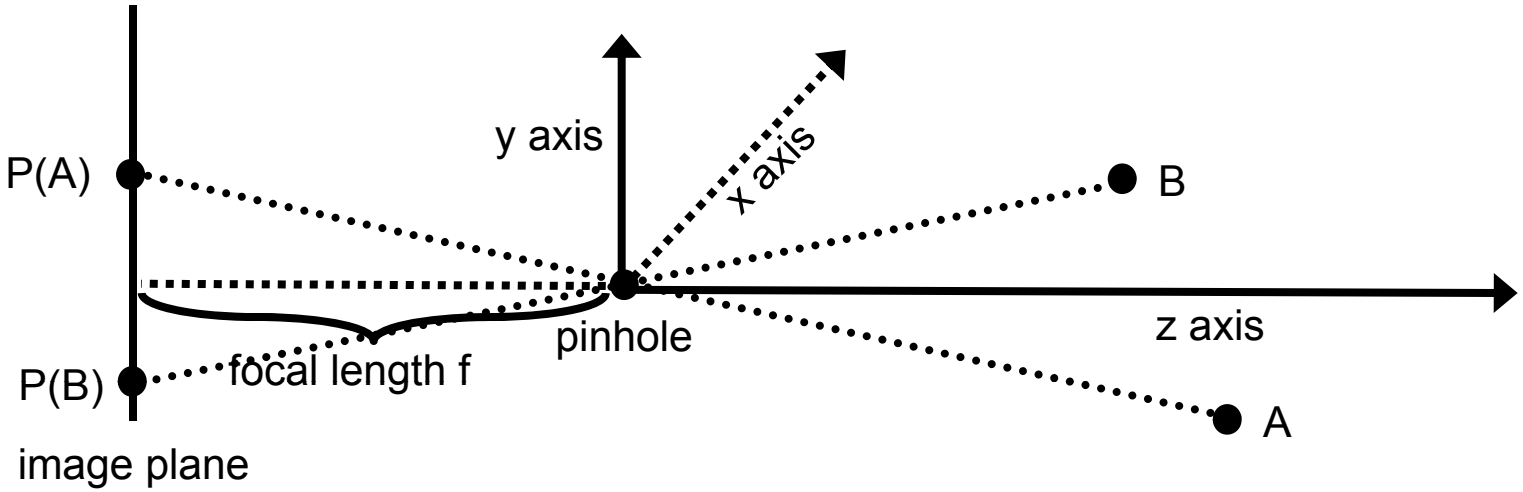
- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}$, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  - $P(A) = C_1 * R * T * A$ accounts for translation and rotation.
  - Translation: moving the camera.
  - Rotation: rotating the camera.
  - Scaling: corresponds to zooming (changing focal length).

Handling Scale



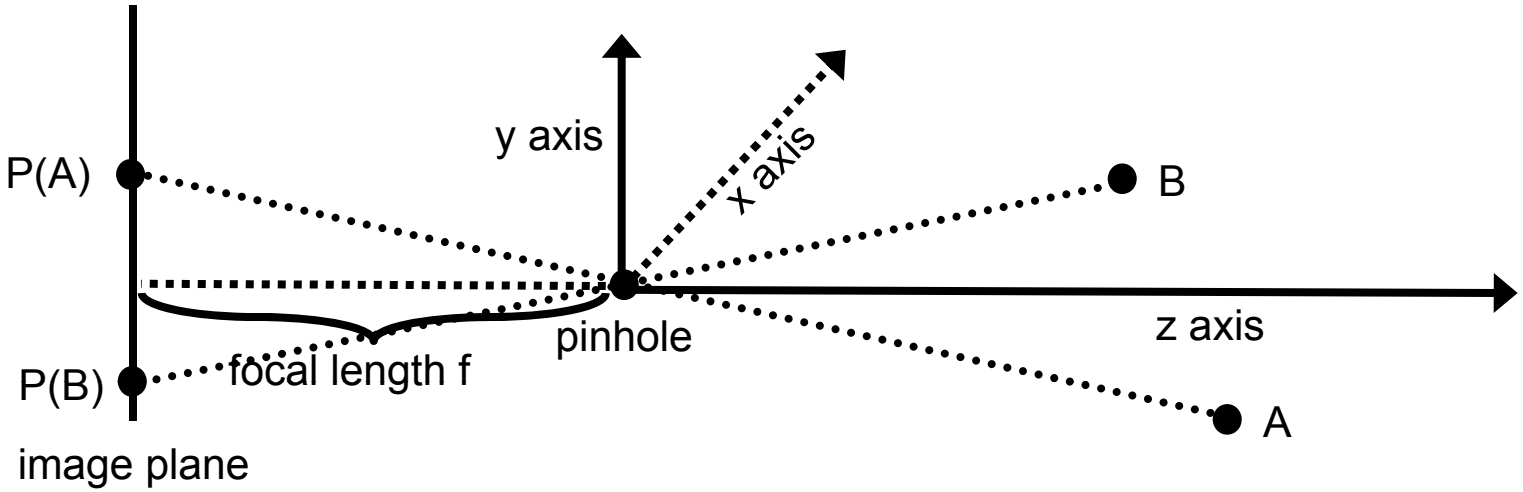
- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}$, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  - $P(A) = C_1 * R * T * A$ accounts for translation and rotation.
  - Translation: moving the camera.
  - Rotation: rotating the camera.
  - How do we model scaling?

Handling Scale



- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}$, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  - $P(A) = C_1 * R * T * A$ accounts for translation and rotation.
- How do we model scaling?
  - Scaling is already handled by parameter f in matrix C_1.
  - If we change the focal length we must update f.

World to Normalized Image Coords



- Let A= $\begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}$, R = $\begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ T = $\begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  - P(A) = C_1 * R * T * A maps world coordinates to normalized image coordinates
  - Equation holds for any camera following the pinhole camera model.

Computing Pixel Coordinates

- The normalized image coordinate system does not produce pixel coordinates.
  - Example: the center of the image is at (0, 0).
- What is needed to map normalized image coordinates to pixel coordinates?
  - Translation?
  - Scaling?
  - Rotation?

Computing Pixel Coordinates

- The normalized image coordinate system does not produce pixel coordinates.
  - Example: the center of the image is at (0, 0).
- What is needed to map normalized image coordinates to pixel coordinates?
  - Translation? Yes, we must move center of image to (image_columns/2, image_rows/2).
  - Scaling?
  - Rotation?

Computing Pixel Coordinates

- The normalized image coordinate system does not produce pixel coordinates.
  - Example: the center of the image is at (0, 0).
- What is needed to map normalized image coordinates to pixel coordinates?
  - Translation? Yes, we must move center of image to (image_columns/2, image_rows/2).
  - Scaling? Yes, according to pixel size (how much area of the image plane does a pixel correspond to?).
    - In the general case, two constants, S_x and S_y, if the pixel corresponds to a non-square rectangle on the image plane.
    - In the typical case, S_x and S_y.
  - Rotation?

Computing Pixel Coordinates

- The normalized image coordinate system does not produce pixel coordinates.
  - Example: the center of the image is at (0, 0).
- What is needed to map normalized image coordinates to pixel coordinates?
  - Translation? Yes, we must move center of image to (image_columns/2, image_rows/2).
  - Scaling? Yes, according to pixel size.
    - In the general case, two constants, S_x and S_y, if the pixel corresponds to a non-square rectangle on the image plane.
    - In the typical case, S_x and S_y.
  - Rotation? NO.
    - The x and y axes of the two systems match.

Homography

- The matrix mapping normalized image coordinates to pixel coordinates is called a *homography*.
- A homography matrix H looks like this:

$$H = \begin{pmatrix} S_x & 0 & u_0 \\ 0 & S_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}$$

where:

  - S_x and S_y define scaling (typically $S_x = S_y$).
  - u_0 and v_0 translate the image so that its center moves from (0, 0) to (u_0, v_0).

Putting It All Together

- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$.
- What pixel coordinates (u, v) will A be mapped to?

$$C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}, \quad R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad H = \begin{pmatrix} S_x & 0 & u_0 \\ 0 & S_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Putting It All Together

- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$.
- What pixel coordinates (u, v) will A be mapped to?

$$C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}, \quad R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad H = \begin{pmatrix} S_x & 0 & u_0 \\ 0 & S_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}.$$

- $\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = H * C_1 * R * T * A.$
- $u = u'/w'$, $v = v'/w'$.

An Alternative Formula

- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $H = \begin{pmatrix} -fS_x & 0 & u_0 & 0 \\ 0 & -fS_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.
  - C_1 was only useful for storing f, but we can store f in H.

- What pixel coordinates (u, v) will A be mapped to?
- $\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = H * R * T * A.$
- u = u'/w', v = v'/w'.
- (H * R * T) is called the *camera* matrix.
  - What size is it? What does it map to what?

Calibration Matrix

- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $H = \begin{pmatrix} -fS_x & 0 & u_0 & 0 \\ 0 & -fS_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.
  - C_1 was only useful for storing f, but we can store f in H.
- What pixel coordinates (u, v) will A be mapped to?
- $\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = H * R * T * A.$
- u = u'/w', v = v'/w'.
- H is called the *calibration* matrix.
  - It does not change if we rotate/move the camera.

Orthographic Projection

- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $H = \begin{pmatrix} S_x & 0 & 0 & u_0 \\ 0 & S_y & 0 & v_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.
- What pixel coordinates (u, v) will A be mapped to?
- $\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = H * R * T * A.$
- $u = u'/w'$, $v = v'/w'$.
- Main difference from perspective projection: z coordinate gets ignored.
  - To go from camera coordinates to normalized image coordinates, we just drop the z value.

Part 2

Calibration

Calibration

- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $H = \begin{pmatrix} -fS_x & 0 & u_0 & 0 \\ 0 & -fS_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.
- $\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = H * R * T * A.$
- C = (H * R * T) is called the *camera* matrix.
- Question: How do we compute C?
- The process of computing C is called *camera calibration*.

