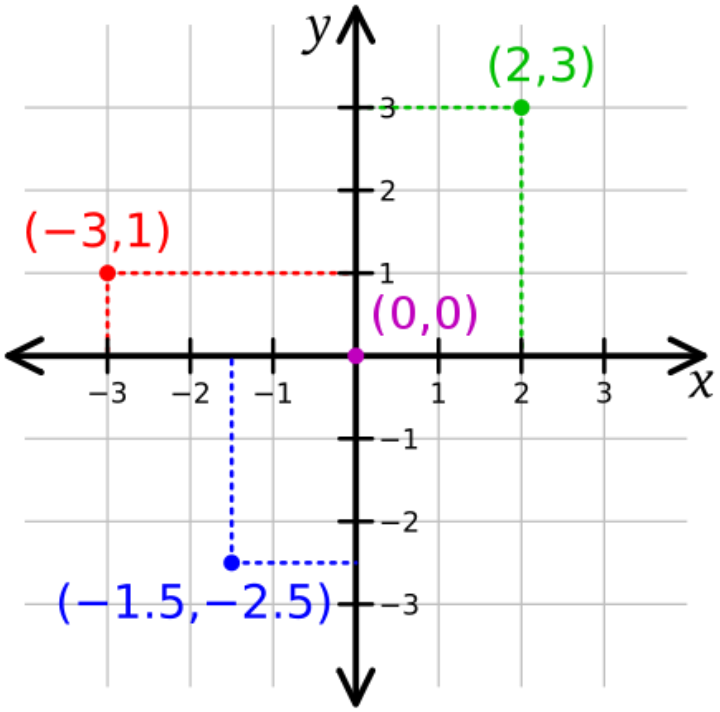
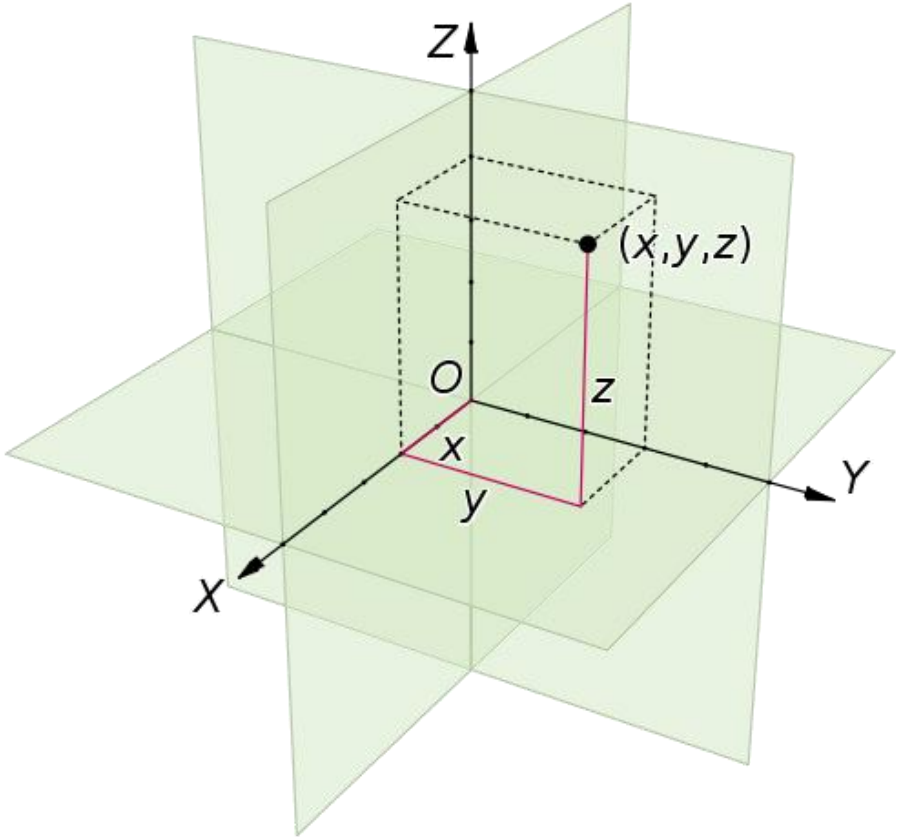
Graphics Programming

Fall 2024
9/19/2024
Kyoung Shin Park
Computer Engineering
Dankook University

Coordinate Systems

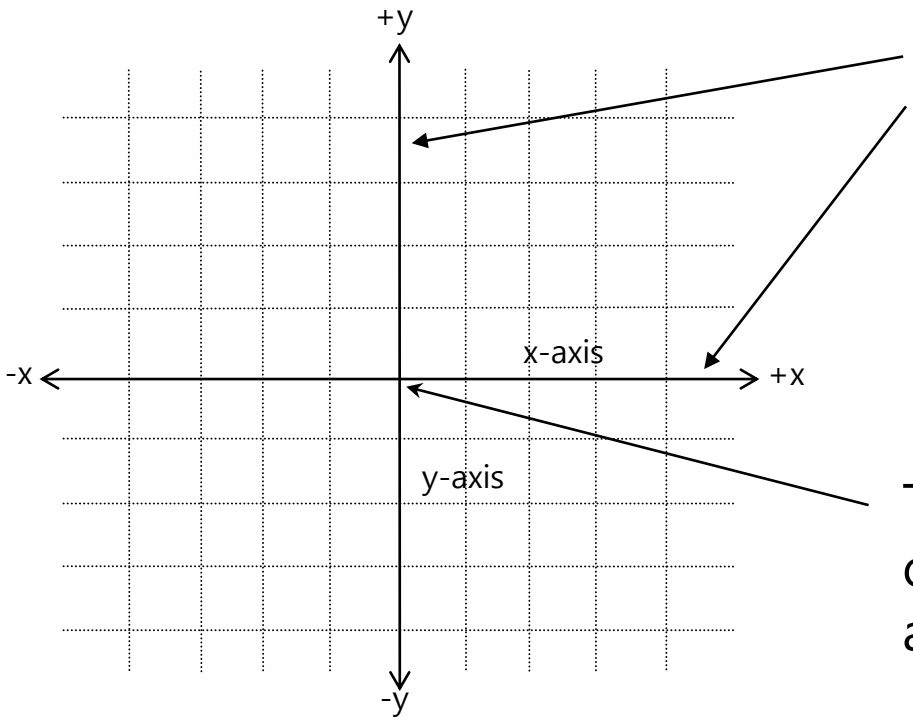
- 2D Cartesian Coordination Systems
- 3D Cartesian Coordination Systems





2D Cartesian Coordinate Systems

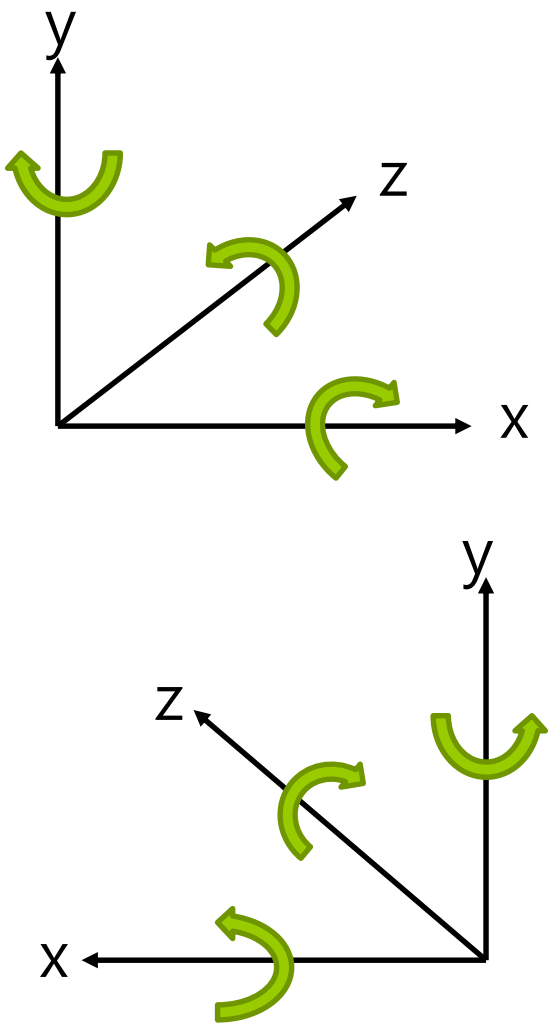
- Cartesian Coordination Systems



Two axes: **x-axis** and **y-axis**, two straight lines perpendicular to each other, both pass through origin and extends infinitely in two opposite directions

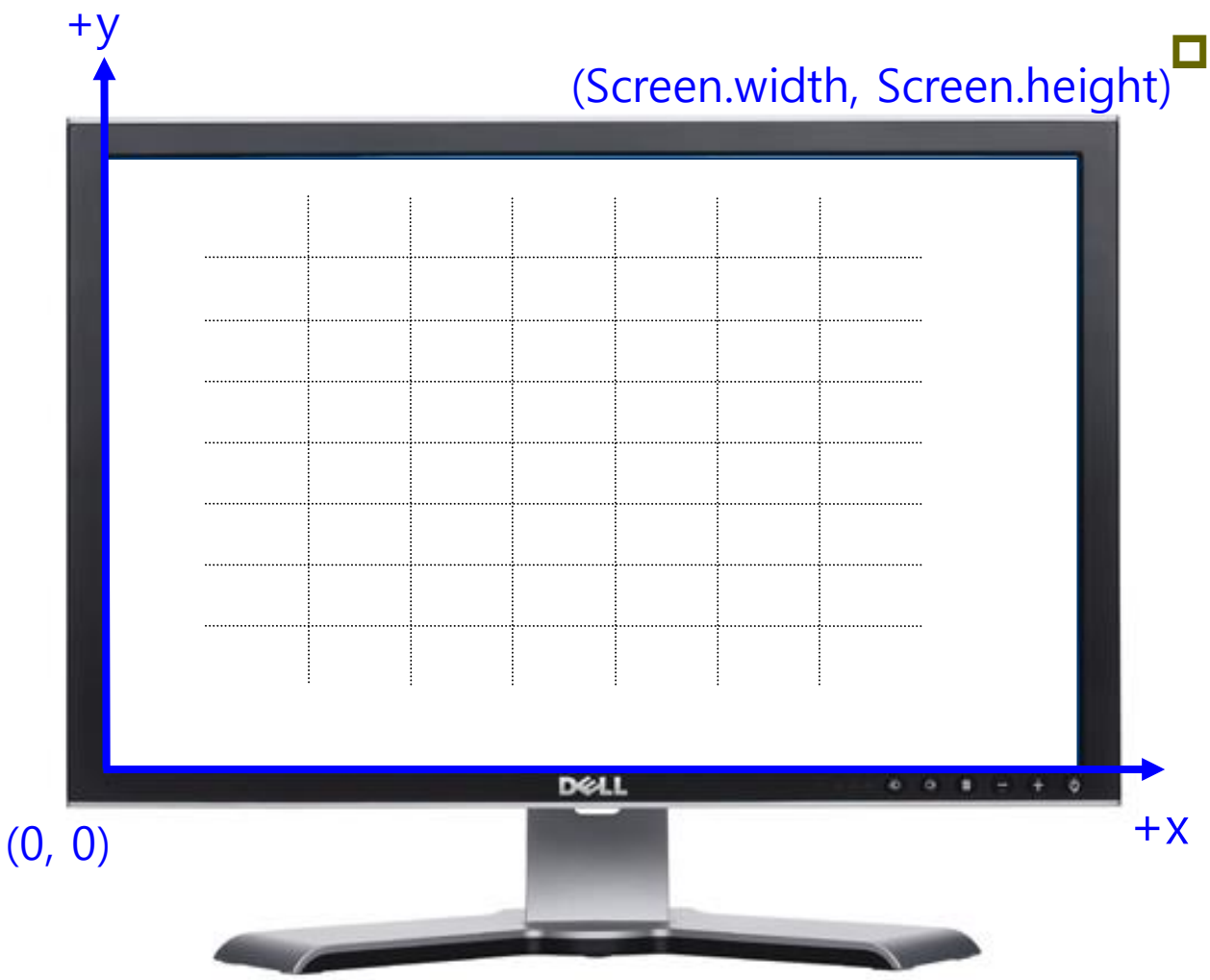
The origin is located in the center of the coordinate system and its value is (0, 0).

