

Unity3D Lecture Note (Part3)

Advanced Topics



- Editor Properties
 - 3D Terrain Editor
 - Particle System
 - Trail Renderer
 - NaviMesh System
 - Networking
 - ML Agent
 - Plugin for Android, IOS
-
- Probuilder package
 - ScriptableObject
 - Joint / Fixed Joint Component
 - Shader
 - Light Probe *



Properties for Inspector

<https://mentum.tistory.com/223>



#region ... #endregion

```
#region Private Mebers  
private Animator _animator;  
.....  
#endregion
```

```
#region Public Members  
public float Speed = 5.0f;  
...  
#endregionm
```



[SerializeField] [System.Serializable]

- private 변수를 인스펙터에 노출시킴
- 단순변수에는 [SerializeField]를, 클래스나 구조체 변수에는 [System.Serializable] 을

[System.NonSerialized]

- public 변수를 인스펙터에 노출시킴

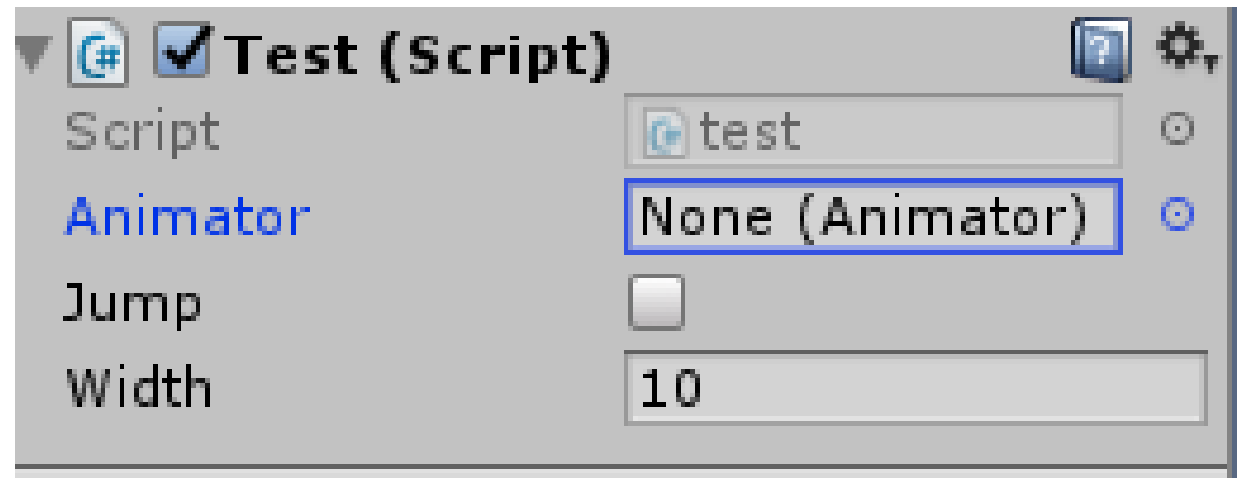
참고: Serialization 이란, C#등에서 제공하는 Serialization API를 이용하여 스트림형태의 데이터변환을 하여 파일로 저장하는 것



```
#region region 1
//[System.Serializable]
[SerializeField]
private Animator _animator;
#endregion
```

```
#region region 2
public bool jump = false;
private float weight = 0.5f;
```

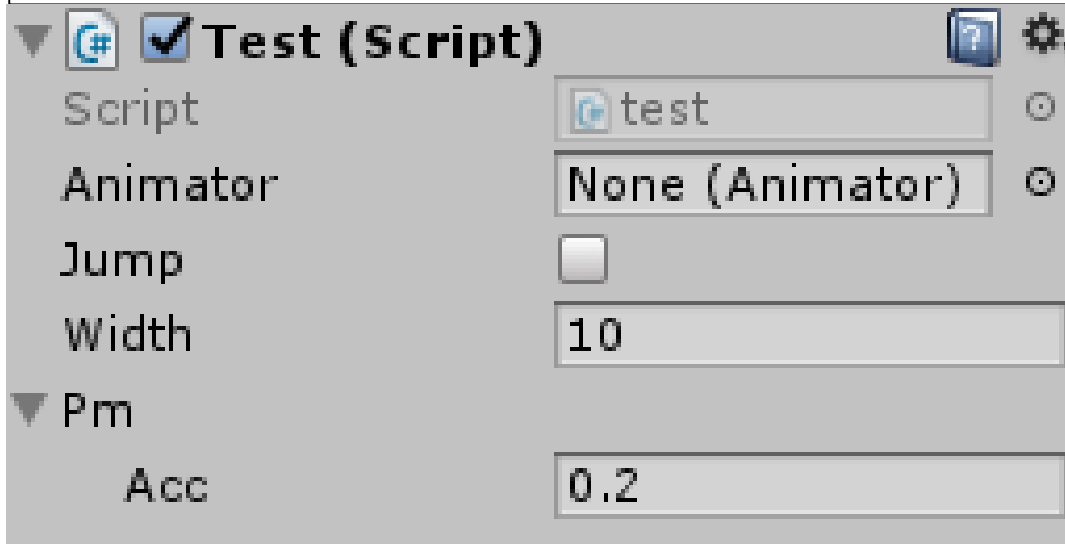
```
[System.NonSerialized]
public int height = 10;
public int width = 10;
#endregion
```



```

using fungame;
...
public class Test : MonoBehaviour
{
...
    public PhysicsManager pm;
...
}