Calibration

- Camera matrix C is always of the following form:

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & 1 \end{pmatrix}$$

- C is equivalent to any sC, where s != 0.
  - Why?

Calibration

- Camera matrix C is always of the following form:

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & 1 \end{pmatrix}$$

- C is equivalent to any sC, where s != 0.
  - That is why we can assume that $c_{34} = 1$. If not, we can just multiply by $s = 1/c_{34}$.
- To compute C, one way is to manually establish correspondences between points in 3D world coordinates and pixels in the image.

Using Correspondences

- Suppose that $[x_j, y_j, z_j, 1]$ maps to $[u_j, v_j, 1]$.
- This means that $C * [x_j, y_j, z_j, 1]' = [s_j u_j, s_j v_j, s_j]'$.
  - Note that vectors $[x_j, y_j, z_j, 1]$ and $[s_j u_j, s_j v_j, s_j]$ are transposed.
- This gives the following equations:
  1. $s_j u_j = c_{11} * x_j + c_{12} * y_j + c_{13} * z_j + c_{14}$.
  2. $s_j v_j = c_{21} * x_j + c_{22} * y_j + c_{23} * z_j + c_{24}$.
  3. $s_j = c_{31} * x_j + c_{32} * y_j + c_{33} * z_j + 1$.
- Multiplying Equation 3 by u_j we get:
  - $s_j u_j = c_{31} * u_j * x_j + c_{32} * u_j * y_j + c_{33} * u_j * z_j + u_j$.
- Multiplying Equation 3 by v_j we get:
  - $s_j v_j = c_{31} * v_j * x_j + c_{32} * v_j * y_j + c_{33} * v_j * z_j + v_j$.

Obtaining a Linear Equation

- We combine two equations:
  - $s_j u_j = c_{11} * x_j + c_{12} * y_j + c_{13} * z_j + c_{14}$.
  - $s_j u_j = c_{31} * u_j * x_j + c_{32} * u_j * y_j + c_{33} * u_j * z_j + u_j$.

  to obtain:

  $c_{11}x_j+c_{12}y_j+c_{13}z_j+c_{14} = c_{31}u_jx_j+c_{32}u_jy_j+c_{33}u_jz_j+u_j \Rightarrow$

  $u_j = c_{11}x_j+c_{12}y_j+c_{13}z_j+c_{14} - c_{31}u_jx_j-c_{32}u_jy_j-c_{33}u_jz_j \Rightarrow$

  $u_j = [x_j,y_j,z_j,1,-u_jx_j,-u_jy_j,-u_jz_j]*[c_{11},c_{12},c_{13},c_{14}, c_{31},c_{32}, c_{33}]^{trans} \Rightarrow$

  $u_j = [x_j, y_j, z_j, 1, 0, 0, 0, 0, -u_jx_j, -u_jy_j, -u_jz_j] *$
  $[c_{11}, c_{12}, c_{13}, c_{14}, c_{21}, c_{22}, c_{23}, c_{24}, c_{31}, c_{32}, c_{33}]^{trans}$
- In the above equations:
  - What is known, what is unknown?

Obtaining a Linear Equation

- We combine two equations:
  - $s_j u_j = c_{11} * x_j + c_{12} * y_j + c_{13} * z_j + c_{14}$.
  - $s_j u_j = c_{31} * u_j * x_j + c_{32} * u_j * y_j + c_{33} * u_j * z_j + u_j$.

  to obtain:

$c_{11}x_j+c_{12}y_j+c_{13}z_j+c_{14} = c_{31}u_jx_j+c_{32}u_jy_j+c_{33}u_jz_j+u_j \Rightarrow$

$u_j = c_{11}x_j+c_{12}y_j+c_{13}z_j+c_{14} - c_{31}u_jx_j-c_{32}u_jy_j-c_{33}u_jz_j \Rightarrow$

$u_j = [x_j,y_j,z_j,1,-u_jx_j,-u_jy_j,-u_jz_j]*[c_{11},c_{12},c_{13},c_{14}, c_{31},c_{32}, c_{33}]^{trans} \Rightarrow$

$u_j = [x_j, y_j, z_j, 1, 0, 0, 0, 0, -u_jx_j, -u_jy_j, -u_jz_j] *$
$[c_{11}, c_{12}, c_{13}, c_{14}, c_{21}, c_{22}, c_{23}, c_{24}, c_{31}, c_{32}, c_{33}]^{trans}$

- In the above equations:
  - $c_{11},c_{12},c_{13},c_{14},c_{21},c_{22},c_{23},c_{24},c_{31}, c_{32}, c_{33}$ are unknown.
  - x_j, y_j, z_j, u_j, v_j are assumed to be known.

Obtaining Another Linear Equation

- We combine two equations:
  - $s_j v_j = c_{21} * x_j + c_{22} * y_j + c_{23} * z_j + c_{24}$.
  - $s_j v_j = c_{31} * v_j * x_j + c_{32} * v_j * y_j + c_{33} * v_j * z_j + v_j$.

  to obtain:

  $c_{21}x_j+c_{22}y_j+c_{23}z_j+c_{24} = c_{31}v_jx_j+c_{32}v_jy_j+c_{33}v_jz_j+v_j \Rightarrow$

  $v_j = c_{21}x_j+c_{22}y_j+c_{23}z_j+c_{24} - c_{31}v_jx_j-c_{32}v_jy_j-c_{33}v_jz_j \Rightarrow$

  $v_j = [x_j,y_j,z_j,1,-v_jx_j,-v_jy_j,-v_jz_j]*[c_{21},c_{22},c_{23},c_{24}, c_{31},c_{32}, c_{33}]^{trans} \Rightarrow$

  $v_j = [\ 0,\ \ 0,\ \ 0,\ \ 0,\ \ x_j,\ \ y_j,\ \ z_j,\ \ 1, -v_jx_j, -v_jy_j, -v_jz_j] *$
  $[c_{11}, c_{12}, c_{13}, c_{14}, c_{21}, c_{22}, c_{23}, c_{24}, c_{31}, c_{32}, c_{33}]^{trans}$
- In the above equations:
  - What is known, what is unknown?

Obtaining Another Linear Equation

- We combine two equations:
  - $s_j v_j = c_{21} * x_j + c_{22} * y_j + c_{23} * z_j + c_{24}$.
  - $s_j v_j = c_{31} * v_j * x_j + c_{32} * v_j * y_j + c_{33} * v_j * z_j + v_j$.

  to obtain:

  $c_{21}x_j+c_{22}y_j+c_{23}z_j+c_{24} = c_{31}v_jx_j+c_{32}v_jy_j+c_{33}v_jz_j+v_j \Rightarrow$

  $v_j = c_{21}x_j+c_{22}y_j+c_{23}z_j+c_{24} - c_{31}v_jx_j-c_{32}v_jy_j-c_{33}v_jz_j \Rightarrow$

  $v_j = [x_j,y_j,z_j,1,-v_jx_j,-v_jy_j,-v_jz_j]*[c_{21},c_{22},c_{23},c_{24}, c_{31},c_{32}, c_{33}]^{trans} \Rightarrow$

  $v_j = [\ 0,\ \ 0,\ \ 0,\ \ 0,\ \ x_j,\ \ y_j,\ \ z_j,\ \ 1, -v_jx_j, -v_jy_j, -v_jz_j] *$
  $[c_{11}, c_{12}, c_{13}, c_{14}, c_{21}, c_{22}, c_{23}, c_{24}, c_{31}, \ c_{32}, \ c_{33}]^{trans}$
- In the above equations:
  - $c_{11},c_{12},c_{13},c_{14},c_{21},c_{22},c_{23},c_{24},c_{31}, c_{32}, c_{33}$ are unknown.
  - x_j, y_j, z_j, u_j, v_j are assumed to be known.

Setting Up Linear Equations

- Let $A = \begin{pmatrix} x_j, y_j, z_j, 1, 0, 0, 0, 0, -x_j u_j, -y_j u_j, -z_j u_j \\ 0, 0, 0, 0, x_j, y_j, z_j, 1, -x_j v_j, -y_j v_j, -z_j v_j \end{pmatrix}$
- Let $x = [c_{11}, c_{12}, c_{13}, c_{14}, c_{21}, c_{22}, c_{23}, c_{24}, c_{31}, c_{32}, c_{33}]'$.
  - Note the transpose.
- Let $b = [u_j, v_j]'$.
  - Again, note the transpose.
- Then, A*x = b.
- This is a system of linear equations with 11 unknowns, and 2 equations.
- To solve the system, we need at least 11 equations.
- How can we get more equations?

Solving Linear Equations

- Suppose we use 20 point correspondences between $[x_j, y_j, z_j, 1]$ and $[u_j, v_j, 1]$.
- Then, we get 40 equations.
- They can still be jointly expressed as A*x = b, where:
  - A is a 40*11 matrix.
  - x is an 11*1 matrix.
  - b is a 40 * 1 matrix.
  - Row 2j-1 of A is equal to: $x_j, y_j, z_j, 1, 0, 0, 0, 0, -x_ju_j, -y_ju_j, -z_ju_j$.
  - Row 2j of A is equal to: $0, 0, 0, 0, x_j, y_j, z_j, 1, -x_jv_j, -y_jv_j, -z_jv_j$
  - Row 2j-1 of b is equal to u_j.
  - Row 2j of b is equal to v_j.
  - x = $[c_{11}, c_{12}, c_{13}, c_{14}, c_{21}, c_{22}, c_{23}, c_{24}, c_{31}, c_{32}, c_{33}]'$.
- How do we solve this system of equations?

Solving A*x = b

- If we have > 11 equations, and only 11 unknowns, then the system is *overconstrained.*
- If we try to solve such a system, what happens?

Solving A*x = b

- If we have > 11 equations, and only 11 unknowns, then the system is *overconstrained.*
- There are two cases:
  - (Rare). An exact solution exists. In that case, usually only 11 equations are needed, the rest are redundant.
  - (Typical). No exact solution exists. Why?