3D Cartesian Coordinate Systems



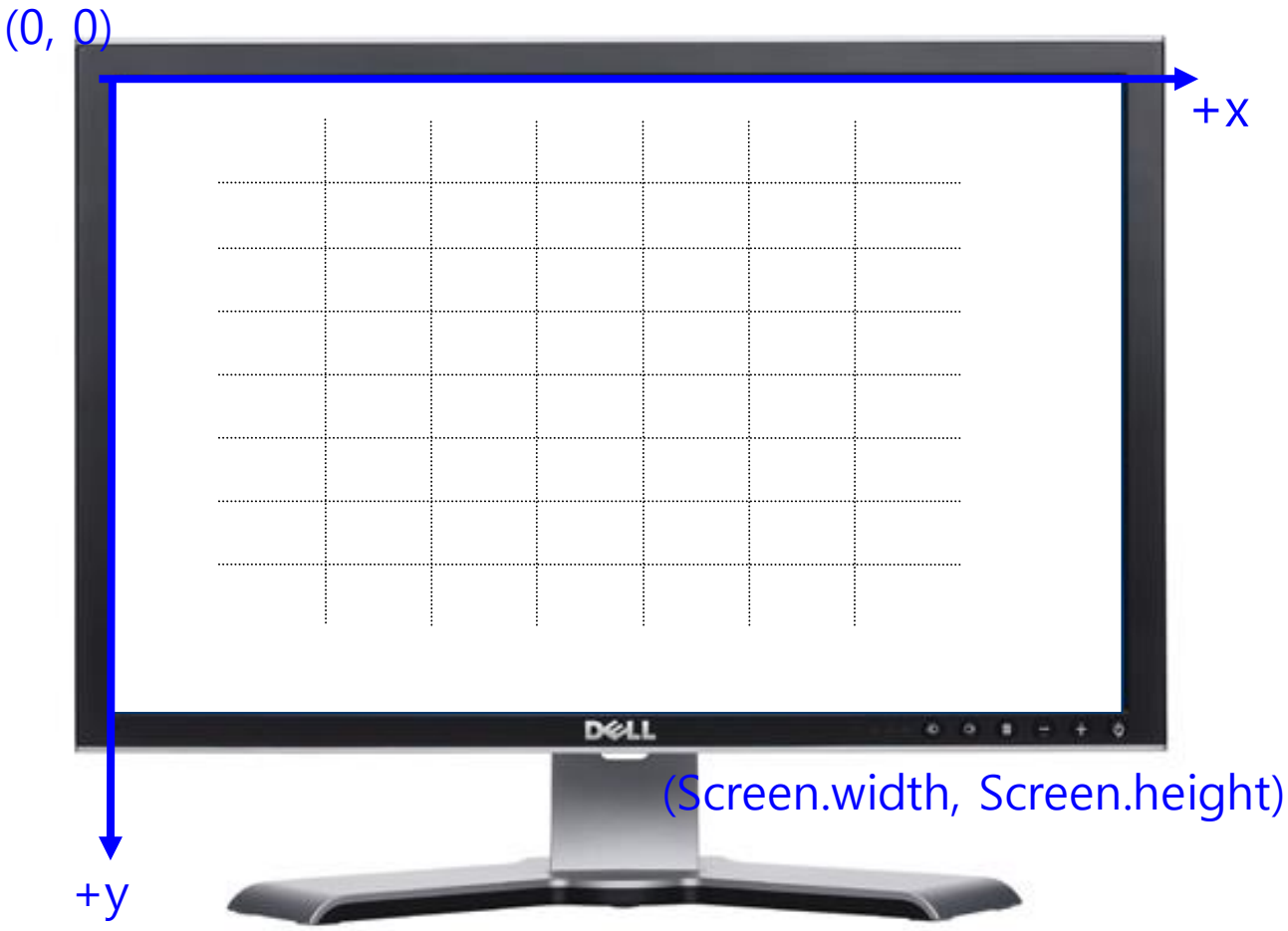
- In left-handed coordinate system, x+ is right, y+ is up, z+ is inside the screen.
- In right-handed coordinate system, x+ is left, y+ is up, z+ is inside the screen.

Screen Space



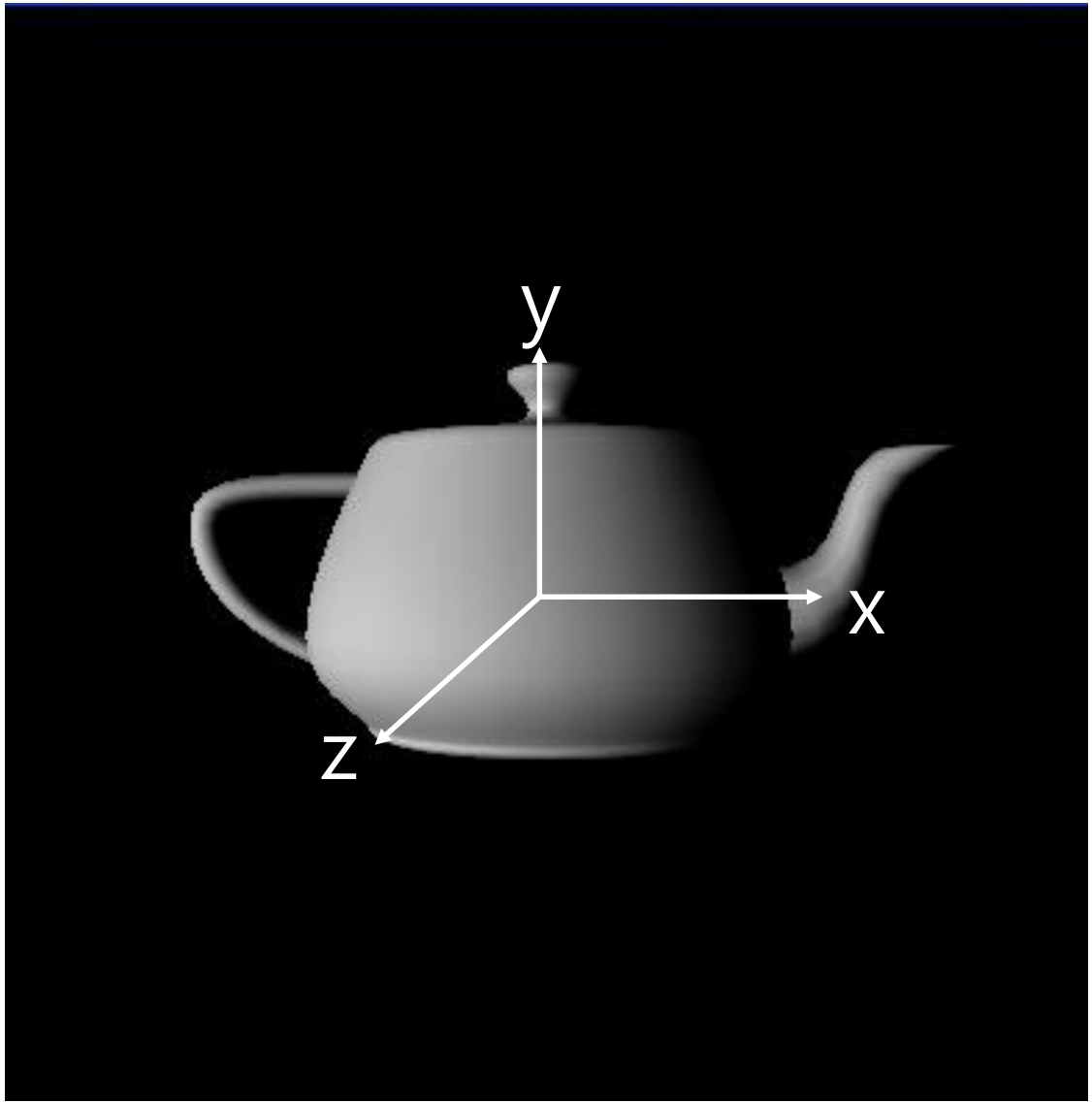
- In Unity, the screen coordinates are defined in pixels. The origin is located at the lower left corner of the screen and the value is (0, 0). x+ is right. y+ is up. The upper right corner is the (Screen.width, Screen.height).

GUI Space

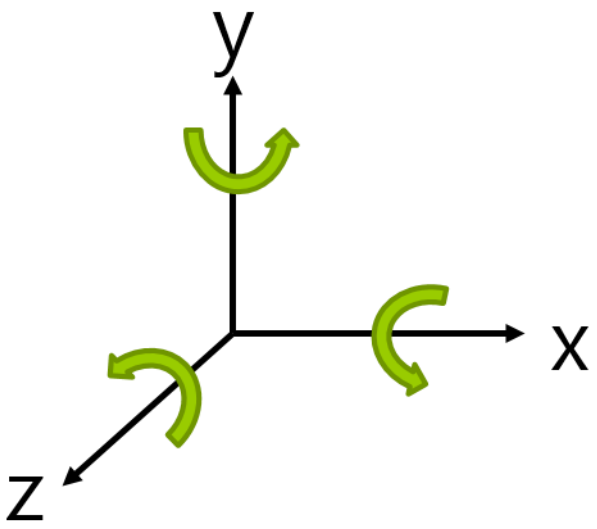


- In Unity, when drawing GUI on the screen, it uses a new coordinate system. The origin is located at the upper left corner of the screen (0, 0). x+ is right. y+ is down. The lower right corner is the (Screen.width, Screen.height).

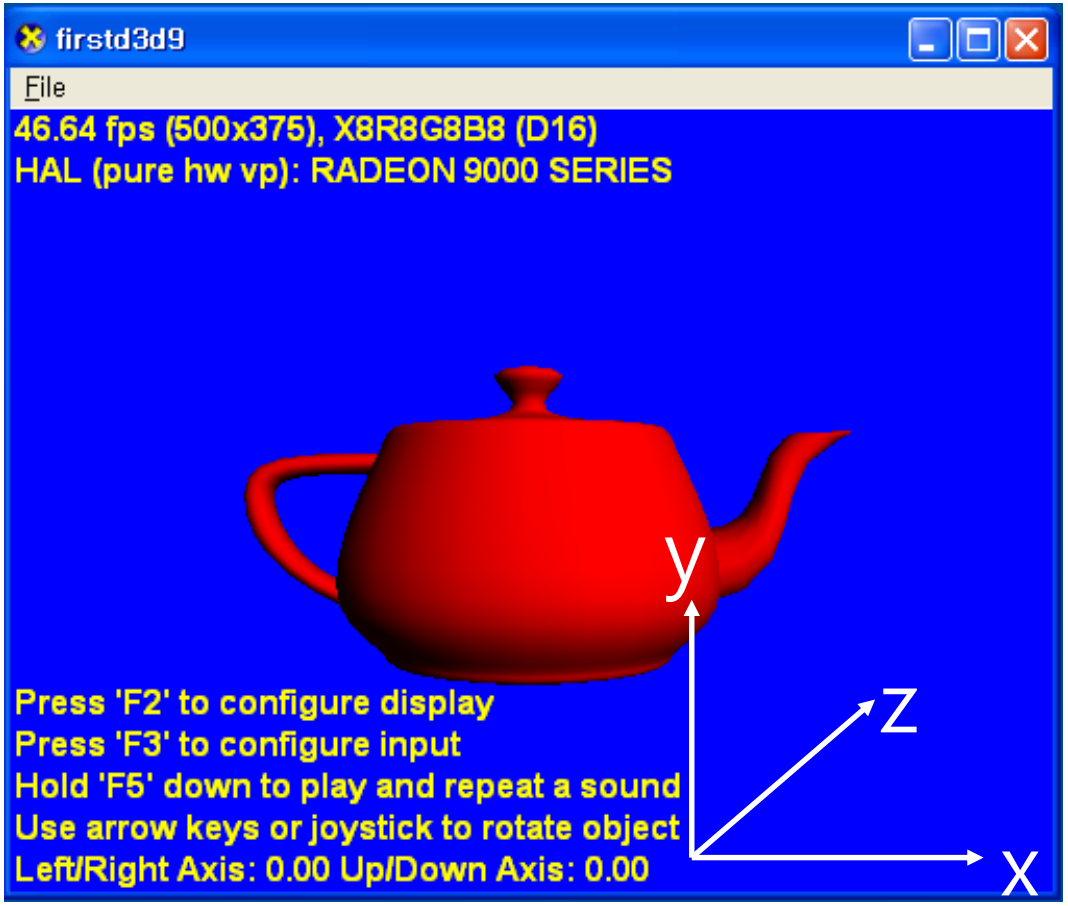
World Space – 3D Coordinate Systems



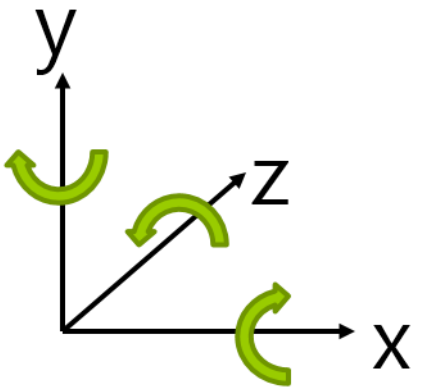
- **OpenGL** use a **right-handed** coordinate system
- x+ is right, y+ is up, z+ is out of the screen.



