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?

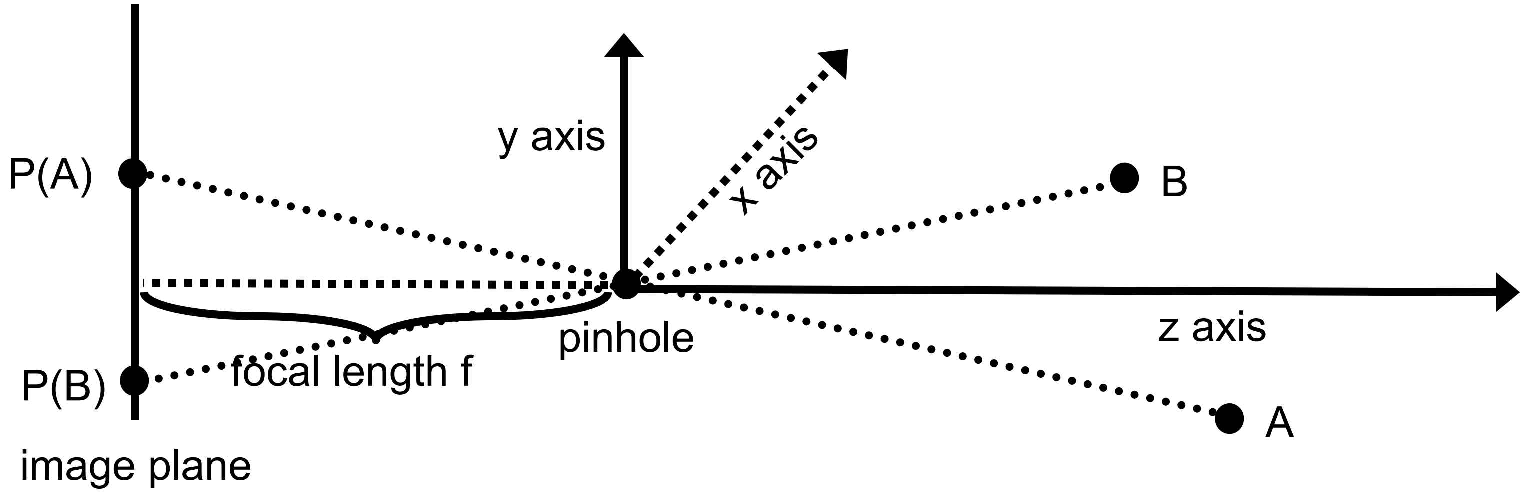
The Need for 3D Information

- 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

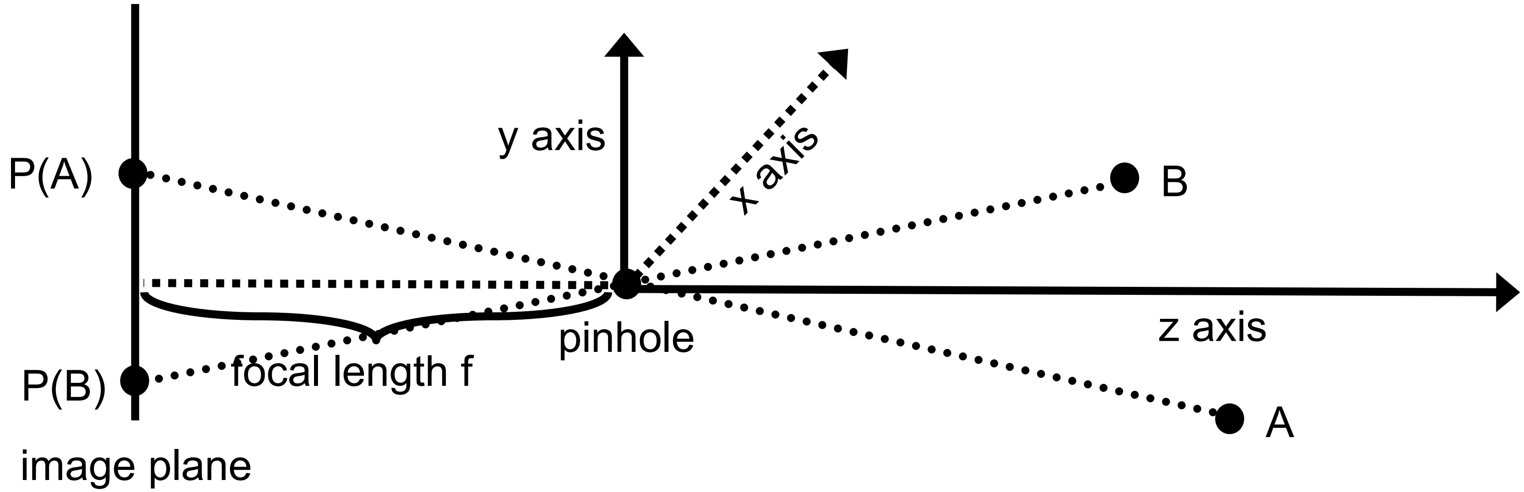
- To estimate 3D information, we ask the question:
  - Given a pixel (u, v), what 3D point (x, y, z) is seen at that pixel?
- That is a hard problem (one-to-many).
  - Can be solved if we have additional constraints.
  - For example, if we have two cameras (stereo vision).
- We start by solving the inverse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



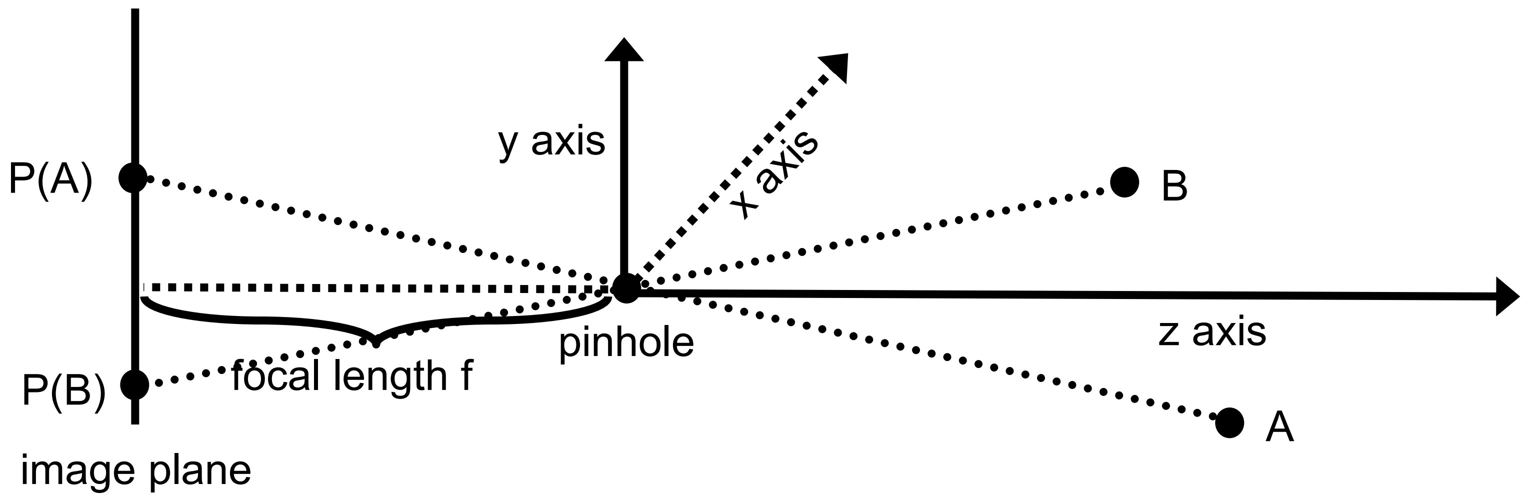
- Terminology:
  - *image plane* is a planar surface of sensors. The response of those sensors to light is the signal that forms the image.
  - The *focal length f* is the distance between the image plane and the pinhole.
  - A set of points is *collinear* if there exists a straight line going through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

Pinhole Model



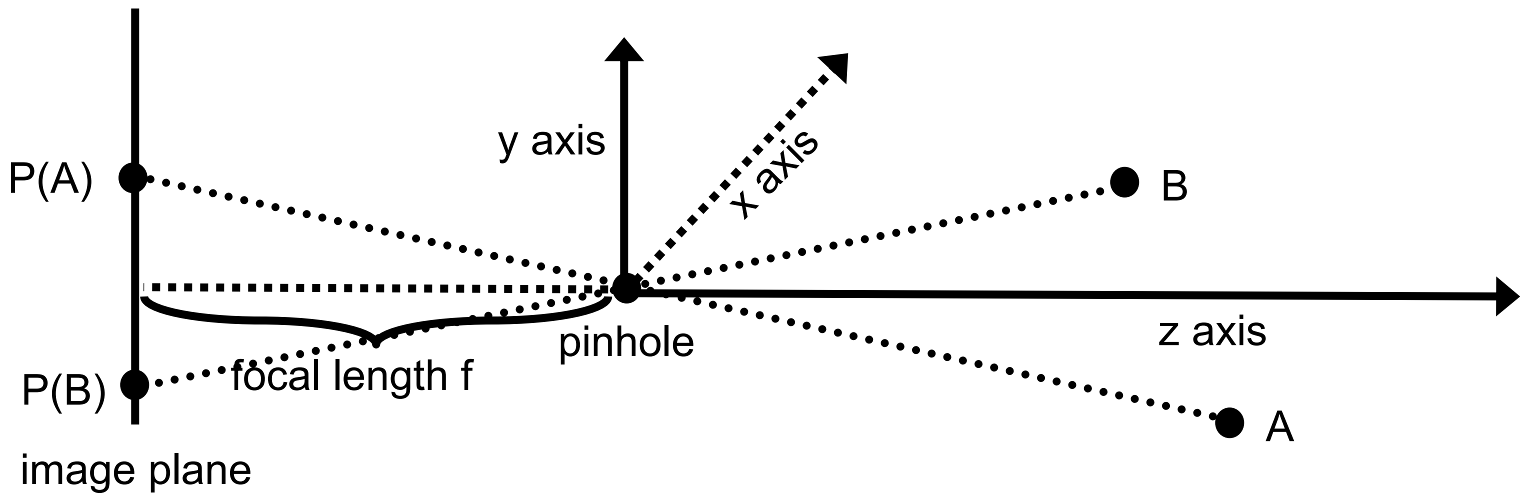
- Pinhole model:
  - light from all points enters the camera through an infinitesimal hole, and then reaches the *image plane*.
  - The *focal length f* is the distance between the image plane and the pinhole.
  - the light from point A reaches image location P(A), such that A, the pinhole, and P(A)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

