CSE 167:
Introduction to Computer Graphics
Lecture #3: Projection

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2012

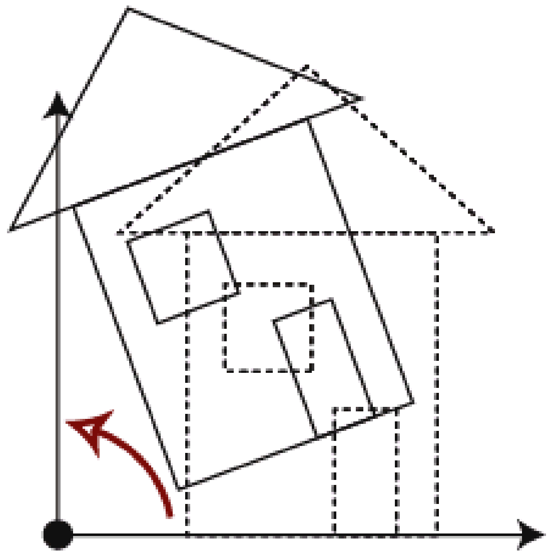
Announcements

- Project 1 due tomorrow (Friday), presentation in CSE lab 260, starting at 1:30pm
  - Have source code and executable ready for us. We might ask questions about the code.
  - List your name on the whiteboard once you get to the lab. Homework will be graded in this order. If you have a class that starts at 2, please put a star next to your name so we can give you priority.
- Project 2 is due Friday October 12th
  - Homework tutorial by Sid on Monday at 2:30pm in lab 260

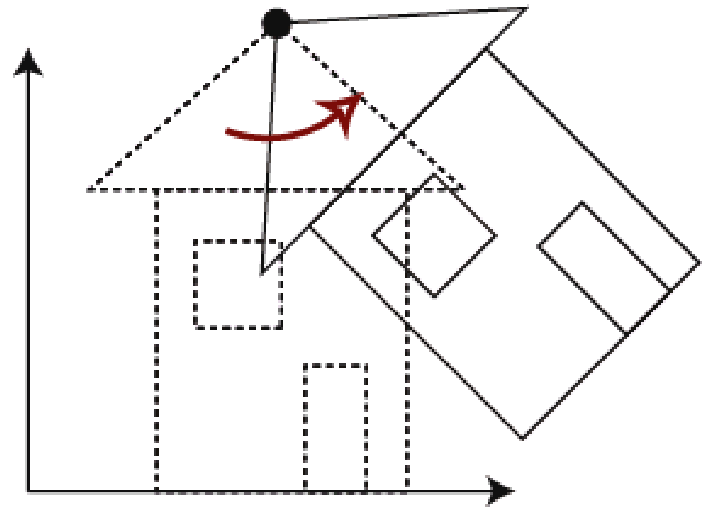
Lecture Overview

- Concatenating Transformations
- Coordinate Transformation
- Typical Coordinate Systems
- Projection

How to rotate around a Pivot Point?

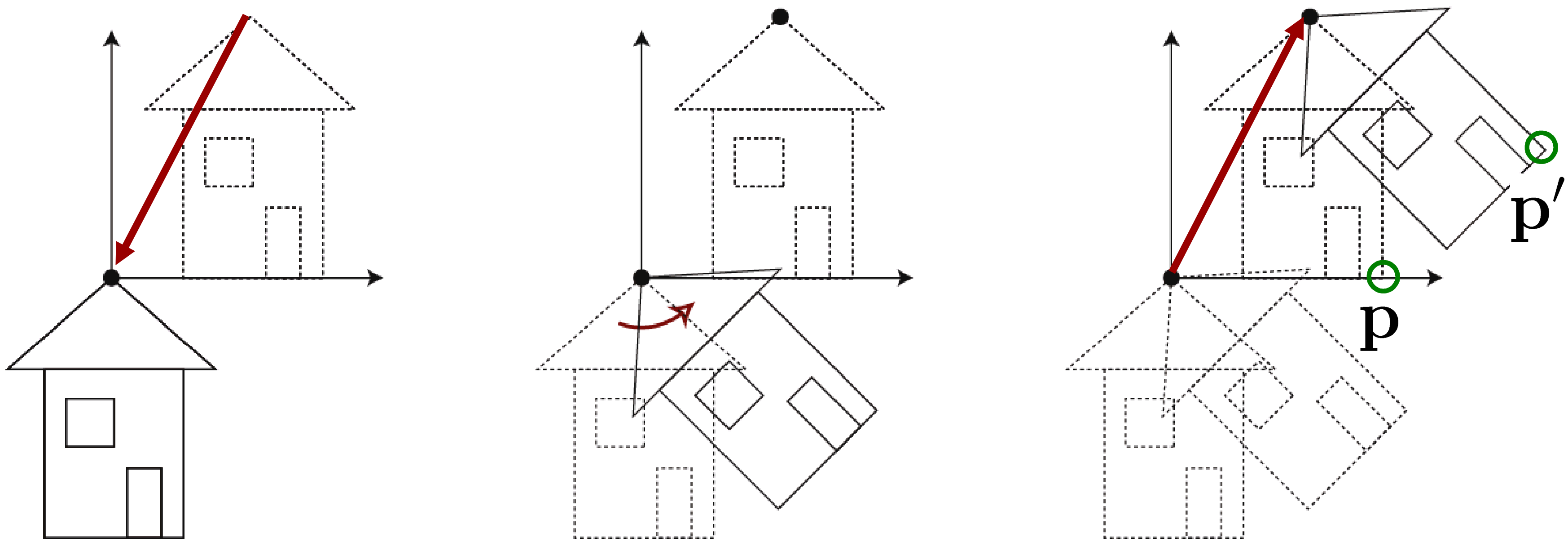


Rotation around origin:
p' = R p



Rotation around pivot point:
p' = ?

Rotating point p around a pivot point



1. Translation T   2. Rotation R   3. Translation T^{-1}

$$p' = T^{-1} R\, T\, p$$

Concatenating transformations

- Given a sequence of transformations $M_3M_2M_1$

$$\mathbf{p}' = \mathbf{M}_3\mathbf{M}_2\mathbf{M}_1\mathbf{p}$$

$$\mathbf{M}_{total} = \mathbf{M}_3\mathbf{M}_2\mathbf{M}_1$$

$$\mathbf{p}' = \mathbf{M}_{total}\mathbf{p}$$

- Note: associativity applies:

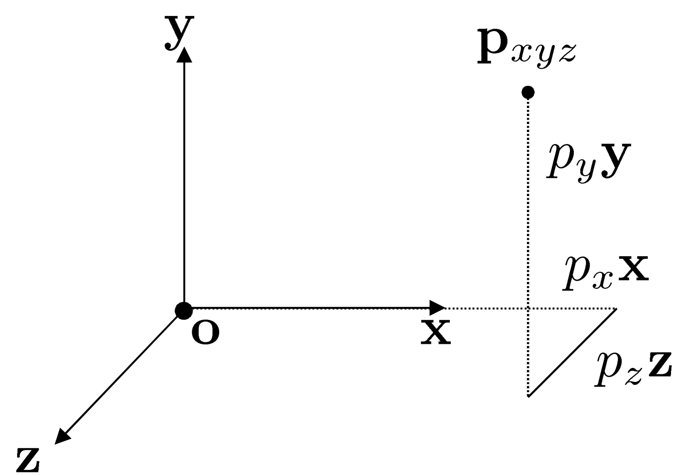
$$\mathbf{M}_{total} = (\mathbf{M}_3\mathbf{M}_2)\mathbf{M}_1 = \mathbf{M}_3(\mathbf{M}_2\mathbf{M}_1)$$

Lecture Overview

- Concatenating Transformations
- Coordinate Transformation
- Typical Coordinate Systems
- Projection

Coordinate System

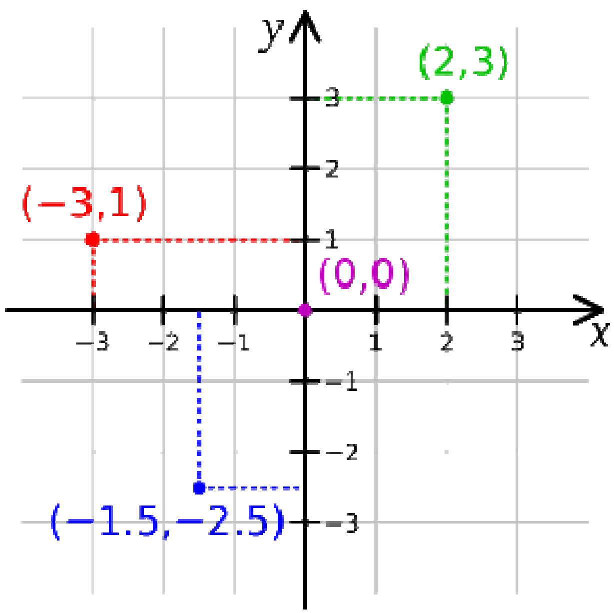
- Given point p in homogeneous coordinates: $\begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$
- Coordinates describe the point's 3D position in a coordinate system with basis vectors x, y, z and origin o:



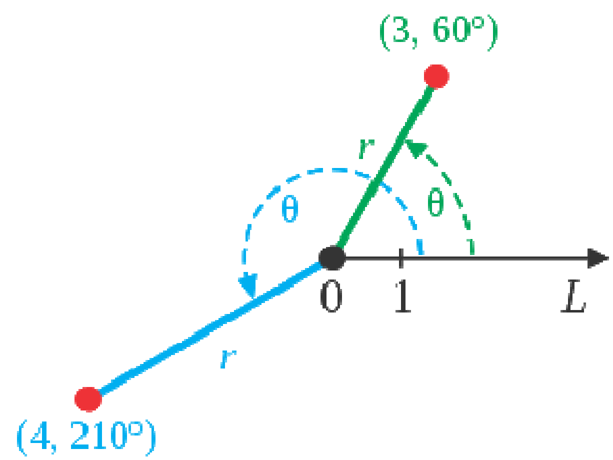
$$\mathbf{p}_{xyz} = p_x\mathbf{x} + p_y\mathbf{y} + p_z\mathbf{z} + \mathbf{o}$$

Example: Cartesian and Polar Coordinates

Cartesian Coordinates



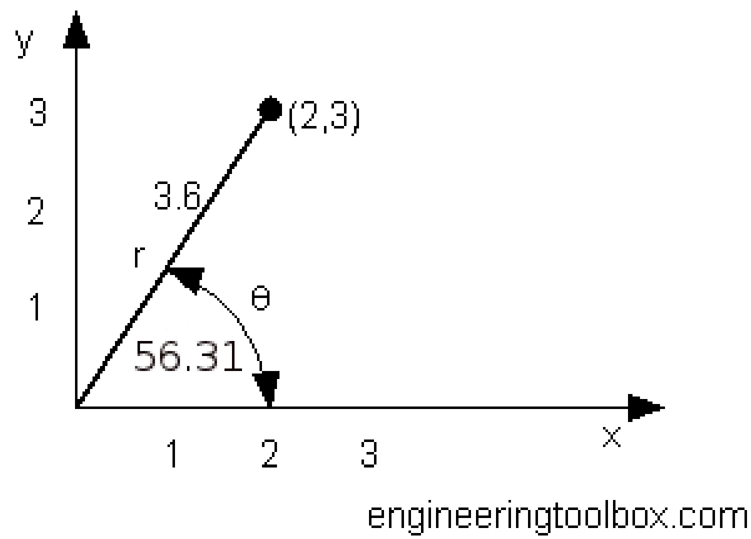
Polar Coordinates



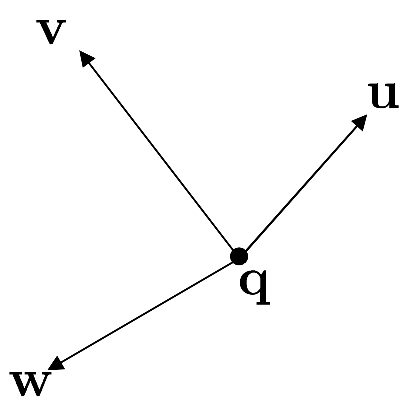
Images: Wikipedia

Cartesian and Polar Coordinates

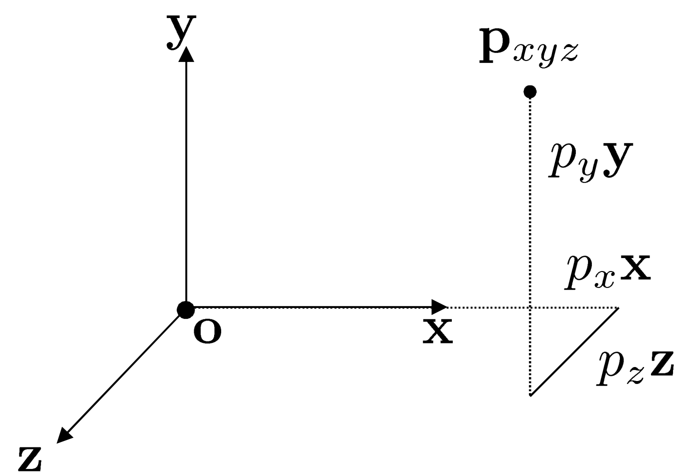
- The point's position can be expressed in cartesian coordinates (2,3) or polar coordinates (3.6, 56.31deg.)
- Both describe the same point!



