

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

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2018

Announcements

- ▶ **Next Friday: homework 2 due at 2pm**
 - ▶ Upload to TritonEd
 - ▶ Demonstrate in CSE basement labs

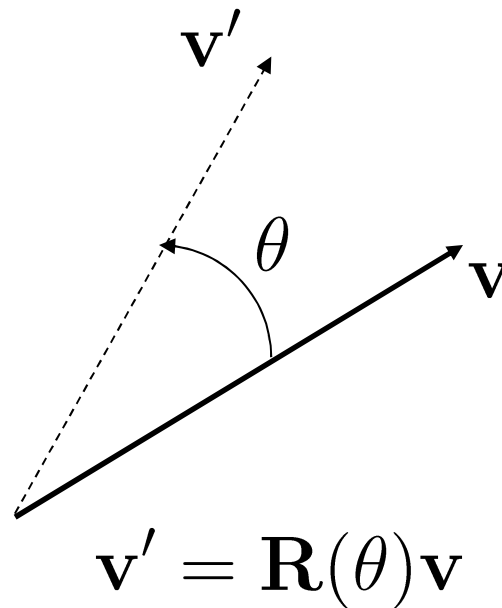
Today

- ▶ Finish up linear algebra foundations
- ▶ Coordinate system transformations

Rotation in 2D

- ▶ Convention: positive angle rotates counterclockwise
- ▶ Rotation matrix

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$



Rotation in 3D

Rotation around coordinate axes

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix}$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation in 3D

- ▶ Concatenation of rotations around x, y, z axes

$$\mathbf{R}_{x,y,z}(\theta_x, \theta_y, \theta_z) = \mathbf{R}_x(\theta_x)\mathbf{R}_y(\theta_y)\mathbf{R}_z(\theta_z)$$

- ▶ $\theta_x, \theta_y, \theta_z$ are called Euler angles
- ▶ Result depends on matrix order!

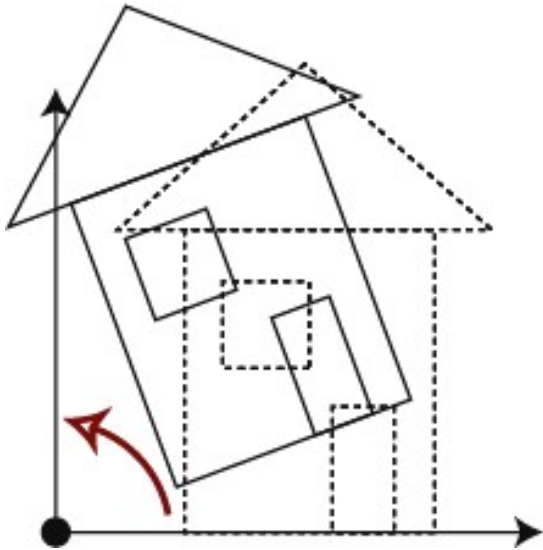
$$\mathbf{R}_x(\theta_x)\mathbf{R}_y(\theta_y)\mathbf{R}_z(\theta_z) \neq \mathbf{R}_z(\theta_z)\mathbf{R}_y(\theta_y)\mathbf{R}_x(\theta_x)$$

Rotation about an Arbitrary Axis

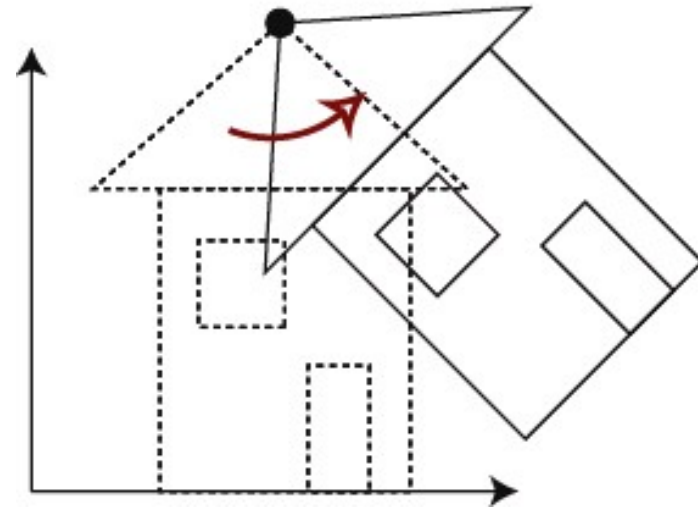
- ▶ Complicated!
- ▶ Rotate point $[x,y,z]$ about axis $[u,v,w]$ by angle θ :

$$\begin{bmatrix} \frac{u(ux+vy+wz)(1-\cos\theta) + (u^2+v^2+w^2)x\cos\theta + \sqrt{u^2+v^2+w^2}(-wy+yz)\sin\theta}{u^2+v^2+w^2} \\ \frac{v(ux+vy+wz)(1-\cos\theta) + (u^2+v^2+w^2)y\cos\theta + \sqrt{u^2+v^2+w^2}(wx-uz)\sin\theta}{u^2+v^2+w^2} \\ \frac{w(ux+vy+wz)(1-\cos\theta) + (u^2+v^2+w^2)z\cos\theta + \sqrt{u^2+v^2+w^2}(-vx+uy)\sin\theta}{u^2+v^2+w^2} \end{bmatrix}$$

How to rotate around a Pivot Point?