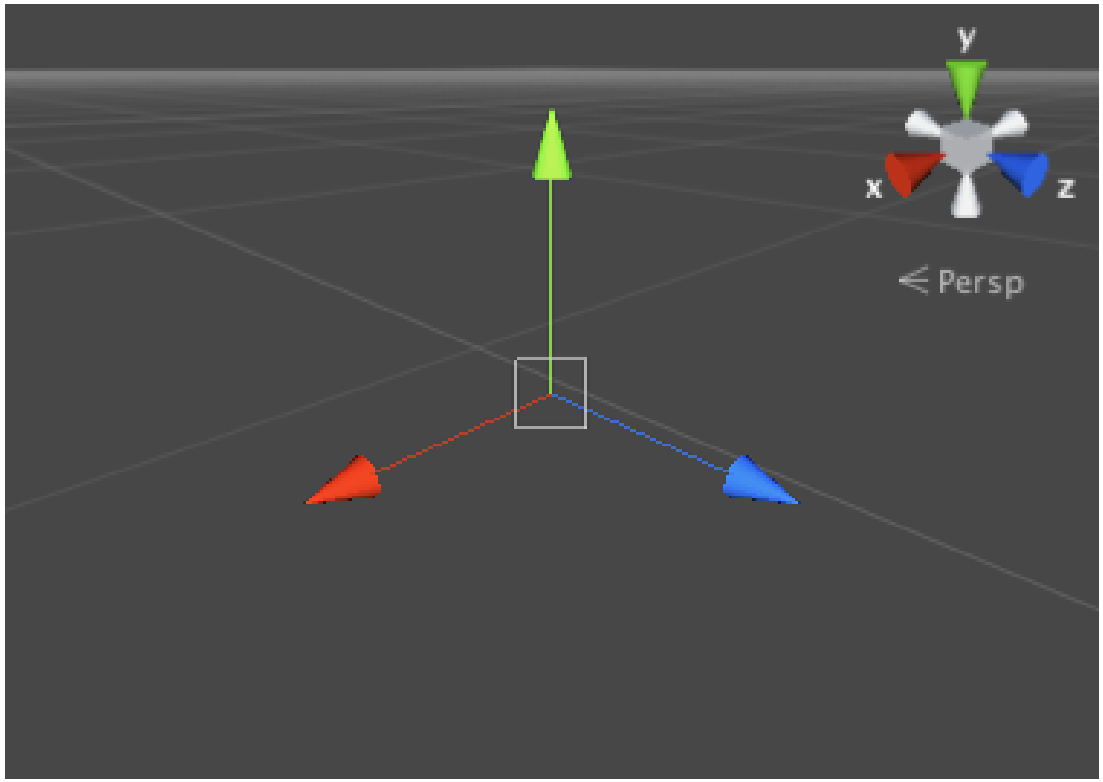
World Space – 3D Coordinate Systems



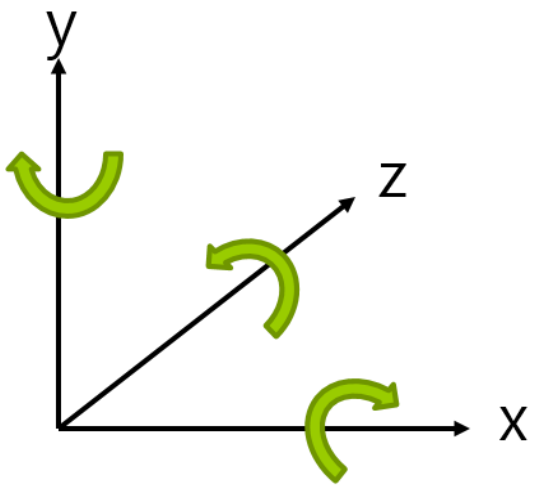
- **Direct3D** coordinate system is **left-handed**
- x+ to the right
- y+ up
- z+ forward



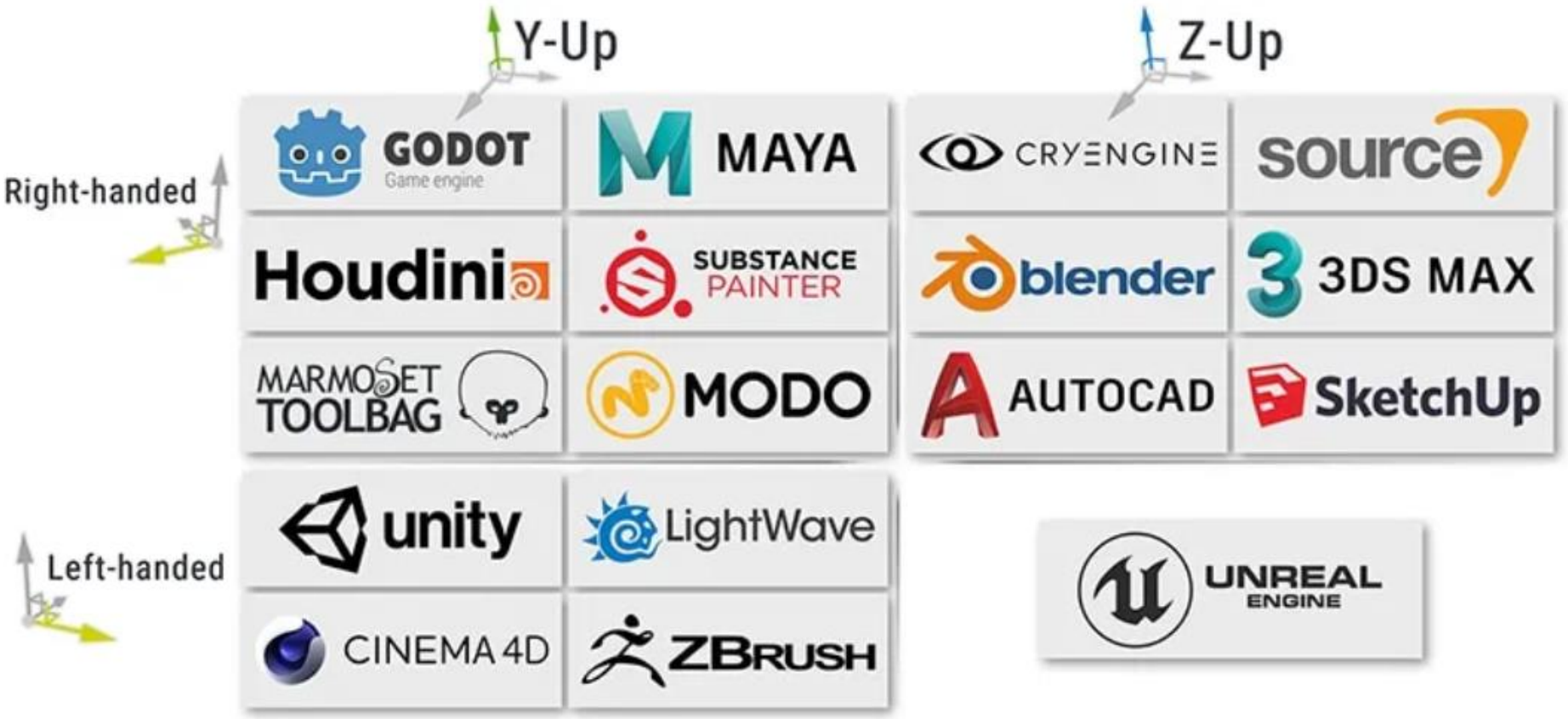
World Space – 3D Coordinate Systems



- **Unity3D** coordinate system is **left-handed**
- **x+ to the right**
- **y+ up**
- **z+ forward (inside the screen)**

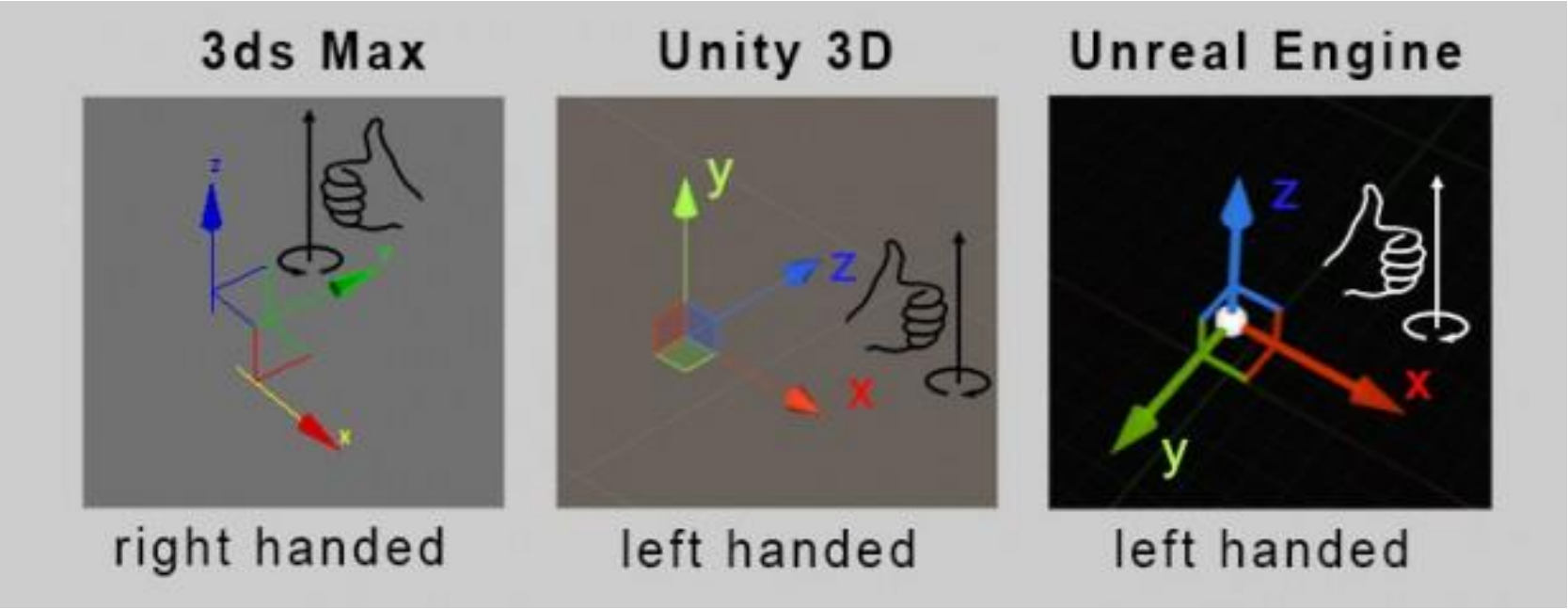


3D Coordinate Systems



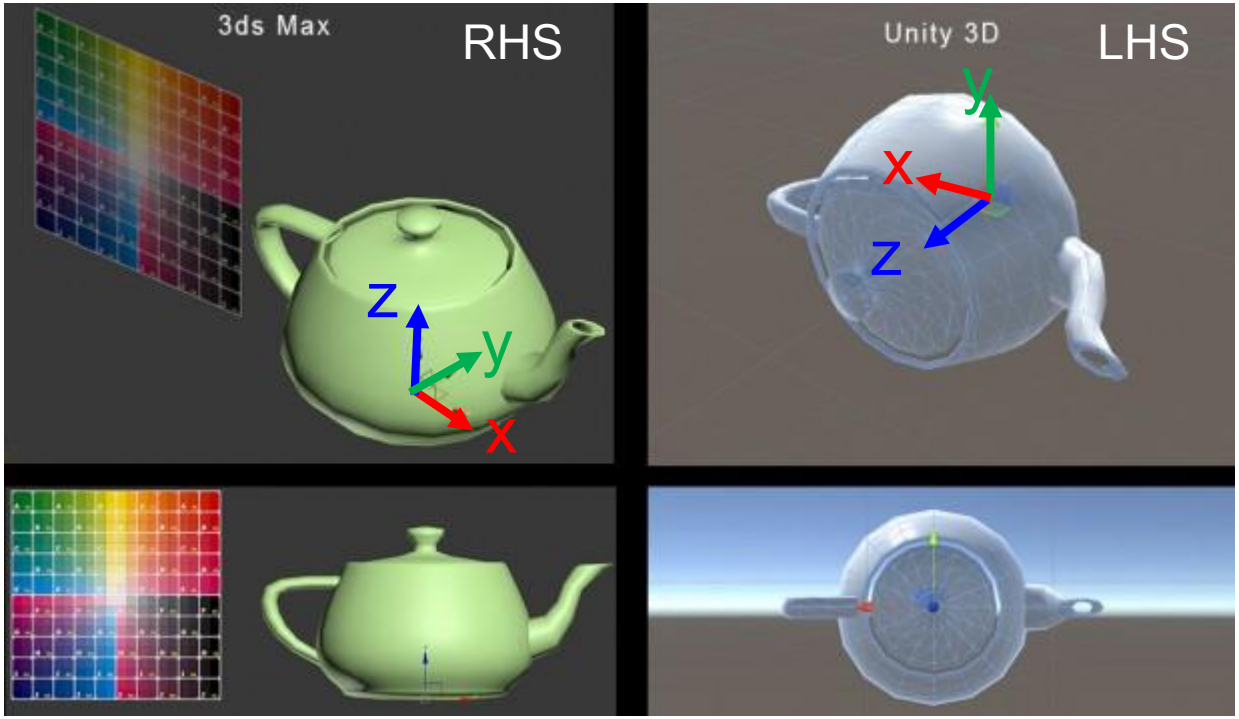
A Guide to Unity's Coordinate System (With Practical Examples)
https://www.techarthub.com/a-guide-to-unitys-coordinate-system-with-practical-examples/