Different Coordinate Systems



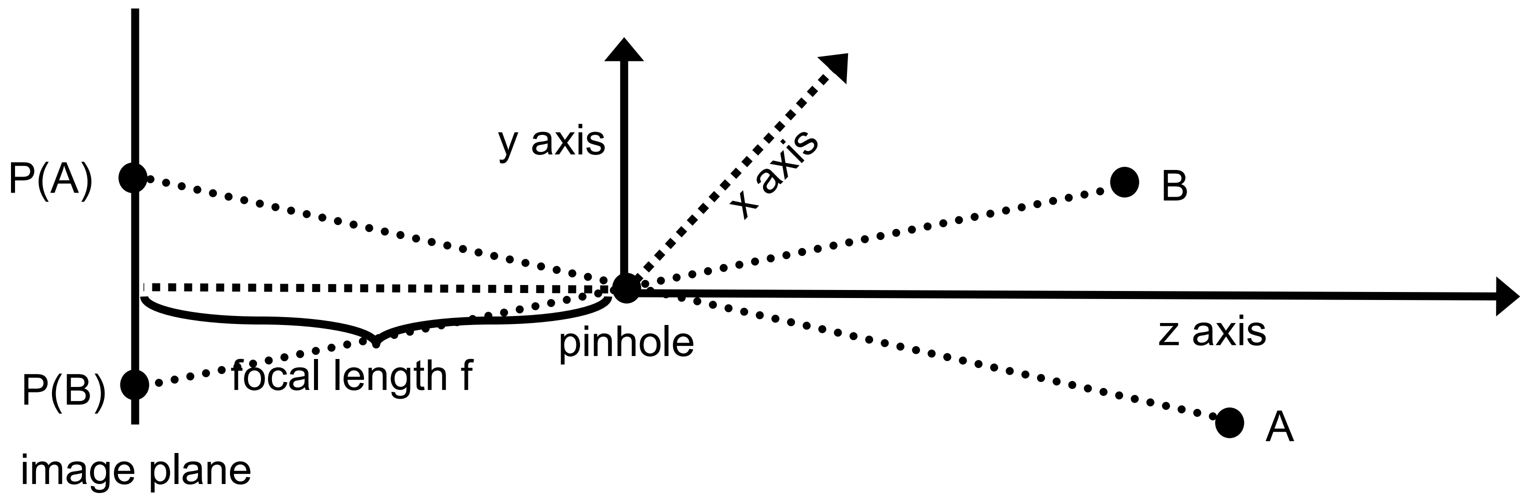
- World coordinate system (3D):
  - Pinhole is at location t, and at orientation R.
- Camera coordinate system (3D):
  - Pinhole is at the origin.
  - The camera faces towards the positive side of the z axis.

Different Coordinate Systems



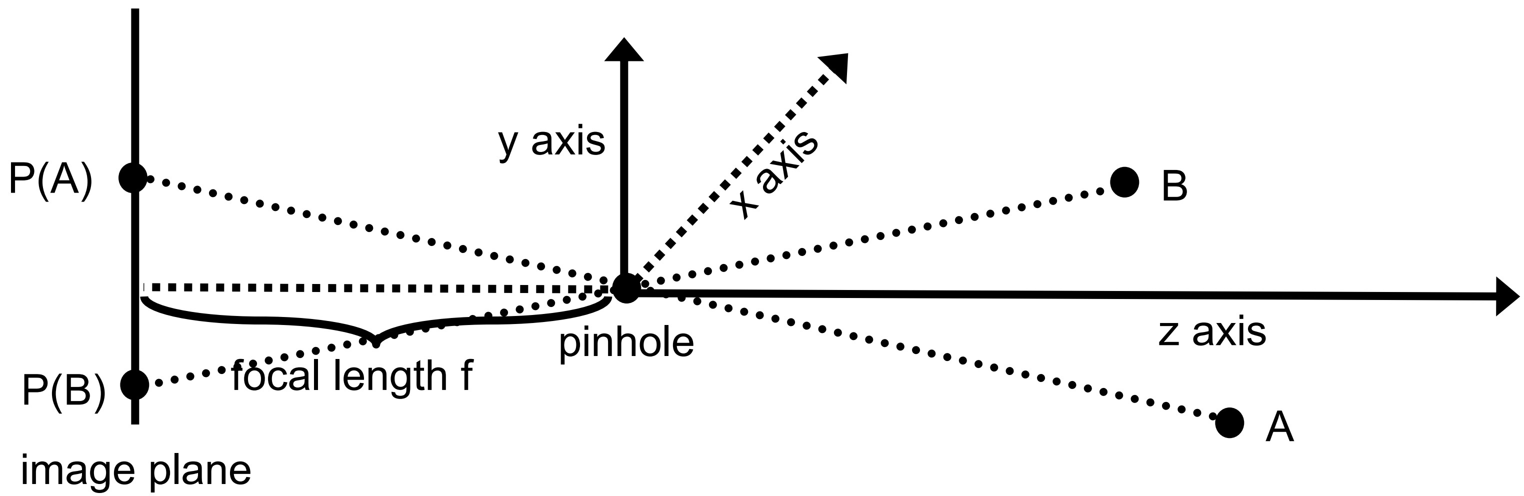
- Normalized image coordinate system (2D):
  - Coordinates on the image plane.
    - The (x, y) values of the camera coordinate system.
    - We drop the z value (always equal to f, not of interest).
  - Center of image is (0, 0).
- Image (pixel) coordinate system (2D):
  - pixel coordinates.

Pinhole Model



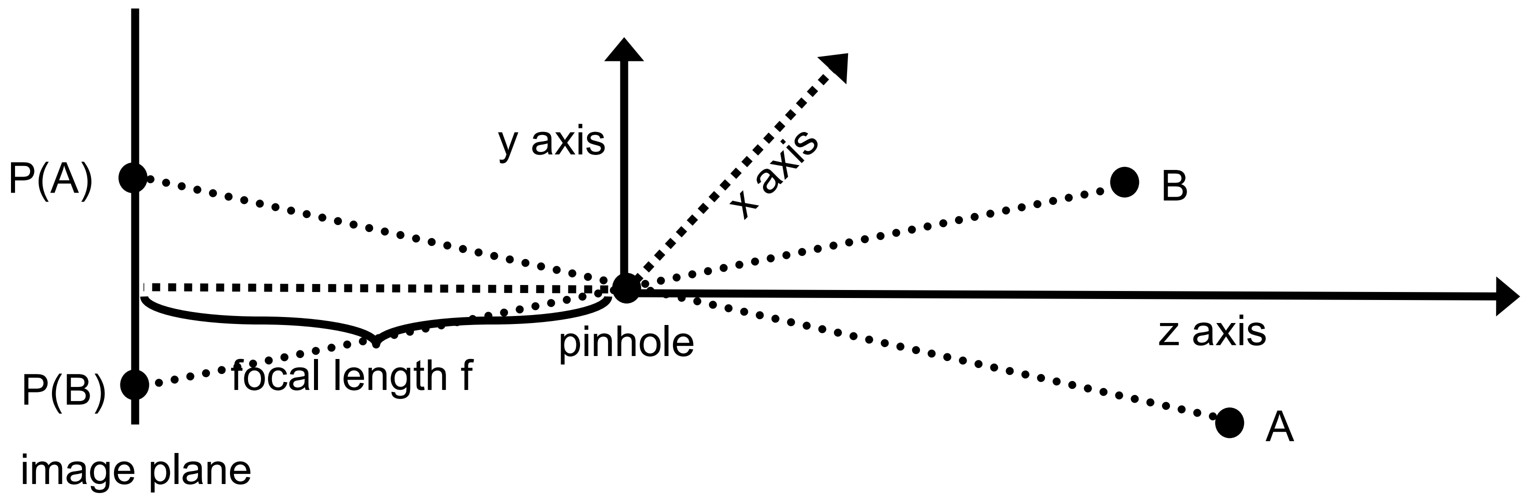
- A simple example:
  - Assume that world coordinates = camera coordinates.
  - Assume that the z axis points right, the y axis points up.
    - The x axis points away from us.
- If A is at position (A_x, A_y, A_z), what is P(A)?
  - Note: A is in world coordinates, P(A) is in normalized image coordinates.

Pinhole Model



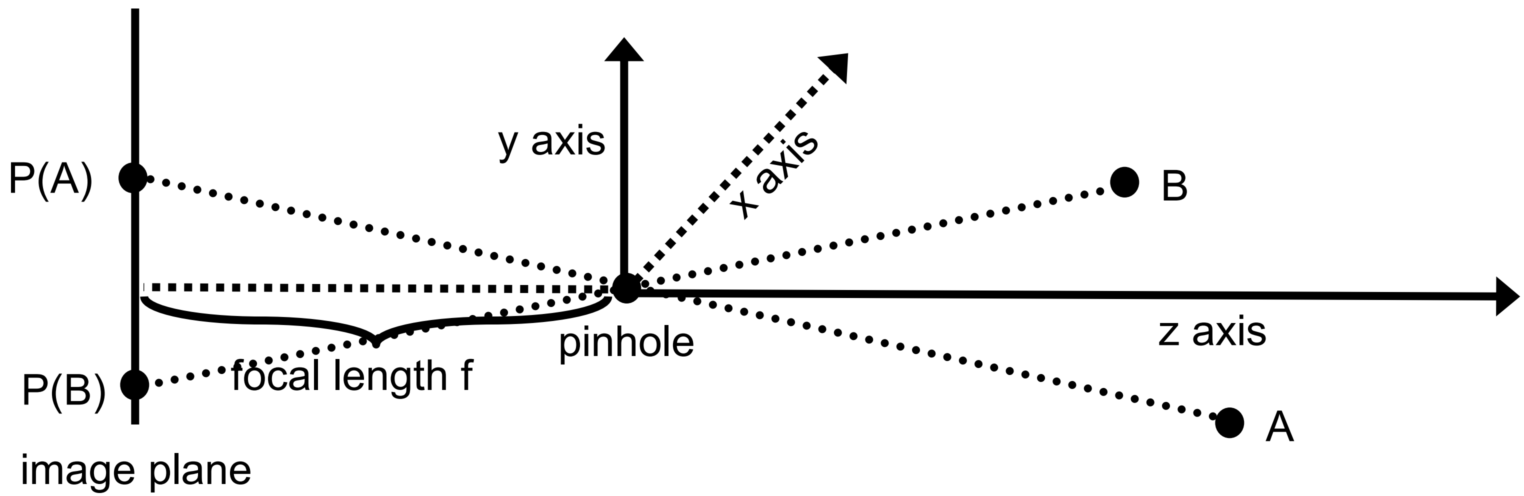
- P(A) = ($-A_x/A_z * f$, $-A_y/A_z * f$).
  - P(A) is **two-dimensional** (normalized image coordinates).
- This is a simple formula, because we chose a convenient coordinate system (world coordinates = camera coordinates).
- What happens if the pinhole is at (C_x, C_y, C_z)?