Solving A*x = b

- If we have > 11 equations, and only 11 unknowns, then the system is *overconstrained.*
- There are two cases:
  - (Rare). An exact solution exists. In that case, usually only 11 equations are needed, the rest are redundant.
  - (Typical). No exact solution exists. Why? Because there is always some measurement error in estimating world coordinates and pixel coordinates.
- We need an approximate solution.
- Optimization problem. We take the standard two steps:
  - Step 1: define a measure of how good any solution is.
  - Step 2: find the *best* solution according to that measure.
- Note. “solution” here is not the BEST solution, just any proposed solution. Most “solutions” are really bad!

Least Squares Solution

- Each solution produces an error for each equation.
- Sum-of-squared-errors is the measure we use to evaluate a solution.
- The least squares solution is the solution that minimizes the sum-of-squared-errors measure.
- Example:
  - let x2 be a proposed solution.
  - Let b2 = A * x2.
  - If x2 was the mathematically perfect solution, b2 = b.
  - The error e(i) at position i is defined as |b2(i) – b(i)|.
  - The squared error at position i is defined as $|b2(i) – b(i)|^2$.
  - The sum of squared errors is sum(sum((b2(i) – b(i)).^2)).

Least Squares Solution

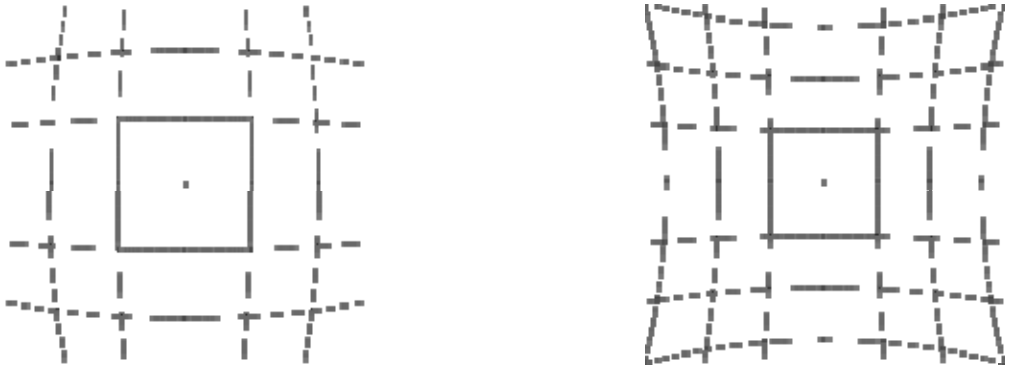
- Each solution produces an error for each equation.
- Sum-of-squared-errors is the measure we use to evaluate a solution.
- The least squares solution is the solution that minimizes the sum-of-squared-errors measure.
- Finding the least-squares solution to a set of linear equations is mathematically involved.
- However, in Matlab it is really easy:
  - Given a system of linear equations expressed as A*x = b, to find the least squares solution, type:
  - x = A\b

Producing World Coordinates

- Typically, a calibration object is used.
- Checkerboard patterns and laser pointers are common.
- A point on the calibration object is designated as the origin.
- The x, y and z directions of the object are used as axis directions of the world coordinate system.
- Correspondences from world coordinates to pixel coordinates can be established manually or automatically.
  - With a checkerboard pattern, automatic estimation of correspondences is not hard.

Calibration in the Real World

- Typically, cameras do not obey the perspective model closely enough.
- Radial distortion is a common deviation.
- Calibration software needs to account for radial distortion.

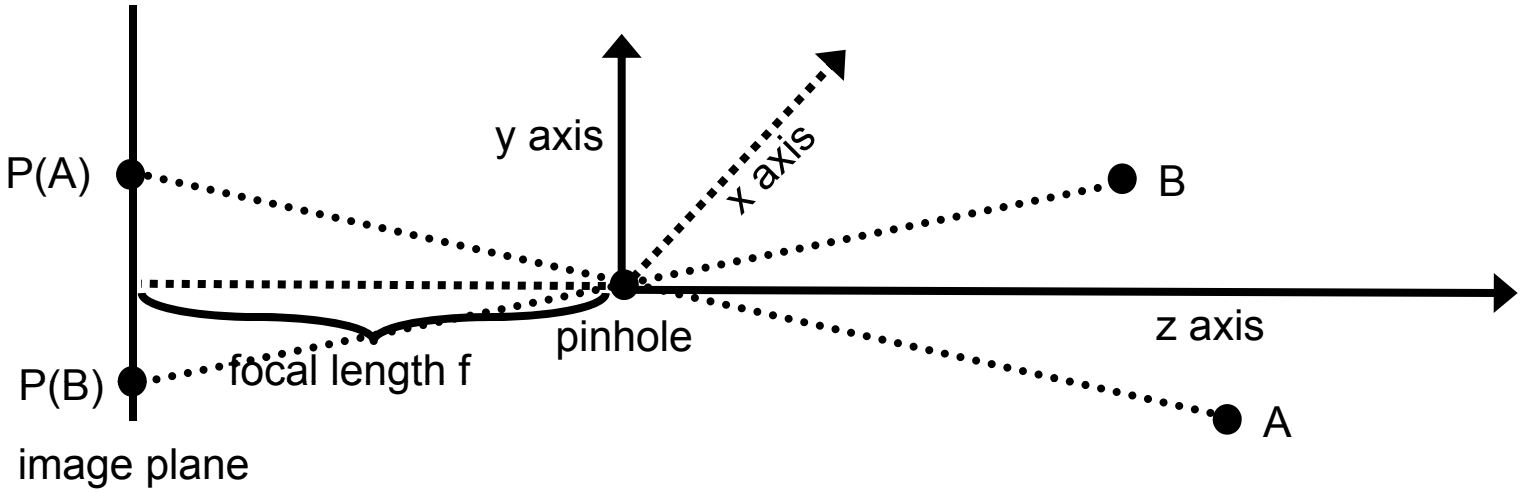


Two types of radial distortion: barrel distortion and pincushion distortion.
Images from Wikipedia

Part 3

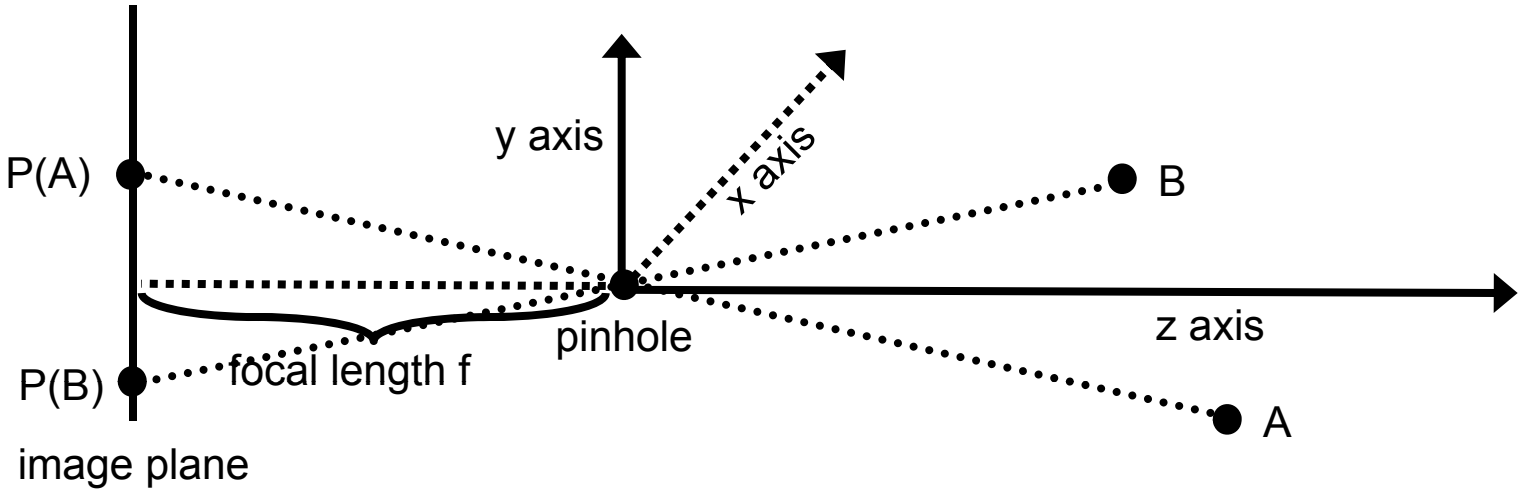
Depth from Stereo

Image Projection Review



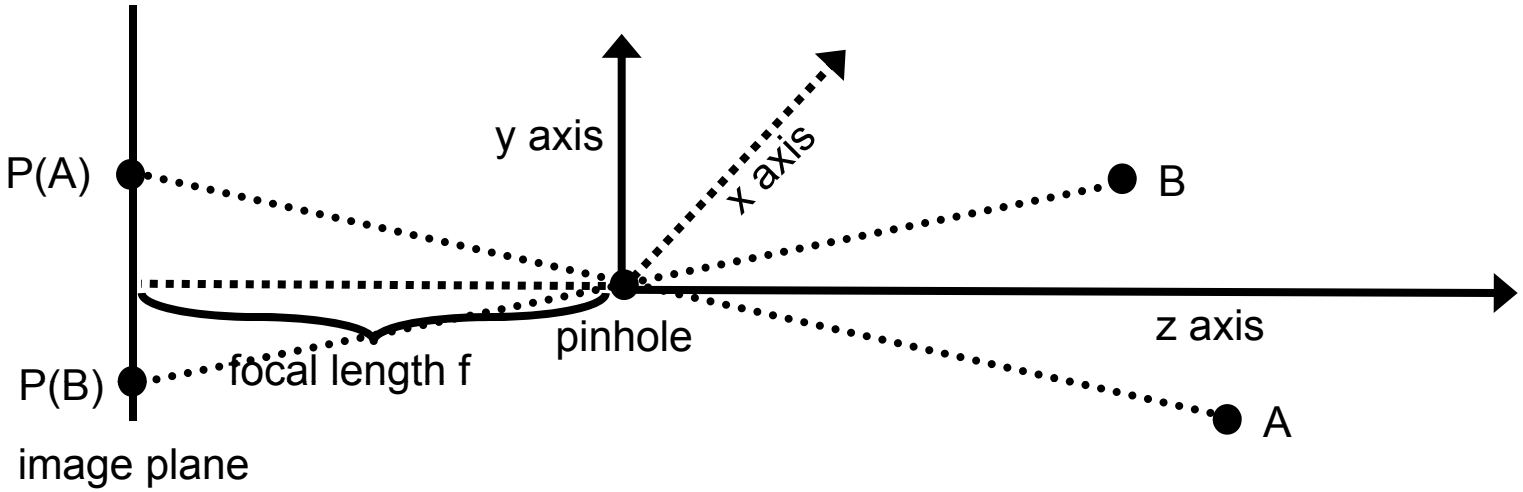
- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $H = \begin{pmatrix} -fS_x & 0 & u_0 & 0 \\ 0 & -fS_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.
  - P(A) = R * T * A gives us the projection (in world coordinates) of A on an image plane of what focal length?

