유니티(Unity)를 활용한

그래픽스 프로그래밍

06 Rendering Pipeline Geometry

1

3D Coordinate System

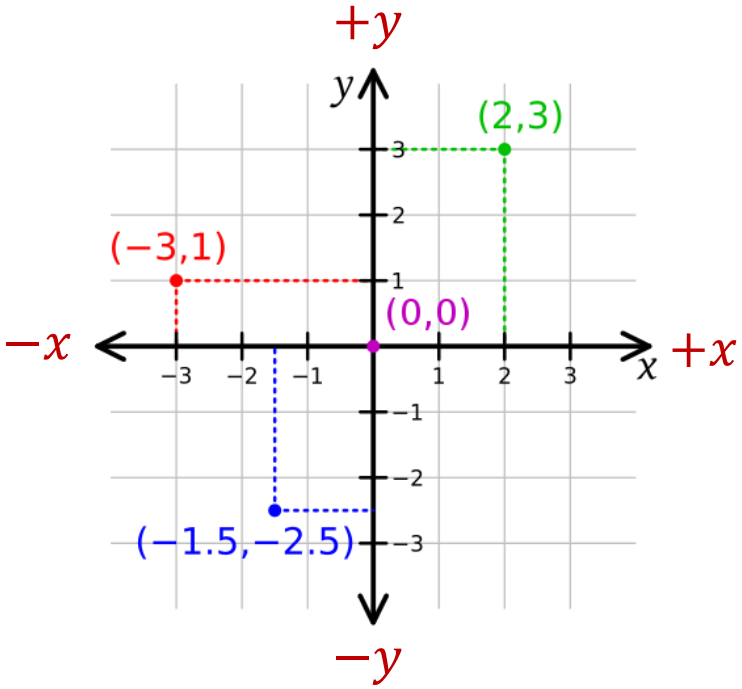
Overview

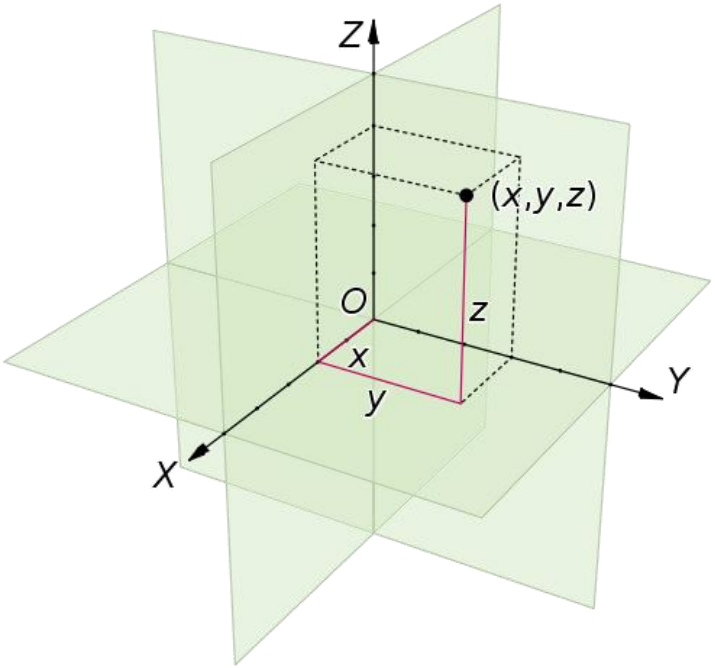
- 3D Coordinate System
- Rendering Pipeline in Unity
  - The process of taking a geometric description of a 3D scene and generating a 2D image from it
- Create a 3D Scene in Unity
- Geometry for Graphics

Overview

Coordinate Systems

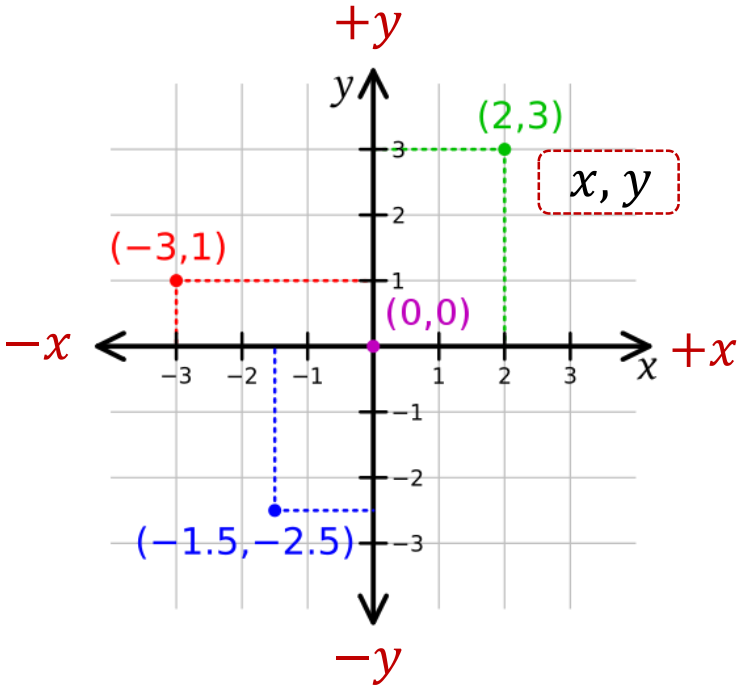
- 2D Cartesian Coordination Systems
- 3D Cartesian Coordination Systems

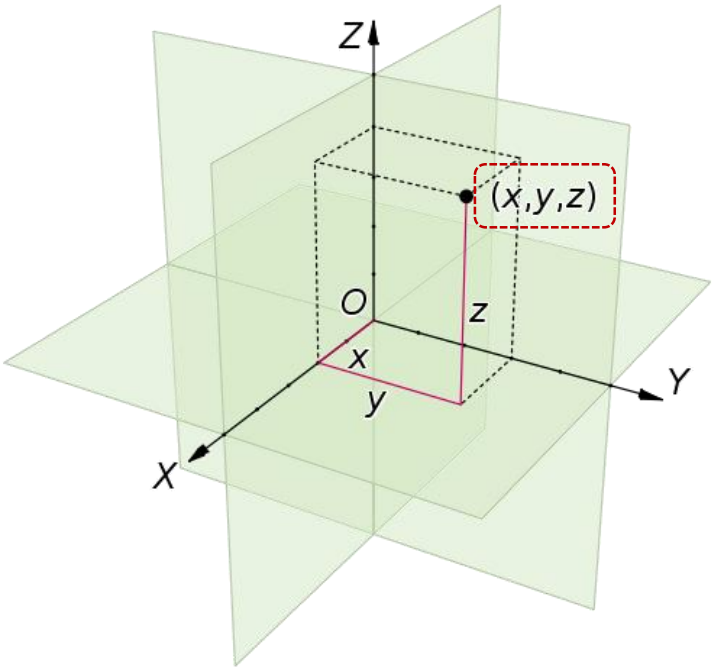




Coordinate Systems

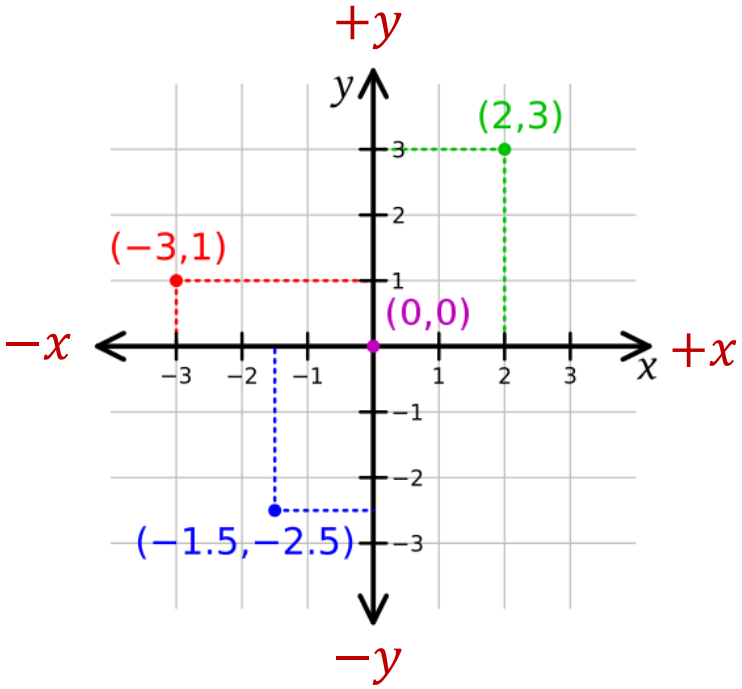
- 2D Cartesian Coordination Systems
- 3D Cartesian Coordination Systems

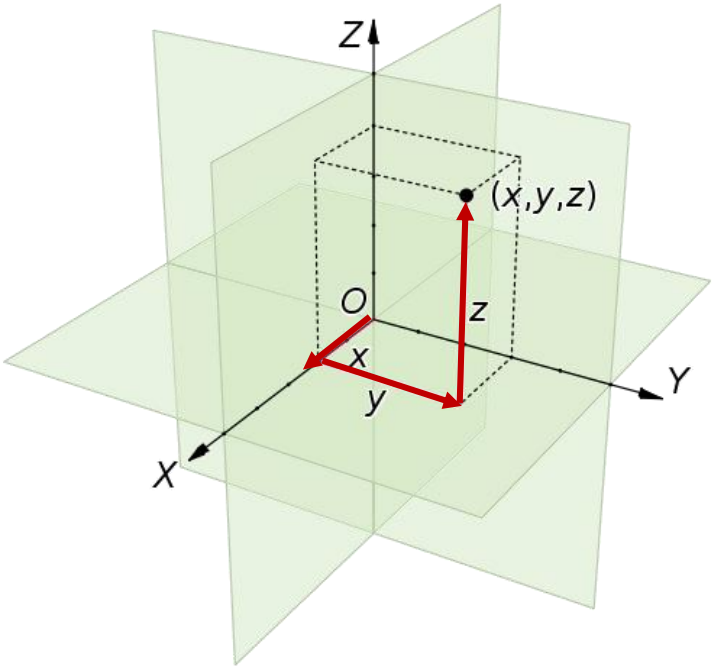




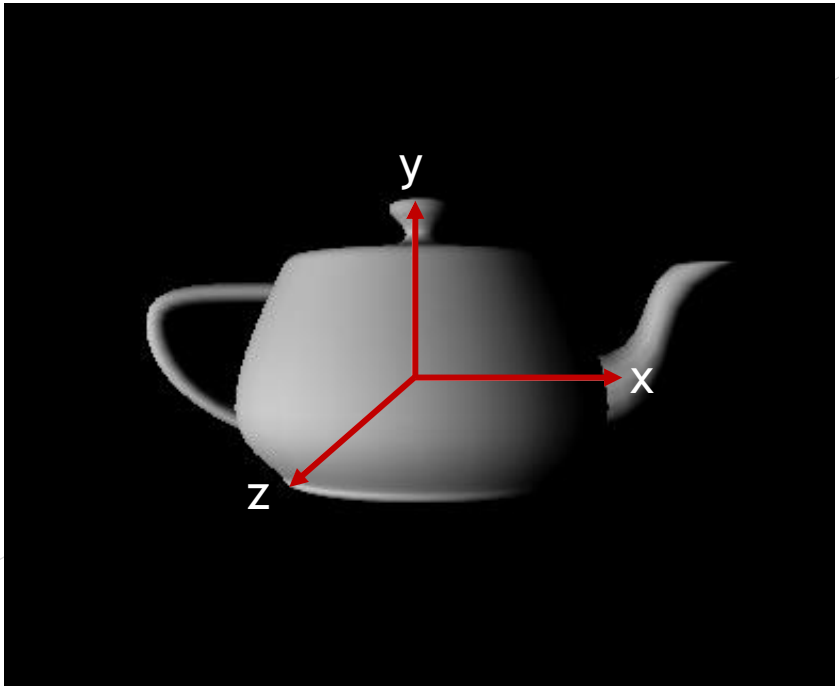
Coordinate Systems

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





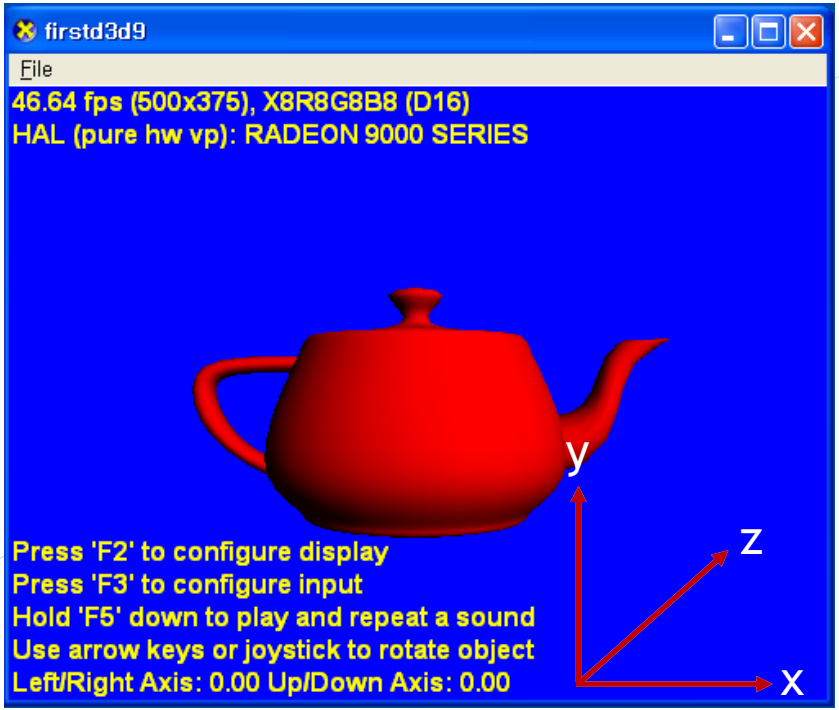
3D Coordinate Systems



이미지 출처 : OpenGL

- **OpenGL** coordinate system is right-handed
- x+ to the right
- y+ up
- z+ coming out of the screen

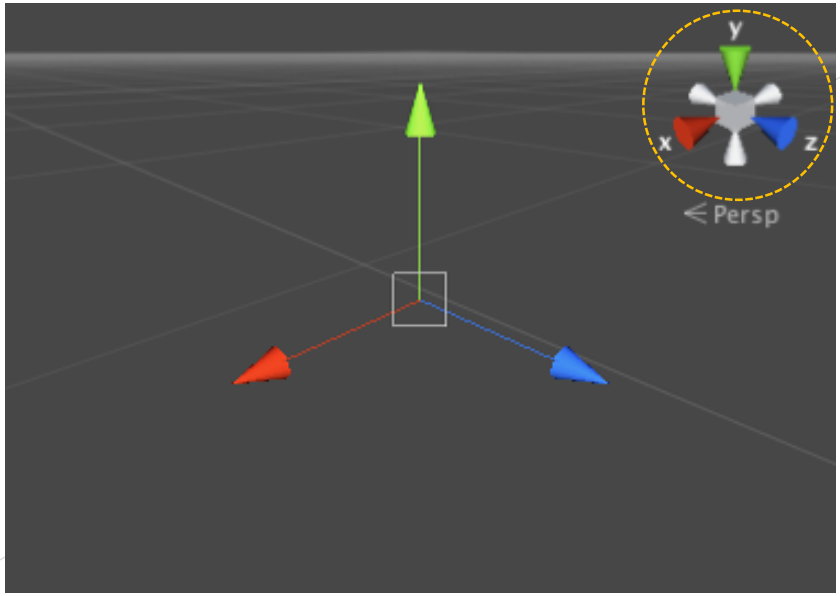
3D Coordinate Systems



이미지 출처 : DirectX

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

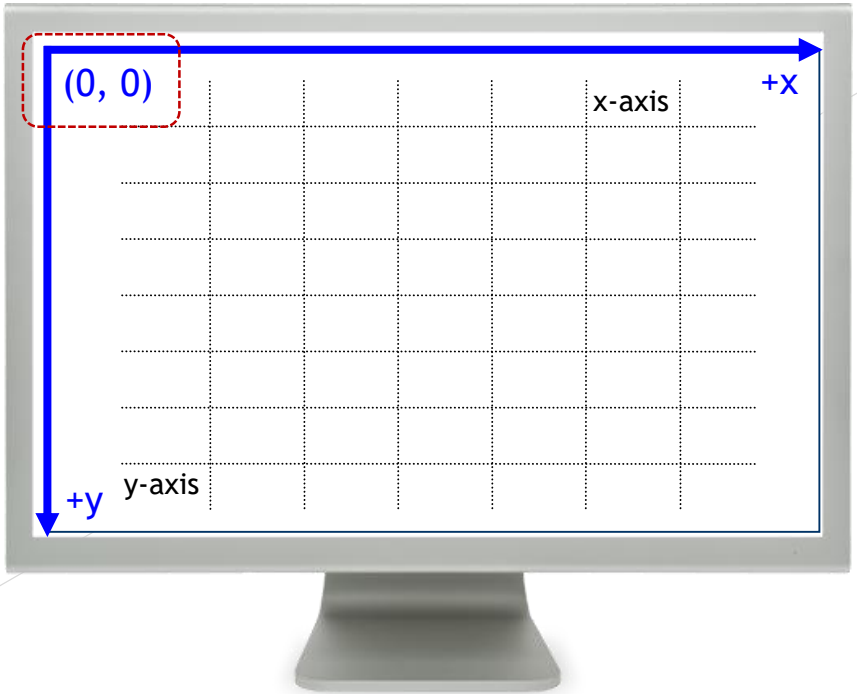
3D Coordinate Systems



이미지 출처 : Unity

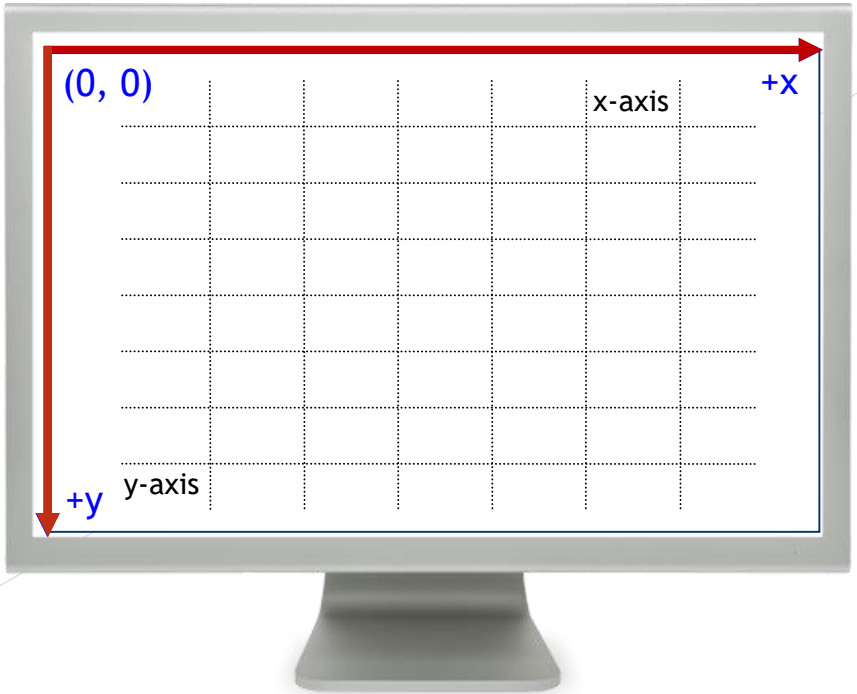
- **Unity3D** coordinate system is left-handed
- x+ to the right
- y+ up
- z+ forward

Screen Coordinate System



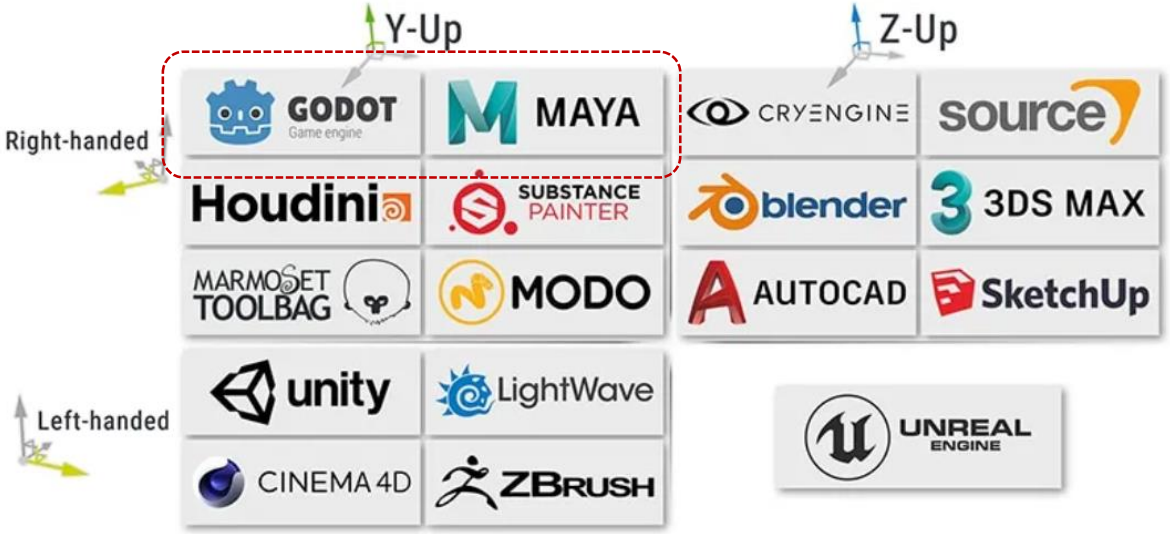
- In screen coordinate system, the origin is located at the top left of the screen and the value is (0, 0). x+ is right. y+ is down.
- 1 unit = 1 pixel

Screen Coordinate System



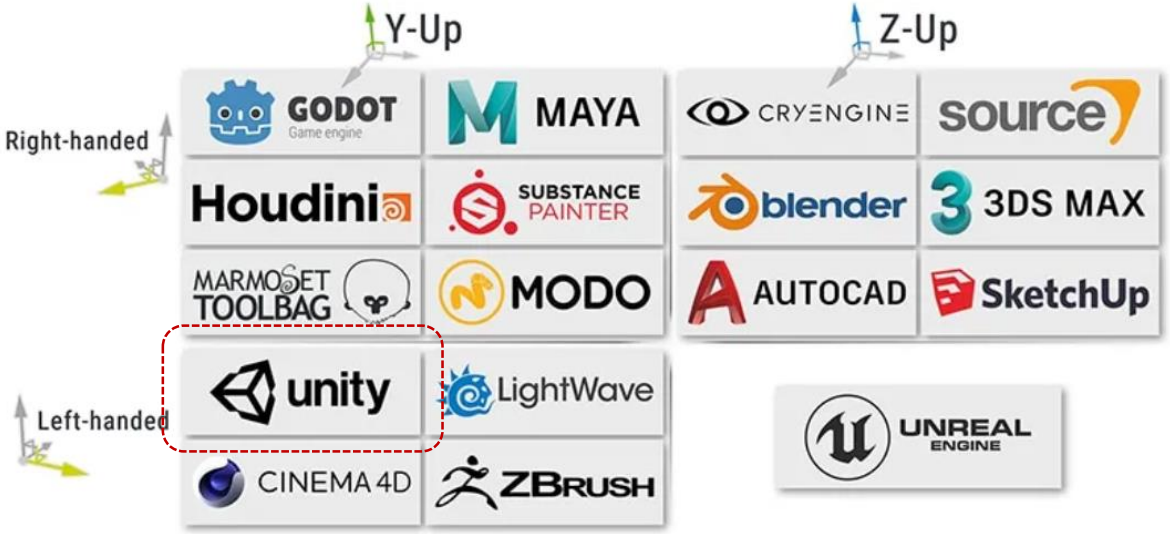
- In screen coordinate system, the origin is located at the top left of the screen and the value is (0, 0). x+ is right. y+ is down.
- 1 unit = 1 pixel

3D Coordinate Systems



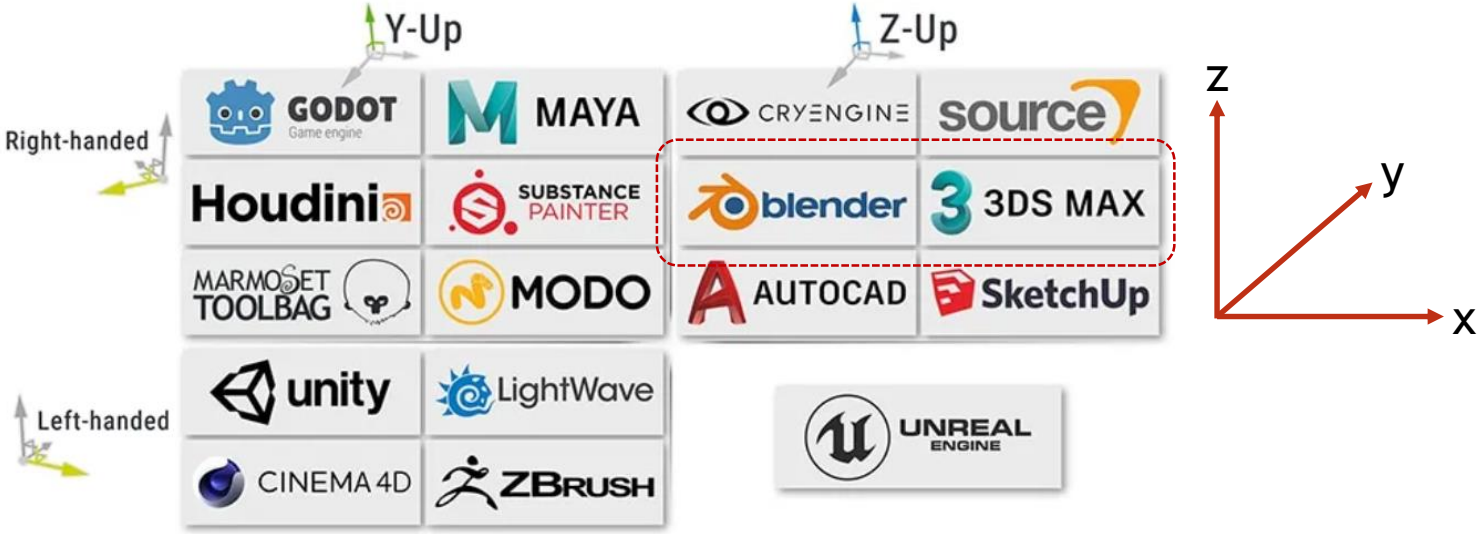
이미지 출처 : A Guide to Unity's Coordinate System (With Practical Examples)
https://www.techarthub.com/a-guide-to-unitys-coordinate-system-with-practical-examples/

