CSE 167:
Introduction to Computer Graphics
Lecture #4: Coordinate Systems

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2020

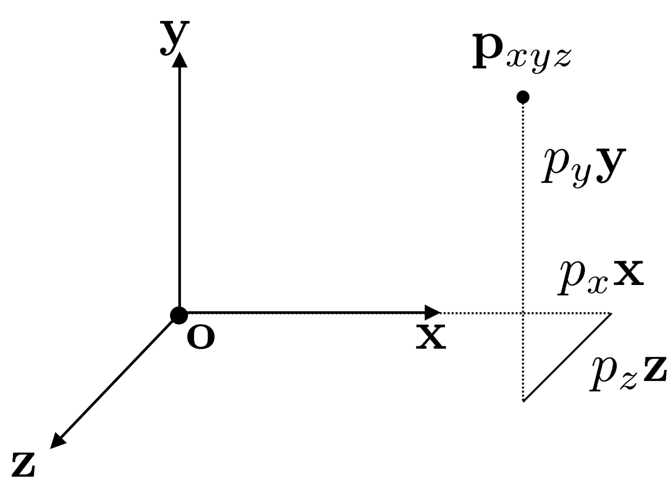
Announcements

- Homework Project 1 due October 25
- Discussion Project 1: Wednesday 1pm

Coordinate System

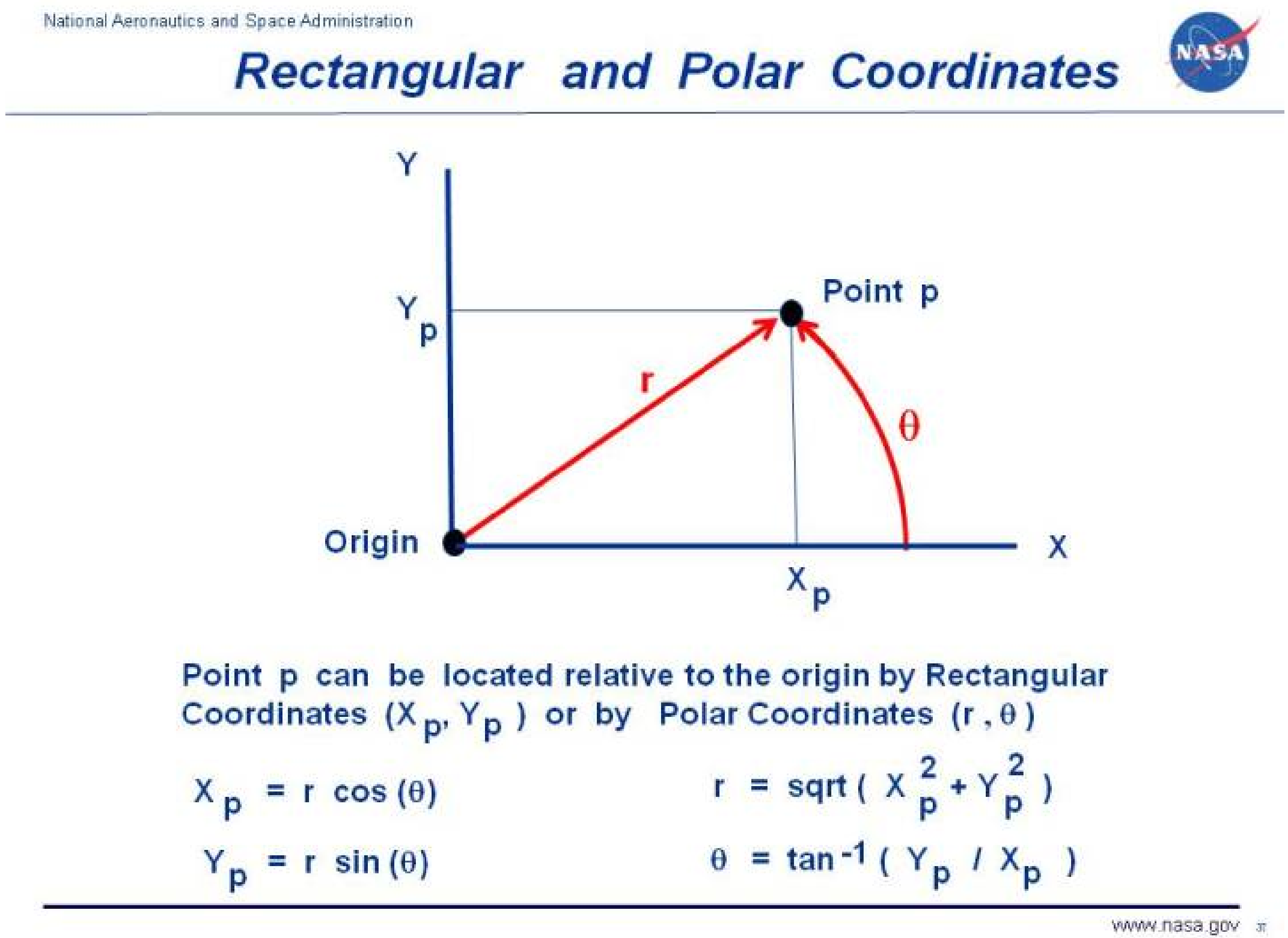
- Given point **p** in homogeneous coordinates: $\begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$