Image Projection Review



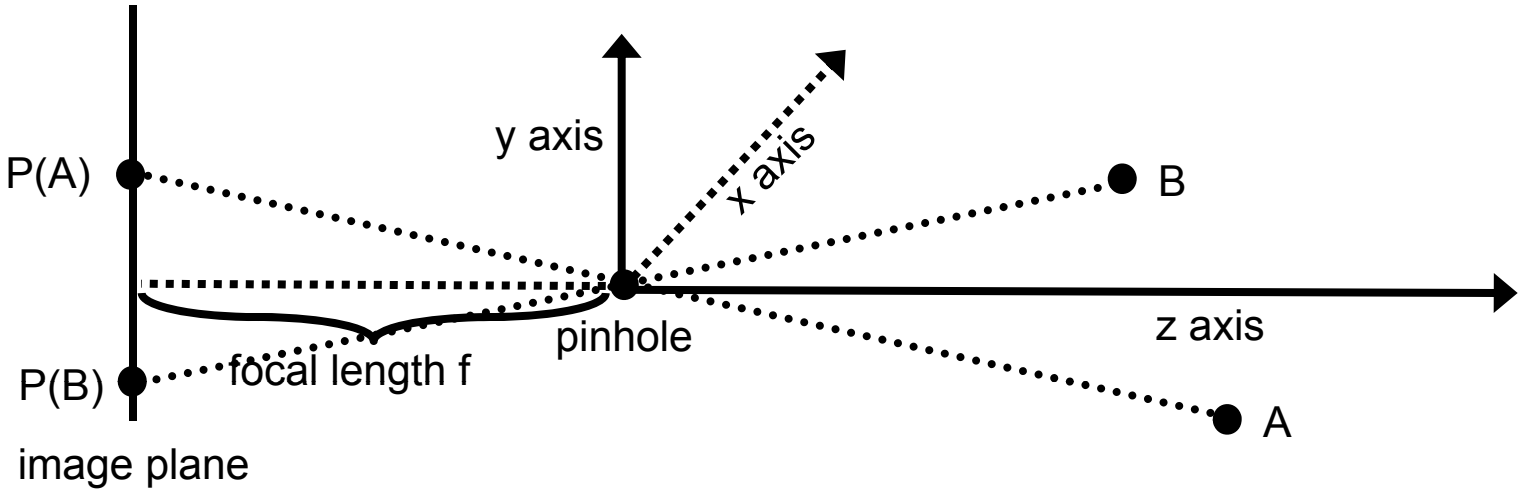
- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $H = \begin{pmatrix} -fS_x & 0 & u_0 & 0 \\ 0 & -fS_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.
  - P(A) = R * T * A gives us the projection (in world coordinates) of A on an image plane with focal length 1.
  - H * P(A)) gives us the pixel coordinates corresponding to P(A). For simplicity, the focal length is encoded in H.

Image-to-World Projection



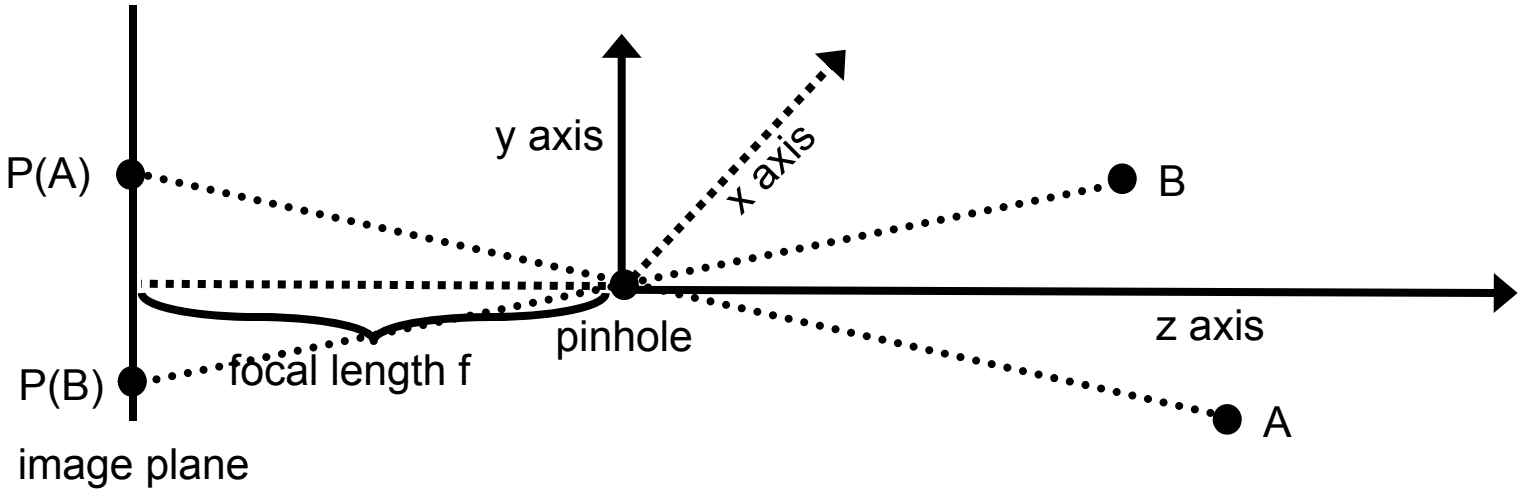
- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $H = \begin{pmatrix} -fS_x & 0 & u_0 & 0 \\ 0 & -fS_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.
  - Given pixel location W = (u, v), how can we get the world coordinates of the corresponding position on the image plane?

Image-to-World Projection



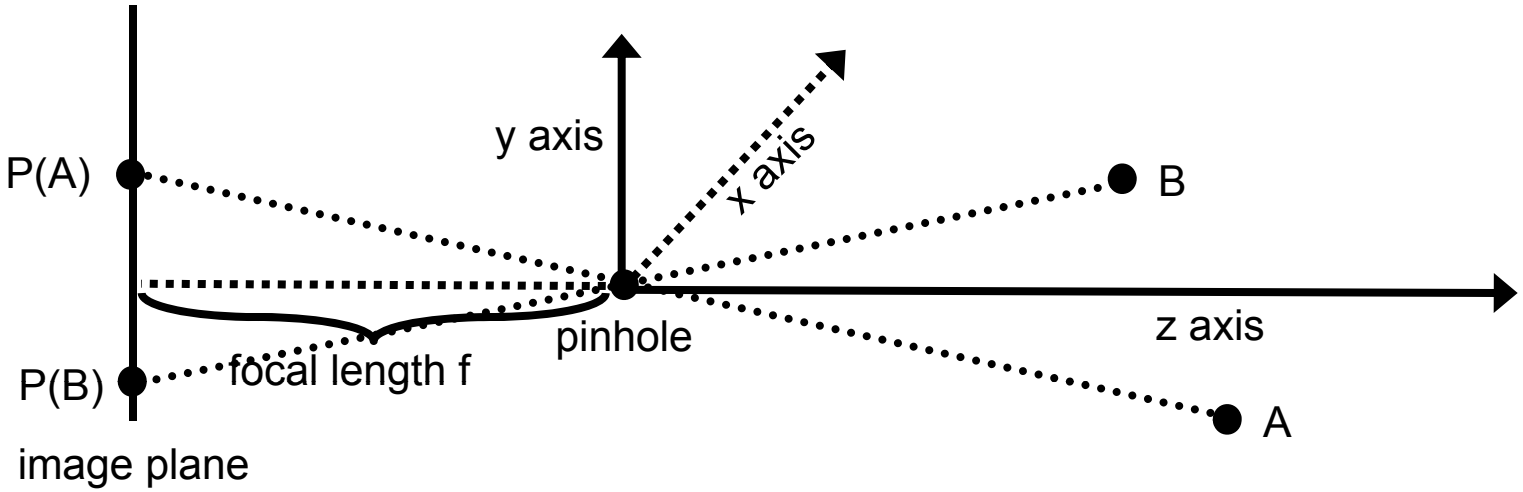
- Let $A=\begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $R=\begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $T=\begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $H=\begin{pmatrix} -fS_x & 0 & u_0 & 0 \\ 0 & -fS_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.
  - Define $G=\begin{pmatrix} -fS_x & 0 & u_0 \\ 0 & -fS_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}$. G maps $(x_0, y_0, 1)^{trans}$ to $(u, v, 1)^{trans}$.
    - (x_0, y_0) are the normalized image coordinates corresponding to (u, v).

