

Graphics Programming

Fall 2025

9/18/2025

Kyoung Shin Park
Computer Engineering
Dankook University

Angles, Degrees, and Radians

- ❑ General math library functions uses radians.
- ❑ $360 \text{ degrees} (^{\circ}) = 1 \text{ full circle} = 2 \pi \text{ radians}$
- ❑ $1 \text{ radian} = 180.0/\pi \text{ degree} \approx 57.29578 \text{ degree}$
or $1 \text{ degree} = \pi/180.0 \text{ radian} \approx 0.01745329 \text{ radian}$

```
#ifndef M_PI
```

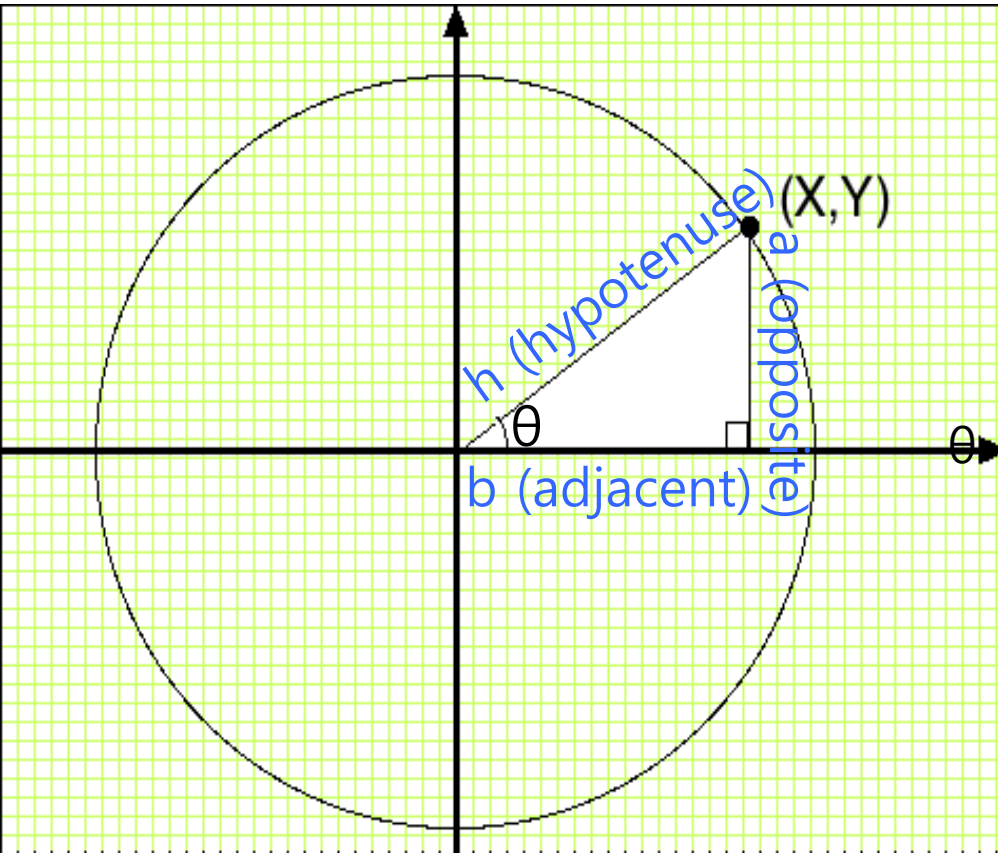
```
#define M_PI 3.141592654f
```

```
#endif
```

```
#define DegreesToRadians(degree) ((degree) * (M_PI / 180.0f))
```

```
#define RadiansToDegrees(radian) ((radian) * (180.0f / M_PI))
```

Trigonometry



- $\sin\theta = a/h$
 $\cos\theta = b/h$
 $\tan\theta = a/b$
- $b = h \cdot \cos\theta$
 $a = h \cdot \sin\theta$
- $x^2 + y^2 = 1$
 $x = \cos\theta$
 $y = \sin\theta$
 $y/x = \sin\theta/\cos\theta = \tan\theta$
- $x = \text{distance} \cdot \cos\theta$
 $y = \text{distance} \cdot \sin\theta$