Coordinate Transformation



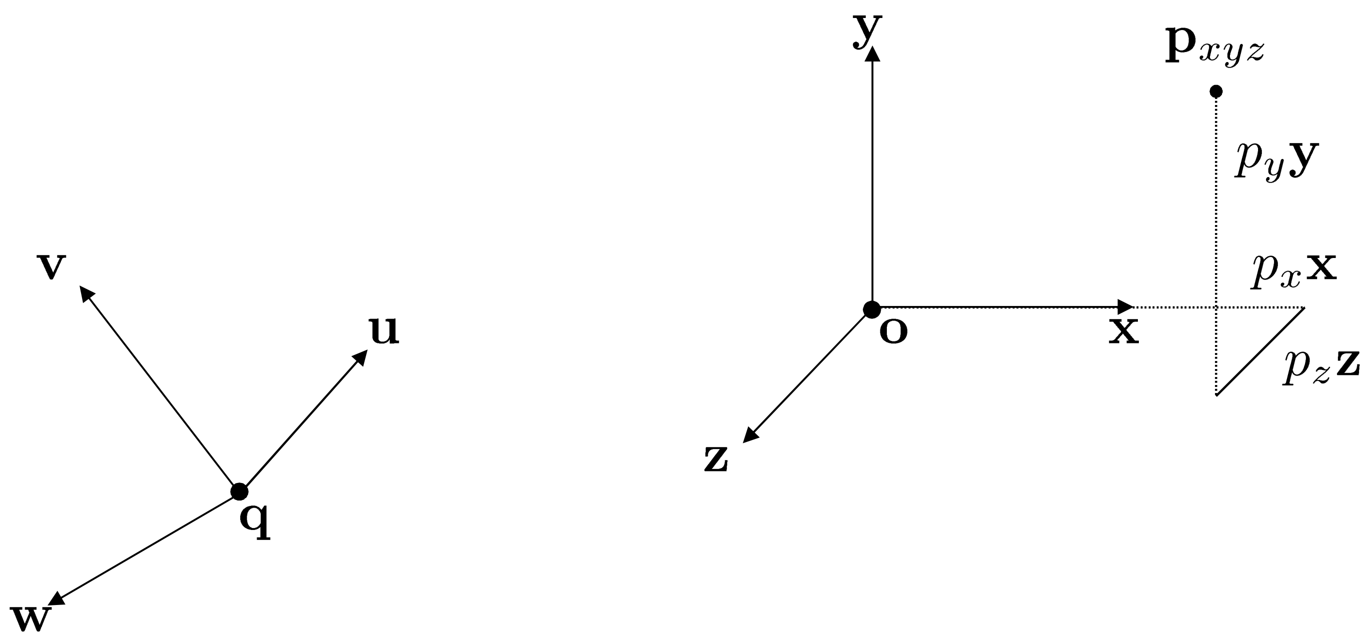
New **uvwq** coordinate system



Original **xyzo** coordinate system

Goal: Find coordinates of p_{xyz} in new **uvwq** coordinate system

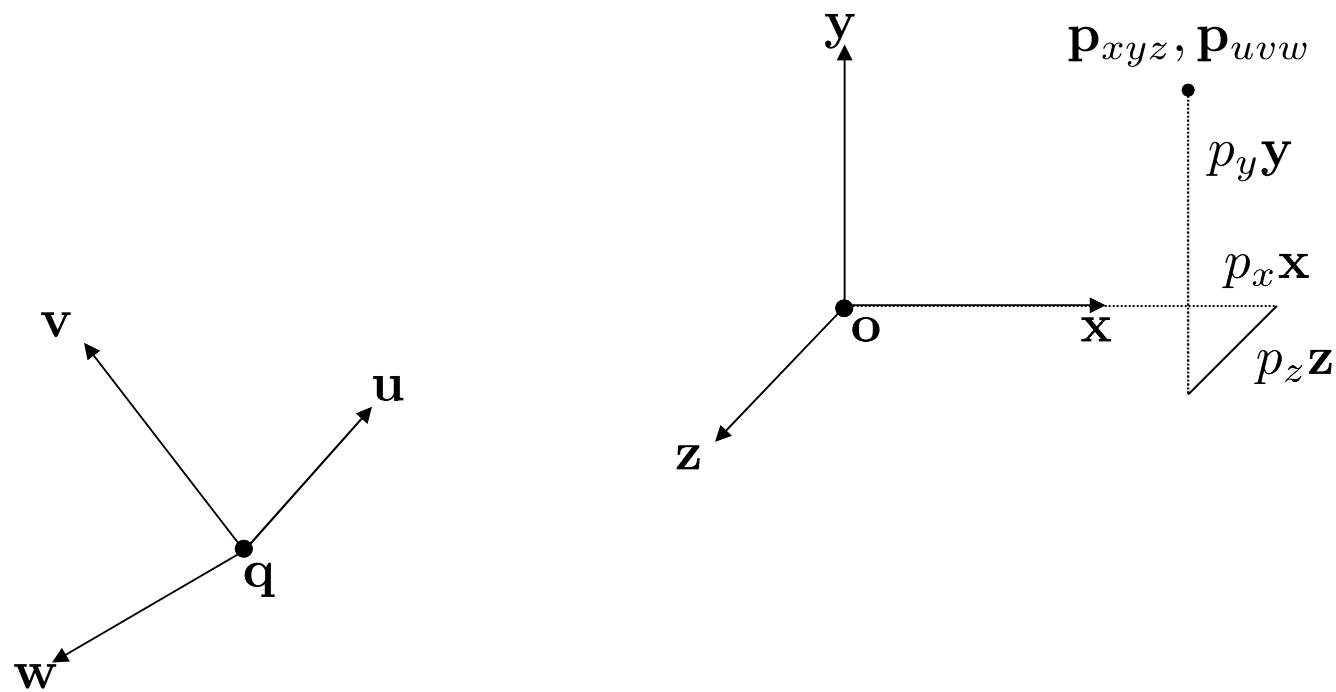
Coordinate Transformation



Express coordinates of **xyzo** reference frame with respect to **uvwq** reference frame:

$$\mathbf{x} = \begin{bmatrix} x_u \\ x_v \\ x_w \\ 0 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_u \\ y_v \\ y_w \\ 0 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} z_u \\ z_v \\ z_w \\ 0 \end{bmatrix} \quad \mathbf{o} = \begin{bmatrix} o_u \\ o_v \\ o_w \\ 1 \end{bmatrix}$$