Image-to-World Projection



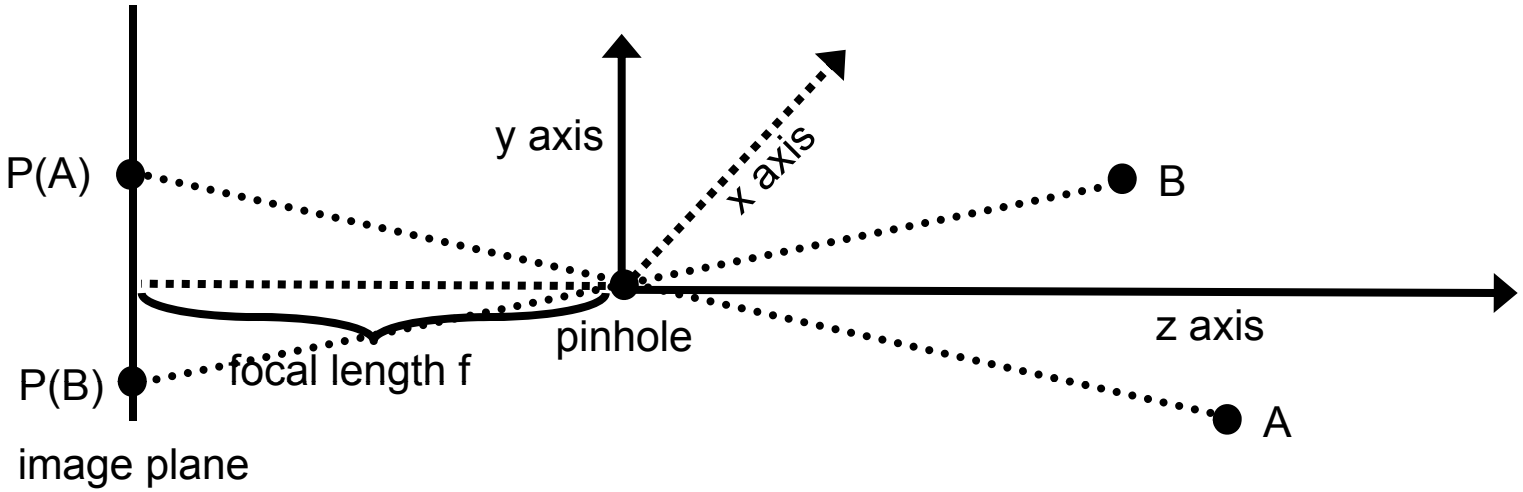
- Let $A=\begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $R=\begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $T=\begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $H=\begin{pmatrix} -fS_x & 0 & u_0 & 0 \\ 0 & -fS_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.
  - Define $G=\begin{pmatrix} -fS_x & 0 & u_0 \\ 0 & -fS_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}$. G maps $(x_0, y_0, 1)^{trans}$ to $(u, v, 1)^{trans}$.
    - (x_0, y_0) are the normalized image coordinates corresponding to (u, v).
  - G^{-1} maps (u, v) to normalized image coordinates.

Image-to-World Projection



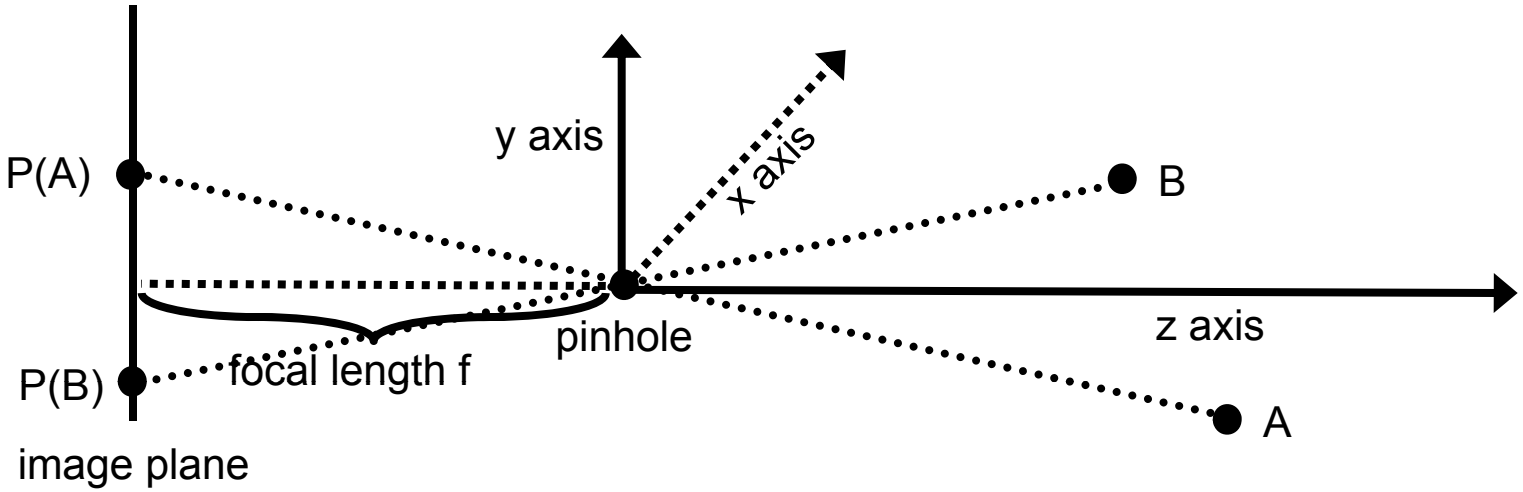
- Define G = $\begin{pmatrix} -fS_x & 0 & u_0 \\ 0 & -fS_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}$. G maps $(x_0, y_0, 1)^{trans}$ to $(u, v, 1)^{trans}$.
  - (x_0, y_0) are the normalized image coordinates corresponding to (u, v).
- G^{-1} maps (u, v) to normalized image coordinates (x_0, y_0).
  - In camera coordinates, what is the z coordinate of $G^{-1}(u, v)$?

Image-to-World Projection



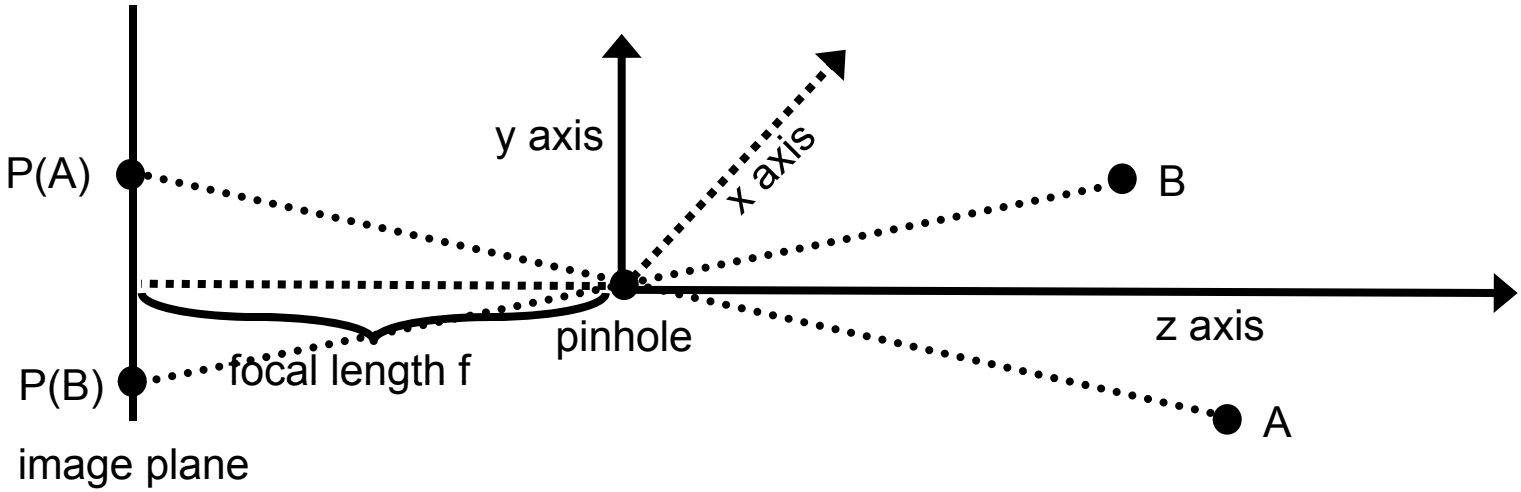
- Define G = $\begin{pmatrix} -fS_x & 0 & u_0 \\ 0 & -fS_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}$. G maps $(x_0, y_0, 1)^{trans}$ to $(u, v, 1)^{trans}$.
  - (x_0, y_0) are the normalized image coordinates corresponding to (u, v).
- G^{-1} maps (u, v) to normalized image coordinates (x_0, y_0).
  - In camera coordinates, what is the z coordinate of $G^{-1}(u, v)$?
  - Remember, G^{-1} maps pixels into an image plane corresponding to focal length = ?