3D Coordinate Systems



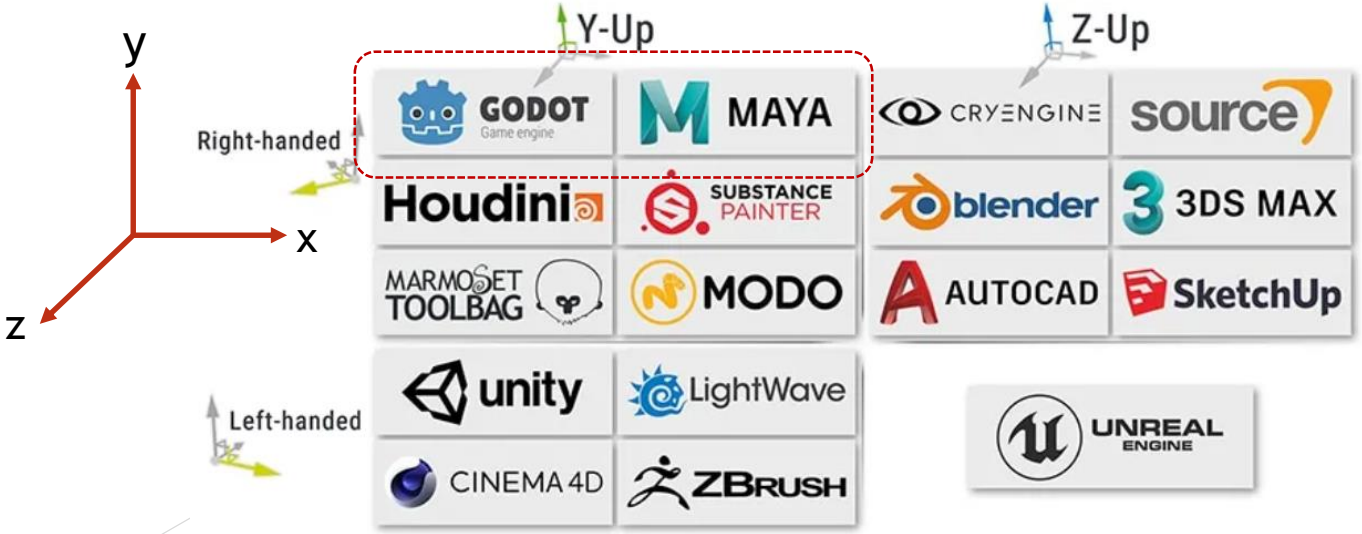
이미지 출처 : A Guide to Unity's Coordinate System (With Practical Examples)
https://www.techarthub.com/a-guide-to-unitys-coordinate-system-with-practical-examples/

3D Coordinate Systems



이미지 출처 : 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



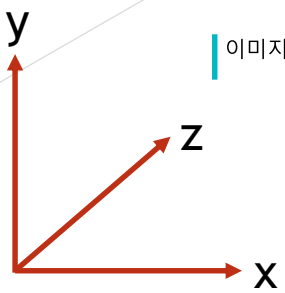
이미지 출처 : A Guide to Unity's Coordinate System (With Practical Examples)
https://www.techarthub.com/a-guide-to-unitys-coordinate-system-with-practical-examples/

3D Coordinate Systems

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

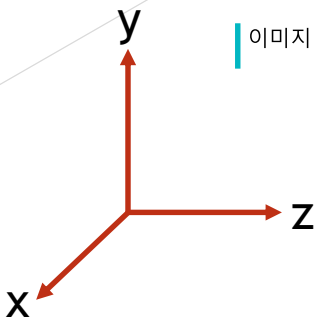
3D Coordinate Systems

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

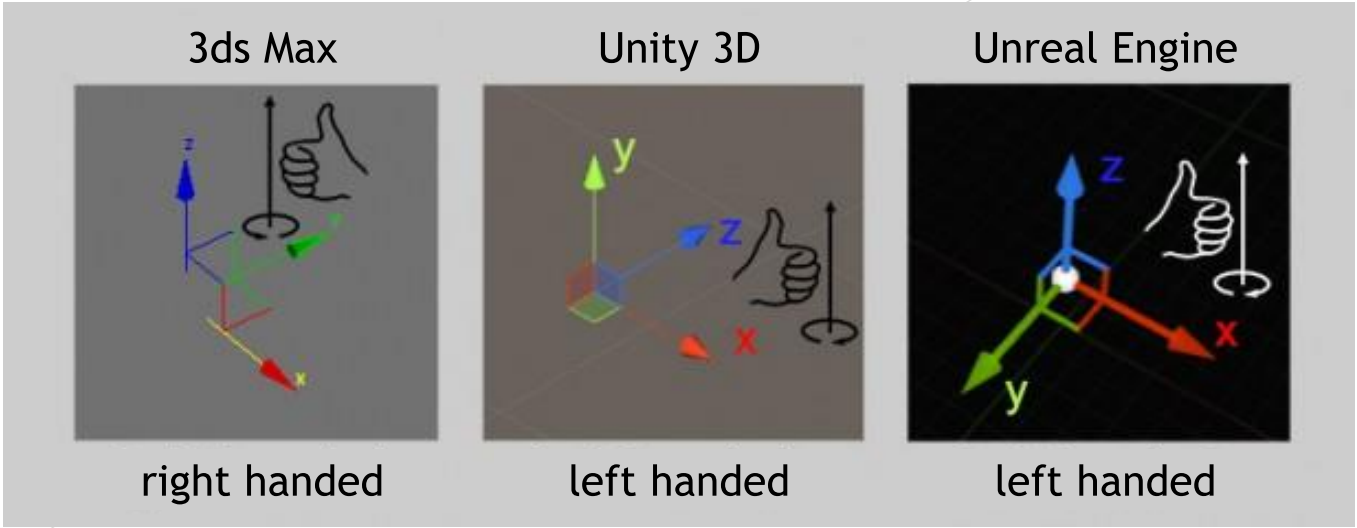


3D Coordinate Systems

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

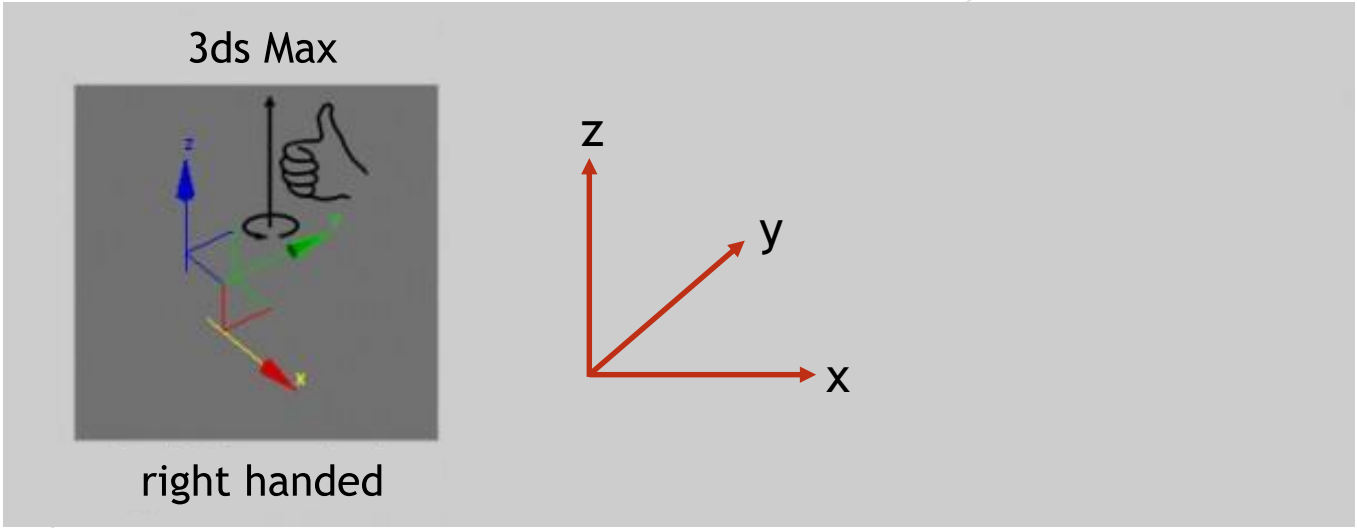


3D Coordinate Systems



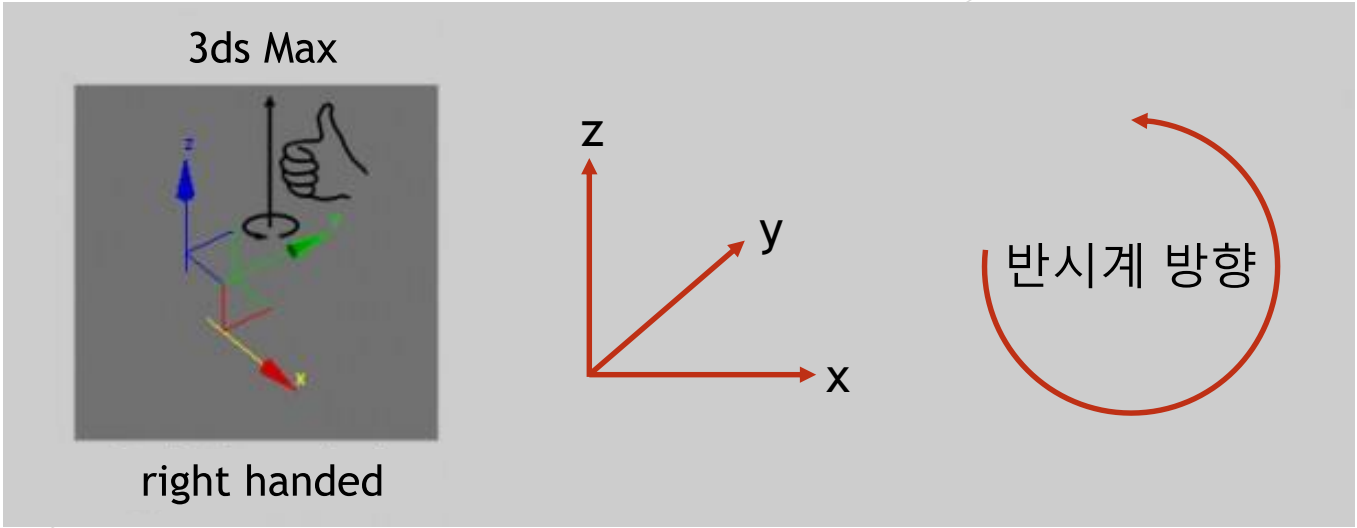
이미지 출처 :World Coordinate Systems in 3ds Max, Unity and Unreal Engine
http://www.aclockworkberry.com/world-coordinate-systems-in-3ds-max-unity-and-unreal-engine/

3D Coordinate Systems



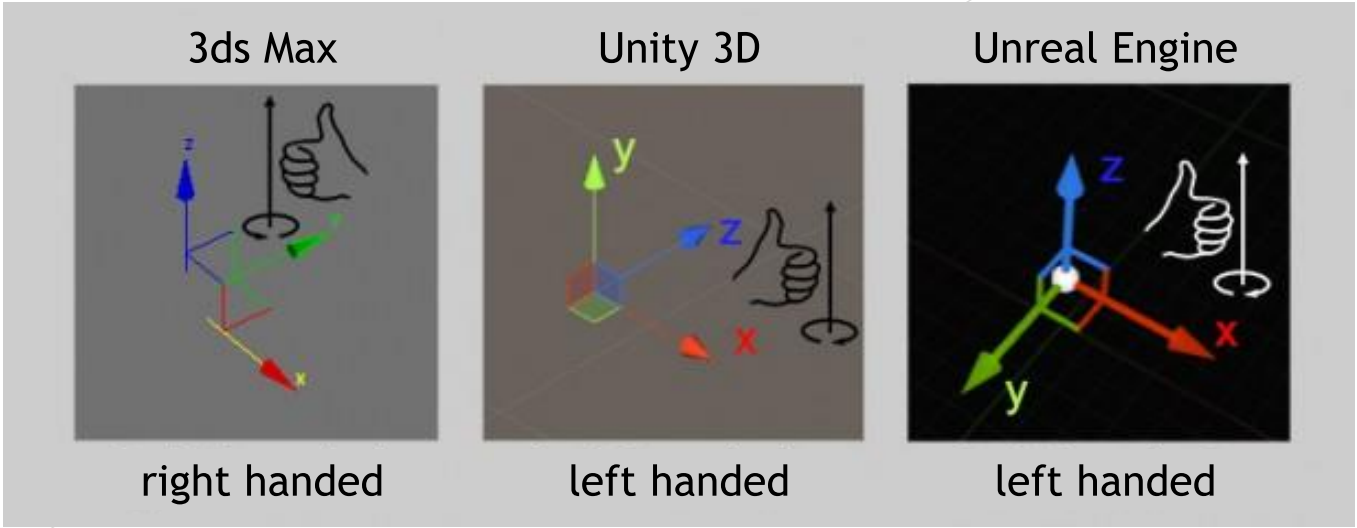
이미지 출처 :World Coordinate Systems in 3ds Max, Unity and Unreal Engine
http://www.aclockworkberry.com/world-coordinate-systems-in-3ds-max-unity-and-unreal-engine/

3D Coordinate Systems



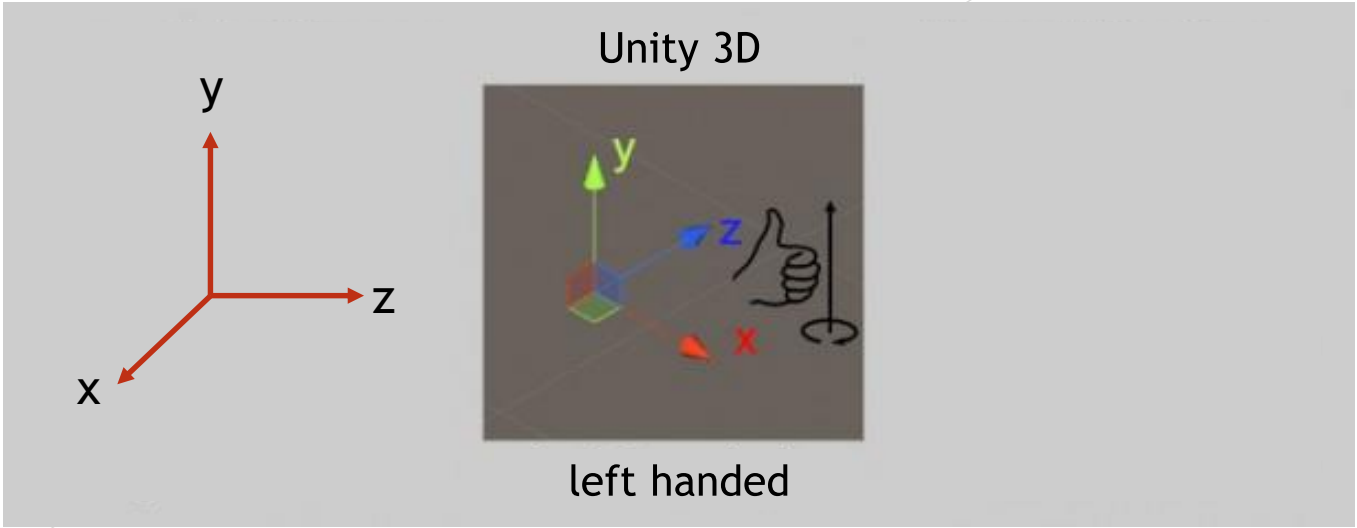
이미지 출처 :World Coordinate Systems in 3ds Max, Unity and Unreal Engine
http://www.aclockworkberry.com/world-coordinate-systems-in-3ds-max-unity-and-unreal-engine/

3D Coordinate Systems



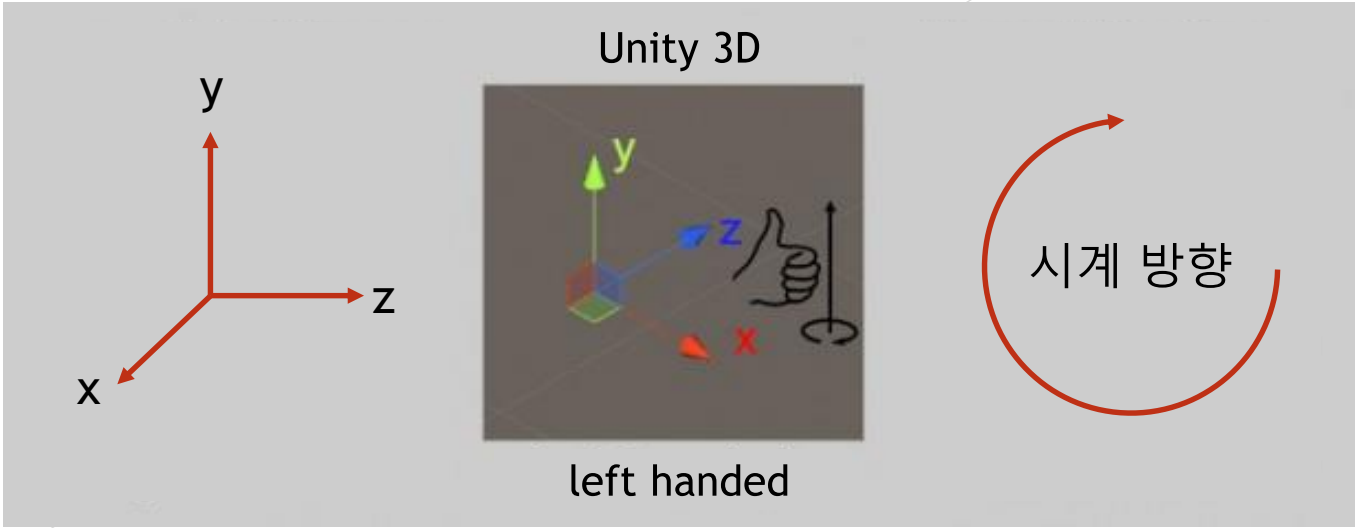
이미지 출처 :World Coordinate Systems in 3ds Max, Unity and Unreal Engine
http://www.aclockworkberry.com/world-coordinate-systems-in-3ds-max-unity-and-unreal-engine/

3D Coordinate Systems



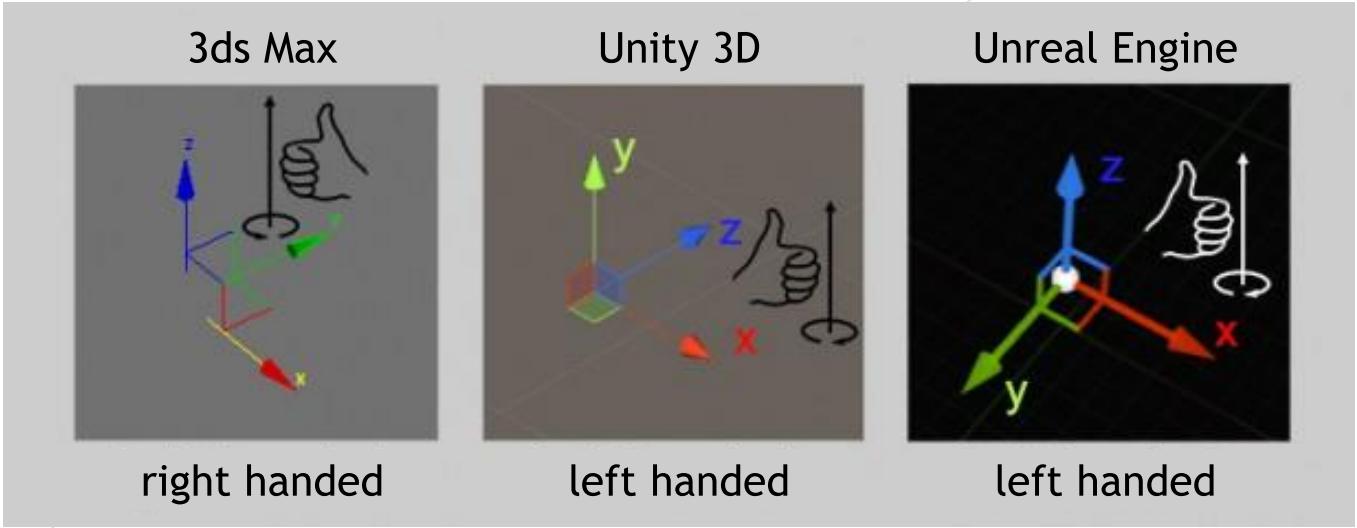
이미지 출처 :World Coordinate Systems in 3ds Max, Unity and Unreal Engine
http://www.aclockworkberry.com/world-coordinate-systems-in-3ds-max-unity-and-unreal-engine/

3D Coordinate Systems



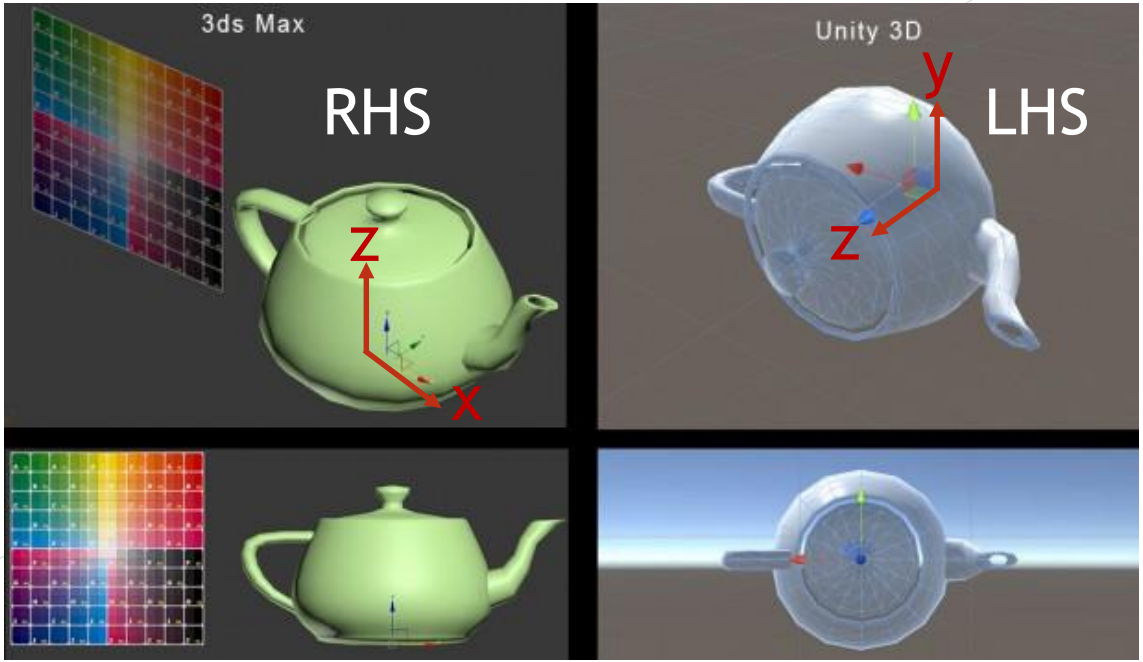
이미지 출처 :World Coordinate Systems in 3ds Max, Unity and Unreal Engine
http://www.aclockworkberry.com/world-coordinate-systems-in-3ds-max-unity-and-unreal-engine/

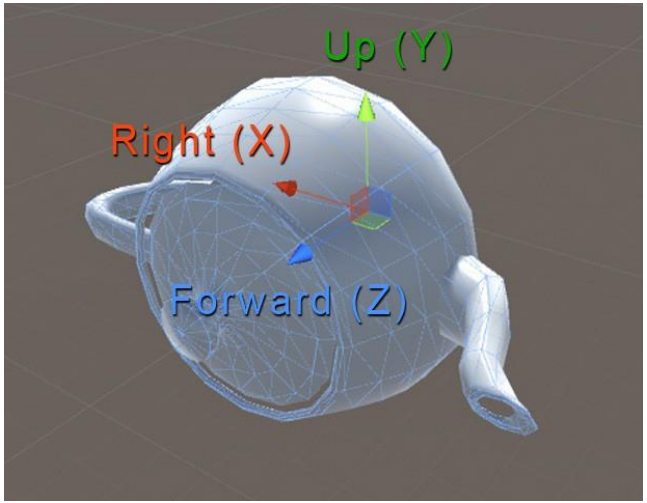
3D Coordinate Systems



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

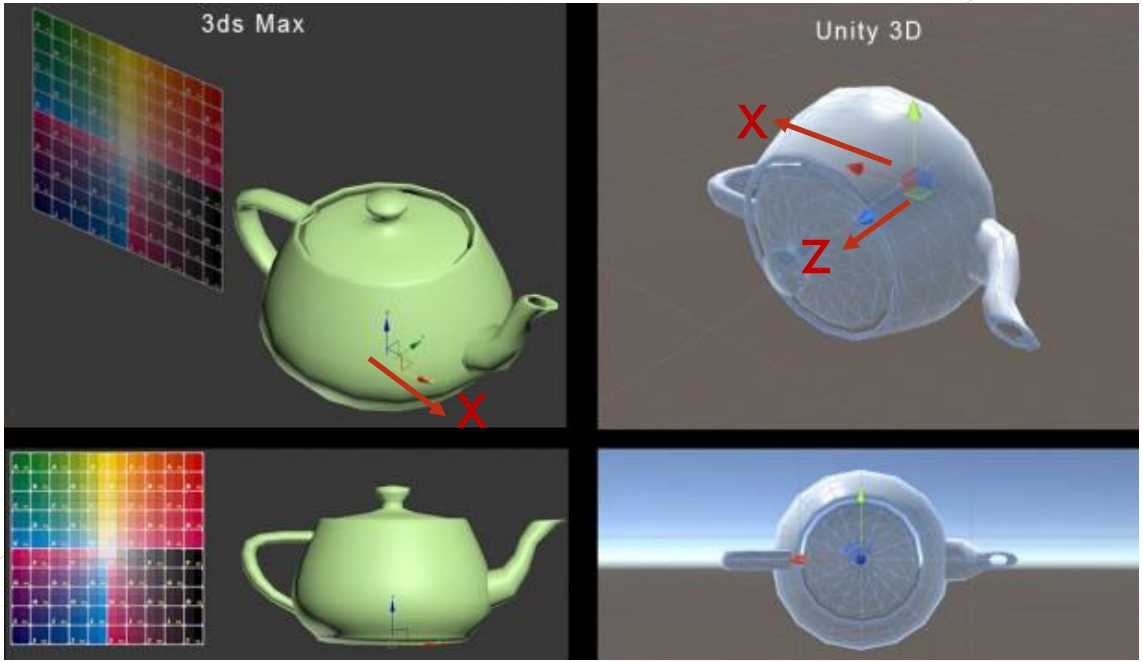
3D Coordinate Systems

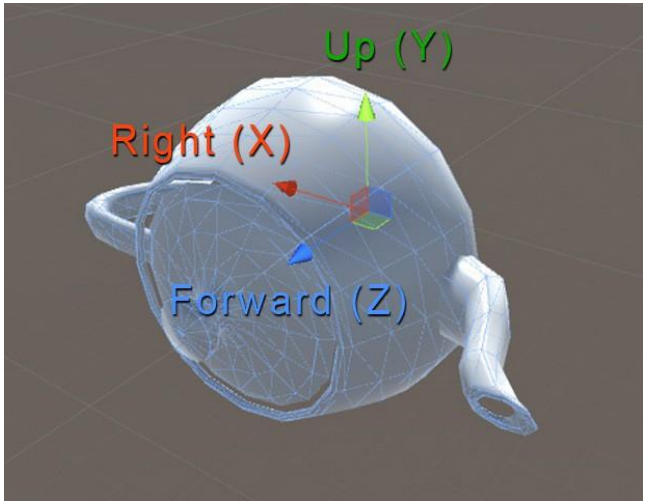




이미지 출처 :World Coordinate Systems in 3ds Max, Unity and Unreal Engine
http://www.aclockworkberry.com/world-coordinate-systems-in-3ds-max-unity-and-unreal-engine/

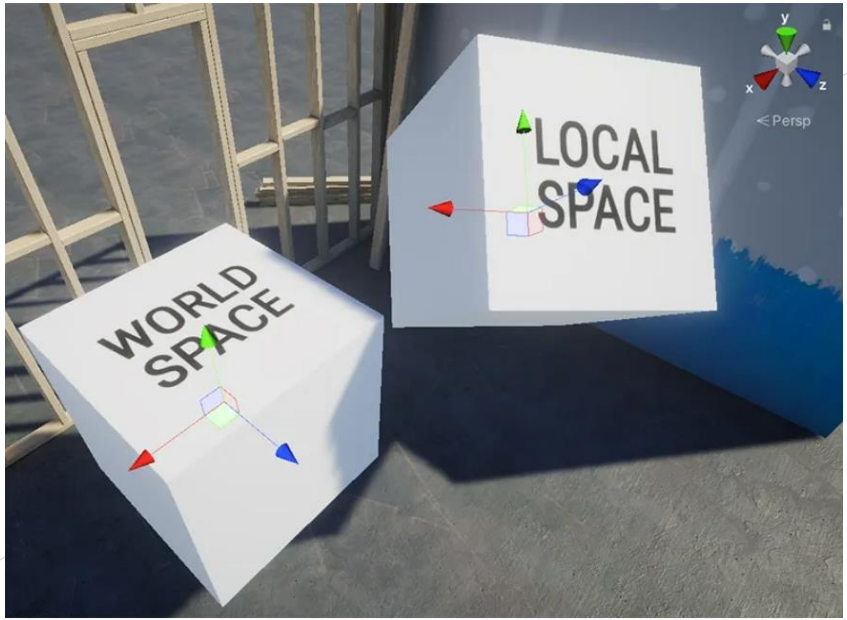
3D Coordinate Systems





이미지 출처 :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



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

- **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.