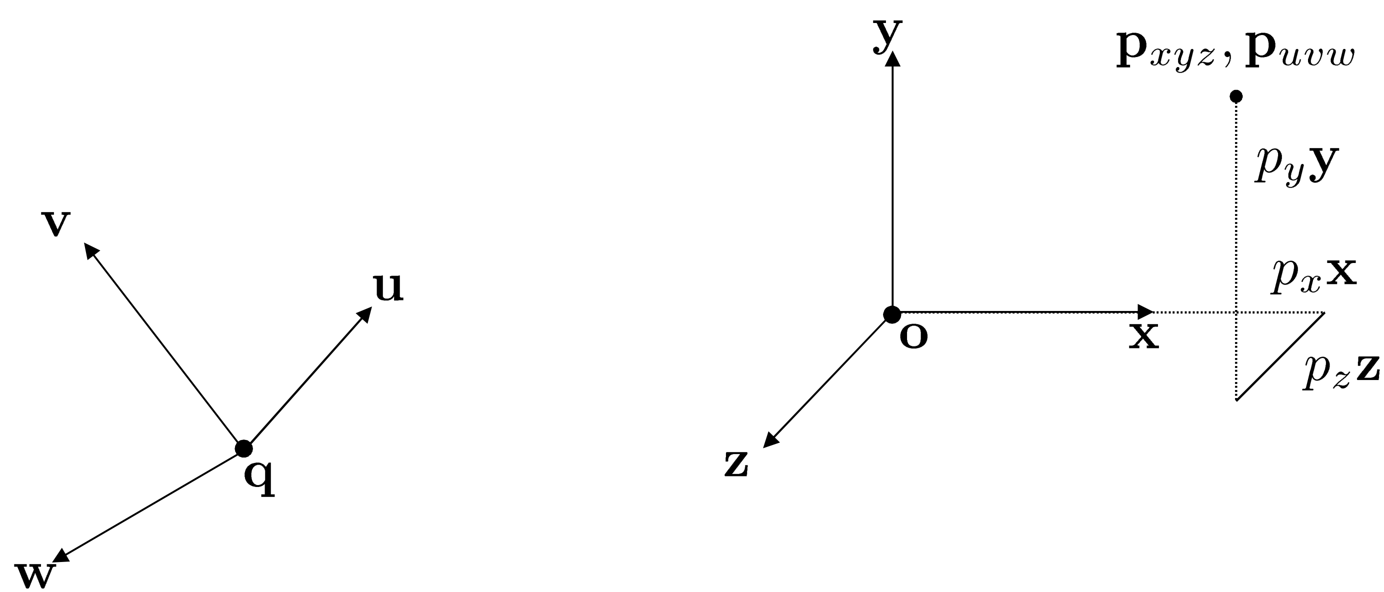
Coordinate Transformation



Point $\mathbf{p}$ expressed in new $\mathbf{uvwq}$ reference frame:

$$\mathbf{p}_{uvw} = p_x \begin{bmatrix} x_u \\ x_v \\ x_w \\ 0 \end{bmatrix} + p_y \begin{bmatrix} y_u \\ y_v \\ y_w \\ 0 \end{bmatrix} + p_z \begin{bmatrix} z_u \\ z_v \\ z_w \\ 0 \end{bmatrix} + \begin{bmatrix} o_u \\ o_v \\ o_w \\ 1 \end{bmatrix}$$

Coordinate Transformation



$$\mathbf{p}_{uvw} = \begin{bmatrix} x_u & y_u & z_u & o_u \\ x_v & y_v & z_v & o_v \\ x_w & y_w & z_w & o_w \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} & \mathbf{o} \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

Coordinate Transformation

Inverse transformation

- Given point $\mathbf{p}_{uvw}$ w.r.t. reference frame **uvwq**
- Coordinates $\mathbf{p}_{xyz}$ w.r.t. reference frame **xyzo**

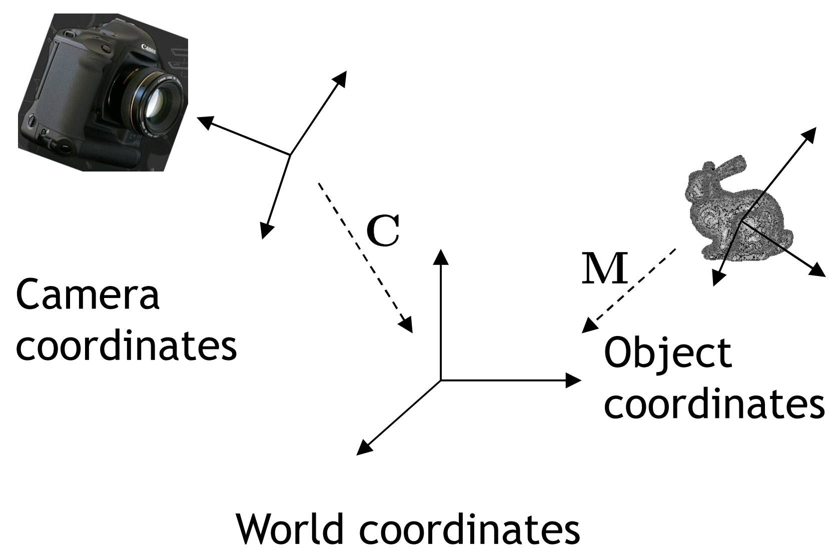
$$\mathbf{p}_{xyz} = \begin{bmatrix} x_u & y_u & z_u & o_u \\ x_v & y_v & z_v & o_v \\ x_w & y_w & z_w & o_w \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} p_u \\ p_v \\ p_w \\ 1 \end{bmatrix}$$

Lecture Overview

- Concatenating Transformations
- Coordinate Transformation
- Typical Coordinate Systems
- Projection

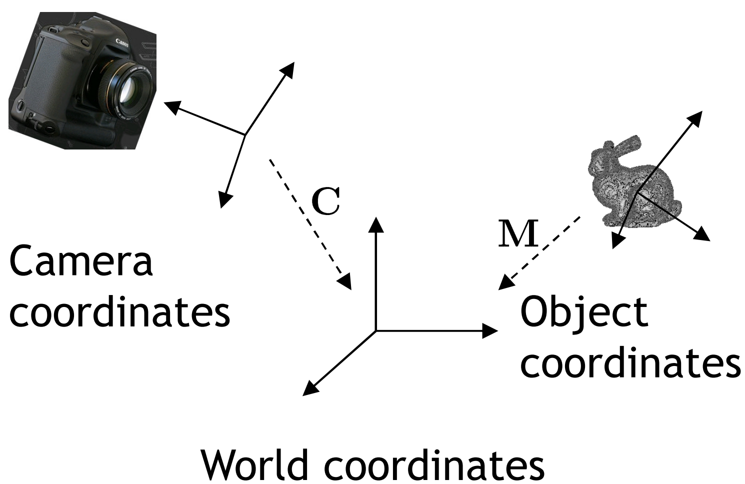
Typical Coordinate Systems

- In computer graphics, we typically use at least three coordinate systems:
  - World coordinate system
  - Camera coordinate system
  - Object coordinate system



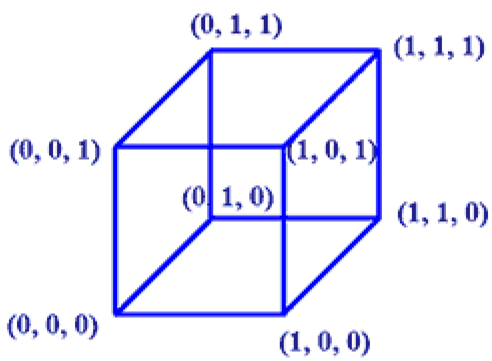
World Coordinates

- Common reference frame for all objects in the scene
- No standard for coordinate system orientation
  - If there is a ground plane, usually x/y is horizontal and z points up (height)
  - In OpenGL x/y is screen plane, z comes out