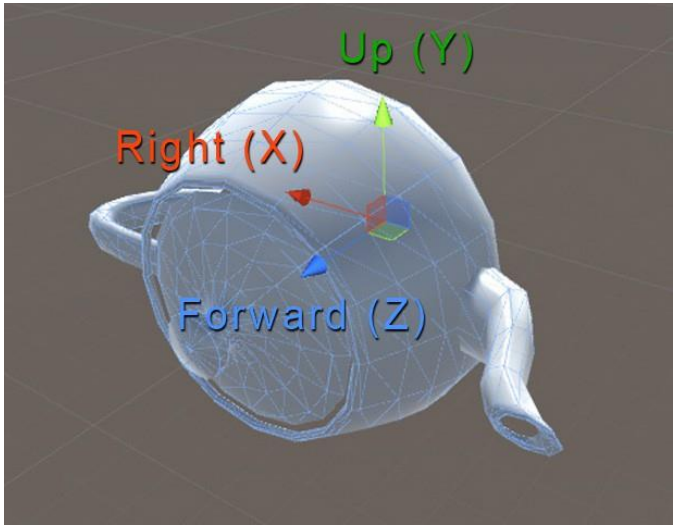
3D Coordinate Systems



World Coordinate Systems in 3ds Max, Unity and Unreal Engine
http://www.aclockworkberry.com/world-coordinate-systems-in-3ds-max-unity-and-unreal-engine/

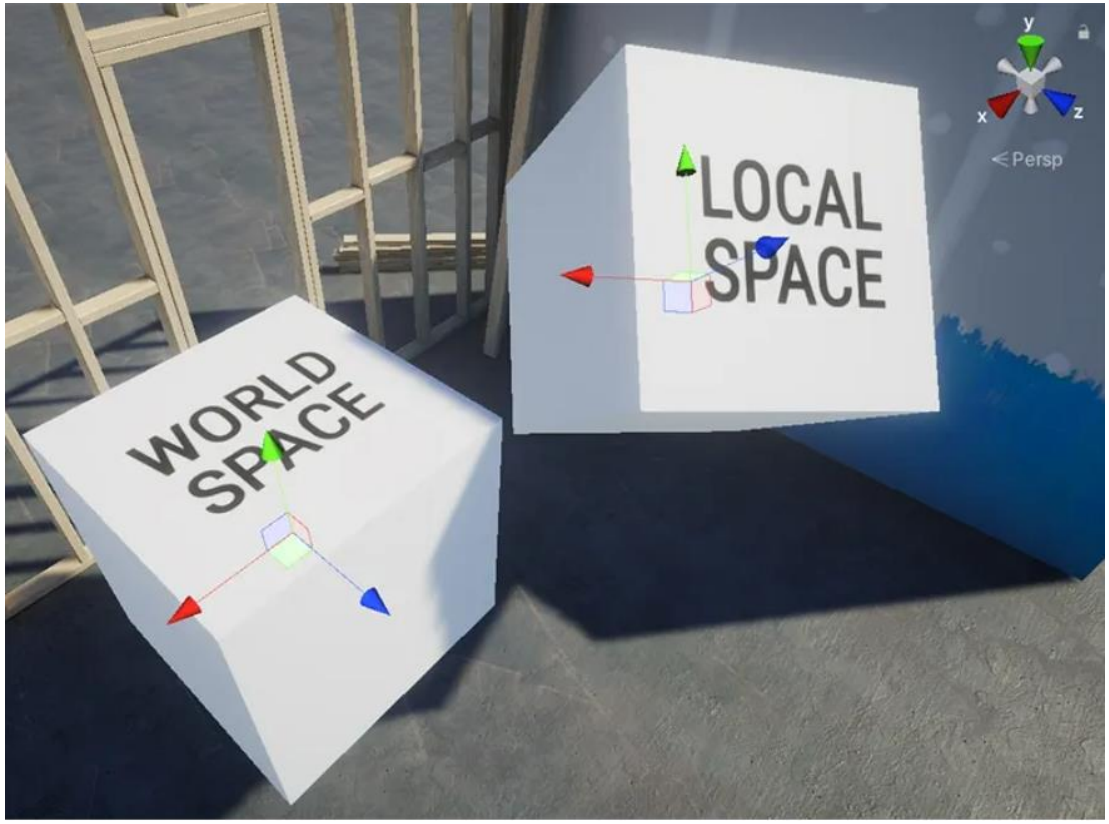
3D Coordinate Systems





World Coordinate Systems in 3ds Max, Unity and Unreal Engine
http://www.aclockworkberry.com/world-coordinate-systems-in-3ds-max-unity-and-unreal-engine/

World vs Local Space

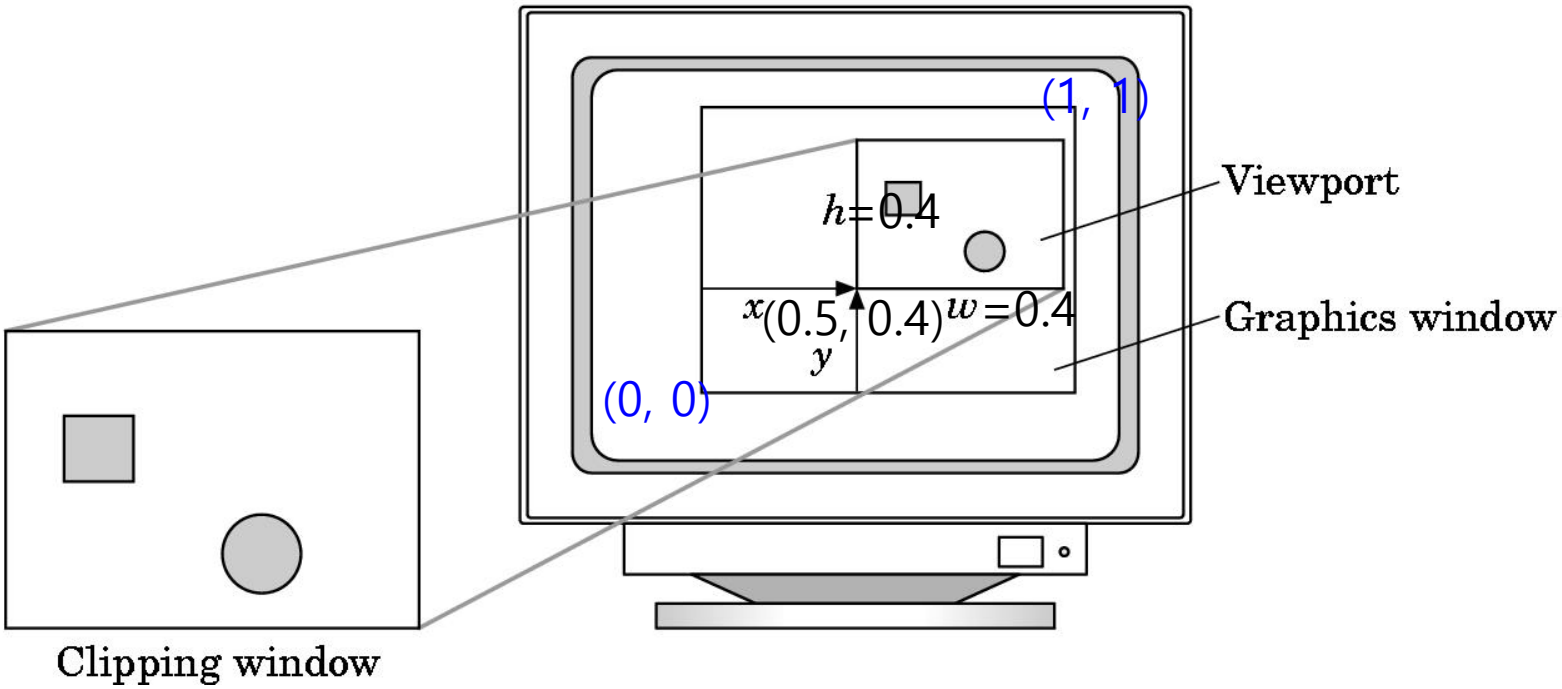


- **World space** is the coordinate system for the scene itself.
- **Local space** is a coordinate system that is relative to the rotation of a specific object.

https://www.techarthub.com/a-guide-to-unitys-coordinate-system-with-practical-examples/

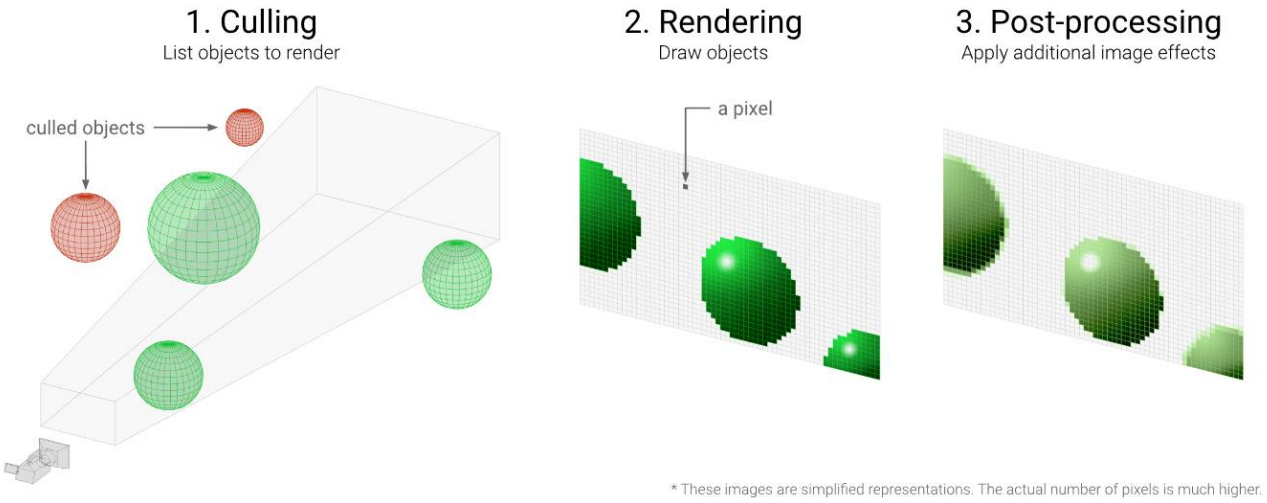
Viewport Space

- Viewport
  - The space set inside the window. Drawing is restricted to inside the viewport.
  - The viewport coordinates are relative to the camera. The lower left corner of the camera is the (0, 0) point, and the upper right corner is the (1, 1) point.