```



```

namespace fungame
{
    [System.Serializable]
    public class PhysicsManager
    {
        private float velocity
= 0.1f;
        public float acc = 0.2f;

        ...
    }
}

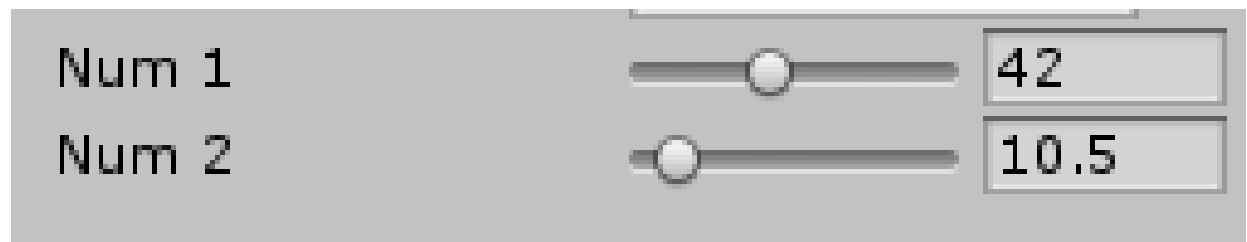
```



```
[HideInInspector]  
public string str2;
```

public 이라도 인스펙터에 노출시키지 않음

```
[Range(0, 100)]  
public int num1 = 0;  
  
[Range(0, 100f)]  
public float num2 = 0;
```



- 인스펙터에서 입력할 때 변수의 수치를 제한해서 입력할 수 있다.
- 실제 값을 제한시키는 효과는 없어서 넘는 수치를 대입하면 그대로 들어간다.




```
[Header( "테 스트 " )]  
public int num1 = 0;
```

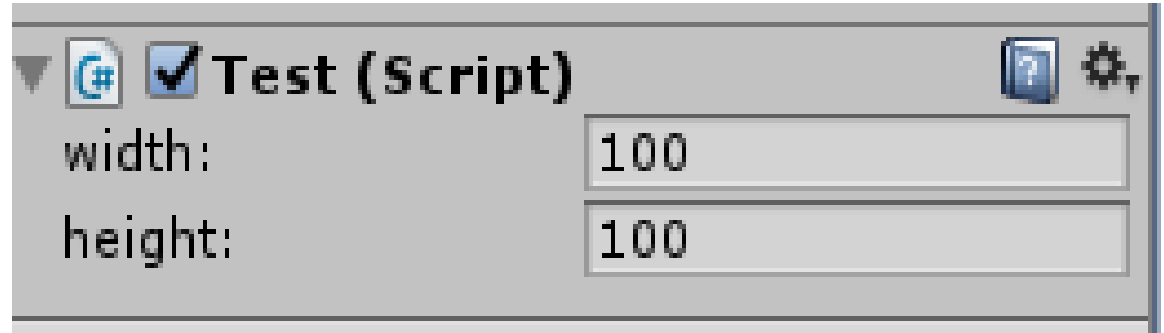
테스트
Num 1

- 변수 앞에 라벨을 표기한다.
- 보통 구분이 잘 안되서 _ 나 * 같은 특수문자를 넣어서 표시.

Component별 전용 Inspector 만들기




- [[CustomEditor](#)(typeof(Component이름))]
- [SerializedObject](#) 클래스,
SerializedProperty 클래스

```
using UnityEngine;
using UnityEditor;
using System.Collections;
//testEditor.cs
[CustomEditor(typeof(test))]
public class testEditor : Editor {
    private test _inv;
    void Awake() {
        _inv = (test)target;
    }
    public override void OnInspectorGUI() {
        _inv.width = EditorGUILayout.IntField("width:", 100);
        _inv.height = EditorGUILayout.IntField("height:", 100);
    }
}
```






// C# 예제: [Transform](#) 컴포넌트 전용 인스펙터. TransformInspector.cs 파일
using UnityEngine;
using UnityEditor;

```
[CustomEditor(typeof(Transform))]  
public class TransformInspector : Editor {  
  
    SerializedObject m_Object;  
    SerializedProperty m_Property;  
  
    void OnEnable () {  
        m_Object = new SerializedObject (target);  
        m_Property = m_Object.FindProperty ("m_LocalPosition.x");  
    }  
  
    void OnInspectorGUI () {  
        // 객체로부터 최신 데이터를 가져옵니다.  
        m_Object.Update ();  
  
        // 속성을 위한 Editor UI  
        EditorGUILayout.PropertyField (m_Property);  
  
        // 속성을 적용하고 되돌리기를 다룹니다.  
        m_Object.ApplyModifiedProperties ();  
    }  
}
```

 **Transform**  

Position	X	<input type="text" value="0"/>	Y	<input type="text" value="0"/>	Z	<input type="text" value="0"/>
Rotation	X	<input type="text" value="0"/>	Y	<input type="text" value="0"/>	Z	<input type="text" value="0"/>
Scale	X	<input type="text" value="1"/>	Y	<input type="text" value="1"/>	Z	<input type="text" value="1"/>



 **Transform**  

▶ Local Rotation

Local Position

X	<input type="text" value="0"/>	Y	<input type="text" value="0"/>	Z	<input type="text" value="0"/>
---	--------------------------------	---	--------------------------------	---	--------------------------------

Local Scale

X	<input type="text" value="1"/>	Y	<input type="text" value="1"/>	Z	<input type="text" value="1"/>
---	--------------------------------	---	--------------------------------	---	--------------------------------

[CustomEditor(typeof(Inventory))]

개체 타입(예제에서는 Inventory)을 사용자 지정 에디터 클래스로 수정할 수 있는 것을 정의함.



```

using UnityEngine;
using UnityEngine.UI;
public class Inventory : MonoBehaviour
{
    public Image[] itemImages = new
Image[numItemSlots];
    public Item[] items = new Item[numItemSlots];
    public const int numItemSlots = 4;
    public void AddItem(Item itemToAdd)
    {
        for (int i = 0; i < items.Length; i++)
        {
            if (items[i] == null)
            {
                items[i] = itemToAdd;
                itemImages[i].sprite = itemToAdd.sprite;
                itemImages[i].enabled = true;
                return;
            }
        }
    }
}