Trigonometry

- ▣ Multiplicative inverse:

$$\csc\theta = 1/\sin\theta$$

$$\sec\theta = 1/\cos\theta$$

$$\cot\theta = 1/\tan\theta = \cos\theta/\sin\theta = x/y$$

- ▣ Inverse:

$$\arcsin(x) = \sin^{-1}(x)$$

$$\text{where } y = \arcsin(x) \quad x: [-1, 1] \rightarrow y: [-\pi/2, \pi/2]$$

$$\arccos(x) = \cos^{-1}(x)$$

$$\text{where } y = \arccos(x) \quad x: [-1, 1] \rightarrow y: [0, \pi]$$

$$\arctan(x) = \tan^{-1}(x)$$

$$\text{where } y = \arctan(x) \quad x: [-\infty, \infty] \rightarrow y: [-\pi/2, \pi/2]$$

Trigonometric Identity

- $\sin^2\theta + \cos^2\theta = 1$

- $1 + \tan^2\theta = \sec^2\theta$

- $1 + \cot^2\theta = \csc^2\theta$

- $\sin(\pi/2 - \theta) = \cos\theta$

- $\cos(\pi/2 - \theta) = \sin\theta$

- $\tan(\pi/2 - \theta) = \cot\theta$

- $\sin(x+y) = \sin x \cos y + \cos x \sin y$

- $\sin(x-y) = \sin x \cos y - \cos x \sin y$

- $\cos(x+y) = \cos x \cos y - \sin x \sin y$

- $\cos(x-y) = \cos x \cos y + \sin x \sin y$

- $\sin 2\theta = 2\sin\theta\cos\theta$

- $\cos 2\theta = \cos^2\theta - \sin^2\theta = 2\cos^2\theta - 1 = 1 - 2\sin^2\theta$

Geometric Primitives□

- The most basic elements in expressing object
- In real-time graphics, linear primitives are used
 - Point
 - Line, Line Segment, Ray
 - Sphere, Cylinder, Cone
 - Cube (Box)
 - Triangle
 - Polygon, ...
- Requirements for polygons
 - The polygon specified must **not intersect** itself.
 - Must be **convex**.
 - Its vertices are co-planar.

Primitive Types

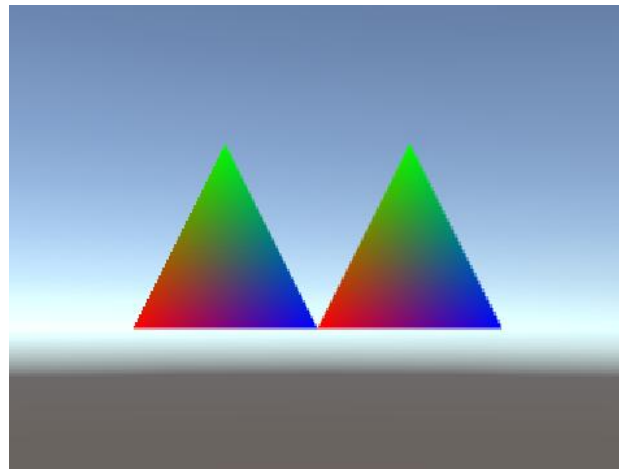
- Unity GL primitive types
 - **GL.LINES**
 - **GL.LINE_STRIP**
 - **GL.TRIANGLES**
 - **GL.TRIANGLE_STRIP**
 - **GL.QUADS**

2 Triangles

□ Draw 2 triangles(Unity LHS x+ right y+ up z+ inside) CW

■ GL_TRIANGLES

```
void FilledTriangle(Color c1, Color c2, Color c3, Vector3 p1, Vector3 p2, Vector3 p3) {  
    GL.PushMatrix();  
    GL.Begin(GL_TRIANGLES);  
    GL.Color(c1);  
    GL.Vertex(p1);  
    GL.Color(c2);  
    GL.Vertex(p2);  
    GL.Color(c3);  
    GL.Vertex(p3);  
    GL.End();  
    GL.PopMatrix();  
}
```



```
GLGeometry.FilledTriangle(Color.red, Color.green, Color.blue, new Vector3(-2, 0, 0),  
new Vector3(-1, 2, 0), new Vector3(0, 0, 0));  
GLGeometry.FilledTriangle(Color.red, Color.green, Color.blue, new Vector3(0, 0, 0),  
new Vector3(1, 2, 0), new Vector3(2, 0, 0));
```