- Coordinates describe the point's 3D position in a coordinate system with basis vectors **x**, **y**, **z** and origin **o**:



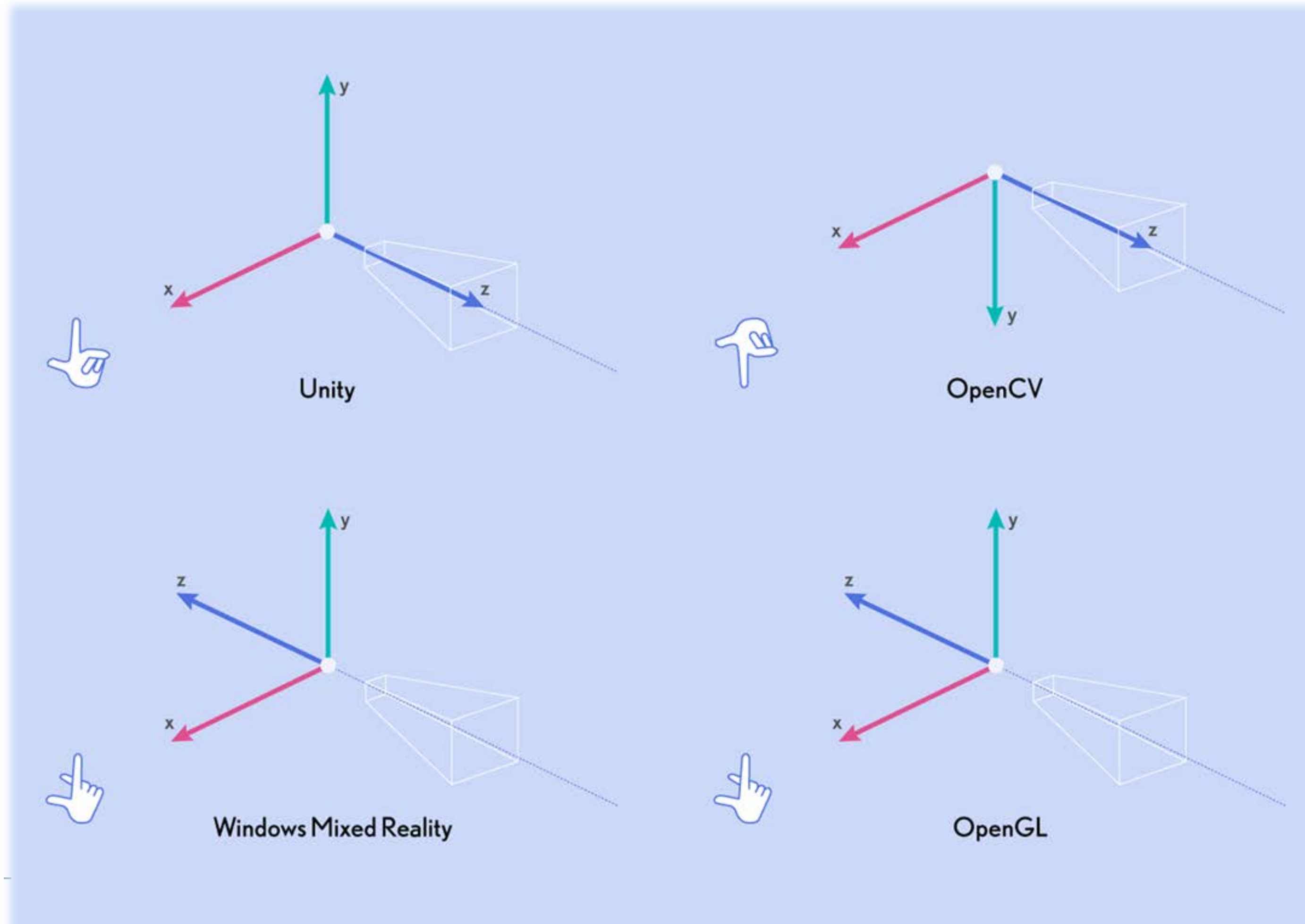
$$\mathbf{p}_{xyz} = p_x\mathbf{x} + p_y\mathbf{y} + p_z\mathbf{z} + \mathbf{o}$$

