Geometry Representation

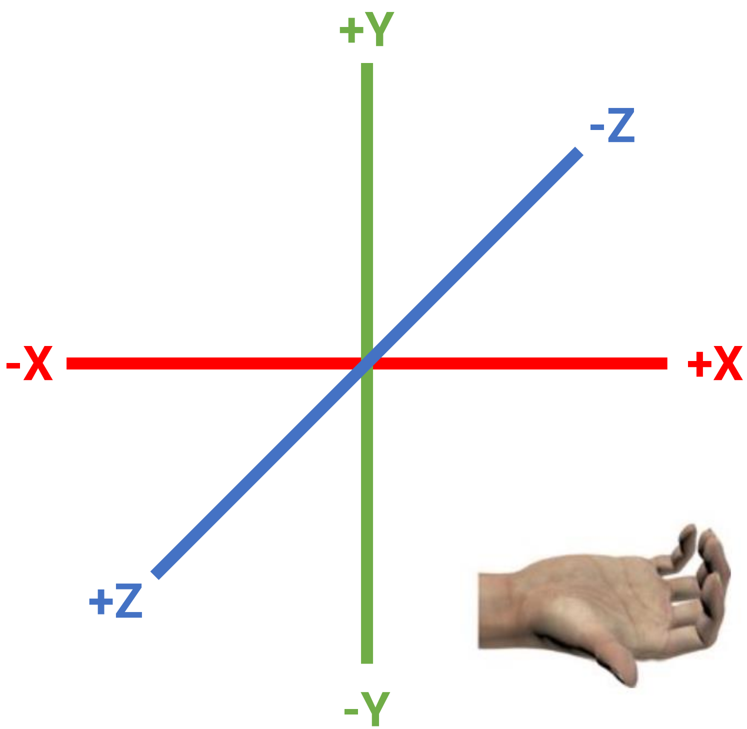
Introduction to Computer Graphics

Yu-Ting Wu

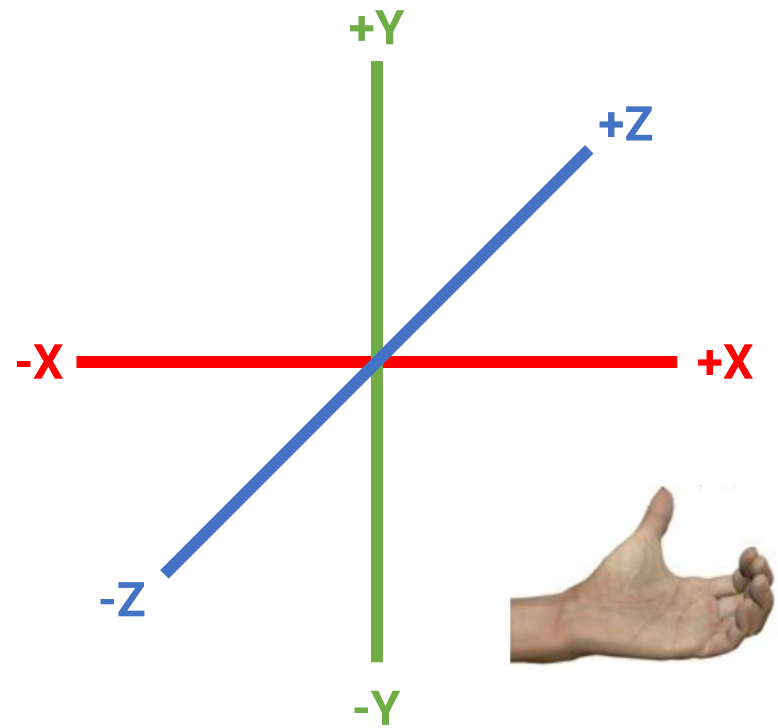
Define the 3D World

Description of the 3D World

- 3D coordinate systems

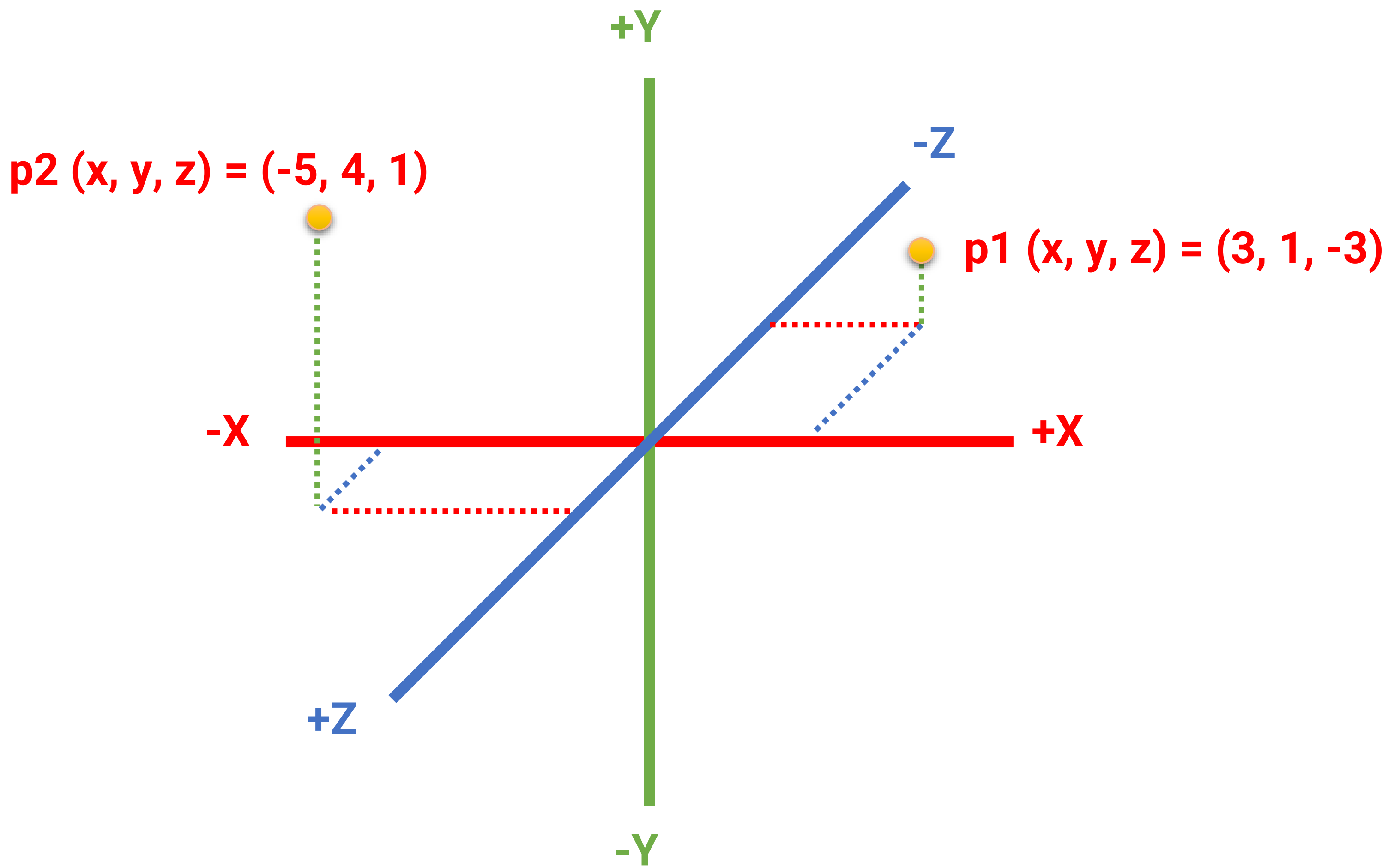


OpenGL
(Right-Hand-Side)



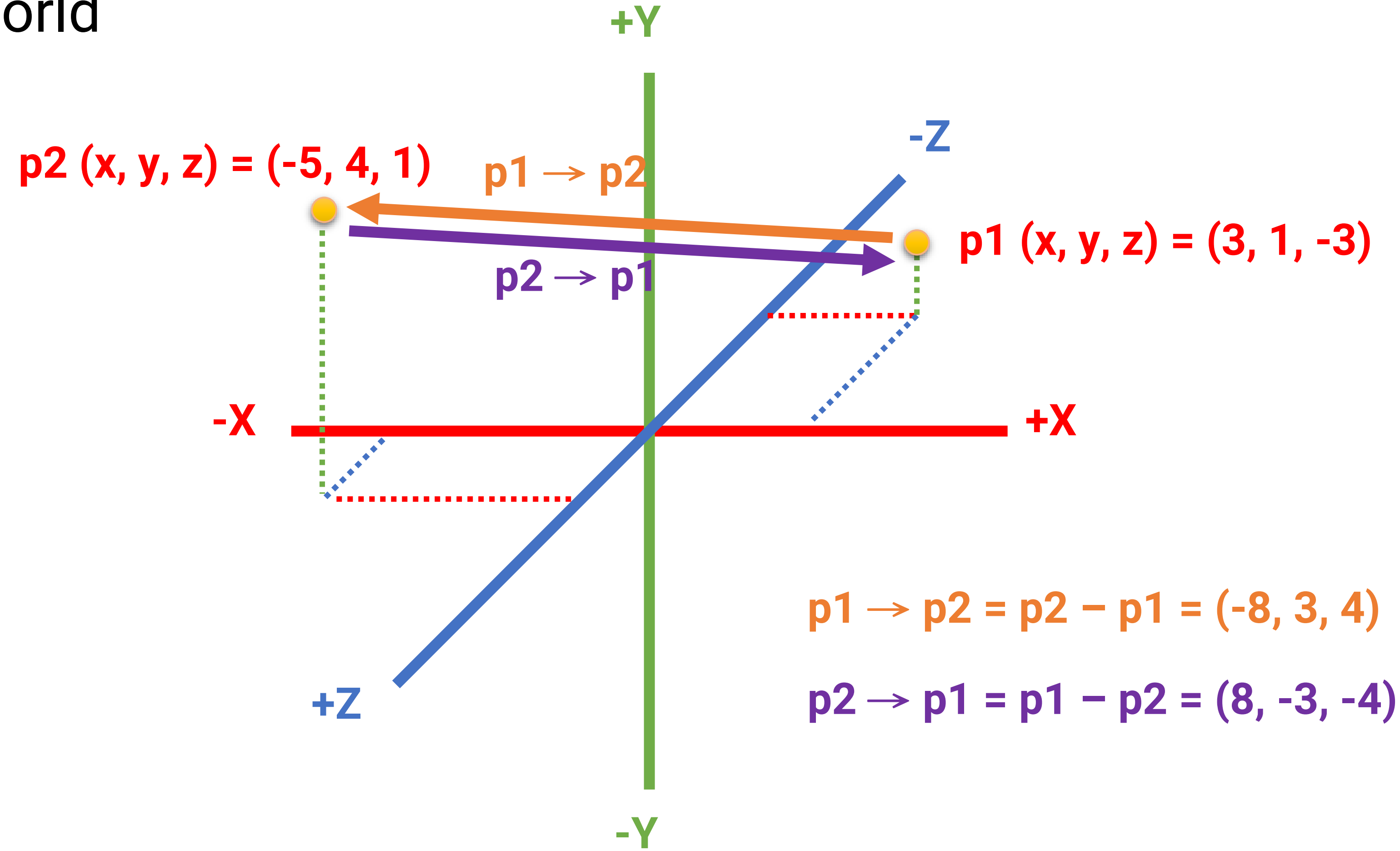
DirectX
(Left-Hand-Side)