```

```

public void RemoveItem (Item
itemToRemove)
{
    for (int i = 0; i < items.Length;
i++)
    {
        if (items[i] == itemToRemove)
        {
            items[i] = null;
            itemImages[i].sprite = null;
            itemImages[i].enabled =
false;
            return;
        }
    }
}

```

```

using UnityEngine;
using UnityEditor;
[CustomEditor(typeof(Inventory))]
public class InventoryEditor : Editor
{
    private void OnEnable ()
    {
        itemImagesProperty = serializedObject.FindProperty
(inventoryPropItemImagesName);
        itemsProperty = serializedObject.FindProperty (inventoryPropItemsName);
    }
    public override void OnInspectorGUI ()
    { .... void ItemSlotGUI (...) ... }
    private void ItemSlotGUI (int index)
    { ...}
    • }

```

<https://learn.unity.com/tutorial/adventure-game-phase-2-inventory-system#5c7f8528edbc2a002053b392>



[MenuItem("Edit/Delete All PlayerPrefs")]

Editor 메뉴 밑에 Edit/Delete All PlayerPrefs 메뉴 아이টে
를 추가하기.

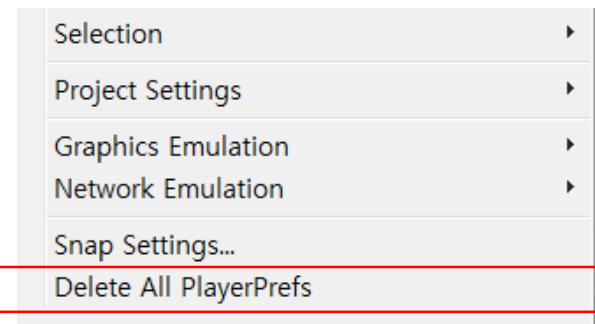
```
//DeletePrefs.cs
using UnityEditor;
using UnityEngine;

public class DeletePrefs : EditorWindow {

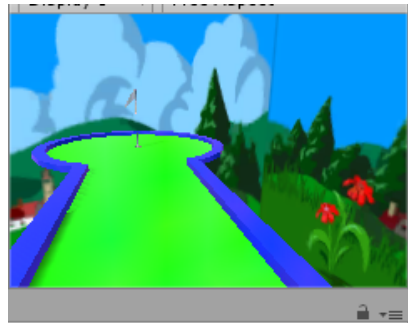
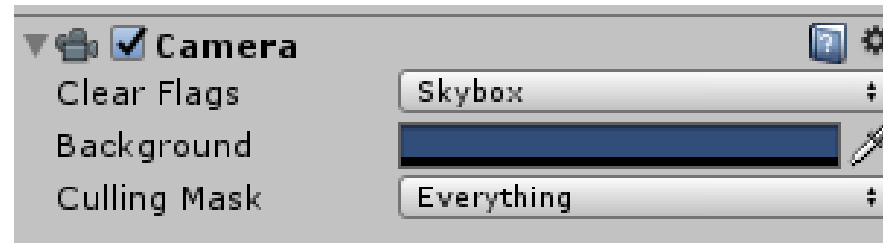
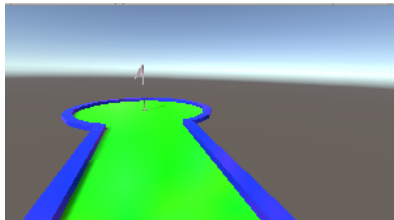
    [MenuItem("Edit/Delete All PlayerPrefs")]

    public static void DeletePlayerPrefs(){
        PlayerPrefs.DeleteAll();
        Debug.Log("delete prefs");
    }
}
```

- (1) Create a new folder in your Assets folder called Editor.
- (2) Create a new C# Script and name it DeletePrefs.cs within the Editor

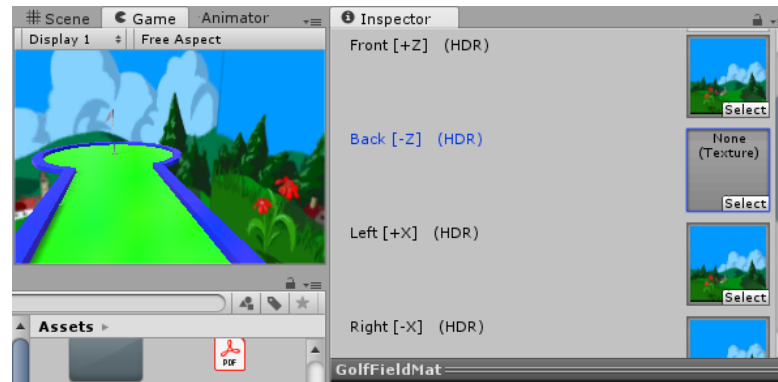
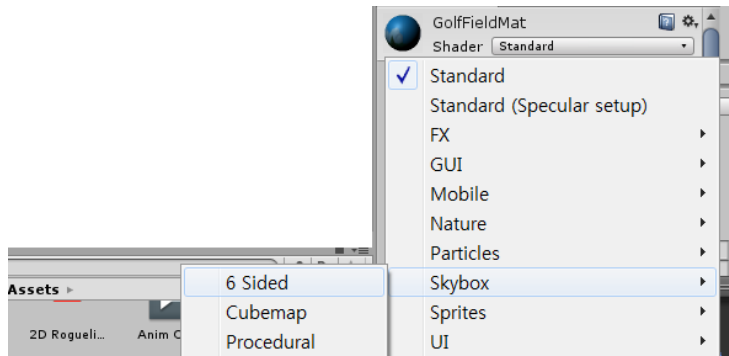