Rectangular and Polar Coordinates

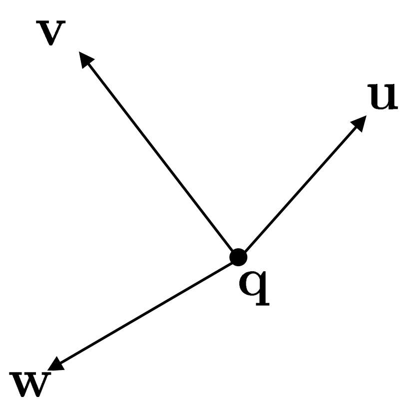


Coordinate System Orientation

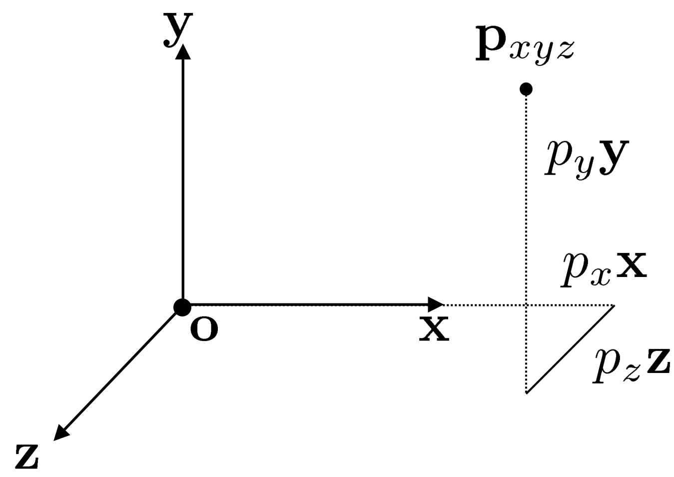
- Right-handed and left-handed coordinate systems