Points in 3D

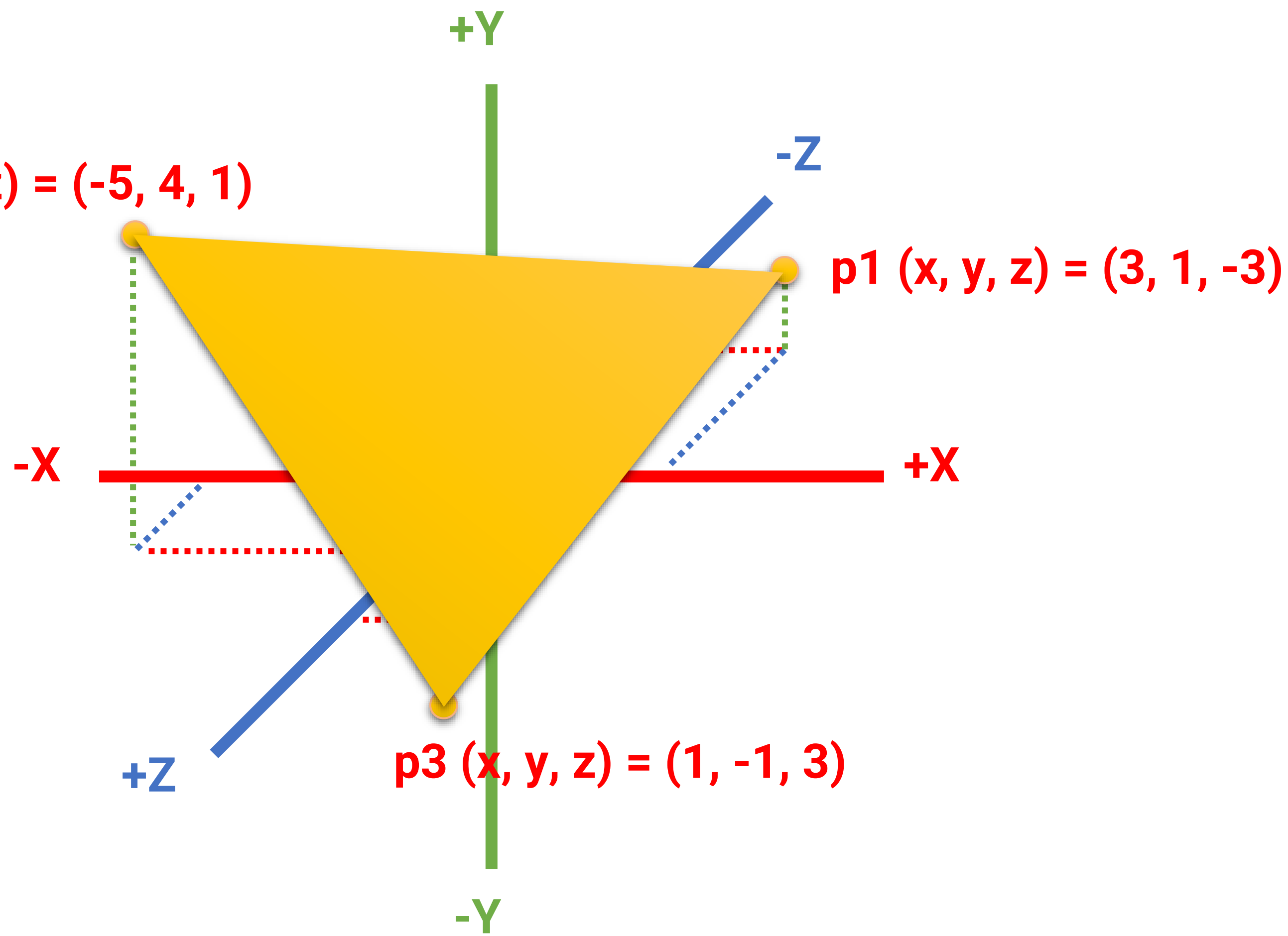


Vector in 3D Space

- Use to represent direction (e.g., movement) in the 3D world

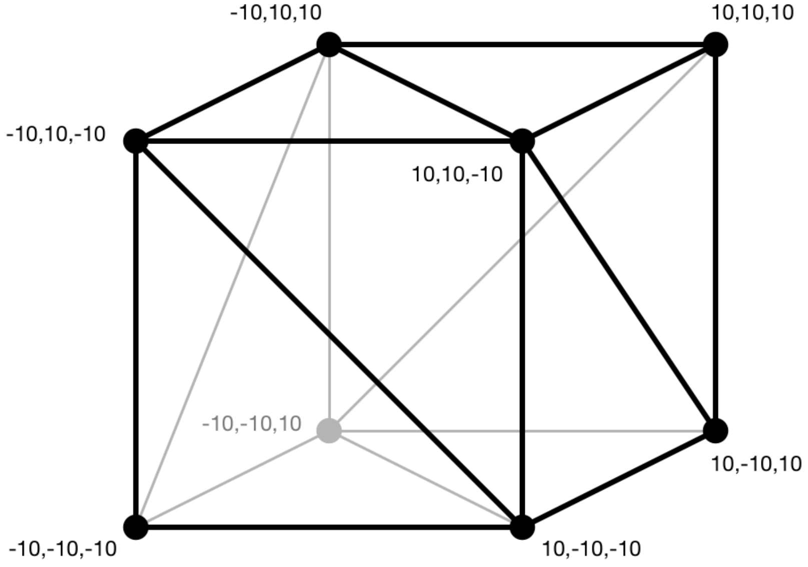


Triangles in 3D

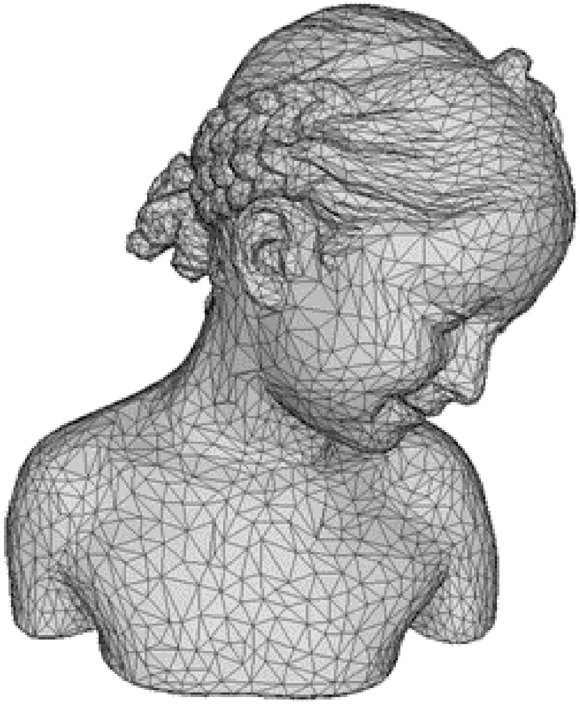


Triangle Mesh

- We can define the geometry of an object by specifying the coordinates of the vertices and their adjacencies