Image-to-World Projection



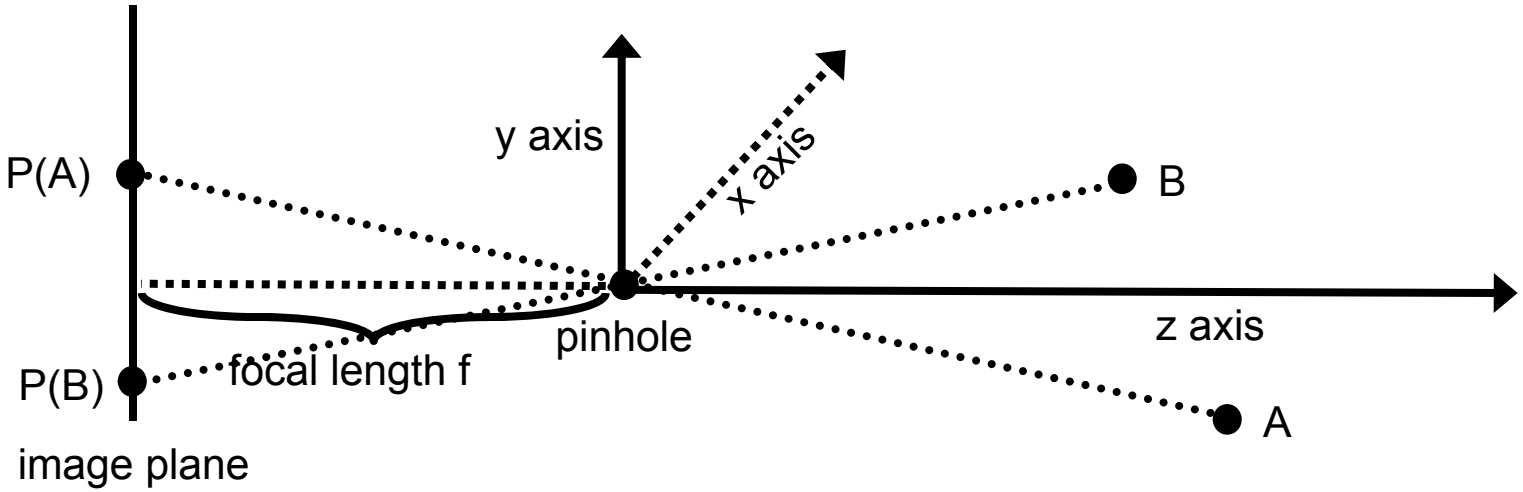
- Define G = $\begin{pmatrix} -fS_x & 0 & u_0 \\ 0 & -fS_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}$. G maps $(x_0, y_0, 1)^{trans}$ to $(u, v, 1)^{trans}$.
  - (x_0, y_0) are the normalized image coordinates corresponding to (u, v).
- G^{-1} maps (u, v) to normalized image coordinates (x_0, y_0).
  - In camera coordinates, what is the z coordinate of $G^{-1}(u, v)$? z = -1.
  - Remember, G^{-1} maps pixels into an image plane corresponding to focal length f = 1.

Image-to-World Projection



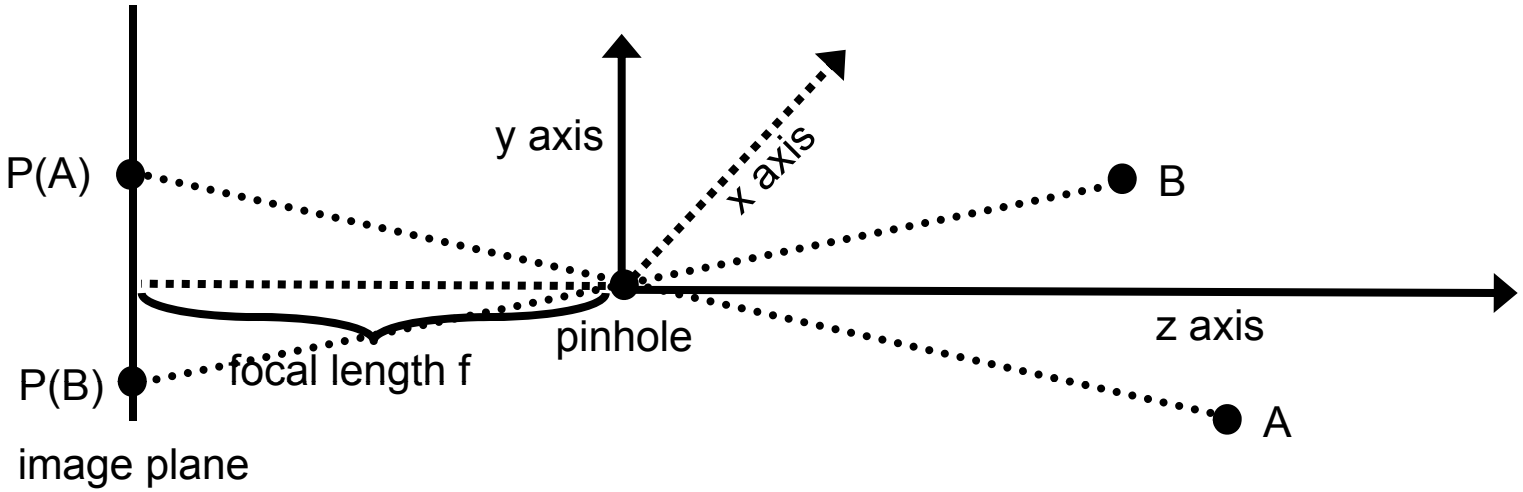
- Now we have mapped pixel (u, v) to image plane position $(x_0, y_0, -1)$.
- Next step: map image plane position to position in the world.
  - First in camera coordinates.
- What world position does image plane position $(x_0, y_0, -1)$ map to?

Image-to-World Projection



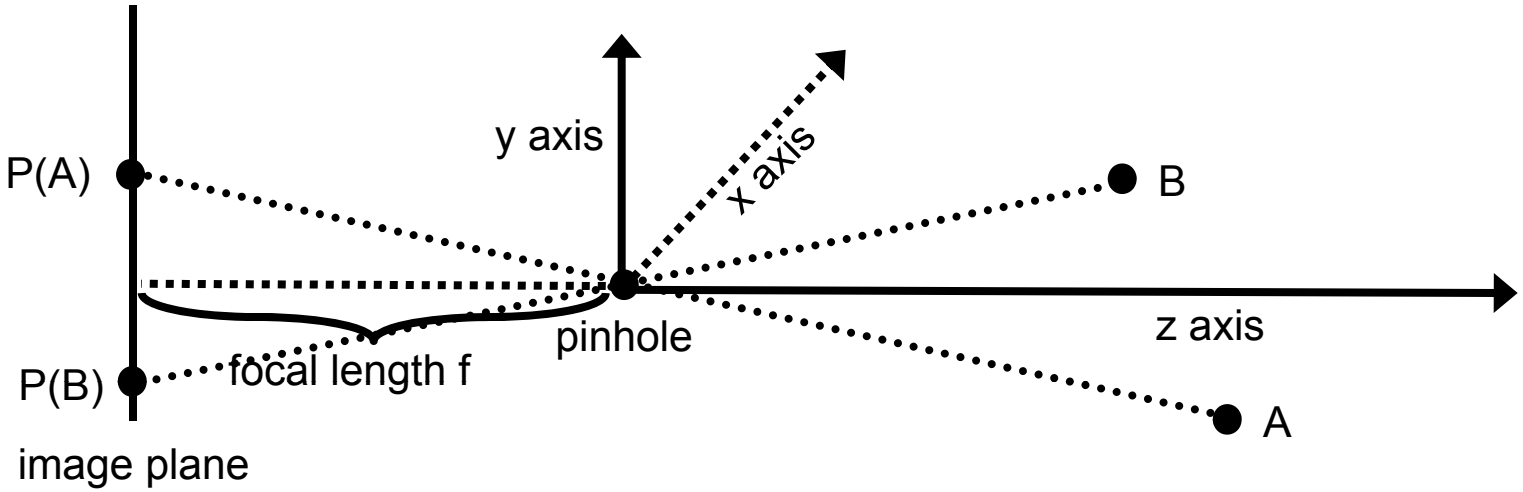
- Now we have mapped pixel (u, v) to image plane position $(x_0, y_0, -1)$.
- Next step: map image plane position to position in the world.
  - First in camera coordinates.
- What world position does image plane position $(x_0, y_0, -1)$ map to?
- $(x_0, y_0, -1)$ maps to a line. In camera coordinates, the line goes through the origin.
- How can we write that line in camera coordinates?

Image-to-World Projection



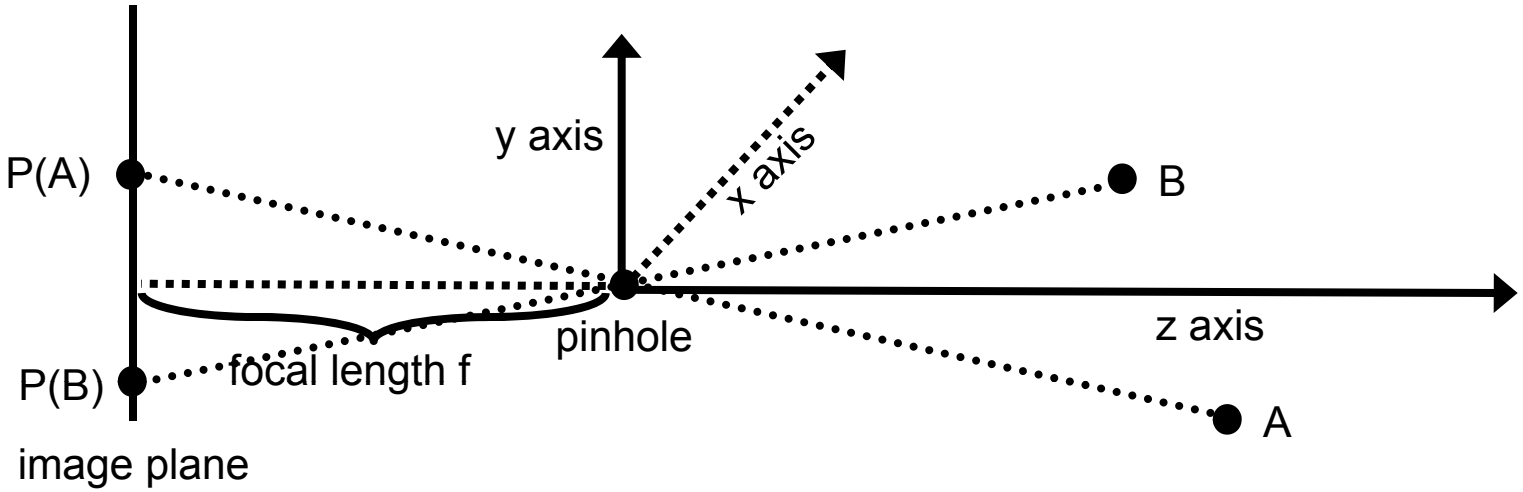
- $(x_0, y_0, -1)$ maps to a line. In camera coordinates, the line goes through the origin.
- How can we write that line in camera coordinates?
- Suppose that the line goes through point (x, y, z). What equations does that point have to satisfy?
  - $x / x_0 = z / (-1) \Rightarrow z = x * (-1)/x_0$.
  - $y / y_0 = x / x_0 \Rightarrow y = x * y_0/x_0$.
- These equations define a line $(y, z) = f(x)$. Borderline cases: $x_0 = 0$, $y_0 = 0$.