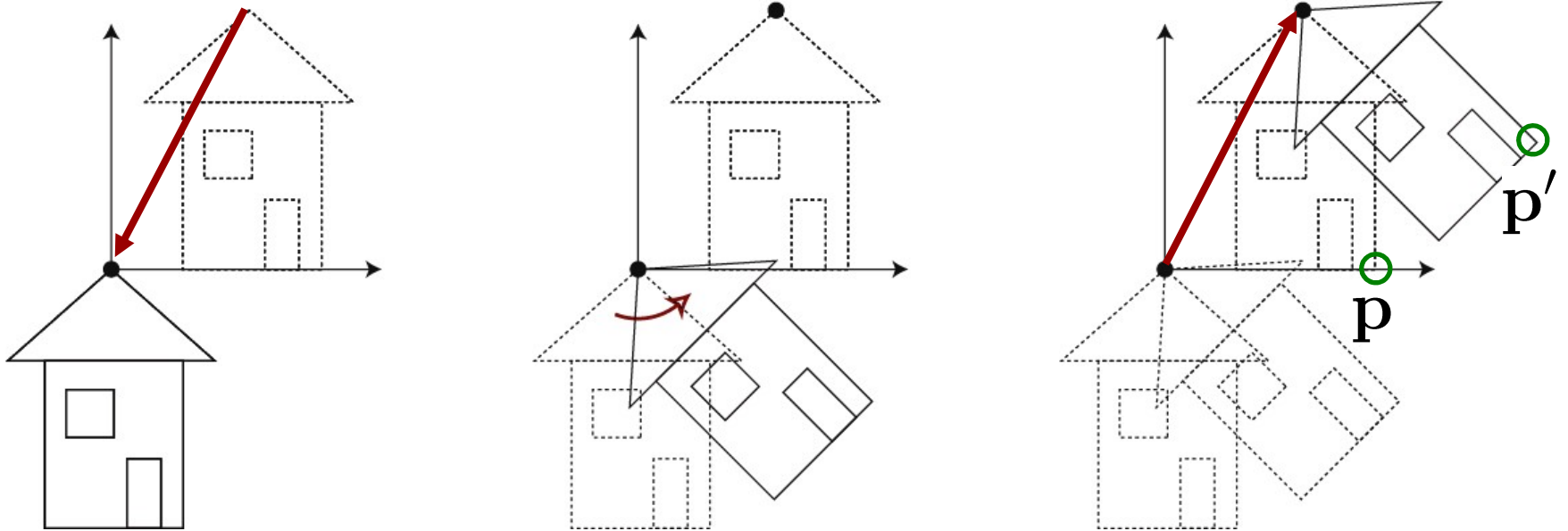


Rotation around
origin:
 $\mathbf{p}' = \mathbf{R} \mathbf{p}$



Rotation around
pivot point:
 $\mathbf{p}' = ?$

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 $\mathbf{M}_3\mathbf{M}_2\mathbf{M}_1$

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

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

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

- ▶ Note: associativity applies

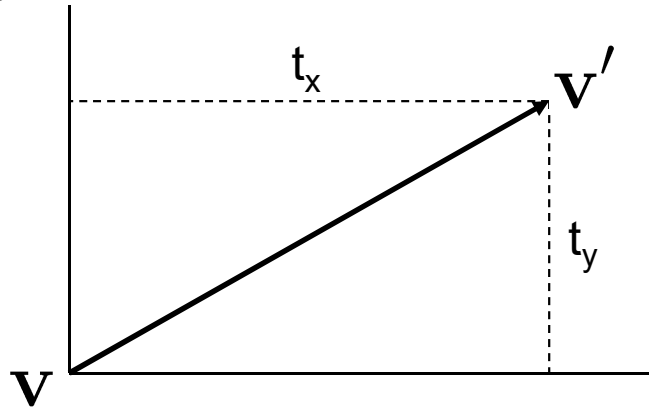
$$\mathbf{M}_{total} = (\mathbf{M}_3\mathbf{M}_2)\mathbf{M}_1 = \mathbf{M}_3(\mathbf{M}_2\mathbf{M}_1)$$

Today

- ▶ Vectors and matrices
- ▶ Affine transformations
- ▶ Homogeneous coordinates

Translation

► Translation in 2D



► Translation matrix T =?

$$v' = \begin{bmatrix} v_x \\ v_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = T v = T \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

Translation

- ▶ Translation in 2D: 3x3 matrix

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- ▶ Analogous in 3D: 4x4 matrix

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Homogeneous Coordinates

- ▶ Basic: a trick to unify/simplify computations.
- ▶ Deeper: projective geometry
 - ▶ Interesting mathematical properties
 - ▶ Good to know, but less immediately practical
 - ▶ We will use some aspect of this when we do perspective projection

Homogeneous Coordinates

- ▶ Add an extra component. 1 for a point, 0 for a vector:

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \quad \mathbf{r}_v = \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix}$$

- ▶ Combine **M** and **d** into single 4x4 matrix:

$$\begin{bmatrix} m_{xx} & m_{xy} & m_{xz} & d_x \\ m_{yx} & m_{yy} & m_{yz} & d_y \\ m_{zx} & m_{zy} & m_{zz} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- ▶ And see what happens when we multiply...