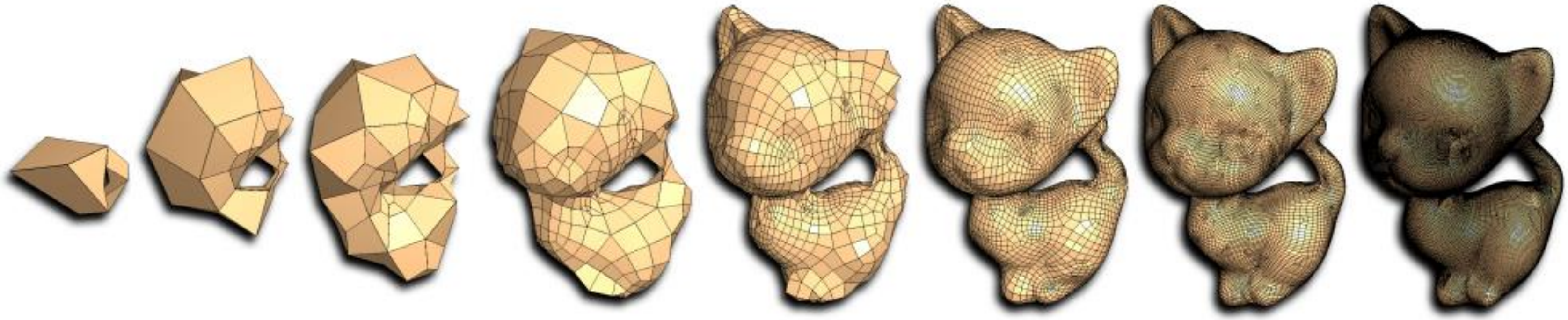
12 triangles

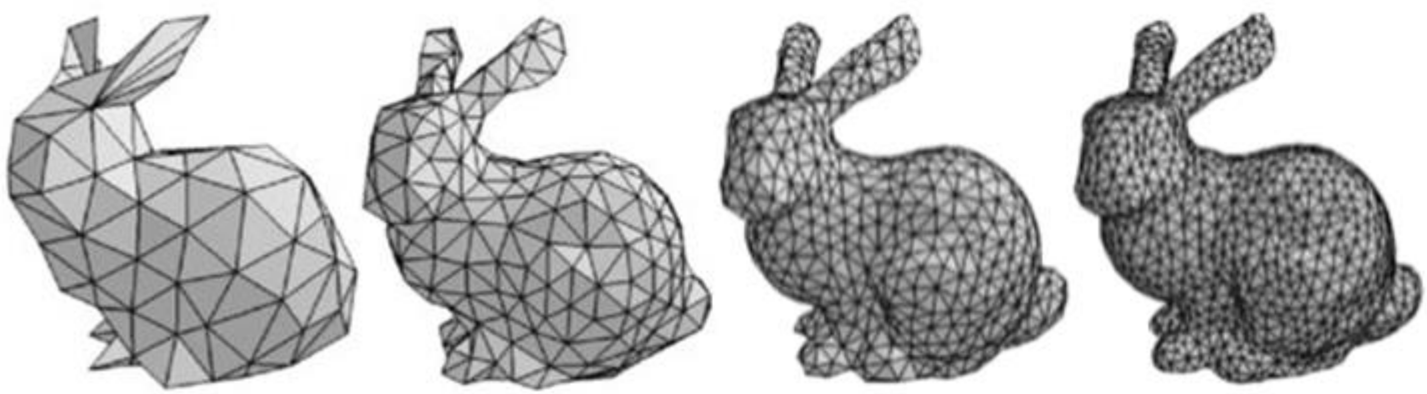


10K triangles

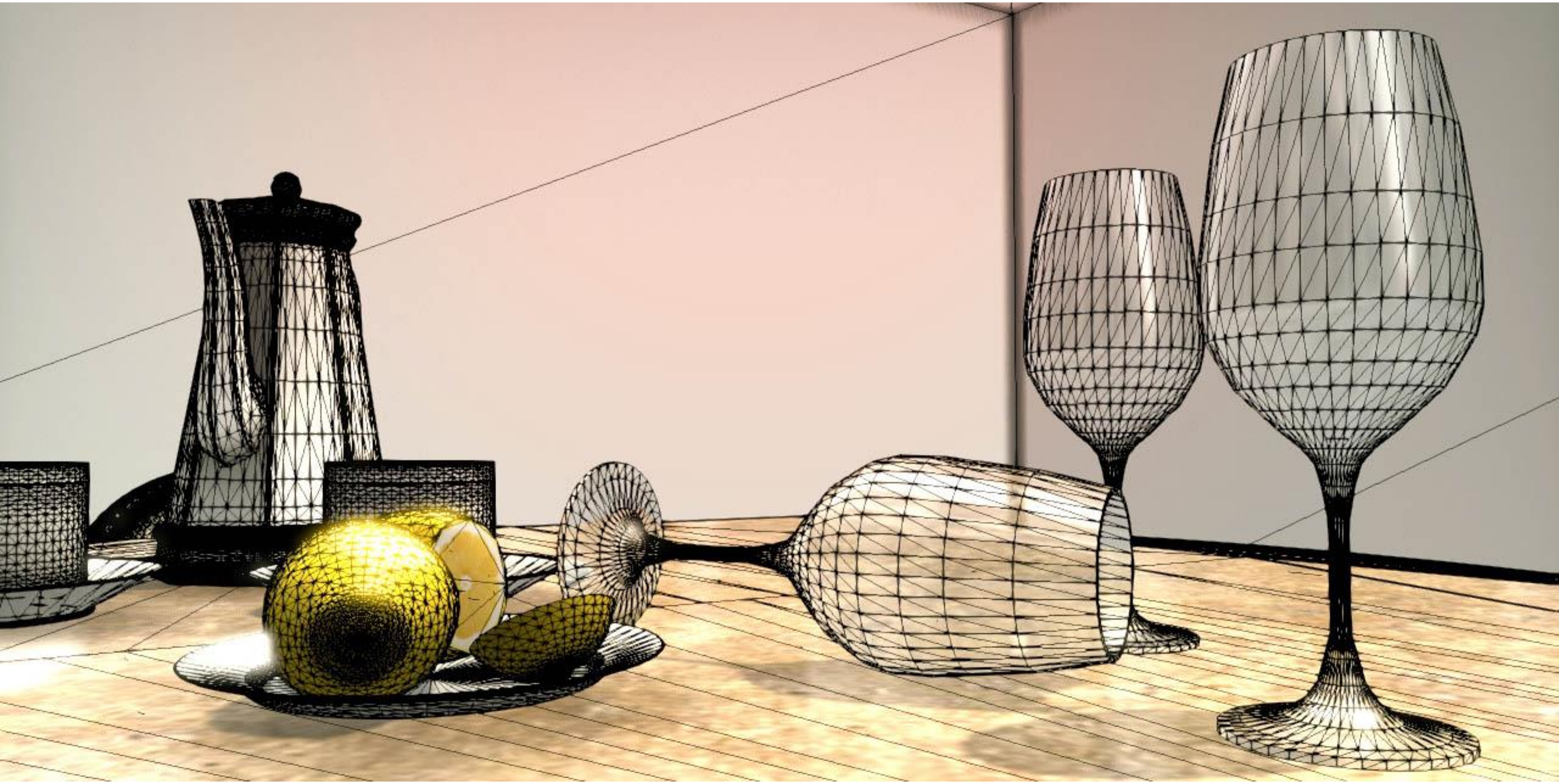
Triangle Mesh (cont.)

- Using more triangles can lead to higher-quality meshes
  - However, takes more time to render



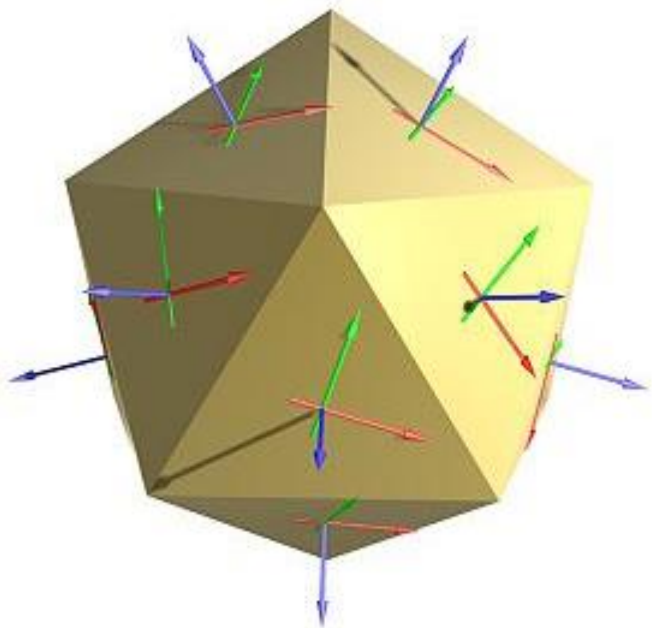


Scene Built with Triangle Mesh



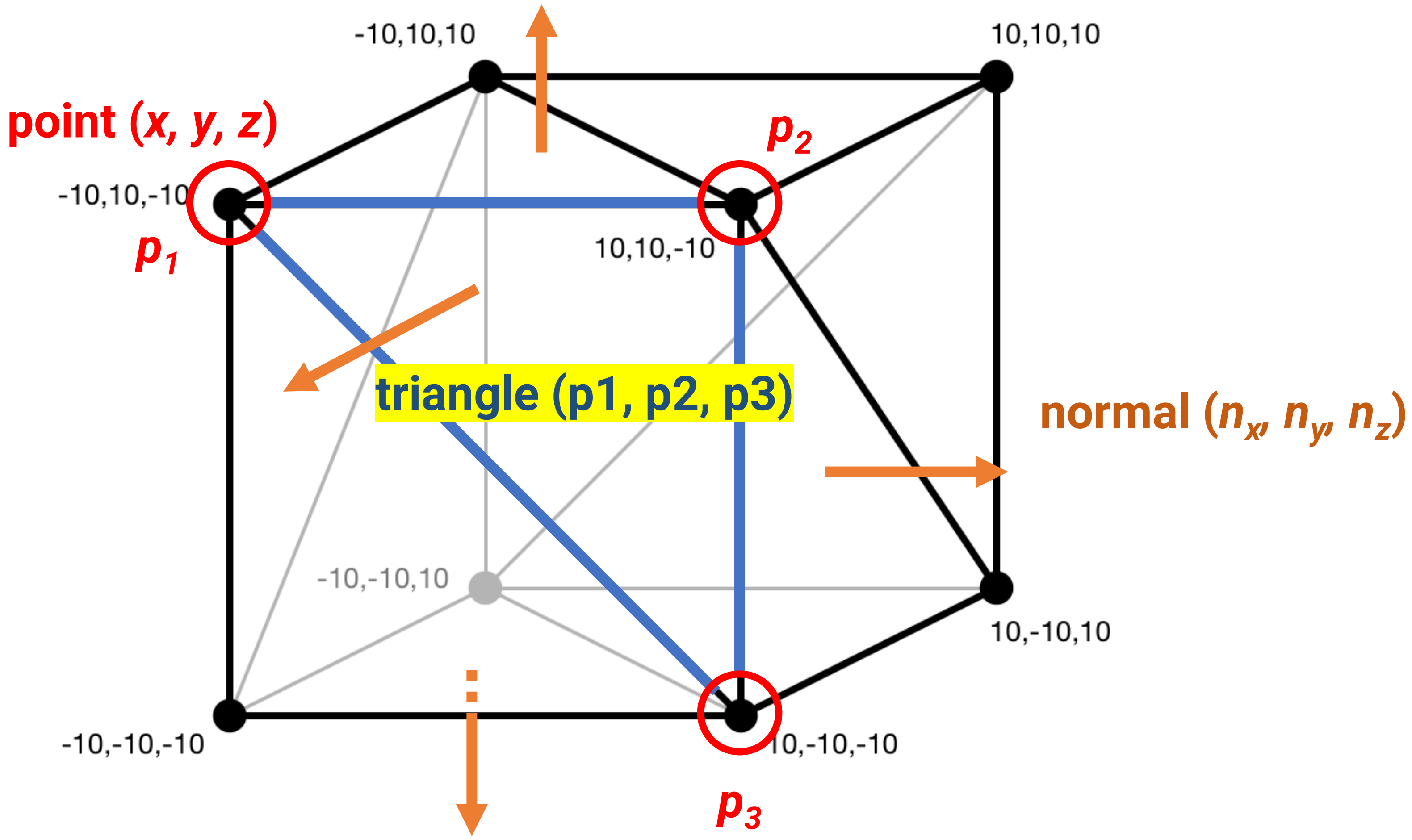
Surface Normal

- A **surface normal** is a vector that is **perpendicular** to a surface at a particular position
- Represent the orientation of the face
- The length of a normal should be equal to **1**



