Geometry Representation

Computer Graphics

Yu-Ting Wu

Outline

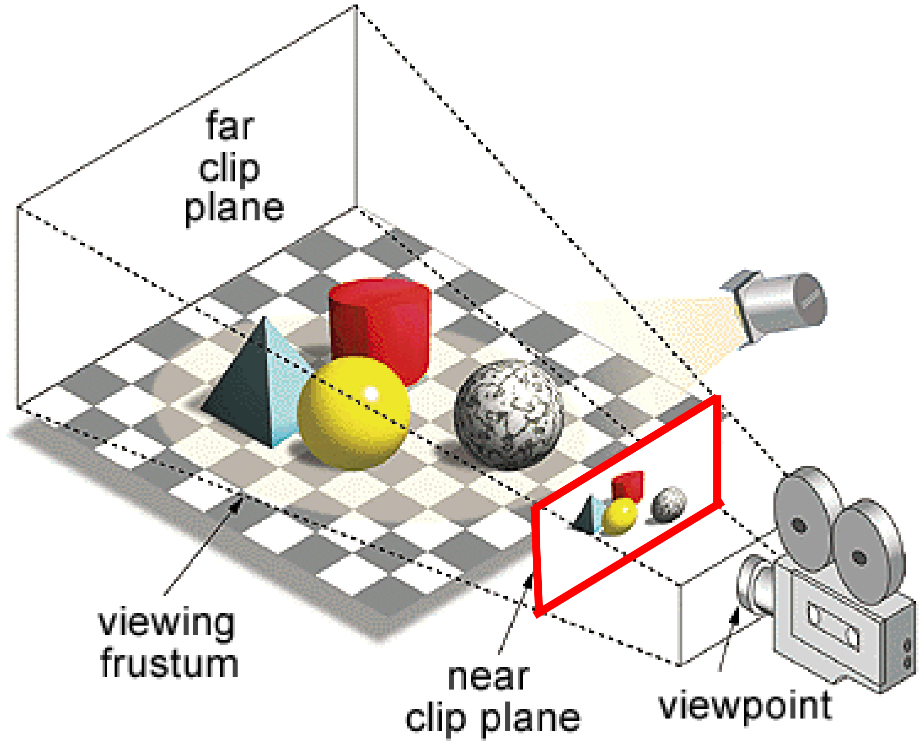
- Geometric properties and coordinate systems
- Draw shapes with OpenGL
- Triangle meshes

Outline

- **Geometric properties and coordinate systems**
- Draw shapes with OpenGL
- Triangle meshes

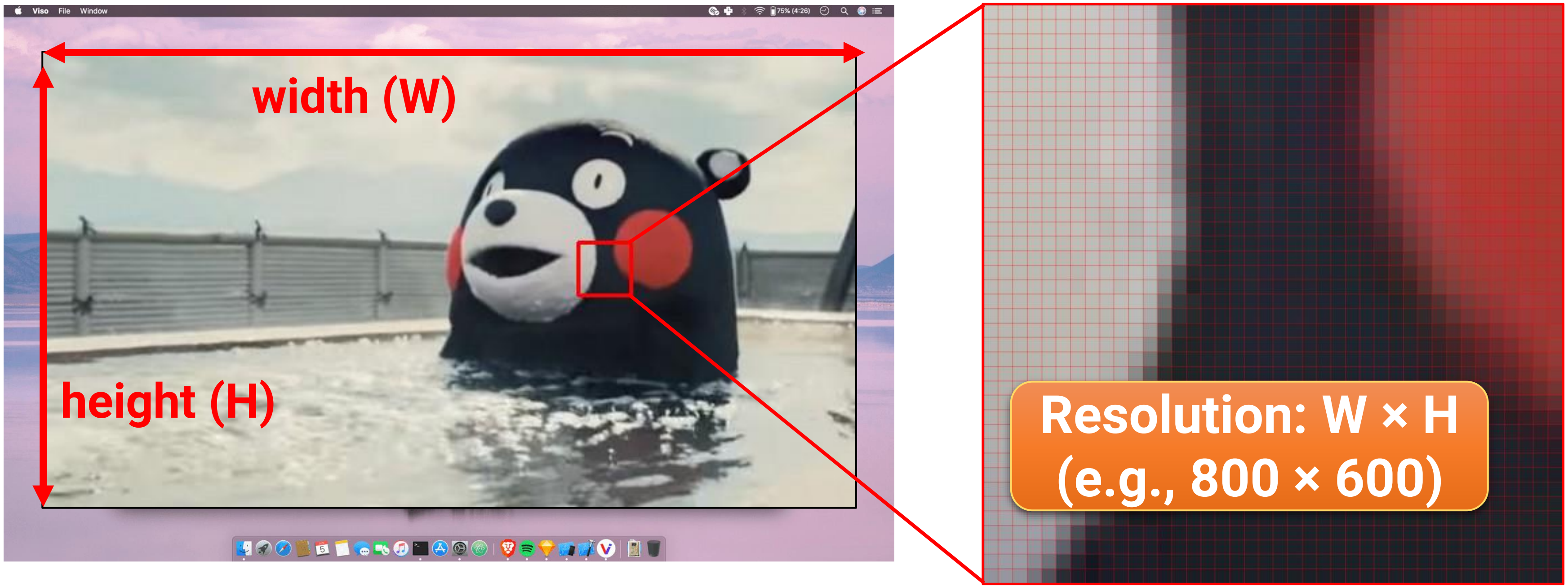
Rendering Process: 3D to 2D

- In computer graphics, we generate an **image** from a **virtual 3D world**
- We are going to introduce the image representation first



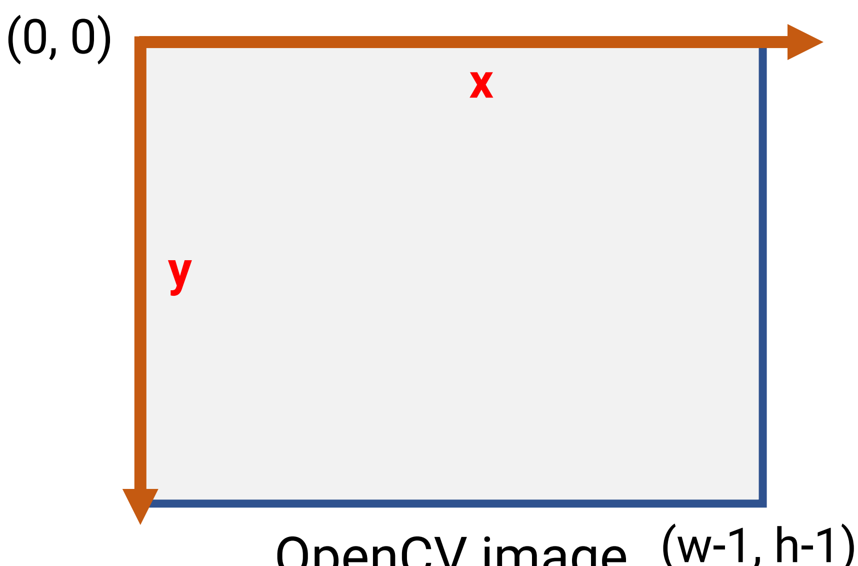
Pixels

- A 2D image (on a screen) is a **rectangular array of pixels** (small, usually square, dots of color)
  - Merge optically when viewed at a suitable distance to produce the impression of continuous tones

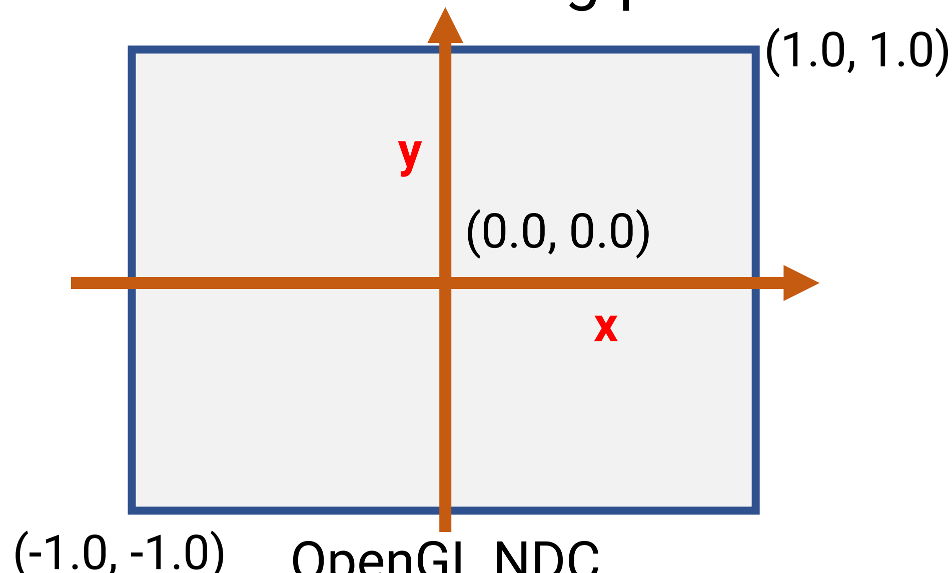


2D Coordinate

- Used to identify the position on a 2D surface (e.g., image)
- The coordinate of a 2D image **depends on libraries**
- For an image (or screen), the coordinate is a pair of positive integers
- For other cases, the coordinates can be floating-point



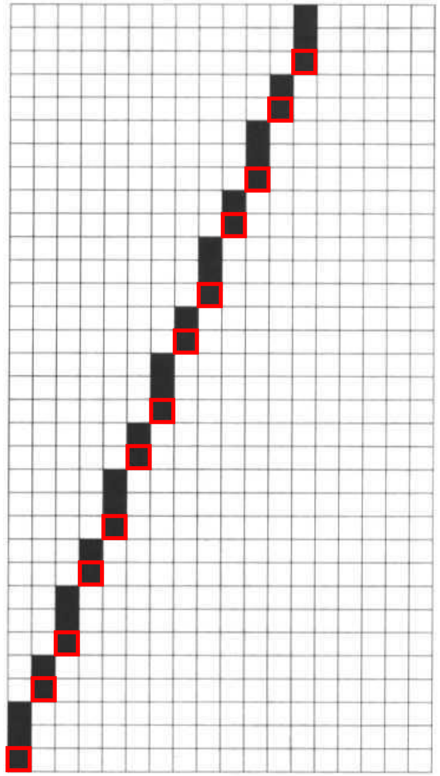
OpenCV image



OpenGL NDC

Rendering of Math (Continuous to Discrete)

- The coordinates of a shape can be floating-point
- However, when the shapes come to the screen and become pixels, they should be **discretized**



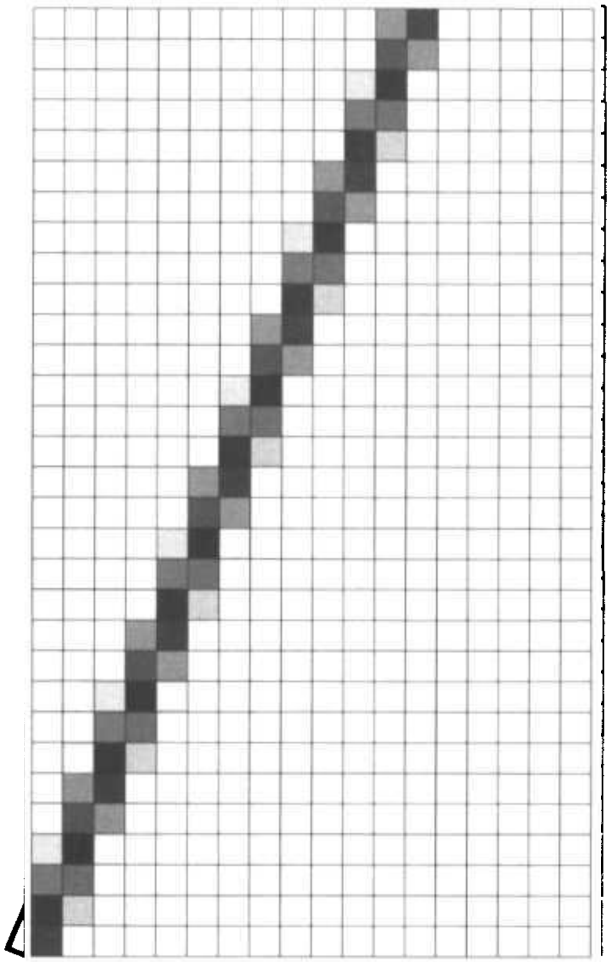
- Example: y = 5x/2 + 1

  pass through (0, 1), (1, 4), (2, 6), (3, 9) …

- Jaggedness is inevitable!
  - Due to the use of a grid of discrete pixels

Anti-aliasing

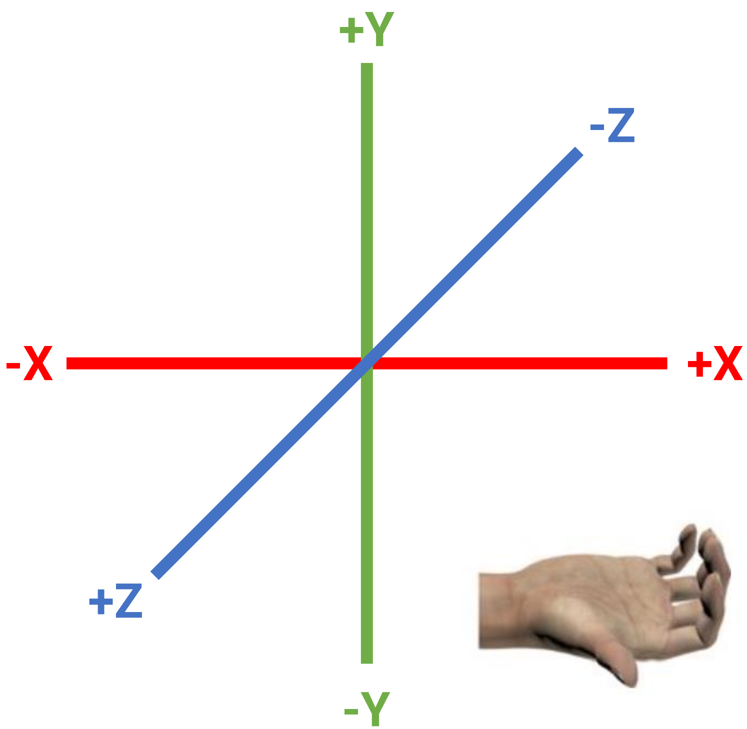
- Anti-aliasing is a **practical** technique to reduce the jaggies
- Use intermediate grey values
  - In the frequency domain, it relates to reducing the frequency of the signal
- Coloring each pixel in a shade of grey whose **brightness is proportional to the area** of the intersection between the pixels and a "**one-pixel-wide**" line



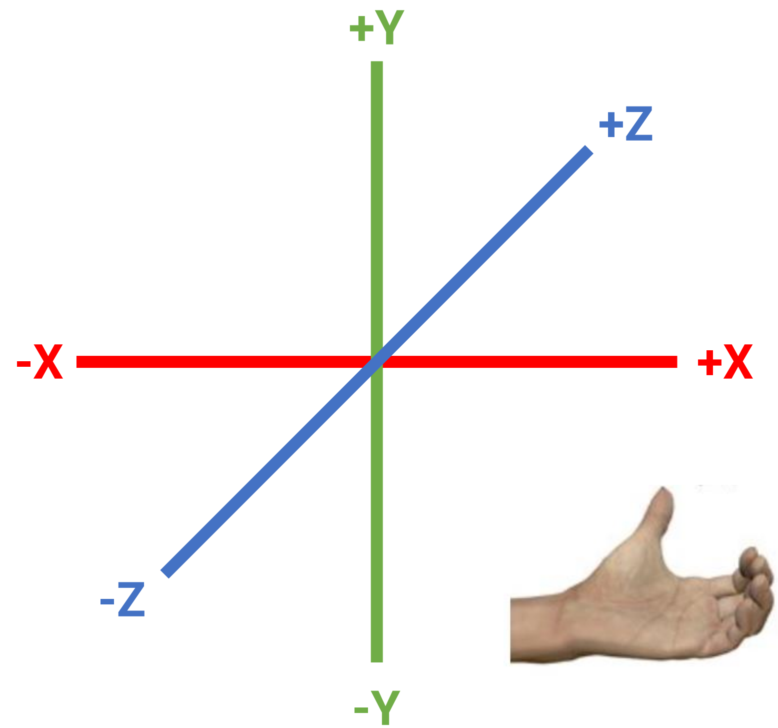
Anti-aliasing (cont.)