Homogeneous Point Transform

- Transform a point:

$$\begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix} = \begin{bmatrix} m_{xx} & m_{xy} & m_{xz} & d_x \\ m_{yx} & m_{yy} & m_{yz} & d_y \\ m_{zx} & m_{zy} & m_{zz} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} m_{xx}p_x + m_{xy}p_y + m_{xz}p_z + d_x \\ m_{yx}p_x + m_{yy}p_y + m_{yz}p_z + d_y \\ m_{zx}p_x + m_{zy}p_y + m_{zz}p_z + d_z \\ 0 + 0 + 0 + 1 \end{bmatrix}$$


$$\mathbf{M} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} + \mathbf{d}$$

- Top three rows are the affine transform!
- Bottom row stays 1

Homogeneous Vector Transform

- Transform a vector:

$$\begin{bmatrix} v'_x \\ v'_y \\ v'_z \\ 0 \end{bmatrix} = \begin{bmatrix} m_{xx} & m_{xy} & m_{xz} & d_x \\ m_{yx} & m_{yy} & m_{yz} & d_y \\ m_{zx} & m_{zy} & m_{zz} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} = \begin{bmatrix} m_{xx}v_x + m_{xy}v_y + m_{xz}v_z + 0 \\ m_{yx}v_x + m_{yy}v_y + m_{yz}v_z + 0 \\ m_{zx}v_x + m_{zy}v_y + m_{zz}v_z + 0 \\ 0 + 0 + 0 + 0 \end{bmatrix}$$


 $\mathbf{M} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$

- Top three rows are the linear transform
 - Displacement **d** is properly ignored
- Bottom row stays 0

Homogeneous Arithmetic

- ▶ Legal operations always end in 0 or 1!

vector+vector: $\begin{bmatrix} M \\ 0 \end{bmatrix} + \begin{bmatrix} M \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} M \\ 0 \end{bmatrix}$

vector-vector: $\begin{bmatrix} M \\ 0 \end{bmatrix} - \begin{bmatrix} M \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} M \\ 0 \end{bmatrix}$

scalar*vector: $s \begin{bmatrix} M \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} M \\ 0 \end{bmatrix}$

point+vector: $\begin{bmatrix} M \\ 1 \end{bmatrix} + \begin{bmatrix} M \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} M \\ 1 \end{bmatrix}$

point-point: $\begin{bmatrix} M \\ 1 \end{bmatrix} - \begin{bmatrix} M \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} M \\ 0 \end{bmatrix}$

point+point: $\begin{bmatrix} M \\ 1 \end{bmatrix} + \begin{bmatrix} M \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} M \\ 2 \end{bmatrix}$

scalar*point: $s \begin{bmatrix} M \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} M \\ s \end{bmatrix}$

$\left\{ \begin{array}{l} \text{weighted average} \\ \text{affine combination} \end{array} \right\}$ of points: $\frac{1}{3} \begin{bmatrix} M \\ 1 \end{bmatrix} + \frac{2}{3} \begin{bmatrix} M \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} M \\ 1 \end{bmatrix}$

Homogeneous Transforms

- ▶ Rotation, Scale, and Translation of points and vectors unified in a single matrix transformation:

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

- ▶ Matrix has the form:
 - ▶ Last row always 0,0,0,1

$$\begin{bmatrix} m_{xx} & m_{xy} & m_{xz} & d_x \\ m_{yx} & m_{yy} & m_{yz} & d_y \\ m_{zx} & m_{zy} & m_{zz} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- ▶ Transforms can be composed by matrix multiplication
 - ▶ Same caveat: order of operations is important
 - ▶ Same note: transforms operate right-to-left

4x4 Scale Matrix

► Generic form:

$$\begin{bmatrix} s & 0 & 0 & 0 \\ 0 & t & 0 & 0 \\ 0 & 0 & u & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

► Inverse:

$$\begin{bmatrix} \frac{1}{s} & 0 & 0 & 0 \\ 0 & \frac{1}{t} & 0 & 0 \\ 0 & 0 & \frac{1}{u} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4x4 Rotation Matrix

► Generic form:

$$\begin{bmatrix} r_1 & r_2 & r_3 & 0 \\ r_4 & r_5 & r_6 & 0 \\ r_7 & r_8 & r_9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

► Inverse:

$$\begin{bmatrix} r_1 & r_4 & r_7 & 0 \\ r_2 & r_5 & r_8 & 0 \\ r_3 & r_6 & r_9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4x4 Translation Matrix

- Generic form:

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Inverse:

$$\begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Today

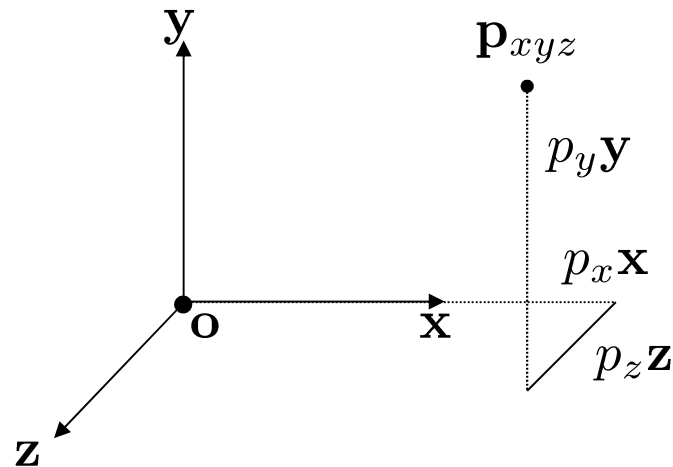
- ▶ **Coordinate Transformation**
- ▶ Typical Coordinate Systems

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**:

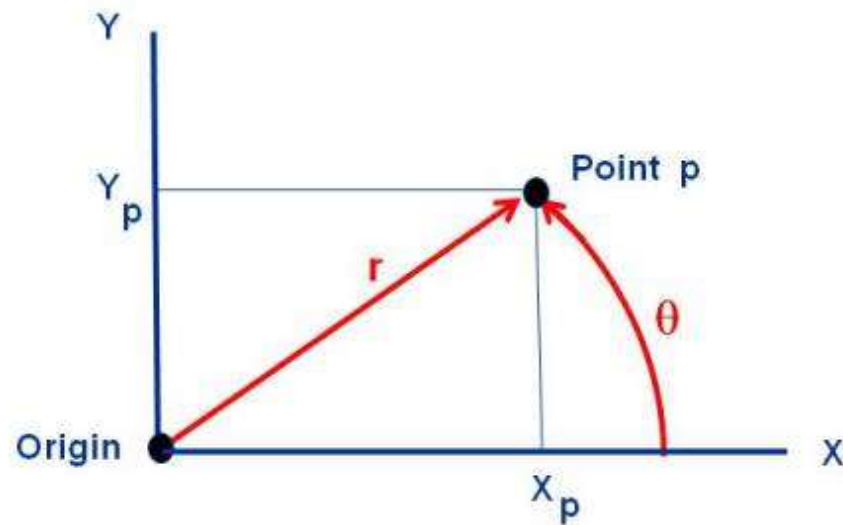


$$\mathbf{p}_{xyz} = p_x\mathbf{x} + p_y\mathbf{y} + p_z\mathbf{z} + \mathbf{o}$$

Rectangular and Polar Coordinates

National Aeronautics and Space Administration

Rectangular and Polar Coordinates



Point p can be located relative to the origin by Rectangular Coordinates (X_p, Y_p) or by Polar Coordinates (r, θ)