→ **normal (n_x, n_y, n_z)**
→ tangent
→ binormal

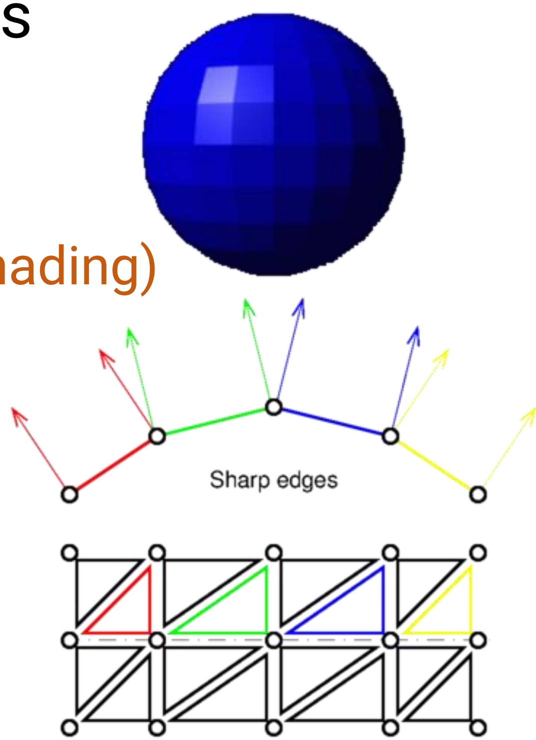
Point, Triangle, and Surface Normal

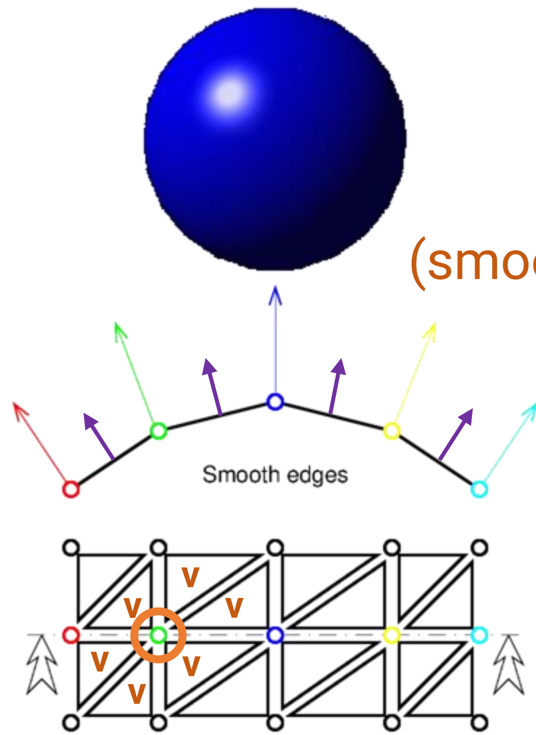


Vertex Normal

- Compute by **averaging** the surface normals of the faces that contain that vertex
- Can achieve much **smooth** shading than using triangle normals

(flat shading)





(smooth shading)

3D Model Format

- A model is often stored in a file
- Common file format includes
  - **Wavefront (*.obj)**
  - Polygon file format (*.ply)
  - **Filmbox (*.fbx)**
  - MAX (*.max)
  - Digital Asset Exchange File (*.dae)
  - STereoLithography (*.stl)

Example: Wavefront OBJ File Format

- cube.obj

cube.obj - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
# Unit-volume cube with the same texture coordinates on each face.
#
# Created by Morgan McGuire and released into the Public Domain on
# July 16, 2011.
#
# http://graphics.cs.williams.edu/data
```

comments

```
mtllib default.mtl
```

specify material file

```
v -0.5 0.5 -0.5
v -0.5 0.5 0.5
v 0.5 0.5 0.5
v 0.5 0.5 -0.5
v -0.5 -0.5 -0.5
v -0.5 -0.5 0.5
v 0.5 -0.5 0.5
v 0.5 -0.5 -0.5
```

vertex position declaration

```
vt 0 1
vt 0 0
vt 1 0
vt 1 1
```

vertex texture coordinate declaration

```
vn 0 1 0
vn -1 0 0
vn 1 0 0
vn 0 0 -1
vn 0 0 1
vn 0 -1 0
```

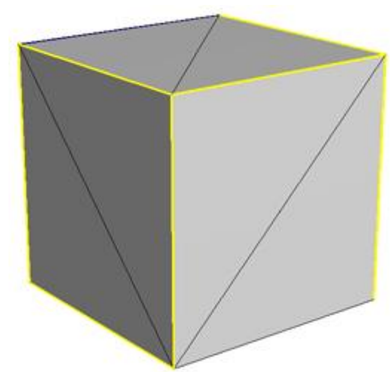
vertex normal declaration

第 39 列，第 29 行 160% Unix (LF) UTF-8

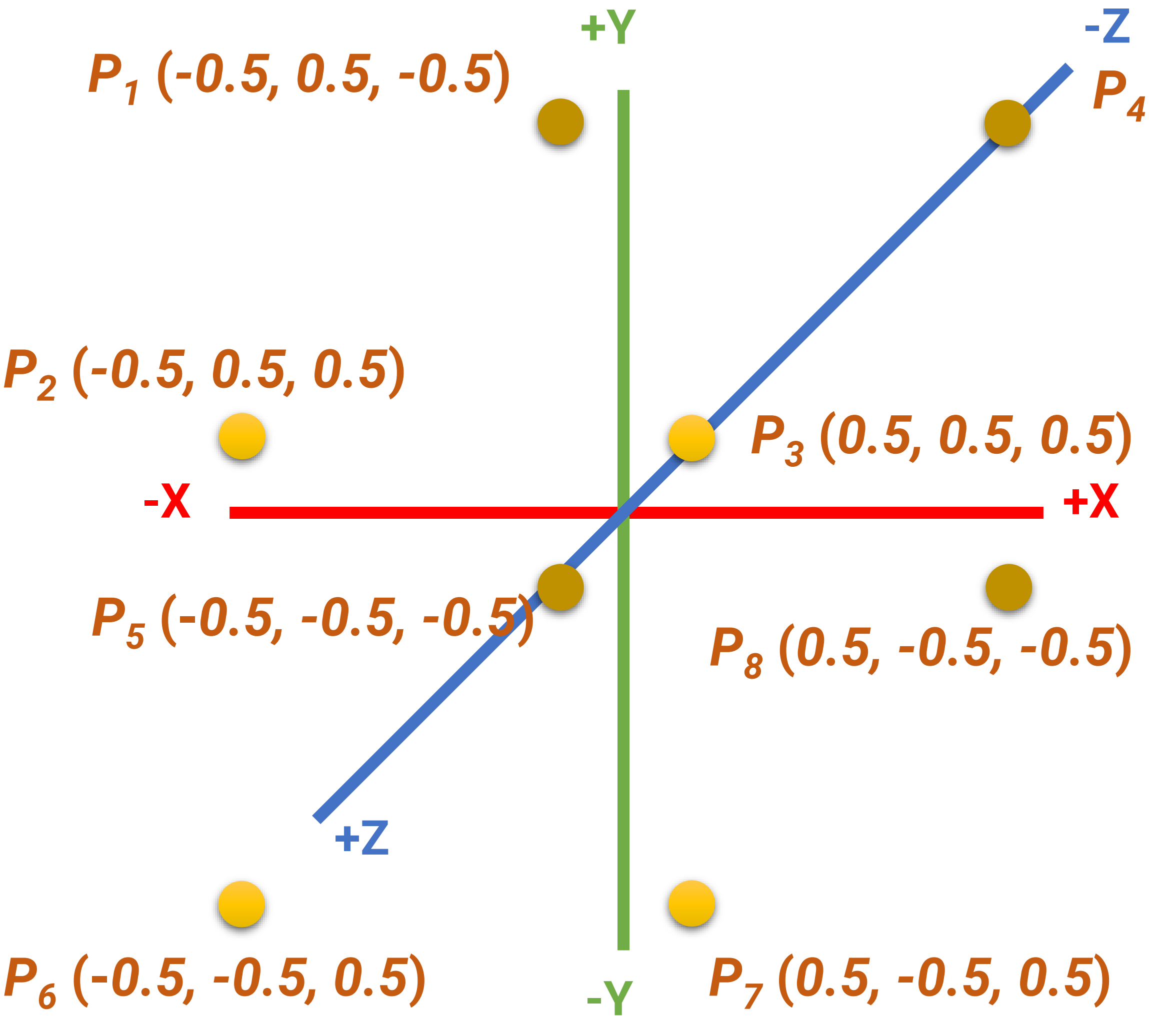
```
g cube
usemtl default