Image-to-World Projection



- $(x_0, y_0, -1)$ maps to a line. In camera coordinates, the line goes through the origin.
- Suppose that the line goes through point (x, y, z). What equations does that point have to satisfy?
  - $x / x_0 = z / (-1) \Rightarrow z = x * (-1)/x_0$.
  - $y / y_0 = x / x_0 \Rightarrow y = x * y_0/x_0$.
- These equations define a line (y, z) = f(x). Borderline cases: $x_0 = 0$, $y_0 = 0$.

Image-to-World Projection



- $(x_0, y_0, -1)$ maps to a line. Suppose that the line goes through point (x, y, z). What equations does that point have to satisfy?
  - $x / x_0 = z / (-1) \Rightarrow z = x * (-1)/x_0$.
  - $y / y_0 = x / x_0 \Rightarrow y = x * y_0/x_0$.
- These equations define a line (y, z) = f(x). Borderline cases: $x_0 = 0$, $y_0 = 0$.
- World-to-camera mapping of A is done by camera(A) = R*T*A.
  - Maps 3D world point to point on camera plane.
- $T^{-1} * R^{-1} *$ camera(A) maps camera(A) to a line in the 3D world.

Stereo Vision

- Also called *stereopsis*.
- Key idea:
  - Each point in an image corresponds to a line in the 3D world.
  - To compute that line, we need to know the camera matrix.
  - If the same point is visible from two images, the two corresponding lines intersect in a single 3D point.
- Challenges:
  - Identify correspondences between images from the two cameras.
  - Compute the camera matrix.

A Simple Stereo Setup

- Simple arrangement:
  - Both cameras have same intrinsic parameters.
  - Image planes belong to the same world plane.
- Then, correspondences appear on the same horizontal line.
- The displacement from one image to the other is called *disparity*.
- Disparity is proportional to depth.
- External calibration parameters are not needed.

A Simple Stereo Setup

- Assume that:
  - Both cameras have pinholes at z=0, y = 0.
  - Both image planes correspond to f=1, z = -1.
  - Both cameras have the same intrinsic parameters f, S_x, S_y, u_0, v_0.
  - Both camera coordinate systems have the same x, y, z axes.
  - Cameras only differ at the x coordinate of the pinhole.
    - Camera 1 is at (x_1, 0, 0), camera 2 is at (x_2, 0, 0).
- Then:
  - Suppose a point A is at (x_A, y_A, z_A).
  - On camera 1, A maps to normalized image coordinates:

A Simple Stereo Setup

- Assume that:
  - Both cameras have pinholes at z=0, y = 0.
  - Both image planes correspond to f=1, z = -1.
  - Both cameras have the same intrinsic parameters f, S_x, S_y, u_0, v_0.
  - Both camera coordinate systems have the same x, y, z axes.
  - Cameras only differ at the x coordinate of the pinhole.
    - Camera 1 is at (x_1, 0, 0), camera 2 is at (x_2, 0, 0).
- Then:
  - Suppose a point A is at (x_A, y_A, z_A).
  - On camera 1, A maps to normalized image coordinates:
    - $(x_{1A}, y_{1A}) = ((x_A - x_1) / z_A, y_A / z_A)$
  - On camera 2, A maps to normalized image coordinates:

A Simple Stereo Setup

- Assume that:
  - Both cameras have pinholes at z=0, y = 0.
  - Both image planes correspond to f=1.
  - Both cameras have the same intrinsic parameters f, S_x, S_y, u_0, v_0.
  - Both camera coordinate systems have the same x, y, z axes.
  - Cameras only differ at the x coordinate of the pinhole.
    - Camera 1 is at (x_1, 0, 0), camera 2 is at (x_2, 0, 0).
- Then:
  - Suppose a point A is at (x_A, y_A, z_A).
  - On camera 1, A maps to normalized image coordinates:
    - $(x_{1A}, y_{1A}) = ((x_A - x_1) / z_A, y_A / z_A)$
  - On camera 2, A maps to normalized image coordinates:
    - $(x_{2A}, y_{2A}) = ((x_A - x_2) / z_A, y_A / z_A)$
  - $(x_{1A} - x_{2A}) = ((x_A - x_1) - (x_A - x_2)) / z_A = (x_2 - x_1) / z_A = c / z_A$.
  - $(x_{1A} - x_{2A})$ is called ***disparity***. Disparity is inversely proportional to z_A.

A Simple Stereo Setup

- Suppose a point A is at (x_A, y_A, z_A).
- On camera 1, A maps to normalized image coordinates:
  - $(x_{1A}, y_{1A}) = ((x_A - x_1) / z_A, y_A / z_A)$
- On camera 2, A maps to normalized image coordinates:
  - $(x_{2A}, y_{2A}) = ((x_A - x_2) / z_A, y_A / z_A)$
- $(x_{1A} - x_{2A}) = ((x_A - x_1) - (x_A - x_2)) / z_A = (x_2 - x_1) / z_A = c / z_A$.
- $(x_{1A} - x_{2A})$ is called ***disparity***. Disparity is inversely proportional to z_A.
- If we know (x_{1A}, y_{1A}) and (x_{2A}, y_{2A}) (i.e., we know the locations of A in each image), what else do we need to know in order to figure out z_A?

A Simple Stereo Setup

- Suppose a point A is at (x_A, y_A, z_A).
- On camera 1, A maps to normalized image coordinates:
  - $(x_{1A}, y_{1A}) = ((x_A - x_1) / z_A, y_A / z_A)$
- On camera 2, A maps to normalized image coordinates:
  - $(x_{2A}, y_{2A}) = ((x_A - x_2) / z_A, y_A / z_A)$
- $(x_{1A} - x_{2A}) = ((x_A - x_1) - (x_A - x_2)) / z_A = (x_2 - x_1) / z_A = c / z_A$.
- $(x_{1A} - x_{2A})$ is called ***disparity***. Disparity is inversely proportional to z_A.
- If we know (x_{1A}, y_{1A}) and (x_{2A}, y_{2A}) (i.e., we know the locations of A in each image), what else do we need to know in order to figure out z_A?
- We need to know $c = (x_2 - x_1)$.

A More General Case

- Suppose that we start with the simple system:
  - Both cameras have pinholes at z=0, y = 0.
  - Both image planes correspond to f=1, z=-1.
  - Both cameras have the same intrinsic parameters f, S_x, S_y, u_0, v_0.
  - Both camera coordinate systems have the same x, y, z axes.
  - Cameras only differ at the x coordinate of the pinhole.
    - Camera 1 is at (x_1, 0, 0), camera 2 is at (x_2, 0, 0).
- Then we rotate by R and translate by T the whole system.
- To find point A, we just need to:
  - go back to simple coordinates, by translating back and rotating back. This is done via matrix ?

A More General Case

- Suppose that we start with the simple system:
  - Both cameras have pinholes at z=0, y = 0.
  - Both image planes correspond to f=1, z=-1.
  - Both cameras have the same intrinsic parameters f, S_x, S_y, u_0, v_0.
  - Both camera coordinate systems have the same x, y, z axes.
  - Cameras only differ at the x coordinate of the pinhole.
    - Camera 1 is at (x_1, 0, 0), camera 2 is at (x_2, 0, 0).
- Then we rotate by R and translate by T the whole system.
- Given point A in the new coordinate system, how do we translate it back to simple(A), in the simple coordinate system?

A More General Case

- Suppose that we start with the simple system:
  - Both cameras have pinholes at z=0, y = 0.
  - Both image planes correspond to f=1, z=-1.
  - Both cameras have the same intrinsic parameters f, S_x, S_y, u_0, v_0.
  - Both camera coordinate systems have the same x, y, z axes.
  - Cameras only differ at the x coordinate of the pinhole.
    - Camera 1 is at (x_1, 0, 0), camera 2 is at (x_2, 0, 0).
- Then we rotate by R and translate by T the whole system.
- Given point A in the new coordinate system, how do we translate it back to simple(A), in the simple coordinate system?
  - simple(A) = $R^{-1} * T^{-1}(A)$.

The General Case

- Given two calibrated cameras, and a corresponding pair of locations, we compute two lines.
- In the mathematically ideal case, the lines intersect.
- By finding the intersection, we compute where the 3D location is.

The General Case

- Given two calibrated cameras, and a corresponding pair of locations, we compute two lines.
- In the mathematically ideal case, the lines intersect.
- In practice, what happens?

The General Case

- Given two calibrated cameras, and a corresponding pair of locations, we compute two lines.
- In the mathematically ideal case, the lines intersect.
- In practice, the two lines don't intersect because of rounding/measurement errors (pixels are discretized).
  - What is it that makes it very unlikely that the two lines will intersect?

The General Case

- Given two calibrated cameras, and a corresponding pair of locations, we compute two lines.
- In the mathematically ideal case, the lines intersect.
- In practice, the two lines don't intersect because of rounding/measurement errors (pixels are discretized).
  - What is it that makes it very unlikely that the two lines will intersect?
  - For lines to intersect, the rounding/measurement error must be exactly 0.

The General Case

- Given two calibrated cameras, and a corresponding pair of locations, we compute two lines.
- In the mathematically ideal case, the lines intersect.
- In practice, the two lines don’t intersect because of rounding/measurement errors (pixels are discretized).
- Best estimate for the 3D point is obtained by:

The General Case

- Given two calibrated cameras, and a corresponding pair of locations, we compute two lines.
- In the mathematically ideal case, the lines intersect.
- In practice, the two lines don't intersect because of rounding/measurement errors (pixels are discretized).
- Best estimate for the 3D point is obtained by:
  - Finding the shortest line segment that connects the two lines.
  - Returning the midpoint of that segment.