$$X_p = r \cos(\theta)$$

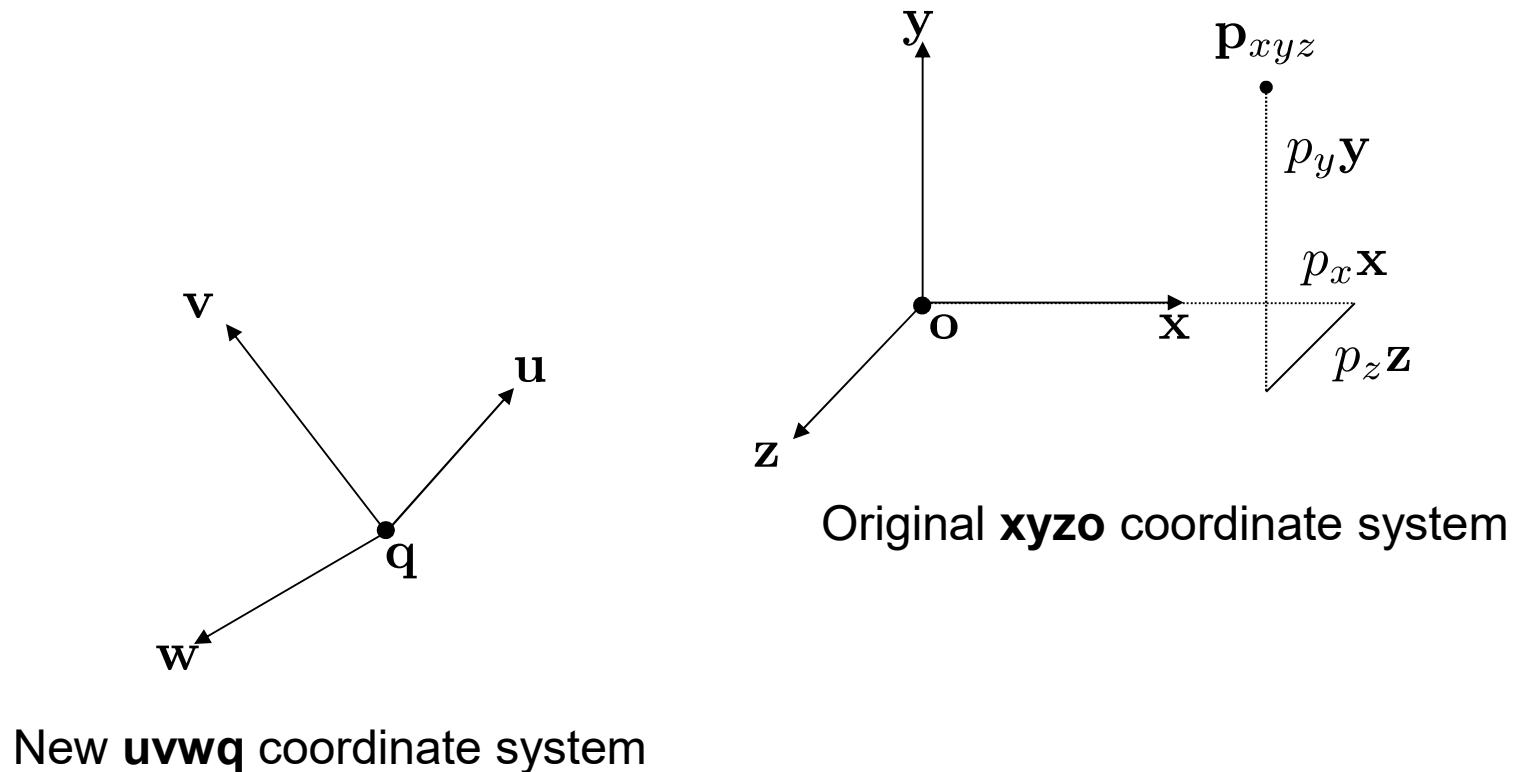
$$Y_p = r \sin(\theta)$$

$$r = \sqrt{X_p^2 + Y_p^2}$$

$$\theta = \tan^{-1}(Y_p / X_p)$$

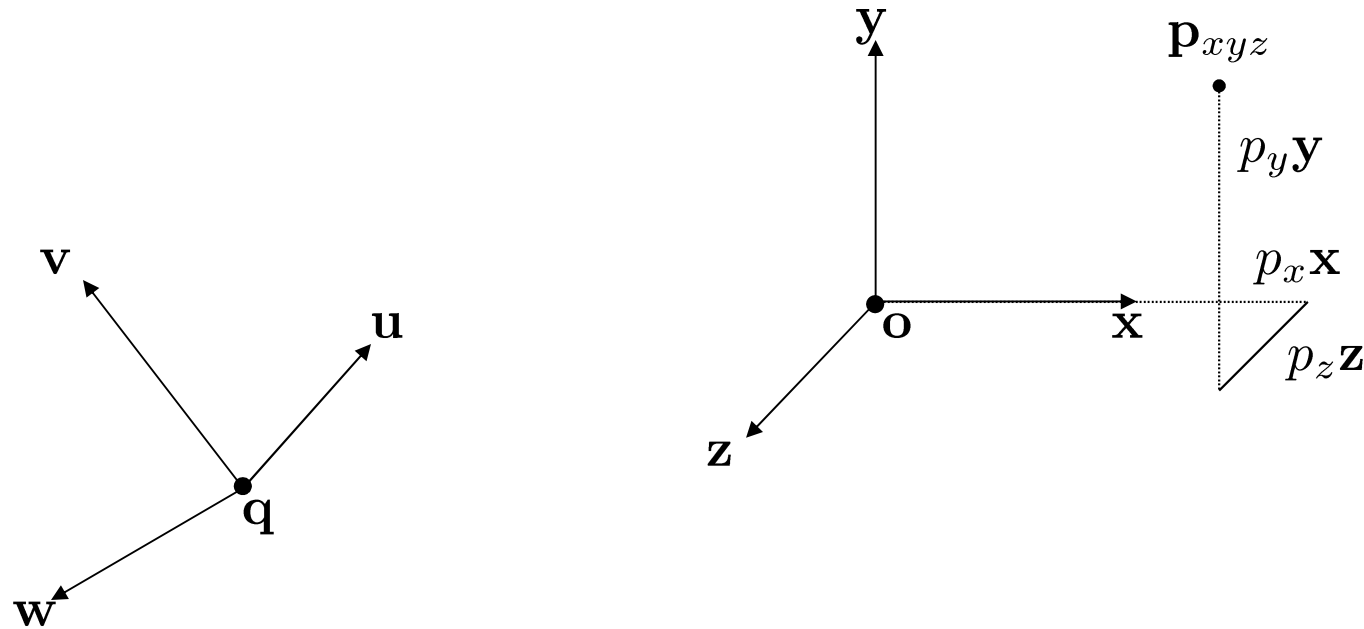
www.nasa.gov 37

Coordinate Transformation



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:

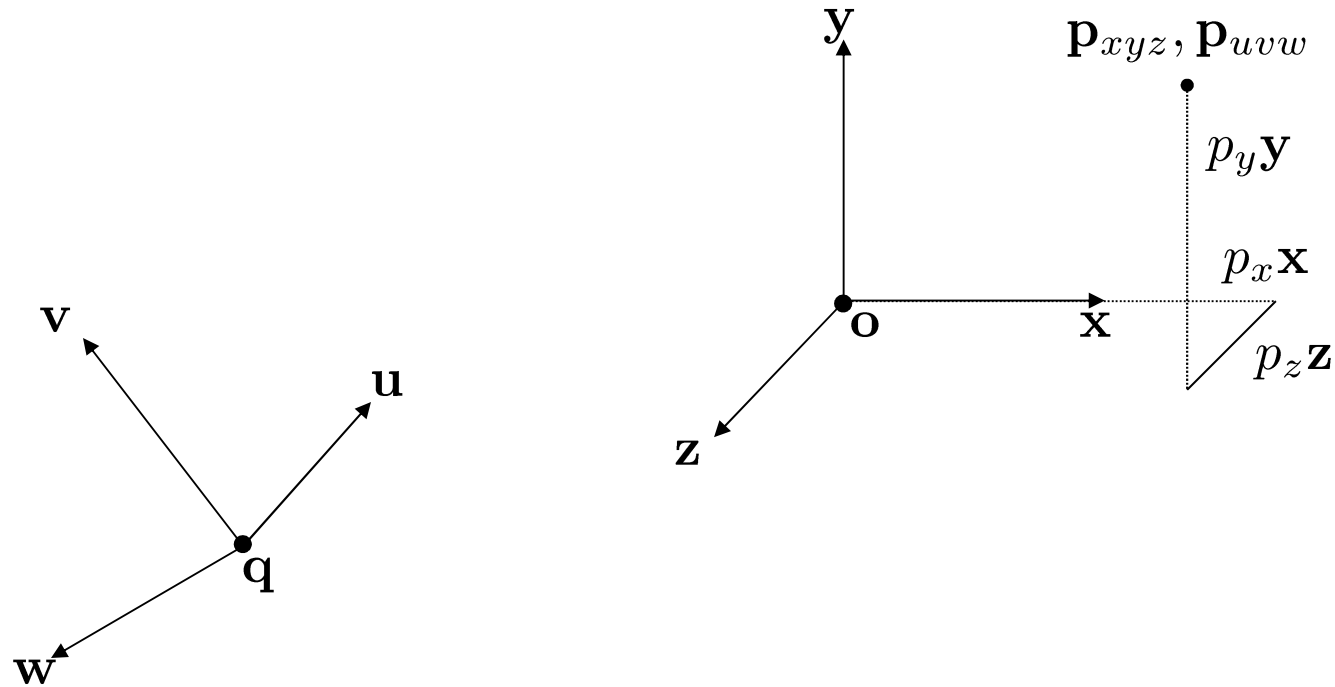
$$\mathbf{x} = \begin{bmatrix} x_u \\ x_v \\ x_w \\ 0 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} y_u \\ y_v \\ y_w \\ 0 \end{bmatrix}$$