Skybox 생성/설정

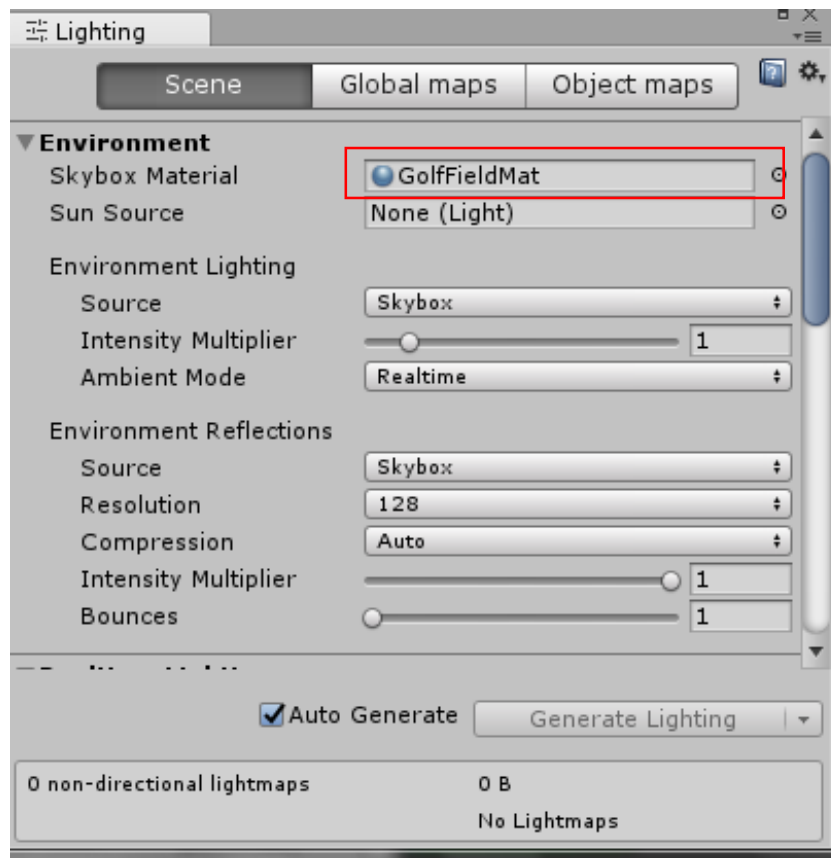


Step 1) Skybox Material 생성

- <https://docs.unity3d.com/560/Documentation/Manual/HOWTO-UseSkybox.html>



Step 2) Window->Lighting->Setting 에서 "Scene" tab 에서 스카이박스 설정하기



Particle System

Trail Renderer

File Edit Assets GameObject Component Window Help



Pivot Local



Collab



Account

Layers

Layout

Hierarchy

Create All

scene_main*

- Main Camera
- WallTop
- WallBottom
- WallRight
- WallLeft
- DottedLine
- RacketLeft
- RacketRight
- Ball
 - trail_effect
 - Sphere
 - Cube

Project Console

Create

Favorites

- All Materials
- All Models
- All Prefabs
- All Modified
- All Conflicts

Assets

Trail

Assets Trail



AfterExplos...



Jellyfish



Mat



New Prefab



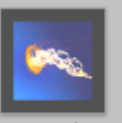
Physic2D...



Physic_Mat



Steam_A



TestTrailM...



trail_effect

Steam A.mat

Rebuilding GUID cache: Deleting metadata D:/Program Files/Unity/Editor/Data/UnityExtensions/Unity/GUISystem/Editor/UnityEditor.UI.dll because the asset doesn't exist anymore.