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

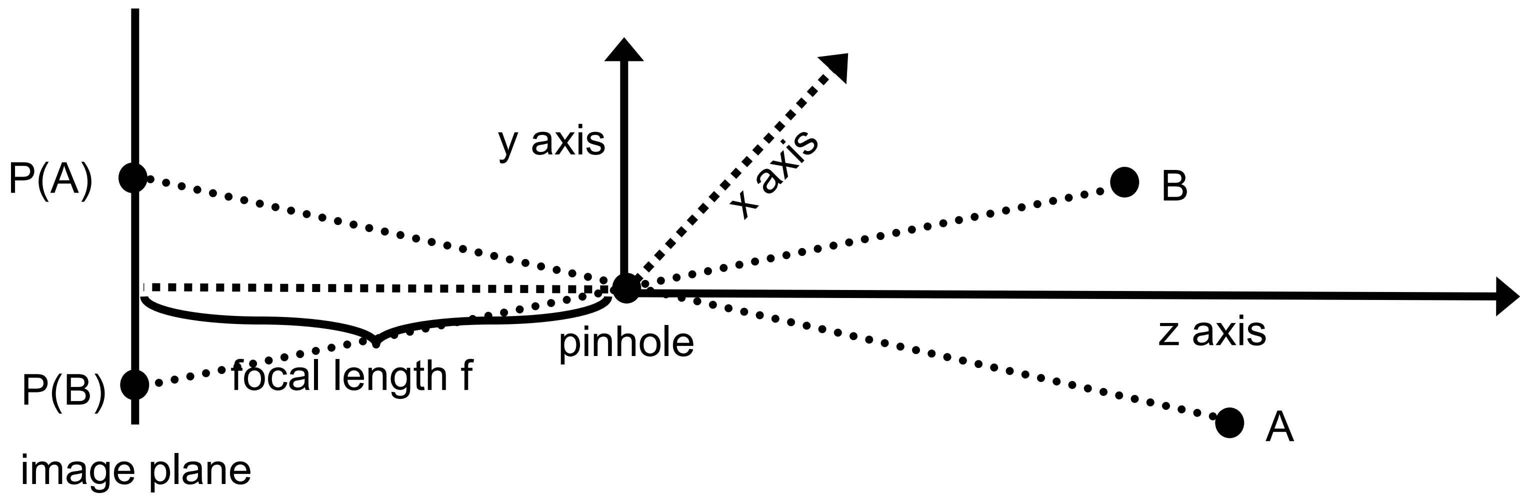


- If the pinhole is at (C_x, C_y, C_z):
  - P(A) = $(-(A_x-C_x)/(A_z-C_z) * f, -(A_y-C_y)/(A_z-C_z) * f)$.
- The concept is simple, but the formulas are messy.
- Formulas get a lot more messy in order to describe arbitrary camera placements.
  - We also need to allow for rotations.
- We simplify notation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

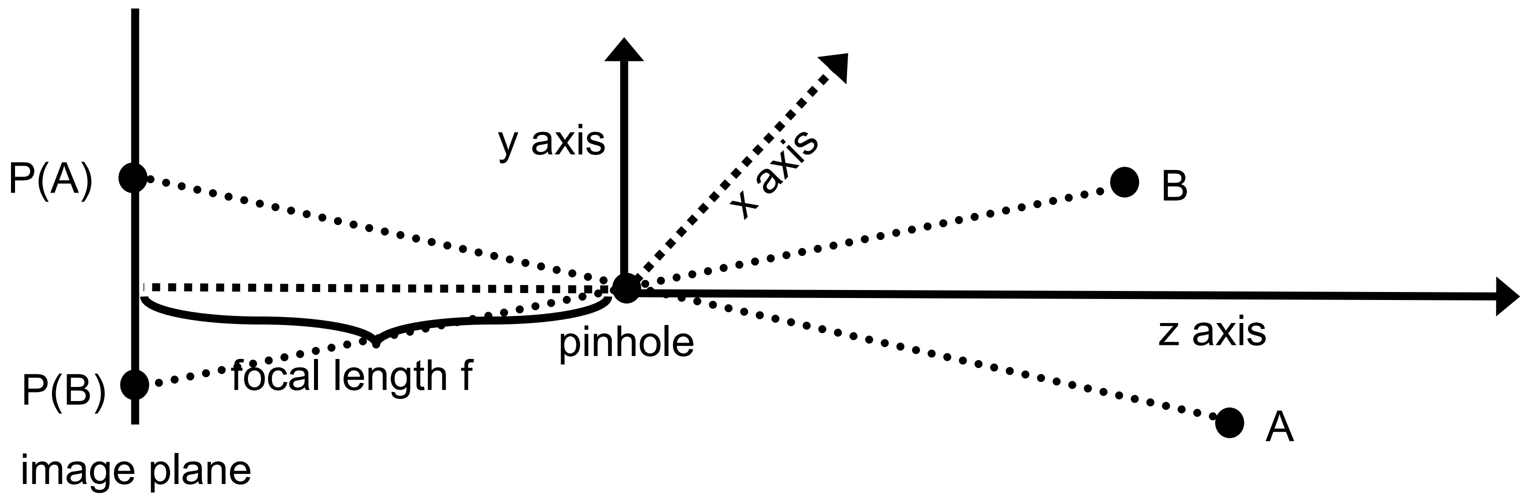
Revisiting Simple Case



- 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



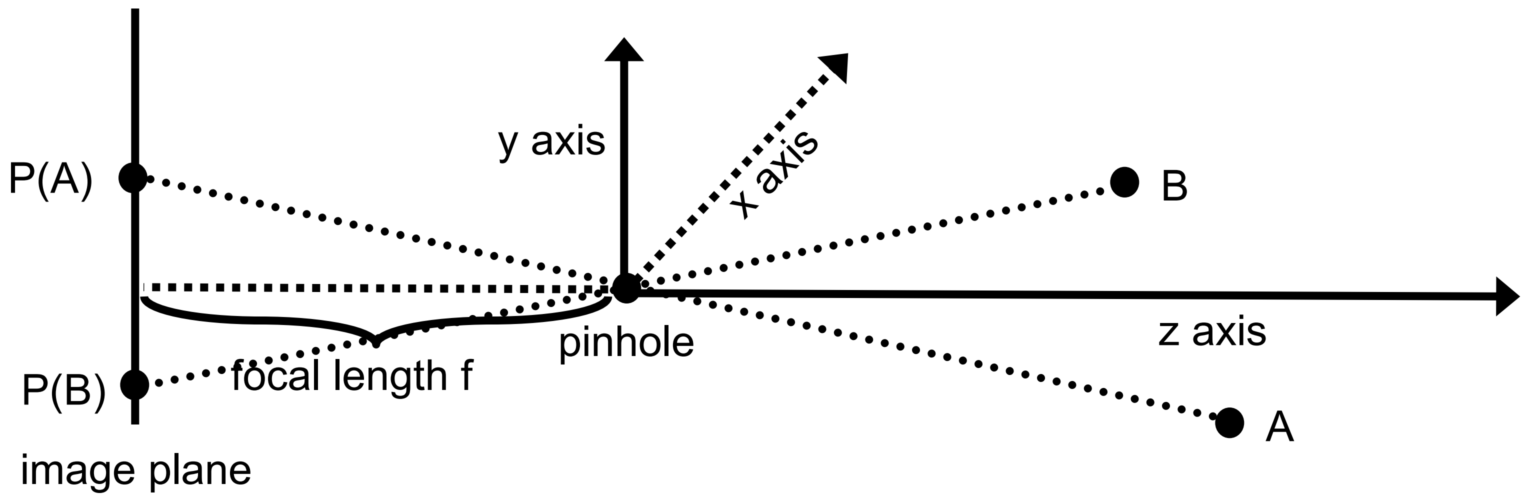
- World coordinates = camera coordinates.
- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $P(A) = \begin{pmatrix} (-A_x/A_z) * f \\ (-A_y/A_z) * f \\ 1 \end{pmatrix}$. Then:

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

Why?

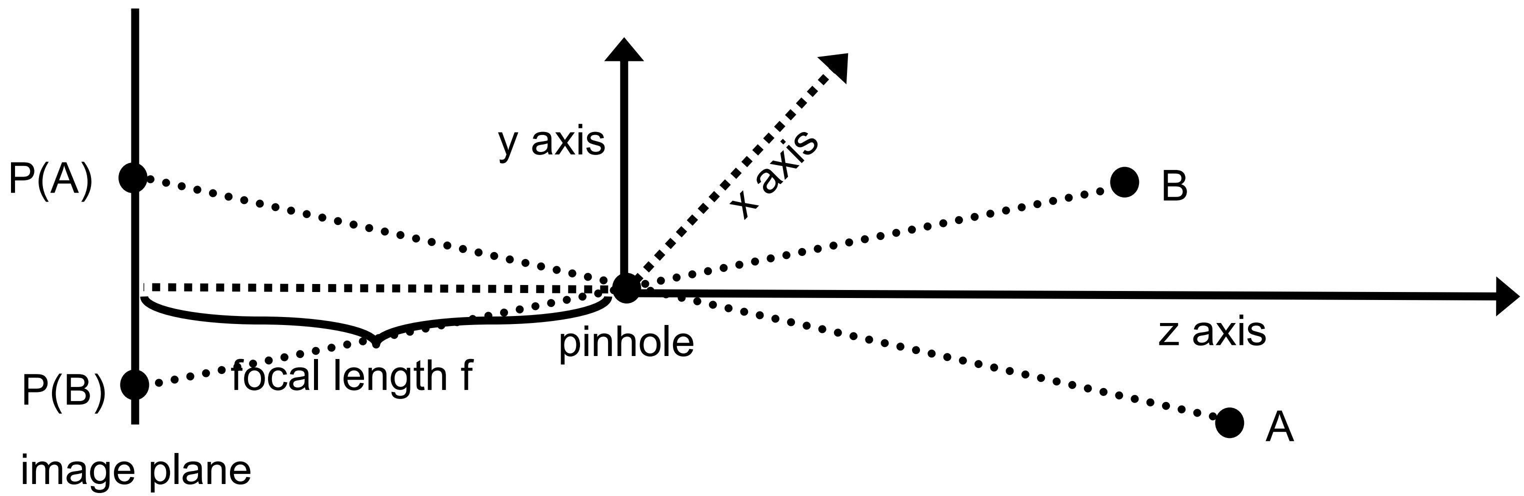
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix} \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix} = \begin{pmatrix} A_x \\ A_y \\ -A_z/f \end{pmatrix} = \begin{pmatrix} (-A_x/A_z) * f \\ (-A_y/A_z) * f \\ 1 \end{pmatrix}$$