$$\mathbf{z} = \begin{bmatrix} z_u \\ z_v \\ z_w \\ 0 \end{bmatrix}$$

$$\mathbf{o} = \begin{bmatrix} o_u \\ o_v \\ o_w \\ 1 \end{bmatrix}$$

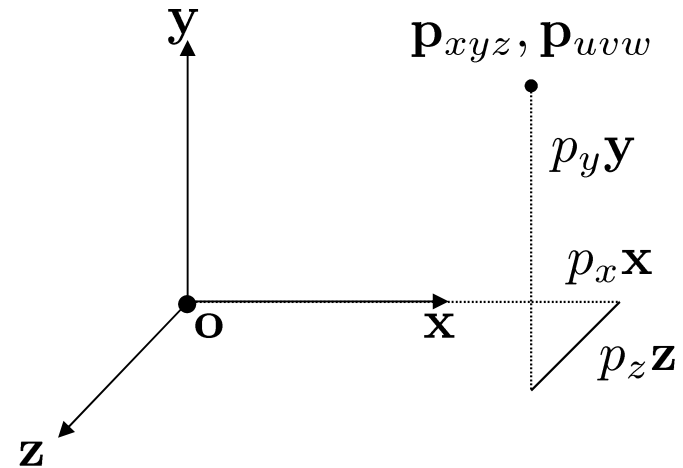
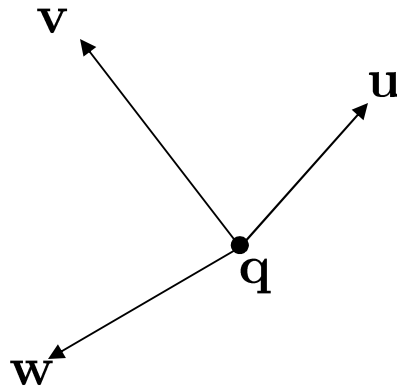
Coordinate Transformation



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

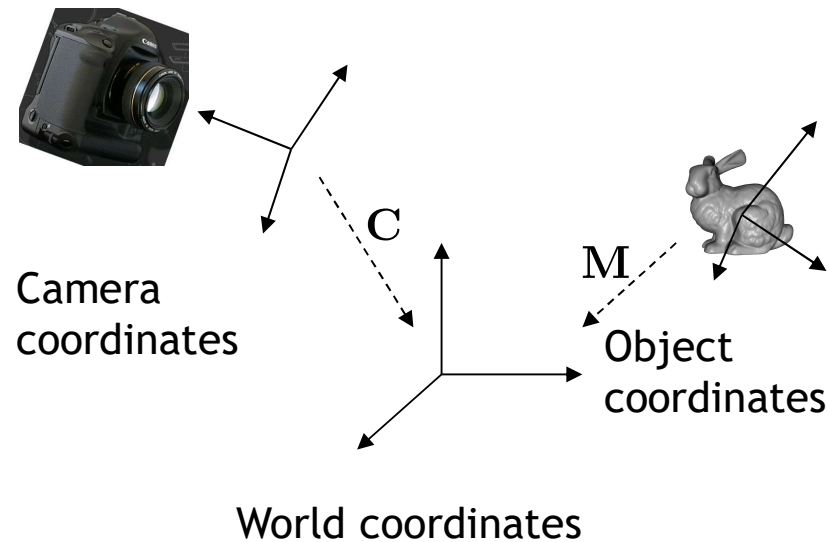
$$\mathbf{P}_{xyz} = \begin{bmatrix} x_u & y_u & z_u & o_u \\ x_v & y_v & z_v & o_v \\ x_w & y_w & z_w & o_w \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} p_u \\ p_v \\ p_w \\ 1 \end{bmatrix}$$