2

Rendering Pipeline in Unity

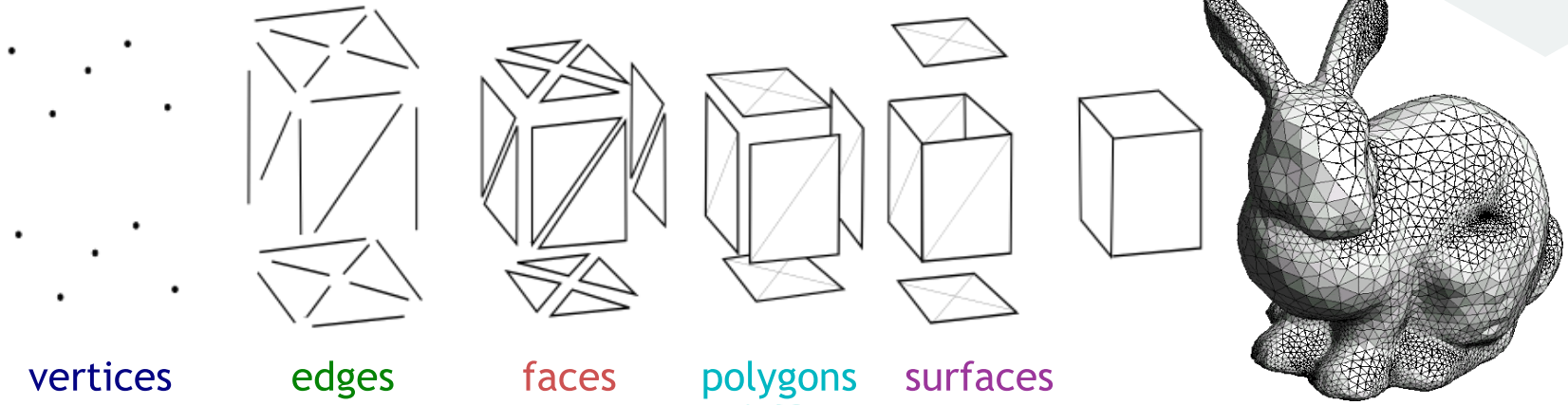
Rendering

- Rendering is the process of drawing a scene on the computer screen.
- Rendering involves a combination of geometry calculations, textures, surface treatments, the viewer’s perspective, and lighting.
- The rendering pipeline is the process of transforming all of these data into the virtual environment on the screen.

Rendering

Geometry

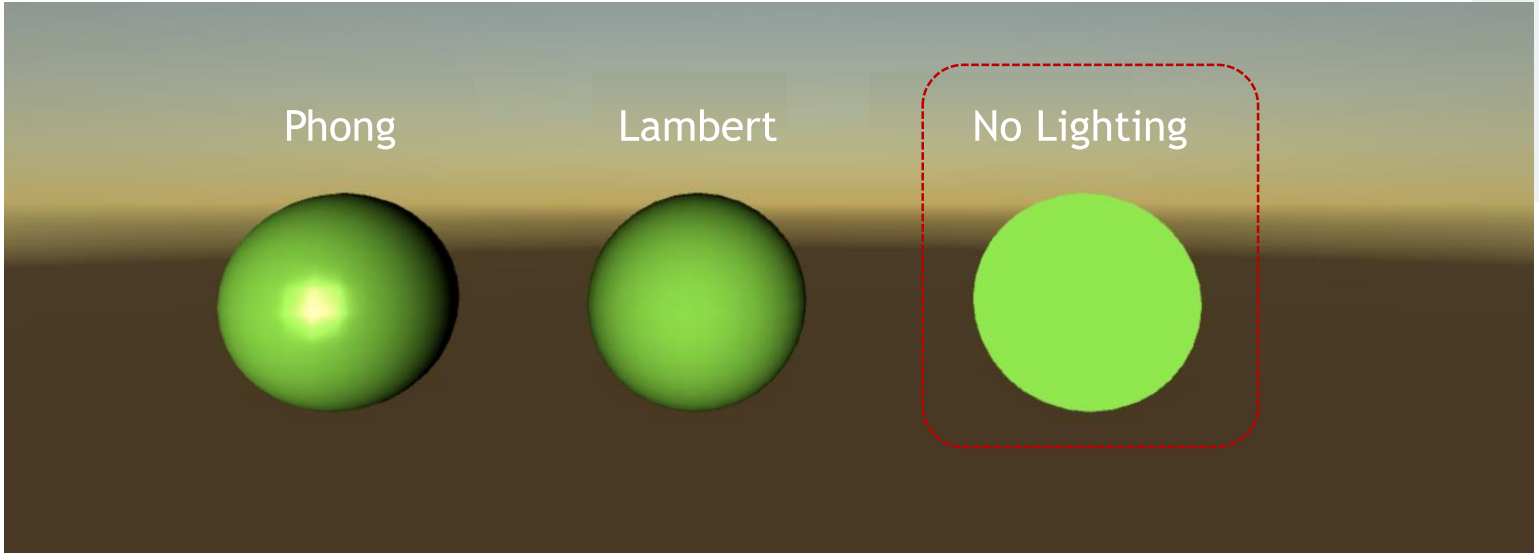
- Creating 3d and 2d models using the mesh data and simulating the virtual world with its dimensions.
- Geometry mesh data are collected (Vertices Array, Normals Array, Triangle Array and UV Array).



이미지 출처 :https://medium.com/shader-coding-in-unity-from-a-to-z/rendering-pipe-line-f0471aa0904b

Geometry Lighting

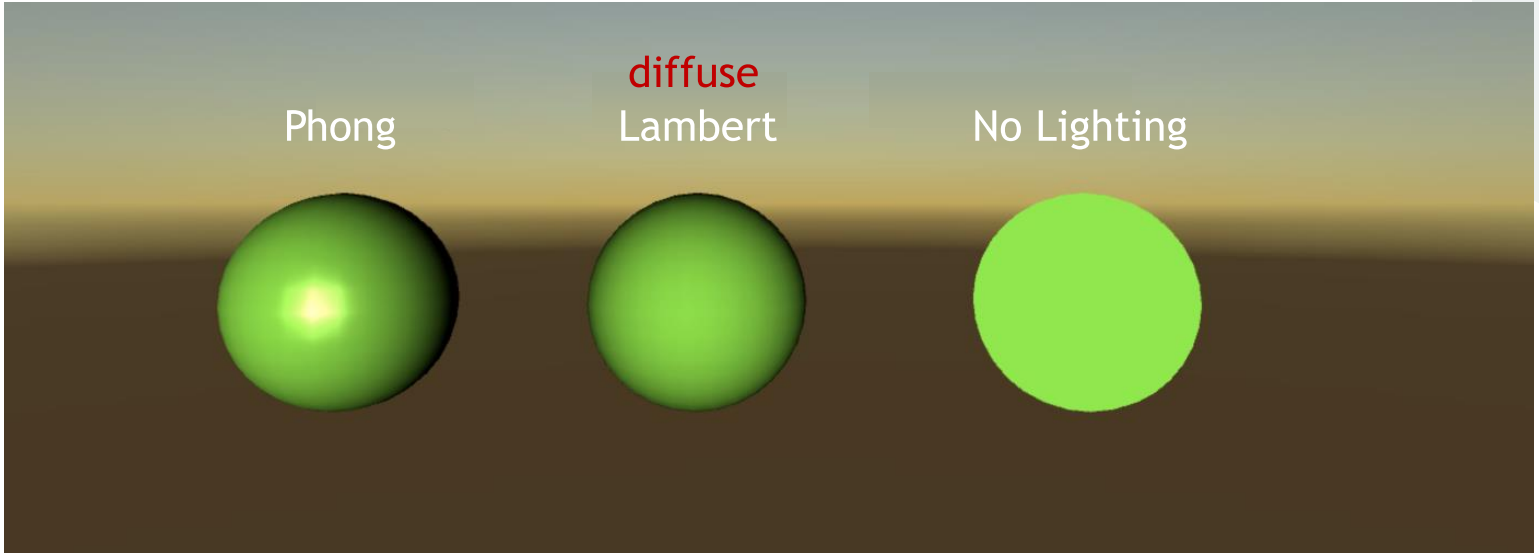
- In the Illumination stage also we add lighting effects to the virtual world



이미지 출처 :https://medium.com/shader-coding-in-unity-from-a-to-z/rendering-pipe-line-f0471aa0904b

Geometry Lighting

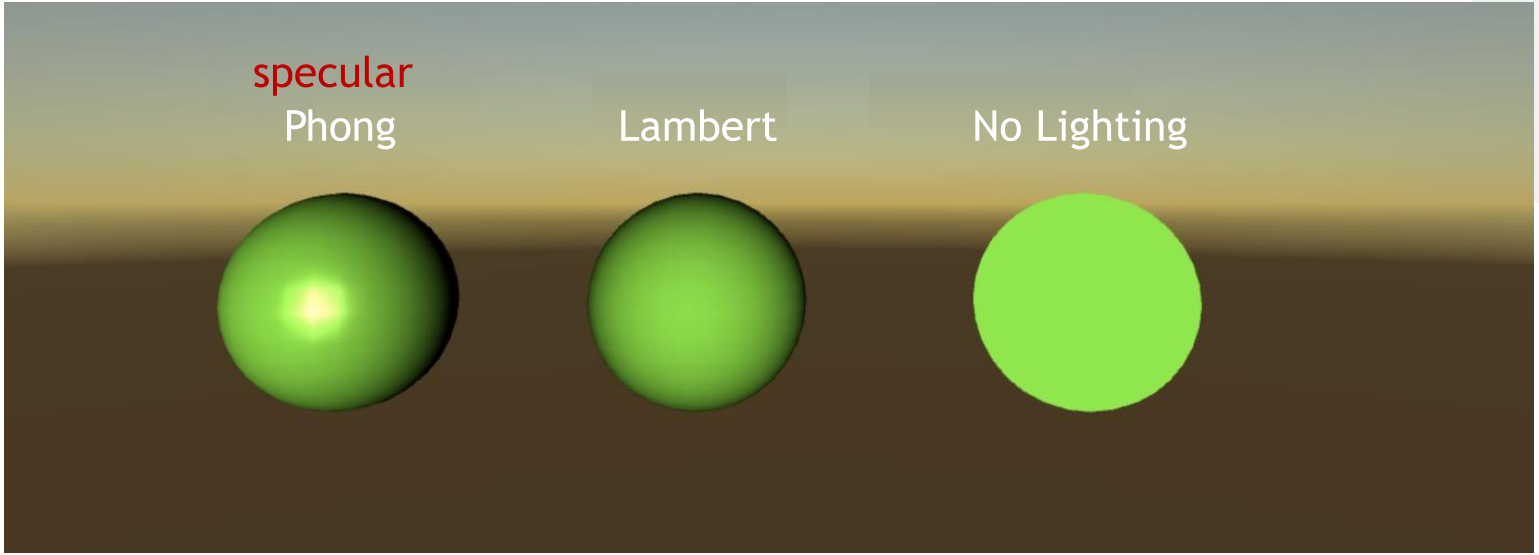
- In the Illumination stage also we add lighting effects to the virtual world



이미지 출처 :https://medium.com/shader-coding-in-unity-from-a-to-z/rendering-pipe-line-f0471aa0904b

Geometry Lighting

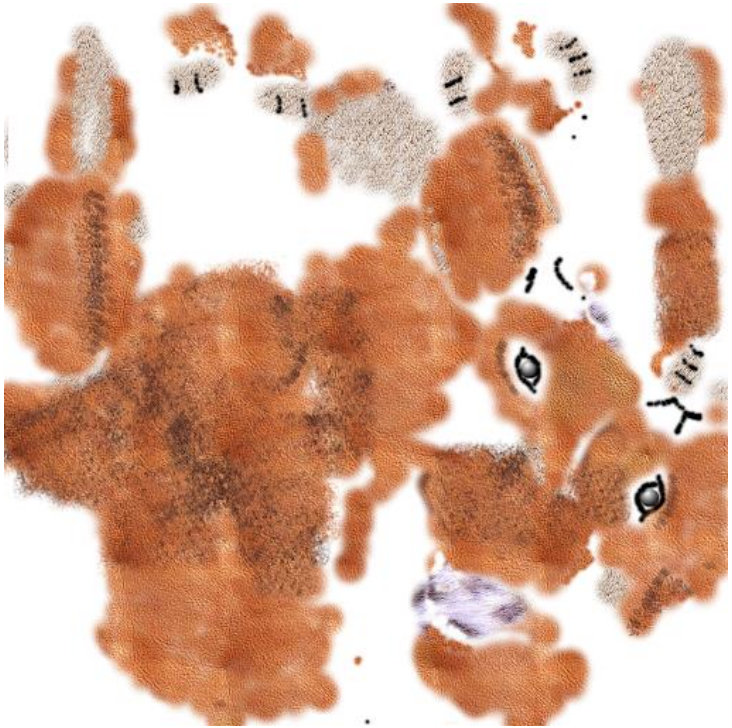
- In the Illumination stage also we add lighting effects to the virtual world

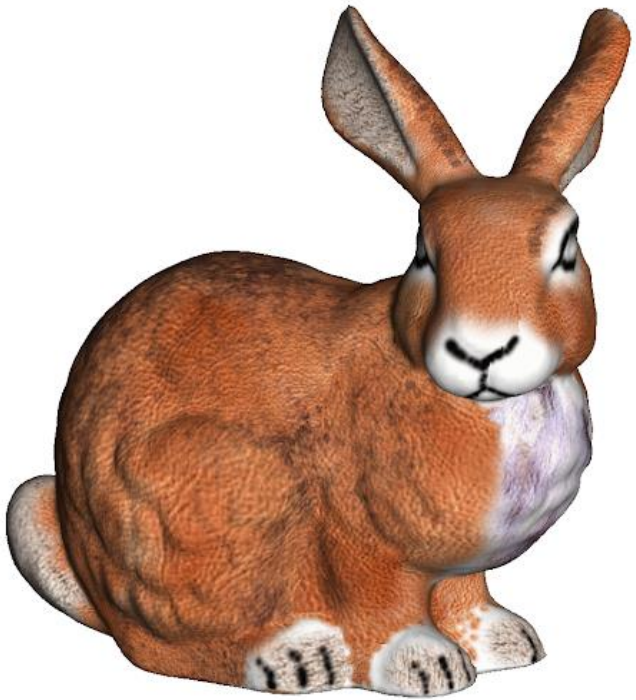


이미지 출처 :https://medium.com/shader-coding-in-unity-from-a-to-z/rendering-pipe-line-f0471aa0904b

Geometry Textures

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

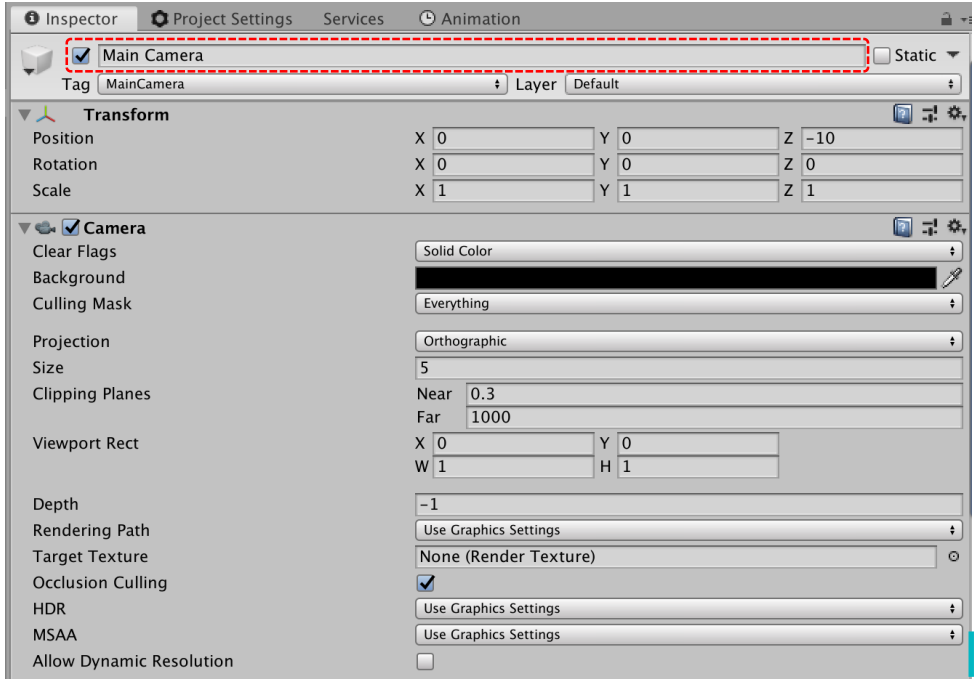




이미지 출처 :https://medium.com/shader-coding-in-unity-from-a-to-z/rendering-pipe-line-f0471aa0904b

Viewer’s Perspective (Camera Input)

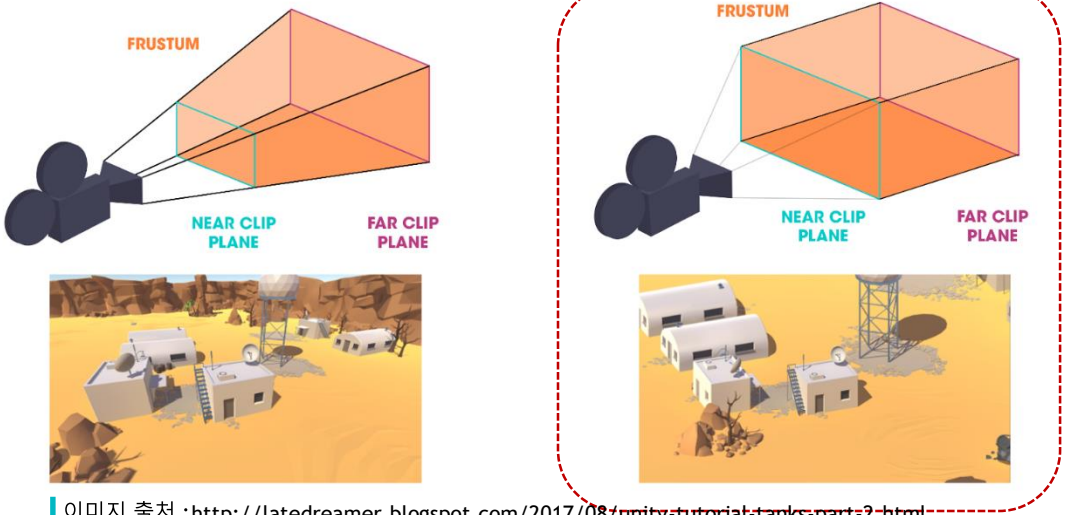
- Before rendering the environment on the screen we consider the camera input such as (field of view, Projection Mode [Orthographic or Perspective]).



이미지 출처 :Unity

Orthographic vs Perspective Viewing

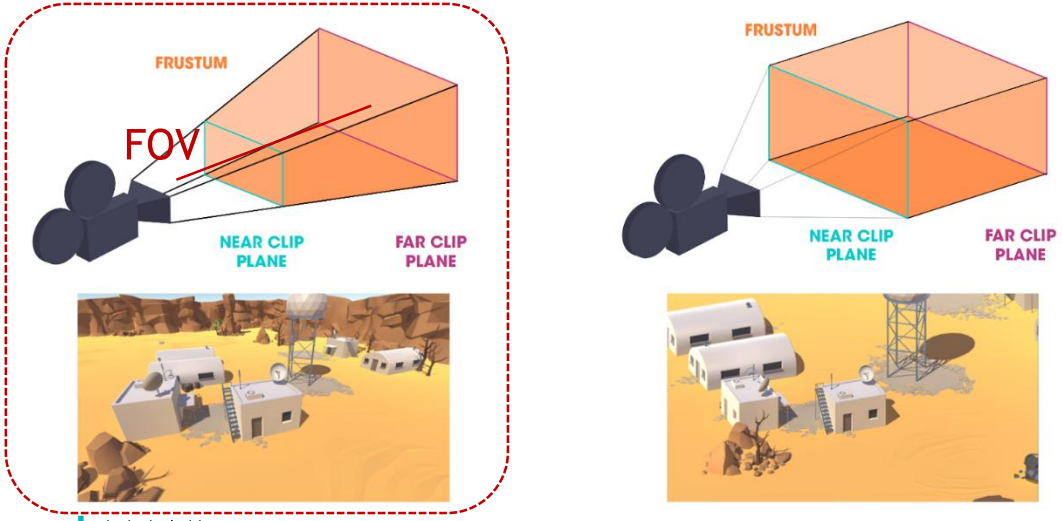
- Orthographic parallel projection
  - Points are projected onto the z=0 plane towards the z-axis.
- Perspective projection
  - it uses the y-direction viewing angle (FOV) and the aspect ratio (the value of the width of the nearest clipping plane divided by the height)



이미지 출처 :http://latedreamer.blogspot.com/2017/08/unity-tutorial-tanks-part-2.html

Orthographic vs Perspective Viewing

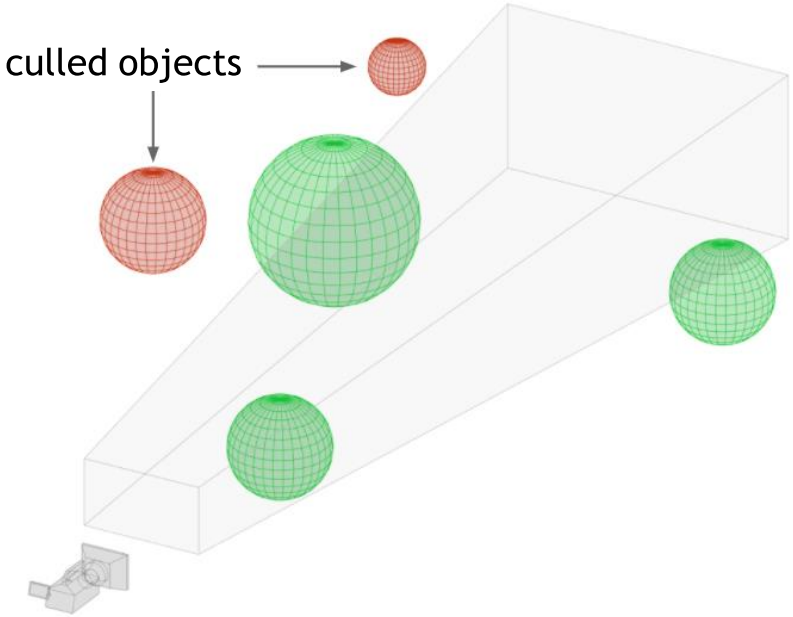
- Orthographic parallel projection
  - Points are projected onto the z=0 plane towards the z- axis.
- Perspective projection
  - it uses the y-direction viewing angle (FOV) and the aspect ratio (the value of the width of the nearest clipping plane divided by the height)



이미지 출처 :http://latedreamer.blogspot.com/2017/08/unity-tutorial-tanks-part-2.html

Clipping

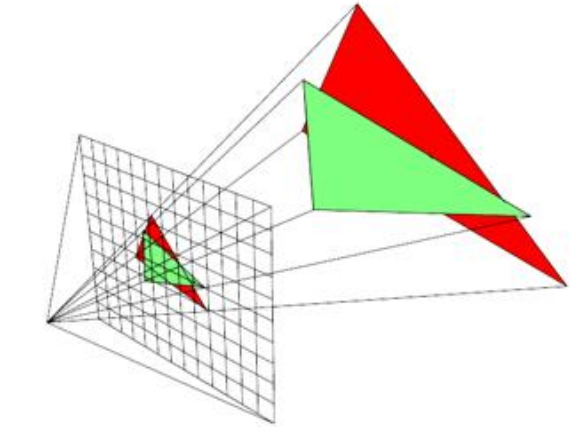
- **Objects projected outside the window are clipped** without appearing as an image by placing a pyramid like clipping volume in front of the camera.

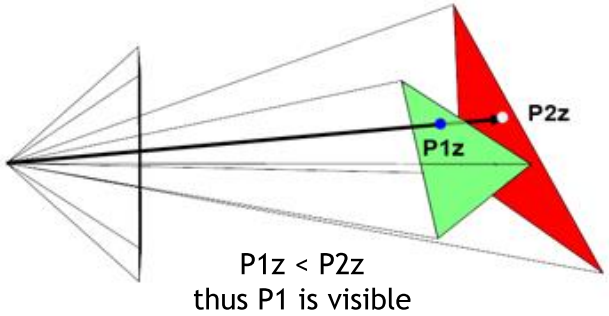


이미지 출처 :https://docs.unity3d.com/kr/2020.1/Manual/BestPracticeLightingPipelines.html

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.