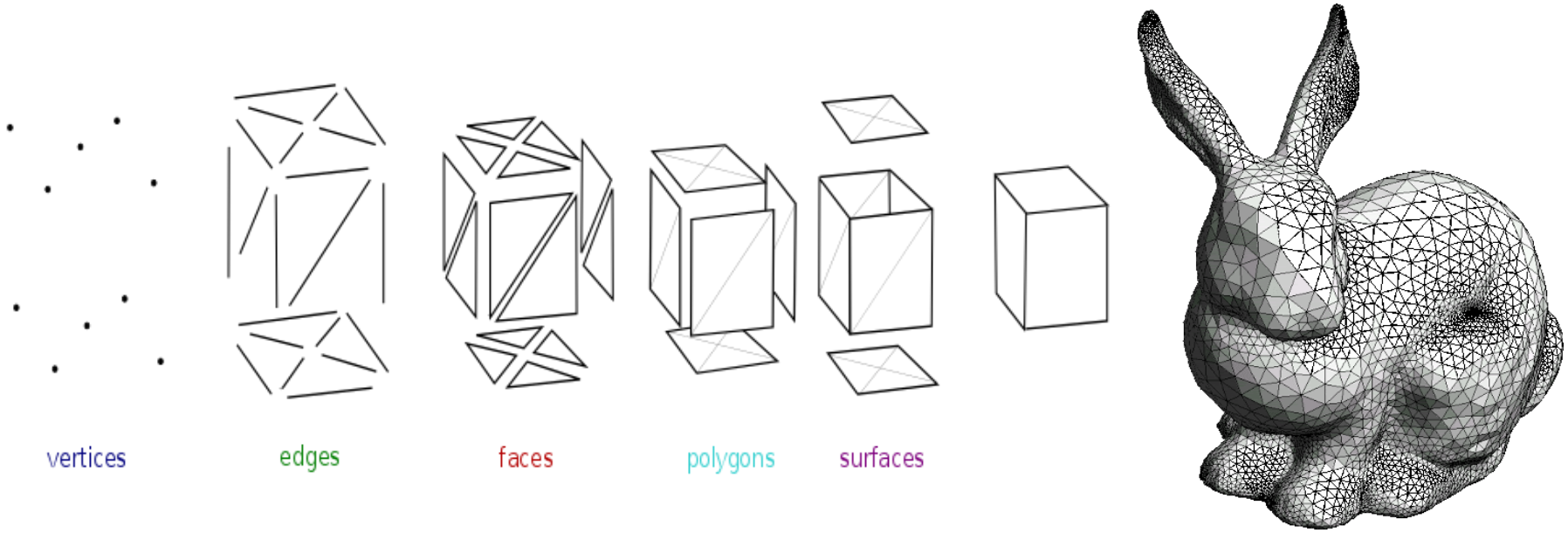
Rendering Pipeline

- The **rendering pipeline** performs a series of operations that take objects in the scene, and displays them on a screen.
  - Culling – frustum culling & occlusion culling
  - Rendering – drawing objects, with lighting, into pixel buffers
  - Post-processing – applying post-processing effects



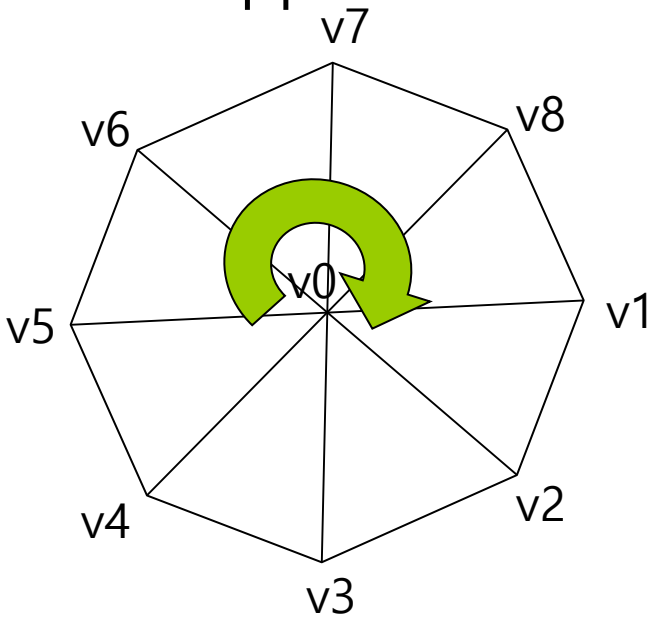
Geometry Mesh

- Creating 3d and 2d models using the **mesh** data.
- A model is represented as a triangle mesh approximation
- **Geometry** mesh data are collected (**Vertices Array, Normals Array, Triangle Array and UV Array**).



Vertex Buffer

- Circle approximation

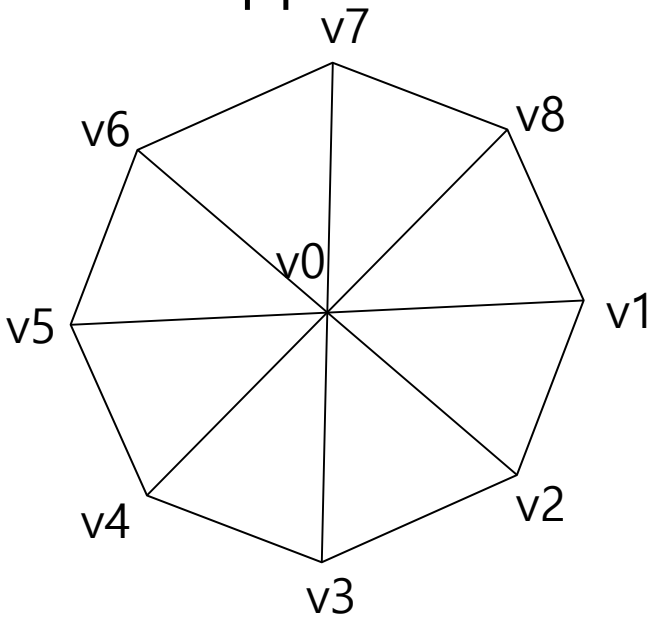


Clock-wise Winding (CW)

```
{ v0, v1, v2,      // triangle 0
  v0, v2, v3,      // triangle 1
  v0, v3, v4,      // triangle 2
  v0, v4, v5,      // triangle 3
  v0, v5, v6,      // triangle 4
  v0, v6, v7,      // triangle 5
  v0, v7, v8,      // triangle 6
  v0, v8, v1};     // triangle 7
```

Index Buffer

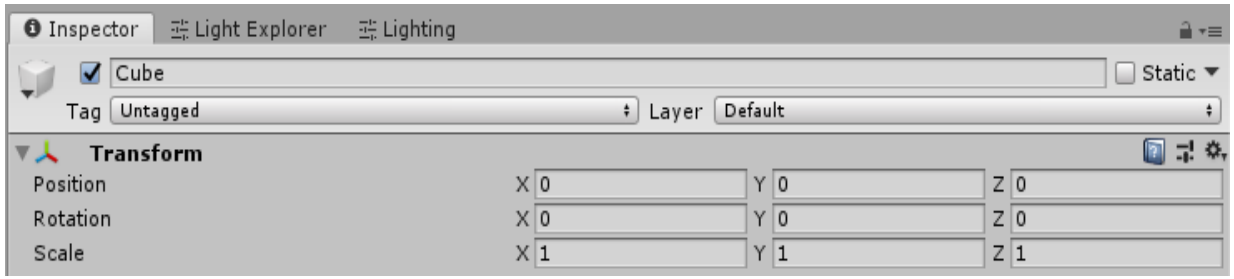
- Circle approximation

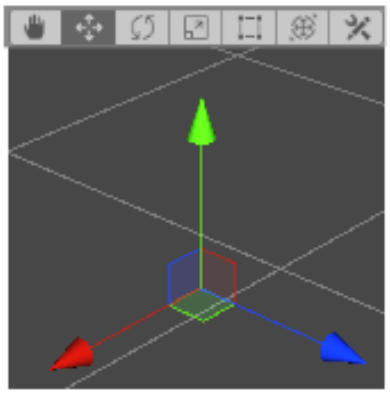



```
vertexList = {v0, v1, v2, v3, v4, v5, v6, v7, v8};
IndexList = { 0, 1, 2, // triangle 0
              0, 2, 3, // triangle 1
              ...
              0, 7, 8, // triangle 6
              0, 8, 1}; // triangle 7
```

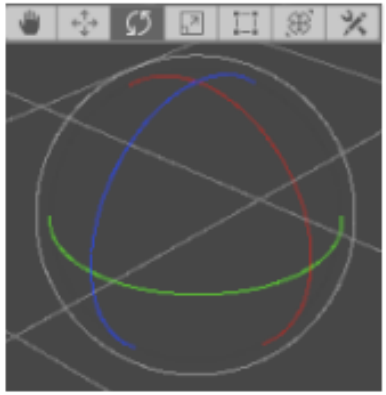
Transformations

- In Unity, every object in a Scene has a **Transform**. It's used to store and manipulate the **position**, **rotation** and **scale** of the object.

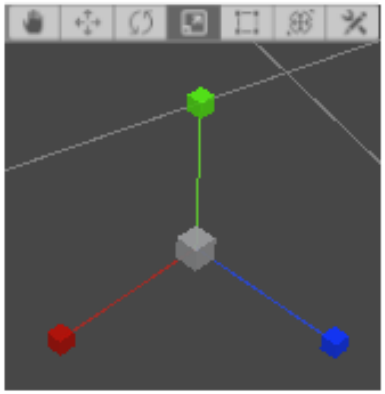




Translate (W)



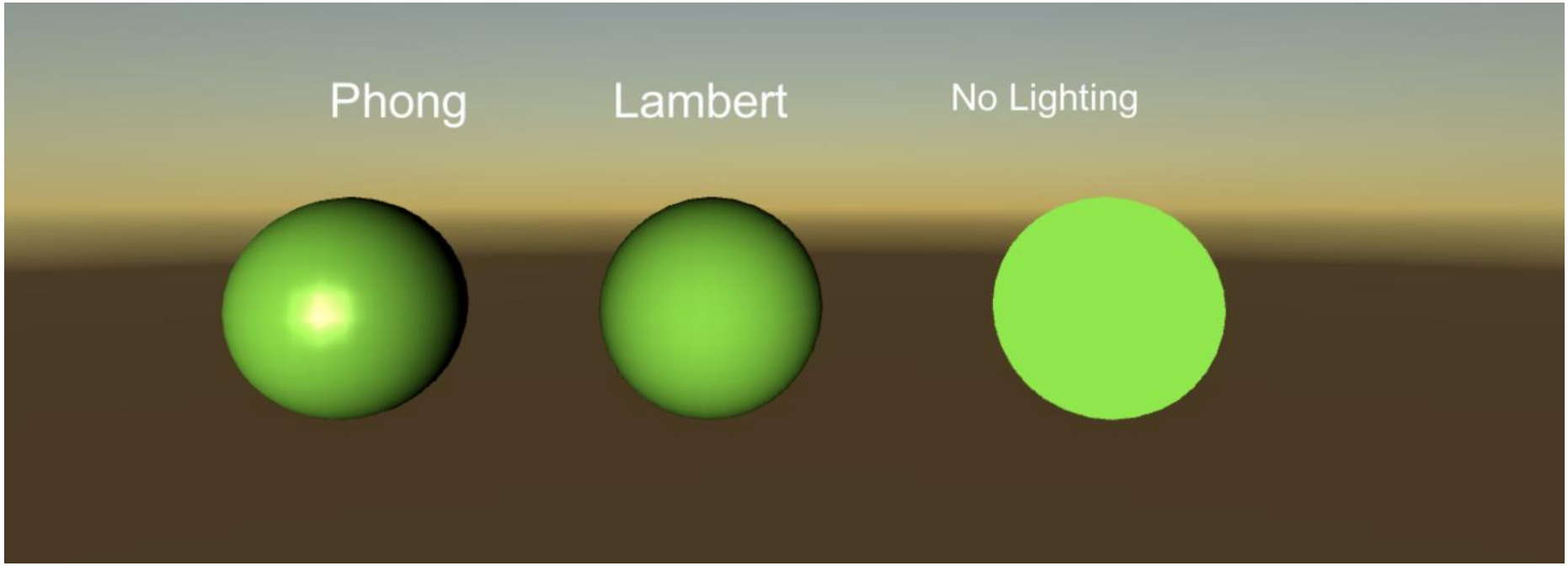
Rotate (E)



Scale (R)