Lecture Overview

- ▶ Coordinate Transformation
- ▶ Typical Coordinate Systems

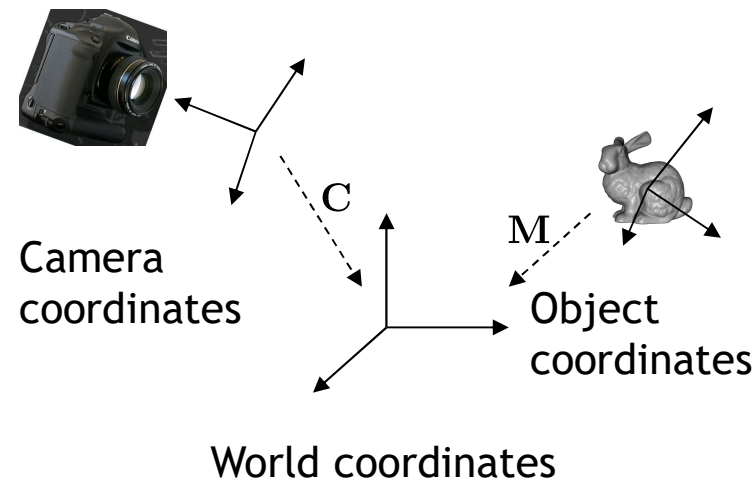
Typical Coordinate Systems

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



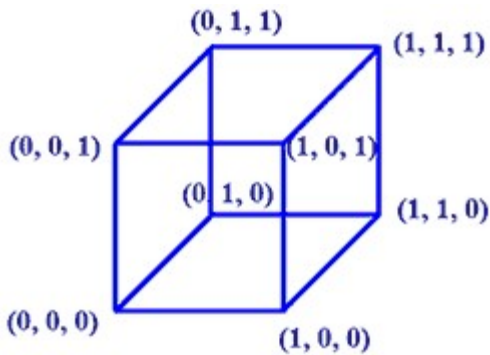
World Coordinates

- ▶ 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

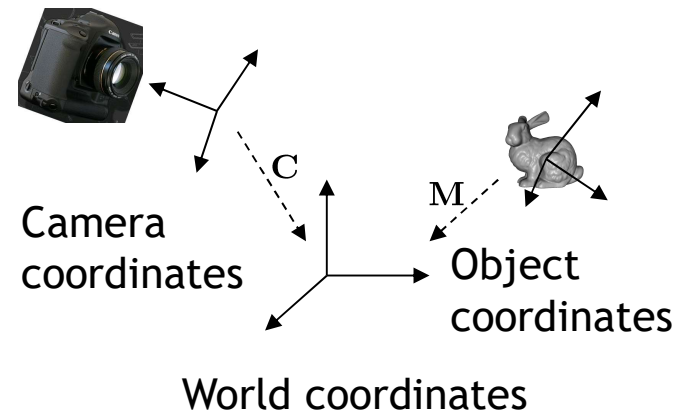


Object Coordinates

- ▶ 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.

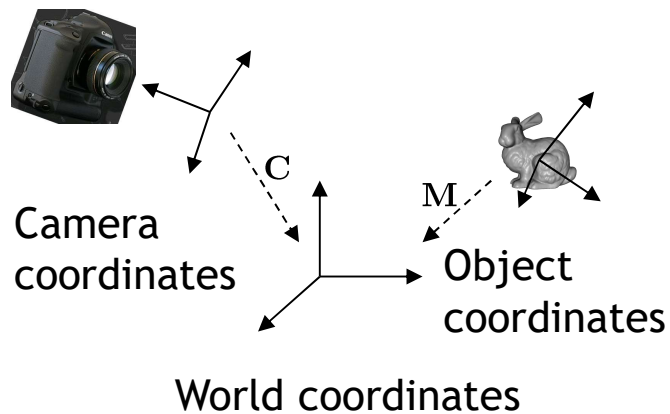


Source: <http://motivate.maths.org>



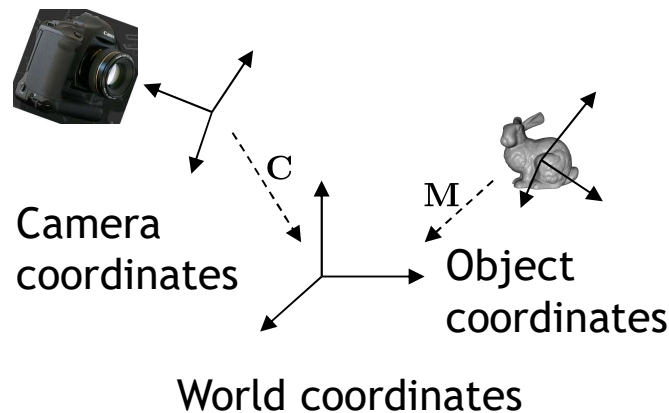
Object Transformation

- ▶ 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**.



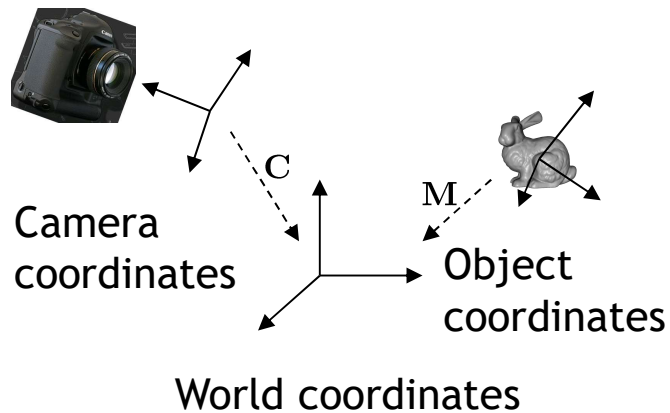
Camera Coordinate System

- ▶ 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 \mathbf{e}
- Look at point \mathbf{d}
- Camera up vector \mathbf{u}