Quad

- Draw a quad(Unity LHS x+ right y+ up z+ inside) CW

- GL_QUADS

```
void Quad(Color c1, Color c2, Color c3, Color c4, Vector3 p1, Vector3 p2, Vector3 p3, Vector3 p4) {
```

```
    GL.PushMatrix();
```

```
    GL.Begin(GL_QUADS);
```

```
    GL.Color(c1);
```

```
    GL.Vertex(p1);
```

```
    GL.Color(c2);
```

```
    GL.Vertex(p2);
```

```
    GL.Color(c3);
```

```
    GL.Vertex(p3);
```

```
    GL.Color(c4);
```

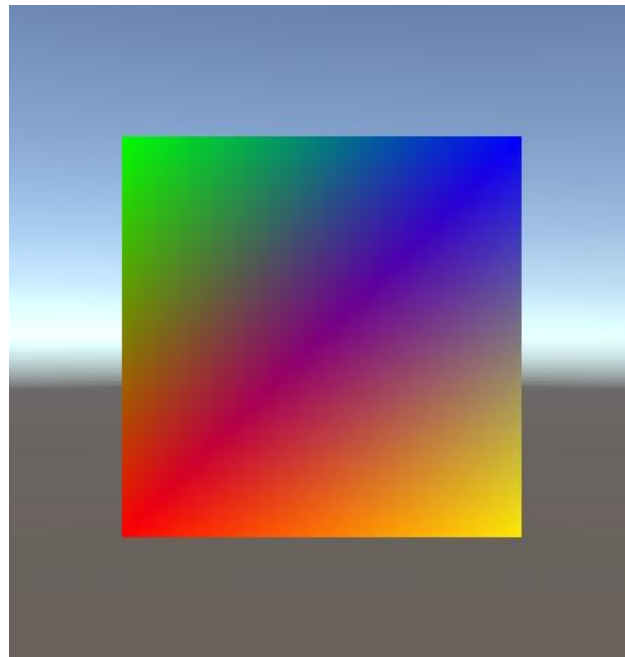
```
    GL.Vertex(p4);
```

```
    GL.End();
```

```
    GL.PopMatrix();
```

```
}
```

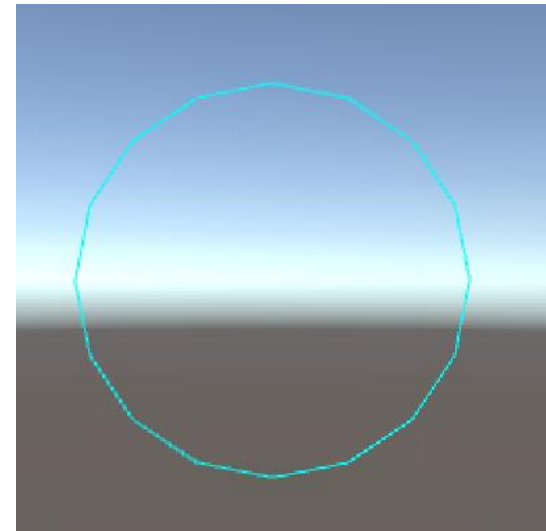
```
GLGeometry.Quad(Color.red, Color.green, Color.blue, Color.yellow, new Vector3(-2, -2, 0), new Vector3(-2, 2, 0), new Vector3(2, 2, 0), new Vector3(2, -2, 0));
```



Circle

- Draw a wireframe circle
 - GL_LINE_STRIP

```
void Circle(Color color, float radius, int segments) {  
    GL.PushMatrix();  
    GL.Begin(GL.LINE_STRIP);  
    GL.Color(color);  
    float deltaTheta = -2.0f * Mathf.PI / segments;  
    for (int i = 0; i <= segments; i++) {  
        float theta = i * deltaTheta;  
        float x = radius * Mathf.Cos(theta);  
        float y = radius * Mathf.Sin(theta);  
        GL.Vertex(new Vector3(x, y, 0));  
    }  
    GL.End();  
    GL.PopMatrix();  
}  
GLGeometry.Circle(Color.cyan, 2, 16);
```