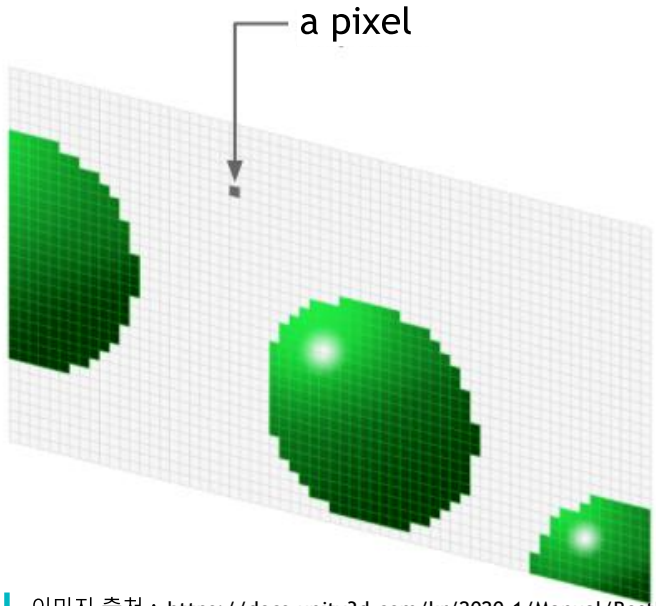
P1z < P2z
thus P1 is visible

이미지 출처 :www.scratchapixel.com

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 virtual environment several times through different filters then output the result on the screen.



이미지 출처 : https://docs.unity3d.com/kr/2020.1/Manual/BestPracticeLightingPipelines.html

Post-Processing

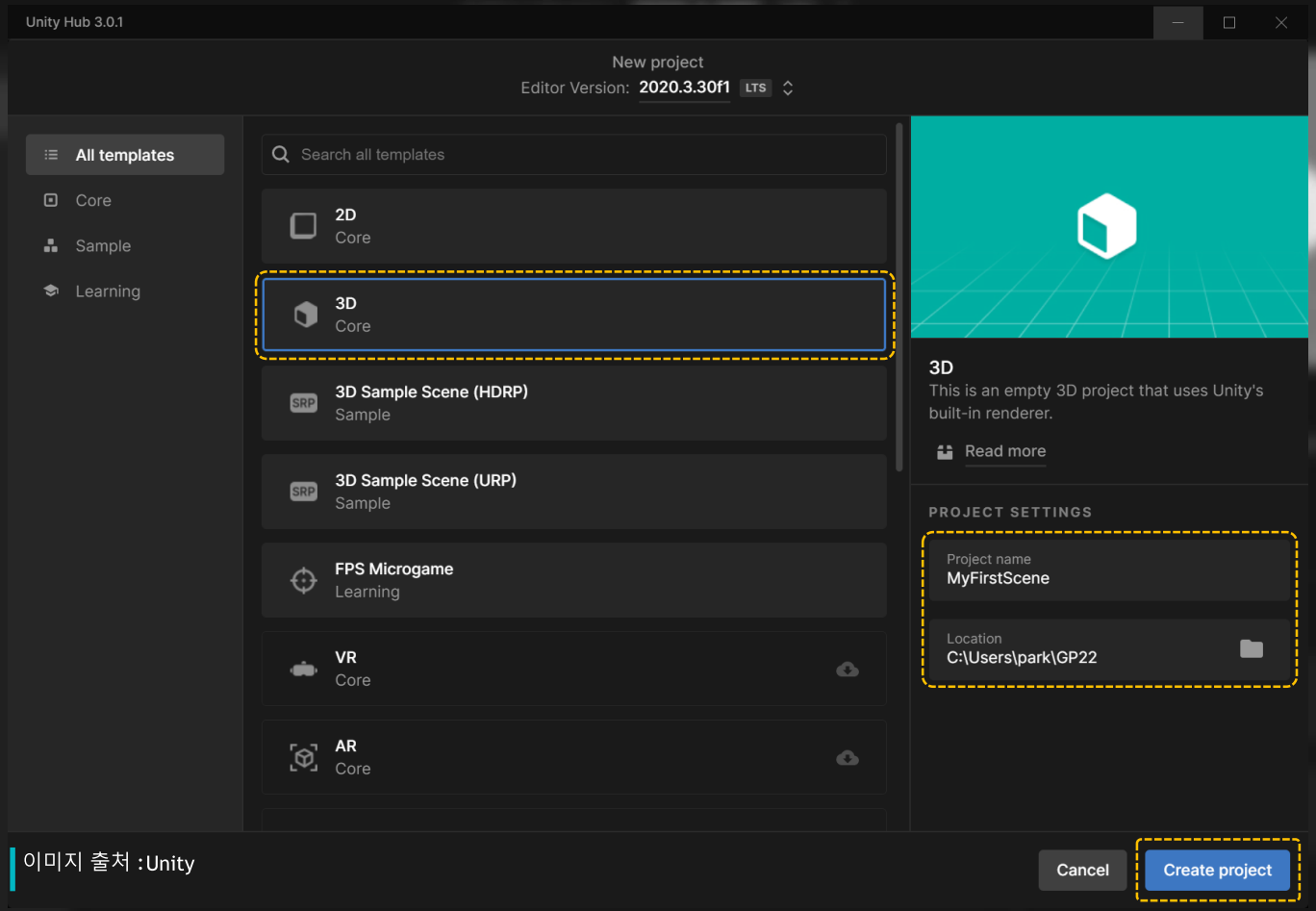
- Post-processing effects we add to the 2D image just before displaying the final output on the screen.
  - Depth of field is a post-processing effect applied to the 2D final image

이미지 출처 https://medium.com/shader-coding-in-unity-from-a-to-z/rendering-pipe-line-f0471aa0904b

3

Creating a 3D Scene in Unity

Creating a Scene



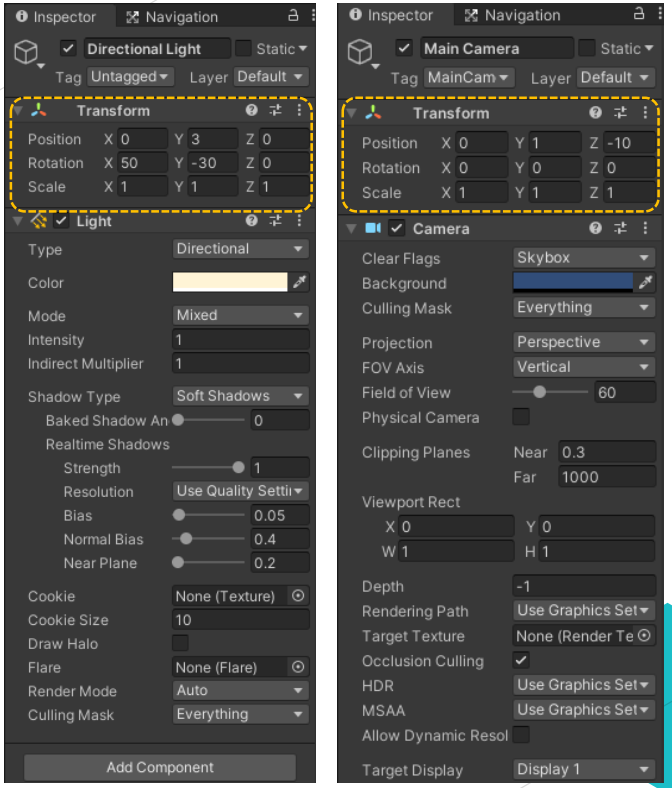
이미지 출처 :Unity

Everything is a GameObject

- GameObject
  - https://docs.unity3d.com/ScriptReference/GameObject.html
- All game objects in your scene hierarchy
  - Cameras
  - Lights
  - Gameplay logic
  - User interface
  - Etc.
- Can be destroyed/created on the fly or be made to not get destroyed when the game changes scenes (this is how you get persistent behavior!)
  - https://docs.unity3d.com/ScriptReference/Object.DontDestroyOnLoad.html

Everything is a GameObject

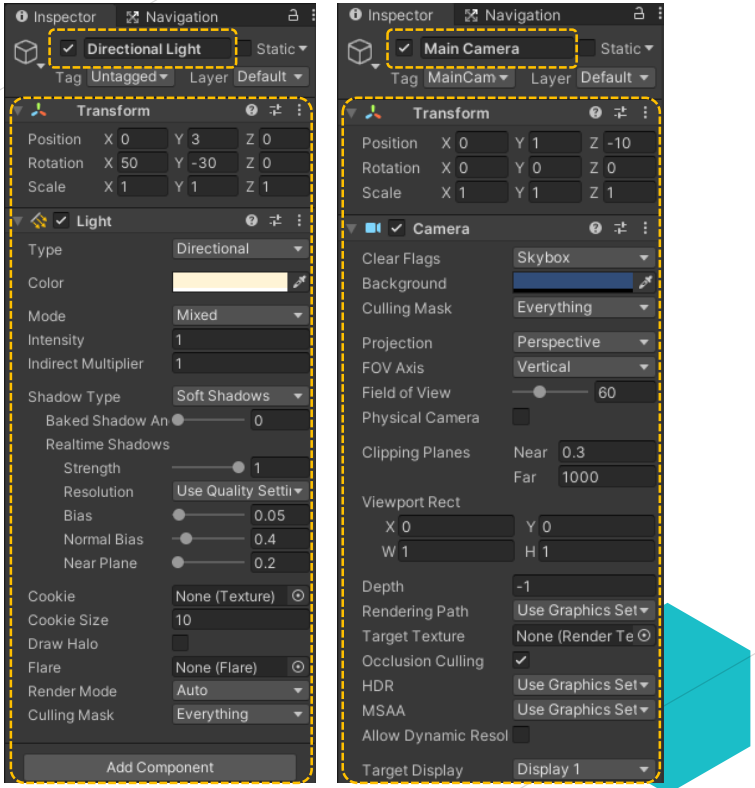
- A **GameObject** does nothing on its own.
- All **GameObject** have a Transform component to let it know its position, rotation, scale.
- Must add Components to the GameObject to give it some behavior.



이미지 출처 :Unity

Everything is a GameObject

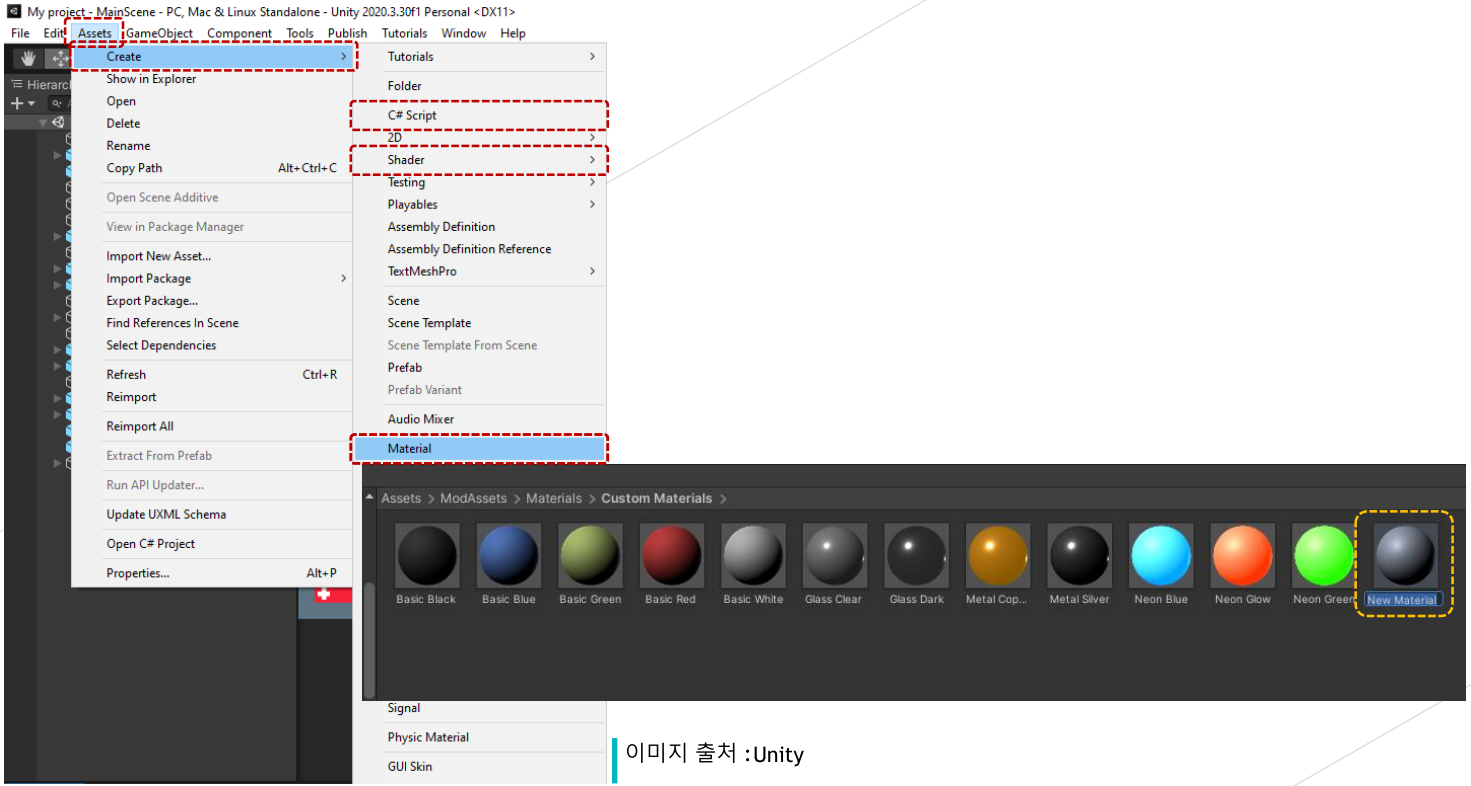
- A **GameObject** does nothing on its own.
- All **GameObject** have a Transform component to let it know its position, rotation, scale.
- Must add Components to the GameObject to give it some behavior.



이미지 출처 :Unity

Everything is a GameObject

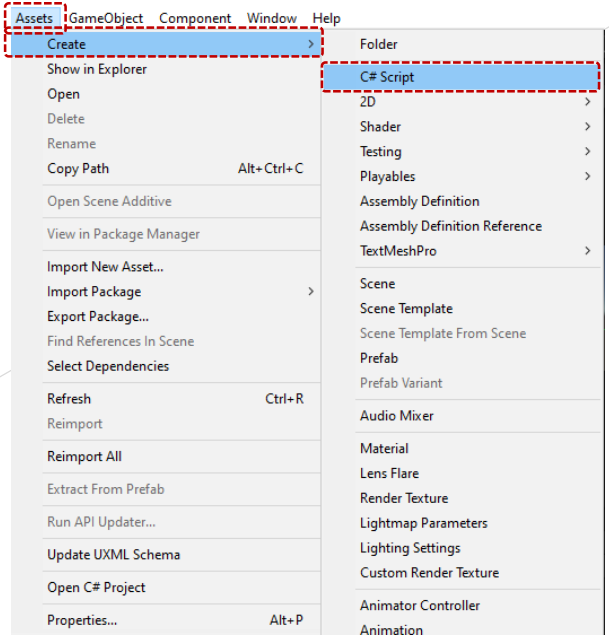
- Many **Components** already exist!

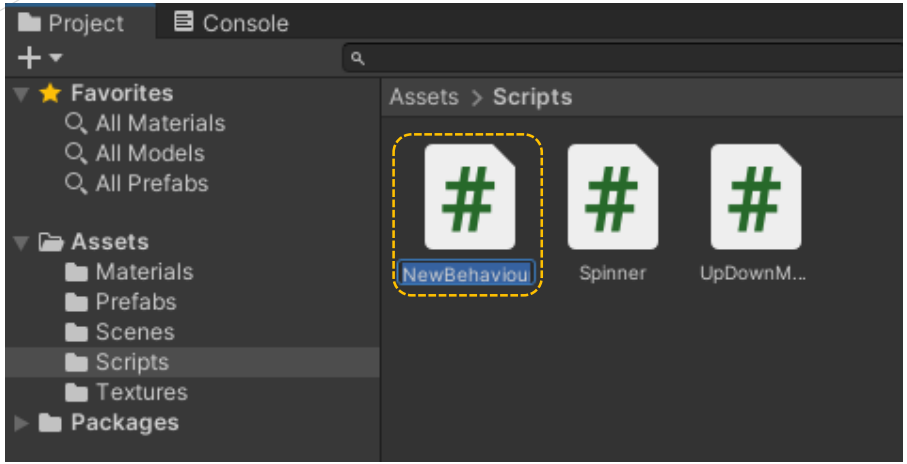


이미지 출처 : Unity

Everything is a GameObject

- Many Components already exist! But you might want to **create your own.**
- Write your **C# scripts** that inherit from **MonoBehavior**
  - http://docs.unity3d.com/ScriptReference/MonoBehaviour.html





이미지 출처 :Unity

Assets

- Scenes
- Models
- Textures
- Materials
- Prefabs
- Scripts
- Animations
- Particles
- Sprites
- Audio
- etc

Assets

Prefabs

- You created some hierarchy of GameObjects and you want to reuse this hierarchy in multiple places. Use Prefabs!
- You can also assign prefabs to scripts which have a public variable of type GameObject to be able to programmatically spawn prefabs!
- Or you could just load prefab by its path (may be more convenient in certain cases)
  - https://docs.unity3d.com/ScriptReference/Resources.Load.html

Prefabs

Editor Camera Controls

- Maya-like Controls
  - Alt + Left Click & Move : Rotate Camera
  - Alt + Right Click & Move (Or Scroll Up, Down) : Move camera back and forth
  - Alt + Middle Click & Move : Move camera up, down or left, right
- Flythrough Mode
  - Click and hold right mouse button and now you can use FPS-like controls to move around through the scene (WASD, Q/E to move up down).
- Scene View Navigation Documentation
  - https://docs.unity3d.com/Manual/SceneViewNavigation.html

Editor Camera Controls

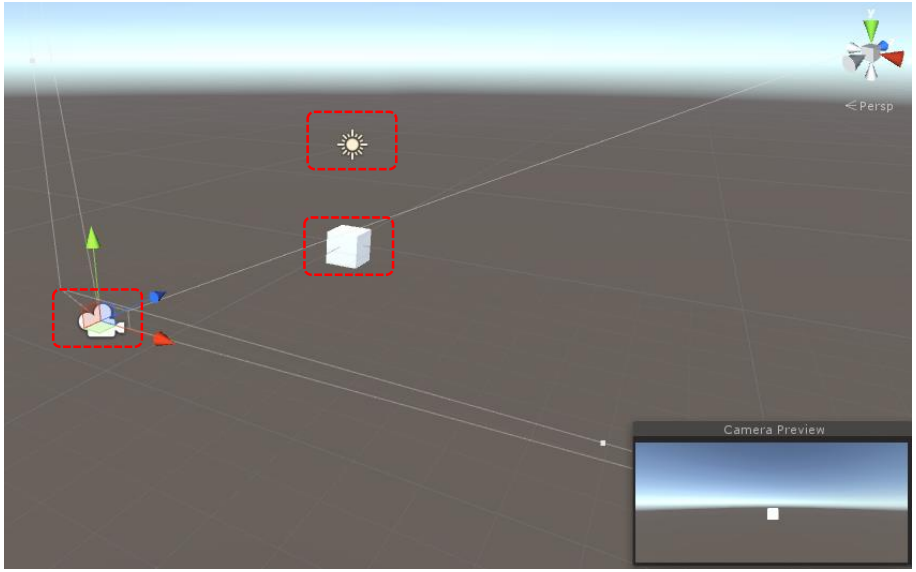
Editor Object Controls

- Maya-like (after selecting an object)
  - W : Activate translation widget
  - E : Activate rotation widget
  - R : Activate scale widget
- Manual Movement
  - Modify position, rotation, scale in the Inspector
  - Rotation is in Euler angles. Rotation order : Z, X, Y
- Positioning GameObjects Documentation
  - https://docs.unity3d.com/Manual/PositioningGameObjects.html

Editor Object Controls

Main Camera

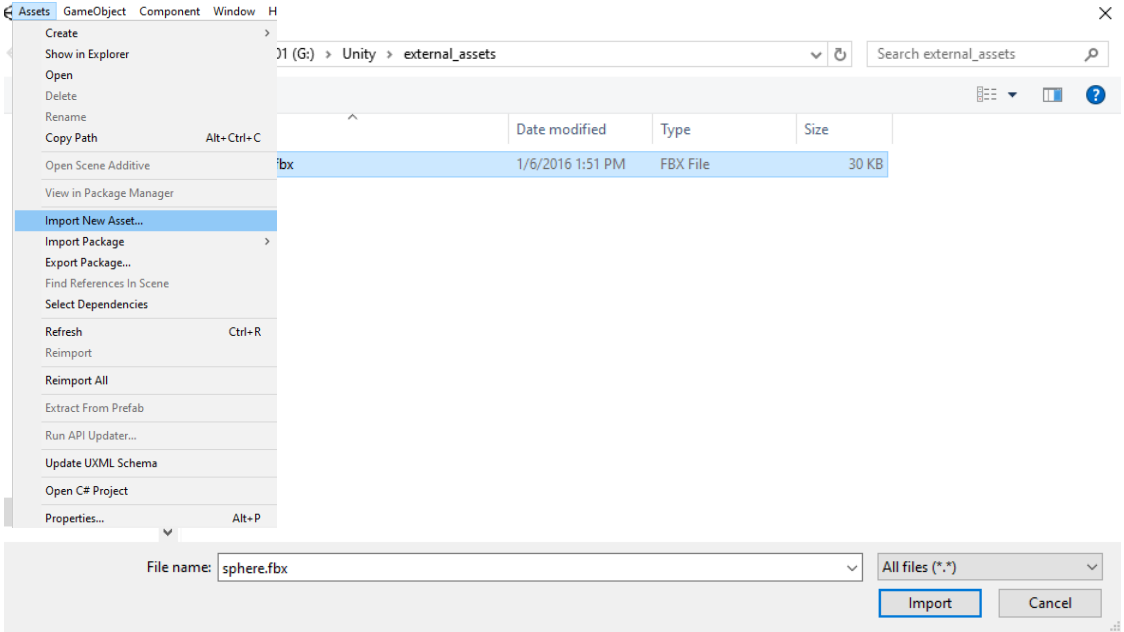
- Unity scenes by default come with a Main Camera. Notice the tag of Main Camera in the inspector, this will be useful for accessing the camera from your scripts.
- Camera Preview box is useful to see what your camera can see. Camera Preview is what you will see when you hit Play!



이미지 출처 :Unity

Import External Objects

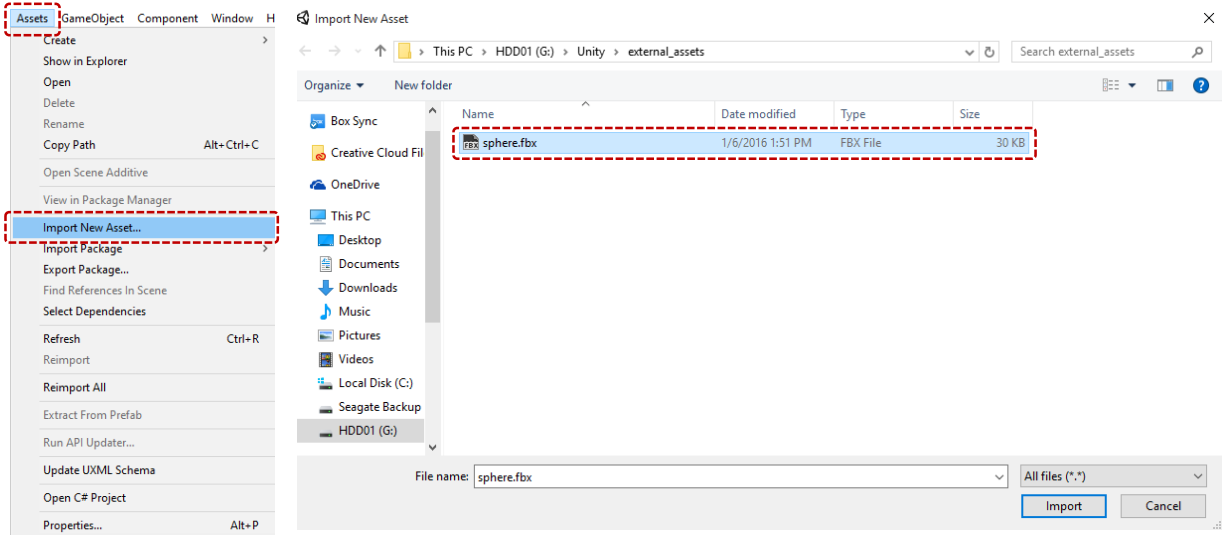
- Create and export an object from Maya, Blender, 3ds Max as either an *.obj or a *.fbx.
- Then import this asset into Unity.



이미지 출처 :Unity

Import External Objects

- Alternatively, you can just save your *.obj or *.fbx inside the Assets folder.
  - You will need to right click on the folder it is in and click “Refresh” to get it to show up.



이미지 출처 :Unity

Shading and Materials

- In most cases you will want to select a shader that comes with Unity and modify the material properties to achieve the look that you want
- Manual Shader Documentation
  - https://docs.unity3d.com/Manual/ShadersOverview.html
- Standard Shader Documentation
  - https://docs.unity3d.com/Manual/shader-StandardShader.html
- Materials Documentation
  - https://docs.unity3d.com/Manual/Materials.html

Shading and Materials

Importing Textures

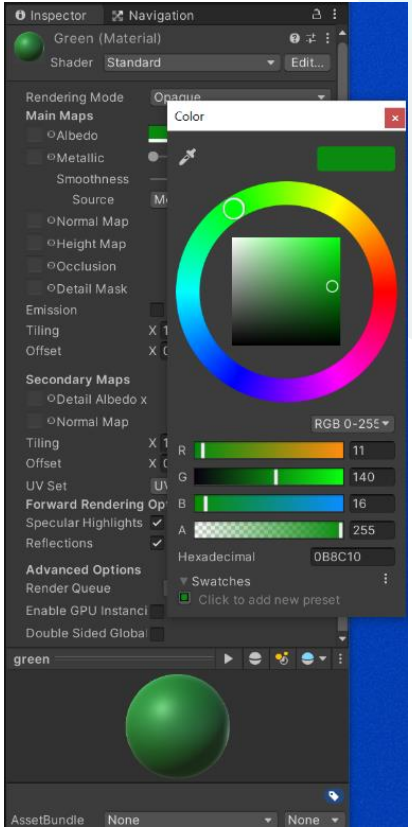
- Importing textures process is the same as importing an external object. This time, select a *.png, *.jpg, etc. You can also place the images inside the Assets folder manually.

- Note that you should make sure your imported object has a UV map before continuing! You can check this in Maya / Blender / 3ds Max.