Description of the 3D World

- 3D coordinate systems



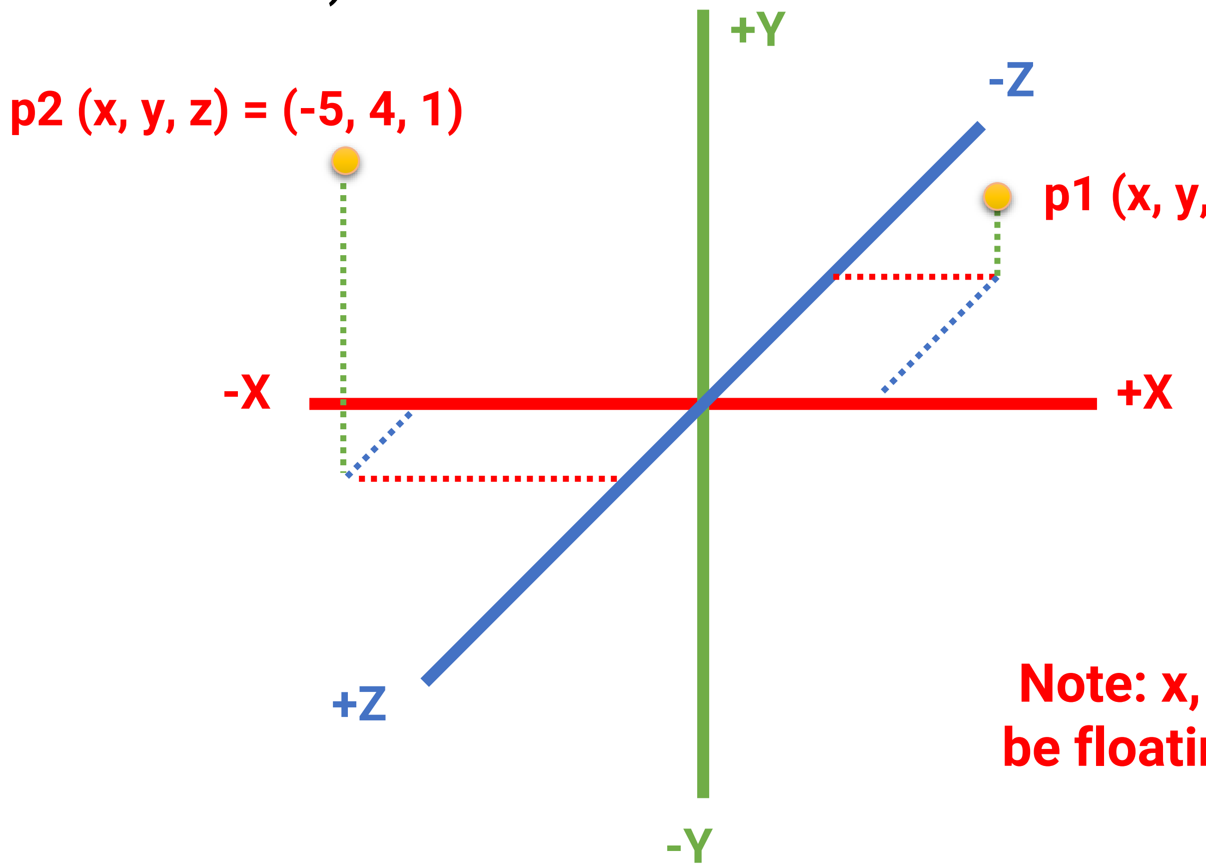
OpenGL
(Right-Hand-Side)



DirectX
(Left-Hand-Side)

Points in 3D Space

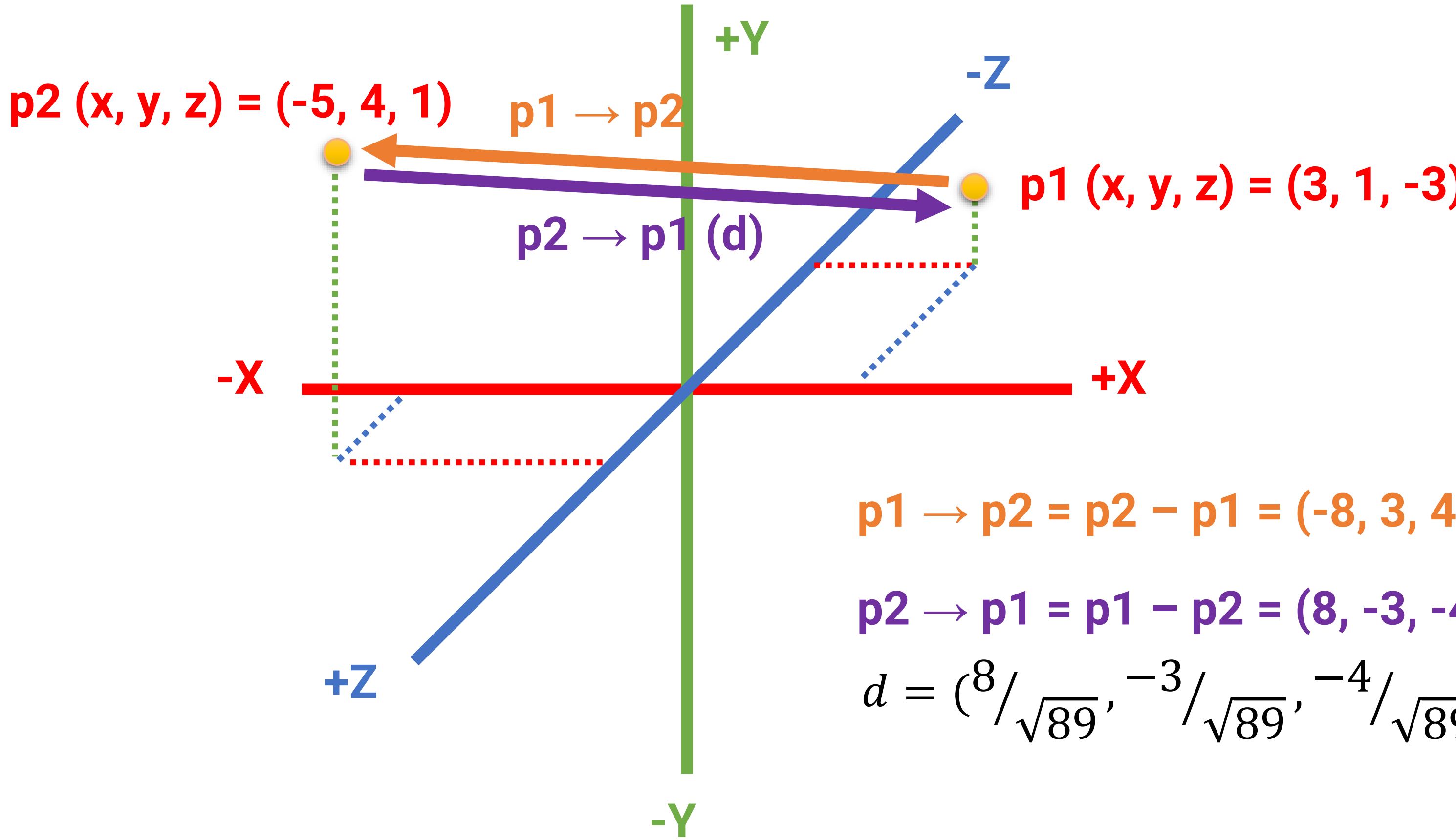
- Described by a 3D coordinate (the components in x, y, and z axises)



Note: x, y, and z can also be floating-point numbers

Vector in 3D Space

- Represent direction (e.g., movement) in the 3D world
- Usually described in a **normalized** version



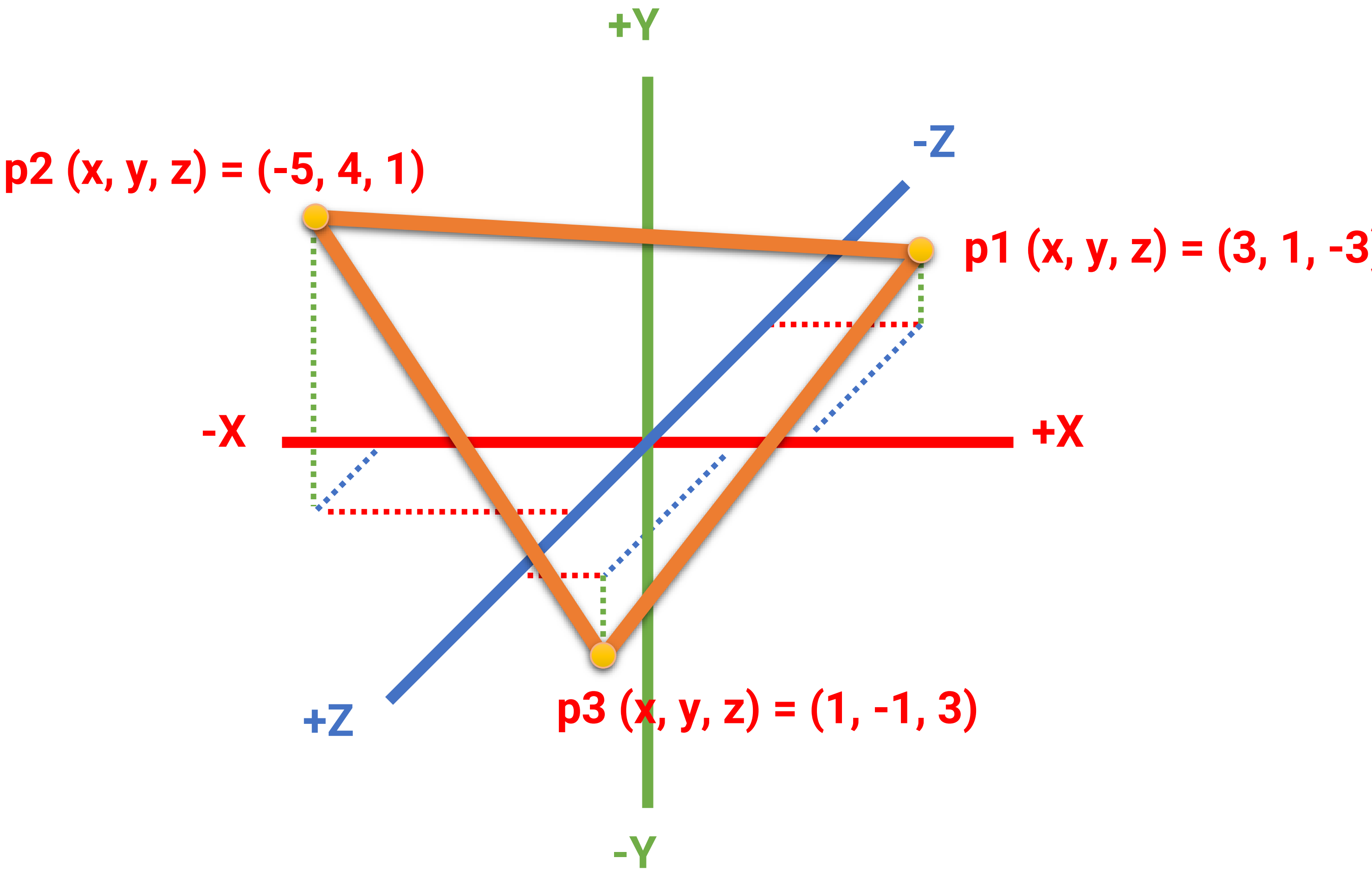
p1 → p2 = p2 − p1 = (-8, 3, 4)

p2 → p1 = p1 − p2 = (8, -3, -4)

$$d = \left({}^{8}/_{\sqrt{89}}, {}^{-3}/_{\sqrt{89}}, {}^{-4}/_{\sqrt{89}}\right)$$

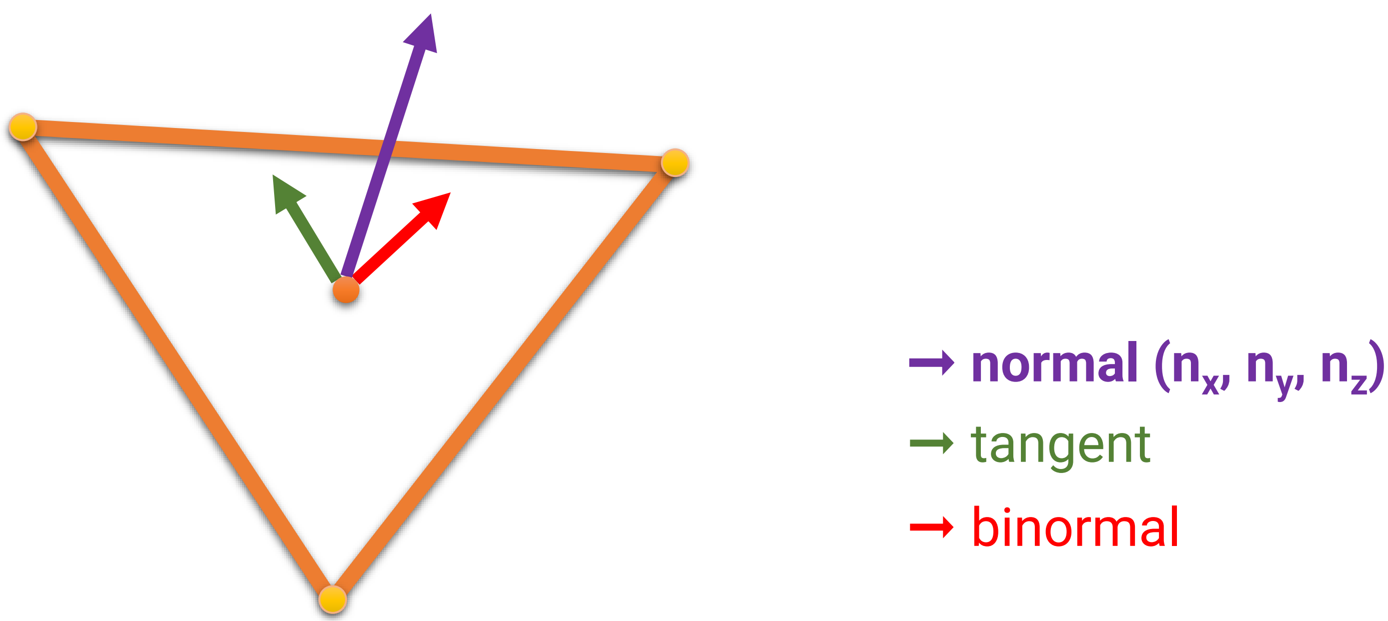
Triangles in 3D

- Composed of three points as endpoints (called **vertices**)



3D 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**



Outline

- Geometric properties and coordinate systems
- **Draw shapes with OpenGL**
- Triangle meshes

Library for Supporting Drawing

- **GLEW: The OpenGL Extension Wrangler Library (link)**
  - A cross-platform open-source C/C++ extension loading library
  - Provide efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform
- **GLM: OpenGL Mathematics (link)**
  - A **header-only** C++ mathematics library for graphics software based on the **OpenGL Shading Language (GLSL) specifications**

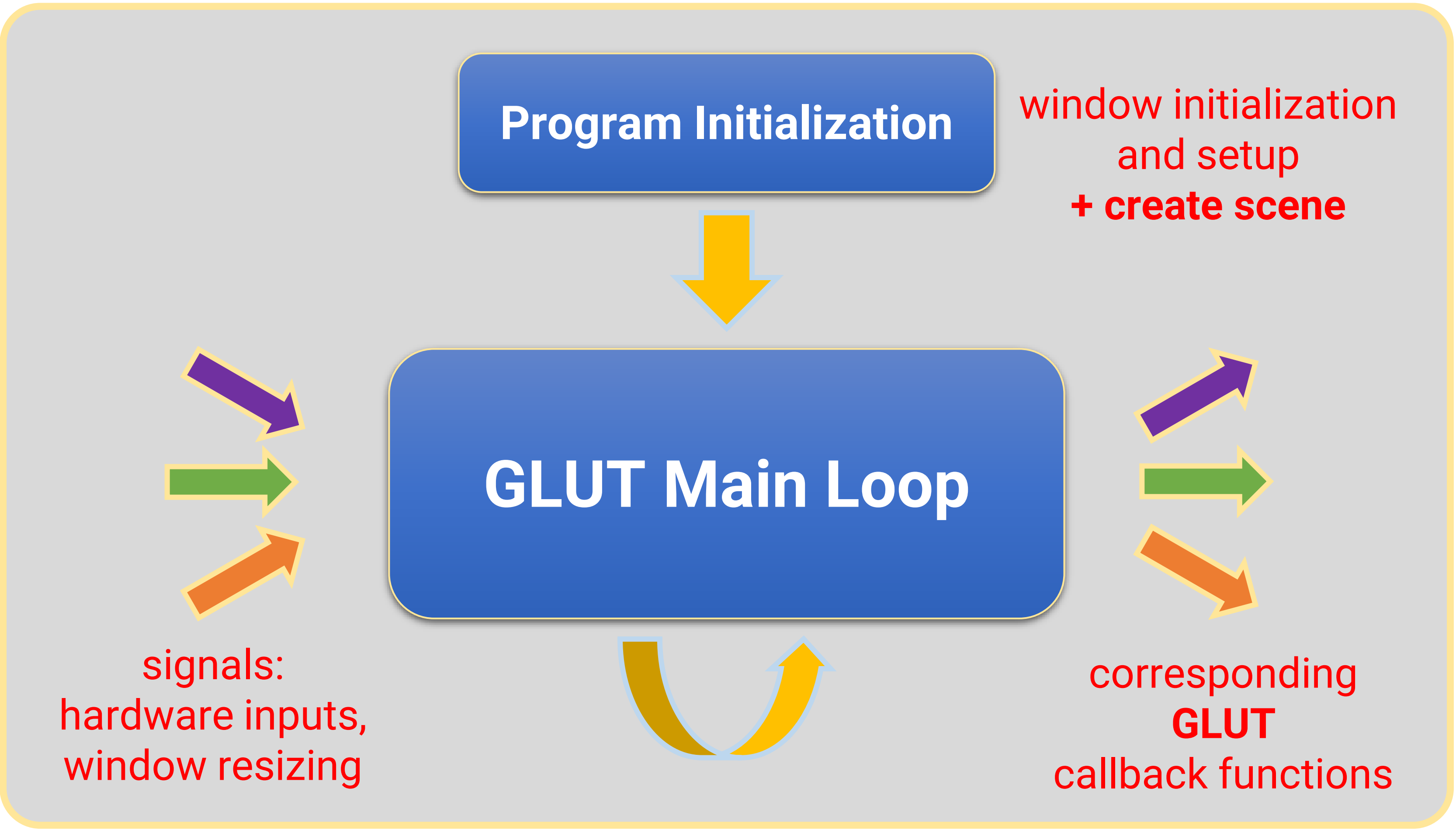
Put the library (*.h, *.lib, *.dll) in the project like what we do for FreeGLUT