Revisiting Simple Case



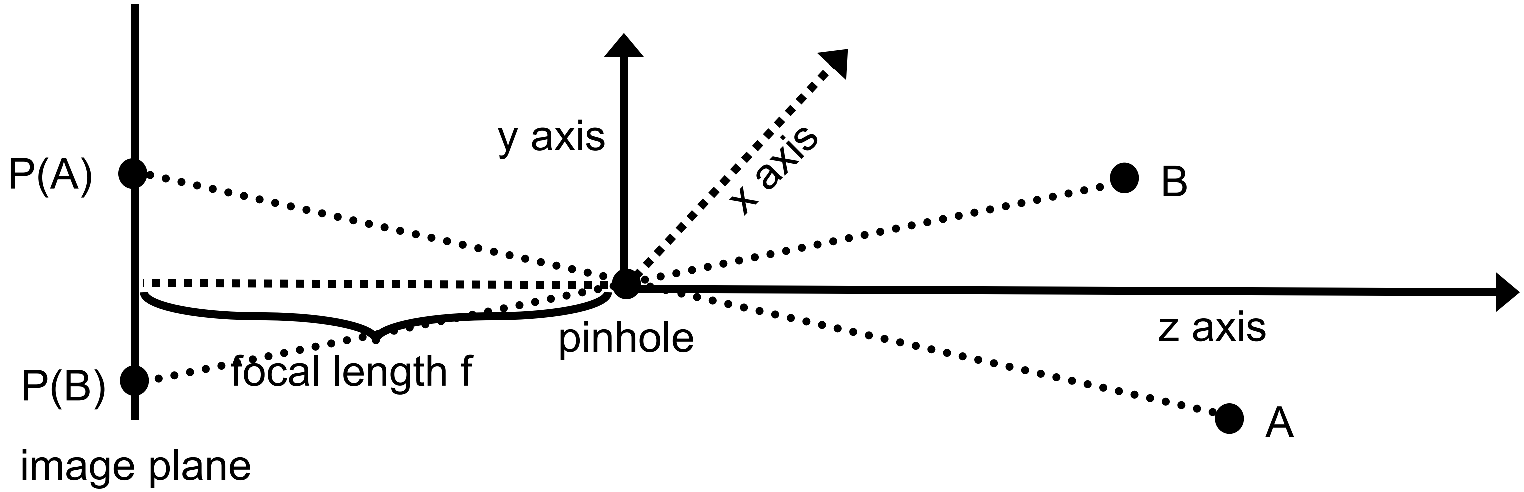
- World coordinates = camera coordinates.
- Let $A = \begin{bmatrix} A_x \\ A_y \\ A_z \\ 1 \end{bmatrix}$. $P(A) = \begin{pmatrix} (-A_x/A_z) * f \\ (-A_y/A_z) * f \\ 1 \end{pmatrix}$. Define $C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}$.
- Then: $P(A) = C_1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

Handling Camera Translation



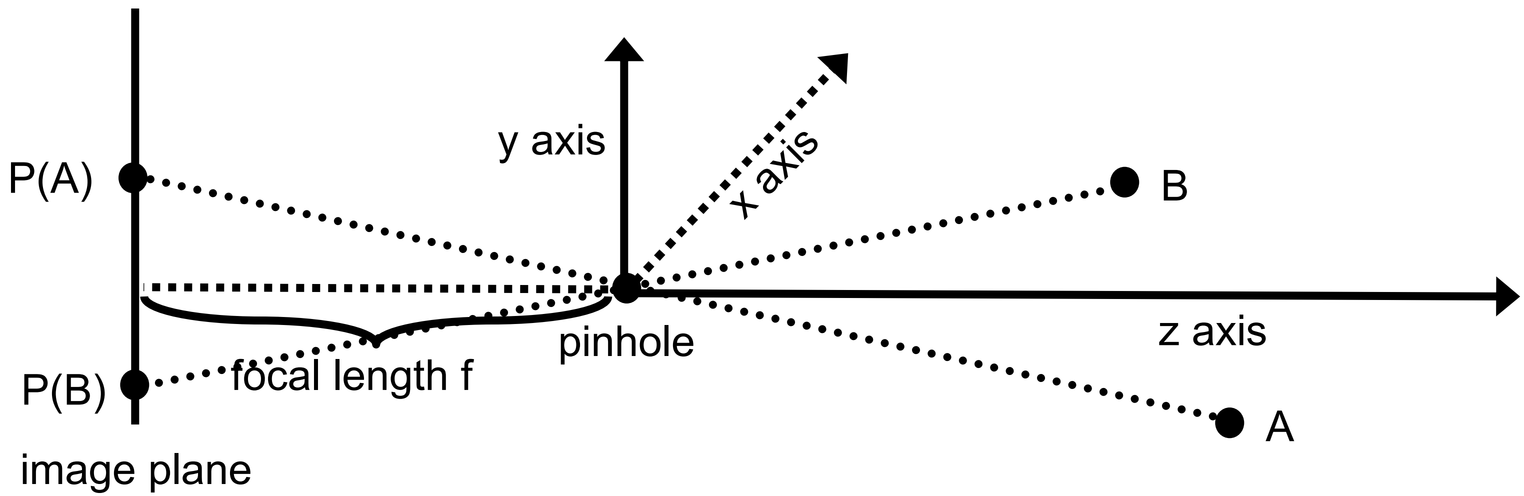
- Suppose camera is at (C_x, C_y, C_z).
  - Camera coordinates and world coordinates are different.
- Define T(A) to be the transformation from world coordinates to camera coordinates.
- If we know T(A), what is P(A)?

Handling Camera Translation



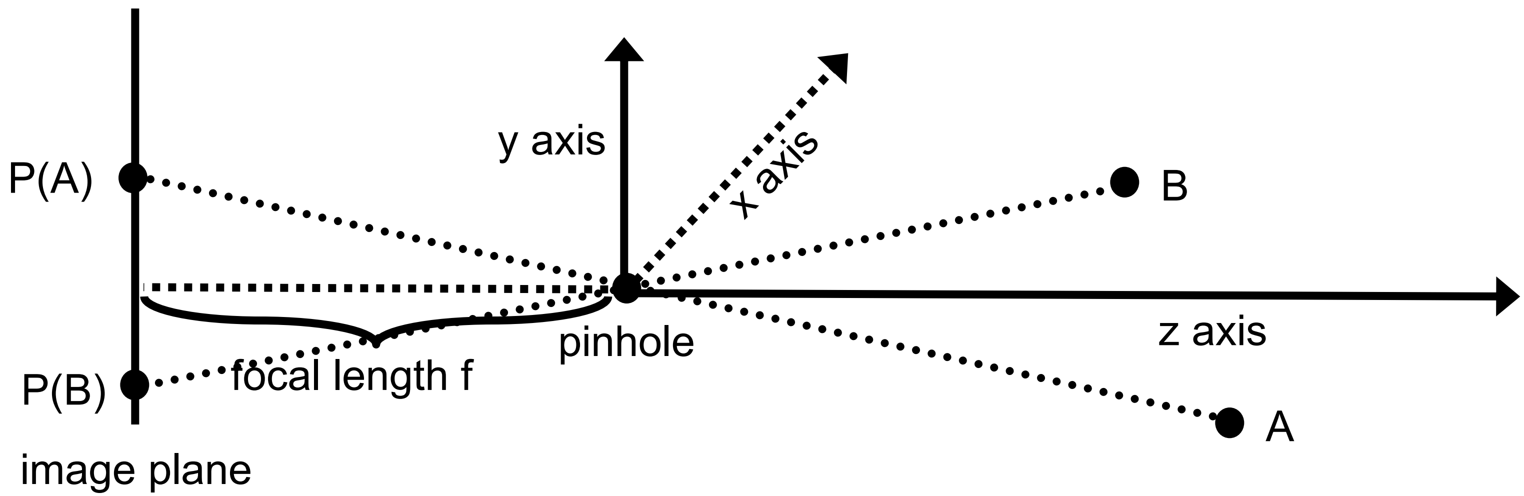
- Suppose camera is at (C_x, C_y, C_z).
  - Camera coordinates and world coordinates are different.
- Define T(A) to be the transformation from world coordinates to camera coordinates.
- If we know T(A), what is P(A)?
- P(A) = C_1 * T(A).

Handling Camera Translation



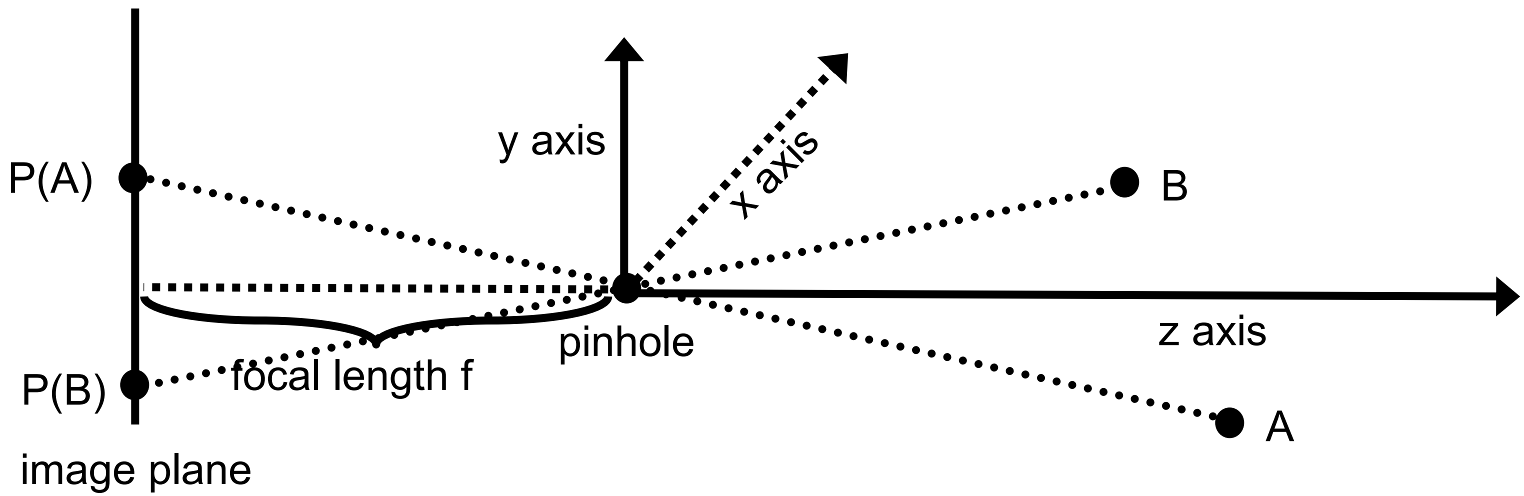
- Suppose camera is at (C_x, C_y, C_z).
- Define T(A) to be the transformation from world coordinates to camera coordinates.
- If we know T(A), P(A) = C_1 * T(A).
- How can we write T(A) as a matrix multiplication?