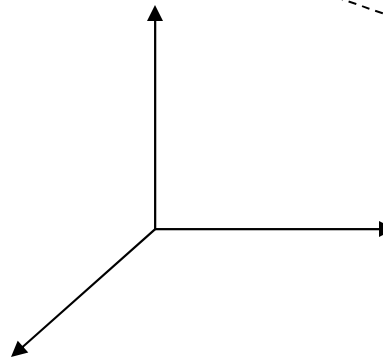


\mathbf{u}
 \mathbf{e}



\mathbf{d}

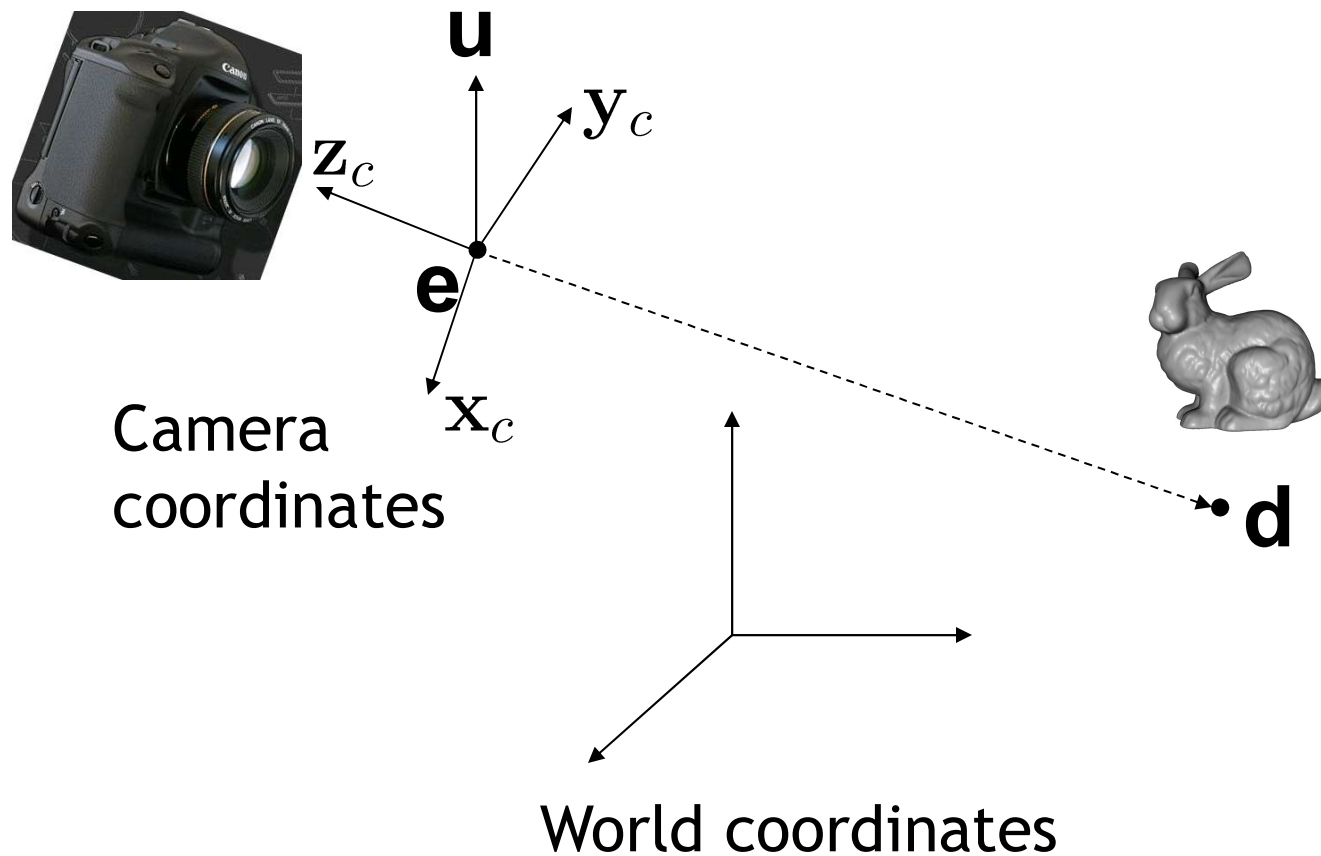
Camera
coordinates



World coordinates

Camera Matrix

- Construct \mathbf{x}_c , \mathbf{y}_c , \mathbf{z}_c



Camera Matrix

- ▶ Step 1: z-axis

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

- ▶ Step 2: x-axis

$$\mathbf{x}_C = \frac{\mathbf{u} \times \mathbf{z}_C}{\|\mathbf{u} \times \mathbf{z}_C\|}$$

- ▶ Step 3: y-axis

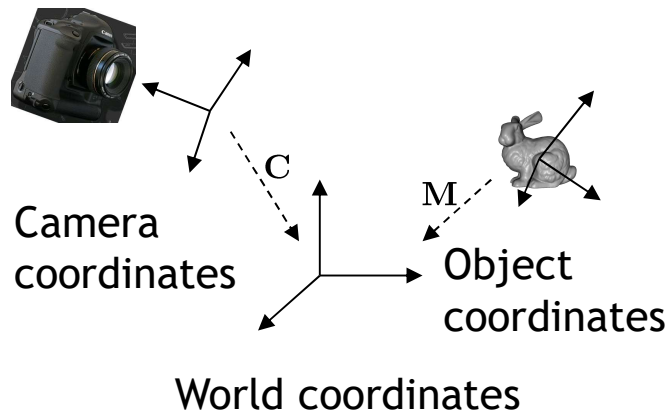
$$\mathbf{y}_C = \mathbf{z}_C \times \mathbf{x}_C = \frac{\mathbf{u}}{\|\mathbf{u}\|}$$

- ▶ Camera Matrix:

$$\mathbf{C} = \begin{bmatrix} \mathbf{x}_C & \mathbf{y}_C & \mathbf{z}_C & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transforming Object to Camera Coordinates

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



Tips for Notation

- ▶ Indicate coordinate systems with every point or matrix

- ▶ Point: $\mathbf{p}_{\text{object}}$

- ▶ Matrix: $\mathbf{M}_{\text{object} \rightarrow \text{world}}$

- ▶ Resulting transformation equation:

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

- ▶ In source 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