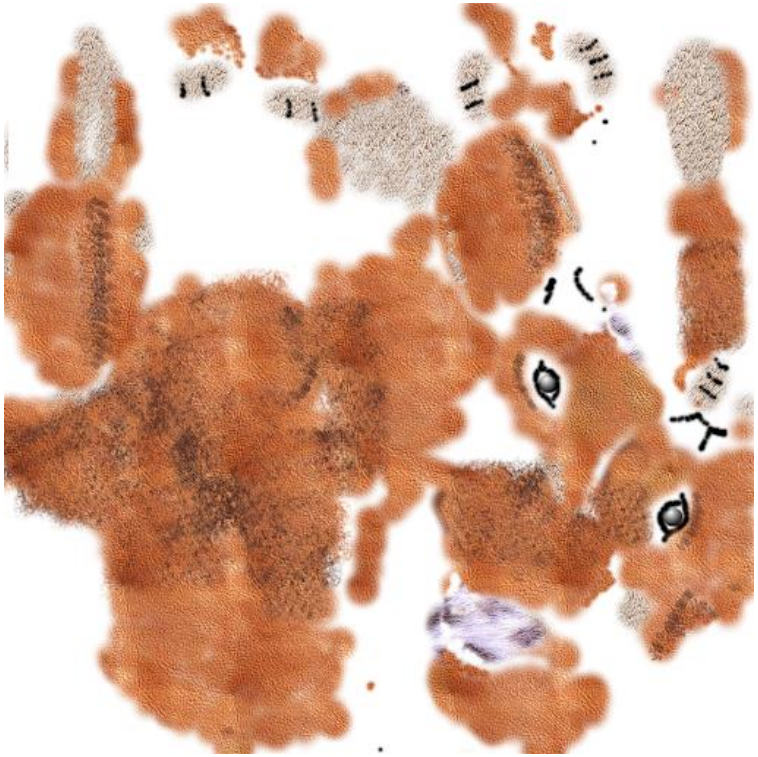
Geometry Lighting

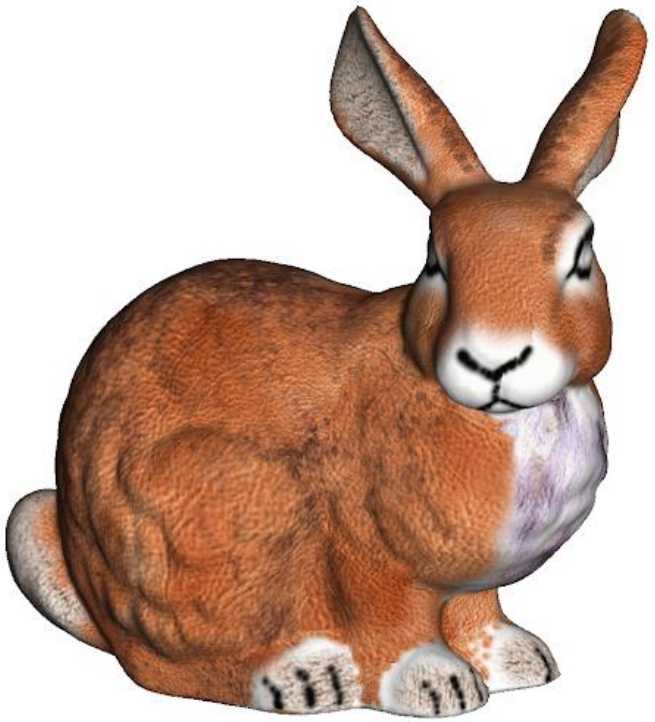
- In the illumination stage we add lighting effects to the scene



Textures

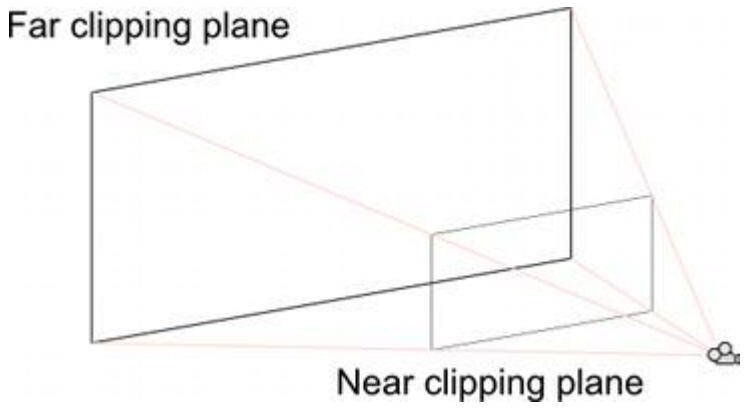
- Using different inputs(textures, normal maps … , etc.) we color objects in the scene.

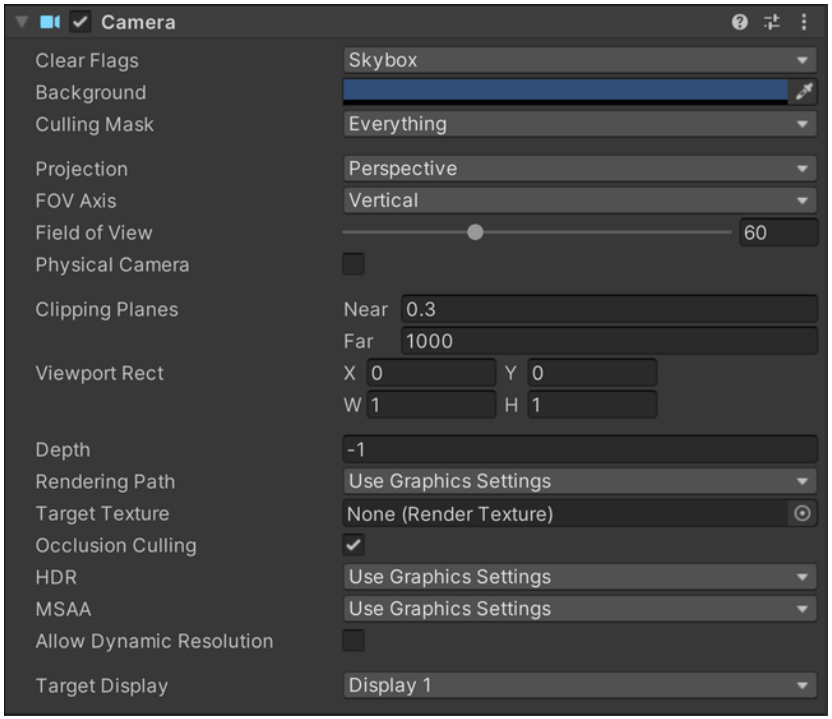




Camera

- In Unity, the camera is located at (0, 0, -10) world coordinate system and is point at the z+ direction.
- By default, a **perspective projection viewing frustum** is used.

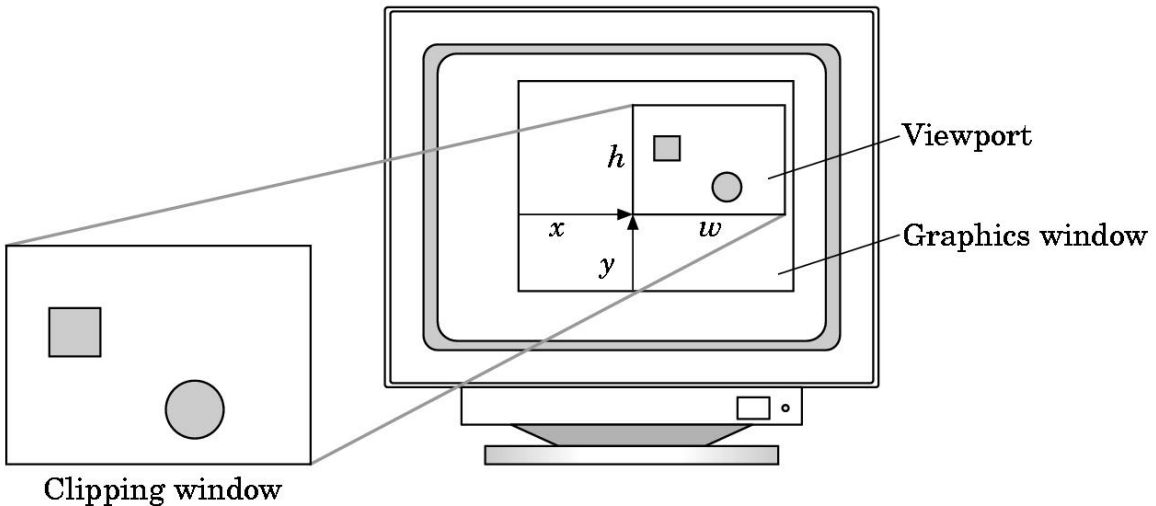




Viewport

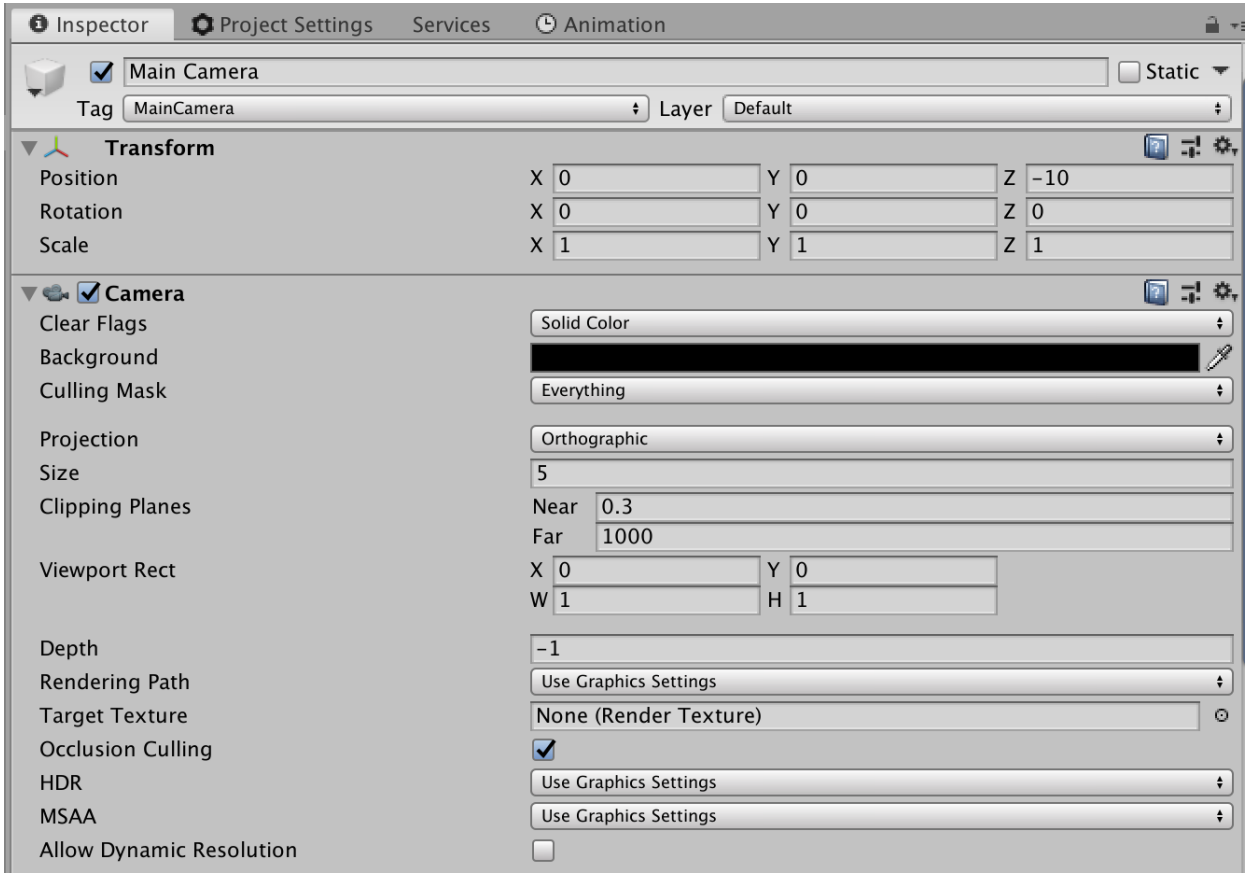
- Viewport
  - The space set inside the window. Drawing is restricted to inside the viewport.

Viewport Rect	Four values that indicate where on the screen this camera view will be drawn. Measured in Viewport Coordinates (values 0–1).
X	The beginning horizontal position that the camera view will be drawn.
Y	The beginning vertical position that the camera view will be drawn.
W (Width)	Width of the camera output on the screen.
H (Height)	Height of the camera output on the screen.