Importing Textures

Using Textures

- Expand the shader properties in the Inspector.
- Click and drag the imported texture onto the square next to “Albedo” and your object should now have a texture on it!
- Or click on the circle next to “Albedo” and select your texture in the dialog box!



이미지 출처 :Unity

Lighting

- Lighting Documentation
  - https://docs.unity3d.com/Manual/Lighting.html
- Global Illumination Documentation
  - https://docs.unity3d.com/Manual/GlobalIllumination.html
- Lighting is accomplished with the “Light” component.

Lighting

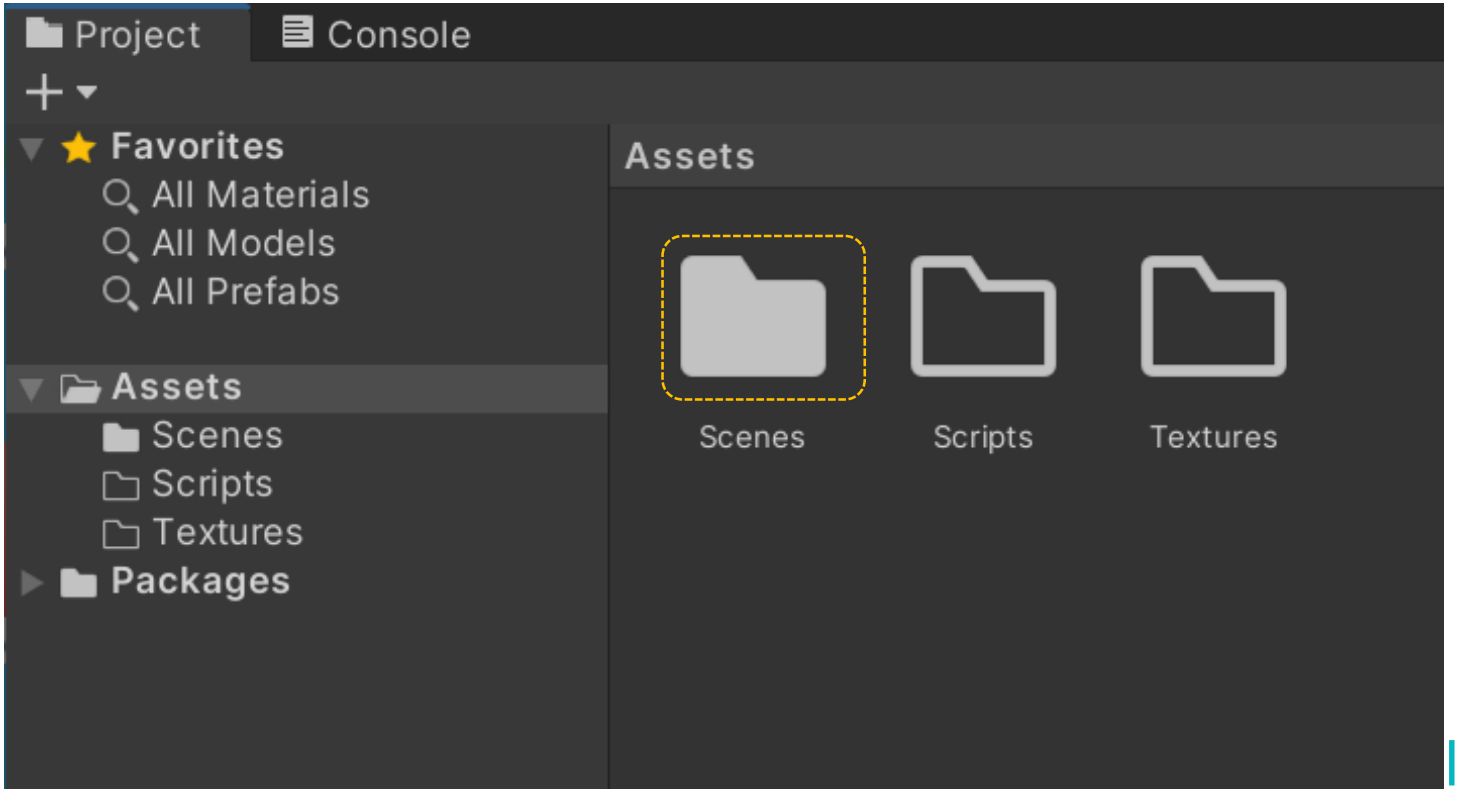
Light Component Properties

- Lights
  - Directional Light
  - Spot Lights
  - Point Light
  - Area Light
- Light Component Properties
  - Color
  - Intensity

Light Component Proper

Create a Scene

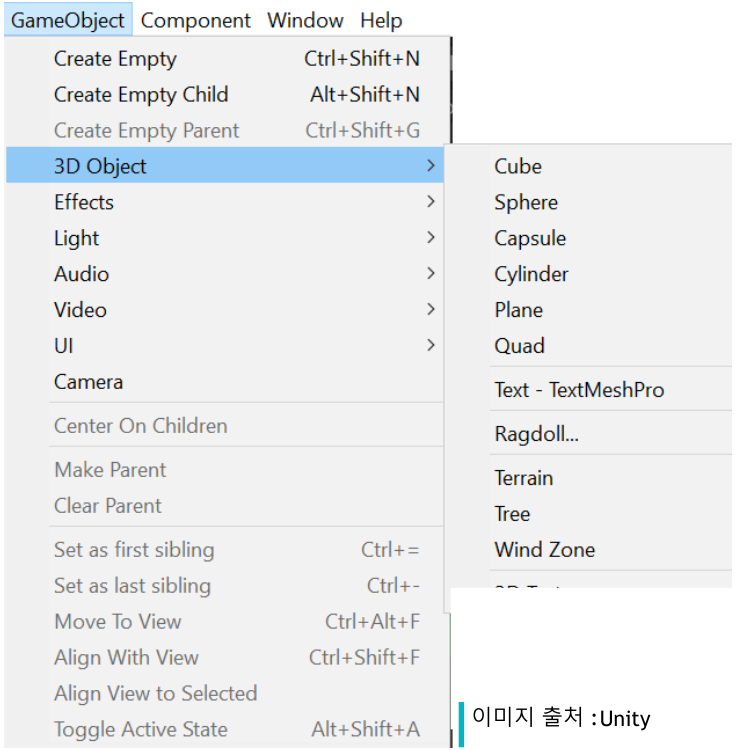
- Create folders (Textures, Materials, Scripts, etc) in project



이미지 출처 :Unity

Creating a Scene

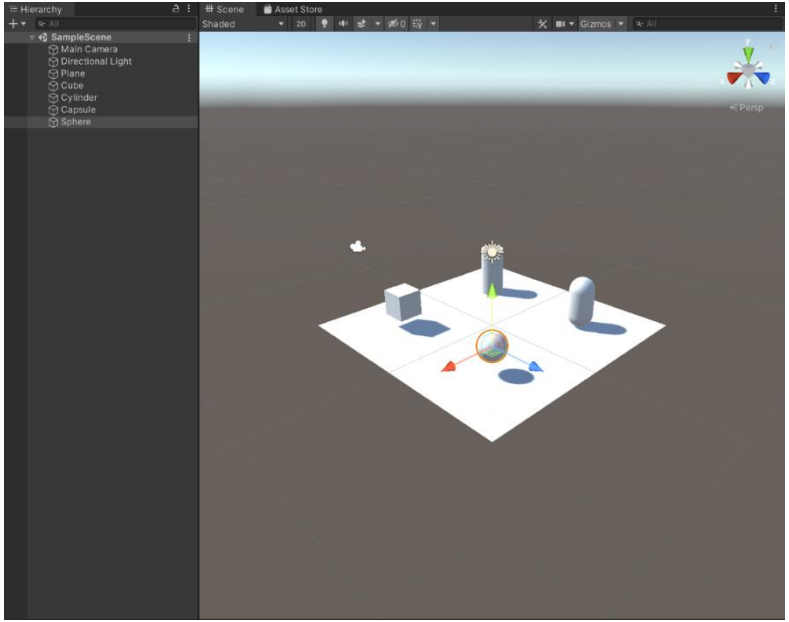
- Create GameObject - GameObject → 3D Object → Plane, Cube, Cylinder, Capsule, Sphere