Enable GLEW and Add Init. Functions

```cpp
int main(int argc, char** argv)
{
    // Setting window properties.
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(640, 360);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("OpenGL Renderer");

    // Initialize GLEW.
    // Must be done after glut is initialized!
    GLenum res = glewInit();
    if (res != GLEW_OK) {
        std::cerr << "GLEW initialization error: "
                  << glewGetErrorString(res) << std::endl;
        return 1;
    }

    // Initialization.
    SetupRenderState();
    SetupScene();

    // Register callback functions.
    glutDisplayFunc(RenderSceneCB);
```

```cpp
// OpenGL and FreeGlut headers.
#include <glew.h>
#include <freeglut.h>
```

Recap: Life Cycle of a GLUT Program



your program

Draw a Single Point

Continue

```
// Global variables.
GLuint vbo;
```

vertex buffer object

```
void SetupScene()
{
    // Draw a single point.
    float VertexPosition[3] = {0.0f, 0.0f, 0.0f};
    // Generate the vertex buffer.
    glGenBuffers(1, &vbo);
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    glBufferData(GL_ARRAY_BUFFER, sizeof(VertexPosition), VertexPosition, GL_STATIC_DRAW);
}
```

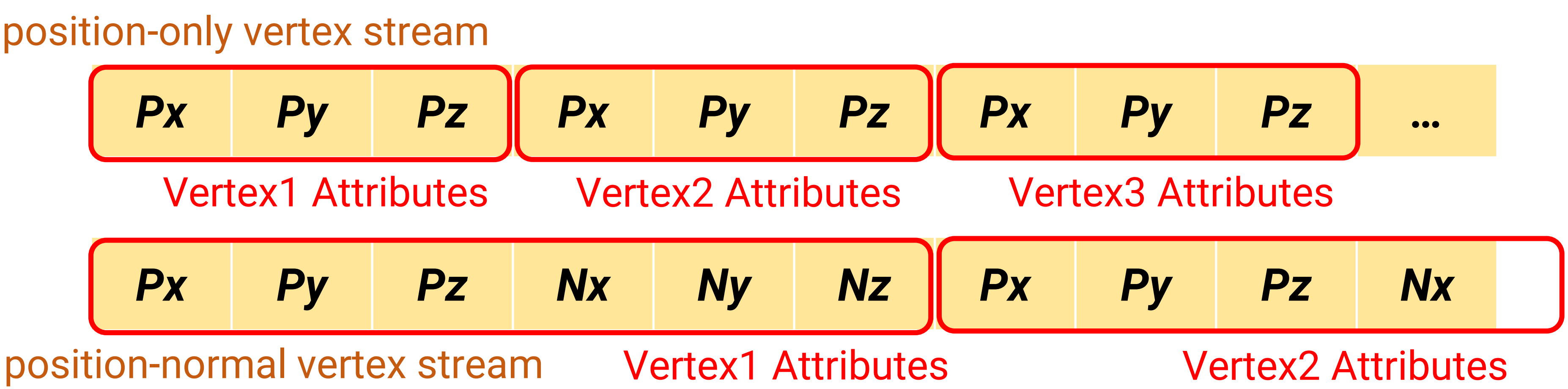
create a vertex buffer and upload vertex data (initialization)

```
void RenderSceneCB()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Render a point on screen.
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(float)*3, 0);
    glDrawArrays(GL_POINTS, 0, 1); // # vertices = 1.
    glDisableVertexAttribArray(0);

    glutSwapBuffers();
}
```

render shapes with the vertex buffer

Vertex Buffer

- A buffer storing the **vertex attribute data**
- Possible vertex attributes include (but are not limited to)
  - **Vertex position**
  - Vertex normal (optional)
  - Texture coordinate (optional)
- Will be passed to GPU for rendering



Vertex Buffer (cont.)

- **Generate a buffer**
  - void **glGenBuffers**(GLsizei n, GLuint * buffers);
- **Upload data into the buffer**
  - void **glBindBuffer**(GLenum target , GLuint buffer); [Link]
  - void **glBufferData**([Link]

    GLenum target ,

    GLsizeiptr size ,

    const void * data ,

    GLenum usage);

```
float VertexPosition[3] = {0.0f, 0.0f, 0.0f};
// Generate the vertex buffer.
glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(VertexPosition), VertexPosition, GL_STATIC_DRAW);
```

Vertex Buffer (cont.)

- **Render with the vertex buffer**
  - void **glEnableVertexAttribArray**(GLuint index);
  - void **glVertexAttribPointer**(

```
            GLuint index ,
            GLint size ,
            GLenum type ,
            GLboolean normalized ,
            GLsizei stride ,
            const void *pointer
    );
```

index: The index of the attribute
E.g., 0 for position, 1 for normal, etc.

size: Number of components of the attribute

type: Type of the attribute component

stride: The byte offset to the same attribute of the next vertex

pointer: The byte offset of the first component

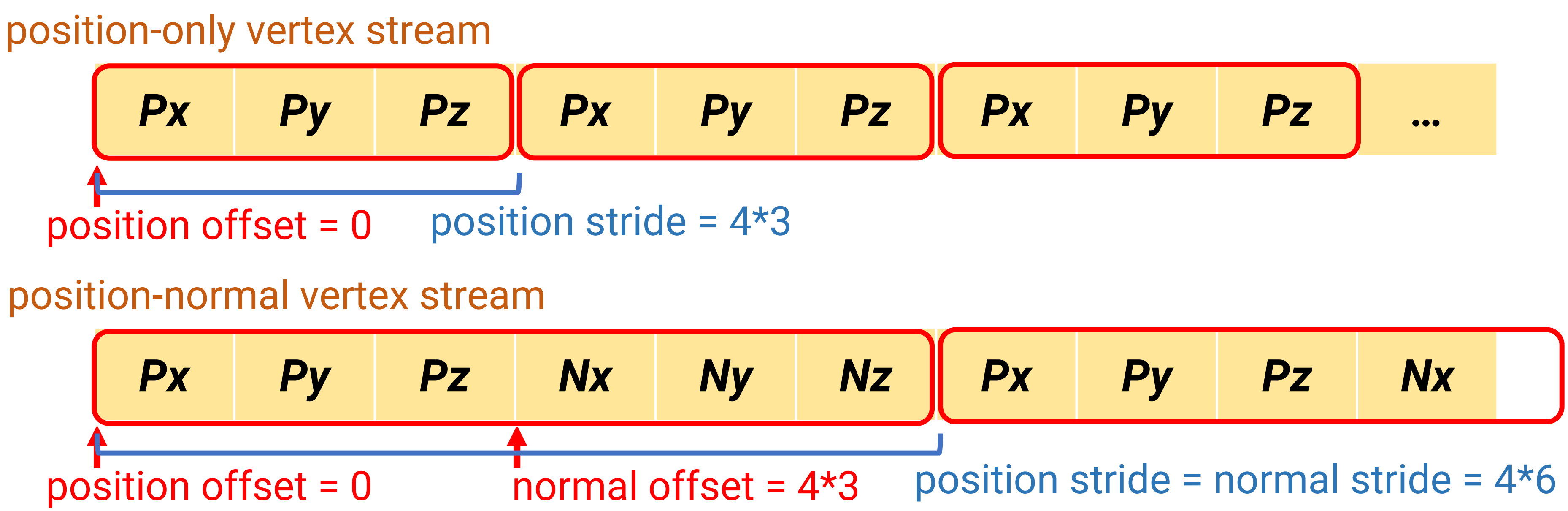
```
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(float)*3, 0);
```

Vertex Buffer (cont.)

- void **glVertexAttribPointer**(

        …
        GLsizei stride , The byte offset to the same attribute of the next vertex
        const void * pointer   The byte offset of the first component

  );



Vertex Buffer (cont.)

- void **glDrawArrays**(
  GLenum mode ,
  GLint first ,
  GLsizei count
  );

  - mode: The type of the primitive. E.g., GL_POINTS, GL_LINE_LOOP, GL_TRIANGLES, etc.
  - first: The start index
  - count: The number of **indices** to be rendered

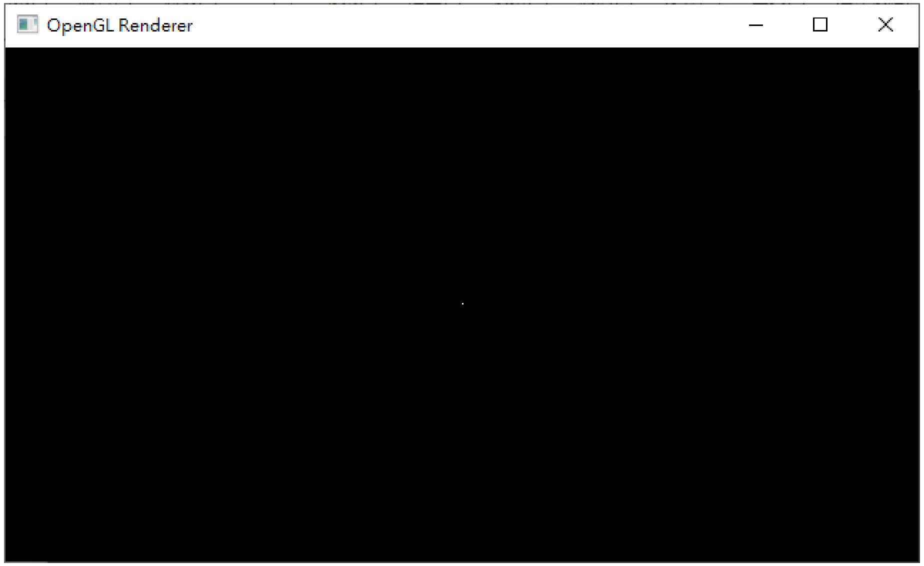
- void **glDisableVertexAttribArray**(GLuint index);

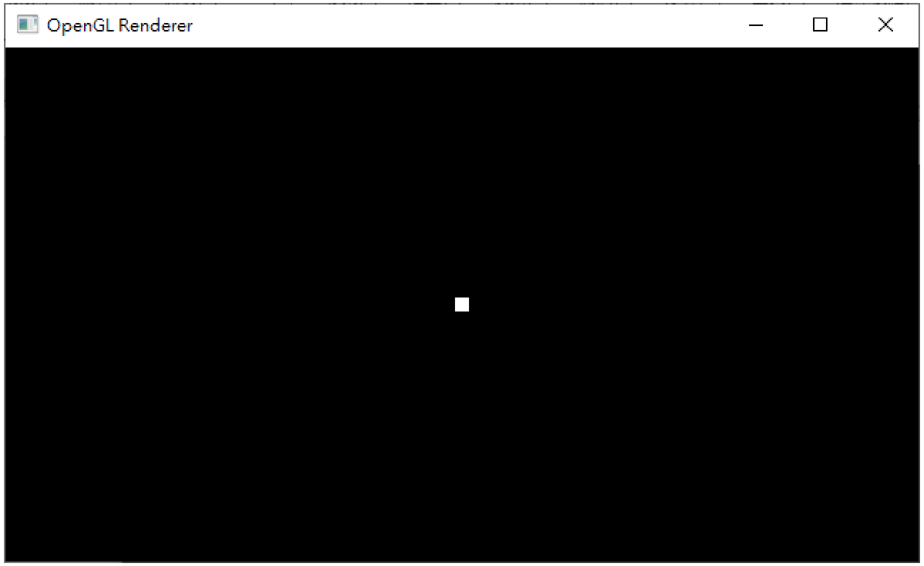
```
// Render a point on screen.
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(float)*3, 0);
glDrawArrays(GL_POINTS, 0, 1); // # vertices = 1.
glDisableVertexAttribArray(0);
```

Continue

Change the Point Size

- void **glPointSize**(GLfloat size)

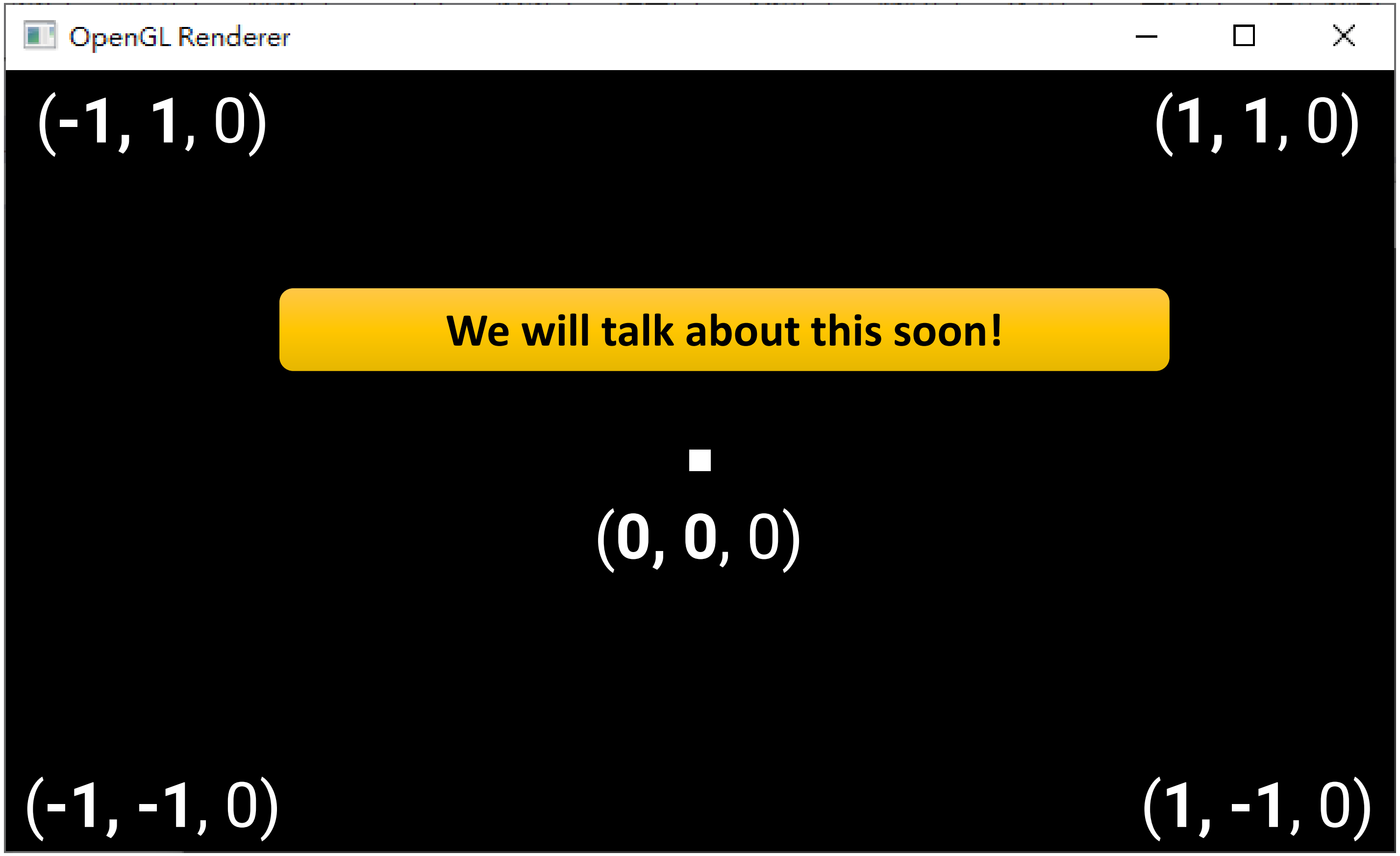





```
void SetupRenderState()
{
    // Default.
    glPointSize(1);
}
```

```
void SetupRenderState()
{
    glPointSize(10);
}
```

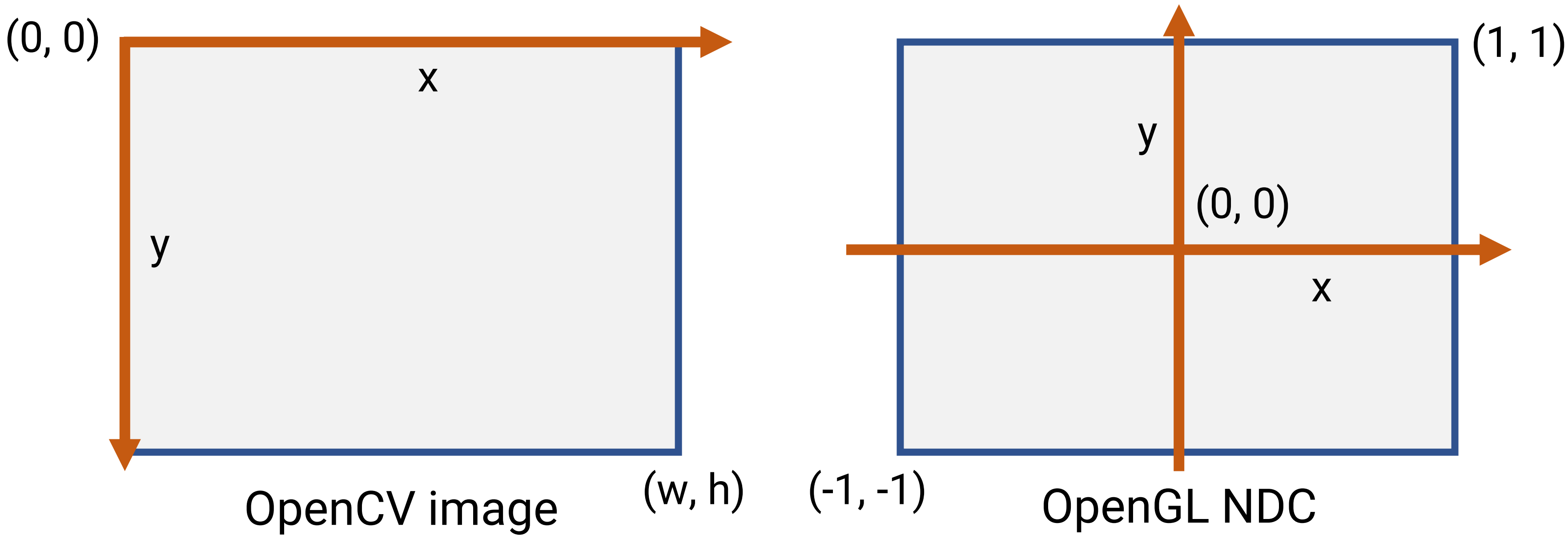
Insight: Coordinate (Recall)