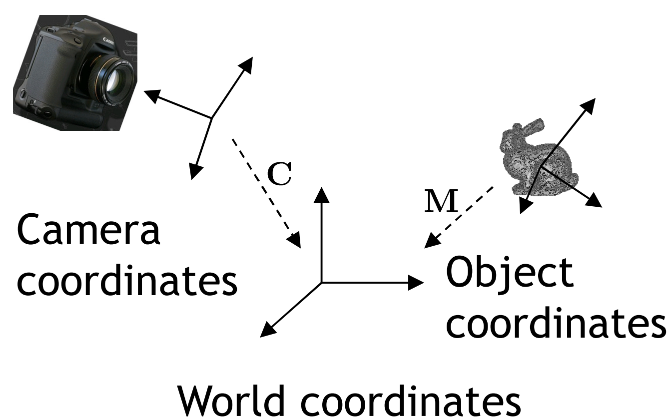


Object Coordinates

- Local coordinates in which points and other object geometry are given
- Often origin is in middle, base, or corner of object
  - Decided by the creator of the object

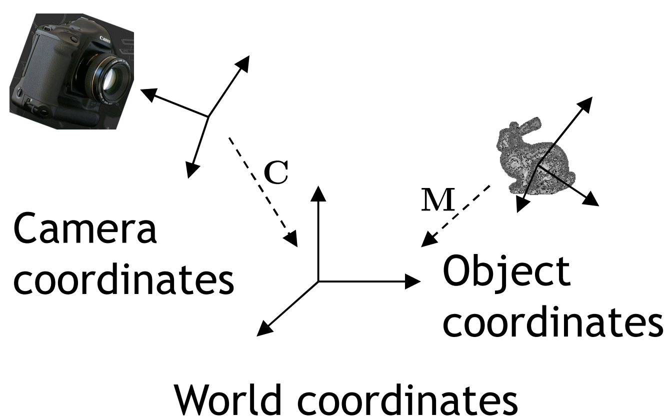


Source: http://motivate.maths.org



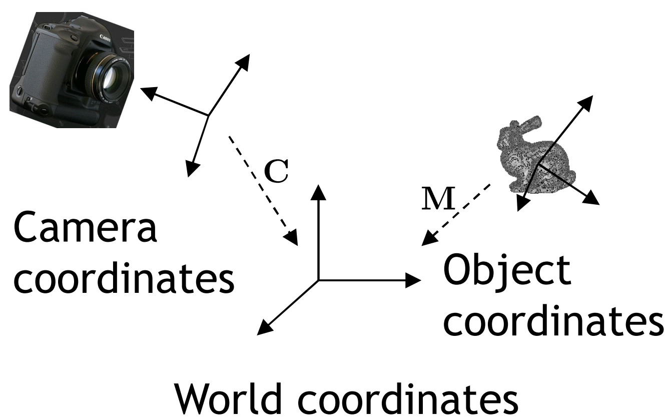
Object Transformation

- The transformation from object to world coordinates is different for each object
- Defines placement of object in scene
- Given by "model matrix" (model-to-world transformation) **M**



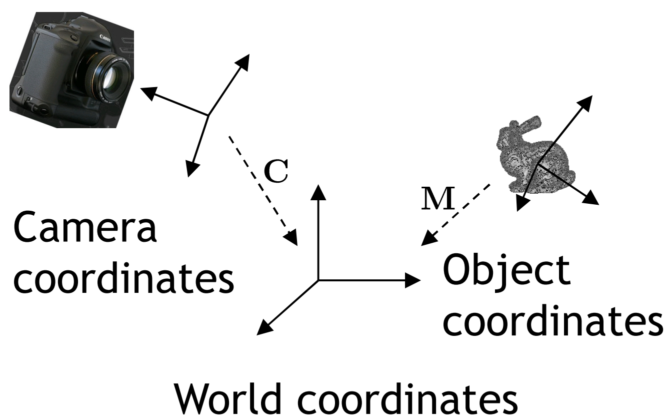
Camera Coordinate System

- Origin defines center of projection of camera
- x-y plane is parallel to image plane
- z-axis is perpendicular to image plane