Inspector

Steam_A

Shader FX/Flare

Particle Texture

Tiling

X 1

Y 1

Offset

X 0

Y 0



Select

Render Queue

From Shader 3000

Double Sided Global Illumination

Steam_A



AssetBundle None

None

오전 10:56
2020-04-12

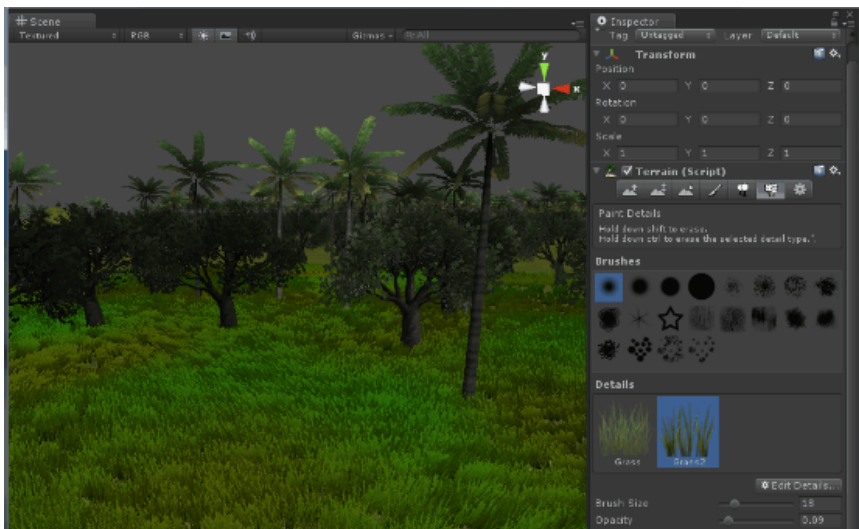
3D Terrain Editor

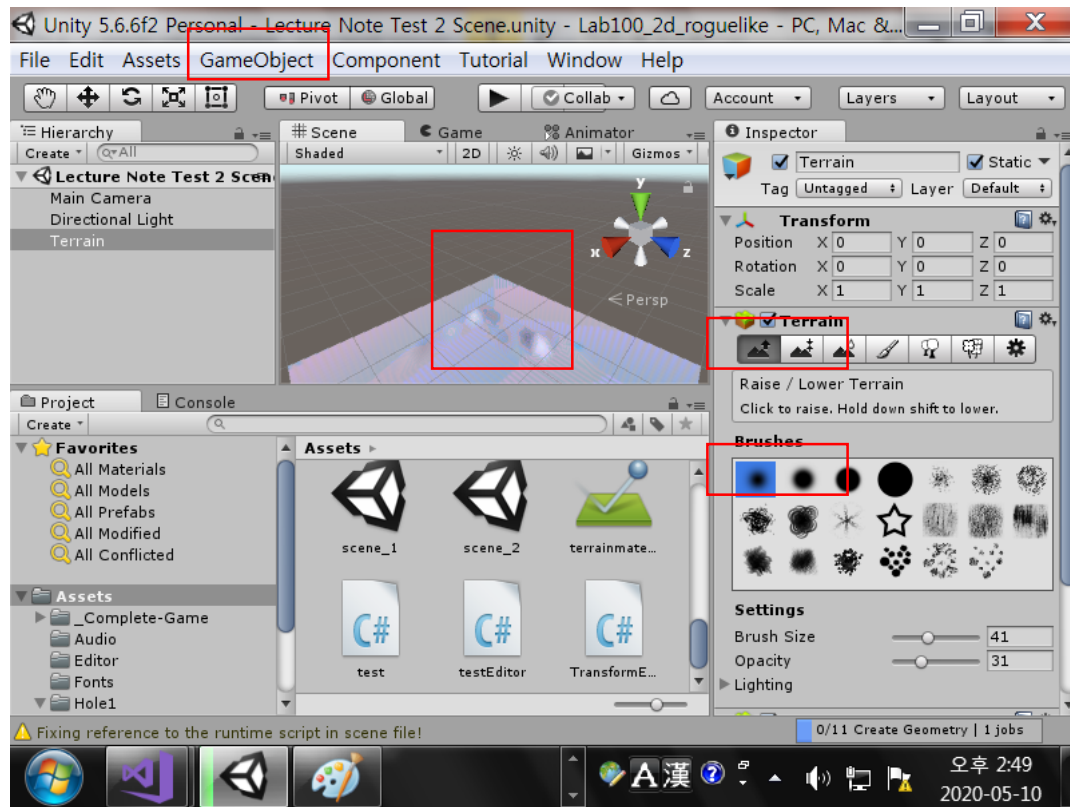
Unity 3D Terrain Editor

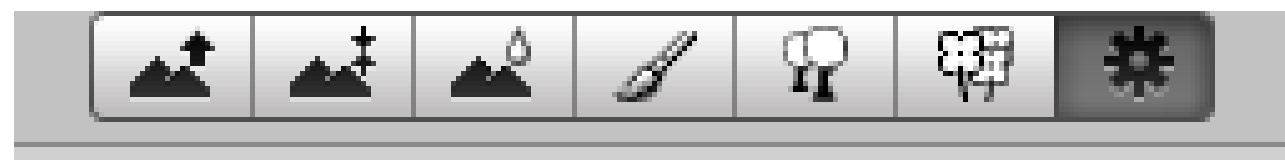
- Create terrain by selecting brush type, brush size and opacity and then sculpting topology
- Set maximum height and smooth corners
- Textures loaded to paint texture onto terrain
- First texture acts as background to subsequent
- Paint on trees and other smaller items e.g grass.

[terrains-with-unity/](#)

<https://gamedevelopment.tutsplus.com/tutorials/unity-terrain-engine-tools--cms-28623>



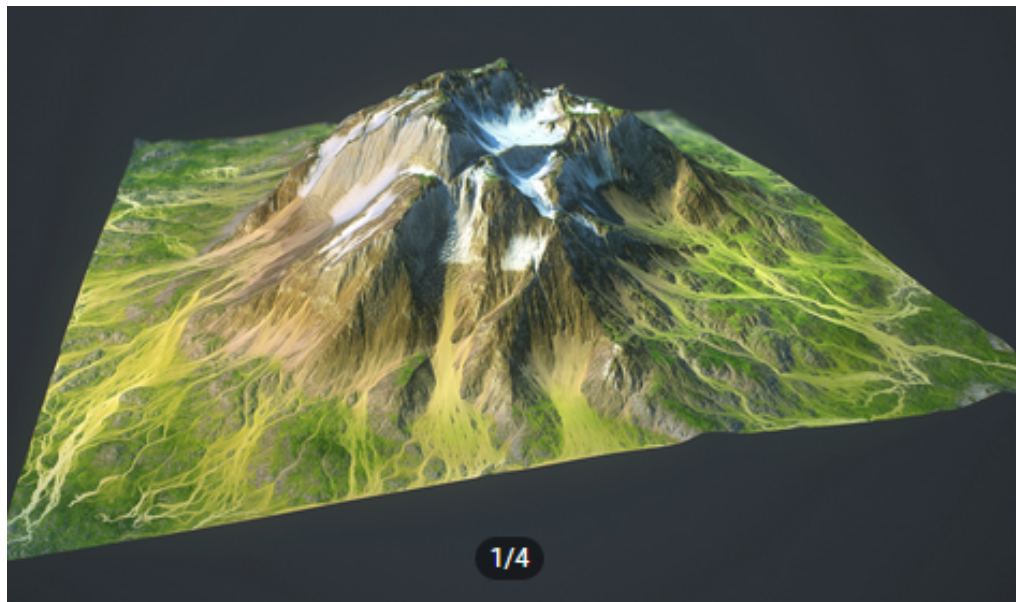




Please note that modifying the resolution will clear the heightmap, detail map or splatmap.