What about the *z* coordinate? You can find the point will only be visible if its *z* value is within **[-1, 1]**

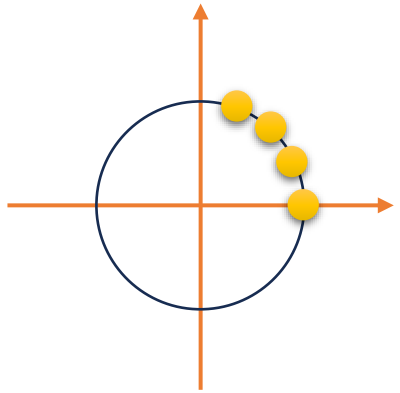
Recall: Image Coordinate

- The coordinate of a 2D image depends on libraries



Draw a Circle (Ellipse)

```
// C++ STL headers.
#include <iostream>
#include <vector>
#define _USE_MATH_DEFINES
#include <math.h>

// Global variables.
GLuint vbo;
const int numCircleSamples = 36;

void SetupScene()
{
    // Draw a circle.
    float VertexPosition[numCircleSamples * 3];
    const float thetaOffset = 2.0f * M_PI / (float)numCircleSamples;
    float startTheta = 0.0f;
    float r = 0.5f;
    for (int i = 0; i < numCircleSamples; ++i) {
        float theta = startTheta + i * thetaOffset;
        VertexPosition[3 * i + 0] = r * std::cos(theta);    // x.
        VertexPosition[3 * i + 1] = r * std::sin(theta);    // y.
        VertexPosition[3 * i + 2] = 0.0f;                   // z.
    }

    // Generate the vertex buffer.
    glGenBuffers(1, &vbo);
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    glBufferData(GL_ARRAY_BUFFER, sizeof(VertexPosition), VertexPosition, GL_STATIC_DRAW);
}
```

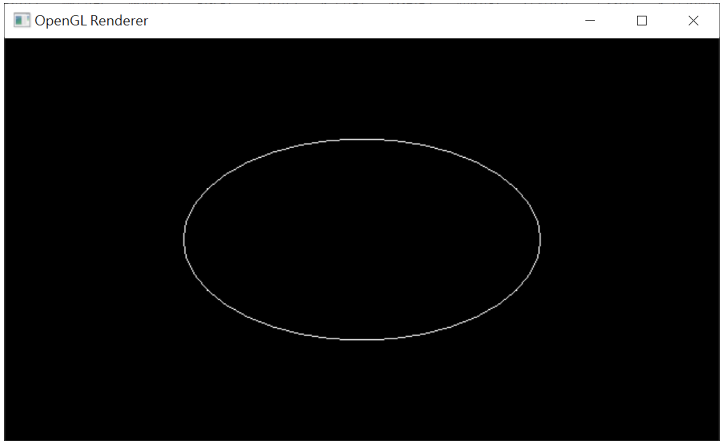


Draw a Circle (Ellipse)

```
void RenderSceneCB()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Render a point on screen.
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(float)*3, 0);
    glDrawArrays(GL_LINE_LOOP, 0, numCircleSamples);
    glDisableVertexAttribArray(0);

    glutSwapBuffers();
}
```

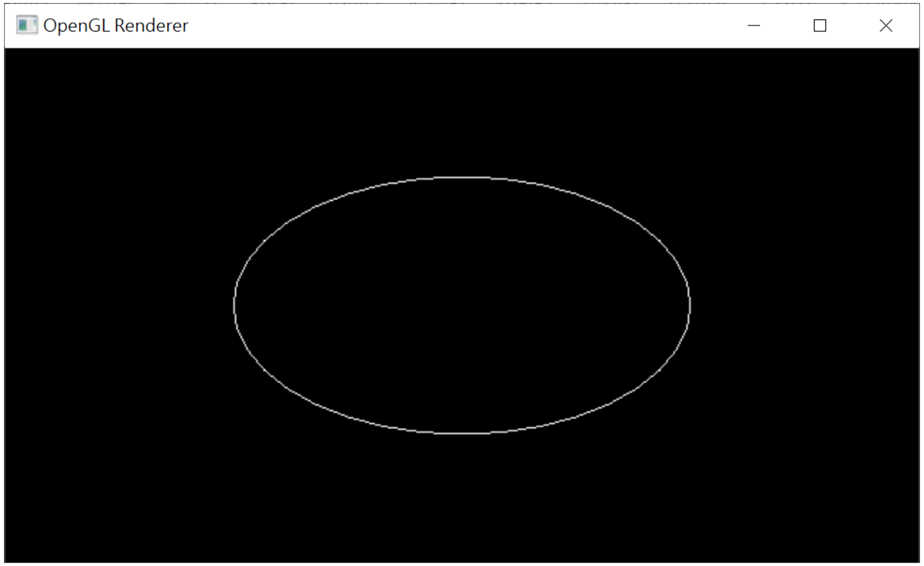
GL_LINES

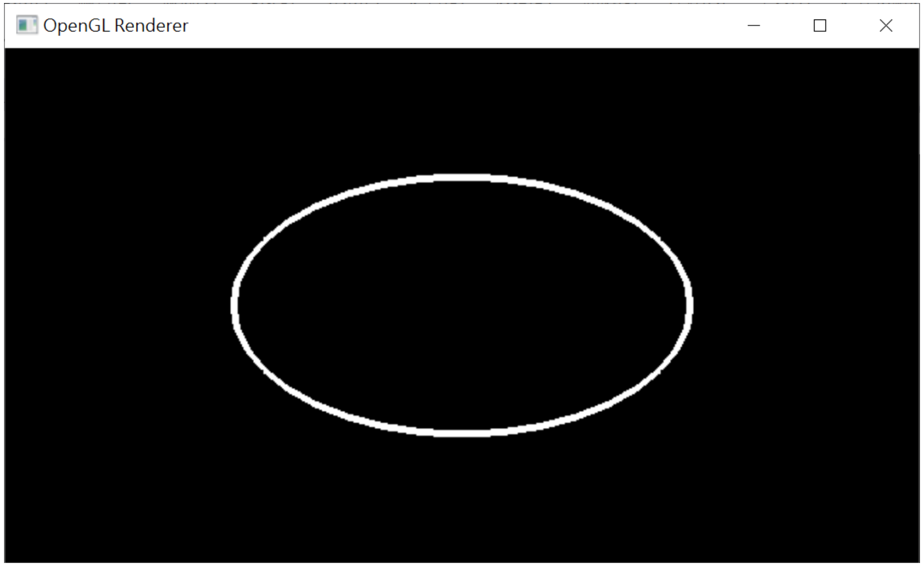
GL_LINE_LOOP



Change the Line Width

- *void **glLineWidth**(GLfloat width)*






```
void SetupRenderState()
{
    glLineWidth(5);
}
```

The GLM Library

- In computer graphics, we need a data structure to store and manipulate **multi-dimensional data**, such as position, normal, texture coordinate, and color
- The GLM library provides an elegant way to process multi-dimensional data
  - Support **operator overloading**
  - Match the syntax of OpenGL shading language (GLSL)
  - Support **alias** of components
    - For position or normal, we used to use *(x, y, z, w)*
    - For texture coordinate, we used to use *(u, v, s, t)*
    - For color, we used to use *(r, g, b, a)*

The GLM Library Examples

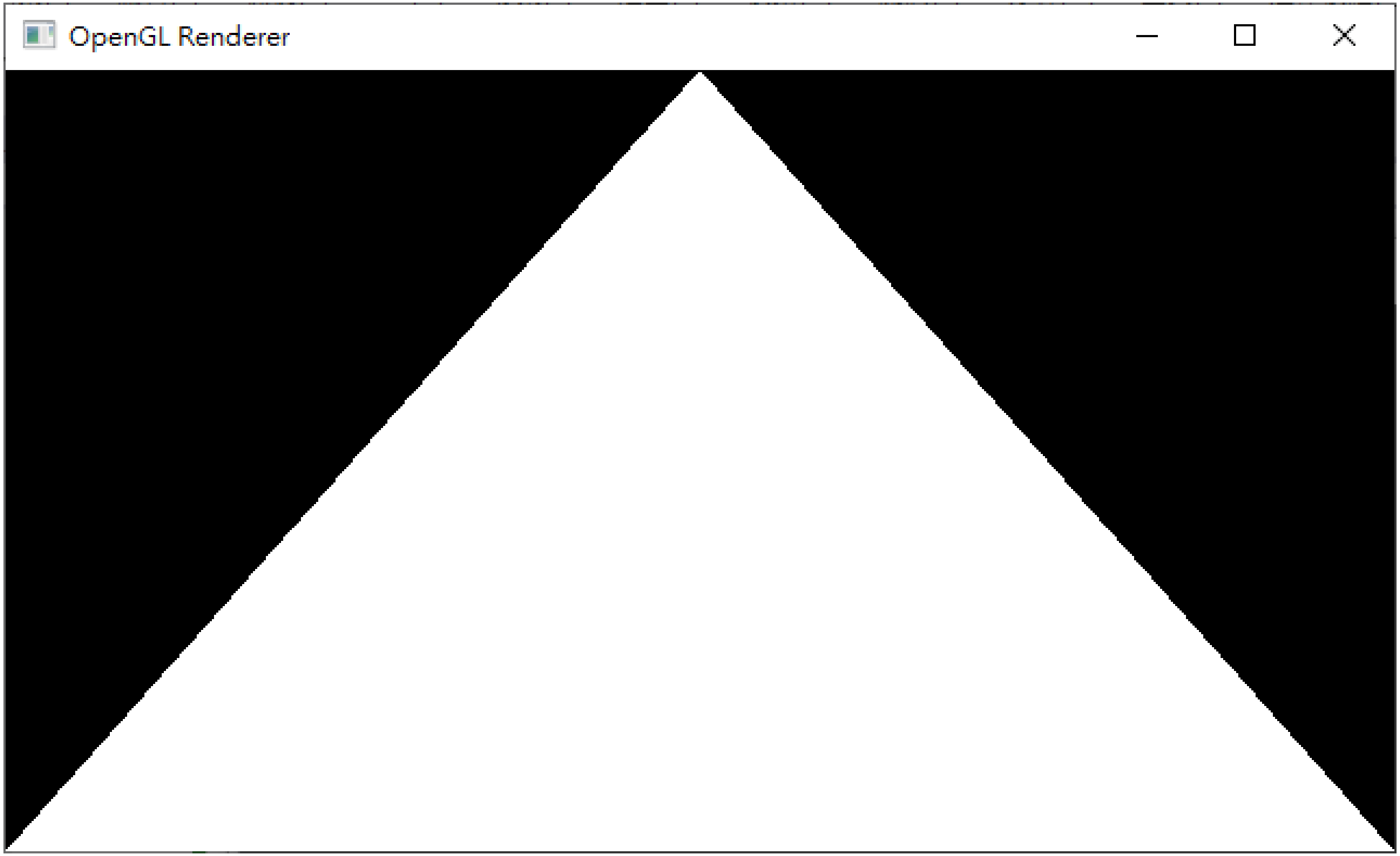
- The most common data types are three/four-dimensional vectors and four-by-four matrices
- Example: compute the average direction of three vectors

```
glm::vec3 dir1 = glm::vec3(1.0f, 0.0f, 0.0f);
glm::vec3 dir2 = glm::vec3(0.0f, 1.0f, 0.0f);
glm::vec3 dir3 = glm::vec3(0.0f, 0.0f, 1.0f);
glm::vec3 avgDir = (dir1 + dir2 + dir3) / 3.0f;
```

Draw a Triangle

```
void SetupScene()
{
    // Draw a triangle.
    glm::vec3 VertexPosition[3];
    VertexPosition[0] = glm::vec3(-1.0f, -1.0f, 0.0f);
    VertexPosition[1] = glm::vec3( 0.0f,  1.0f, 0.0f);
    VertexPosition[2] = glm::vec3( 1.0f, -1.0f, 0.0f);

    // Generate the vertex buffer.
    glGenBuffers(1, &vbo);
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    glBufferData(GL_ARRAY_BUFFER, sizeof(VertexPosition), VertexPosition, GL_STATIC_DRAW);
}
void RenderSceneCB()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Render a point on screen.
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(glm::vec3), 0);
    glDrawArrays(GL_TRIANGLES, 0, 3);
    glDisableVertexAttribArray(0);

    glutSwapBuffers();
}
```

Draw a Triangle (cont.)



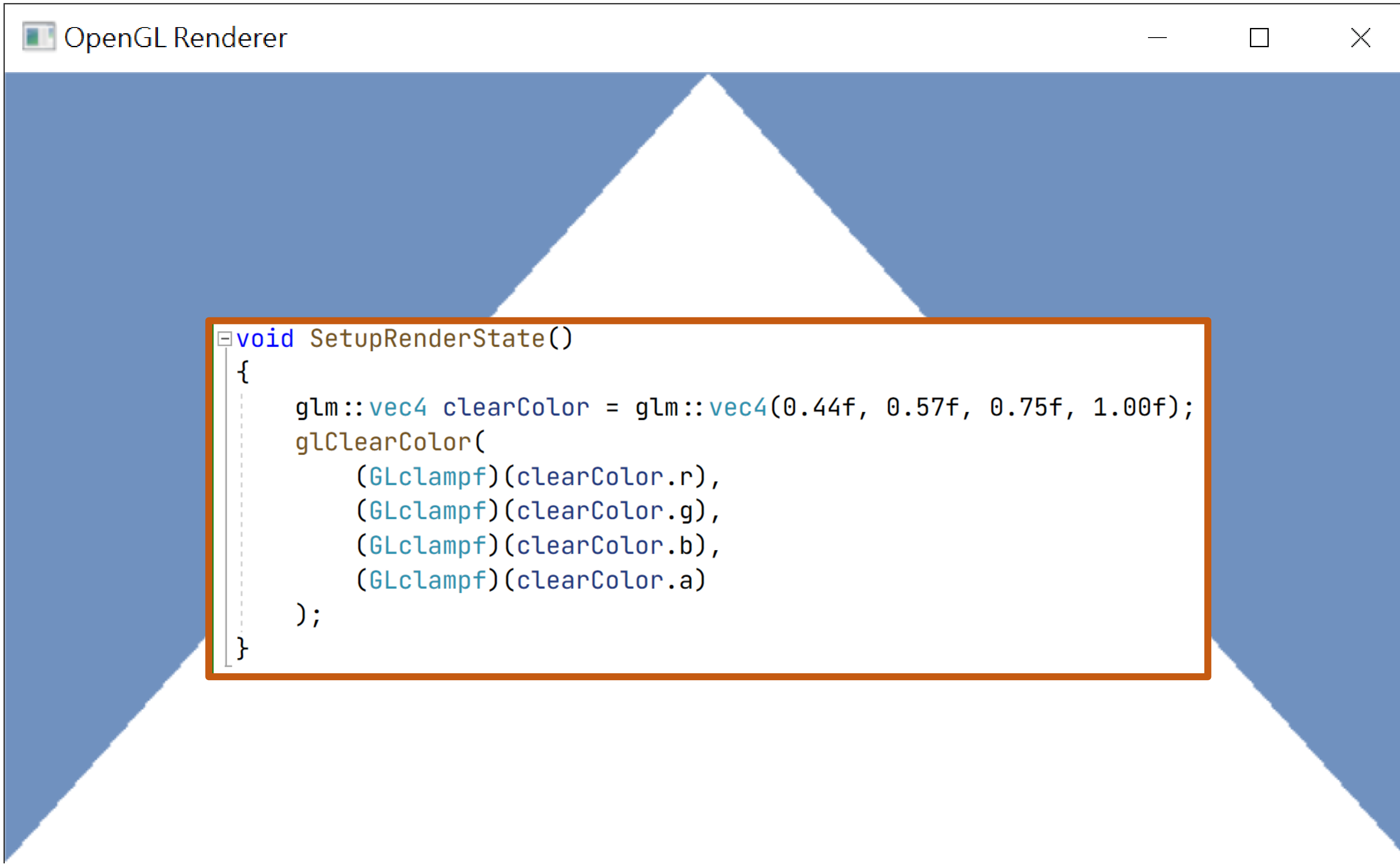
Avoid Deprecated APIs

- You may find online tutorials that use the following APIs:

```
glBegin(GL_POINTS/GL_LINES/GL_TRIANGLES);
        glVertex3f(…);
        glVertex3f(…);
        glVertex3f(…);
glEnd();
```

- Although it seems convenient, do **NOT** use it
- These APIs have been deprecated since OpenGL 3.2 due to the performance issue

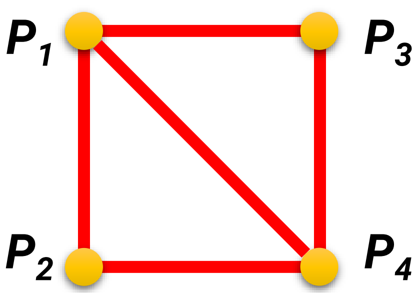
GLM Vector for Representing Color

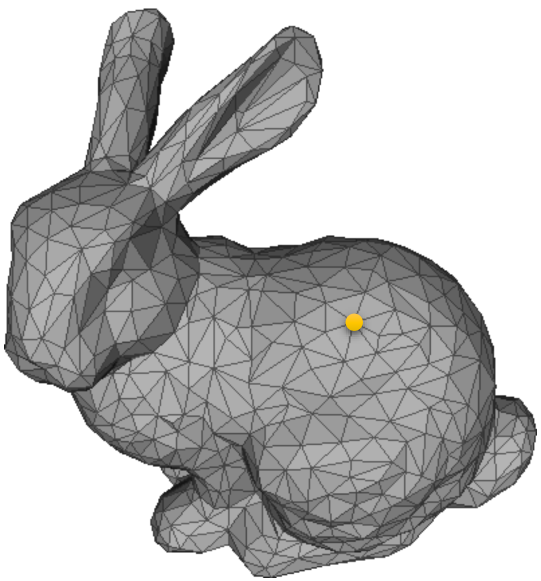



```cpp
void SetupRenderState()
{
    glm::vec4 clearColor = glm::vec4(0.44f, 0.57f, 0.75f, 1.00f);
    glClearColor(
        (GLclampf)(clearColor.r),
        (GLclampf)(clearColor.g),
        (GLclampf)(clearColor.b),
        (GLclampf)(clearColor.a)
    );
}
```

Index Buffer

- Lots of the vertices are shared when drawing triangle mesh with multiple triangles
- E.g., a quad with 2 triangles





Vertex Buffer

P_1 x	P_1 y	P_1 z	P_2 x	P_2 y	P_2 z	P_4 x	P_4 y	P_4 z	P_1 x	P_1 y	P_1 z	P_4 x	P_4 y	P_4 z	P_3 x	P_3 y	P_3 z

Using glDrawArrays will need 6 vertices in the vertex buffer

Index Buffer (cont.)