Handling Camera Translation



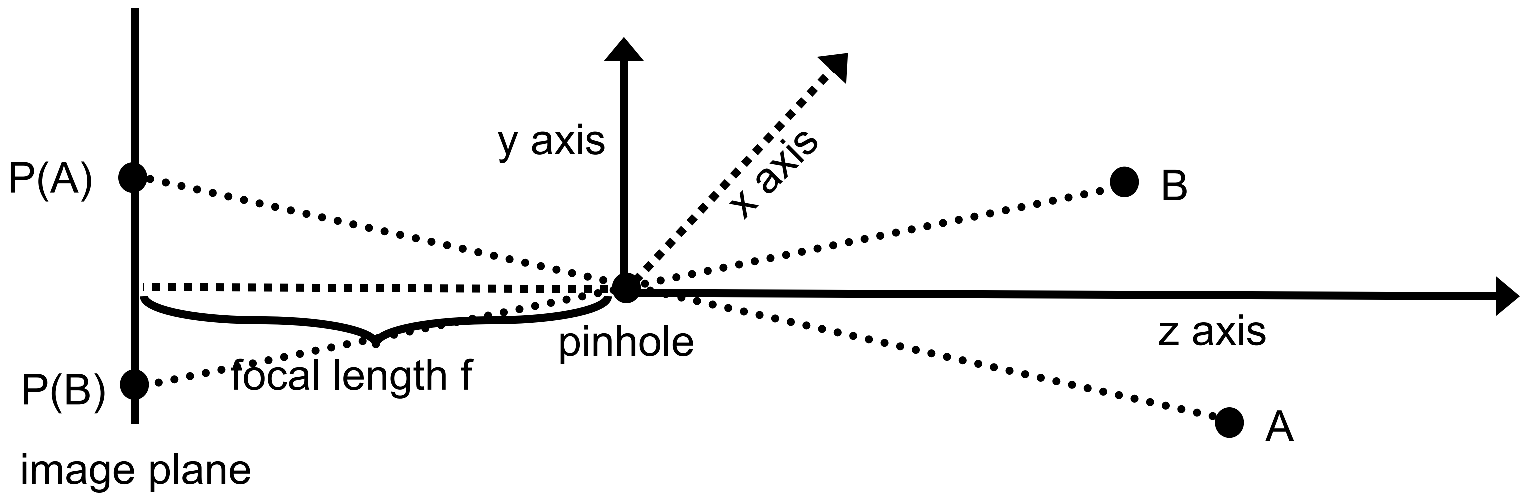
- First of all, how can we write T(A) in the most simple form, in non-homogeneous coordinates? (Forget about matrix multiplication for a second).

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

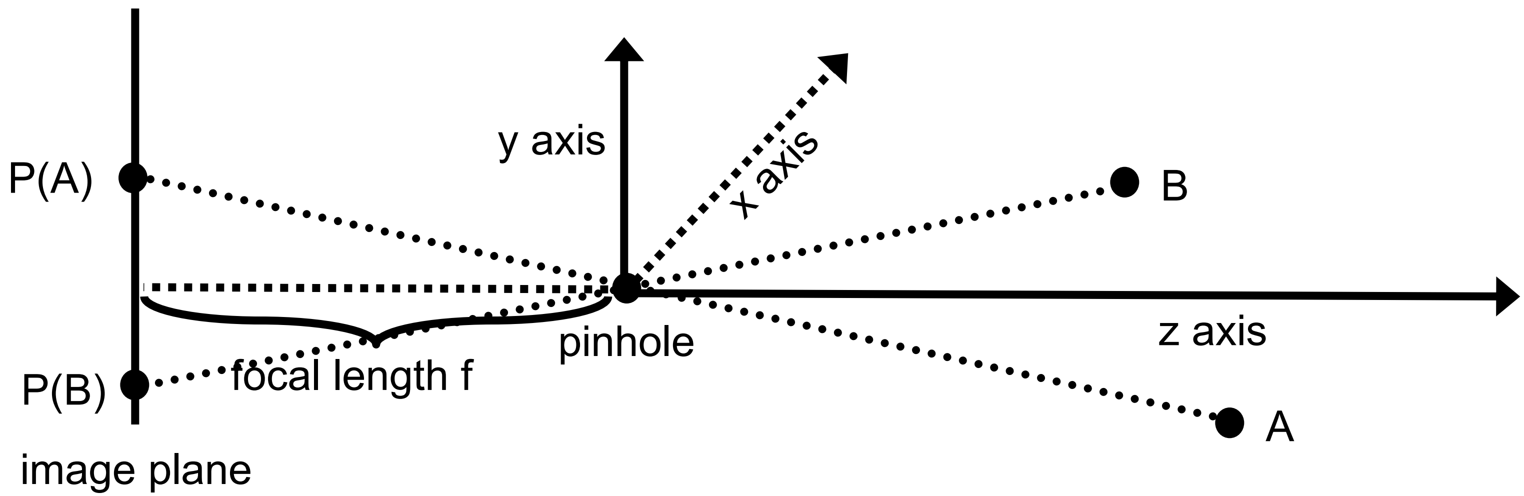


- $T(A) = (A_x, A_y, A_z) – (C_x, C_y, C_z)$.

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

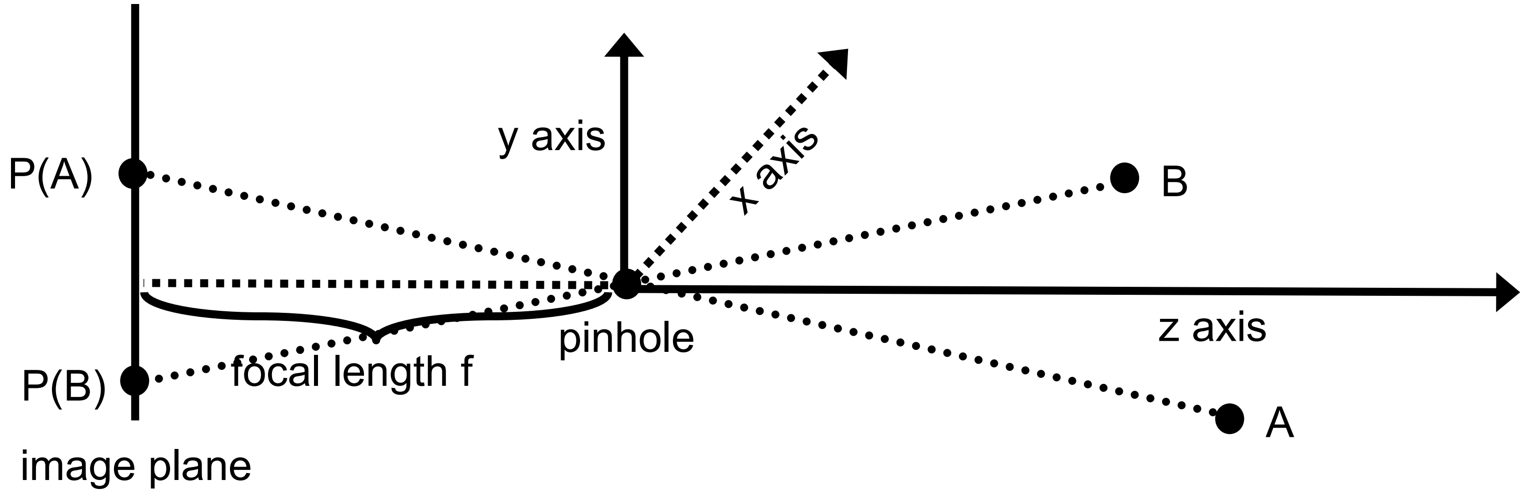
- Homogeneous coordinates allow us to represent translation as matrix multiplication.

Handling Camera Translation



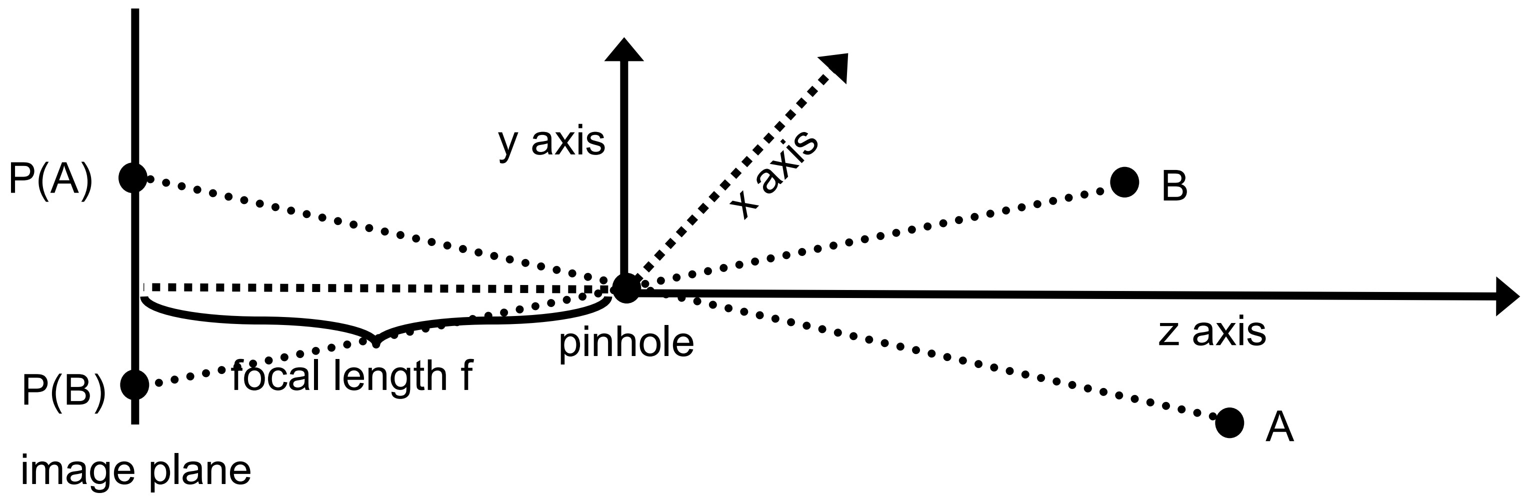
- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$. Define $C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}$, $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- Then: $P(A) = C_1 * T * A$.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