Coordinate Transformation



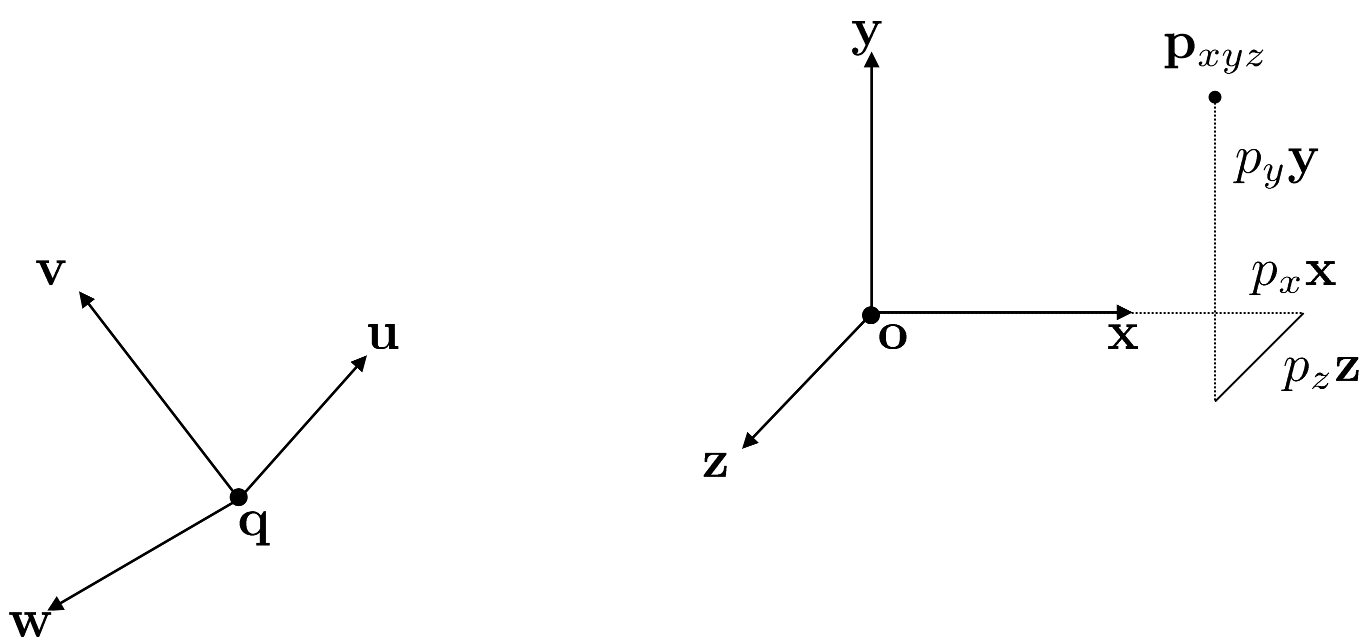
New **uvwq** coordinate system



Original **xyzo** coordinate system

Goal: Find coordinates of $\mathbf{p}_{xyz}$ in new **uvwq** coordinate system

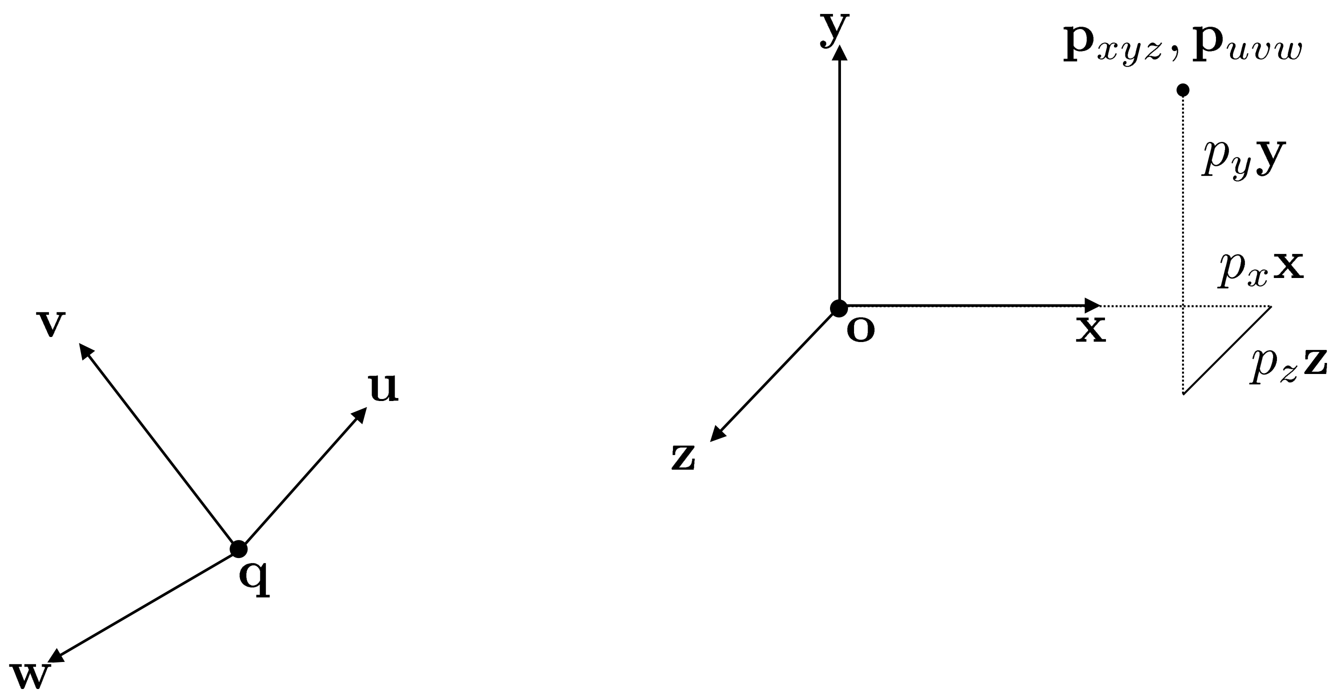
Coordinate Transformation



Express coordinates of $\mathbf{xyzo}$ reference frame with respect to $\mathbf{uvwq}$ reference frame:

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

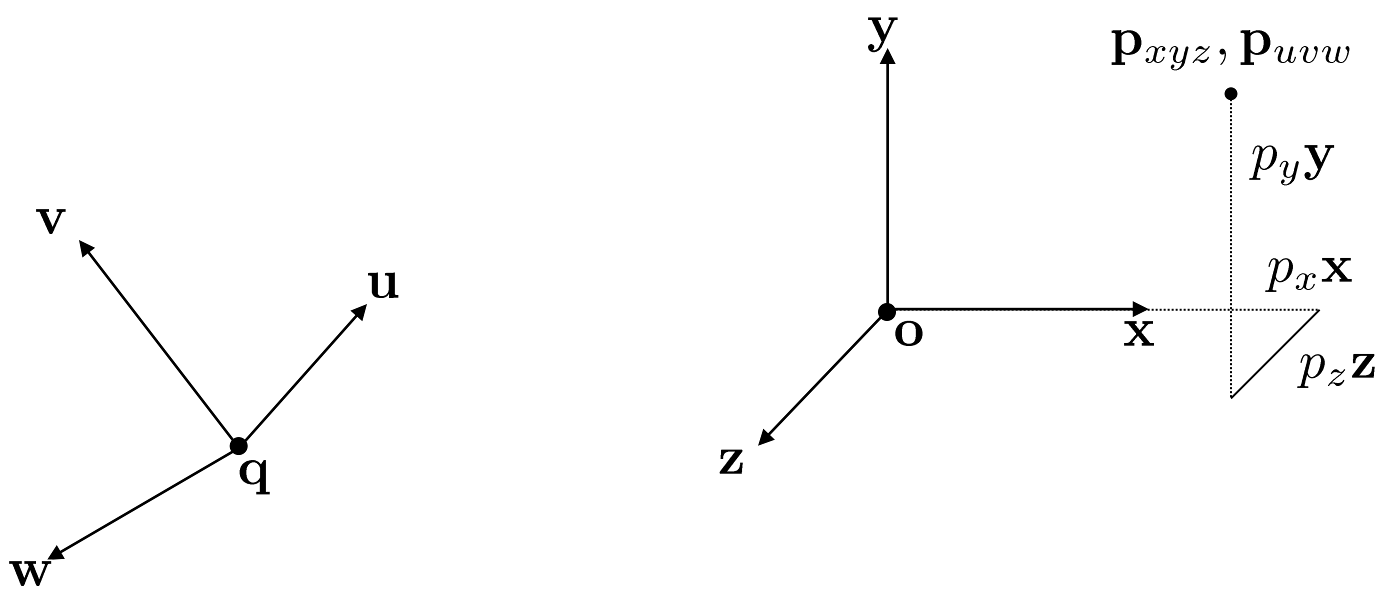
Coordinate Transformation



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** are calculated as:

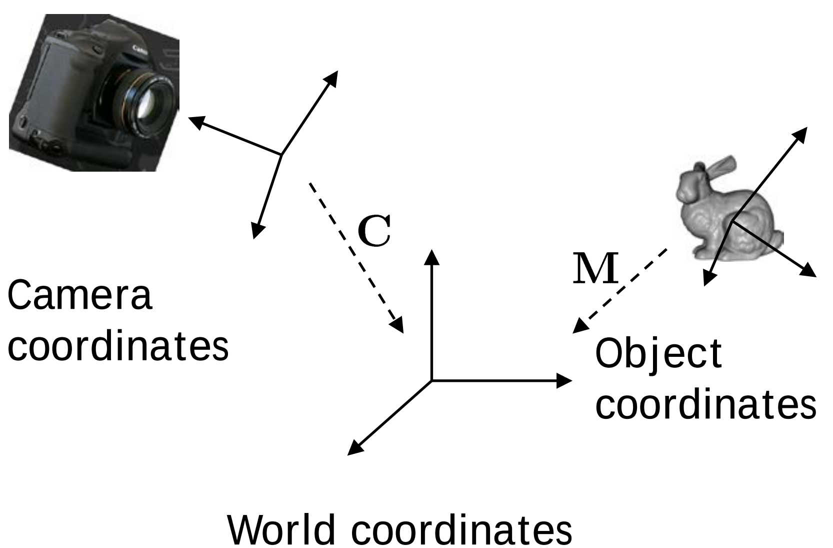
$$\mathbf{p}_{xyz} = \begin{bmatrix} x_u & y_u & z_u & o_u \\ x_v & y_v & z_v & o_v \\ x_w & y_w & z_w & o_w \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} p_u \\ p_v \\ p_w \\ 1 \end{bmatrix}$$