- Lots of the vertices are shared when drawing triangle mesh with multiple triangles
- **We can use an index buffer to identify the vertex defined in the vertex buffer**
- E.g., a quad with 2 triangles

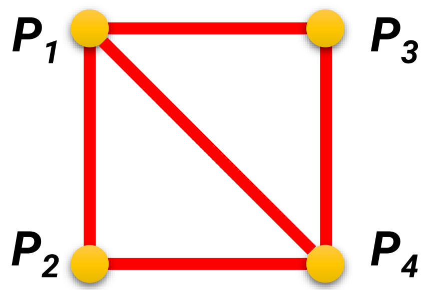
Vertex Buffer

P_1 x	P_1 y	P_1 z	P_2 x	P_2 y	P_2 z	P_3 x	P_3 y	P_3 z	P_4 x	P_4 y	P_4 z

Index Buffer

1	2	4	1	4	3

Note: in C++ impl., the index starts with 0



Now we need only 4 vertices and an integer array (save lots of memory when the vertex has many attributes)

Index Buffer

- **Generate a buffer and upload data**
  - Use the same functions as we create the vertex buffer, but with different parameters

```
// Draw a quad with indexed triangles.
glm::vec3 vertexPosition[4];
vertexPosition[0] = glm::vec3(-0.8f,  0.8f, 0.0f);
vertexPosition[1] = glm::vec3(-0.8f, -0.8f, 0.0f);
vertexPosition[2] = glm::vec3( 0.8f,  0.8f, 0.0f);
vertexPosition[3] = glm::vec3( 0.8f, -0.8f, 0.0f);
// Generate the vertex buffer.
glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertexPosition), vertexPosition, GL_STATIC_DRAW);

unsigned int vertexIndices[6] = { 0, 1, 3, 0, 3, 2 };
// Generate the index buffer.
glGenBuffers(1, &ibo);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ibo);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(vertexIndices), vertexIndices, GL_STATIC_DRAW);
```

Index Buffer (cont.)

- **Render with the vertex buffer and index buffer**

```
// Render a quad on screen.
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(glm::vec3), 0);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ibo);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
glDisableVertexAttribArray(0);
```

void **glDrawElements** (
GLenum mode ,
GLsizei count ,
GLenum type ,
const void * indices
);

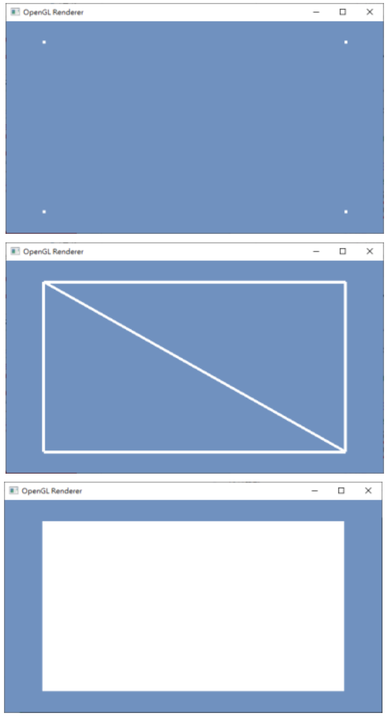
type: The data type of indices

indices: The start location (byte offset)

Change Polygon Render Mode

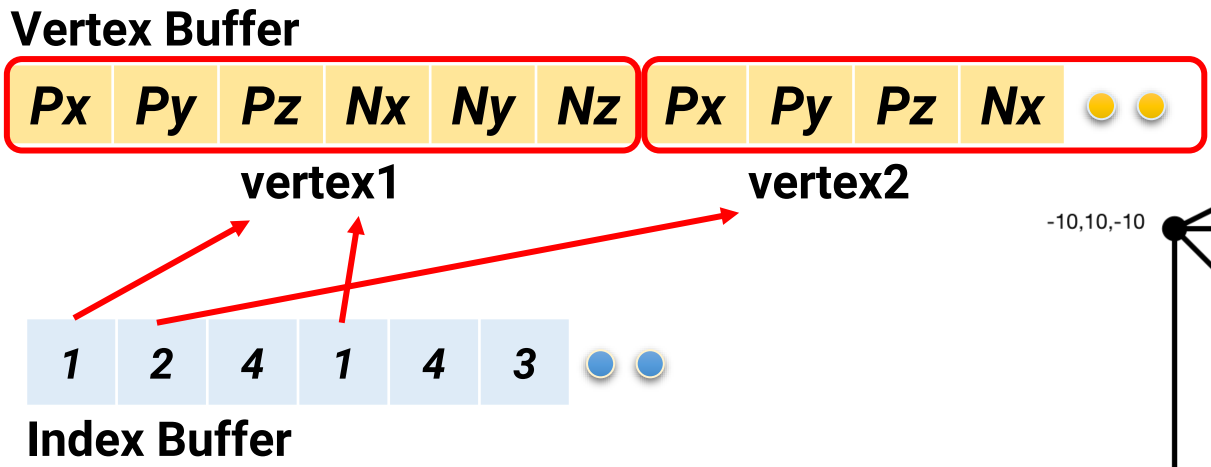
- OpenGL provides API for changing polygon render mode
  - void **glPolygonMode**(GLenum face, GLenum mode);

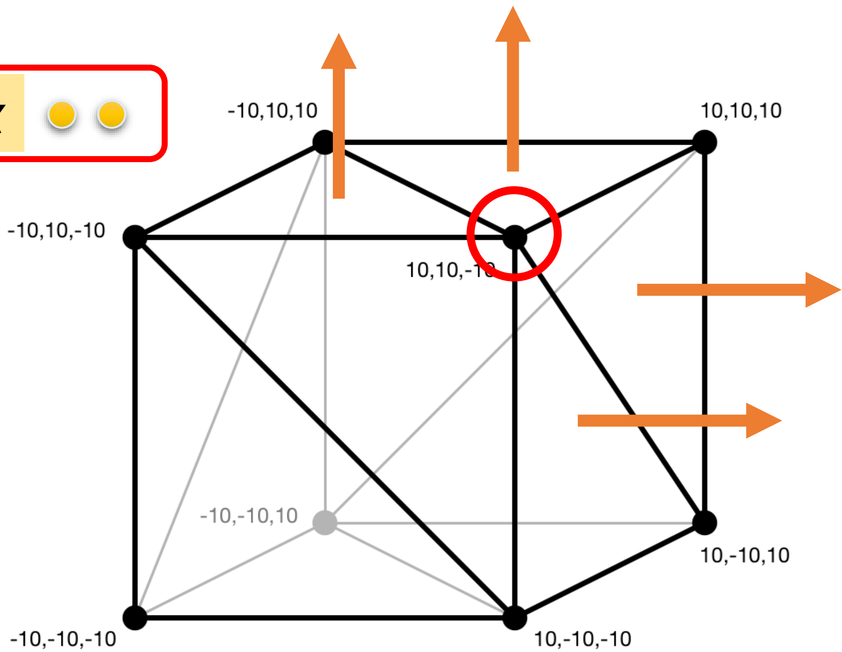
```
void ProcessSpecialKeysCB(int key, int x, int y)
{
    // Handle special (functional) keyboard inputs such as F1, spacebar, page up, etc.
    switch (key) {
    case GLUT_KEY_F1:
        // Render with point mode.
        glPointSize(5);
        glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
        break;
    case GLUT_KEY_F2:
        // Render with line mode.
        glLineWidth(5);
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
        break;
    case GLUT_KEY_F3:
        // Render with fill mode.
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        break;
    default:
        break;
    }
}
```



Pitfalls of Using Index Buffer

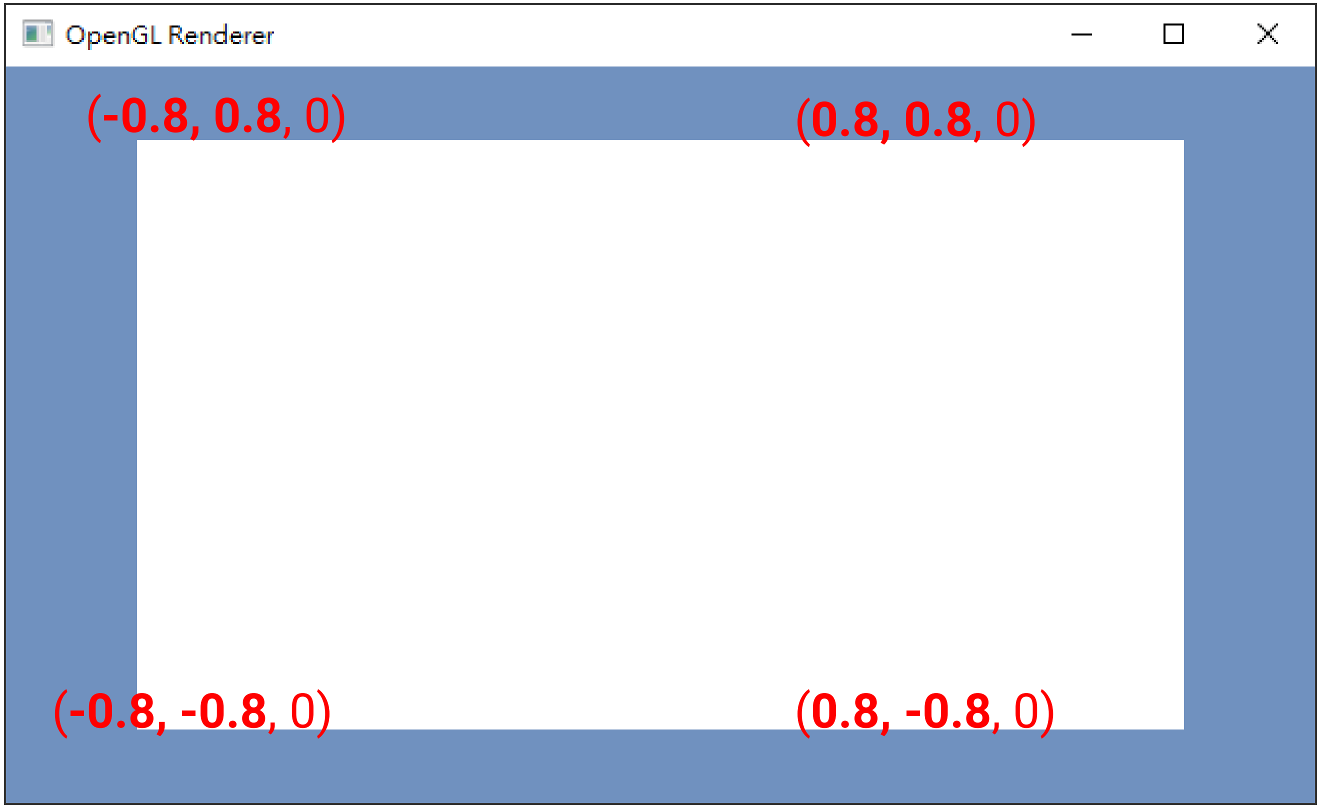
- Sometimes vertices will share the same positions but different other attributes such as **vertex normal** and **texture coordinate**
- These vertices should be stored **individually**





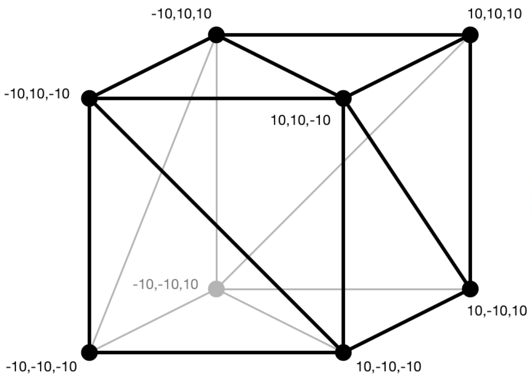
Question

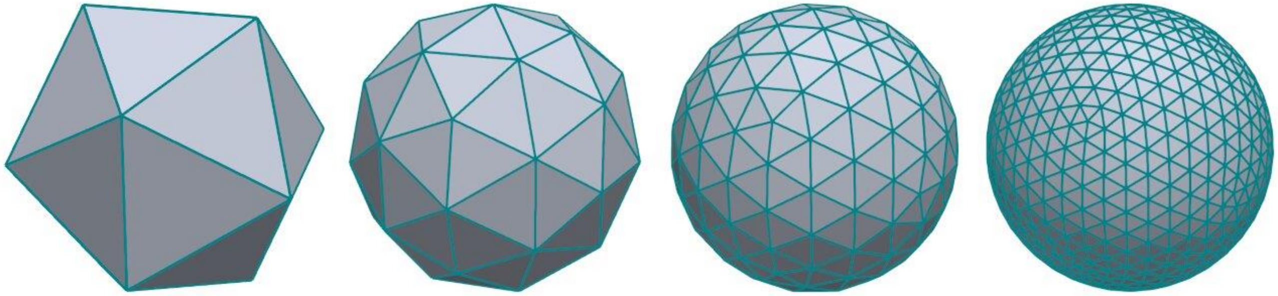
- A rectangle? Why not a square? (we will answer later)



Take Home Assignments

- Try to create some 3D shapes, e.g., cube, sphere …
    - Practice to create the vertex data
    - Practice to create a vertex buffer
    - Practice to render with a vertex buffer
    - Practice to render with a vertex and index buffer



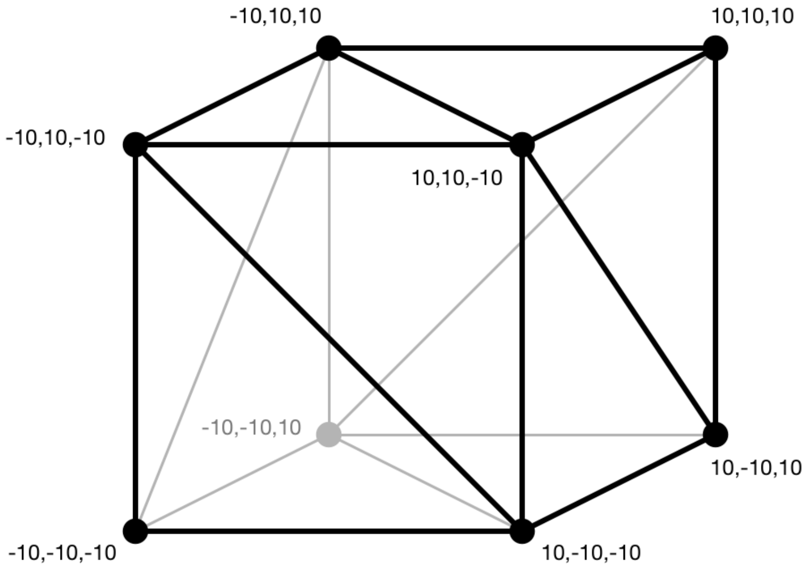


Outline

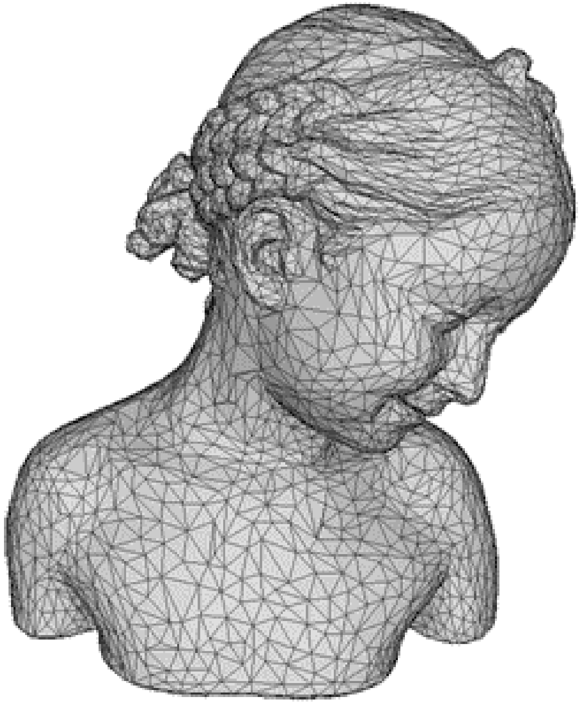
- Geometric properties and coordinate systems
- Draw shapes with OpenGL
- **Triangle meshes**