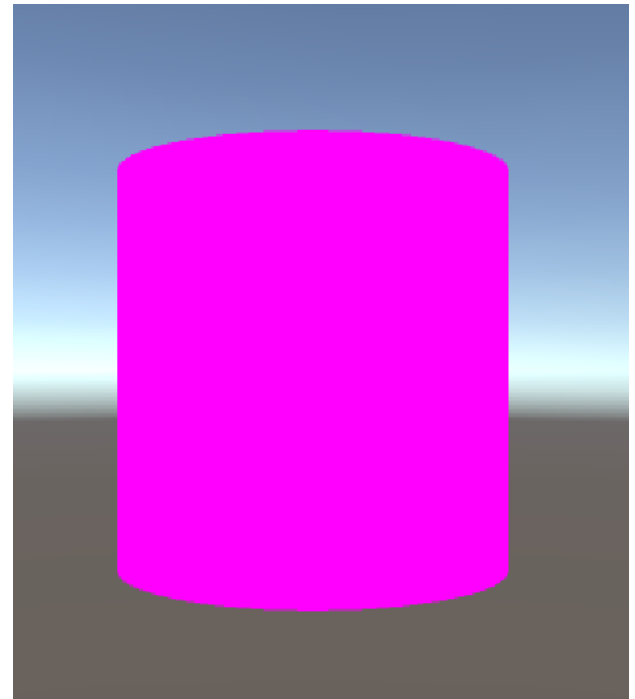


Cylinder

□ Draw a wireframe cylinder

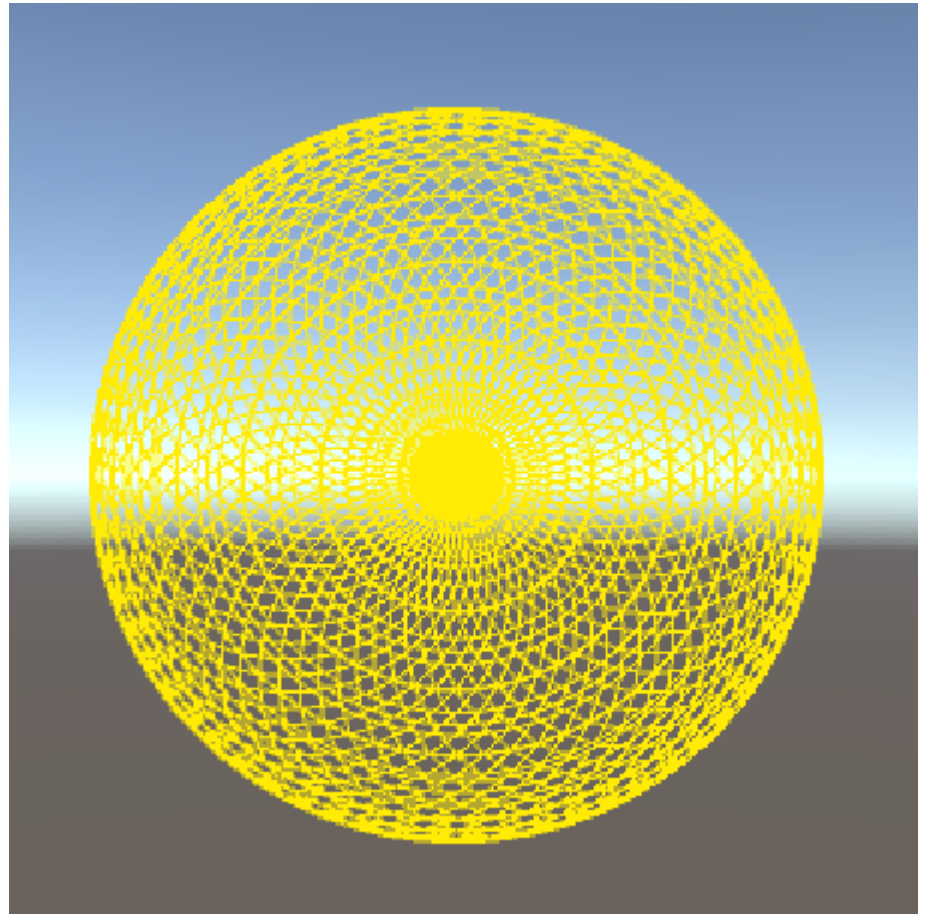
■ GL_LINE_STRIP

```
void Cylinder(Color color, float radius, float height, int segments) {  
    GL.PushMatrix();  
    GL.Begin(GL.TRIANGLE_STRIP);  
    GL.Color(color);  
    float deltaTheta = 2.0f * Mathf.PI / segments;  
    for (int i = 0; i <= segments; i++) {  
        float theta = i * deltaTheta;  
        float x = radius * Mathf.Cos(theta);  
        float y = -height/2;  
        float z = radius * Mathf.Sin(theta);  
        GL.Vertex(new Vector3(x, y, z));  
        y = height/2;  
        GL.Vertex(new Vector3(x, y, z));  
    }  
    GL.End();  
    GL.PopMatrix();  
}
```