Terrain Width	1000
Terrain Height	600
Terrain Length	1000
Heightmap Resolution	513

Free Terrain (mountain) in unity asset store



MetaStudio

★★★★★ 5 | 4 Reviews

FREE

Download

Needs Unity upgrade to version 2018.4.5



Add to List

Share

Screen Coordinate to World Coordinate

예제) 화면에서 마우스 클릭하면, 그 곳으로 paddle 이 위치하도록 함

;

ScreenToWorldPoint(sx, sy, sz)

Transforms position from screen space into world space.

sx, sy : screen coordinates

Screen space is defined in pixels. The bottom-left of the screen is (0,0).

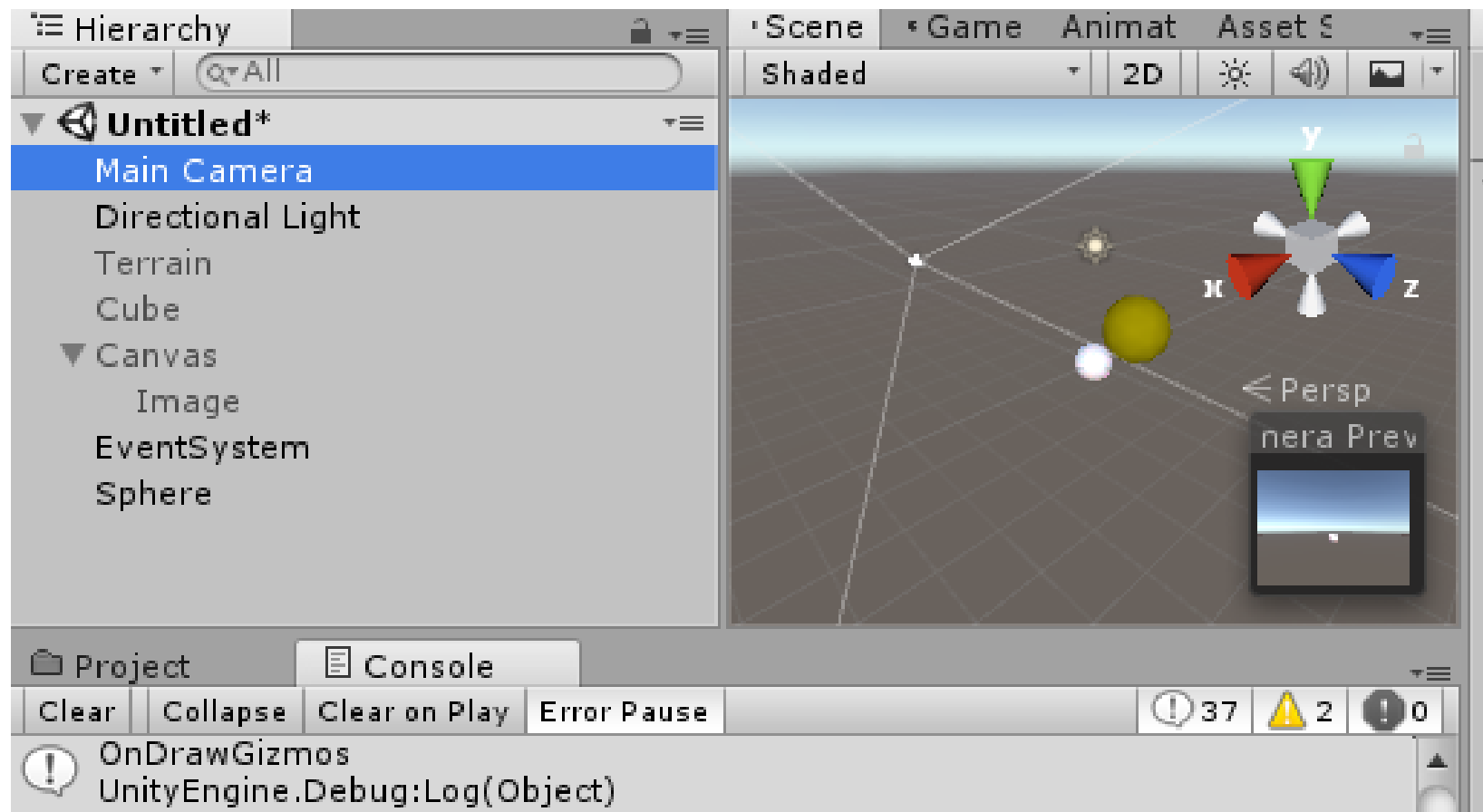
Sz:

The z position is in world units from the camera

```
// Draw a yellow sphere in the scene view at the position  
// on the near plane of the selected camera that is  
// 100 & 100 pixels from lower-left.
```

```
using UnityEngine;  
using System.Collections;
```