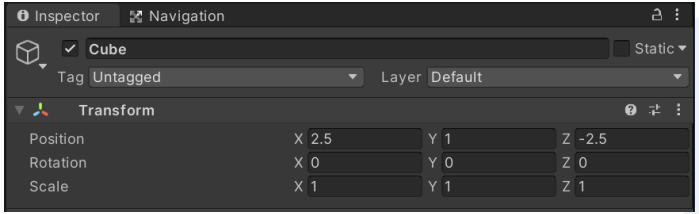


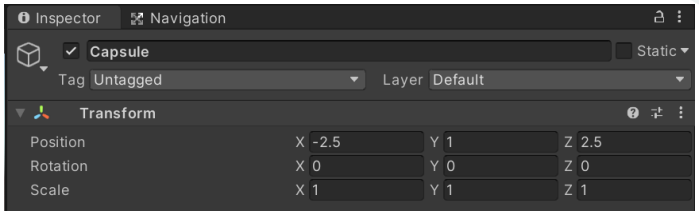
이미지 출처 :Unity

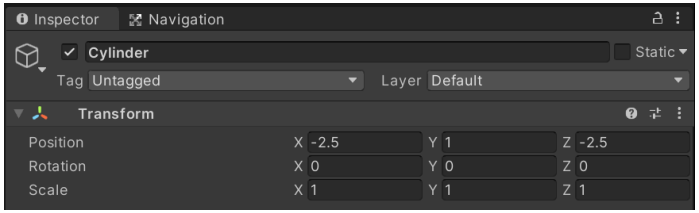
Creating a Scene

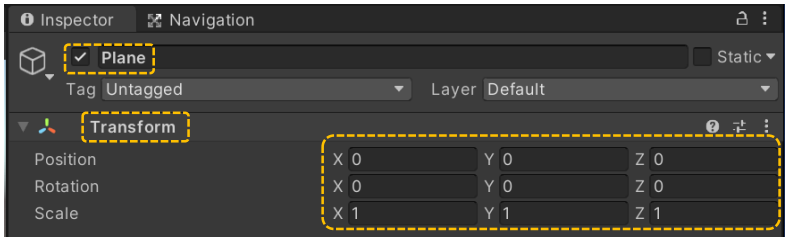
- Set transformation for each object

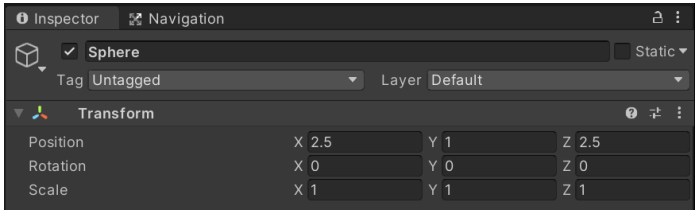








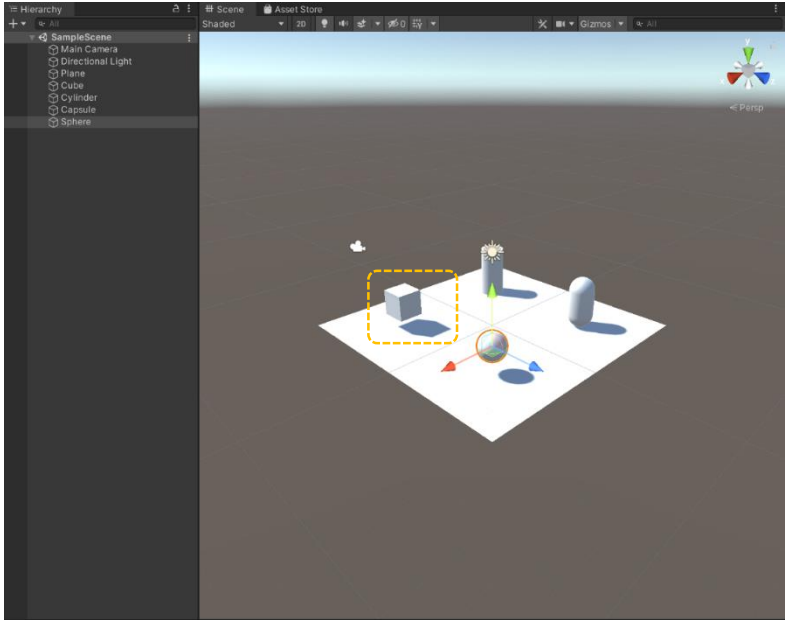


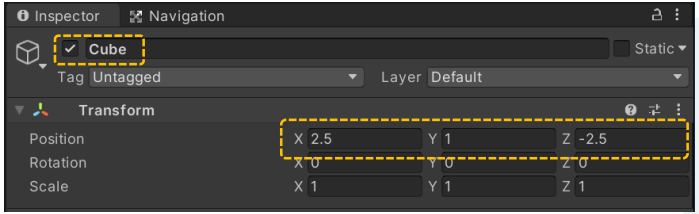


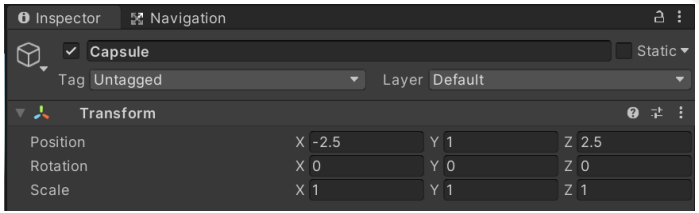
이미지 출처 : Unity

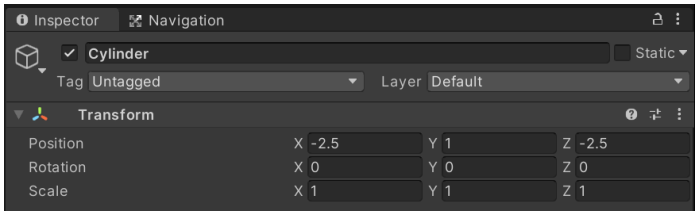
Creating a Scene

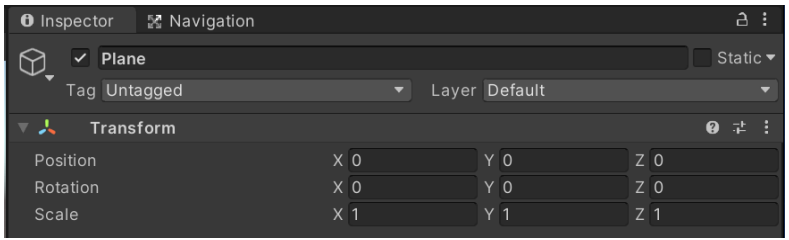
- Set transformation for each object

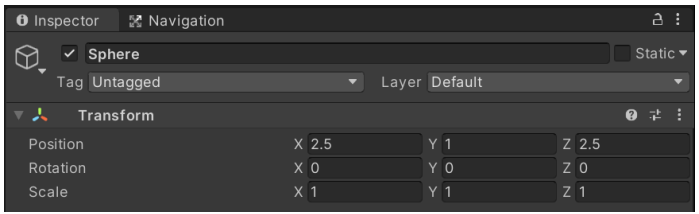








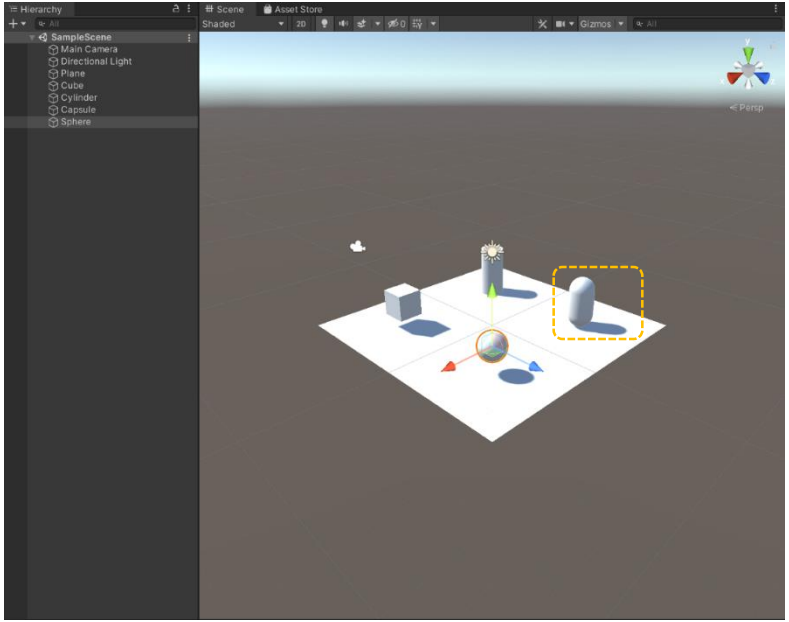


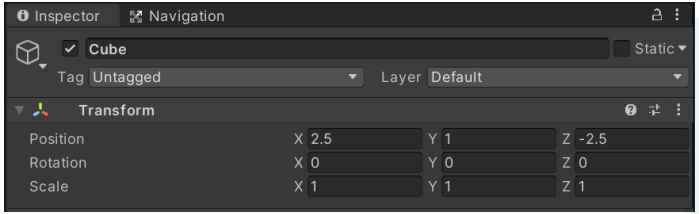


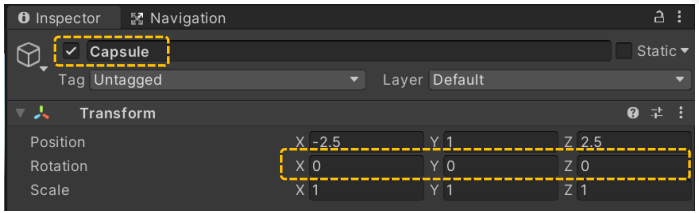
이미지 출처 : Unity

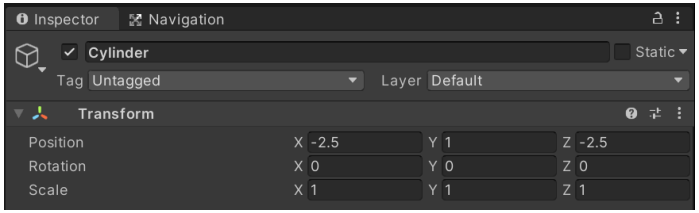
Creating a Scene

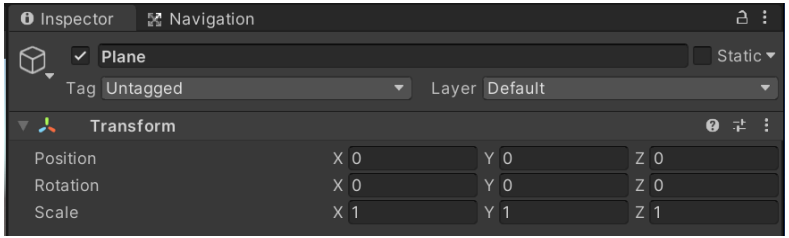
- Set transformation for each object

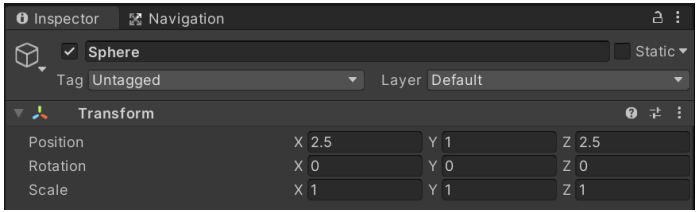






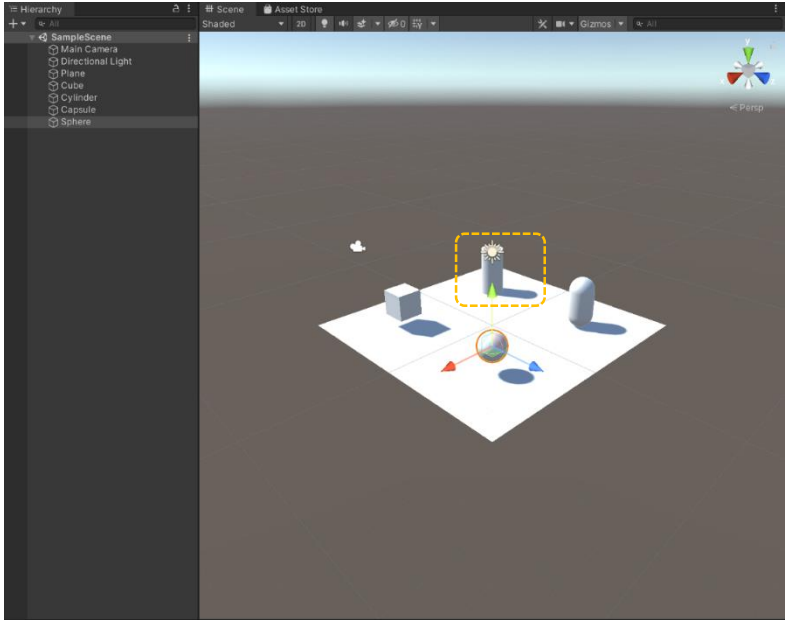


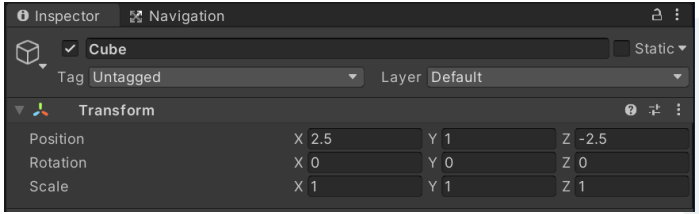


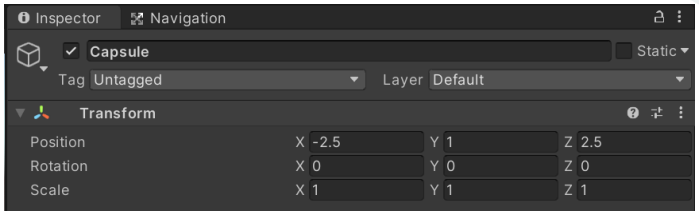


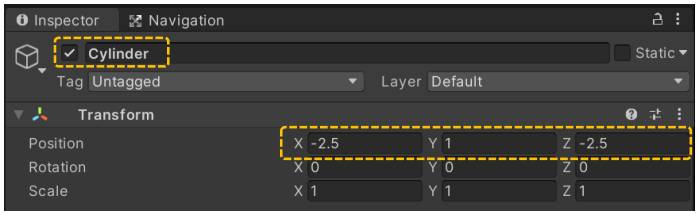
Creating a Scene

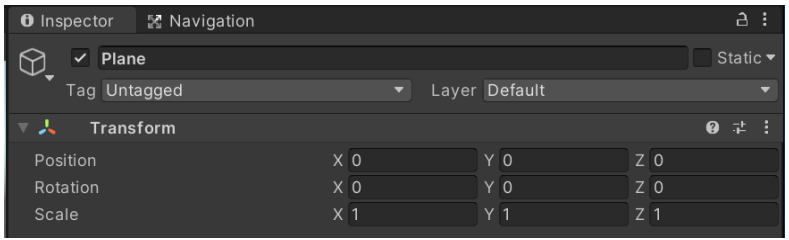
- Set transformation for each object

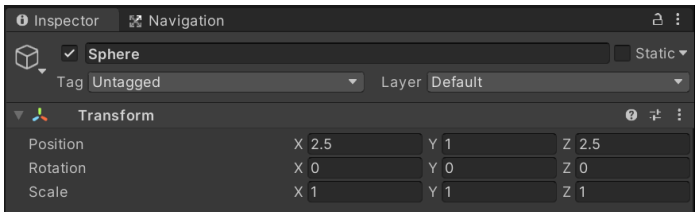








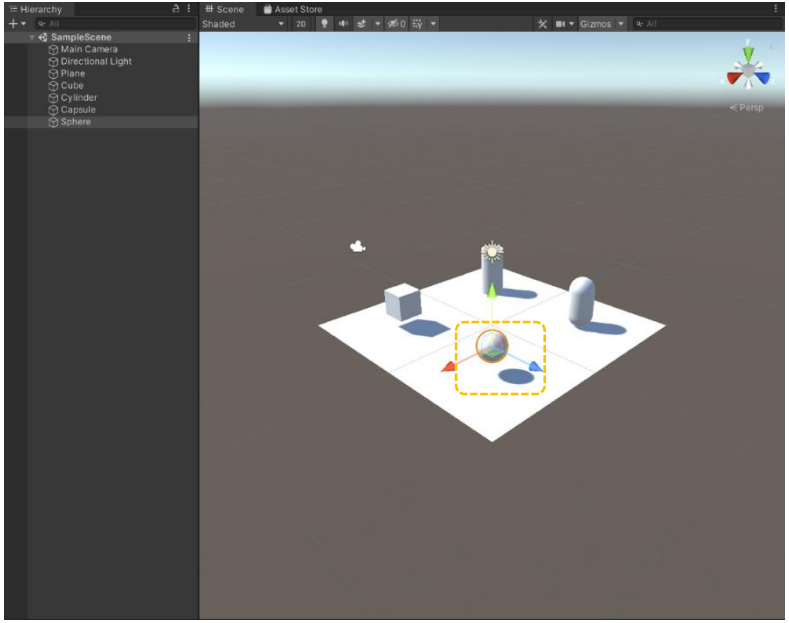


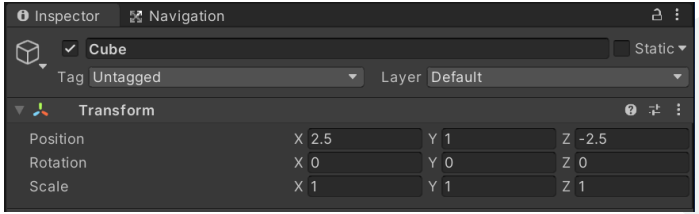


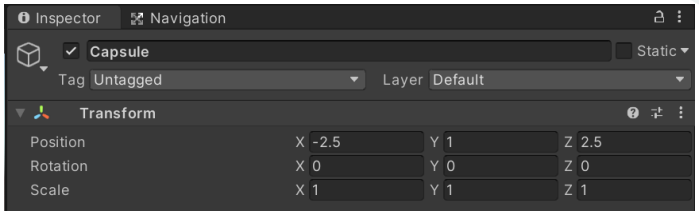
이미지 출처 : Unity

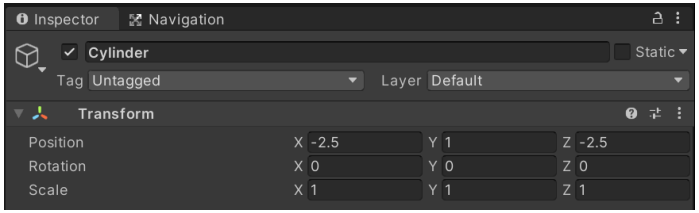
Creating a Scene

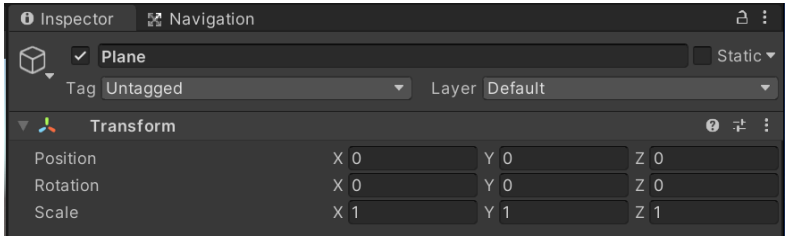
- Set transformation for each object

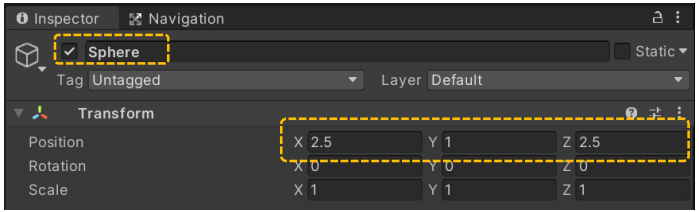








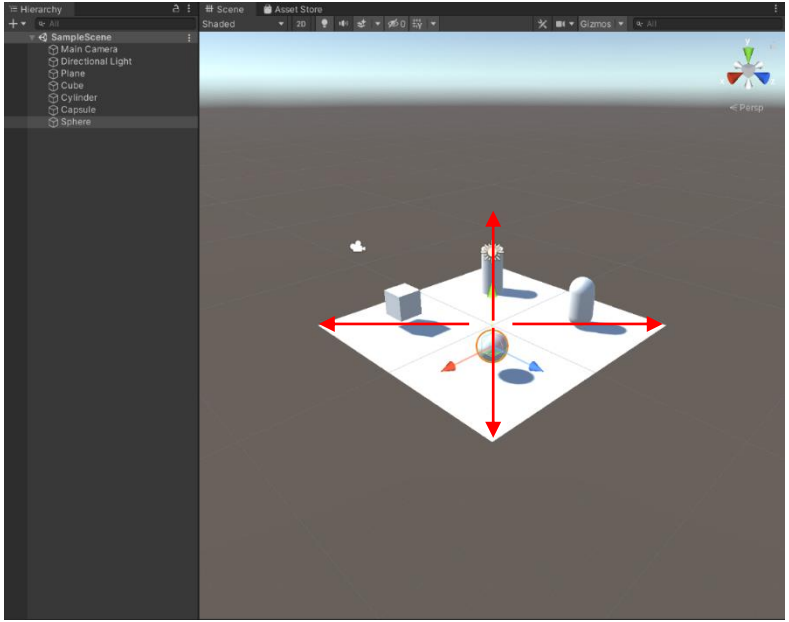


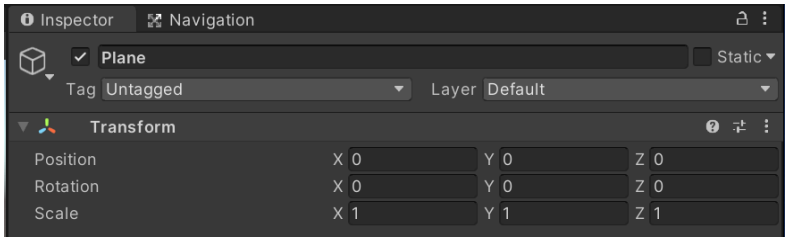


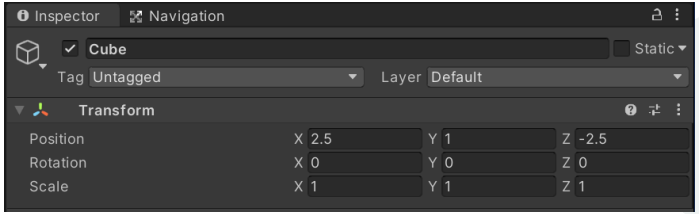
이미지 출처 : Unity

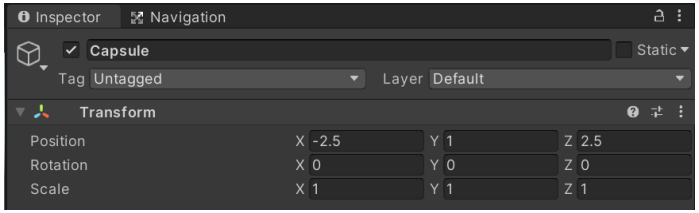
Creating a Scene

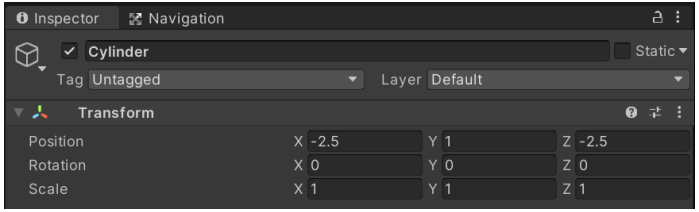
- Set transformation for each object

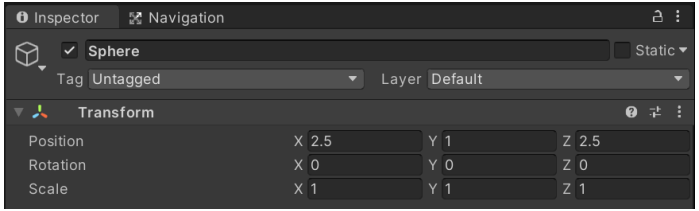








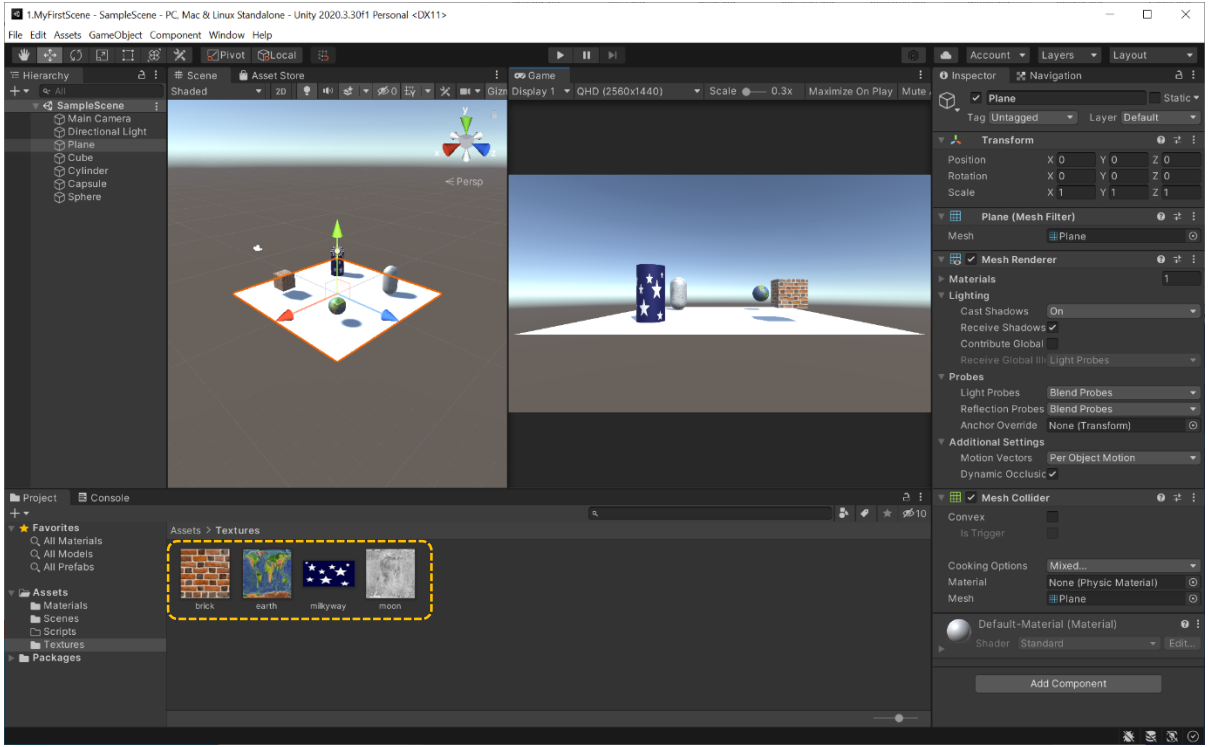




이미지 출처 : Unity

Creating a Scene

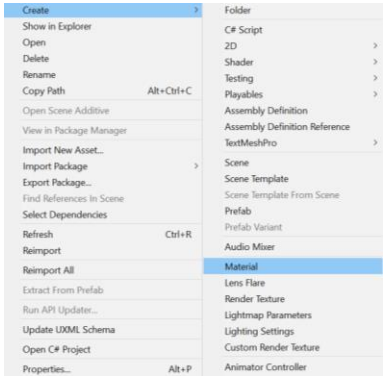
- Set textures - Drag and drop images into Assets/Textures and then drag and drop textures to cube, sphere, etc



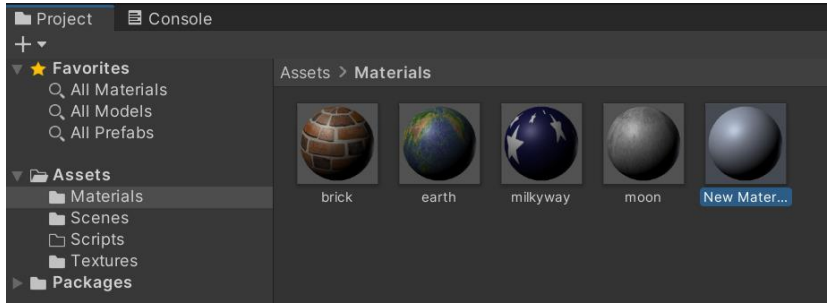
이미지 출처 :Unity

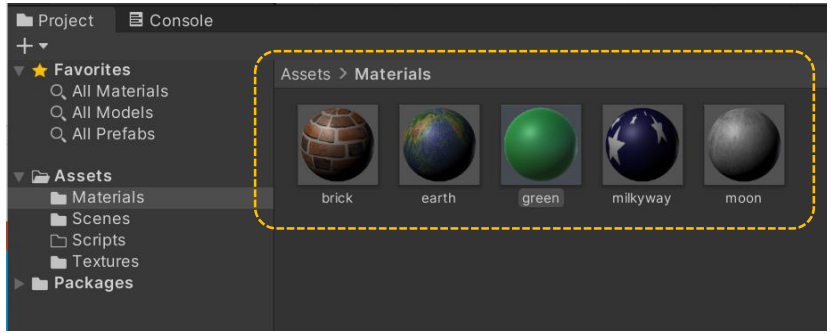
Creating a Scene

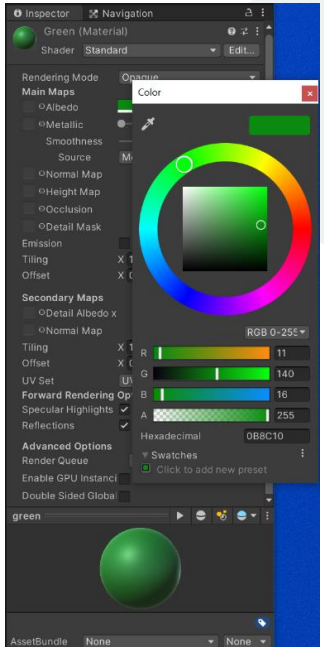
Set materials - Assets → Create → Material



이미지 출처 :Unity

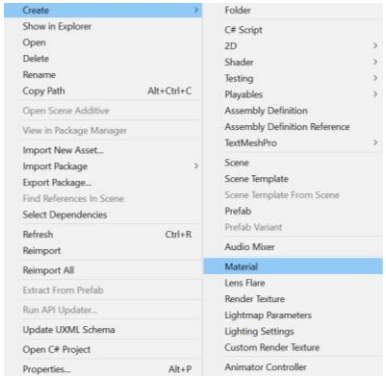




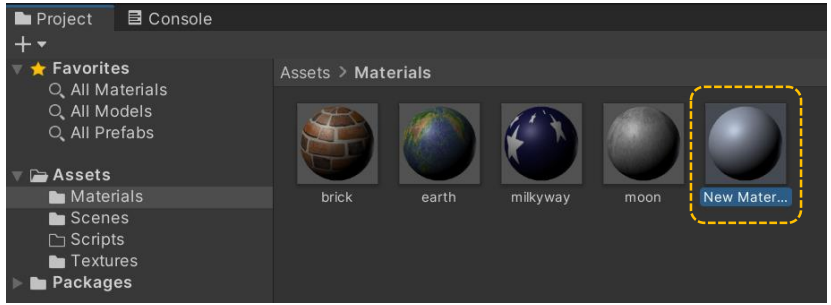


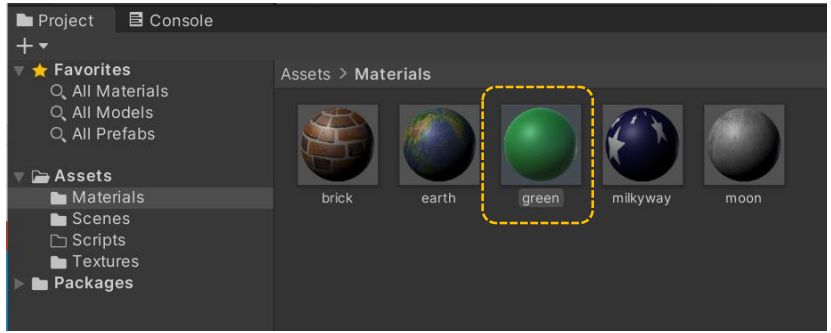
Creating a Scene

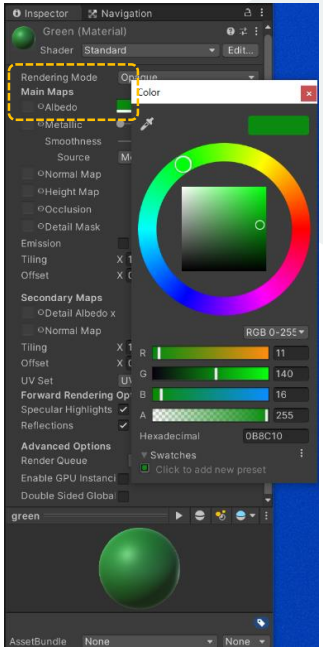
Set materials - Assets → Create → Material



이미지 출처 :Unity

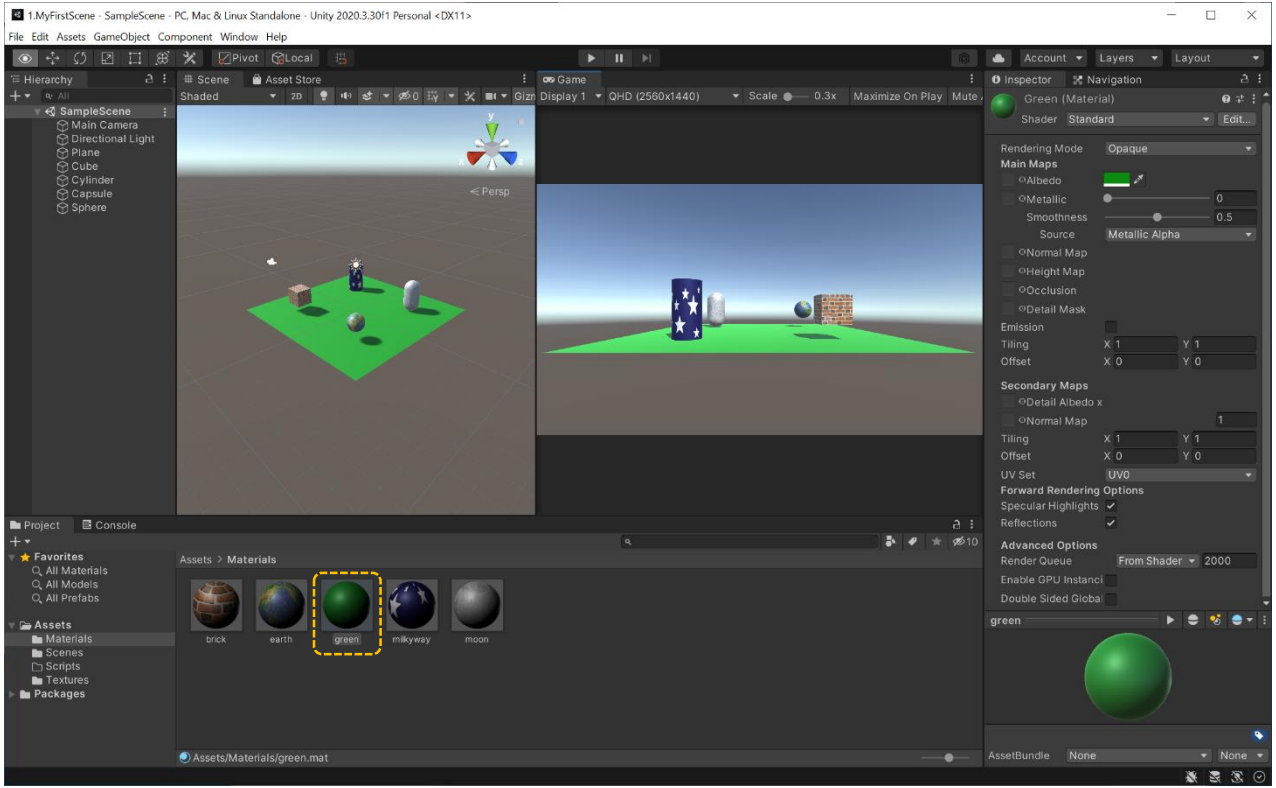






Creating a Scene

- Set materials - drag and drop material onto Plane



이미지 출처 :Unity

Creating a Scene

- Add scripts - Assets → Create → C# Script

Spinner.cs

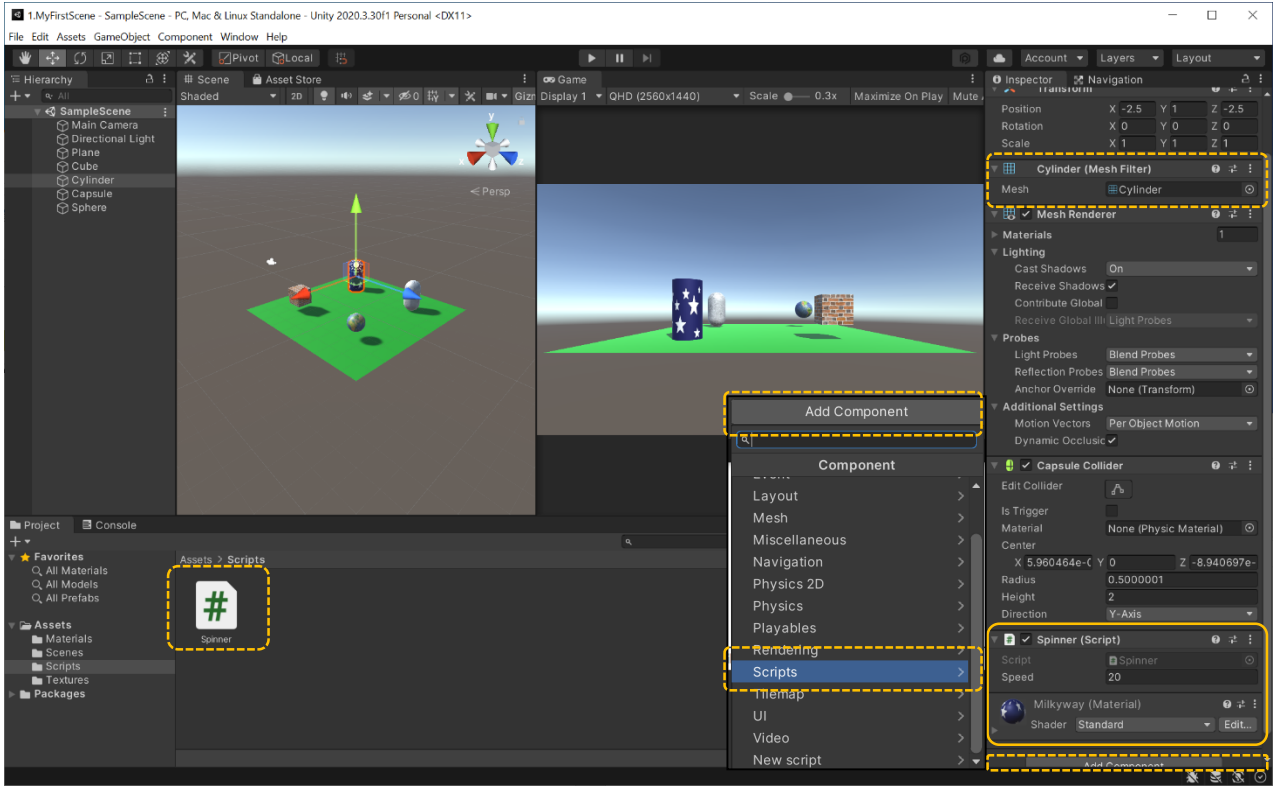
Assembly-CSharp | Spinner | Update()

```
1   using UnityEngine;
2   using System.Collections;
3
    Unity Script | 0 references
4   public class Spinner : MonoBehaviour {
5
6       public float speed;
7
8       // Use this for initialization
        Unity Message | 0 references
9       void Start () {
10
11      }
12
13      // Update is called once per frame
        Unity Message | 0 references
14      void Update () {
15          transform.Rotate(Vector3.up, speed * Time.deltaTime);
16      }
17  }
```

이미지 출처 :Unity

Creating a Scene

- Inspector View - Add Components → Scripts → Spinner



이미지 출처 :Unity

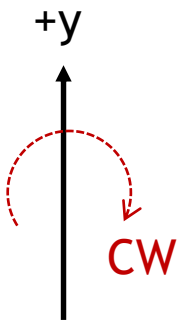
Creating a Scene

- Add scripts - Assets → Create → C# Script

Spinner.cs

Assembly-CSharp | Spinner | Update()

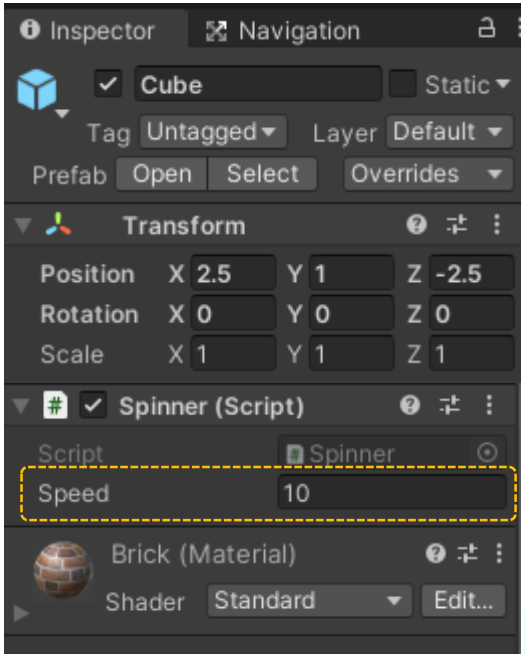
```
1   using UnityEngine;
2   using System.Collections;
3
    Unity Script | 0 references
4   public class Spinner : MonoBehaviour {
5
6       public float speed;
7
8       // Use this for initialization
        Unity Message | 0 references
9       void Start () {
10
11      }
12
13      // Update is called once per frame
        Unity Message | 0 references
14      void Update () {
15          transform.Rotate(Vector3.up, speed * Time.deltaTime);
16      }
17  }
```



이미지 출처 :Unity

Creating a Scene

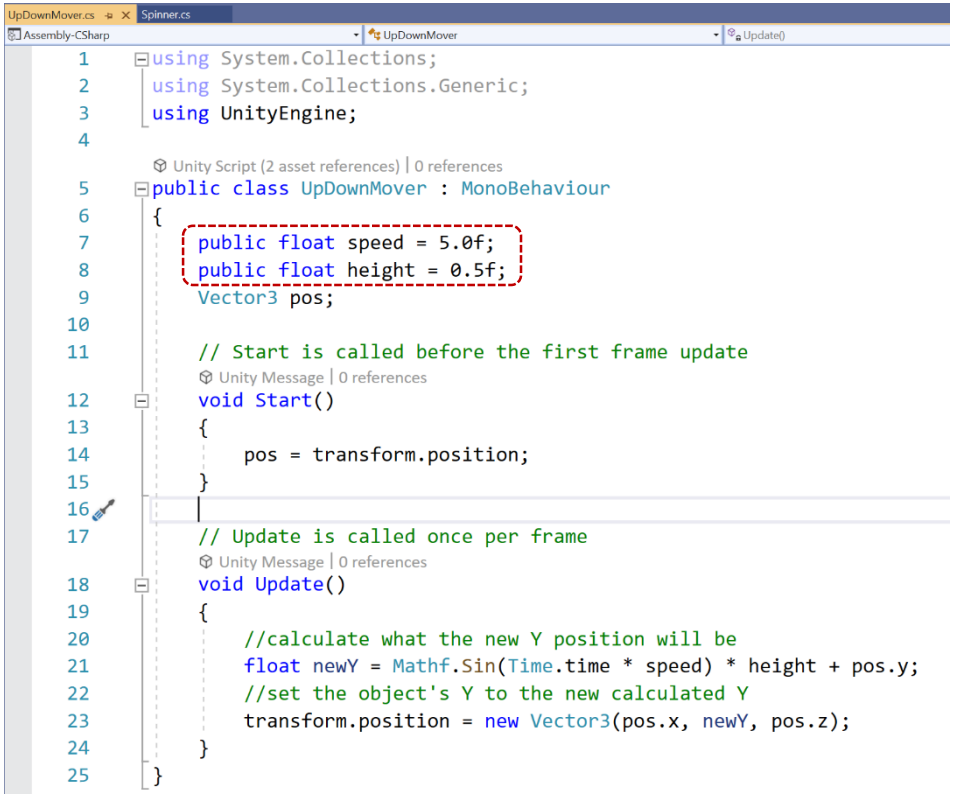
- Public variables in C# script will show up in the Inspector.
- A variable that is a Component can also be modified by the inspector!