Triangle Mesh

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



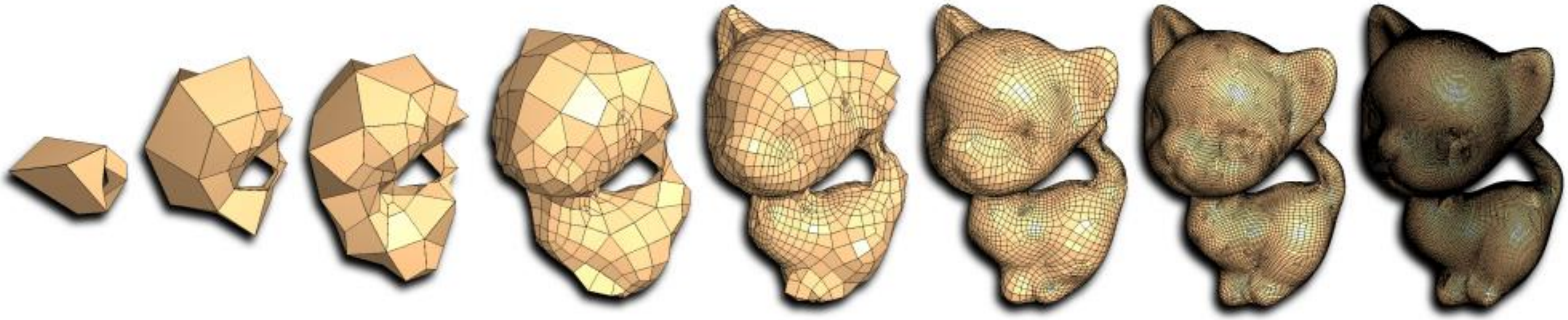
12 triangles

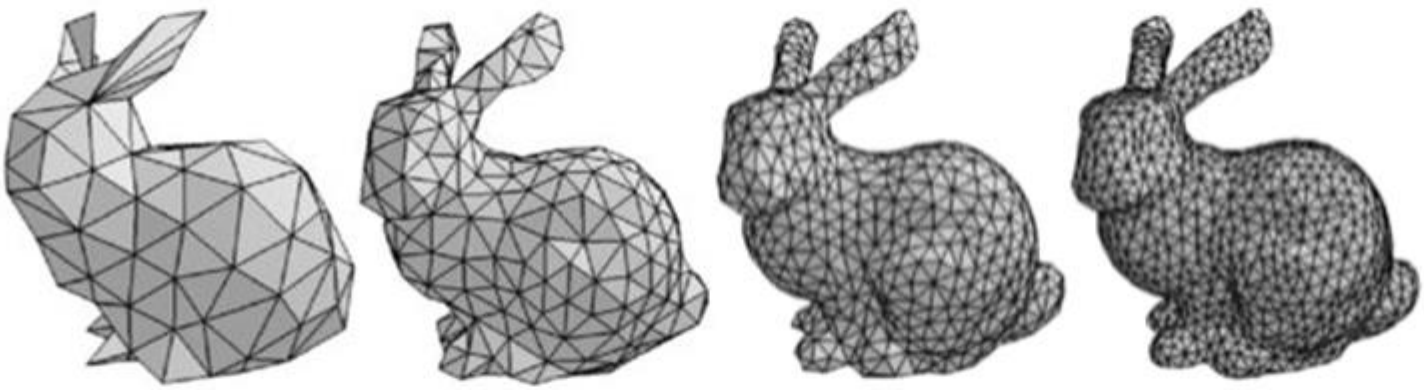


10K triangles

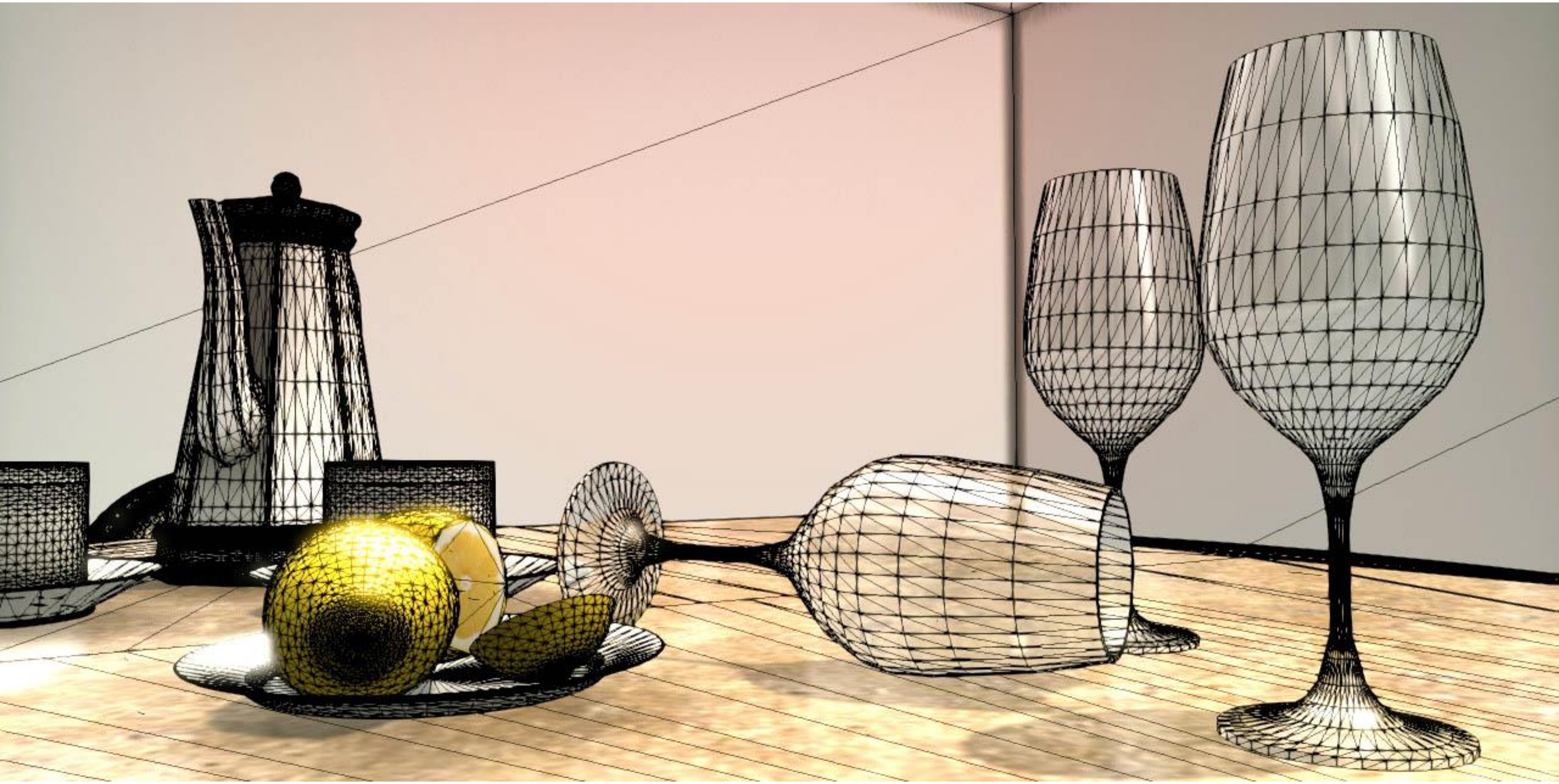
Triangle Mesh (cont.)