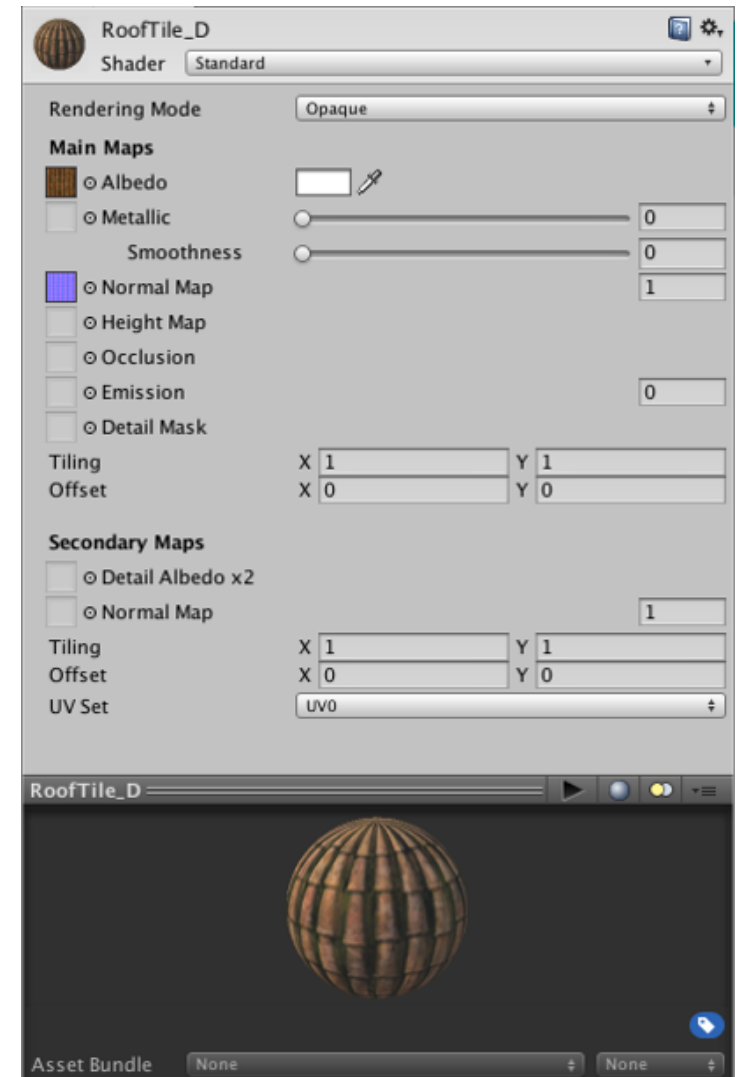
```
public class ExampleClass : MonoBehaviour {  
    ...  
    public Camera camera; //main camera  
    ...  
    void OnDrawGizmos () {  
        Vector3 p = camera.ScreenToWorldPoint(new Vector3(100, 100,  
            5)); //camera.nearClipPlane));  
        Gizmos.color = Color.yellow;  
        Gizmos.DrawSphere(p, 0.5F);  
    }  
}
```



Shader

Material and Shader

- The properties that a Material's inspector displays are determined by the **Shader** that the Material uses. A shader is a specialised kind of graphical program that determines how **texture and lighting information are combined to generate the pixels** of the rendered object onscreen.



- Unity is equipped with a powerful shading and material language called **ShaderLab**. (+ Cg(from Ndivia) + HLSL(from Microsoft)
- [Tutorial](https://docs.unity3d.com/kr/530/Manual/ShaderTut1.html) : 셰이더: ShaderLab과 고정함수 셰이더
<https://docs.unity3d.com/kr/530/Manual/ShaderTut1.html>

Simple Example Shader

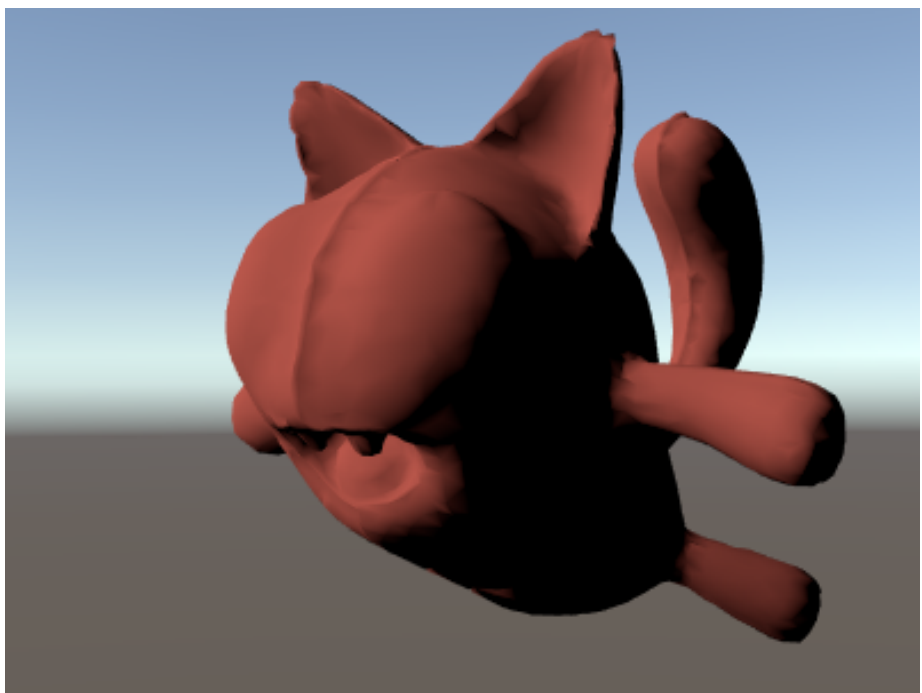
- Unity has a way of writing *very simple* shaders in so-called “fixed-function” notation. We’ll start with this for simplicity. Internally the fixed function shaders are converted to regular [vertex and fragment programs](#) at shader import time.

- 시작하기

쉐이더를 생성하려면 메뉴 바에서 Assets->Create->Shader을 선택하거나 기존의 쉐이더를 복사하고, 거기에서 작업합니다. 새로운 쉐이더는 Project View를 더블 클릭하여 편집할 수 있습니다.

```
Shader "Tutorial/Basic" {  
    Properties {  
        _Color ("Main Color", Color) = (1,0.5,0.5,1)  
    }  
    SubShader {  
        Pass {  
            Material {  
                Diffuse [_Color]  
            }  
            Lighting On  
        }  
    }  
}
```

- 이 간단한 셰이더는 가장 기본적인 셰이더 중 하나를 보여줍니다. 이것은 "Main Color"라는 색상 프로퍼티를 정의하고, 이에 장미와 같은 색상(빨강=100%, 녹색=50%, 파랑=50%, 알파=100%)의 기본값을 할당합니다. 다음 Pass를 호출해 오브젝트를 렌더링하고 해당 패스에서 디퓨즈 메테리얼 컴포넌트를 프로퍼티 "_Color"로 설정하고 정점 당 라이팅을 켭니다.
- 이 셰이더를 테스트하려면 드롭 다운 메뉴에서 셰이더를 선택하고(Tutorial->Basic), 그 메테리얼을 오브젝트에 할당합니다. 메테리얼 인스펙터에서 색상을 조정하고 변경 사항을 확인합니다. 더 복잡한 것을 해봅시다!