f -8/-4/-6 -7/-3/-6 -6/-2/-6
f -8/-4/-6 -6/-2/-6 -5/-1/-6
f -8/-4/-5 -4/-3/-5 -3/-2/-5
f -8/-4/-5 -3/-2/-5 -7/-1/-5
f -6/-4/-4 -2/-3/-4 -1/-2/-4
f -6/-4/-4 -1/-2/-4 -5/-1/-4
f -5/-4/-3 -1/-3/-3 -4/-2/-3
f -5/-4/-3 -4/-2/-3 -8/-1/-3
f -7/-4/-2 -3/-3/-2 -2/-2/-2
f -7/-4/-2 -2/-2/-2 -6/-1/-2
f -3/-4/-1 -4/-3/-1 -1/-2/-1
f -3/-4/-1 -1/-2/-1 -2/-1/-1
```

face data (adjacency, submesh)

第 20 列，第 7 行 160% Unix (LF) UTF-8

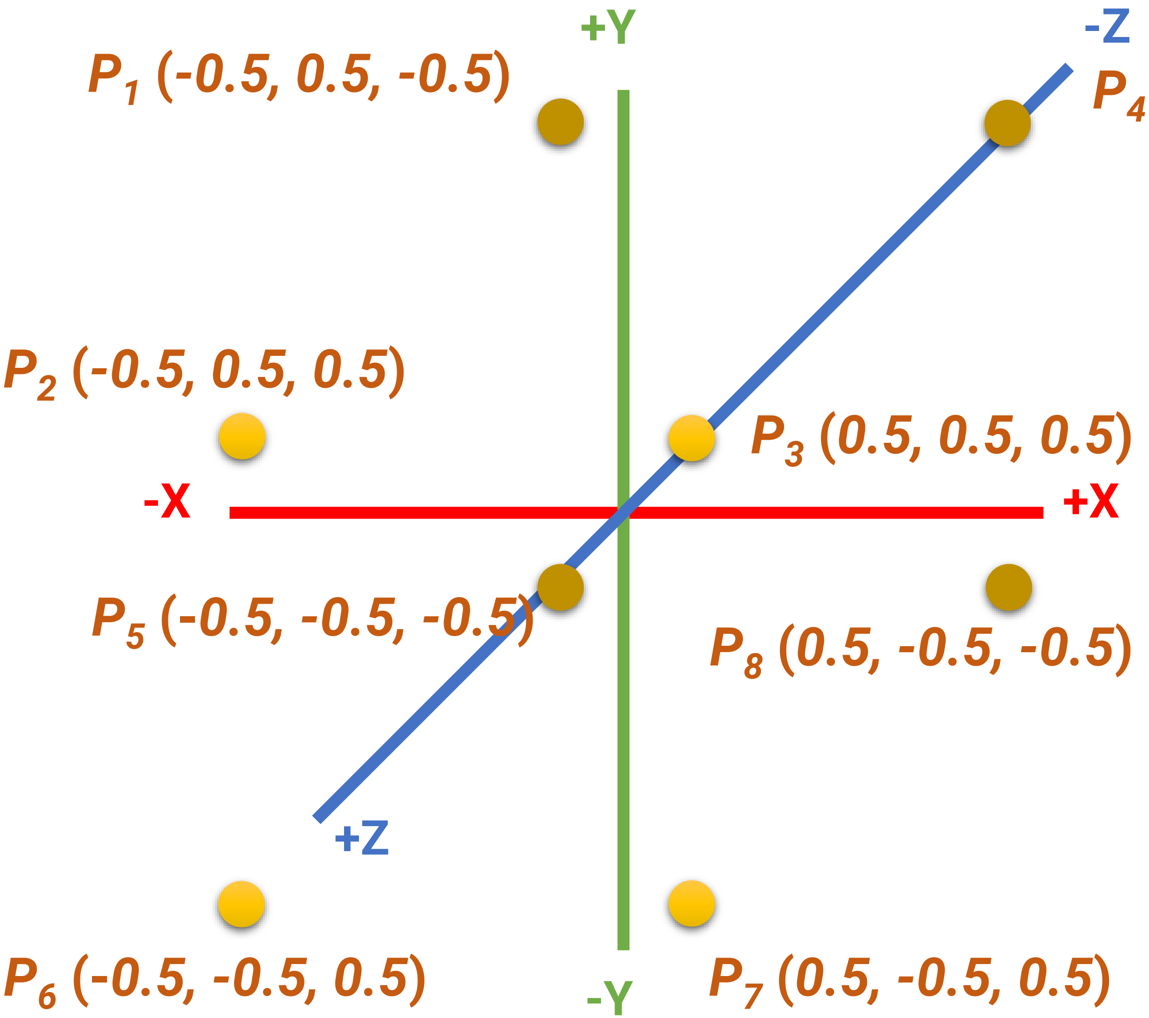


Example: Wavefront OBJ File Format (cont.)



P_1	v -0.5 0.5 -0.5
P_2	v -0.5 0.5 0.5
P_3	v 0.5 0.5 0.5
P_4	v 0.5 0.5 -0.5
P_5	v -0.5 -0.5 -0.5
P_6	v -0.5 -0.5 0.5
P_7	v 0.5 -0.5 0.5
P_8	v 0.5 -0.5 -0.5

Example: Wavefront OBJ File Format (cont.)



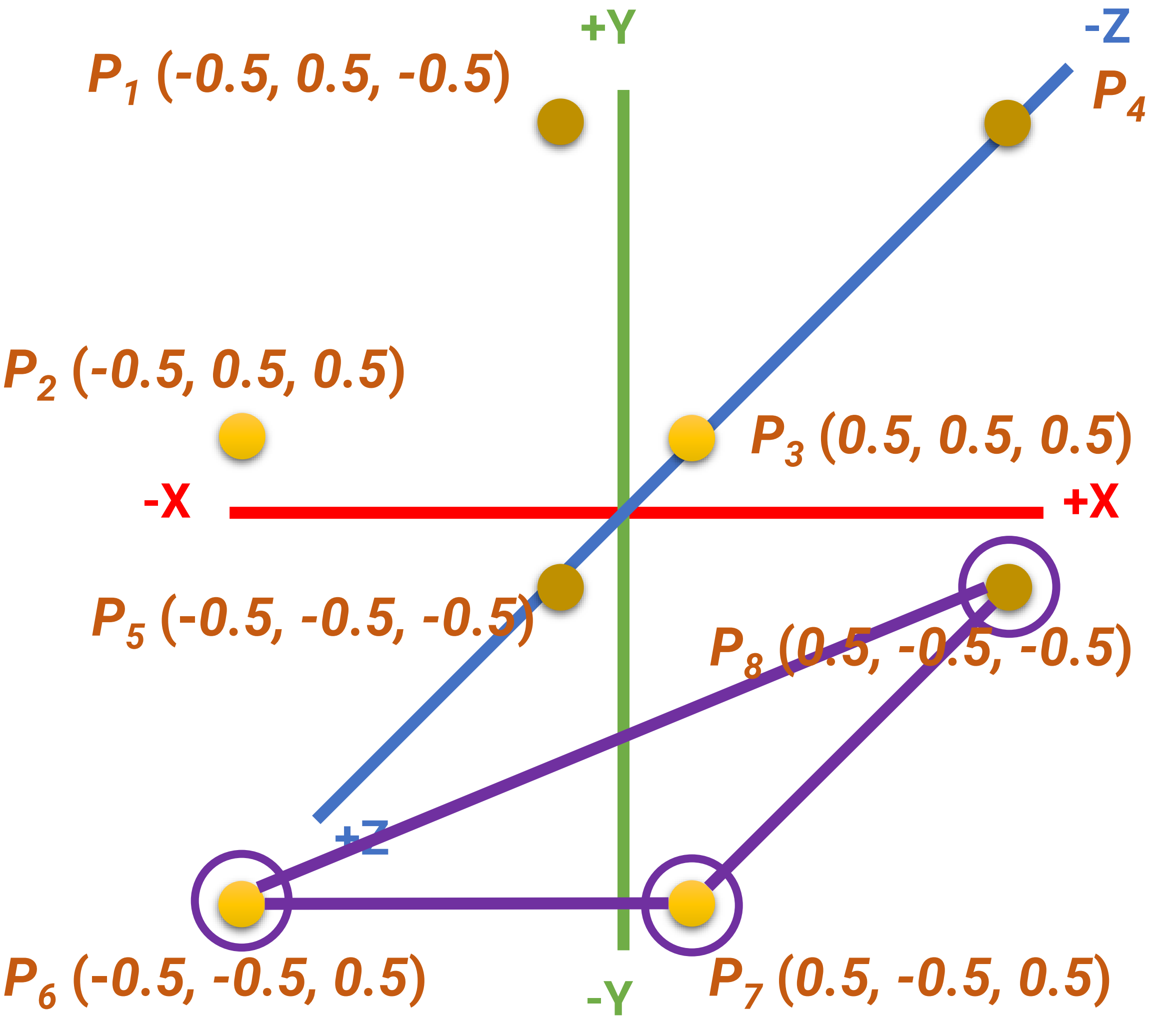
F_1	f -8/-4/-6 -7/-3/-6 -6/-2/-6
	f -8/-4/-6 -6/-2/-6 -5/-1/-6
F_2	f -8/-4/-5 -4/-3/-5 -3/-2/-5
	f -8/-4/-5 -3/-2/-5 -7/-1/-5
F_3	f -6/-4/-4 -2/-3/-4 -1/-2/-4
	f -6/-4/-4 -1/-2/-4 -5/-1/-4
F_4	f -5/-4/-3 -1/-3/-3 -4/-2/-3
	f -5/-4/-3 -4/-2/-3 -8/-1/-3
F_5	f -7/-4/-2 -3/-3/-2 -2/-2/-2
	f -7/-4/-2 -2/-2/-2 -6/-1/-2
F_6	f -3/-4/-1 -4/-3/-1 -1/-2/-1
	f -3/-4/-1 -1/-2/-1 -2/-1/-1

F_7 F_8

	vertex1	vertex2	vertex3
f	*P/T/N*	*P/T/N*	*P/T/N*

P: index of vertex position
T: index of texture coordinate
N: index of vertex normal

Example: Wavefront OBJ File Format (cont.)



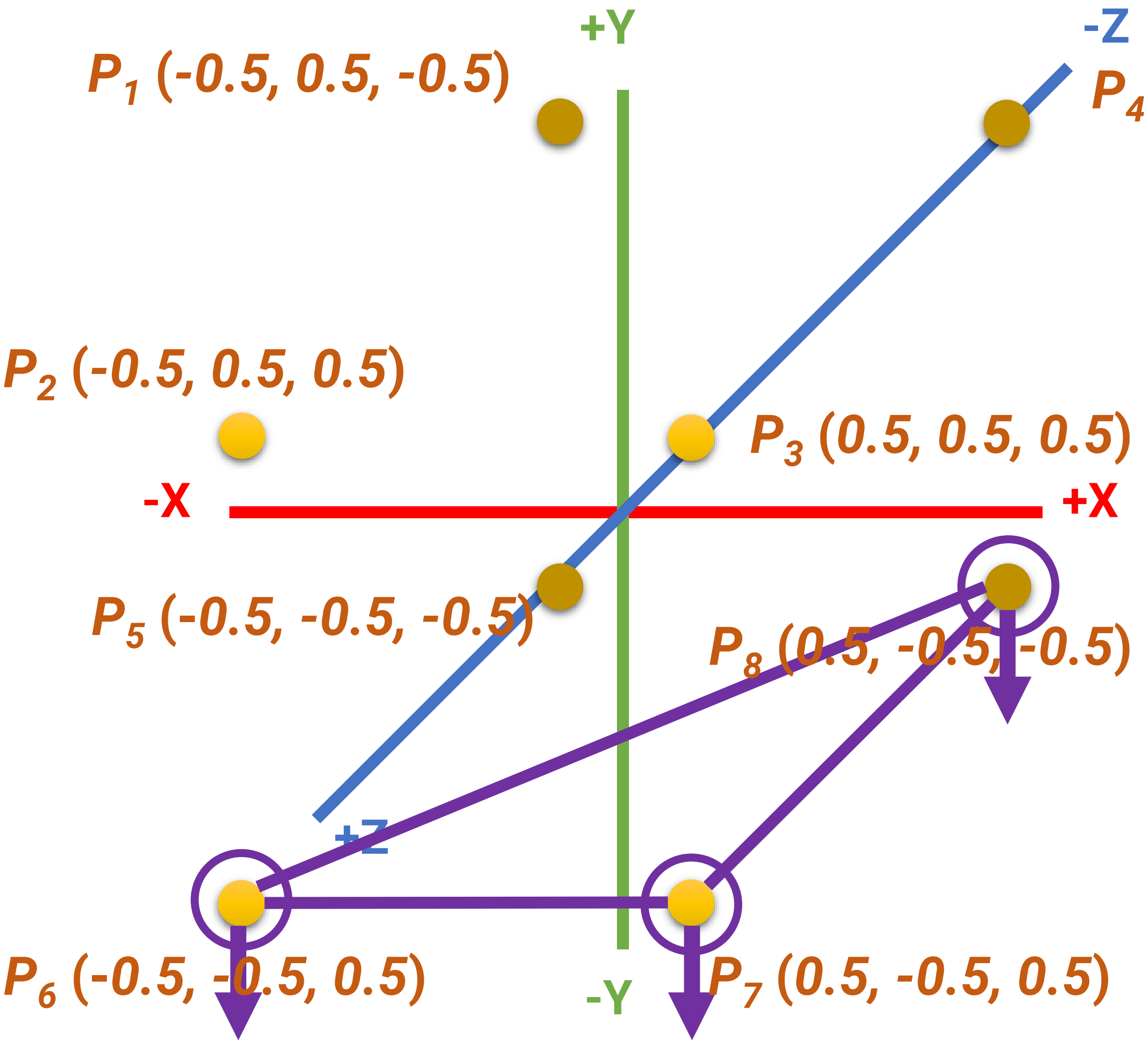
F_1 F_2 F_3 F_4 F_5 F_6 F_7 F_8