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

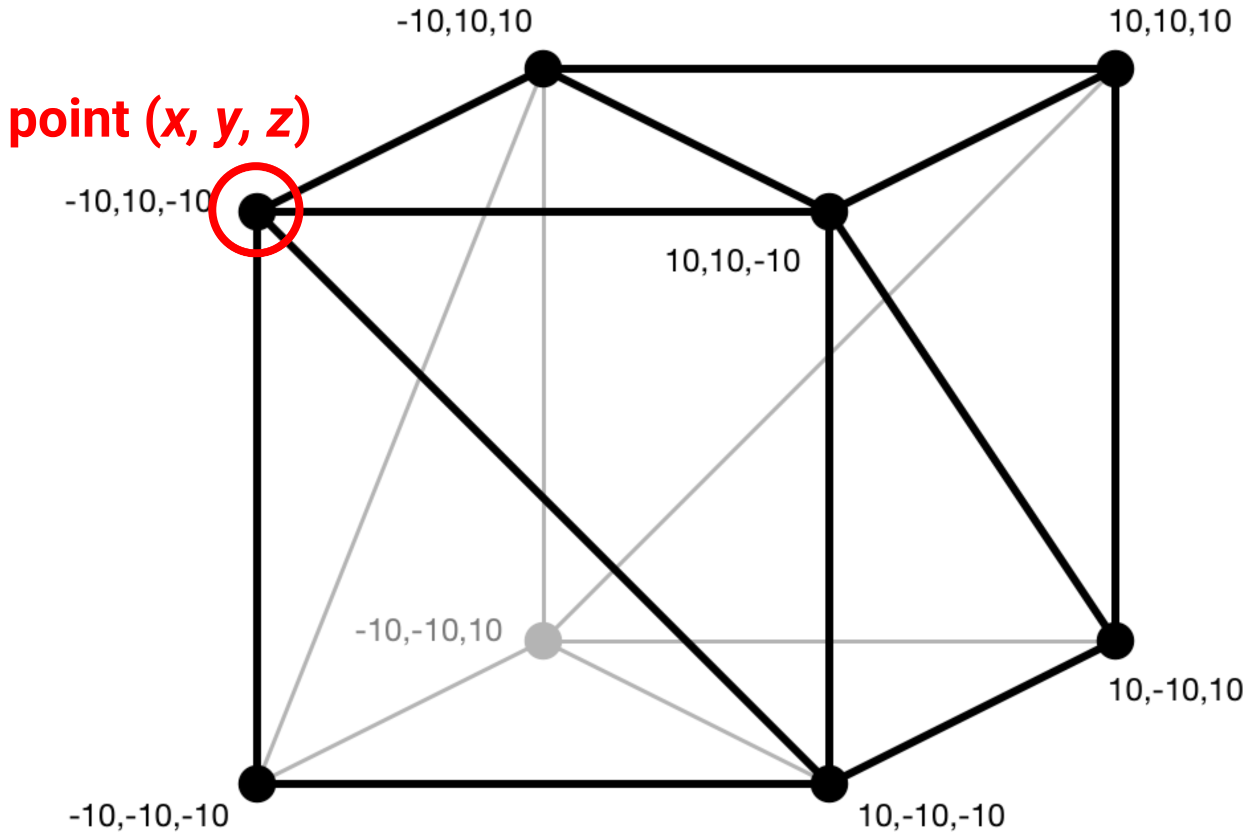




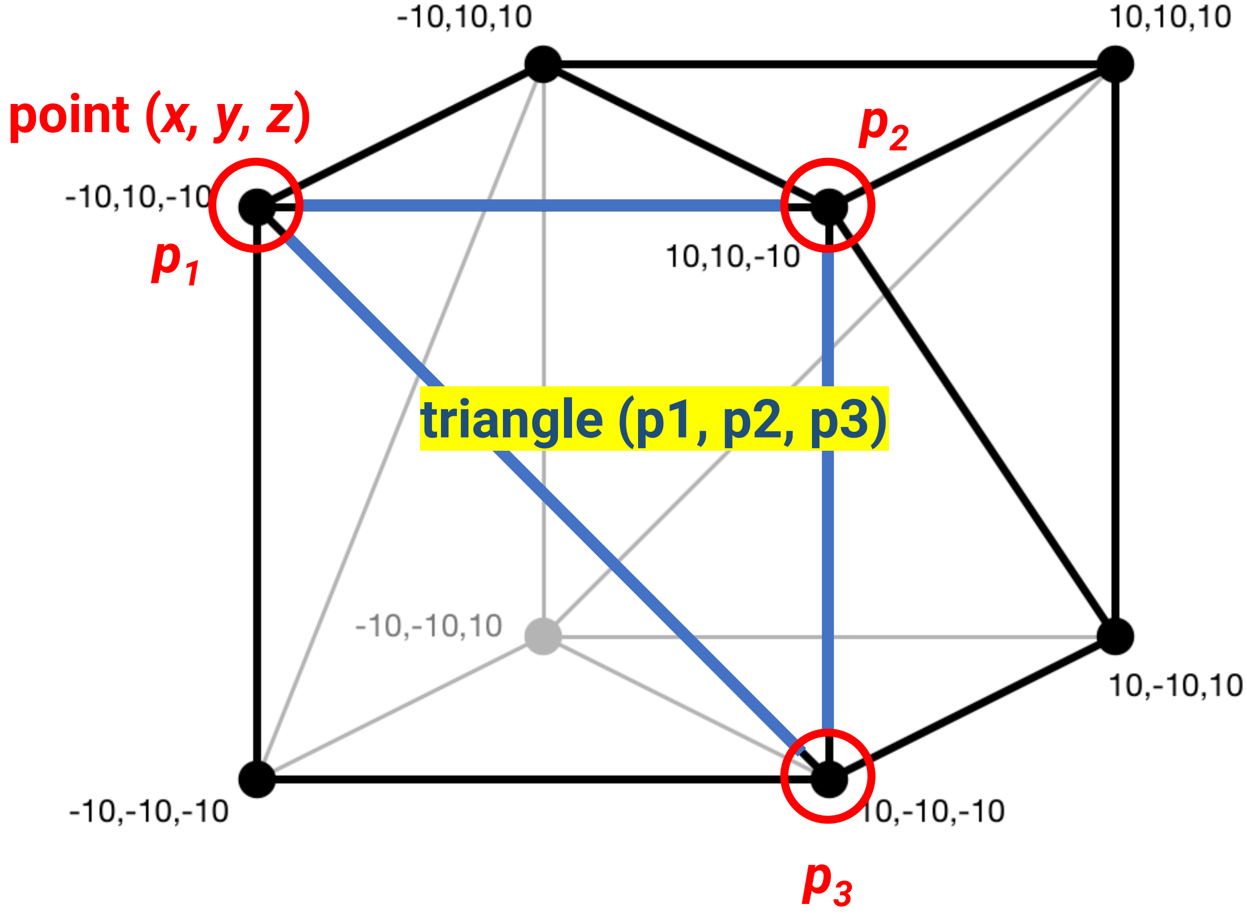
Scene Built with Triangle Mesh



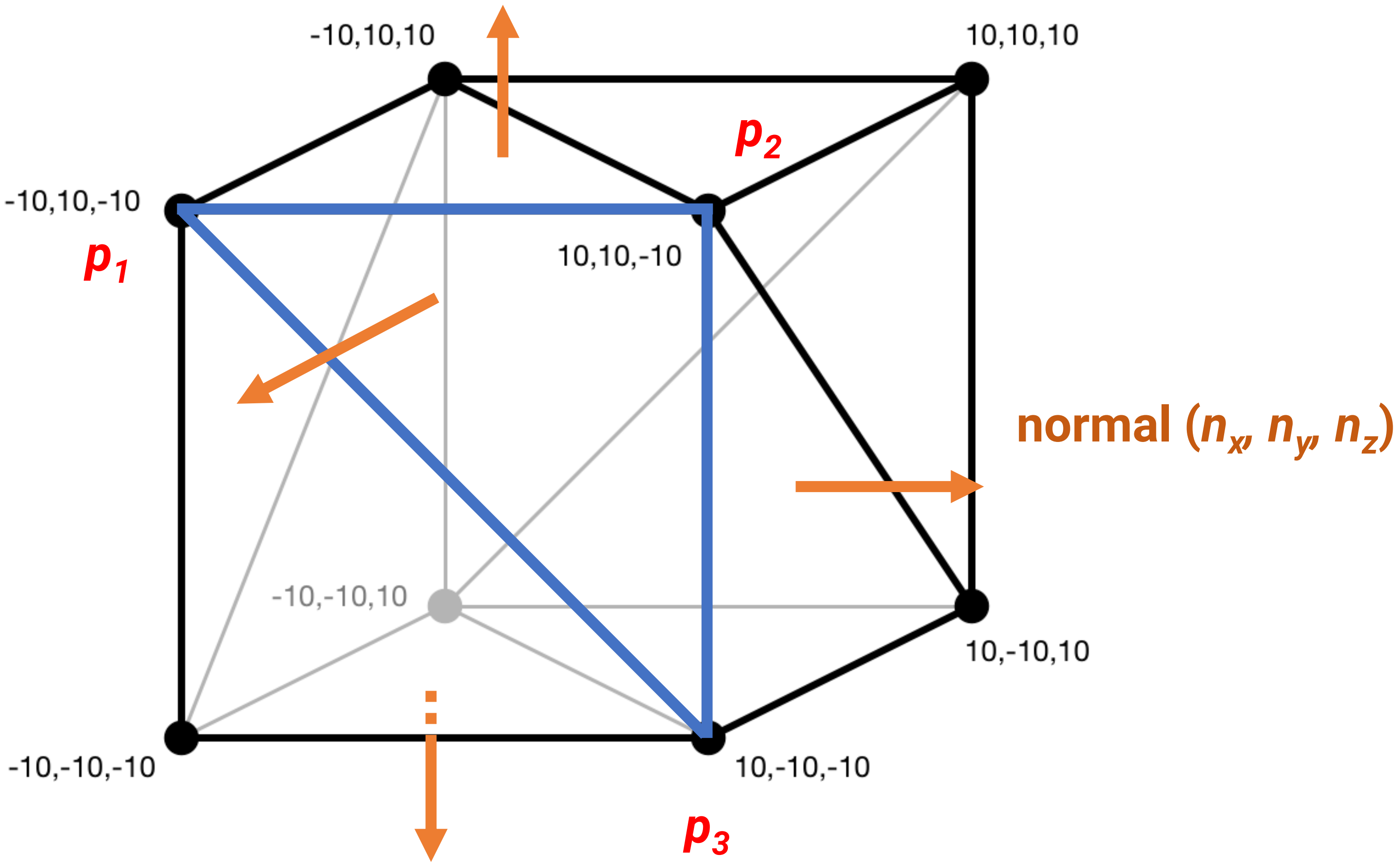
Point, Triangle, and Surface Normal



Point, Triangle, and Surface Normal

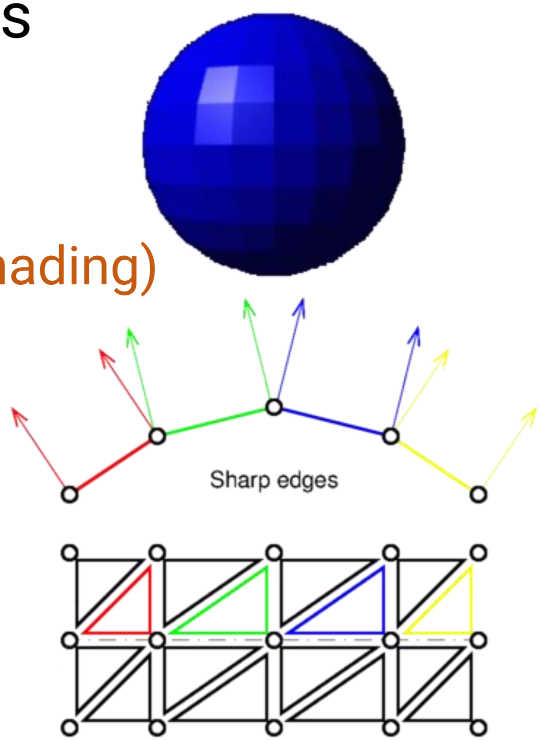


Point, Triangle, and Surface Normal

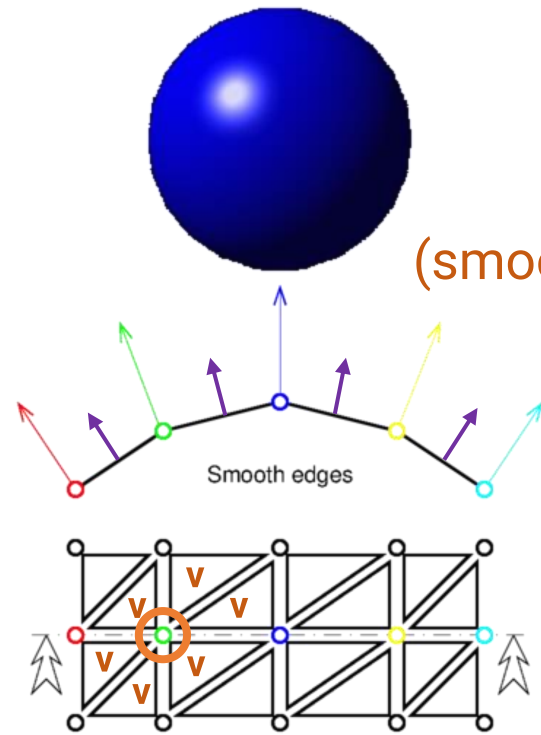


Vertex Normal

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



(flat shading)

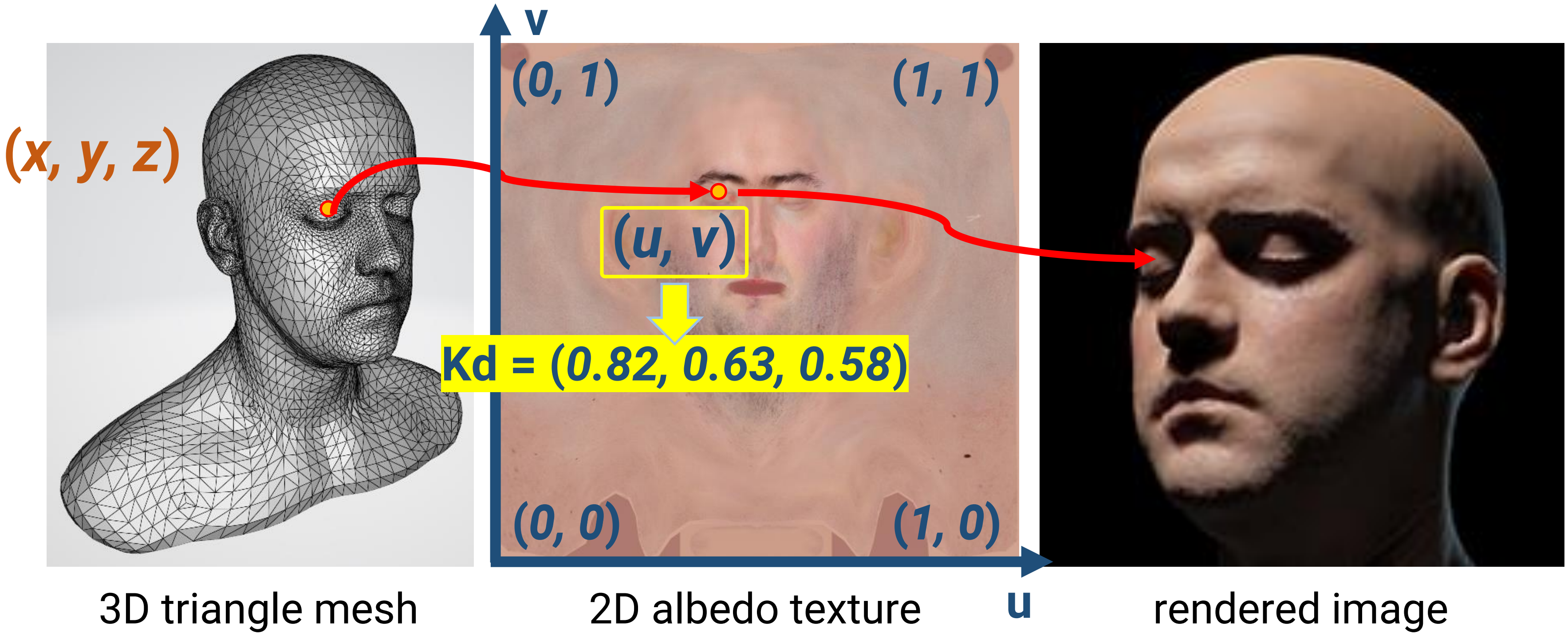


(smooth shading)

Vertex Texture Coordinate

Continue

- 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

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

TexCube.obj - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
# Blender v2.76 (sub 0) OBJ File: ''
# www.blender.org
```
comments

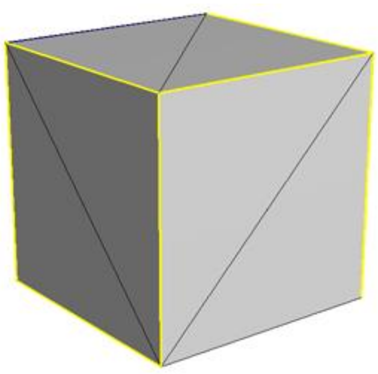
```
mtllib TexCube.mtl
```
specify material file

```
v 1.0 -1.0 -1.0
v 1.0 -1.0 1.0
v -1.0 -1.0 1.0
v -1.0 -1.0 -1.0
v 1.0 1.0 -1.0
v 1.0 1.0 1.0
v -1.0 1.0 1.0
v -1.0 1.0 -1.0
```
vertex position declaration

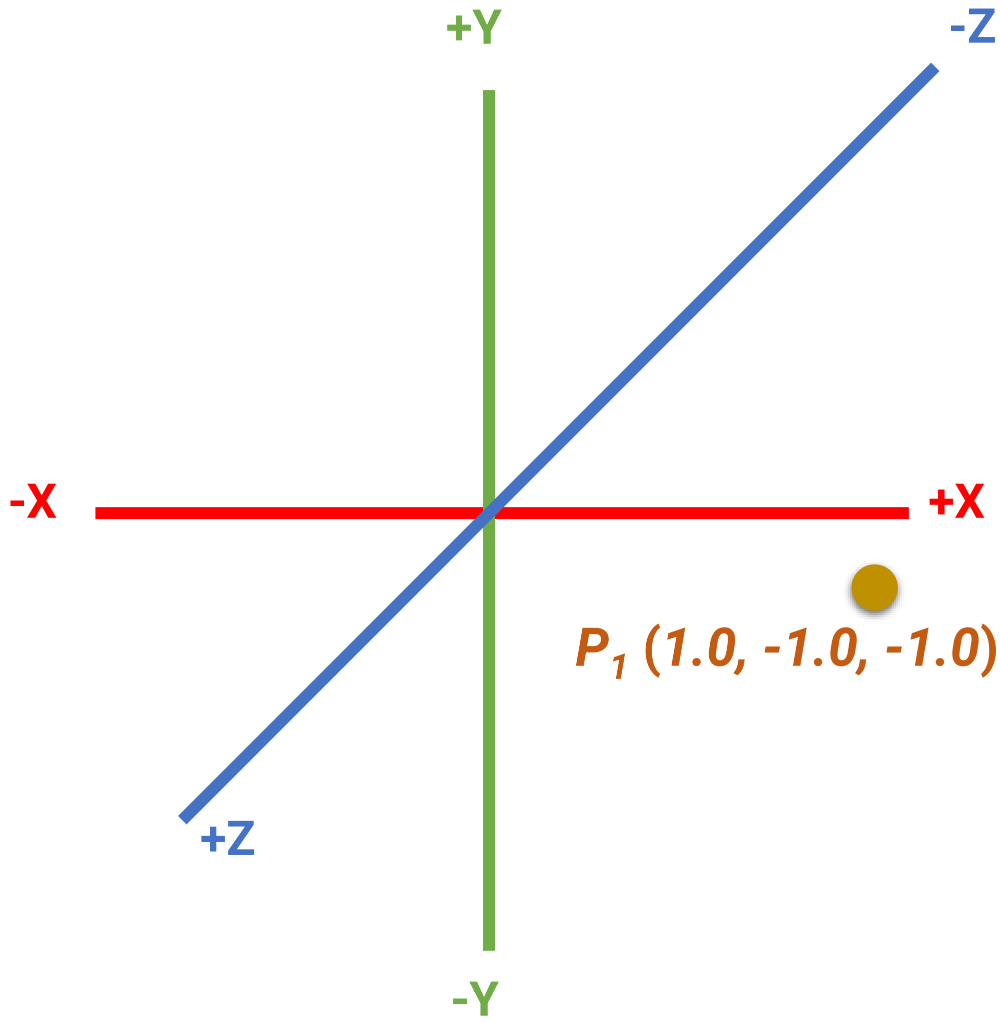
```
vt 0.0 0.0
vt 0.0 1.0
vt 1.0 0.0
vt 1.0 1.0
```
vertex texture coordinate declaration

```
vn 0.0 -1.0 0.0
vn 0.0 1.0 0.0
vn 1.0 0.0 0.0
vn -0.0 0.0 1.0
vn -1.0 -0.0 -0.0
vn 0.0 0.0 -1.0
```
vertex normal declaration