Lecture Overview

- Coordinate Transformation
- Typical Coordinate Systems

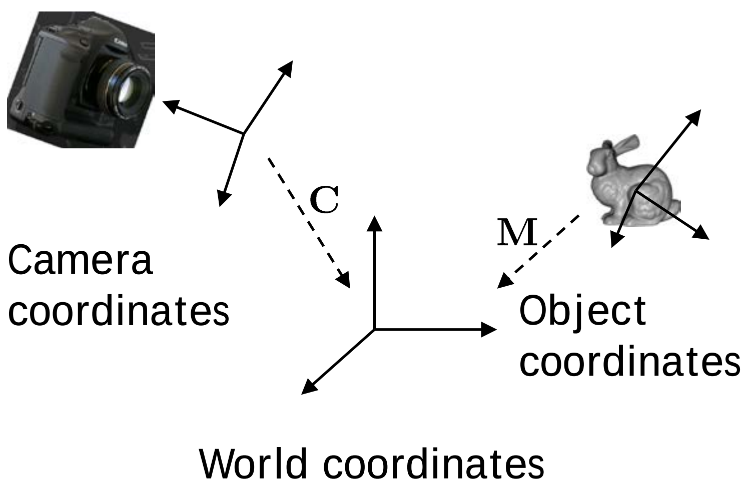
Typical Coordinate Systems

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



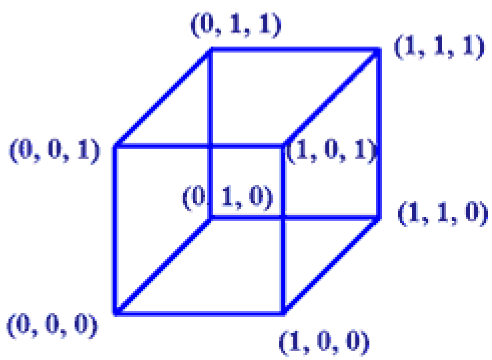
World Coordinates

- 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)
  - Otherwise, x/y is often screen plane, z points out of the screen

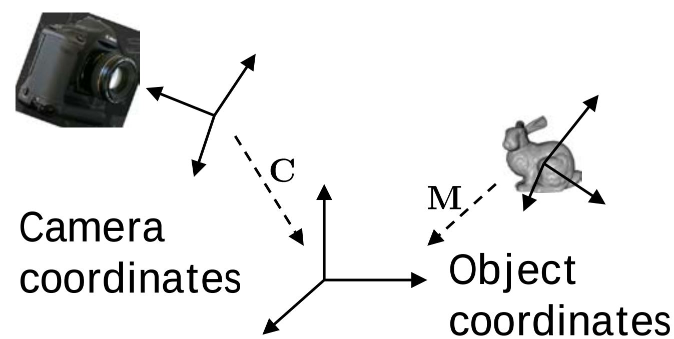


Object Coordinates

- Local coordinates in which points and other object geometry are given
- Often origin is in geometric center, on the base, or in a corner of the object
  - Depends on how object is generated or used.



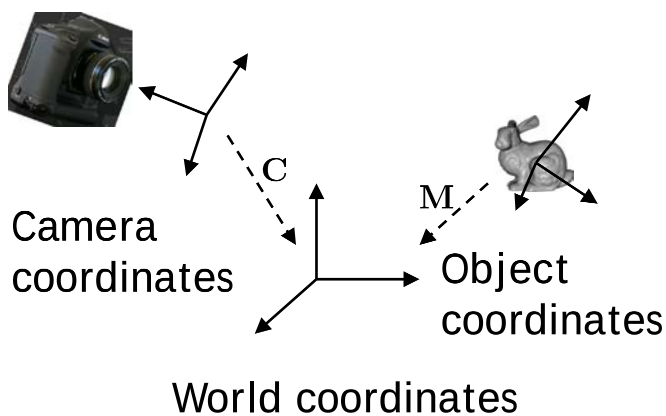
Source: http://motivate.maths.org



World coordinates

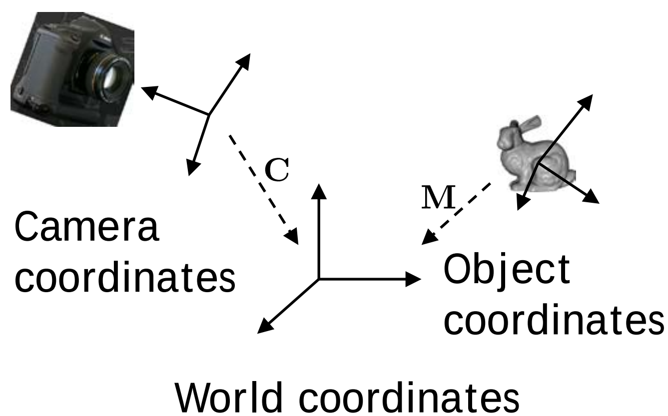
Object Transformation

- 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**.