Sphere

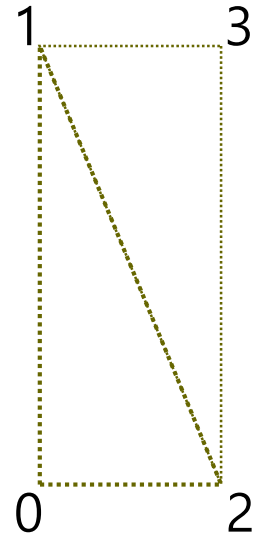
- ❑ Draw a wireframe sphere
 - GL_LINE_STRIP
- ❑ Draw a solid sphere
 - GL_TRIANGLE_STRIP



```

void Sphere(Color color, float radius, int stacks, int slices) {
    GL.PushMatrix();
    GL.Begin(GL.TRIANGLE_STRIP);
    GL.Color(color);
    float lonstep = Mathf.PI / stacks; float latstep = Mathf.PI / slices;
    for (float lon = 0.0f; lon <= 2*Mathf.PI; lon += lonstep) {
        for (float lat = 0.0f; lat <= Mathf.PI + latstep; lat += latstep) {
            float x = radius * Mathf.Cos(lon) * Mathf.Sin(lat);
            float y = radius * Mathf.Sin(lon) * Mathf.Sin(lat);
            float z = radius * Mathf.Cos(lat);
            GL.Vertex(new Vector3(x, y, z));
            x = radius * Mathf.Cos(lon + lonstep) * Mathf.Sin(lat);
            y = radius * Mathf.Sin(lon + lonstep) * Mathf.Sin(lat);
            z = radius * Mathf.Cos(lat);
            GL.Vertex(new Vector3(x, y, z));
        }
    }
    GL.End();
    GL.PopMatrix();
}

```



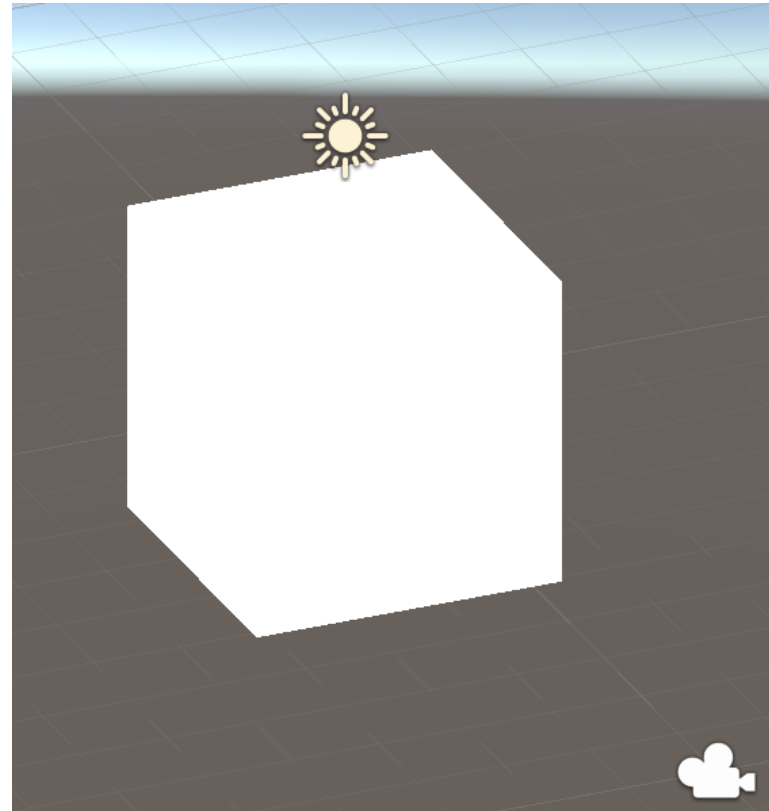
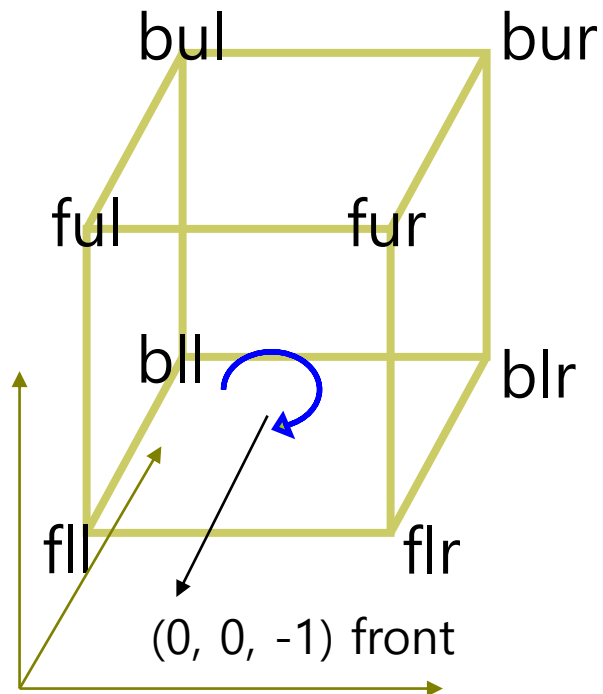
경도(lon) 위도(lat)