Handling Camera Translation



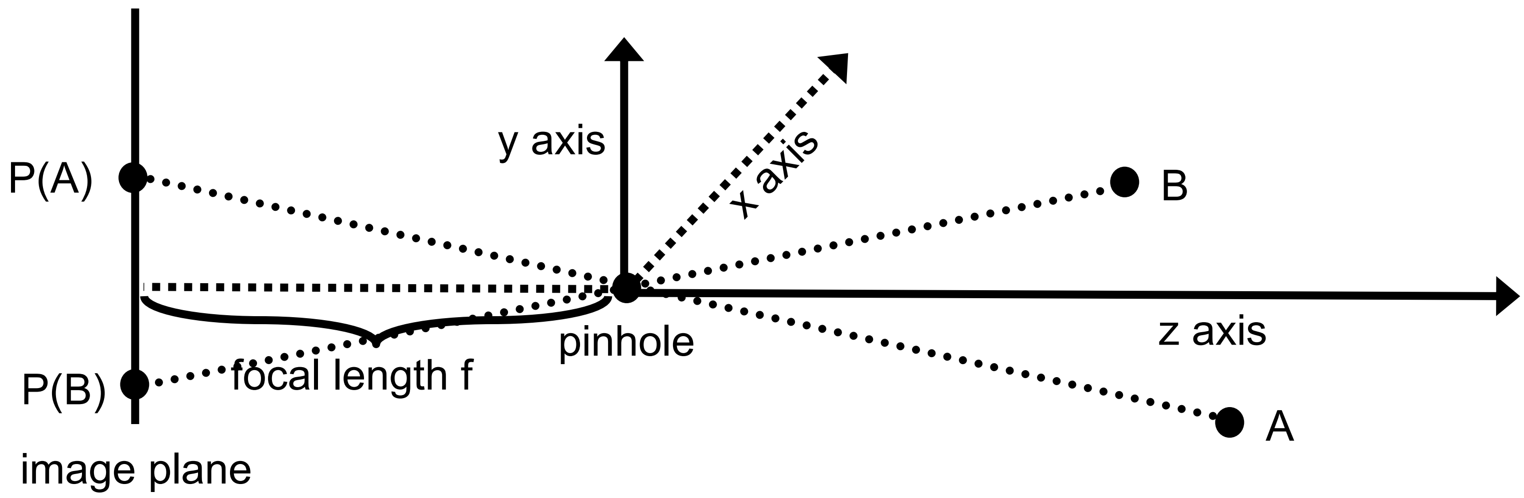
- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$. Define $C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}$, $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- Then: $P(A) = C_1 * T * A$.
- Why is C_1 of size 3x4 and T of size 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 (2D) 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

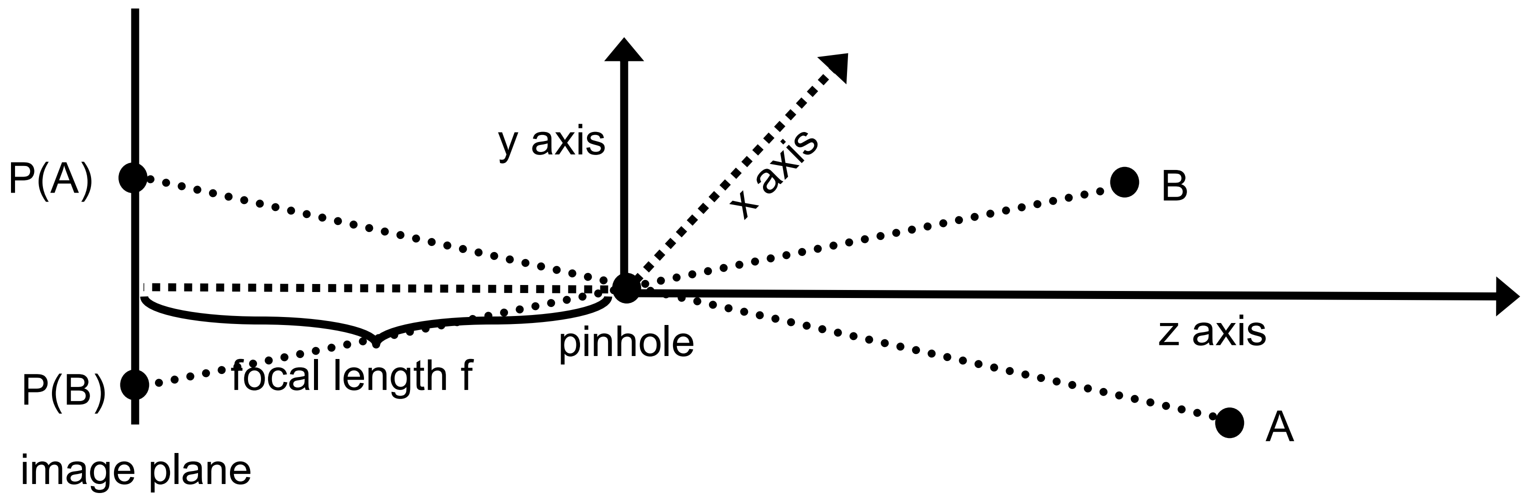
- In non-homogeneous coordinates, rotation of A around the origin can be represented as R*A.
  - R: 3x3 rotation matrix.
- How does camera rotation affect the image?
  - It changes the viewing direction.
    - Determines what is visible.
  - It changes the image orientation.
    - Determines what the “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.

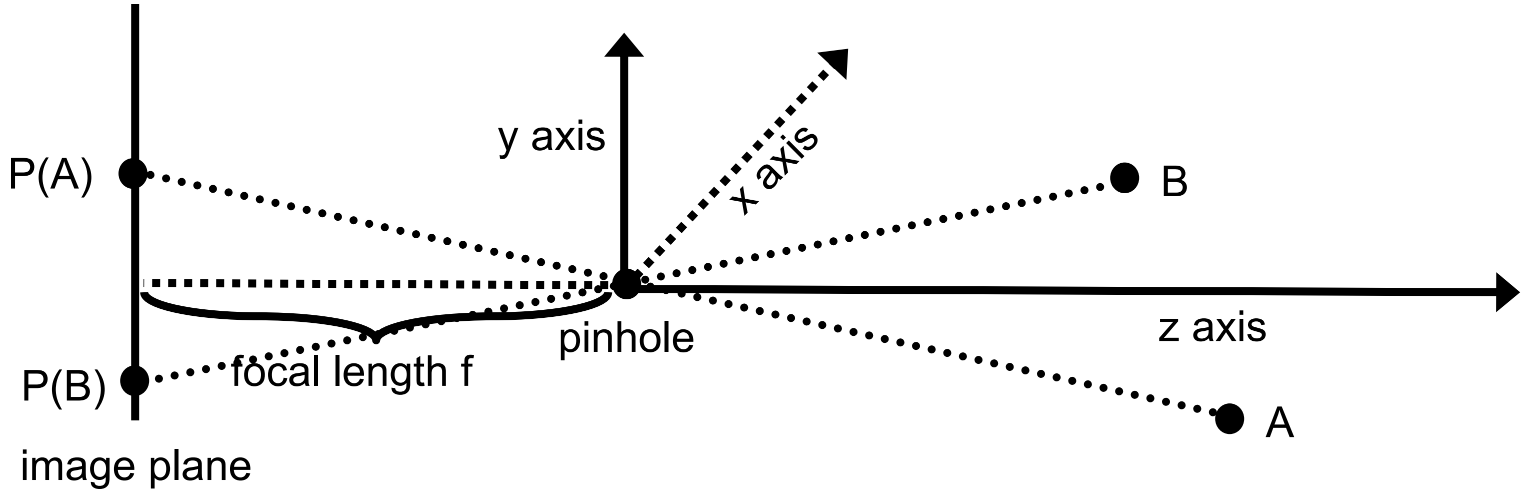
$$\underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{pmatrix}}_{R_x} \quad \underbrace{\begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{pmatrix}}_{R_y} \quad \underbrace{\begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{R_z}$$

Handling Camera Rotation



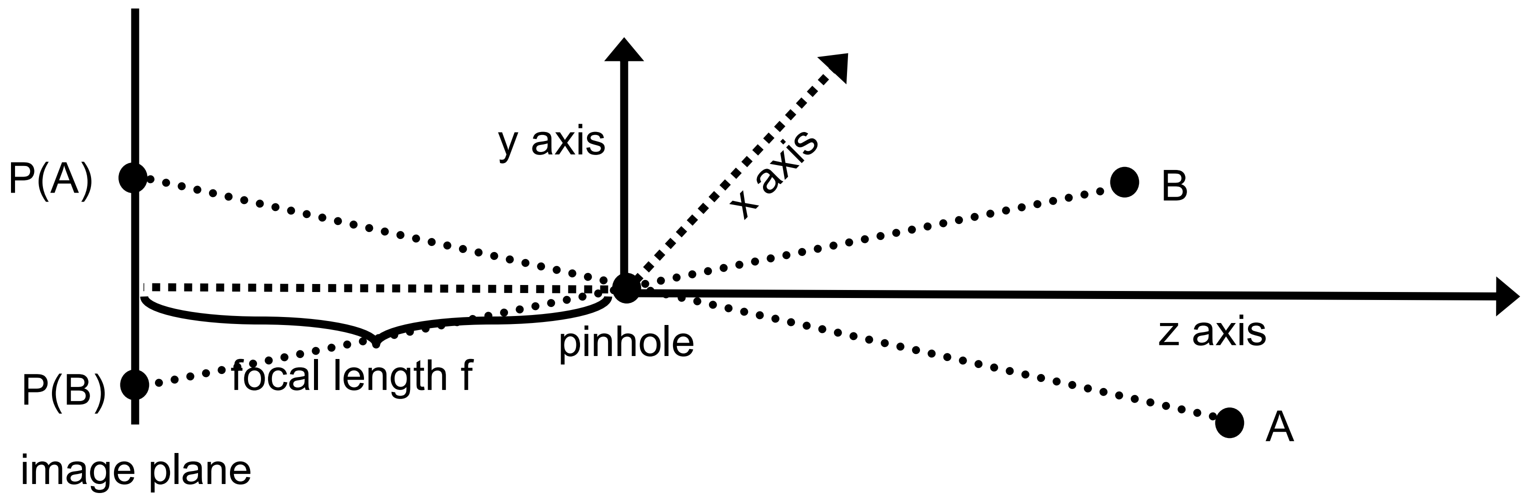
- In homogeneous coordinates, rotation of A around the origin can be represented as R*A.
  - R: 4x4 rotation matrix.
- Let R' = $\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$. Then, R = $\begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\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