Finding Connecting Segment

- P_1: point on first line.
- Q_1: point on second line.
- u_1: unit vector parallel to first line.
- u_2: unit vector parallel to second line.
- $P_1 + a_1u_1$: intersection of segment with first line.
- $Q_1 + a_2u_2$: intersection of segment with second line.

- We know that:
  - The shortest connecting segment is perpendicular to the first line.
  - The shortest connecting segment is perpendicular to the second line.
  - Why?

Finding Connecting Segment

- P_1: point on first line.
- Q_1: point on second line.
- u_1: unit vector parallel to first line.
- u_2: unit vector parallel to second line.
- $P_1 + a_1u_1$: intersection of segment with first line.
- $Q_1 + a_2u_2$: intersection of segment with second line.

- We know that:
  - The shortest connecting segment is perpendicular to the first line.
  - The shortest connecting segment is perpendicular to the second line.
  - Why?
  - If not, then it is not the shortest connecting segment.

Finding Connecting Segment

- $((P_1 + a_1u_1) – (Q_1 + a_2u_2)) * u_1 = 0$
- $((P_1 + a_1u_1) – (Q_1 + a_2u_2)) * u_2 = 0$
  - * here stands for *dot product*.
  - P_1: point on first line.
  - Q_1: point on second line.
  - u_1: unit vector parallel to first line.
  - u_2: unit vector parallel to second line.
  - $P_1 + a_1u_1$: intersection of segment with first line.
  - $Q_1 + a_2u_2$: intersection of segment with second line.
- Only unknowns are a1 and a2.
- We have two equations, two unknowns, can solve.

Essential Matrix

- We define a stereo pair given two cameras (in an arbitrary configuration).
- The essential matrix E of this stereo pair is a matrix that has the following property:
  - If W and W' are homogeneous normalized image coordinates in image 1 and image 2, and these locations correspond to the same 3D point, then: $(W')^{transpose} * E * W = 0$.

Estimating the Essential Matrix

- The essential matrix E of this stereo pair is a matrix that has the following property:
  - If W and W' are homogeneous normalized image coordinates in image 1 and image 2, and these locations correspond to the same 3D point, then: $(W')^{transpose} * E * W = 0$.
- E has size 3x3. To estimate E, we need to estimate 9 unknowns.
- Observations:
  - A trivial and not useful exact solution is E = 0.
  - If E is a solution, then cE is also a solution, for any real number c. So, strictly speaking we can only solve up to scale, and we only need to estimate 8 unknowns.
  - To avoid the E=0 solution, we impose an additional constraint:
    - sum(sum(E.*E)) = 1.

Using a Single Correspondence

- Suppose (u_1, v_1, w_1) in image plane 1 matches (u_2, v_2, w_2) in image plane 2.
  - Remember, (u_1, v_1, w_1) and (u_2, v_2, w_2) are given in homogeneous normalized image coordinates.
- We know that $(u_1, v_1, w_1) * E * (u_2, v_2, w_2)_{transpose} = 0$.
- Let $E = \begin{pmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{pmatrix}$
  - We obtain: $[u_1, v_1, w_1] * \begin{pmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{pmatrix} * [u_2, v_2, w_2]' = 0 \Rightarrow$

$$[u_1e_{11}+v_1e_{21}+w_1e_{31},\ u_1e_{12}+v_1e_{22}+w_1e_{32},\ u_1e_{13}+v_1e_{23}+w_1e_{33}] * [u_2, v_2, w_2]' = 0 \Rightarrow$$

$$[u_1u_2e_{11}+v_1u_2e_{21}+w_1u_2e_{31}+u_1v_2e_{12}+v_1v_2e_{22}+w_1v_2e_{32}+u_1w_2e_{13}+v_1w_2e_{23}+w_1w_2e_{33}] = 0 \Rightarrow$$

$$[u_1u_2,v_1u_2,w_1u_2,u_1v_2,v_1v_2,w_1v_2,u_1w_2,v_1w_2,w_1w_2] * [e_{11},e_{21},e_{31},e_{12},e_{22},e_{32},e_{13},e_{23},e_{33}]' = 0$$

Using Multiple Correspondences

- From previous slide: if (u_1, v_1, w_1) in image plane 1 matches (u_2, v_2, w_2) in image plane 2:

$$[u_1u_2, v_1u_2, w_1u_2, u_1v_2, v_1v_2, w_1v_2, u_1w_2, v_1w_2, w_1w_2] * [e_{11}, e_{21}, e_{31}, e_{12}, e_{22}, e_{32}, e_{13}, e_{23}, e_{33}]' = 0$$

- If we have J correspondences:
  - $(u_{1,j}, v_{1,j}, w_{1,j})$ in image plane 1 matches $(u_{2,j}, v_{2,j}, w_{2,j})$ in image plane 2:
- Define

$$A = \begin{pmatrix} u_{1,1}u_{2,1}, & v_{1,1}u_{2,1}, & w_{1,1}u_{2,1}, & u_{1,1}v_{2,1}, & v_{1,1}v_{2,1}, & w_{1,1}v_{2,1}, & u_{1,1}w_{2,1}, & v_{1,1}w_{2,1}, & w_{1,1}w_{2,1} \\ u_{1,2}u_{2,2}, & v_{1,2}u_{2,2}, & w_{1,2}u_{2,2}, & u_{1,2}v_{2,2}, & v_{1,2}v_{2,2}, & w_{1,2}v_{2,2}, & u_{1,2}w_{2,2}, & v_{1,2}w_{2,2}, & w_{1,2}w_{2,2} \\ u_{1,3}u_{2,3}, & v_{1,3}u_{2,3}, & w_{1,3}u_{2,3}, & u_{1,3}v_{2,3}, & v_{1,3}v_{2,3}, & w_{1,3}v_{2,3}, & u_{1,3}w_{2,3}, & v_{1,3}w_{2,3}, & w_{1,3}w_{2,3} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ u_{1,J}u_{2,J}, & v_{1,J}u_{2,J}, & w_{1,J}u_{2,J}, & u_{1,J}v_{2,J}, & v_{1,J}v_{2,J}, & w_{1,J}v_{2,J}, & u_{1,J}w_{2,J}, & v_{1,J}w_{2,J}, & w_{1,J}w_{2,J} \end{pmatrix}$$

Using Multiple Correspondences

- Using A from the previous slide, the following holds:

$$A * \begin{pmatrix} e_{11} \\ e_{21} \\ e_{31} \\ e_{12} \\ e_{22} \\ e_{32} \\ e_{13} \\ e_{23} \\ e_{33} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \dots \\ 0 \end{pmatrix}$$

- A: Jx9 matrix.
- Matrix of unknowns e_{ij}: size 9x1.
- Result: a zero matrix of size Jx1.

- This is a system of linear homogeneous equations, that can be solved using SVD.
- In Matlab:
  - [u, d, v] = svd(A, 0);
  - x = v(:, end)
- After the above two lines, x is the 9x1 matrix of unknowns.
- This way, using multiple correspondences, we have computed the essential matrix.
  - Strictly speaking, we have computed *one* out of many essential matrices.
  - Solution up to scale.

Epipoles – Epipolar Lines

- In each image of a stereo pair, the epipole is the pixel location where the pinhole of *the other camera* is mapped.
- Given a pixel in an image, where can the corresponding pixel be in the other image?
  - The essential matrix defines a line.
  - All such lines are called epipolar lines, because they always go through the epipole.
  - Why?

Epipoles – Epipolar Lines

- In each image of a stereo pair, the epipole is the pixel location where the pinhole of *the other camera* is mapped.
- Given a pixel in an image, where can the corresponding pixel be in the other image?
  - The essential matrix defines a line.
  - All such lines are called epipolar lines, because they always go through the epipole.
  - Why?
    - Because for any pixel in image 1, the pinhole of camera 1 is a possible 3D location.

Epipoles – Epipolar Lines

- In each image of a stereo pair, the epipole is the pixel location where the pinhole of *the other camera* is mapped.
- Given a pixel in an image, where can the corresponding pixel be in the other image?
  - The essential matrix defines a line.
  - All such lines are called epipolar lines, because they always go through the epipole.
- Given a pixel in one image, the epipolar line in the other image can be computed using the essential matrix.

3D Location from Stereo

- Given a pixel W in one image:
- Find corresponding pixel W’ in the other image.
  - How?

3D Location from Stereo

- Given a pixel W in one image:
- Find corresponding pixel W’ in the other image.
  - How?
  - Use the essential matrix to find the line of possible corresponding pixels in the other image.
  - Use computer vision methods to find the best match.
    - This is a hard problem.
    - Easier when pixels are not matched in isolation, but rather when we are looking for an overall mapping of pixels from one image to pixels of the other image, that is locally consistent.
  - Or, use a human to click on the corresponding pixel.

3D Location from Stereo

- Given a pixel W in one image:
- Find corresponding pixel W’ in the other image.
- Then what?

3D Location from Stereo

- Given a pixel W in one image:
- Find corresponding pixel W’ in the other image.
- Compute the two 3D lines corresponding to the two pixels.
- ???

3D Location from Stereo

- Given a pixel W in one image:
- Find corresponding pixel W’ in the other image.
- Compute the two 3D lines corresponding to the two pixels.
- Find the shortest connecting segment.
- Find the midpoint of that segment.
  - That is the “best” estimate of 3D location.

3D Location from Stereo

- Given a pixel W in one image:
- Find corresponding pixel W’ in the other image.
- Compute the two 3D lines corresponding to the two pixels.
- Find the shortest connecting segment.
- Find the midpoint of that segment.
  - That is the “best” estimate of 3D location.
- What are possible pitfalls? When would this fail?

3D Location from Stereo

- Given a pixel W in one image:
- Find corresponding pixel W' in the other image.
- Compute the two 3D lines corresponding to the two pixels.
- Find the shortest connecting segment.
- Find the midpoint of that segment.
  - That is the "best" estimate of 3D location.
- What are possible pitfalls? When would this fail?
  - If W' is not the true corresponding pixel for W.
  - If the 3D point projected to W is not visible from the other image.