$$\begin{aligned}
 x &= \cos\varphi * \cos\theta \\
 y &= \sin\theta \\
 z &= \sin\varphi * \cos\theta \\
 \text{where } 0 \leq \varphi \leq 2\pi, -\pi/2 \leq \theta \leq \pi/2
 \end{aligned}$$

Cube

- Draw a solid cube (Unity LHS x+ right y+ up z+ inside)
CW

- GL_QUADS



```
void Cube(Color color, Vector3 center, float size) {  
    Vector3 fll = center + new Vector3(-size, -size, -size);  
    Vector3 flr = center + new Vector3( size, -size, -size);  
    Vector3 ful = center + new Vector3(-size,  size, -size);  
    Vector3 fur = center + new Vector3(size,  size, -size);  
    Vector3 bll = center + new Vector3(-size, -size,  size);  
    Vector3 blr = center + new Vector3( size, -size,  size);  
    Vector3 bul = center + new Vector3(-size,  size,  size);  
    Vector3 bur = center + new Vector3(size,  size,  size);
```

```
    GL.PushMatrix();  
    GL.Begin(GL.QUADS);  
    GL.Color(color);
```

```
    // front face
```

```
    GL.Vertex(fll);  
    GL.Vertex(ful);  
    GL.Vertex(fur);  
    GL.Vertex(flr);
```

Cube

// back face

GL.Vertex(blr);

GL.Vertex(bur);

GL.Vertex(bul);

GL.Vertex(bll);

// left face

GL.Vertex(bll);

GL.Vertex(bul);

GL.Vertex(ful);

GL.Vertex(fll);

// right face

GL.Vertex(flr);

GL.Vertex(fur);

GL.Vertex(bur);

GL.Vertex(blr);

Cube

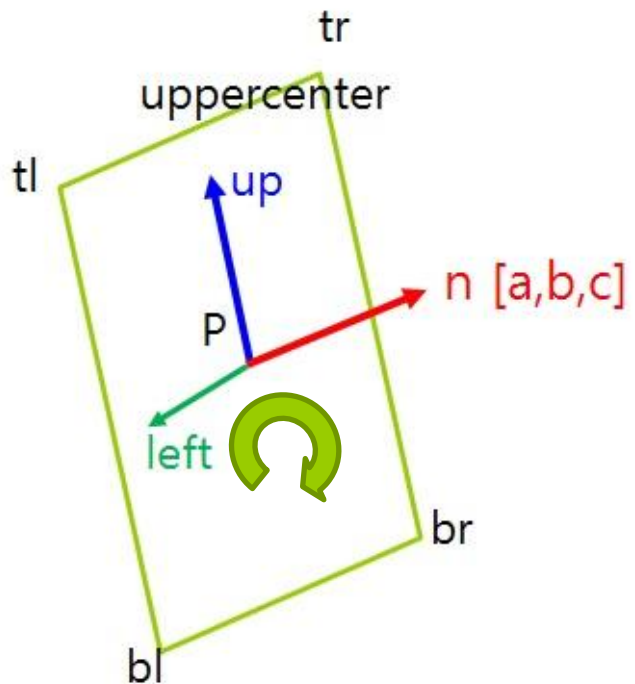
```
// top face  
GL.Vertex(ful);  
GL.Vertex(bul);  
GL.Vertex(bur);  
GL.Vertex(fur);
```

```
// bottom face  
GL.Vertex(blr);  
GL.Vertex(bll);  
GL.Vertex(fll);  
GL.Vertex(flrr);
```

```
GL.End();  
GL.PopMatrix();
```