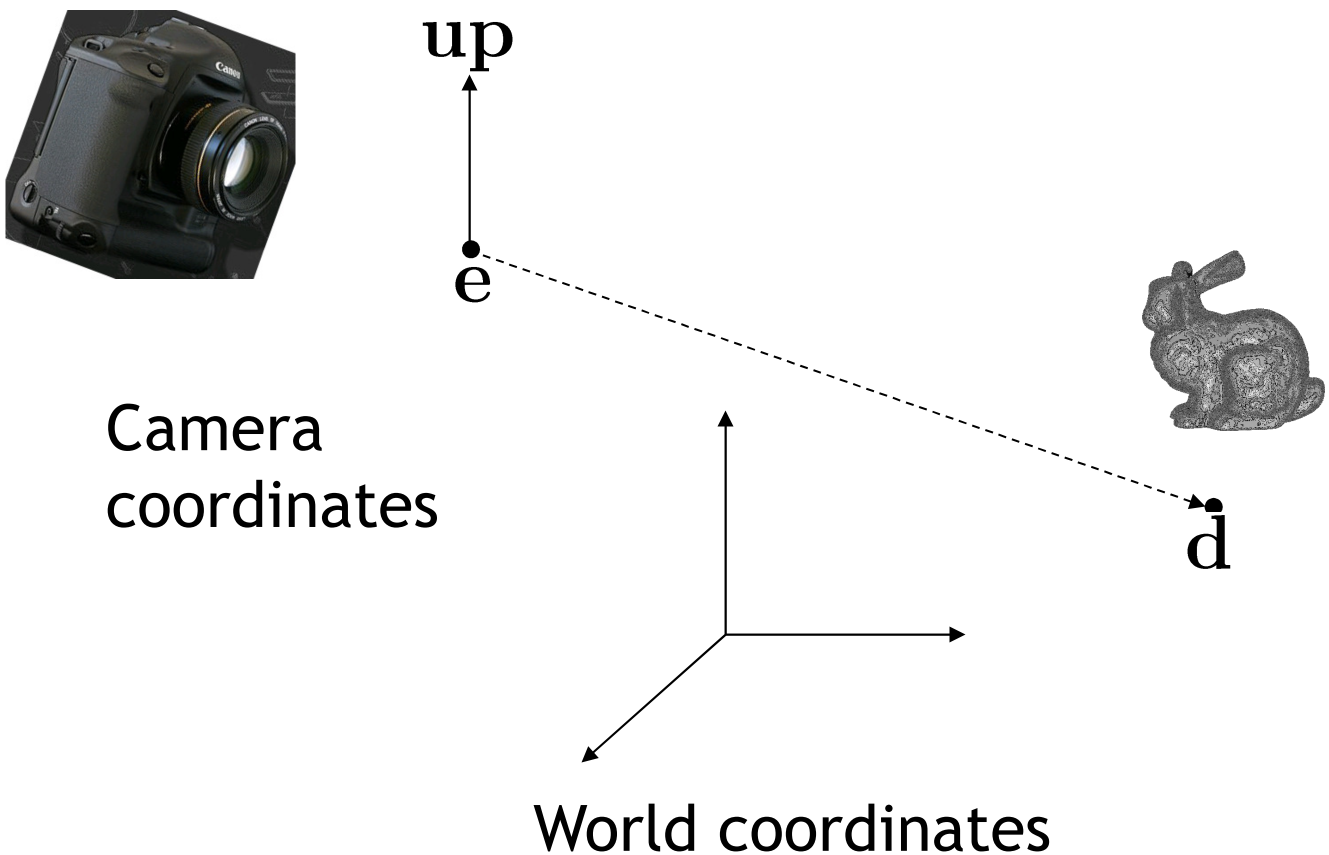
Camera Coordinate System

- The Camera Matrix defines the transformation from camera to world coordinates
  - Placement of camera in world



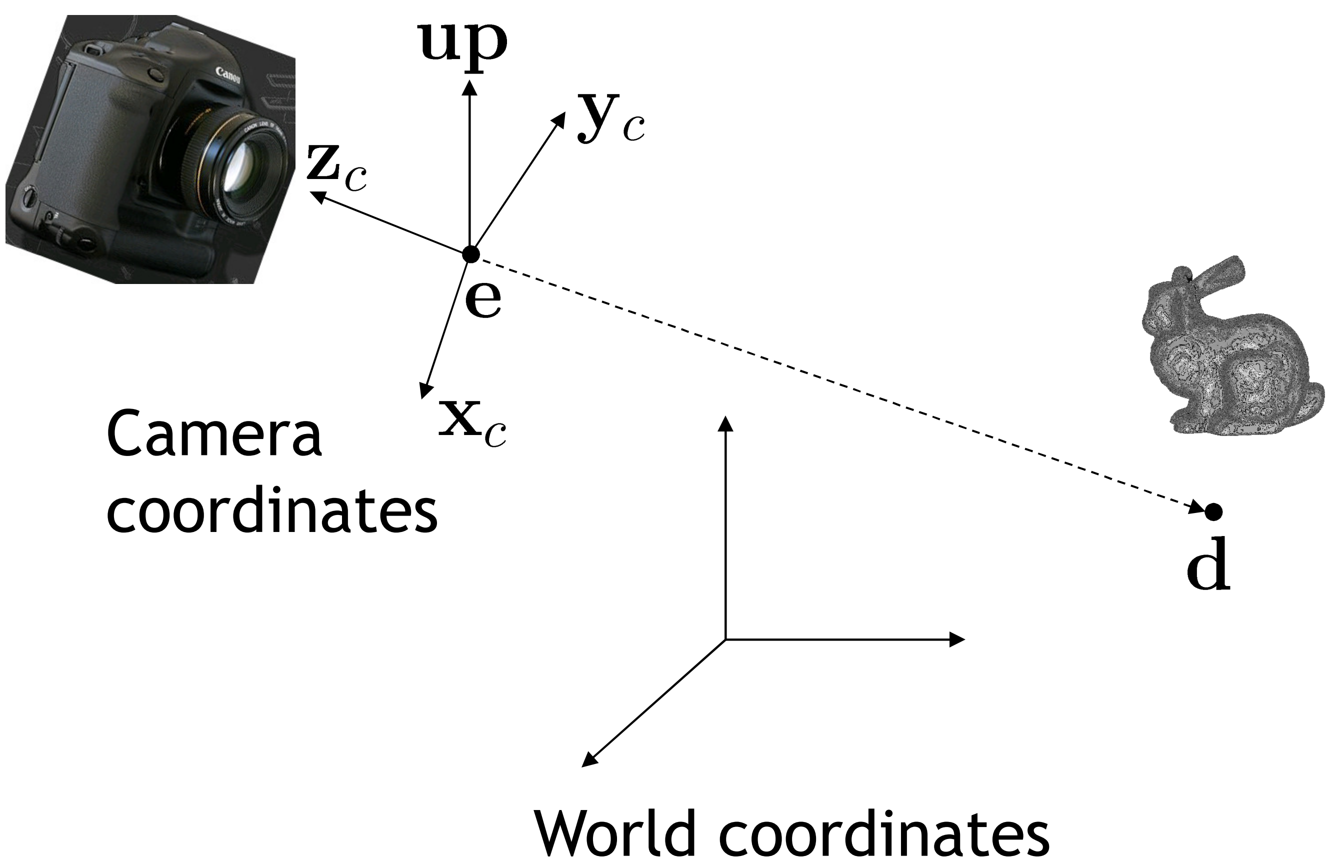
Camera Matrix

- Construct from center of projection **e**, look at **d**, up-vector **up:**



Camera Matrix

- Construct from center of projection **e**, look at **d**, up-vector **up:**



Camera Matrix

- z-axis

$$\mathbf{z}_c = \frac{\mathbf{e} - \mathbf{d}}{\|\mathbf{e} - \mathbf{d}\|}$$

- x-axis

$$\mathbf{x}_c = \frac{\mathbf{up} \times \mathbf{z}_c}{\|\mathbf{up} \times \mathbf{z}_c\|}$$

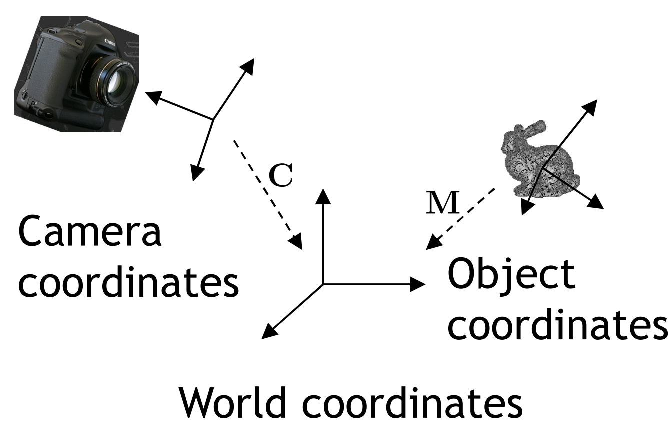
- y-axis

$$\mathbf{y}_c = \mathbf{z_c} \times \mathbf{x}_c$$

$$\mathbf{C} = \begin{bmatrix} \mathbf{x_c} & \mathbf{y_c} & \mathbf{z_c} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transforming Object to Camera Coordinates