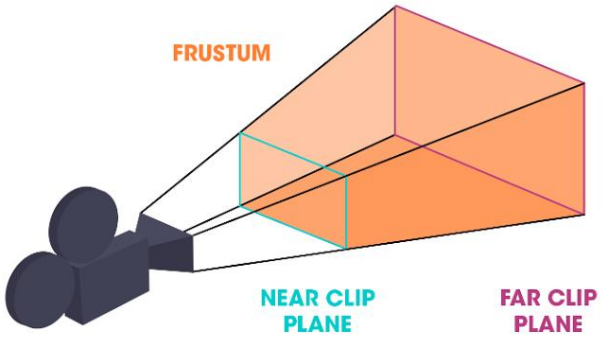
Viewer's Perspective (Camera Input)

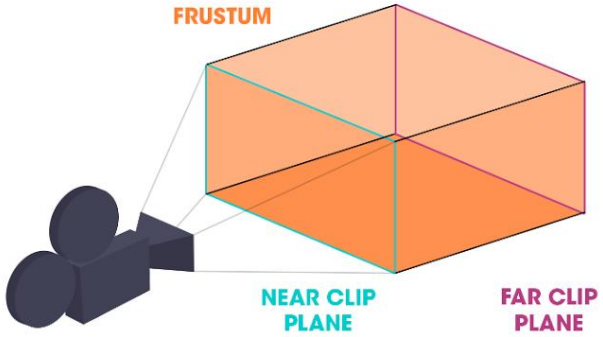
- Before rendering the environment on the screen we consider the camera input such as (**field of view, Projection Mode [Orthographic or Perspective]**).

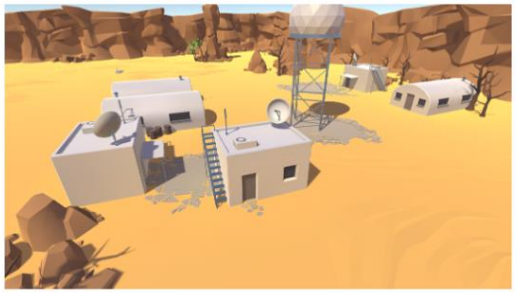


Orthographic vs Perspective Viewing

- **Orthographic parallel projection**
  - Points are projected onto the z=0 plane towards the z- axis.
- **Perspective projection**
  - it uses the y-direction viewing angle (FOV) and the aspect ratio (the value of the width of the nearest clipping plane divided by the height)



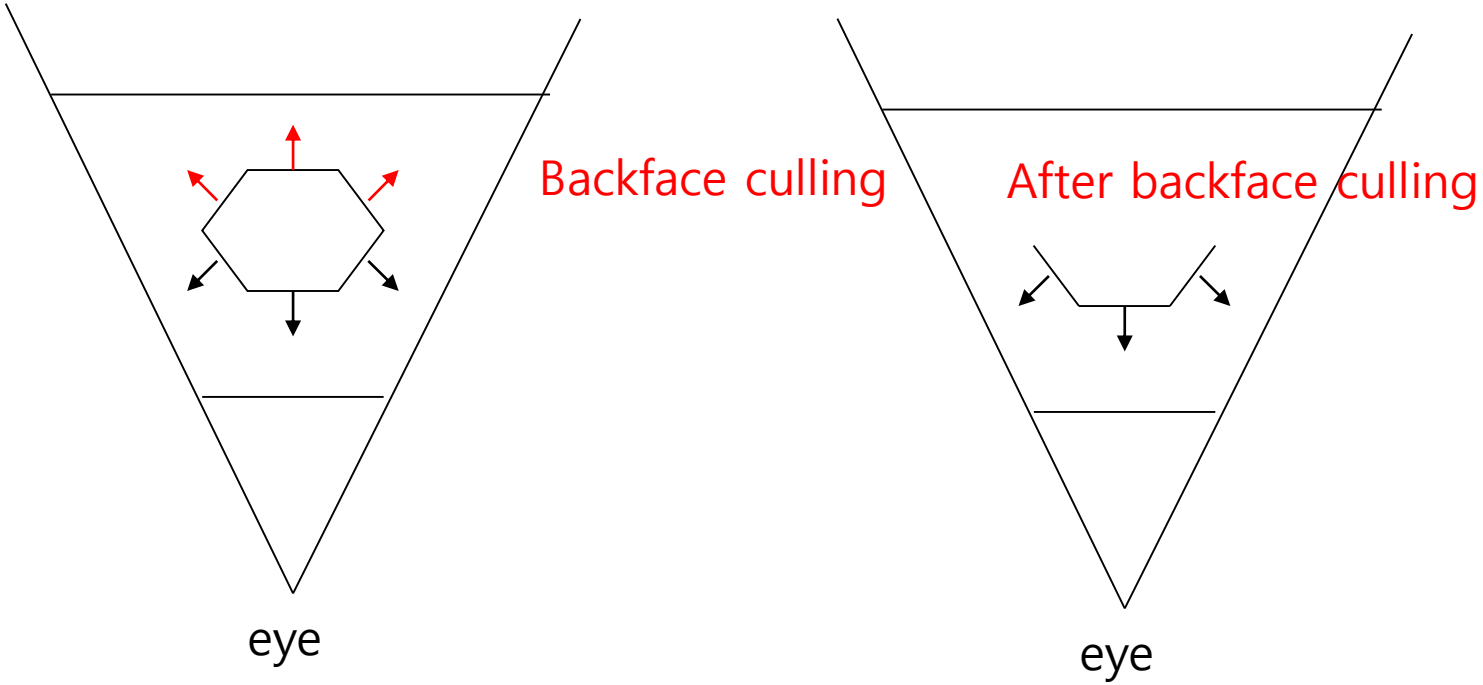




Backface culling

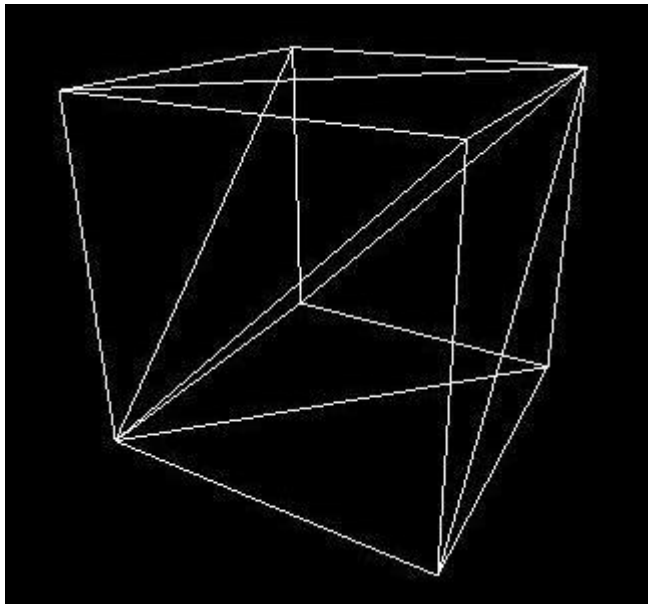
- Backface culling
  - A polygon has the front face and the back face.
  - Backface culling can quickly discard about half of the scene's dataset from further processing – an excellent speed up.
- Determine which polygons are front facing or back facing
  - By default, triangles with clockwise winding order are front facing
  - Visibility test: planeNormal • viewVector > 0
- Set culling
  - RasterizerState.CullMode = CullMode.None;
  - Value
    - NONE: disable backface culling
    - CW: triangles with a clockwise winding are culled
    - CCW: triangles with a counterclockwise winding are culled (default)

Backface culling

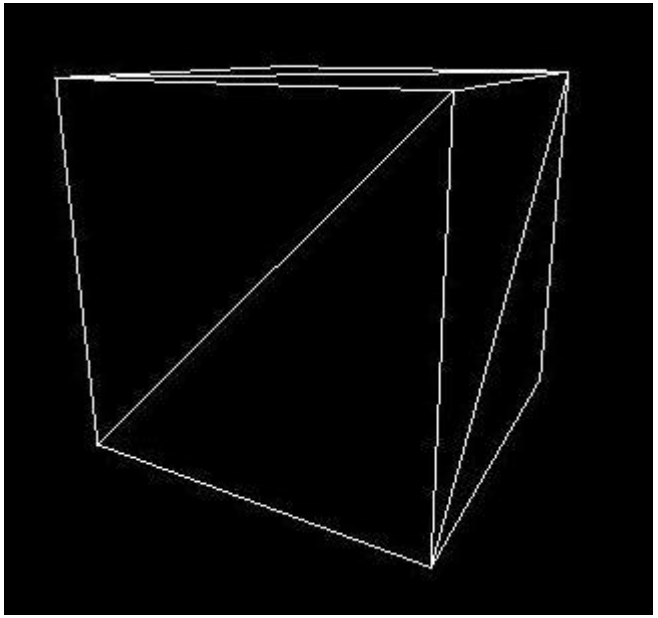


Backface culling

No Culling (All faces are seen)

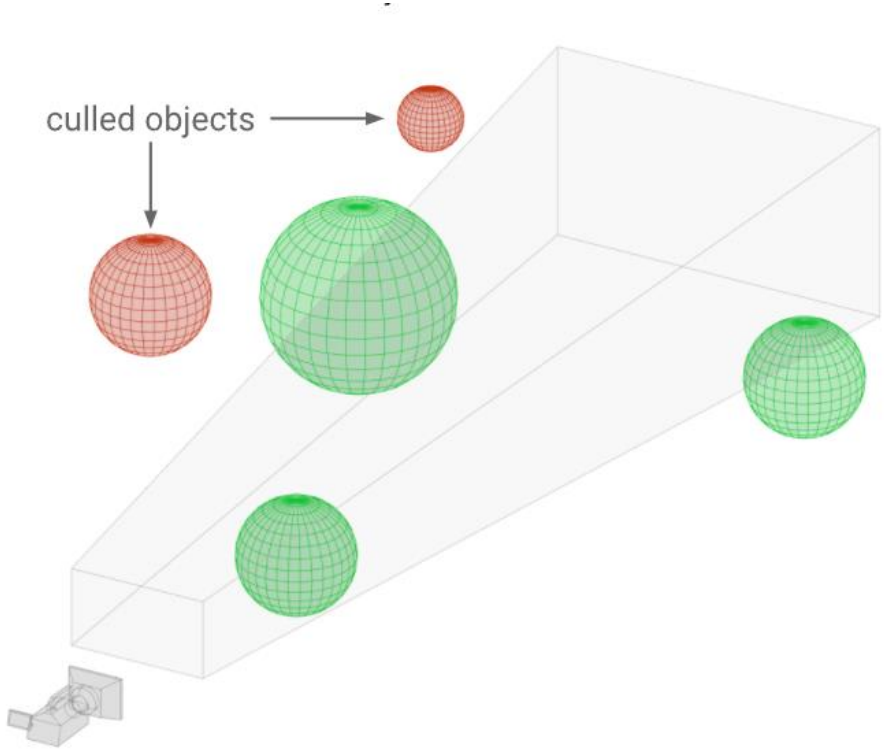


Backface Culling



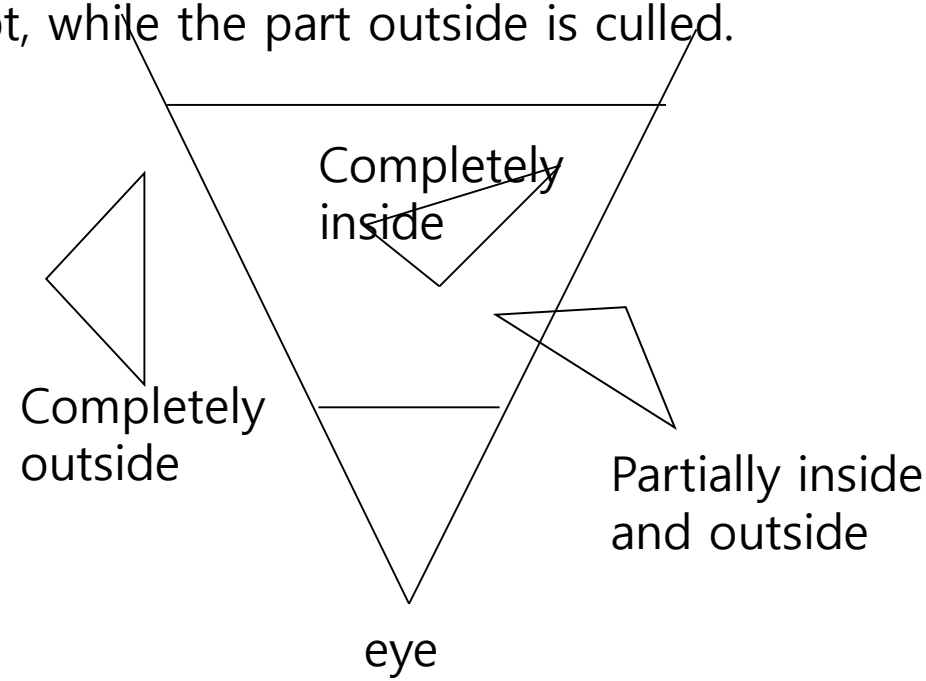
Clipping

- **Objects projected outside the window are clipped** without appearing as an image by placing a pyramid like clipping volume in front of the camera.