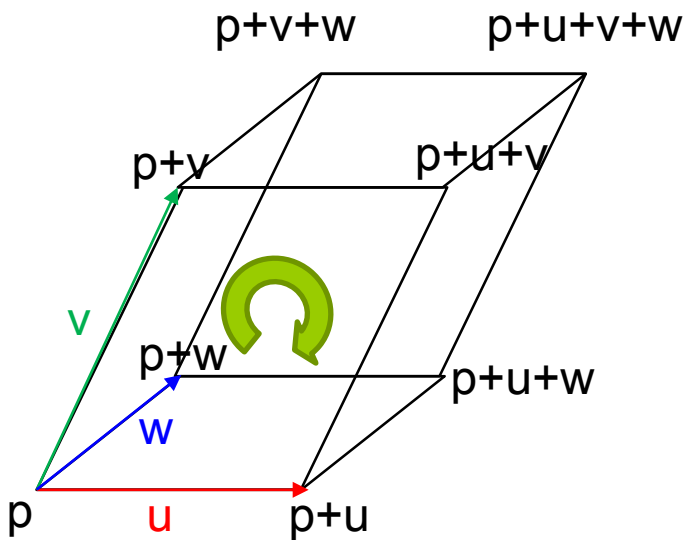
```
}
```

Quad



```
up.Normalize();  
n.Normalize();  
Vector3 left = Vector3.Cross(up, n);  
Vector3 uppercenter = (u * height/2.0f) + p;  
Vector3 tl = uppercenter + (left * width/2.0f);  
Vector3 tr = uppercenter - (left * width/2.0f);  
Vector3 bl = tl - (u * height);  
Vector3 br = tr - (u * height);  
GL.PushMatrix();  
GL.Begin(GL.QUADS);  
GL.Color(c);  
GL.Vertex(bl);  
GL.Vertex(tl);  
GL.Vertex(tr);  
GL.Vertex(br);  
GL.End();  
GL.PopMatrix();
```

Parallelepiped



Vector3 fll = p;

Vector3 flr = p + u;

Vector3 fur = p + u + v;

Vector3 ful = p + v;

Vector3 bll = p + w;

Vector3 blr = p + u + w;

Vector3 bur = p + u + v + w;

Vector3 bul = p + v + w;

// front face

GL.Vertex(fll);

GL.Vertex(ful);