- Object to world coordinates: **M**
- Camera to world coordinates: **C**
- Point to transform: **p**
- Resulting transformation equation: $\mathbf{p'} = \mathbf{C}^{-1}\,\mathbf{M}\,\mathbf{p}$



Tips for Notation

- Indicate coordinate systems with every point or matrix
  - Point: $\mathbf{p}_{object}$
  - Matrix: $\mathbf{M}_{object \rightarrow world}$
- Resulting transformation equation:

$$\mathbf{p}_{camera} = (\mathbf{C}_{camera \rightarrow world})^{-1} \, \mathbf{M}_{object \rightarrow world} \, \mathbf{p}_{object}$$

- In source code:
  - Point: `p_object` or `p_obj`
  - Matrix: `object2world` or `obj2wld`
- Resulting transformation equation:

```
wld2cam = inverse(cam2wld);
p_cam = p_obj * obj2wld * wld2cam;
```

Inverse of Camera Matrix

- How to calculate the inverse of the camera matrix $\mathbf{C}^{-1}$?
- Generic matrix inversion is complex and compute-intensive
- Affine transformation matrices can be inverted more easily
- Observation:
  - Camera matrix consists of rotation and translation: $\mathbf{R} \times \mathbf{T}$
- Inverse of rotation: $\mathbf{R}^{-1} = \mathbf{R}^T$
- Inverse of translation: $\mathbf{T}(t)^{-1} = \mathbf{T}(-t)$
- Inverse of camera matrix: $\mathbf{C}^{-1} = \mathbf{T}^{-1} \times \mathbf{R}^{-1}$

Objects in Camera Coordinates

- We have things lined up the way we like them on screen
  - x to the right
  - y up
  - $-z$ into the screen
  - Objects to look at are in front of us, i.e. have negative z values
- But objects are still in 3D
- Next step: project scene to 2D plane

Lecture Overview

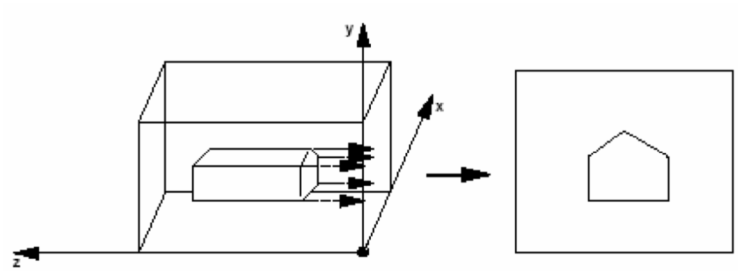
- Concatenating Transformations
- Coordinate Transformation
- Typical Coordinate Systems
- Projection

Projection

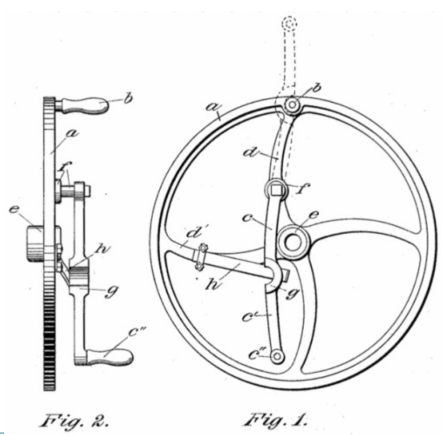
- Goal:
  Given 3D points (vertices) in camera coordinates, determine corresponding image coordinates
- Transforming 3D points into 2D is called Projection
- OpenGL supports two types of projection:
  - Orthographic Projection (=Parallel Projection)
  - Perspective Projection

Orthographic Projection

- Can be done by ignoring *z*-coordinate
  - Use camera space *xy* coordinates as image coordinates
- Project points to *x*-*y* plane along parallel lines



- Often used in graphical illustrations, architecture, 3D modeling

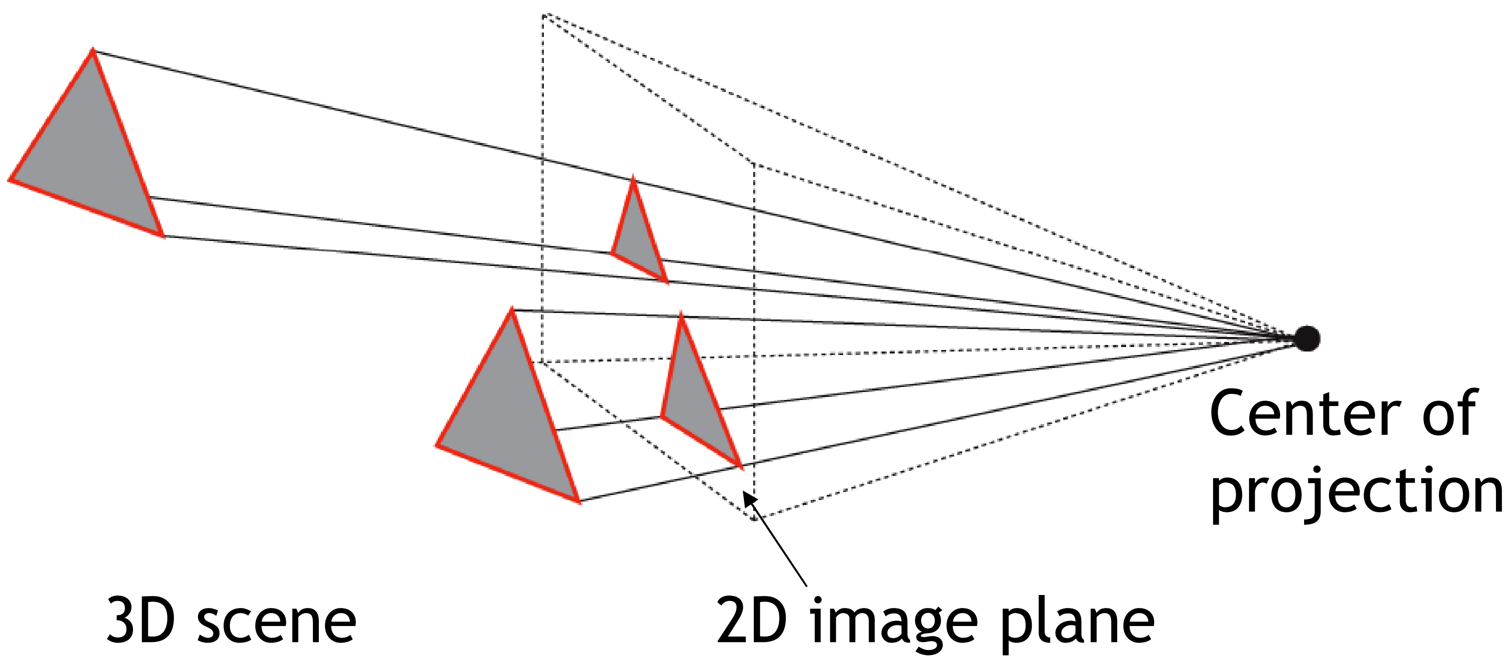


Perspective Projection

- Most common for computer graphics
- Simplified model of human eye, or camera lens (*pinhole camera*)
- Things farther away appear to be smaller
- Discovery attributed to Filippo Brunelleschi (Italian architect) in the early 1400's