Handling Camera Rotation



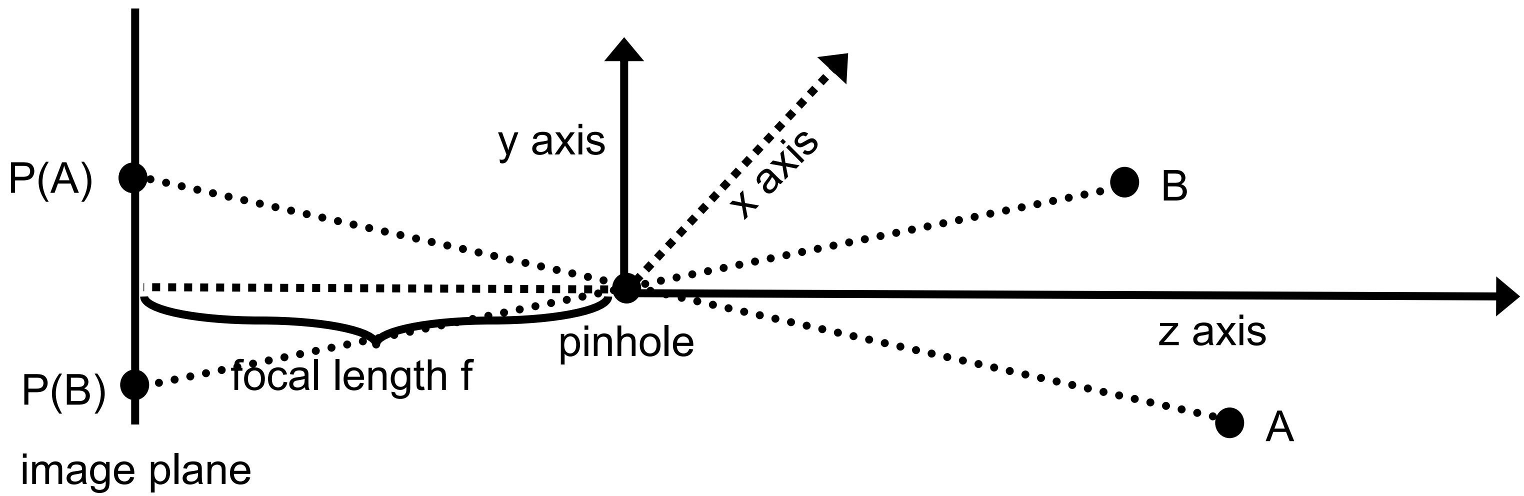
- Let $R' = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$. Then, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.
- What is the right way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

Handling Camera Rotation



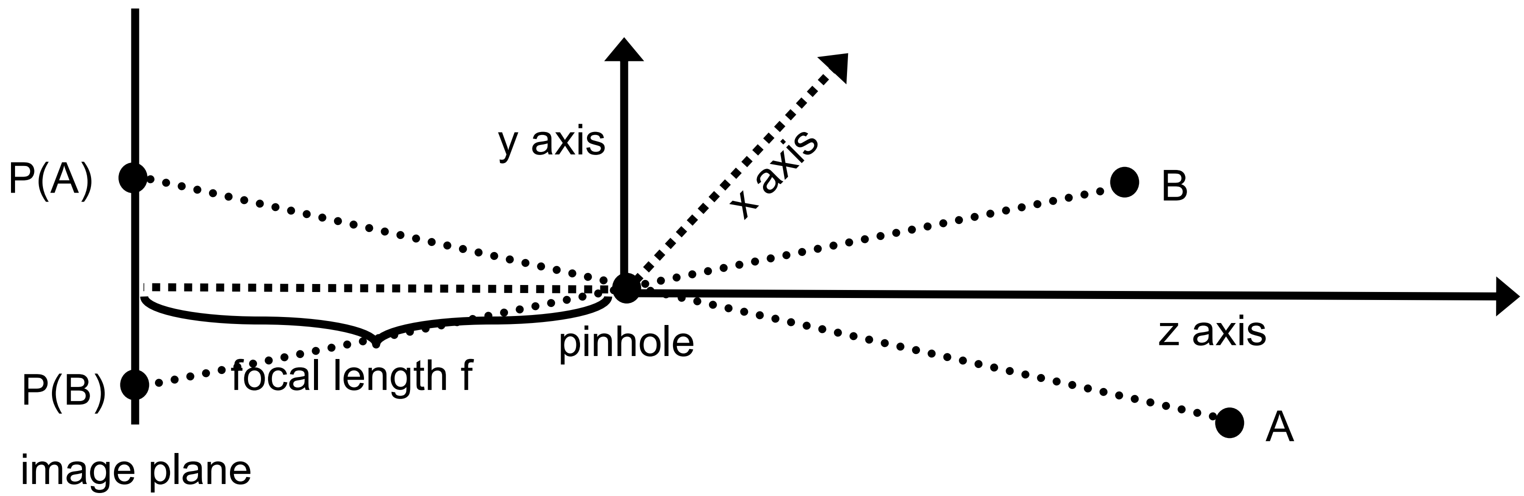
- Let $R' = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$. Then, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.
- What is the right way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

Handling Camera Rotation



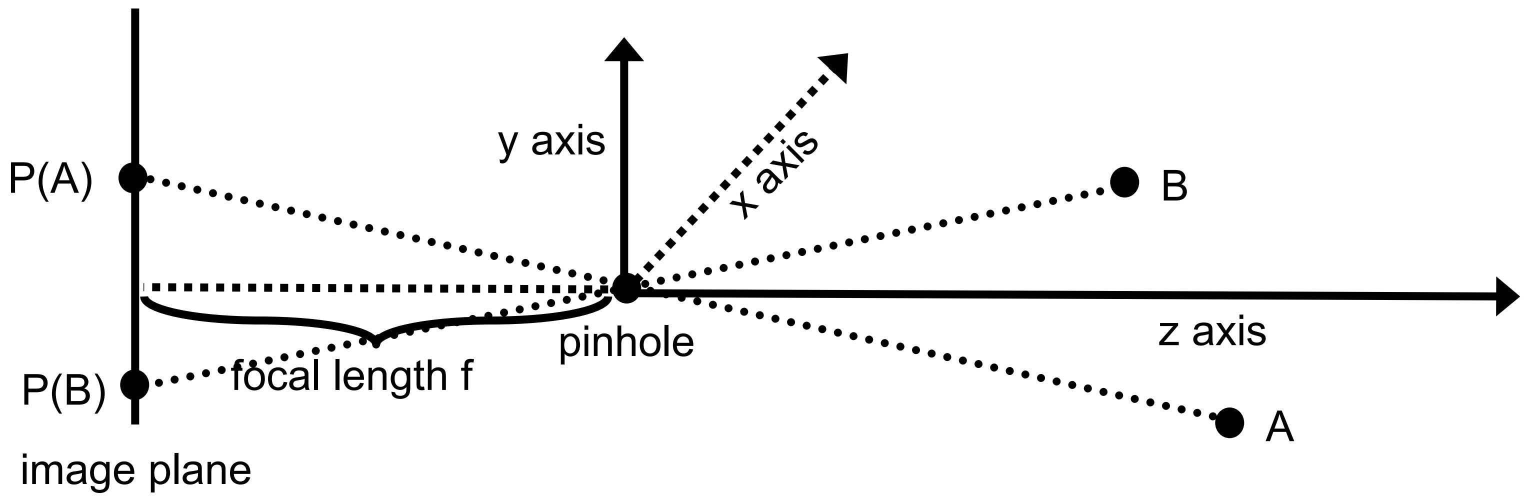
- Let $R' = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$. Then, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.
- Is it true that $P(A)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

Handling Camera Rotation



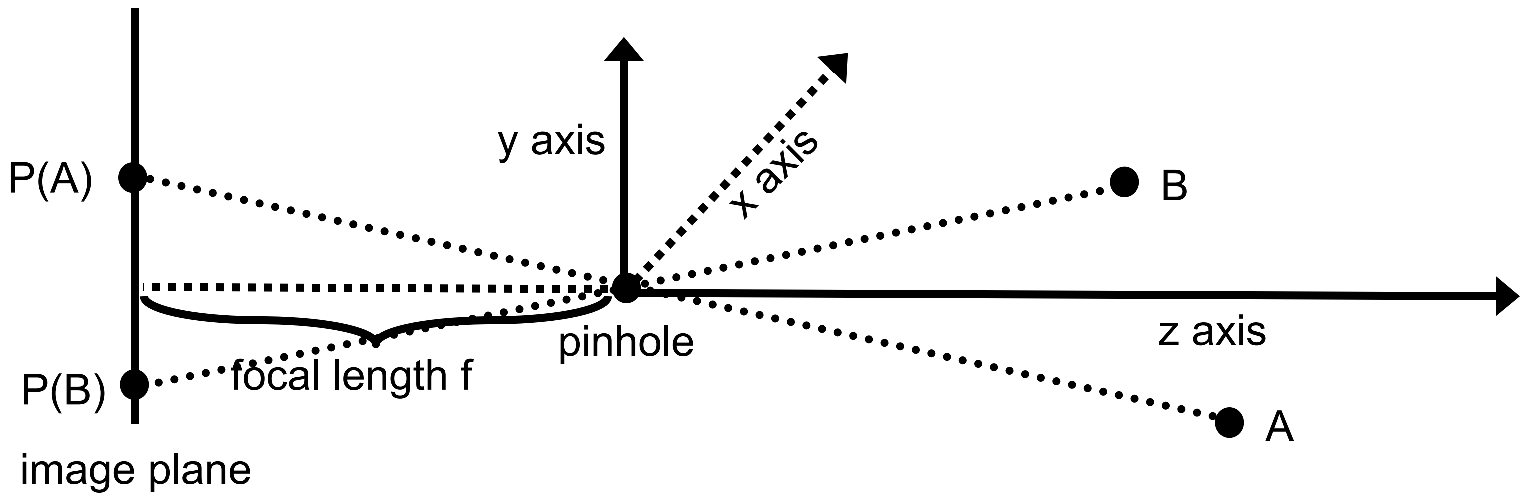
- Let $R' = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$. Then, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.
- Is it true that $P(A) = C_1 * T * R * A$?
  - 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