이미지 출처 : Unity

Creating a Scene

- Add scripts – Assets → Create → C# Script



이미지 출처 :Unity

Creating a Scene

- Add scripts – Assets → Create → C# Script

UpDownMover.cs | Spinner.cs

Assembly-CSharp | UpDownMover | Update()

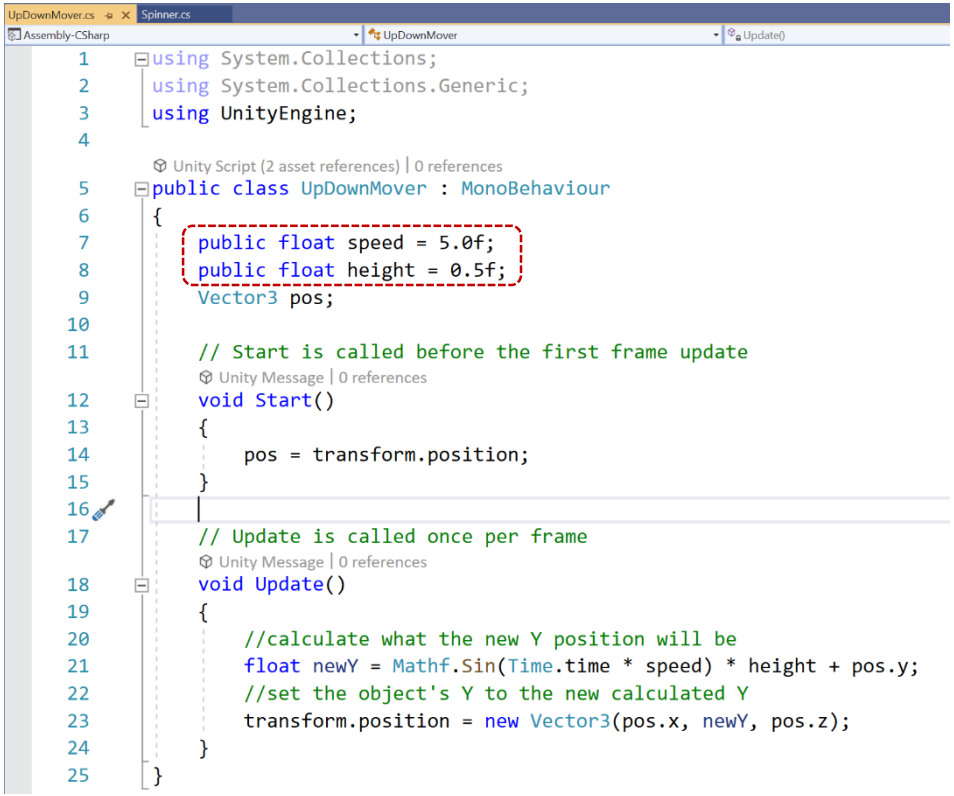
```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
    Unity Script (2 asset references) | 0 references
5   public class UpDownMover : MonoBehaviour
6   {
7       public float speed = 5.0f;
8       public float height = 0.5f;
9       Vector3 pos;
10
11      // Start is called before the first frame update
        Unity Message | 0 references
12      void Start()
13      {
14          pos = transform.position;
15      }
16
17      // Update is called once per frame
        Unity Message | 0 references
18      void Update()
19      {
20          //calculate what the new Y position will be
21          float newY = Mathf.Sin(Time.time * speed) * height + pos.y;
22          //set the object's Y to the new calculated Y
23          transform.position = new Vector3(pos.x, newY, pos.z);
24      }
25  }
```

Position	X	2.5	Y	1	Z -2.5
Rotation	X	0	Y	0	Z 0
Scale	X	1	Y	1	Z 1

이미지 출처 :Unity

Creating a Scene

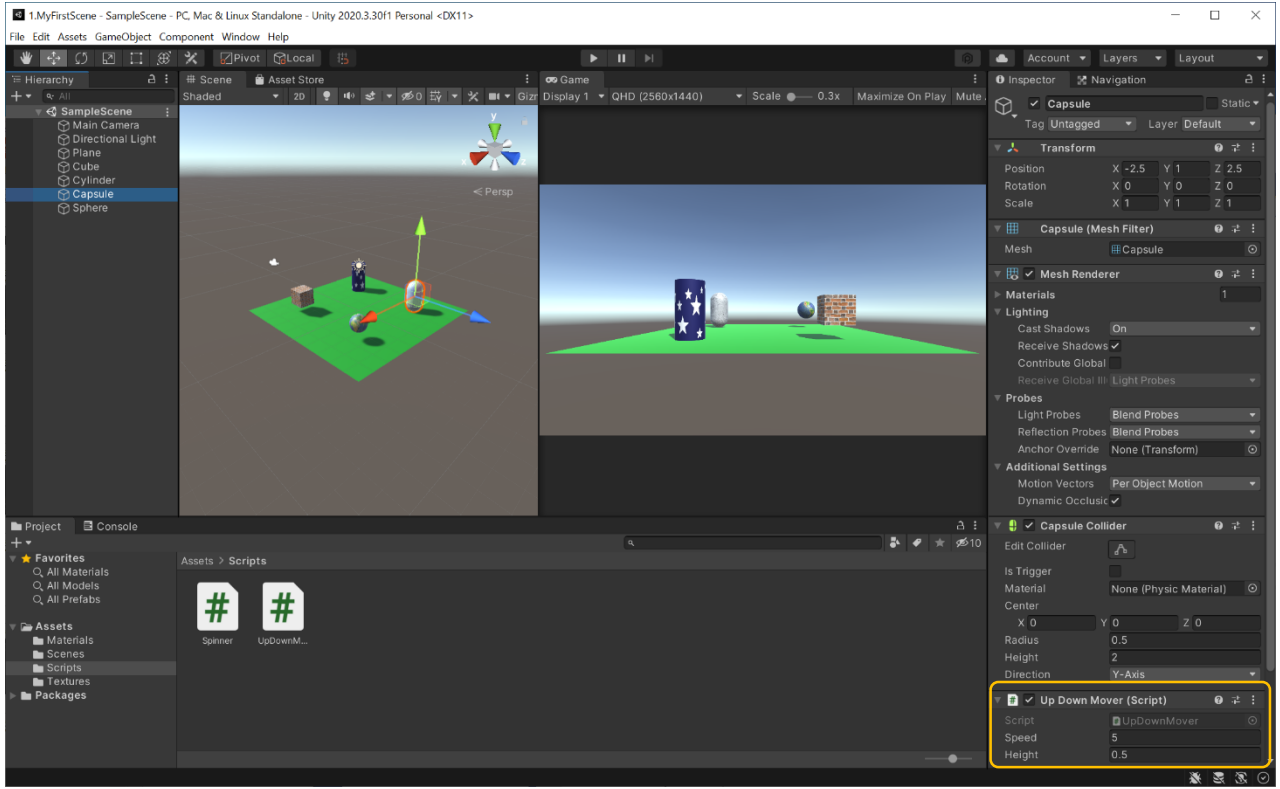
- Add scripts – Assets → Create → C# Script



이미지 출처 :Unity

Creating a Scene

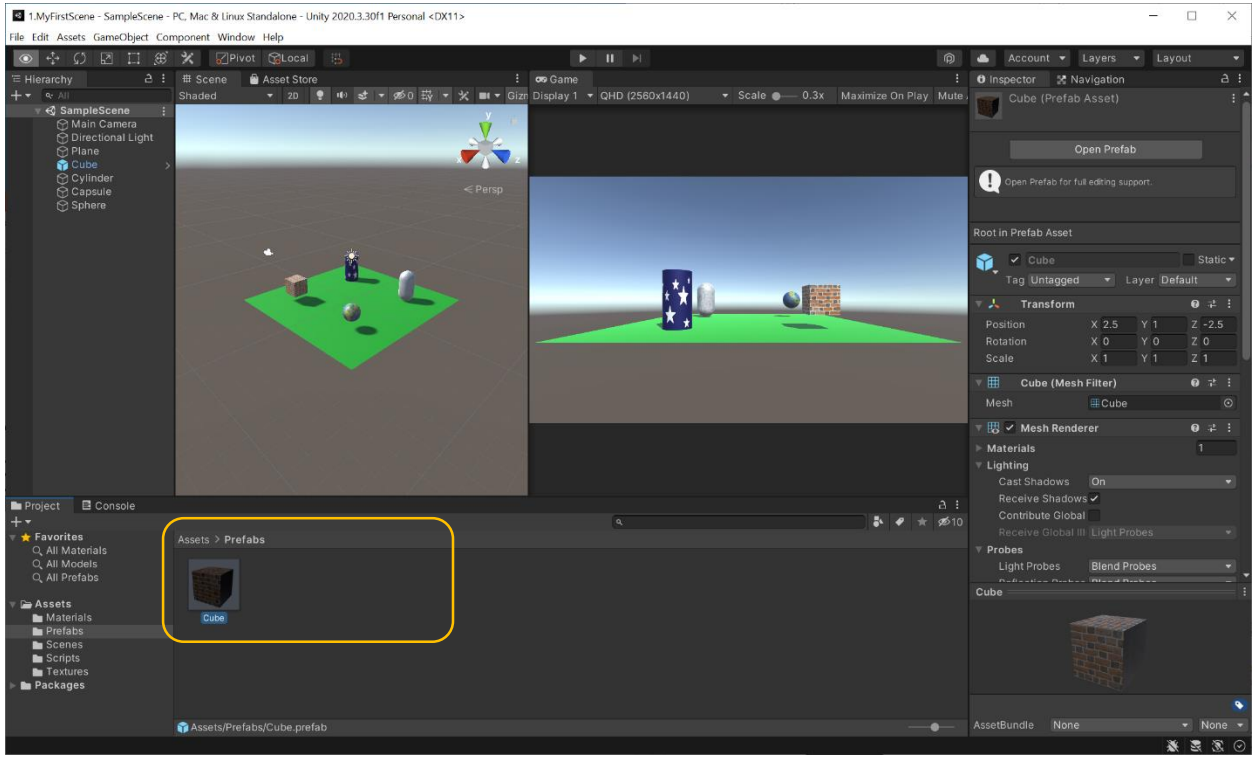
- Inspector View - Add Components → Scripts → UpDownMover



이미지 출처 :Unity

Creating a Scene

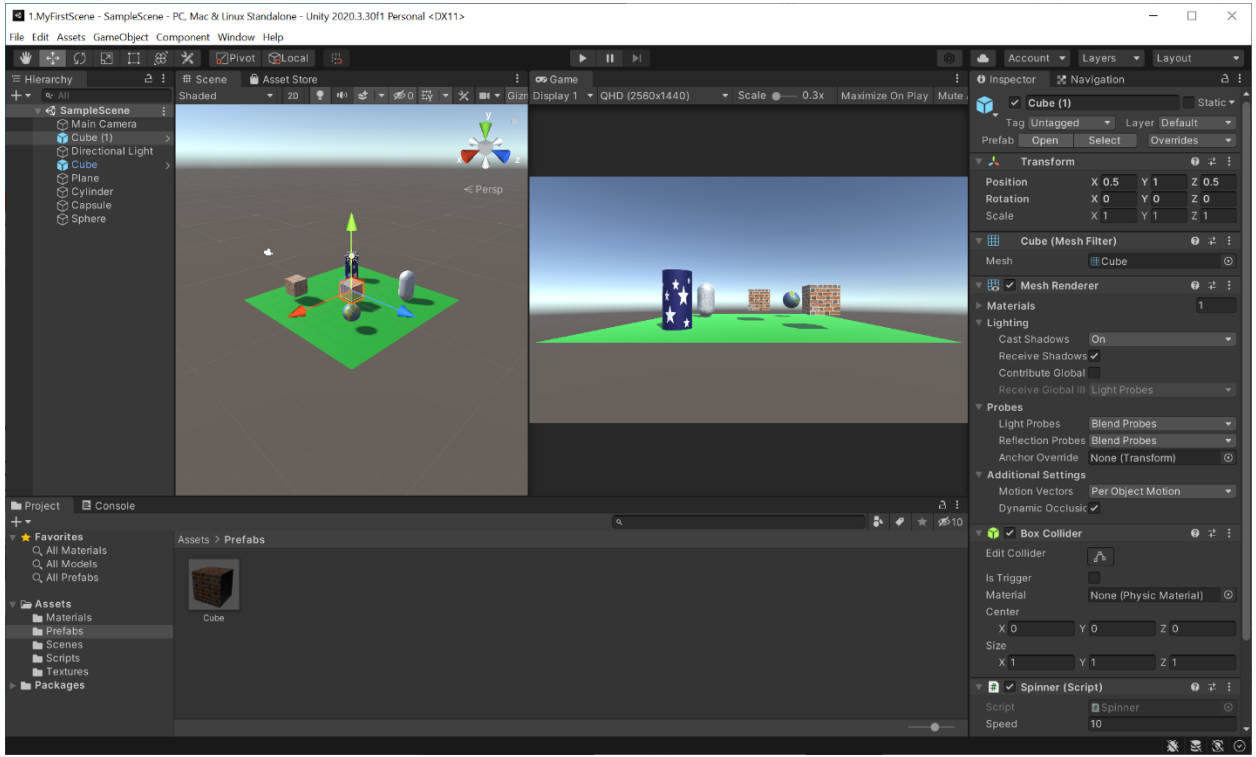
- Create Prefabs - Drag and drop a GameObject from the Hierarchy window into the Project window



이미지 출처 :Unity

Creating a Scene

- Create Prefabs instance - Drag and drop a Prefab into the scene then you will see a Prefab instance created



이미지 출처 :Unity

MonoBehavior Class

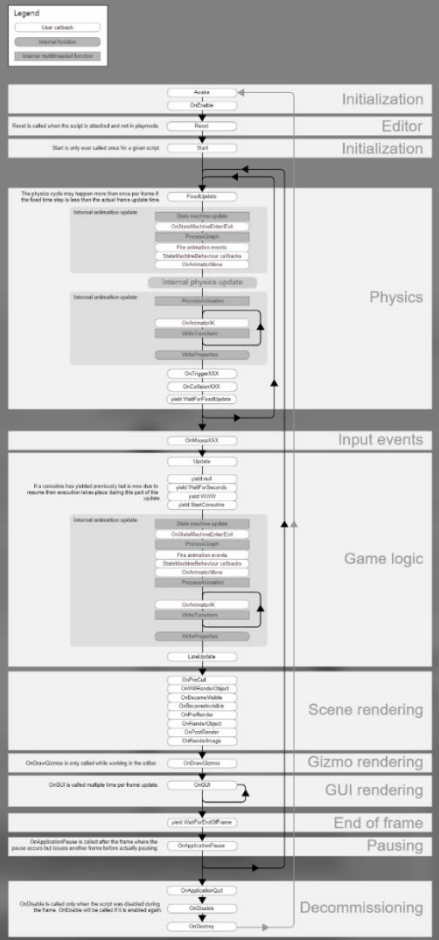
- MonoBehavior is the base class from which every Unity script derives, by default.
- The MonoBehavior class provides the framework which allows you to attach your script to a GameObject in the editor, as well as providing hooks into useful Events such as Start and Update.
- The MonoBehavior class allows you to start, stop, and manage Coroutines, which are a way to write asynchronous code which can include waiting for a certain amount of time, or for certain actions to complete, while allowing other code to continue executing.

MonoBehavior Class

- The MonoBehavior class provides access to a large collection of event messages, which allows you to execute your code based on what is currently happening in your project.

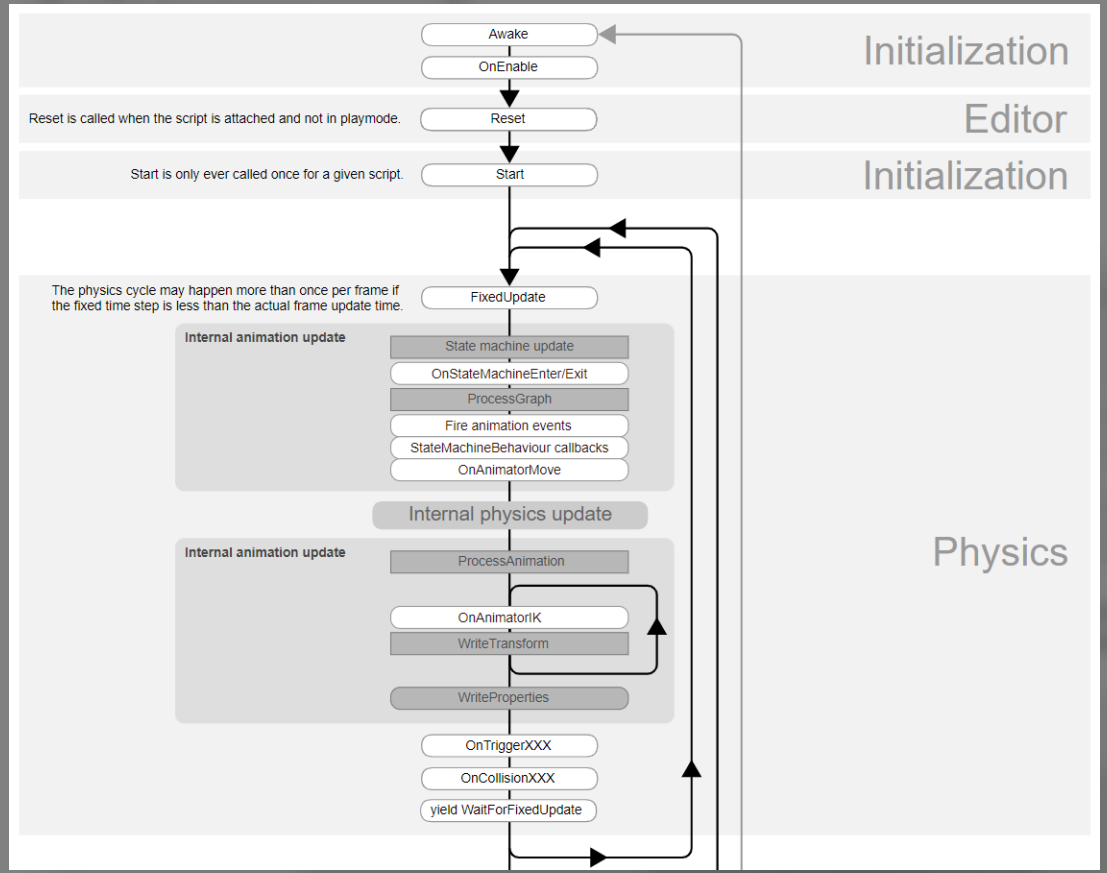
Start	called when the GameObject begins to exist (either when the Scene is loaded, or the GameObject is instantiated).
Update	called every frame.
FixedUpdate	called every physics timestep.
OnBecameVisible and OnBecameInvisible	called when a GameObjec's renderer enters or leaves a camera's view.
OnCollisionEnter and OnTriggerEnter	called when physics **collisions** or **triggers** occur.
OnDestroy	called when the GameObject is destroyed.

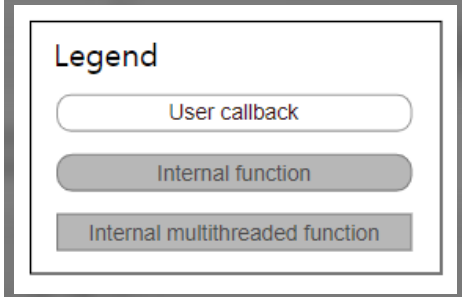
MonoBehavior Class



이미지 출처 : Unity C# Script Lifecycle Flowchart
https://docs.unity3d.com/2019.3/Documentation/Manual/ExecutionOrder.html

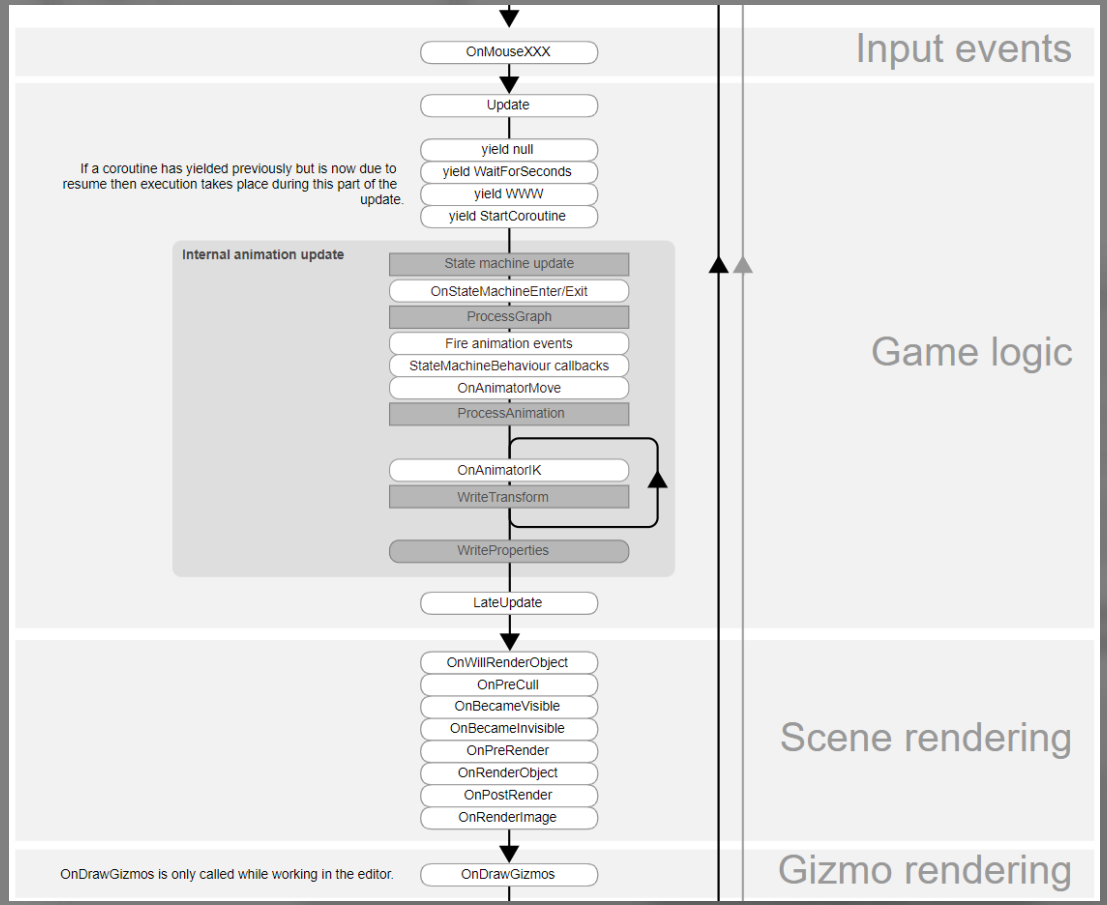
MonoBehavior Class

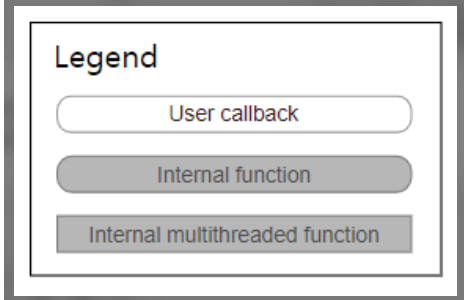




이미지 출처 :Unity C# Script Lifecycle Flowchart
https://docs.unity3d.com/2019.3/Documentation/Manual/ExecutionOrder.html

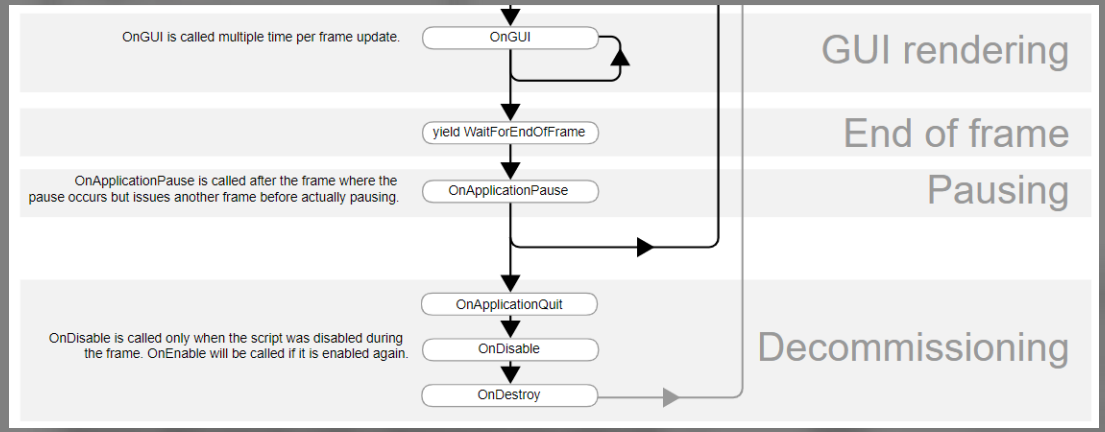
MonoBehavior Class

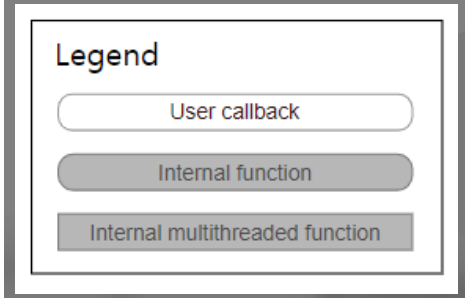




이미지 출처 : Unity C# Script Lifecycle Flowchart
https://docs.unity3d.com/2019.3/Documentation/Manual/ExecutionOrder.html

MonoBehavior Class

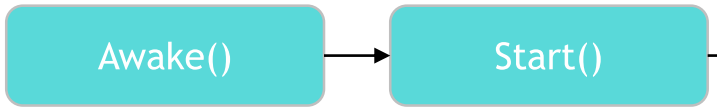




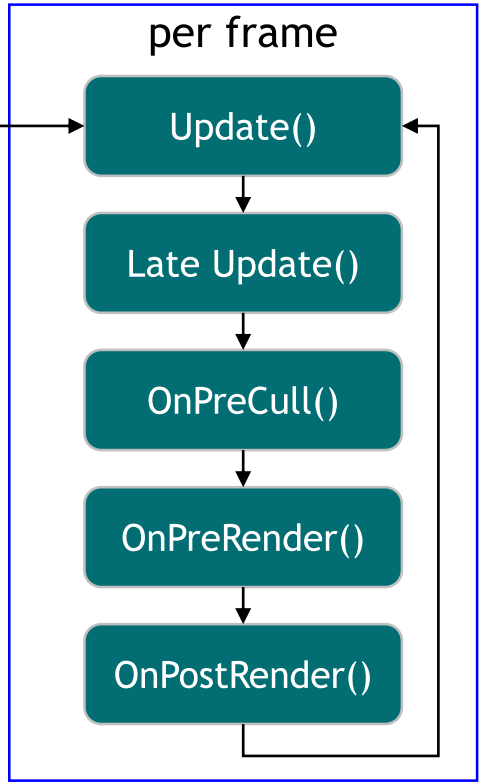
이미지 출처 : Unity C# Script Lifecycle Flowchart
https://docs.unity3d.com/2019.3/Documentation/Manual/ExecutionOrder.html

MonoBehavior Class

- Awake - called before any Start functions and also just after a prefab is instantiated.
- Start - called before the first frame update only if the script instance is enabled.

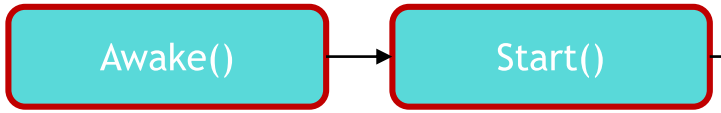


- Update - called once per frame. It is main workhorse function for frame updates.
- LateUpdate - called once per frame, after Update has finished.