```
f -8/-4/-6 -7/-3/-6 -6/-2/-6
f -8/-4/-6 -6/-2/-6 -5/-1/-6
f -8/-4/-5 -4/-3/-5 -3/-2/-5
f -8/-4/-5 -3/-2/-5 -7/-1/-5
f -6/-4/-4 -2/-3/-4 -1/-2/-4
f -6/-4/-4 -1/-2/-4 -5/-1/-4
f -5/-4/-3 -1/-3/-3 -4/-2/-3
f -5/-4/-3 -4/-2/-3 -8/-1/-3
f -7/-4/-2 -3/-3/-2 -2/-2/-2
f -7/-4/-2 -2/-2/-2 -6/-1/-2
f -3/-4/-1 -4/-3/-1 -1/-2/-1
f -3/-4/-1 -1/-2/-1 -2/-1/-1
```

	vertex1	vertex2	vertex3
f	*P/T/N*	*P/T/N*	*P/T/N*

P: index of vertex position
T: index of texture coordinate
N: index of vertex normal

Example: Wavefront OBJ File Format (cont.)



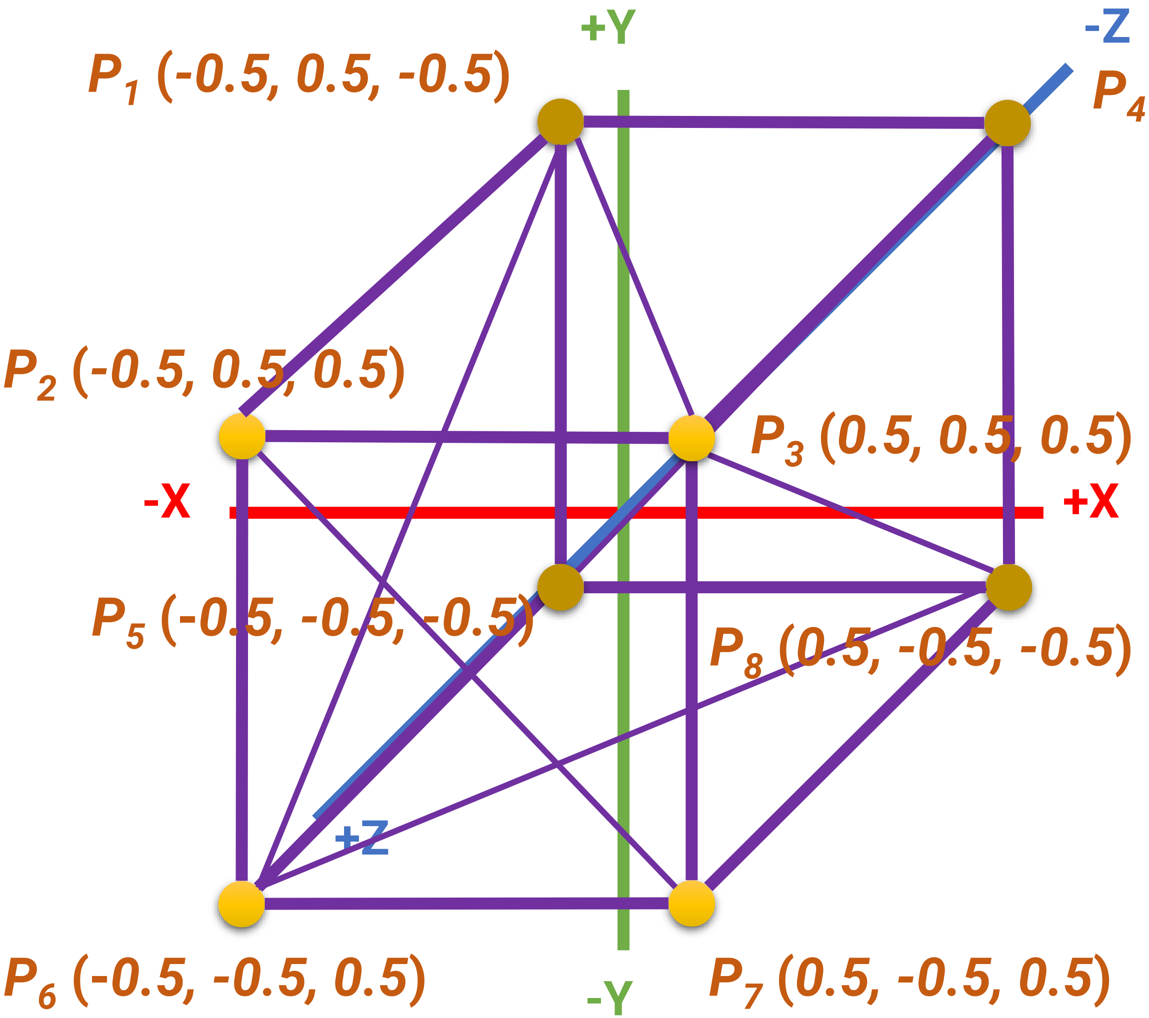
F_1 f -8/-4/-6 -7/-3/-6 -6/-2/-6

N_1 vn 0 1 0
N_2 vn -1 0 0
N_3 vn 1 0 0
N_4 vn 0 0 -1
N_5 vn 0 0 1
N_6 vn 0 -1 0

	vertex1	vertex2	vertex3
f	*P/T/N*	*P/T/N*	*P/T/N*

P: index of vertex position
T: index of texture coordinate
N: index of vertex normal

Example: Wavefront OBJ File Format (cont.)



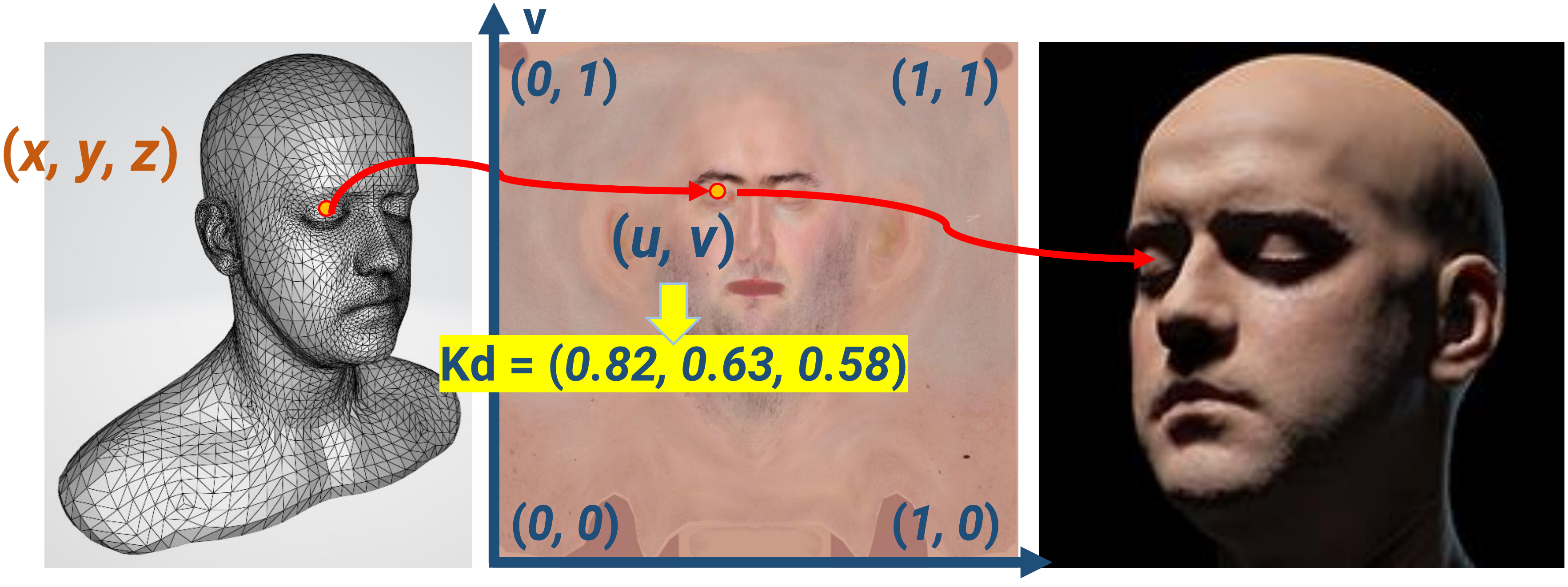
F_1	f -8/-4/-6 -7/-3/-6 -6/-2/-6 f -8/-4/-6 -6/-2/-6 -5/-1/-6
F_2	f -8/-4/-5 -4/-3/-5 -3/-2/-5 f -8/-4/-5 -3/-2/-5 -7/-1/-5
F_3	f -6/-4/-4 -2/-3/-4 -1/-2/-4 f -6/-4/-4 -1/-2/-4 -5/-1/-4
F_4	f -5/-4/-3 -1/-3/-3 -4/-2/-3 f -5/-4/-3 -4/-2/-3 -8/-1/-3
F_5	f -7/-4/-2 -3/-3/-2 -2/-2/-2 f -7/-4/-2 -2/-2/-2 -6/-1/-2
F_6	f -3/-4/-1 -4/-3/-1 -1/-2/-1 f -3/-4/-1 -1/-2/-1 -2/-1/-1
F_7	
F_8	

	vertex1	vertex2	vertex3
f	*P/T/N*	*P/T/N*	*P/T/N*

P: index of vertex position
T: index of texture coordinate
N: index of vertex normal

Texture Coordinate

- A coordinate to look up the texture
  - The way to map a point on the 3D surface to a pixel (texel) on a 2D image texture
- We will introduce texture mapping in the near future

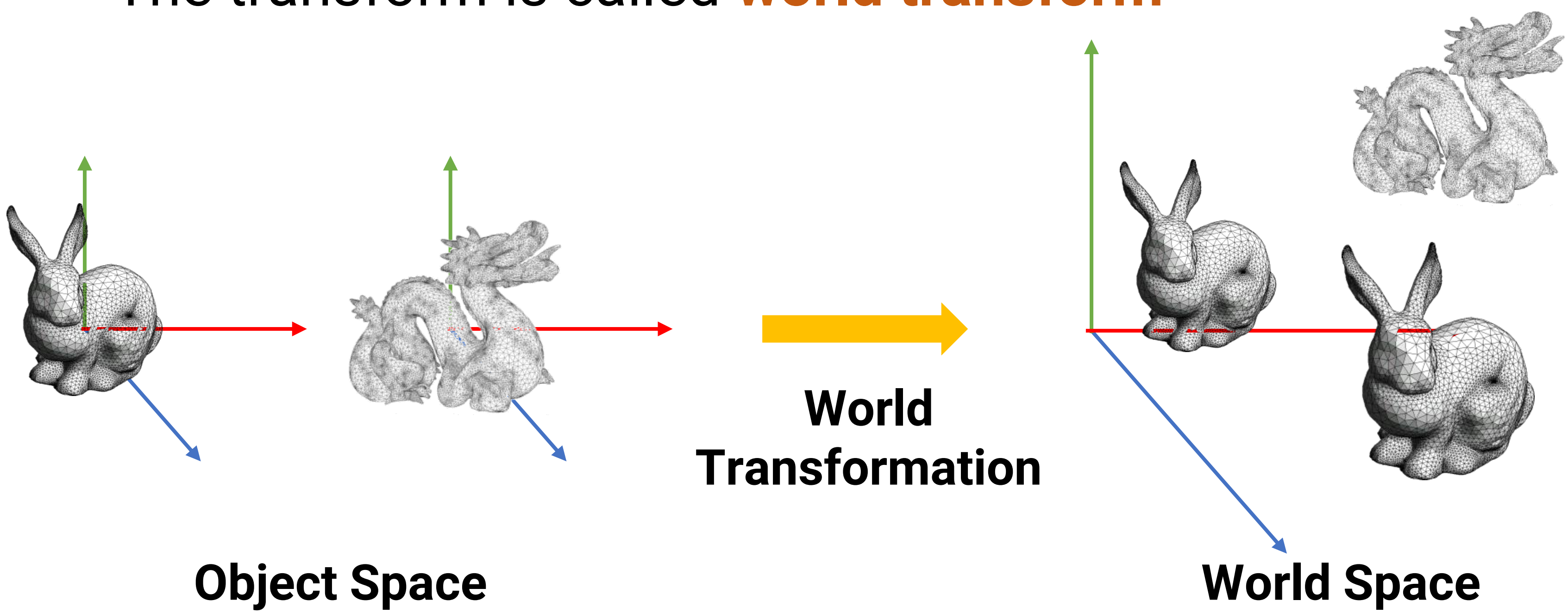


3D triangle mesh 2D albedo texture rendered image

Transformation

World Space and World Coordinate

- Objects are defined in **object space** **individually**
- When building a scene, each object is transformed to a **global** and **unique** space called **world space**
- The transform is called **world transform**



World Space and World Coordinate (cont.)

- Advantages for using “transformation”
  - **Reuse model:** design a model and use it in several scenes
  - **Memory saving:** store a 4x4 matrix instead of duplication of the entire models

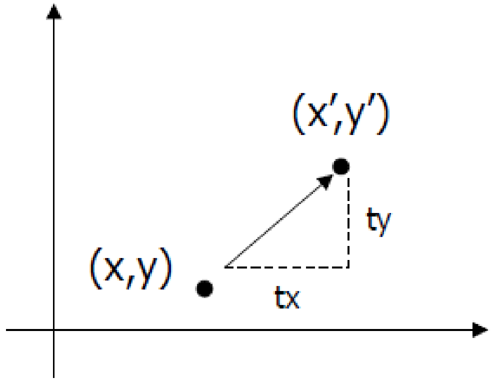
Common Transformations

- Translation
- Scaling
- Rotation

2D Translation