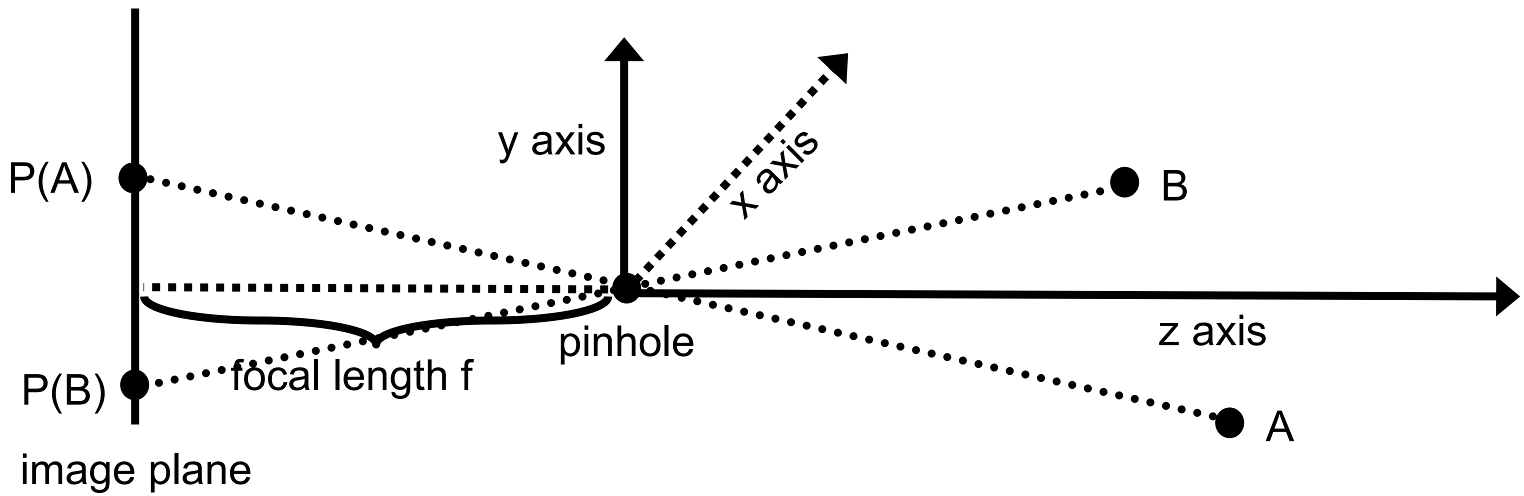
- Let $R' = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$. Then, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.
- $P(A) = C_1 * R * T *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

Handling Scale



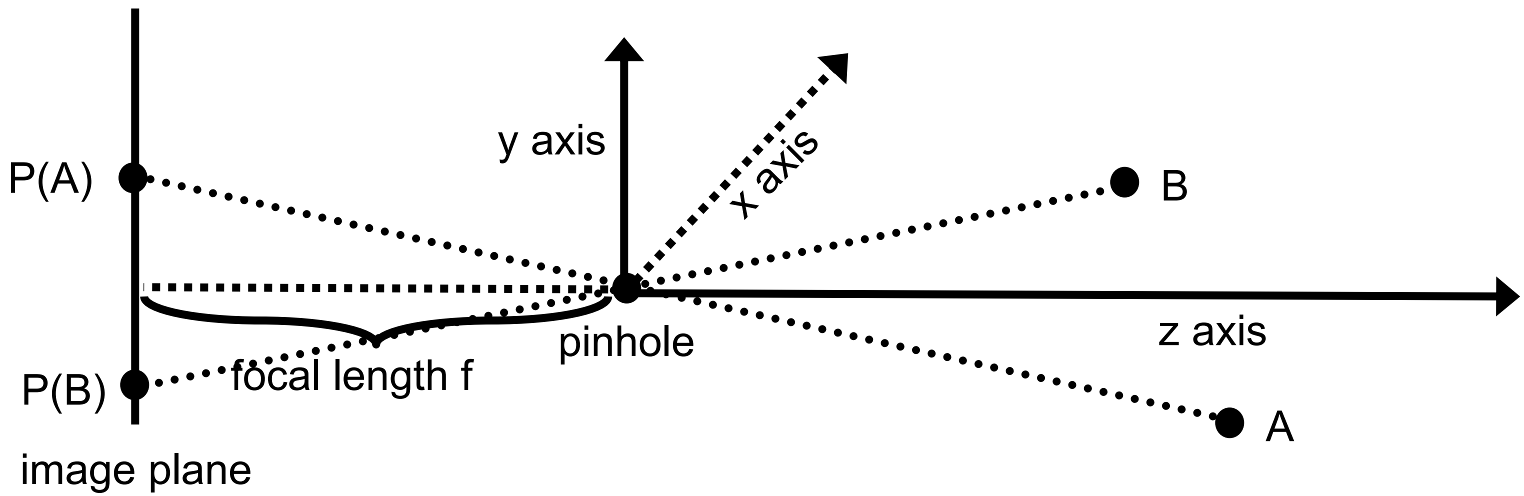
- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}$, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$,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

Handling Scale



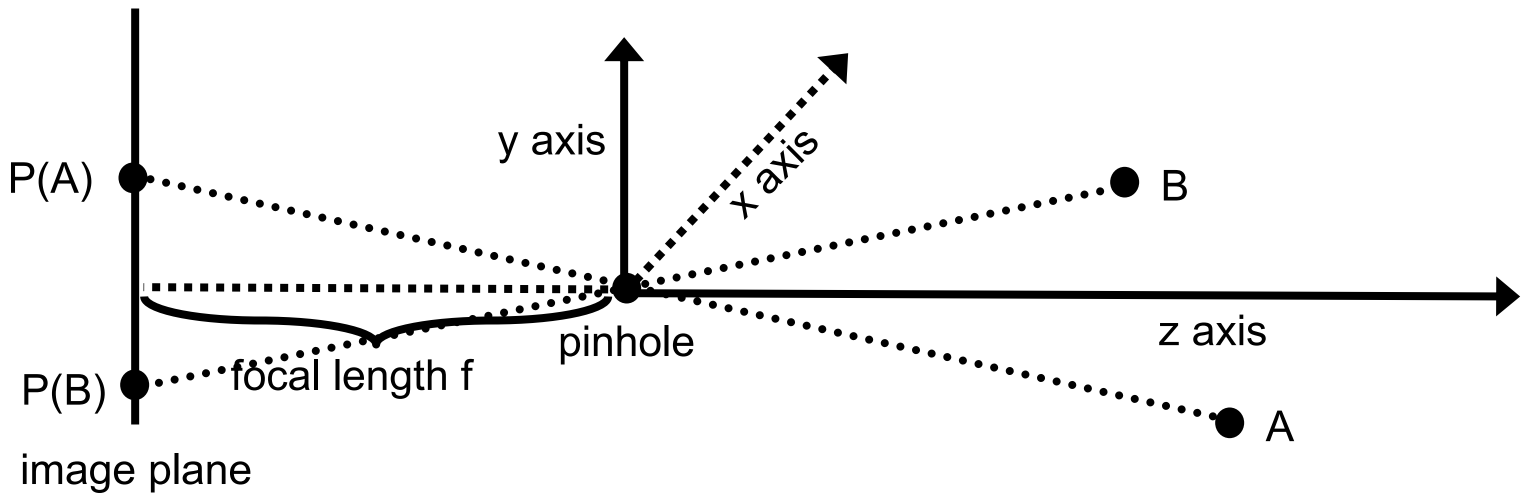
- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}$, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$,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

Handling Scale



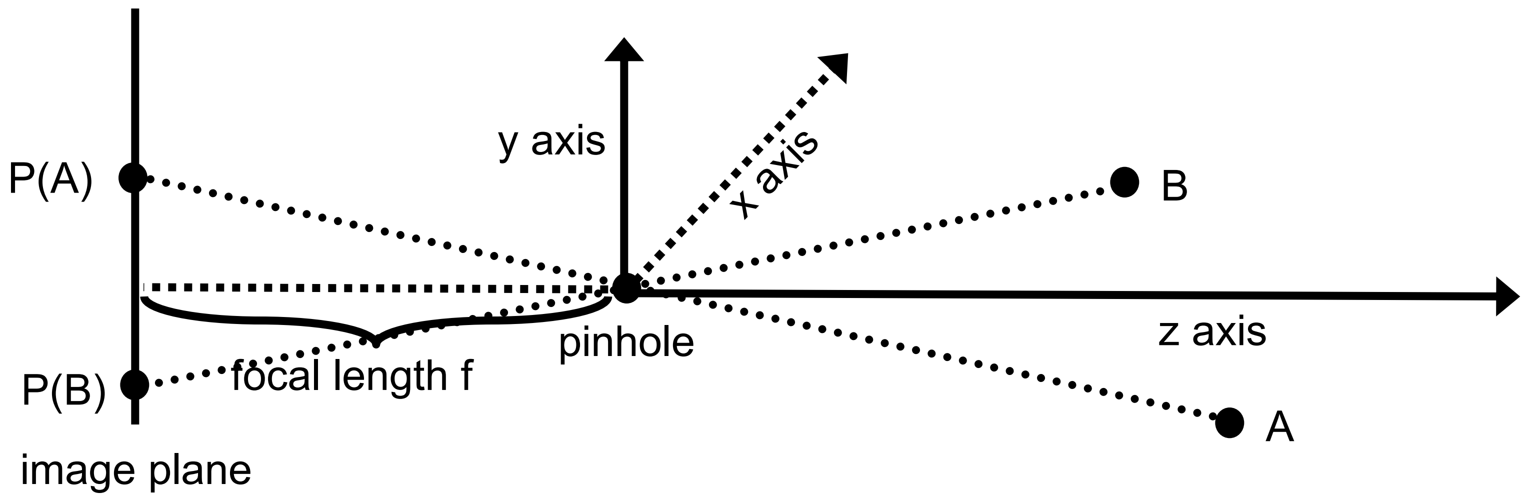
- Let $A = \begin{pmatrix} A_x \\ A_y \\ A_z \\ 1 \end{pmatrix}$, $C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/f & 0 \end{pmatrix}$, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$.
  - $P(A) = C_1 * R * T * A$ 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, $S_x = S_y$.
  - 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, $S_x = S_y$.
  - 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?

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

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?

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

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

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 = $\begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.
- 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 $A = \begin{bmatrix} A_x \\ A_y \\ A_z \\ 1 \end{bmatrix}$, $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $T = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $H = \begin{pmatrix} -fS_x & 0 & u_0 & 0 \\ 0 & -fS_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.
- $\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = H * R * T * A.$
- 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:

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

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

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

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

- 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:

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

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

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

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

- 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

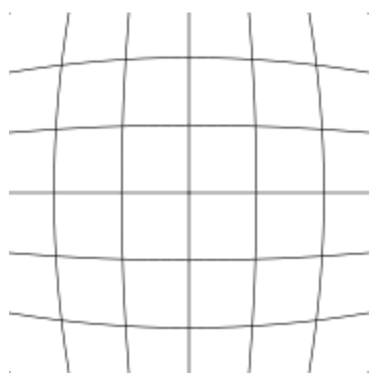
- Each solution produces an error for each equation.
- Sum-of-squared-errors is the measure we use to evaluate a solution.
- The least squares solution is the solution that minimizes the sum-of-squared-errors measure.
- Finding the least-squares solution to a set of linear equations is mathematically 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

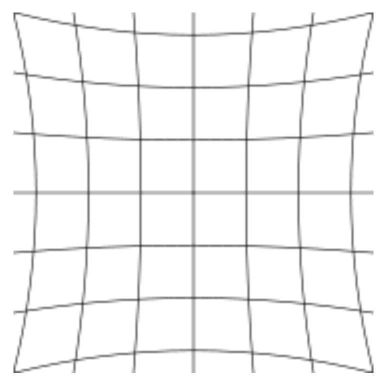
Producing World Coordinates

- 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