```
usemtl cubeMtl
f 8/2/2 7/1/2 6/3/2
f 5/4/2 8/2/2 6/3/2
f 2/4/1 3/2/1 4/1/1
f 1/3/1 2/4/1 4/1/1
f 2/3/4 6/4/4 3/1/4
f 6/4/4 7/2/4 3/1/4
f 5/4/3 6/2/3 2/1/3
f 1/3/3 5/4/3 2/1/3
f 3/3/5 7/4/5 8/2/5
f 4/1/5 3/3/5 8/2/5
f 5/2/6 1/1/6 8/4/6
f 1/1/6 4/3/6 8/4/6
```
face data (adjacency, submesh)

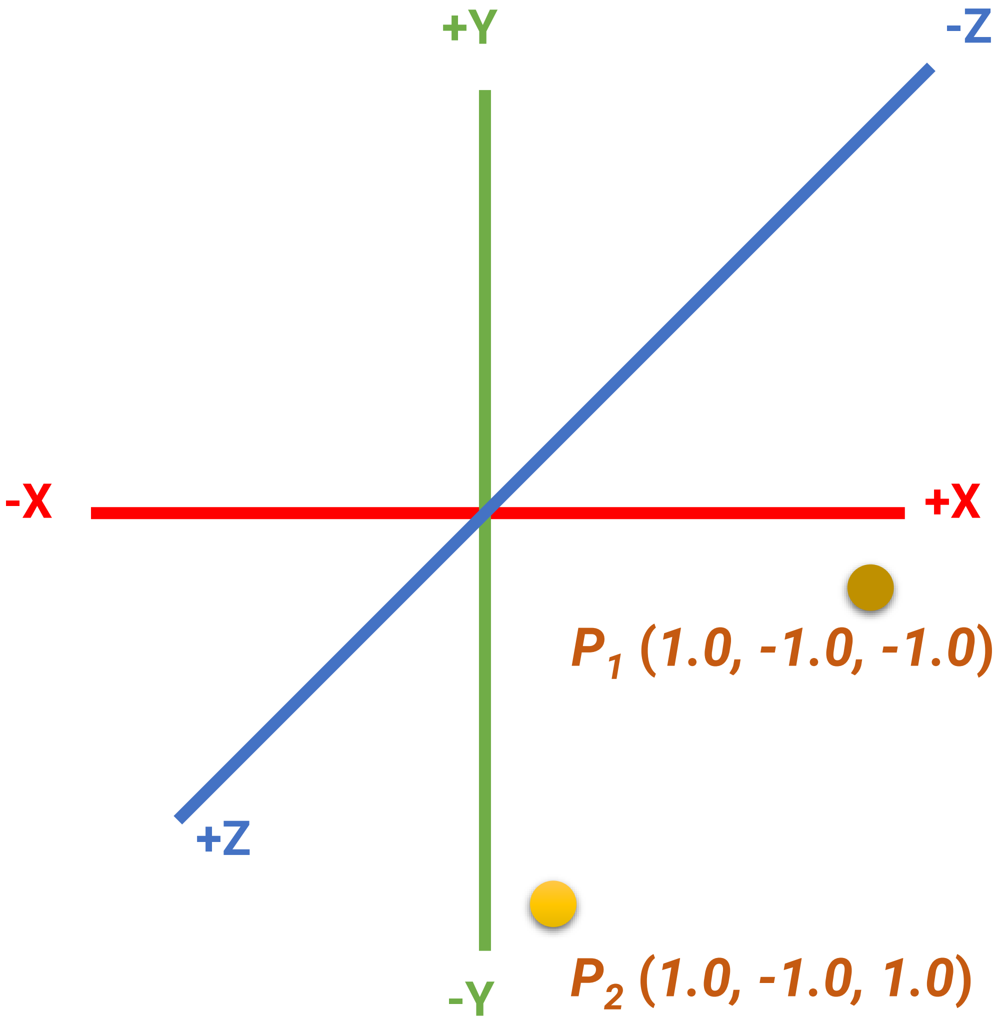


Example: Wavefront OBJ File Format (cont.)



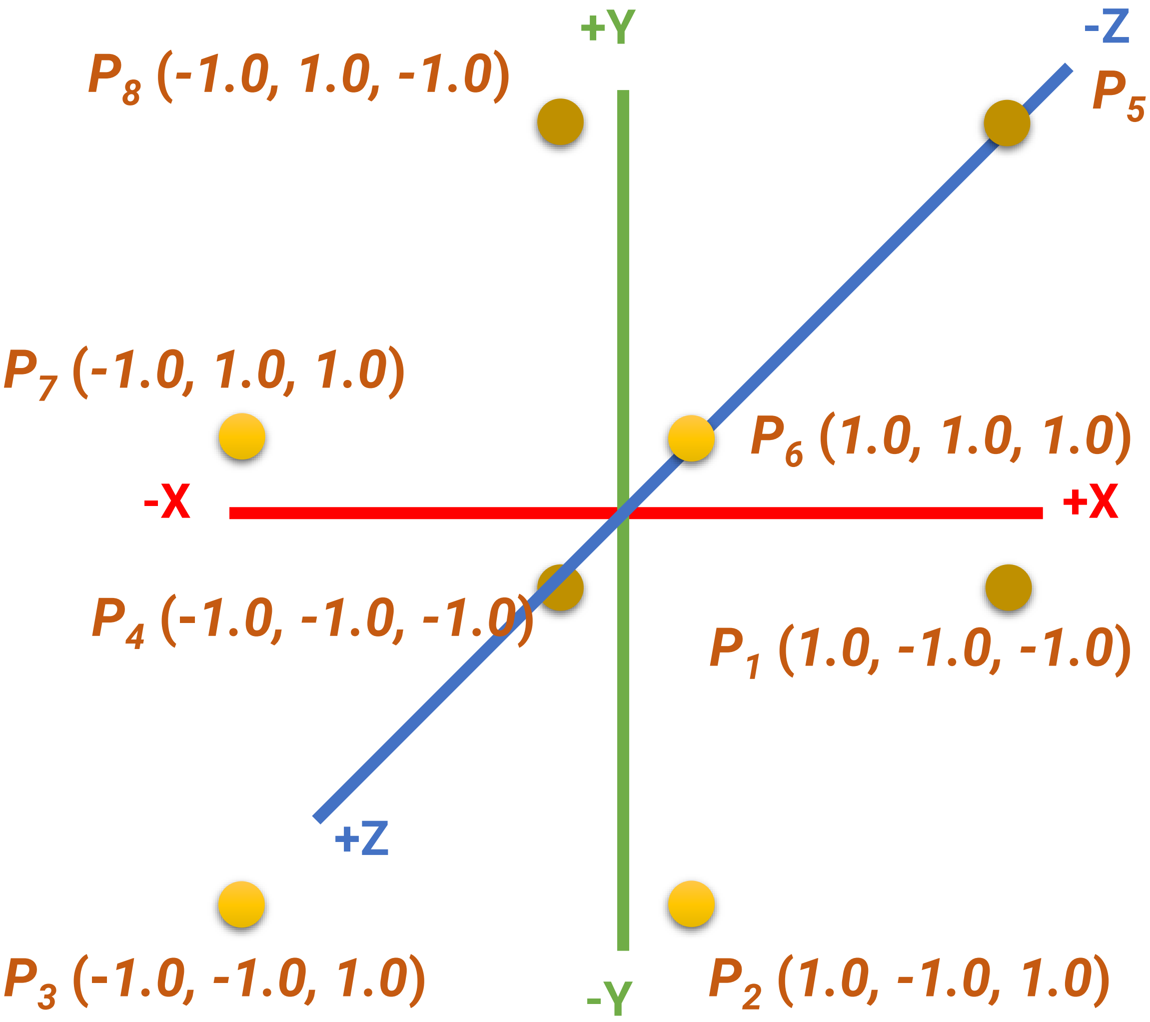
P_1	v 1.0 -1.0 -1.0
P_2	v 1.0 -1.0 1.0
P_3	v -1.0 -1.0 1.0
P_4	v -1.0 -1.0 -1.0
P_5	v 1.0 1.0 -1.0
P_6	v 1.0 1.0 1.0
P_7	v -1.0 1.0 1.0
P_8	v -1.0 1.0 -1.0

Example: Wavefront OBJ File Format (cont.)



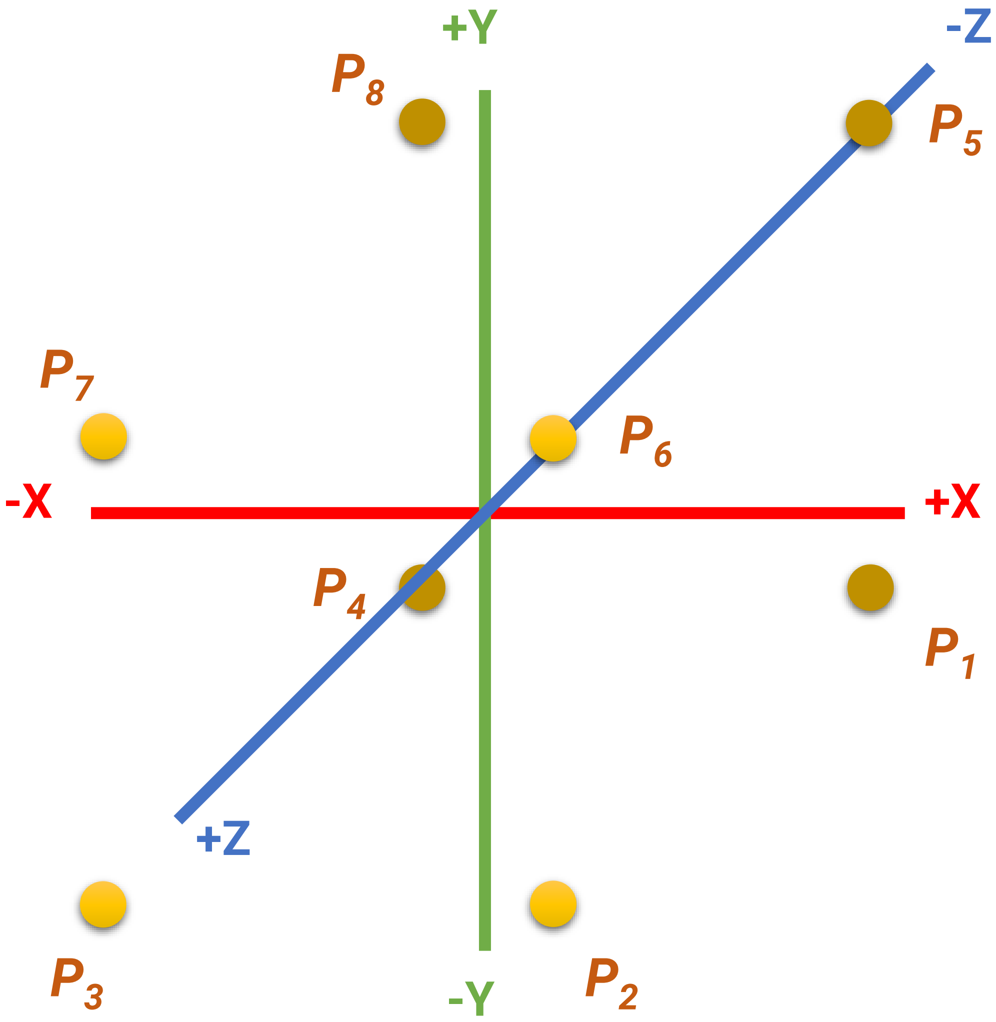
P_1	v 1.0 -1.0 -1.0
P_2	v 1.0 -1.0 1.0
P_3	v -1.0 -1.0 1.0
P_4	v -1.0 -1.0 -1.0
P_5	v 1.0 1.0 -1.0
P_6	v 1.0 1.0 1.0
P_7	v -1.0 1.0 1.0
P_8	v -1.0 1.0 -1.0

Example: Wavefront OBJ File Format (cont.)



P_1	v 1.0 -1.0 -1.0
P_2	v 1.0 -1.0 1.0
P_3	v -1.0 -1.0 1.0
P_4	v -1.0 -1.0 -1.0
P_5	v 1.0 1.0 -1.0
P_6	v 1.0 1.0 1.0
P_7	v -1.0 1.0 1.0
P_8	v -1.0 1.0 -1.0

Example: Wavefront OBJ File Format (cont.)

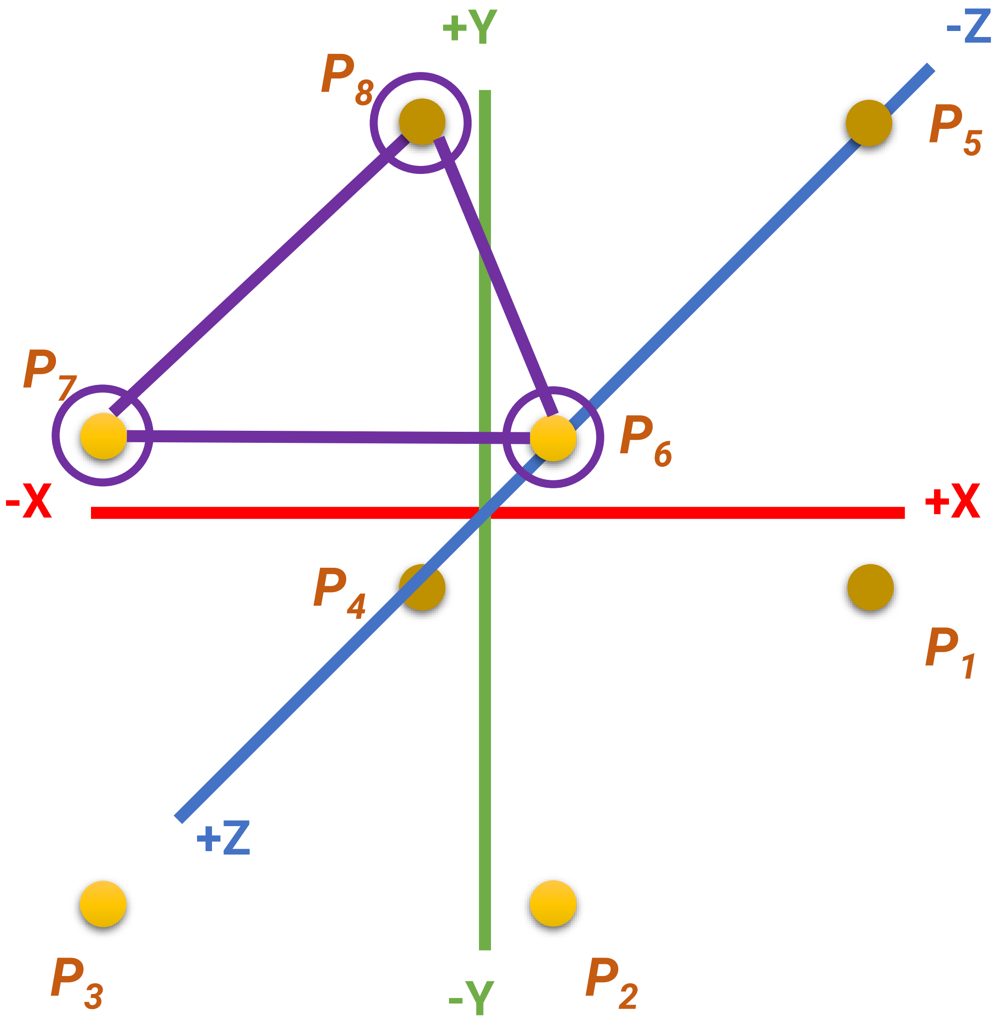


F_1	f	8/2/2	7/1/2	6/3/2
F_2	f	5/4/2	8/2/2	6/3/2
F_3	f	2/4/1	3/2/1	4/1/1
F_4	f	1/3/1	2/4/1	4/1/1
F_5	f	2/3/4	6/4/4	3/1/4
F_6	f	6/4/4	7/2/4	3/1/4
F_7	f	5/4/3	6/2/3	2/1/3
F_8	f	1/3/3	5/4/3	2/1/3
F_9	f	3/3/5	7/4/5	8/2/5
F_{10}	f	4/1/5	3/3/5	8/2/5
F_{11}	f	5/2/6	1/1/6	8/4/6
F_{12}	f	1/1/6	4/3/6	8/4/6

	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.)

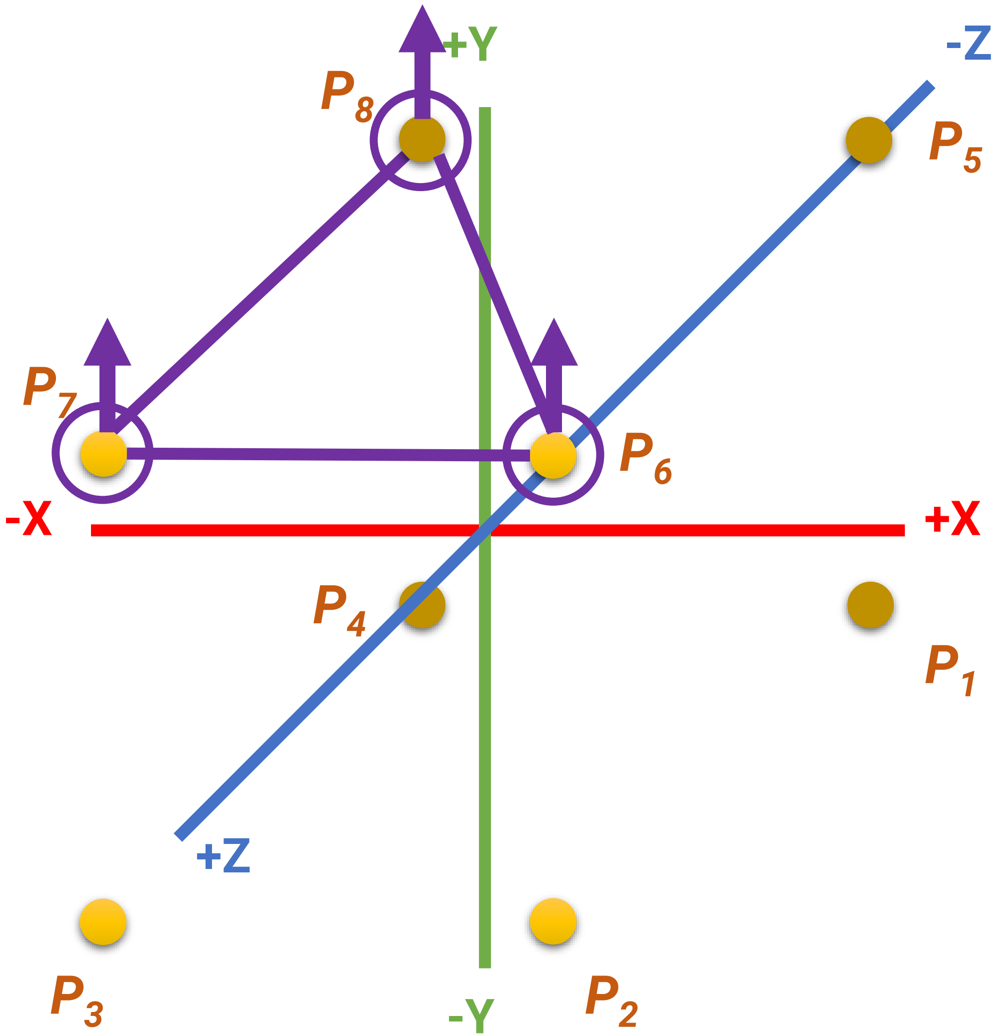


F_1	f	8/2/2	7/1/2	6/3/2
F_2	f	5/4/2	8/2/2	6/3/2
F_3	f	2/4/1	3/2/1	4/1/1
F_4	f	1/3/1	2/4/1	4/1/1
F_5	f	2/3/4	6/4/4	3/1/4
F_6	f	6/4/4	7/2/4	3/1/4
F_7	f	5/4/3	6/2/3	2/1/3
F_8	f	1/3/3	5/4/3	2/1/3
F_9	f	3/3/5	7/4/5	8/2/5
F_{10}	f	4/1/5	3/3/5	8/2/5
F_{11}	f	5/2/6	1/1/6	8/4/6
F_{12}	f	1/1/6	4/3/6	8/4/6

	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

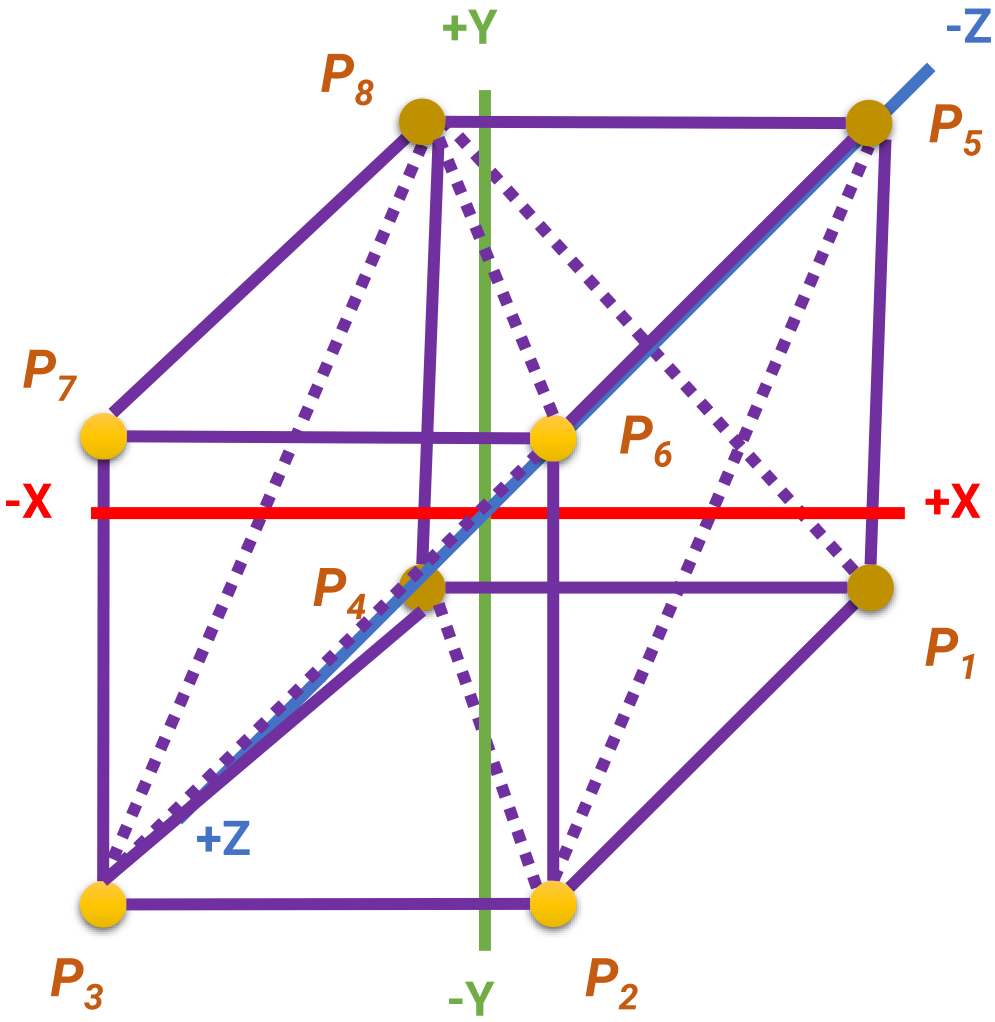
```
f 8/2/2 7/1/2 6/3/2
```

N_1	vn 0.0 -1.0 0.0
N_2	vn 0.0 1.0 0.0
N_3	vn 1.0 0.0 0.0
N_4	vn -0.0 0.0 1.0
N_5	vn -1.0 -0.0 -0.0
N_6	vn 0.0 0.0 -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	8/2/2	7/1/2	6/3/2
F_2	f	5/4/2	8/2/2	6/3/2
F_3	f	2/4/1	3/2/1	4/1/1
F_4	f	1/3/1	2/4/1	4/1/1
F_5	f	2/3/4	6/4/4	3/1/4
F_6	f	6/4/4	7/2/4	3/1/4
F_7	f	5/4/3	6/2/3	2/1/3
F_8	f	1/3/3	5/4/3	2/1/3
F_9	f	3/3/5	7/4/5	8/2/5
F_{10}	f	4/1/5	3/3/5	8/2/5
F_{11}	f	5/2/6	1/1/6	8/4/6
F_{12}	f	1/1/6	4/3/6	8/4/6

	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


END OF PRESENTATION
ANY QUESTIONS?