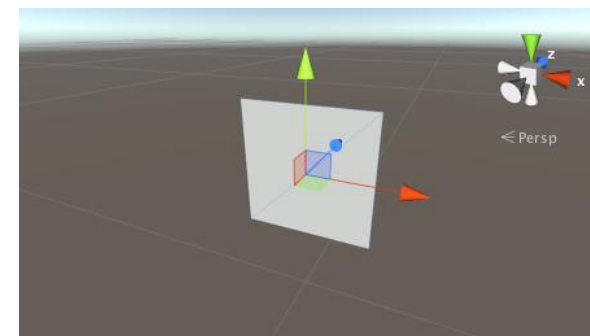
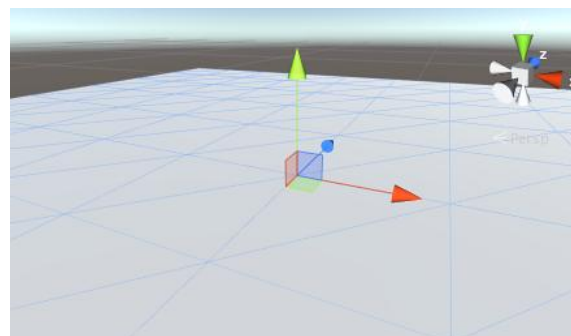
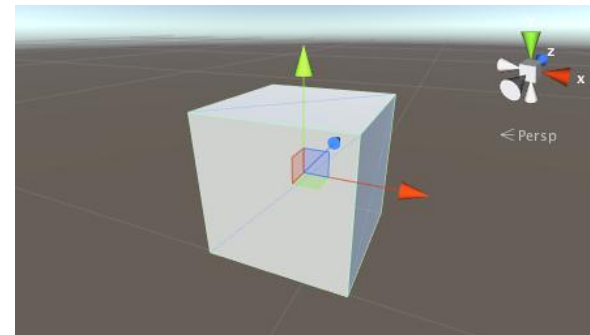
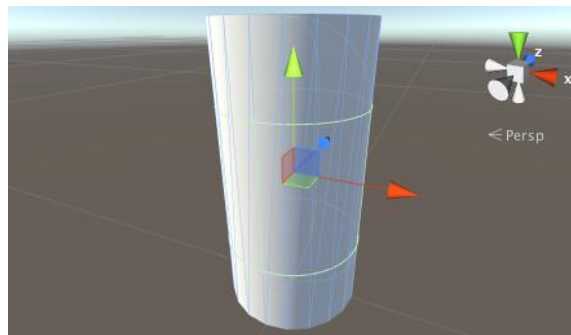
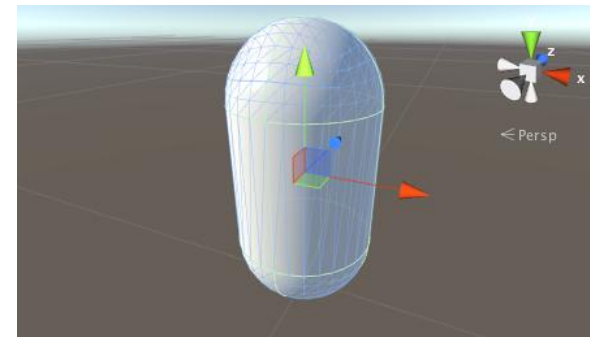
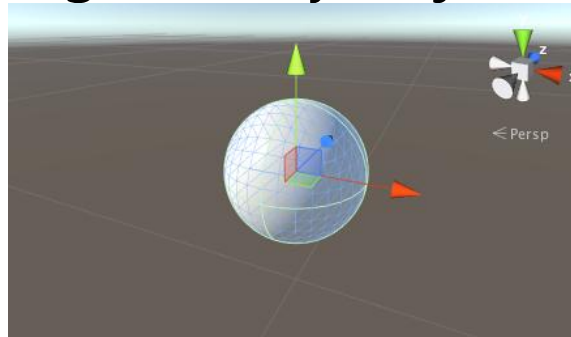
GL.Vertex(fur);

GL.Vertex(flr);

3D Geometry Object

□ Unity basic 3D geometry object

- Sphere = 0
- Capsule = 1
- Cylinder = 2
- Cube = 3
- Plane = 4
- Quad = 5



3D Geometry Object

```
public class Example : MonoBehaviour {  
    // Create a plane, sphere and cube in the Scene.  
    void Start() {  
        GameObject plane = GameObject.CreatePrimitive(PrimitiveType.Plane);  
  
        GameObject cube = GameObject.CreatePrimitive(PrimitiveType.Cube);  
        cube.transform.position = new Vector3(0, 0.5f, 0);  
  
        GameObject sphere = GameObject.CreatePrimitive(PrimitiveType.Sphere);  
        sphere.transform.position = new Vector3(0, 1.5f, 0);  
  
        GameObject capsule = GameObject.CreatePrimitive(PrimitiveType.Capsule);  
        capsule.transform.position = new Vector3(2, 1, 0);  
  
        GameObject cylinder = GameObject.CreatePrimitive(PrimitiveType.Cylinder);  
        cylinder.transform.position = new Vector3(-2, 1, 0);  
    }  
}
```

<https://docs.unity3d.com/ScriptReference/GameObject.CreatePrimitive.html>

3D Geometry Object