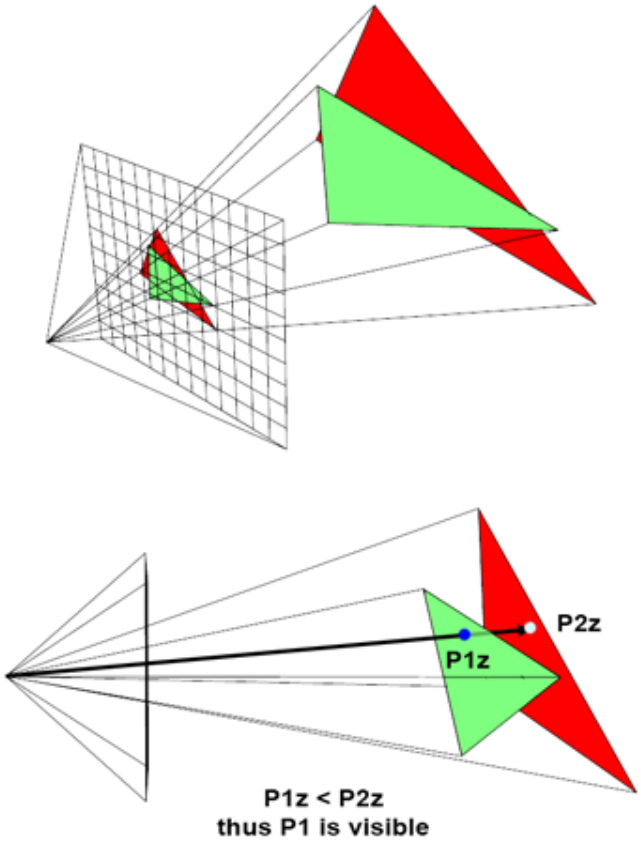
Clipping

- Clipping
  - Clipping culls the geometry that is outside the viewing volume
  - 3 possible locations of triangle in the frustum:
    - Completely inside: it is kept
    - Completely outside: it is culled
    - Partially inside: then, the triangle is split into two parts. The part inside the frustum is kept, while the part outside is culled.



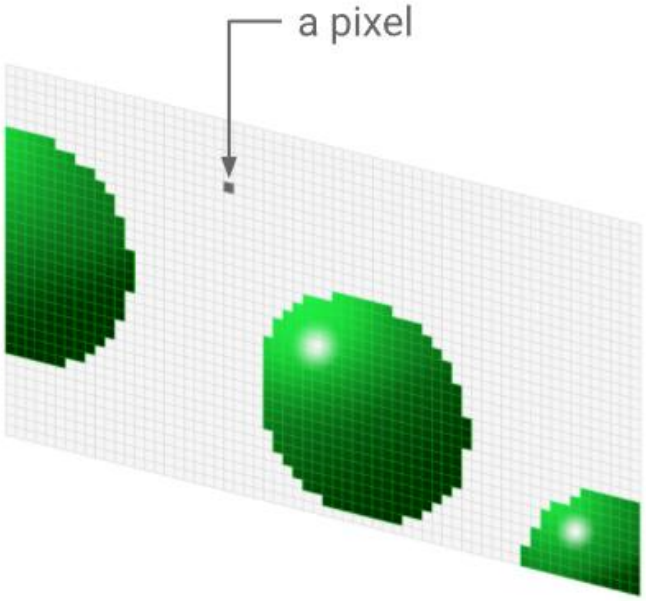
Projection

- **Projection** determines which point on the 2D screen is a point in the 3D space that constitutes an object when the observer composes the composition.



Rasterization

- Since our screens are 2D, **rasterization** is how the Geometry (3D & 2D) will be drawn on our 2D screen.
- Rasterization is where we process the scene several times through different filters then output the result on the screen.



Post-Processing

- Post-processing effects we add to the 2D image just before displaying the final output on the screen.

Depth of field is a post-processing effect applied to the 2D final image

Unity GL Class

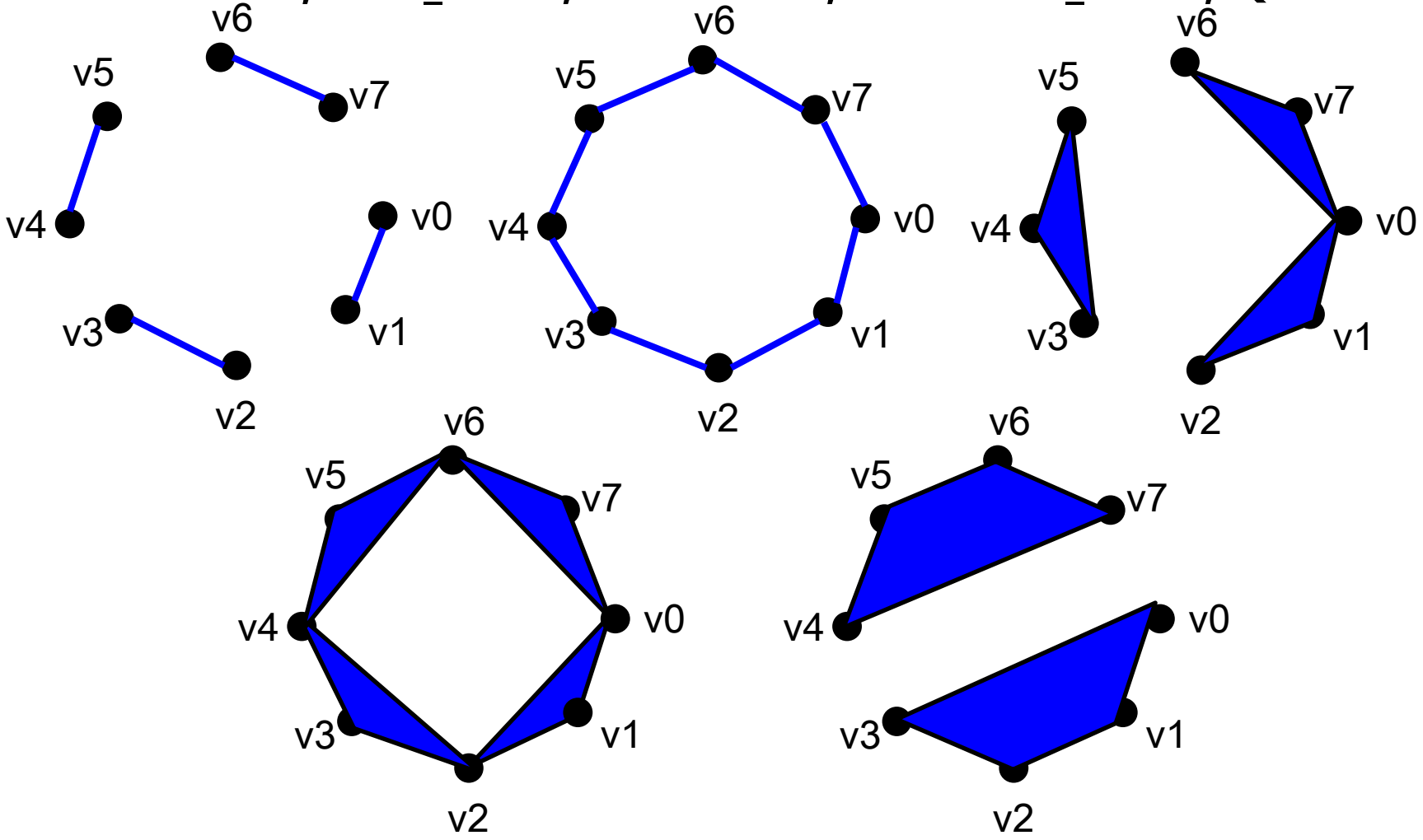
- Low-level graphics library.
- Use this class to manipulate active transformation matrices, issue rendering commands similar to OpenGL's immediate mode and do other low-level graphics tasks.
- GL immediate drawing functions use whatever is the "current material" set up right now.
- The usual place to call GL drawing is most often in **OnPostRender() from a script attached to a camera**, or inside an image effect function (**OnRenderImage**)

Unity GL Geometry Primitives

- In real-time graphics, linear primitives are mainly used, which is the simplest form of graphics expression.
  - Point, vertex
  - Line segments
  - Polygon
  - Polyhedron

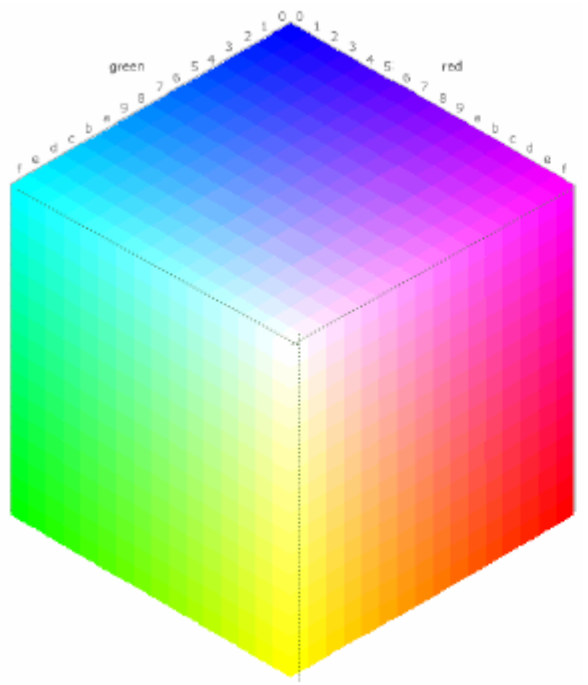
Unity GL Geometry Primitives

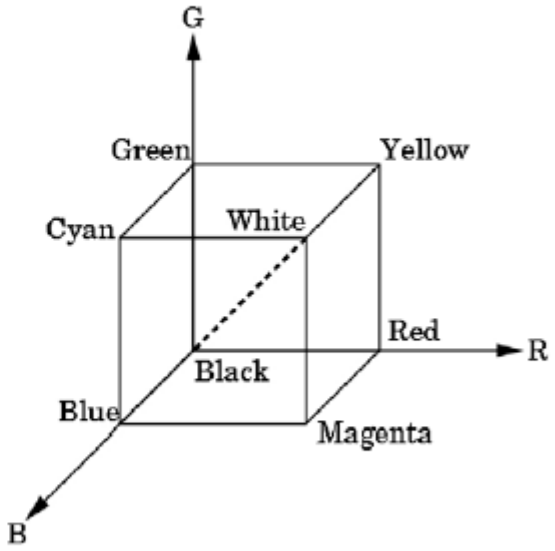
- GL.Begin(mode)
  - LINES, LINE_STRIP, TRIANGLES, TRIANGLE_STRIP, QUADS



Unity Color

- **GL.Color(color)**
  - RGB (Red, Green, Blue) or RGBA(Red, Green, Blue, Alpha)
  - RGB colors are separated and stored in the framebuffer.





Color Triangle

```
public class DrawFilledTriangle : MonoBehaviour {
    public Material mat = null;
    public Vector3 vertex1 = new Vector3(-1, -1, 0);
    public Vector3 vertex2 = new Vector3(1, -1, 0);
    public Vector3 vertex3 = new Vector3(1, 1, 0);
    // will be called after all regular rendering is done
    public void OnPostRender() {
        CreateMaterial(); // 중간 생략
        mat.SetPass(0);
        GL.PushMatrix();
        GL.MultMatrix(transform.localToWorldMatrix);
        GL.Begin(GL.TRIANGLES); // LHS CW winding order
        GL.Color(Color.red);
        GL.Vertex(vertex1);
        GL.Color(Color.blue);
        GL.Vertex(vertex3);
        GL.Color(Color.green);
        GL.Vertex(vertex2);
        GL.End();
        GL.PopMatrix();
    }
}
```