- Given a point ***p(x, y)*** and a translation offset ***T(t_x, t_y)***, the new point ***p'(x', y')*** after translation is ***p' = p + T***

$$x' = x + t_x$$
$$y' = y + t_y$$



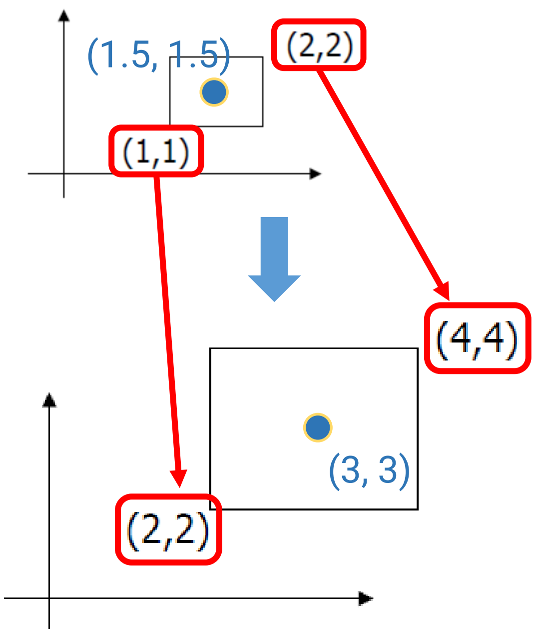
- **Can be represented as** Matrix-vector multiplication

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

2D Scaling

- Given a point ***p(x, y)*** and a scaling factor ***S(s_x, s_y)***, the new point ***p'(x', y')*** after scaling is ***p' = S p***

$$x' = x * s_x$$
$$y' = y * s_y$$

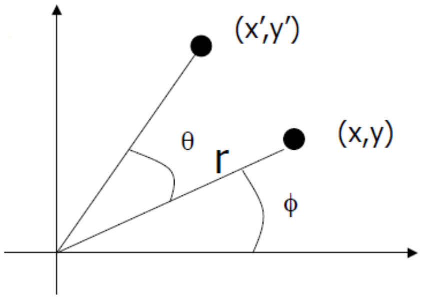


- Matrix-vector multiplication

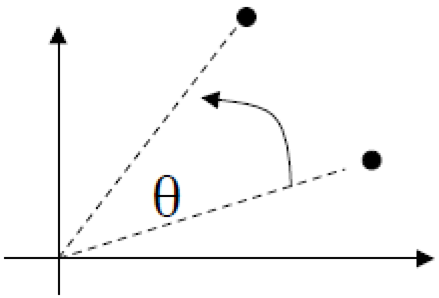
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D Rotation

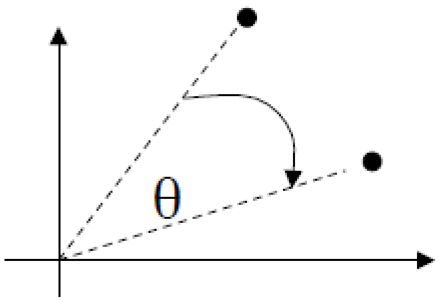
- Given a point ***p(x, y)***, rotate it with respect to the **origin** by ***θ*** and get the new point ***p'(x', y')*** after rotation



- First we define



θ > 0: rotate counterclockwise



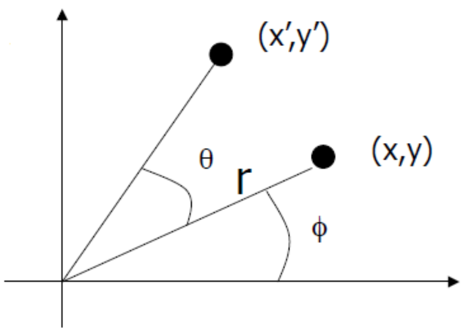
θ < 0: rotate clockwise

2D Rotation (cont.)

- Given a point ***p(x, y)***, rotate it with respect to the **origin** by ***θ*** and get the new point ***p'(x', y')*** after rotation

$$x = r\cos(\phi) \qquad y = r\sin(\phi)$$

$$x' = r\cos(\phi+\theta) \qquad y' = r\sin(\phi+\theta)$$



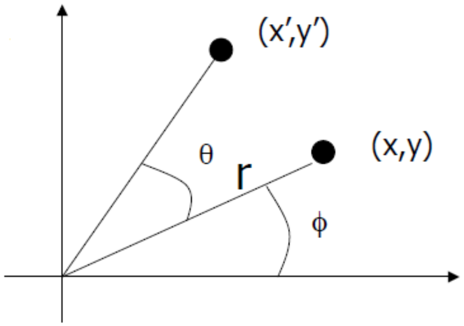
$$\begin{aligned} x' &= r\cos(\phi+\theta) \\ &= r\cos(\phi)\cos(\theta) - r\sin(\phi)\sin(\theta) \\ &= x\cos(\theta) - y\sin(\theta) \end{aligned}$$

$$\begin{aligned} y' &= r\sin(\phi+\theta) \\ &= x\sin(\phi)\cos(\theta) + r\cos(\phi)\sin(\theta) \\ &= y\cos(\theta) + x\sin(\theta) \end{aligned}$$

2D Rotation (cont.)

- Given a point ***p(x, y)***, rotate it with respect to the **origin** by ***θ*** and get the new point ***p'(x', y')*** after rotation

$$
\begin{aligned}
x' &= r\cos(\phi + \theta) \\
&= x\cos(\theta) - y\sin(\theta) \\
y' &= r\sin(\phi + \theta) \\
&= y\cos(\theta) + x\sin(\theta)
\end{aligned}
$$



- Matrix-vector multiplication

$$
\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
$$

2D Translation, Scaling, and Rotation

- Translation

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

- Scaling

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Using a 3x3 matrix allows us to perform all transformations using matrix/vector multiplications
  - We can also **pre-multiply (concatenate)** all the matrices

Homogeneous Coordinate

- We call the *(x, y, 1)* representation the **homogeneous coordinate** for *(x, y)*