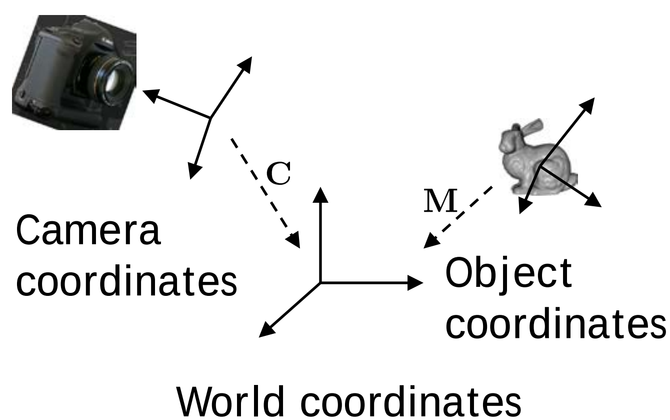
Camera Coordinate System

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



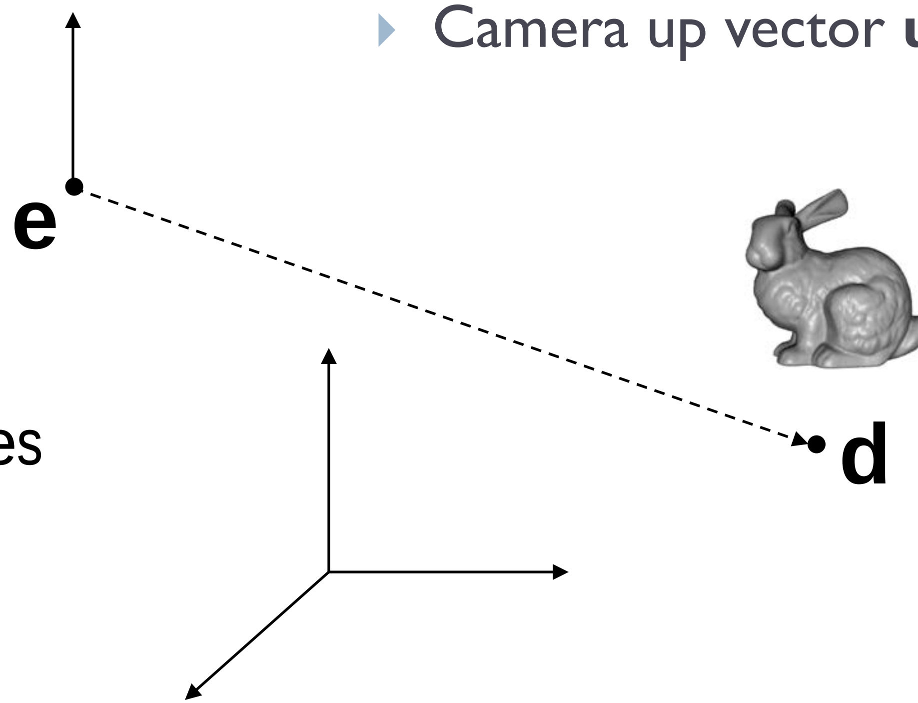
Camera Coordinate System

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



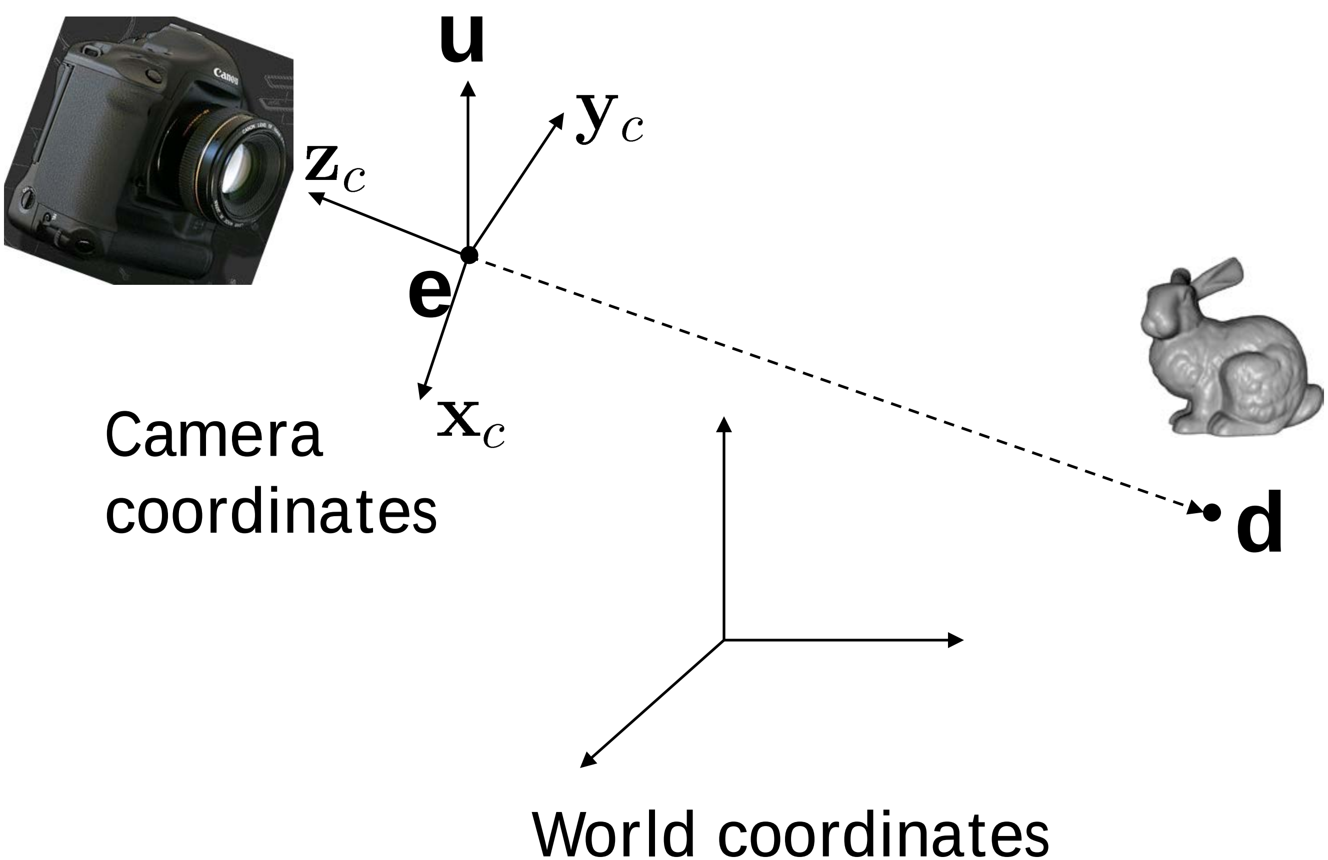
Camera Matrix

- Given:
  - Center point of projection **e**
  - Look at point **d**
  - Camera up vector **u**



Camera Matrix

- Construct $\mathbf{x_c}$, $\mathbf{y_c}$, $\mathbf{z_c}$



Camera Matrix

- Step 1: z-axis
$$\boldsymbol{z}_C = \frac{\boldsymbol{e} - \boldsymbol{d}}{\|\boldsymbol{e} - \boldsymbol{d}\|}$$

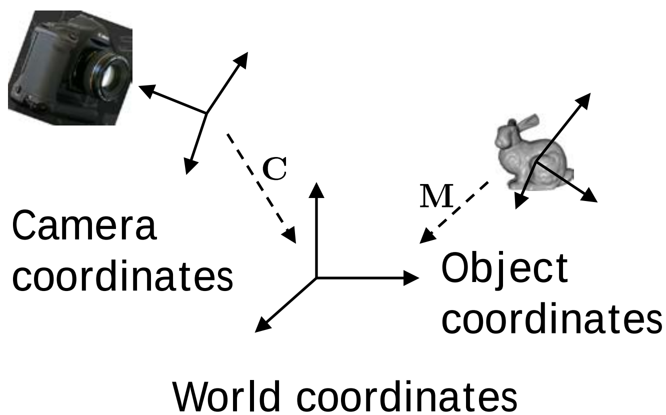
- Step 2: x-axis
$$\boldsymbol{x}_C = \frac{\boldsymbol{u} \times \boldsymbol{z}_C}{\|\boldsymbol{u} \times \boldsymbol{z}_C\|}$$

- Step 3: y-axis
$$\boldsymbol{y}_C = \boldsymbol{z}_C \times \boldsymbol{x}_C = \frac{\boldsymbol{u}}{\|\boldsymbol{u}\|}$$

- Camera Matrix:
$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{x}_C & \boldsymbol{y}_C & \boldsymbol{z}_C & \boldsymbol{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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: $\mathbf{M}_{object \to world}$
- Resulting transformation equation:

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

- In C++ code use similar names:
  - Point: `p_object` or `p_obj` or `p_o`
  - Matrix: `object2world` or `obj2wld` or `o2w`
- Resulting transformation equation:

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

Inverse of Camera Matrix

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

Objects in Camera Coordinates

- We have things lined up the way we like them on screen
  - **x** points to the right
  - **y** points up
  - **-z** into the screen (i.e., **z** points out of 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