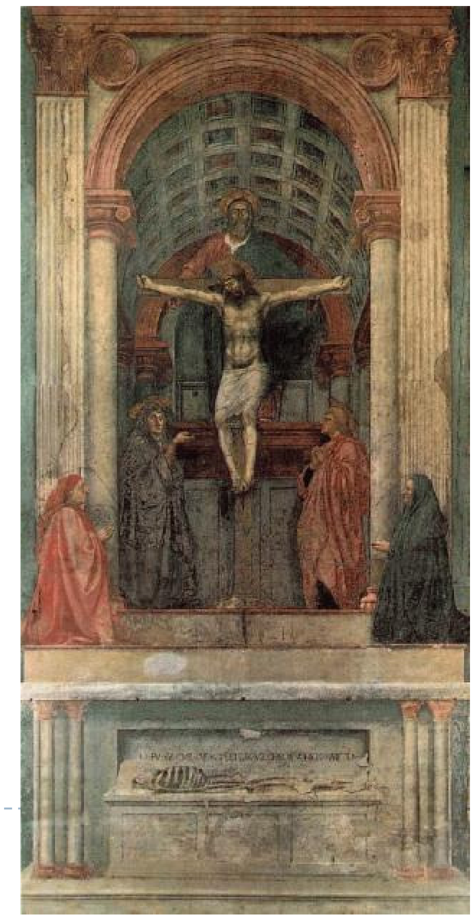
Perspective Projection

- Project along rays that converge in center of projection



Perspective Projection

Parallel lines are
no longer parallel,
converge in one point



Earliest example:
La Trinitá (1427) by Masaccio

Video

- Professor Ravi Ramamoorthi on Perspective Projection
  - Part of the Online Lectures for a 6 week computer graphics course, modeled on UC Berkeley's CS 184
  - http://www.youtube.com/watch?v=VpNJbvZhNCQ

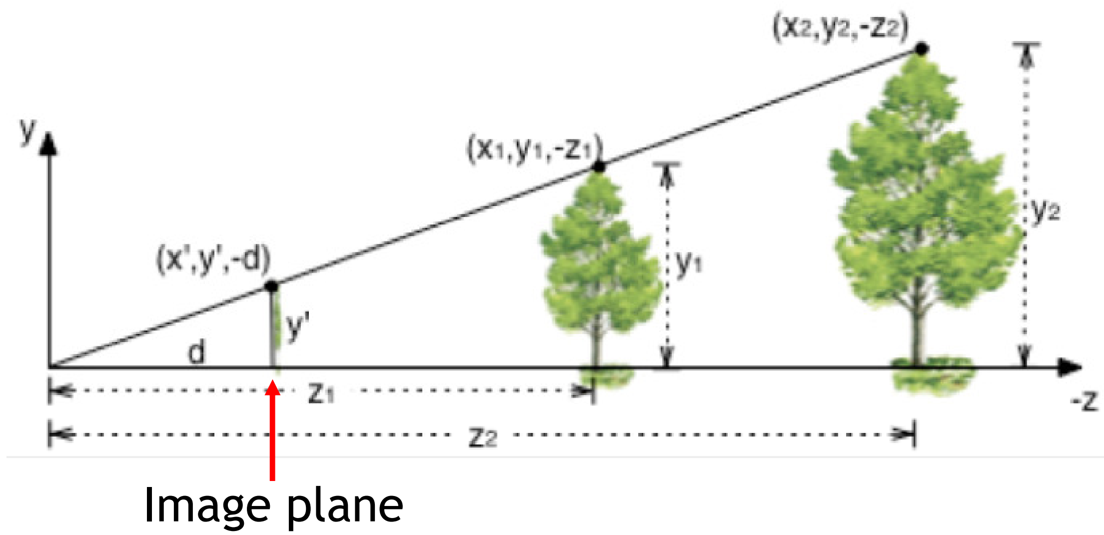
Perspective Projection

From law of ratios in similar triangles follows:

$$\frac{y'}{d} = \frac{y_1}{z_1} \rightarrow y' = \frac{y_1 d}{z_1}$$

Similarly: $$x' = \frac{x_1 d}{z_1}$$

By definition: $$z' = d$$



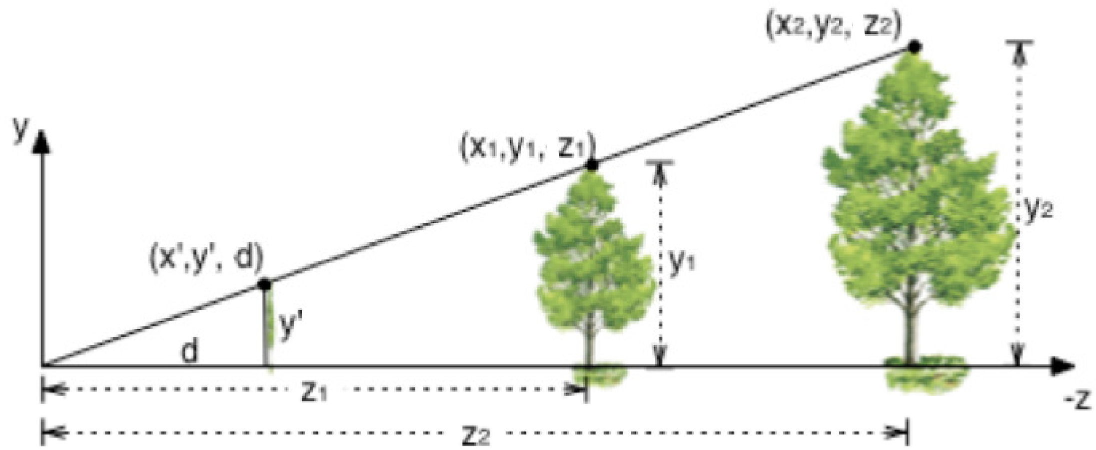
- We can express this using homogeneous coordinates and 4x4 matrices as follows

Perspective Projection

$$x' = \frac{x_1 d}{z_1}$$

$$y' = \frac{y_1 d}{z_1}$$

$$z' = d$$



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \Rightarrow \begin{bmatrix} xd/z \\ yd/z \\ d \\ 1 \end{bmatrix}$$

Projection matrix Homogeneous division

Perspective Projection

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} = \begin{bmatrix} xd/z \\ yd/z \\ d \\ 1 \end{bmatrix}$$

Projection matrix P

- Using projection matrix, homogeneous division seems more complicated than just multiplying all coordinates by d/z, so why do it?
- It will allow us to:
  - Handle different types of projections in a unified way
  - Define arbitrary view volumes