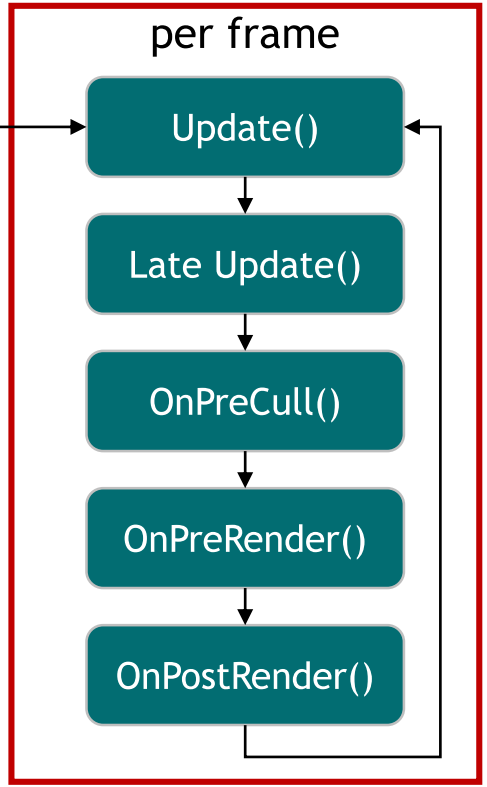


MonoBehavior Class

- Awake - called before any Start functions and also just after a prefab is instantiated.
- Start - called before the first frame update only if the script instance is enabled.

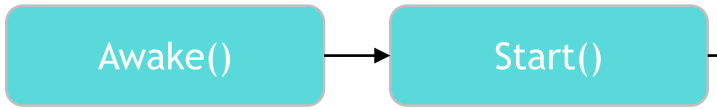


- Update - called once per frame. It is main workhorse function for frame updates.
- LateUpdate - called once per frame, after Update has finished.

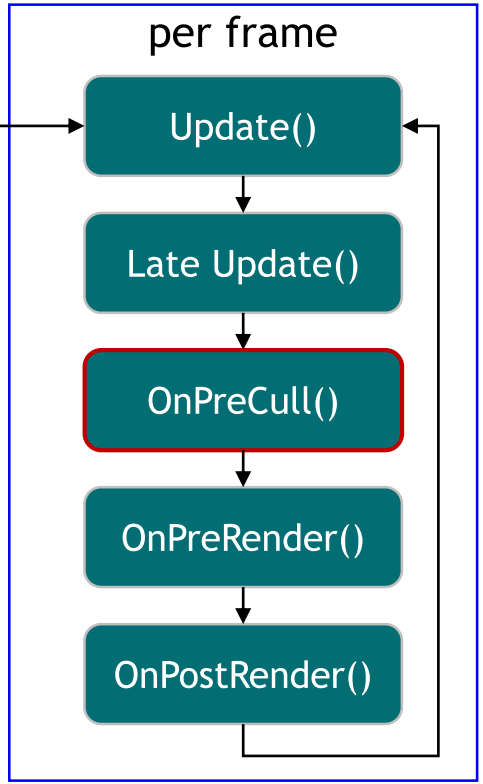


MonoBehavior Class

- Awake - called before any Start functions and also just after a prefab is instantiated.
- Start - called before the first frame update only if the script instance is enabled.

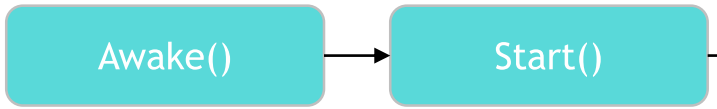


- Update - called once per frame. It is main workhorse function for frame updates.
- LateUpdate - called once per frame, after Update has finished.

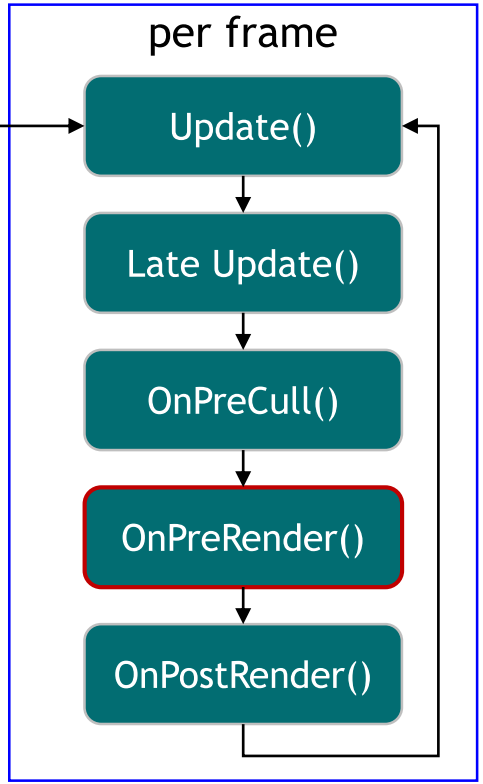


MonoBehavior Class

- Awake - called before any Start functions and also just after a prefab is instantiated.
- Start - called before the first frame update only if the script instance is enabled.

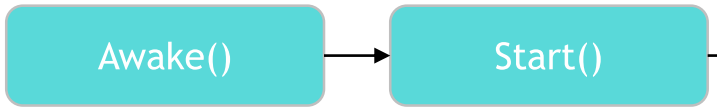


- Update - called once per frame. It is main workhorse function for frame updates.
- LateUpdate - called once per frame, after Update has finished.

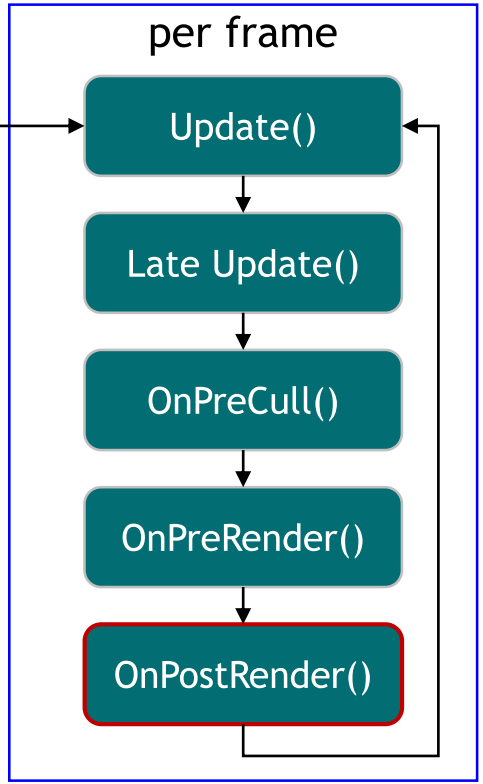


MonoBehavior Class

- Awake - called before any Start functions and also just after a prefab is instantiated.
- Start - called before the first frame update only if the script instance is enabled.



- Update - called once per frame. It is main workhorse function for frame updates.
- LateUpdate - called once per frame, after Update has finished.



Get Components

- GetComponent<Component>()
  - GetComponent<Transform>().Rotate(0.0f, speed * Time.deltaTime, 0.0f);
- Use Component field
  - transform.Rotate(0.0f, speed * Time.deltaTime, 0.0f);

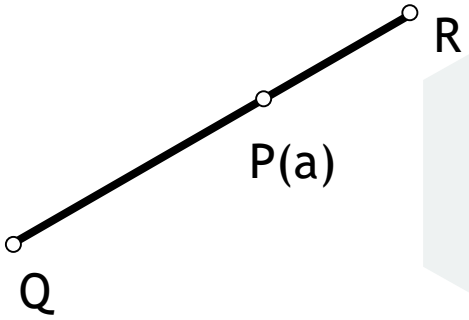
Get Components

4

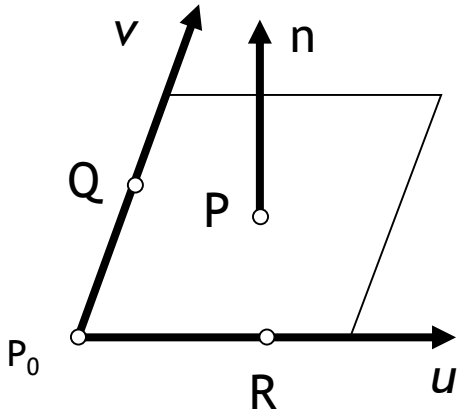
Geometry for Game Programming and Graphics

Geometry for Graphics

- Line
  - 2 points: R, Q
  - v = R- Q
  - P = Q + αv = Q + α(R - Q) = αR + (1 – α)Q
  - P = α_1R + α_2Q where α_1 + α_2 = 1 (affine sum)

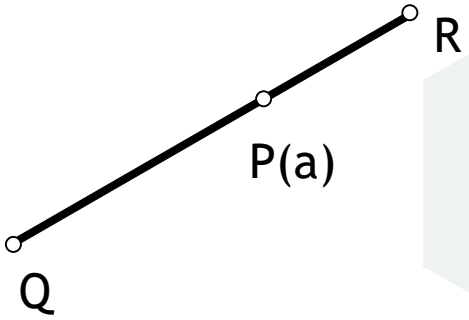


- Plane
  - 3 points: P_0, Q, R
  - T(α, β) = P + αu + βv
  - n•(P – P_0) = 0 where n = u x v

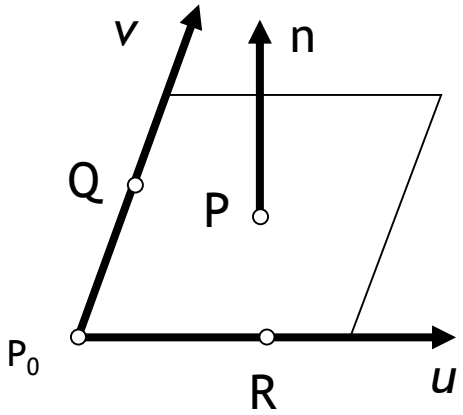


Geometry for Graphics

- Line
  - 2 points: R, Q
  - v = R- Q
  - $P = Q + \alpha v = Q + \alpha(R - Q) = \alpha R + (1 - \alpha)Q$
  - $P = \alpha_1 R + \alpha_2 Q$ where $\alpha_1 + \alpha_2 = 1$ (affine sum)



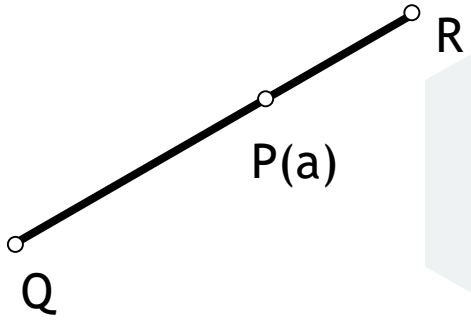
- Plane
  - 3 points: P_0, Q, R
  - $T(\alpha, \beta) = P + \alpha u + \beta v$
  - $n \bullet (P - P_0) = 0$ where n = u x v

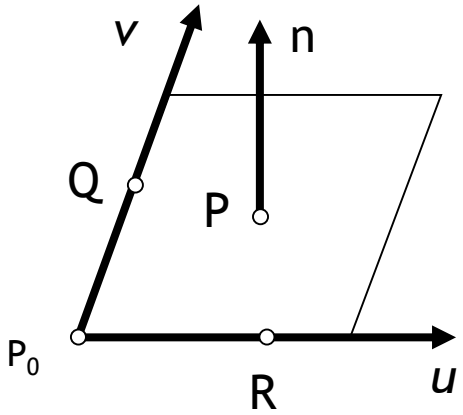


Geometry for Graphics

- Line
  - 2 points: R, Q
  - v = R- Q
  - P = Q + αv = Q + α(R - Q) = αR + (1 – α)Q
  - P = α_1R + α_2Q where $\alpha_1 + \alpha_2 = 1$ (affine sum)

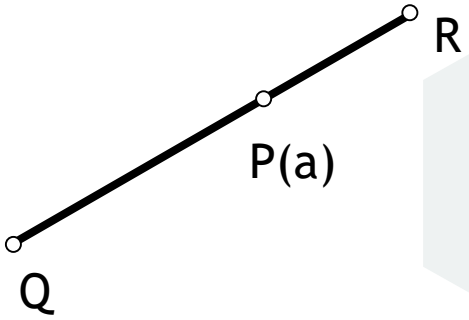
    $\alpha_1, \alpha_2 > 0$
- Plane
  - 3 points: P_0, Q, R
  - T(α, β) = P + αu + βv
  - n•(P – P_0) = 0 where n = u x v



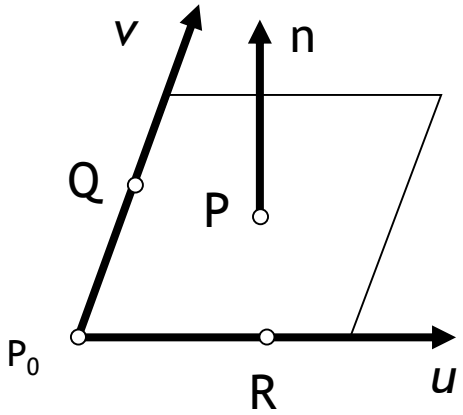


Geometry for Graphics

- Line
  - 2 points: R, Q
  - v = R- Q
  - P = Q + αv = Q + α(R - Q) = αR + (1 – α)Q
  - P = α_1R + α_2Q where α_1 + α_2 = 1 (affine sum)

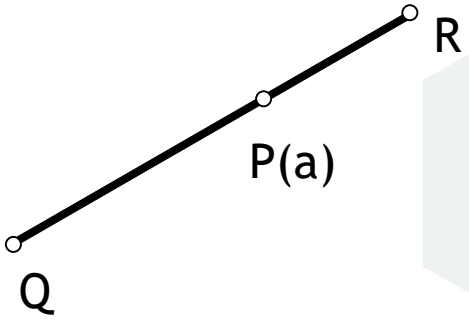


- Plane
  - 3 points: P_0, Q, R
  - T(α, β) = P + αu + βv
  - n•(P – P_0) = 0 where n = u x v

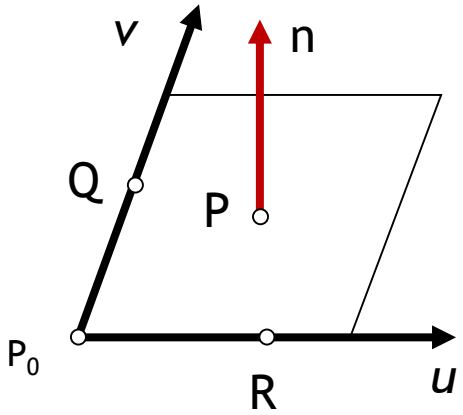


Geometry for Graphics

- Line
  - 2 points: R, Q
  - v = R- Q
  - $P = Q + \alpha v = Q + \alpha(R - Q) = \alpha R + (1 - \alpha)Q$
  - $P = \alpha_1 R + \alpha_2 Q$ where $\alpha_1 + \alpha_2 = 1$ (affine sum)

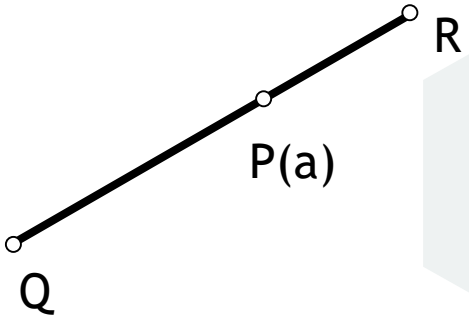


- Plane
  - 3 points: P_0, Q, R
  - $T(\alpha, \beta) = P + \alpha u + \beta v$
  - $n \bullet (P - P_0) = 0$ where n = u x v

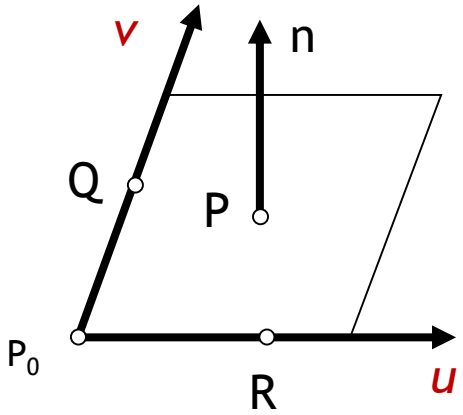


Geometry for Graphics

- Line
  - 2 points: R, Q
  - v = R- Q
  - $P = Q + \alpha v = Q + \alpha(R - Q) = \alpha R + (1 - \alpha)Q$
  - $P = \alpha_1 R + \alpha_2 Q$ where $\alpha_1 + \alpha_2 = 1$ (affine sum)



- Plane
  - 3 points: P_0, Q, R
  - $T(\alpha, \beta) = P + \alpha u + \beta v$
  - $n \bullet (P - P_0) = 0$ where n = u x v



Geometry for Graphics

- 3D objects
  - It is a set of vertices in three dimensional space.
  - It is described by the surface, and is hollow.
  - It can be composed of convex polygons.
  - An arbitrary polygon is divided into triangular polygons, i.e., tessellate.

Geometry for Graphics

- Transformation
  - Position, rotation, scale
- Geometric intersections
  - Hit object
- Orientations
  - Euler angles, quaternion
- Change of coordinates
  - LCS, WCS, VCS, ..

Geometry for Graphics

- Reflection and refractions
  - Light reflects off of shiny objects
  - Light refracts through transparent objects
- Geometric interpolation
  - Interpolate between two different positions and two different rotations in space

Geometry for Graphics

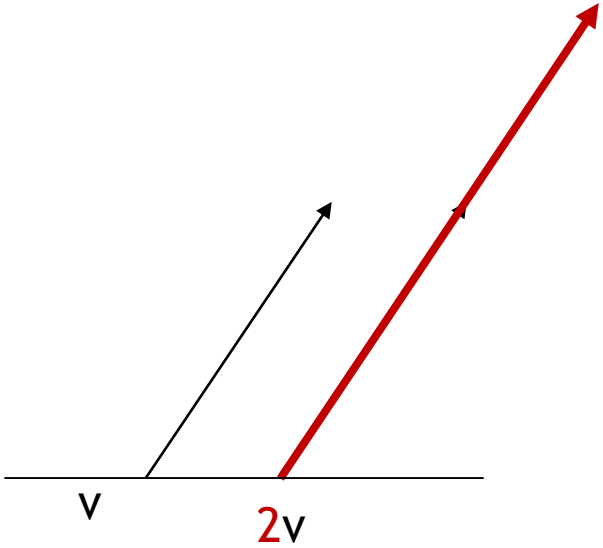
- Affine Geometry
  - Points, lines, planes, line segments, triangles, etc.
- Euclidean Geometry
  - Extension of Affine Geometry which includes distance, angles, orientations
- Projective Geometry
  - In graphics, we often need to deal with infinity. Projective geometry permits notion of infinity.

Geometry for Graphics

- Affine Geometry is basic to all geometric processing.
  - Scalars - real numbers
  - Points - define locations in space
  - Vectors - specify direction and magnitude but have no fixed position

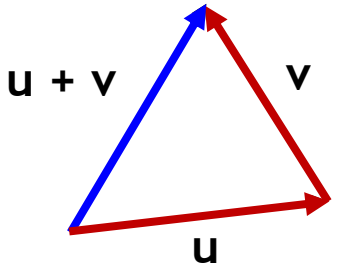
Geometry for Graphics

- Affine Operations
  - Scalar-vector multiplication is vector



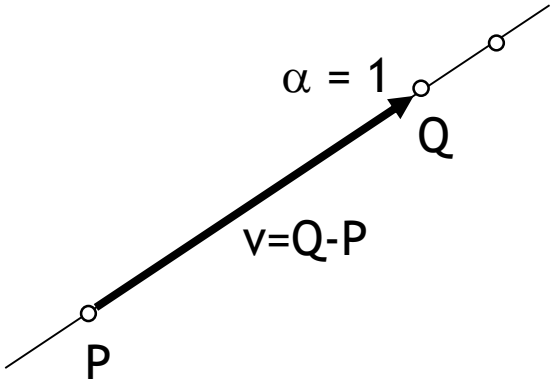
Geometry for Graphics

- Affine Operations
  - Scalar-vector multiplication is vector
  - Vector-vector addition is vector



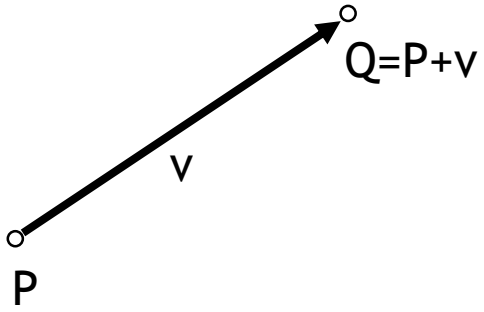
Geometry for Graphics

- Affine Operations
  - Scalar-vector multiplication is vector
  - Vector-vector addition is vector
  - Point-point difference is vector



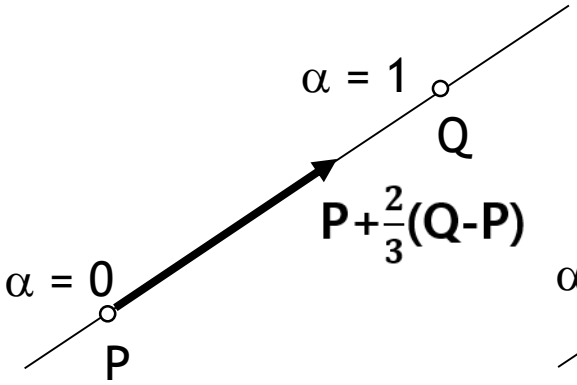
Geometry for Graphics

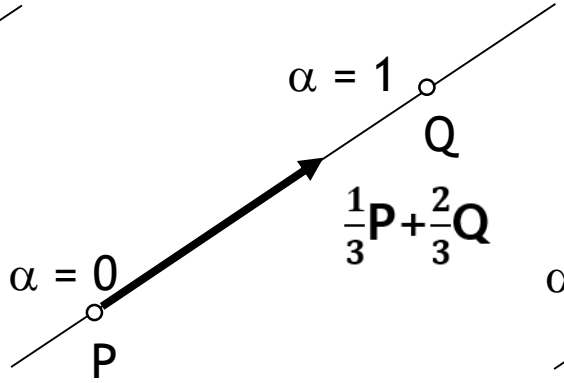
- Affine Operations
  - Scalar-vector multiplication is vector
  - Vector-vector addition is vector
  - Point-point difference is vector
  - Point-vector addition is point

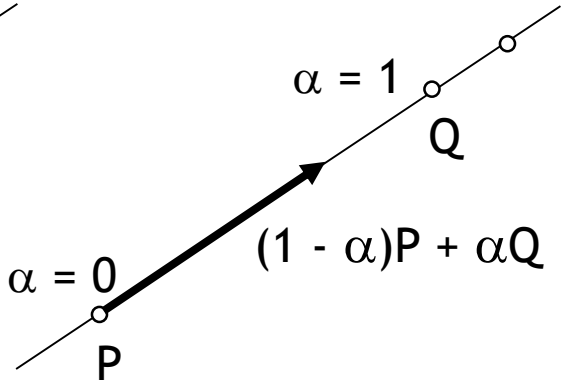


Affine Combination

- Given a sequence of points $P_1, P_2, \ldots P_n$, an affine combination is any sum of the form
  $\alpha_1 P_1 + \alpha_2 P_2 + \ldots + \alpha_n P_n$
- Can show by induction that this sum makes sense if and only if
  $\alpha_1 + \alpha_2 + \ldots \alpha_n = 1$

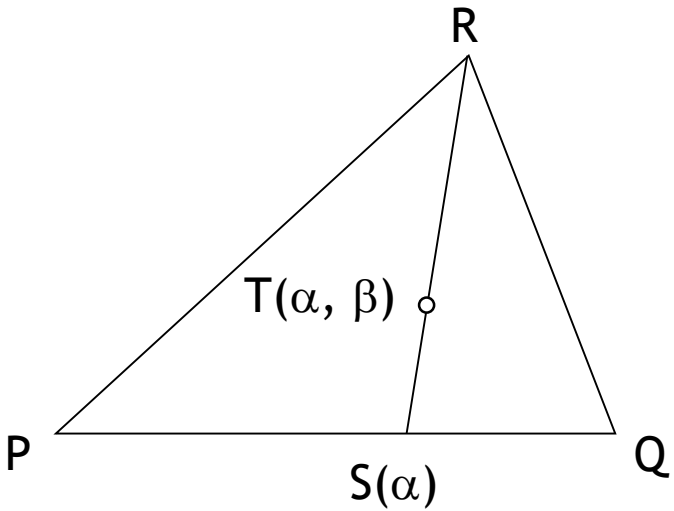






Convex Combination

- Is an affine combination, where in addition we have $\alpha_i >= 0$, i = 1, 2, … , n.



Geometry for Graphics

Euclidean Geometry

- In affine geometry there are no angles or distances.
- Euclidean geometry is an extension of affine geometry which includes one additional operation, called the dot product.

$$u \bullet v = \sum_{i=1}^{n} u_i v_i$$

$$u \bullet v = \|u\|^2$$

- Distance between two points

$$\text{distance}(\mathrm{P}, \mathrm{Q}) = \|d\| = \|q - p\|$$

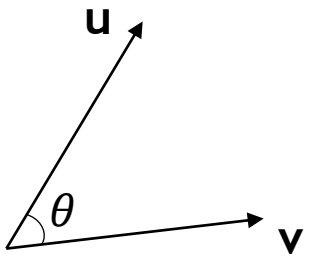
Geometry for Graphics

Euclidean Geometry

Angle between two vectors

$$u \bullet v = \|u\|\|v\|cos\theta$$

$$\theta = \text{acos}(\frac{u \bullet v}{\|u\|\|v\|})$$



$\theta = \text{acos}(u \bullet v)$, where u, v are unit vectors

Geometry for Graphics

- In Unity
  - Vector3 u = v + w; // vector addition
  - Vector3 u = v – w; // vector subtraction
  - If (u == v || u != w) // vector comparison
  - u = v * 2.0f; // scalar-vector multiplication
  - v = w / 2.0f; // scalar-vector division
  - float x = v.magnitude; // length of v
  - Vector3 u = v. normalize; // unit vector of v

Geometry for Graphics

- In Unity
  - float a = Vector3.Angle(u, v); // angle between u, v
  - float b = Vector3.Dot (u, v); // dot product between u, v
  - Vector3 u1 = Vector3.Project(u, v); // orthogonal projection of u onto v
  - Vector3 u2 = Vector3.ProjectOnPlane(u, v); // orthogonal complement
  - float b = Vector3.Distance(p, q); // distance between p and q
  - Vector3 midpoint = Vector3.Lerp(p, q, 0.5 f); // linear interpolation convex combination

Reference

- A Guide to Unity's Coordinate System (With Practical Examples)
  - https://www.techarthub.com/a-guide-to-unitys-coordinate-system-with-practical-examples/
- World Coordinate Systems in 3ds Max, Unity and Unreal Engine
  - http://www.aclockworkberry.com/world-coordinate-systems-in-3ds-max-unity-and-unreal-engine/
- Unity Manual Rendering Pipeline
  - https://docs.unity3d.com/Manual/render-pipelines.html
- Rendering Pipeline in Unity
  - https://medium.com/shader-coding-in-unity-from-a-to-z/rendering-pipe-line-f0471aa0904b