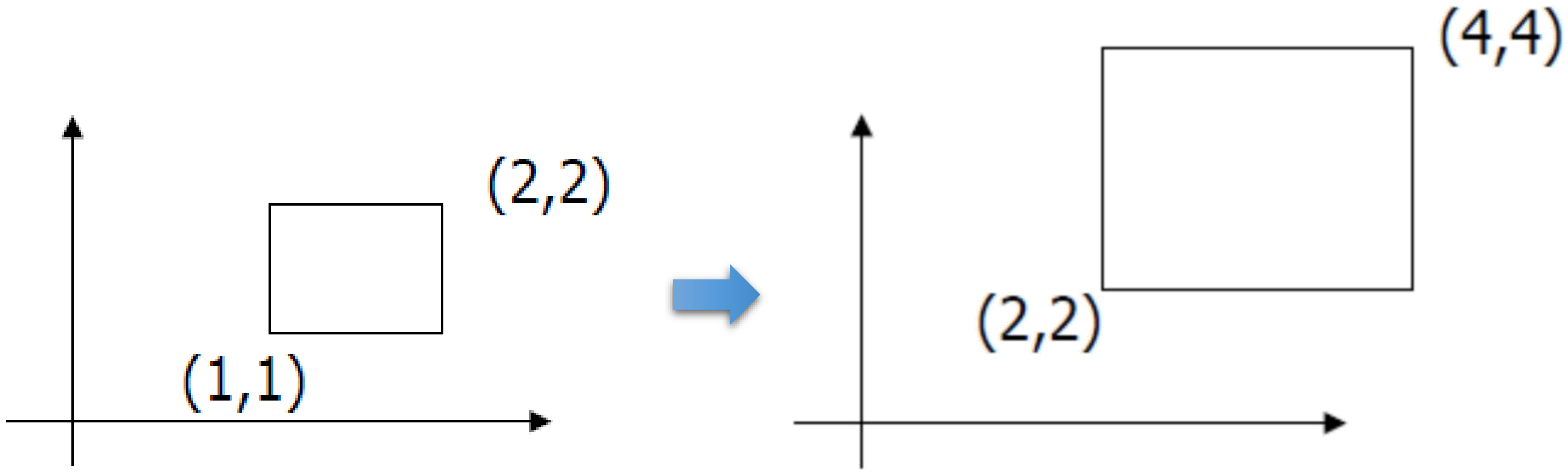
$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- If w is not equal to 1, to make the transformed coordinate also homogeneous, we need to divide the x and y components by w

$$x' = x'/w \qquad y' = y'/w \qquad w = 1$$

Revisit 2D Scaling

- The standard scaling matrix will only anchor at (0, 0)

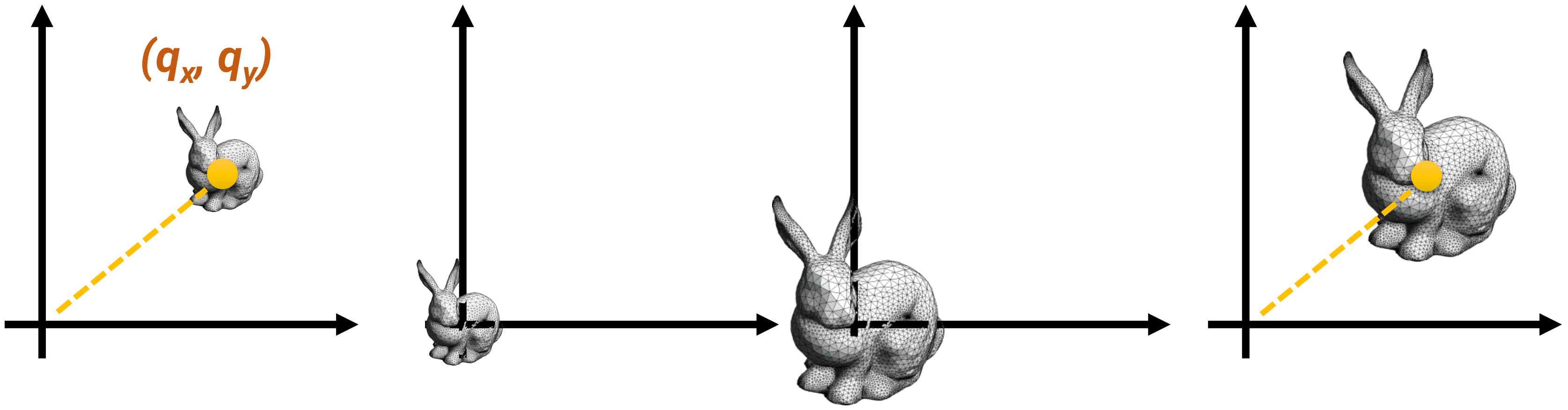


- What if we want the object to be scaled w.r.t its center?

Revisit 2D Scaling (cont.)

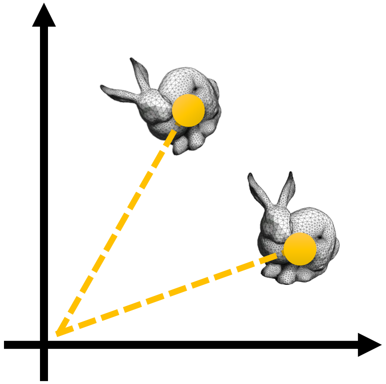
- Scaling about an arbitrary pivot point ***Q(q$_x$, q$_y$)***
  - Translate the objects so that Q will coincide with the origin: ***T(-q$_x$, -q$_y$)***
  - Scale the object: ***S(s$_x$, s$_y$)***
  - Translate the object back: ***T(q$_x$, q$_y$)***
- The final scaling matrix can be written as $\boxed{T(q)S(s)T(-q)}$

Concatenation of matrices



Revisit 2D Rotation

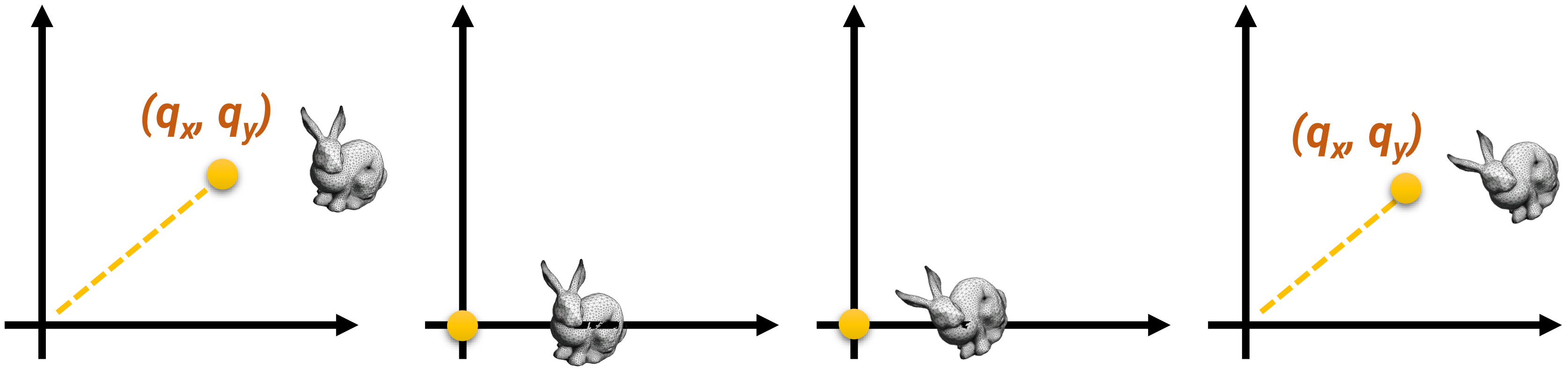
- The standard rotation matrix is used to rotate about the origin (0, 0)



- What if we want the object to be rotated w.r.t a specific pivot?

Revisit 2D Rotation (cont.)

- Rotate about an arbitrary pivot point ***Q(q_x, q_y)*** by ***θ***
  - Translate the objects so that Q will coincide with the origin: ***T(-q_x, -q_y)***
  - Rotate the object: ***R(θ)***
  - Translate the object back: ***T(q_x, q_y)***
- The final rotation matrix can be written as $\boxed{T(q)R(\theta)T(-q)}$



Translation (3D) and Scaling (3D)

- A 3D transformation is represented as a **4x4 matrix**, with **homogeneous coordinate**

translation

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

scaling

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2D 3D

Rotation (3D)

rotation w.r.t x-axis

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \text{-}\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

rotation w.r.t y-axis

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \text{-}\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rotation w.r.t z-axis

$$\begin{bmatrix} \cos\theta & \text{-}\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2D

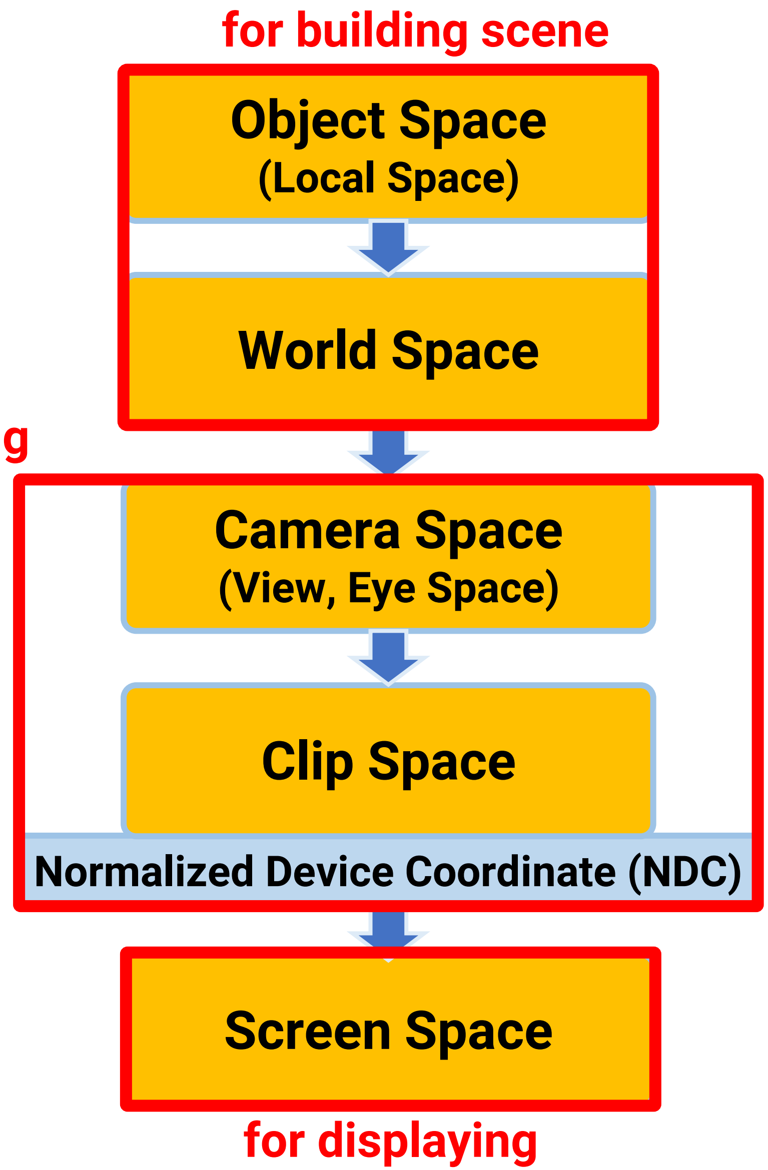
3D

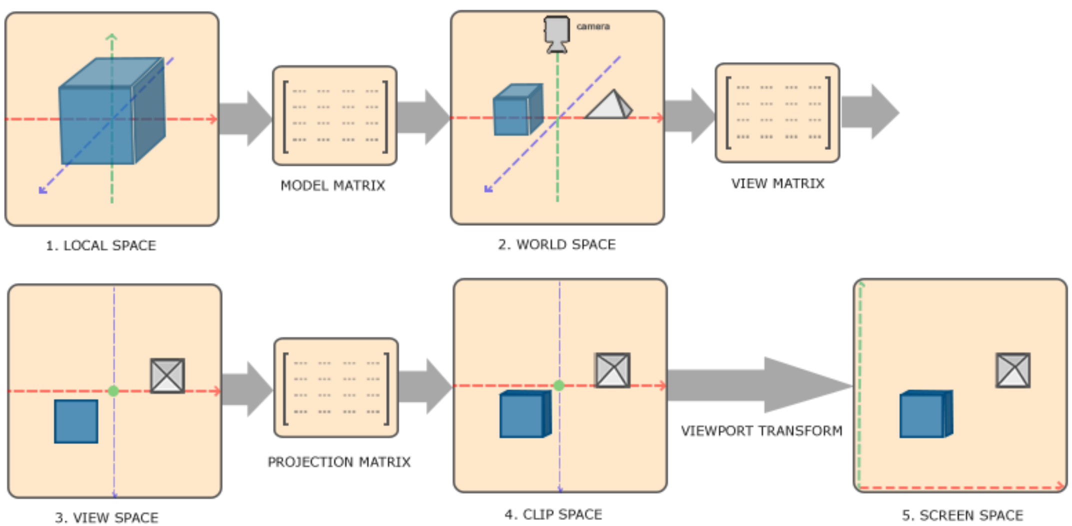
3D Transformation

- Practice
  - Scale w.r.t a given pivot point
  - Rotate w.r.t a given pivot point

Spoiler

- There are other spaces
- We will introduce camera space, clip space, and NDC in the next slides





Any Questions?