Vertex and Fragment Shaders:

<https://docs.unity3d.com/kr/530/Manual/ShaderTut2.html>

Surface Shader

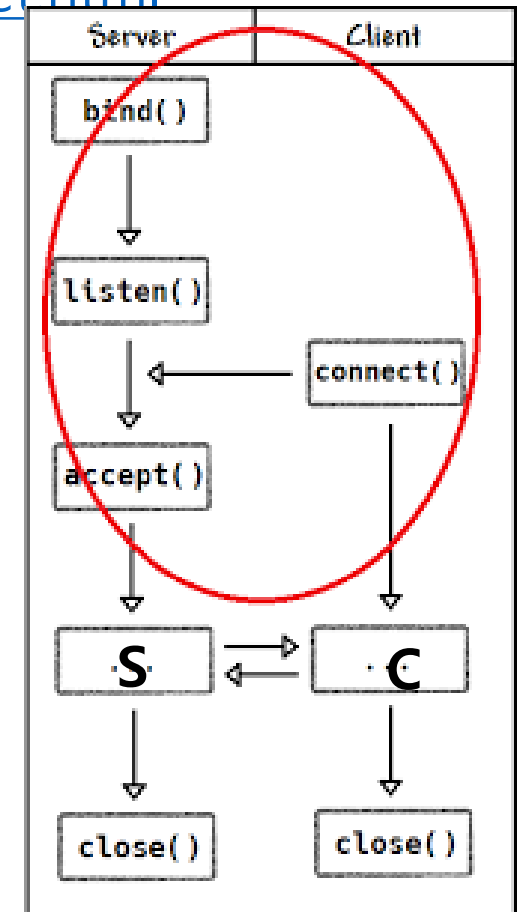
<https://docs.unity3d.com/kr/530/Manual/SL-SurfaceShaders.html>

Networking

Unity Socket

<http://unitynetworkwithcsharp.blogspot.com/2017/11/socket.html>

Lab102_Unity Socket Programming Example.hwp



- UNet

더 이상 사용되지 않는다

Unit is a deprecated solution, and a new Multiplayer and **Networking** Solution (MLAPI) is under development

- MLAPI (Multiplayer Networking API)

https://docs-multiplayer.unity3d.com/?_ga=2.10420145.525614209.1617521643-117529344.1615535912

- MLAPI is an open-source project
- MLAPI 이용 게임 예제 소스 코드

<https://github.com/Unity-Technologies/com.unity.multiplayer.mlapi>

- **com.unity.transport package** to add multiplayer and network features to your project.
(Unity version 2020.1.2 and later)

Advanced Topics

(Unity 5.6 – 2017 – 에 도입된 기능을 중심으로)

GPU instancing

- Use GPU Instancing to draw (or render) multiple copies of the same Mesh at once, using a small number of draw calls. (To draw a GameObject on the screen, the engine has to issue a draw call to the graphics API (such as OpenGL or Direct3D))
- GPU Instancing only renders identical Meshes with each draw call, but each instance can have different parameters (for example, color or scale) to add variation and reduce the appearance of repetition.
- GPU Instancing is useful for drawing objects such as buildings, trees and grass, or other things that appear repeatedly in a Scene.

Unity Compute Shader

- GPGPU (General-Purpose computing on Graphics Processing Units)
 - 일반적으로 컴퓨터 그래픽스를 위한 계산만 맡았던 그래픽 처리 장치(GPU)를, 전통적으로 중앙 처리 장치(CPU)가 맡았던 응용 프로그램들의 계산에 사용하는 기술
 - GPGPU 구현 API & 프레임워크: OpenGL Compute Shader, DirectX DirectCompute, OpenCL, CUDA ("Compute Unified Device Architecture", 쿠다, NVIDIA GPU가 필수)
- Unity Compute Shader
 - DirectX 11 스타일의 HLSL로 구현된 연산용 셰이더

불칸(Vulkan)

불칸(Vulkan)

- 게임이나 상호작용성 미디어와 같은 고성능 실시간 3D 그래픽스 애플리케이션을 **모든 플랫폼에서 고성능으로 CPU를 적게 사용하도록 개발하는** 것을 목표로 만들어진 [API](#)이며 [마이크로소프트](#)의 [Direct3D 12](#) 등과 같은 성격의 [API](#).
- Vulkan은 [멀티 코어](#) CPU의 여러 코어 사이에 로드를 더 잘 분배할 수 있음.
- Unity내 “Player Settings” 에서 Vulkan API 선택가능

텍스트메시 프로(TextMesh Pro) Game Object

- 텍스트메시 프로(TextMesh Pro)
 - 텍스트메시 프로는 텍스트 포맷과 레이아웃을 크게 개선한 컨트롤 외에
- - 다이내믹 비주얼 텍스트 스타일을 갖춘 고급 텍스트 렌더링이 특징

Video Player component

- 새로운 동영상 플레이어가 탑재:
- 4k 영상의 재생 및 360도 영상의 VR 체험이 가능

페이스북 게임룸과 구글 데이드림 등을 지원에 Unity 엔진 사용

- Facebook 게임룸
 - unity-based system for game launching and playing, like, STEAM
 - 페이스북과 유니티 테크놀로지(Unity Technologies) 간 협력 관계로 구축. 모든 캐주얼 게임에 유니티(Unity) 엔진을 사용.
- Google 데이드림 (Daydream)
 - VR platform: primarily for use with a headset into which a smartphone (android OS) is inserted.