UNIVERSITÀ DEGLI STUDI
DI MILANO

Unity 101 - Advanced

A More Sophisticated Look

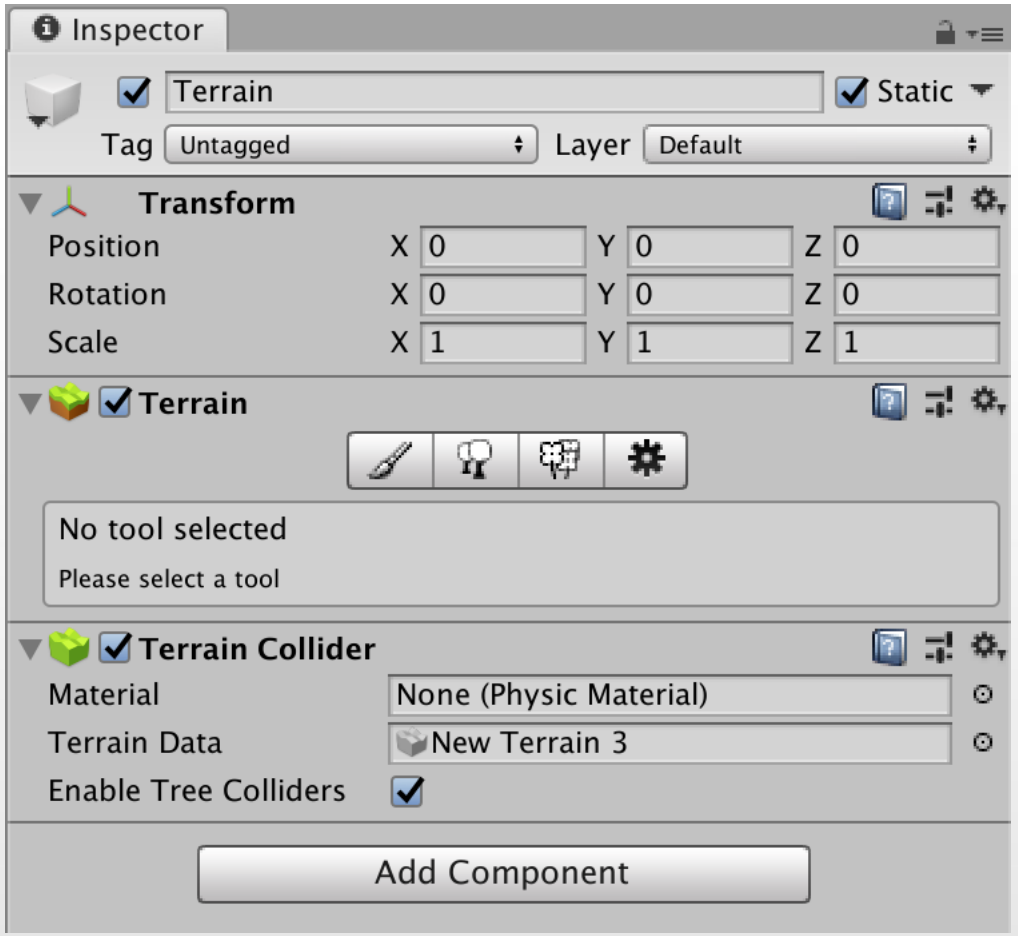
Dario Maggiorini (dario@di.unimi.it)

Playlab fOr inNovation in Games

Terrain

- Unity provides terrain using a large gameobject to be used like a platform for your game

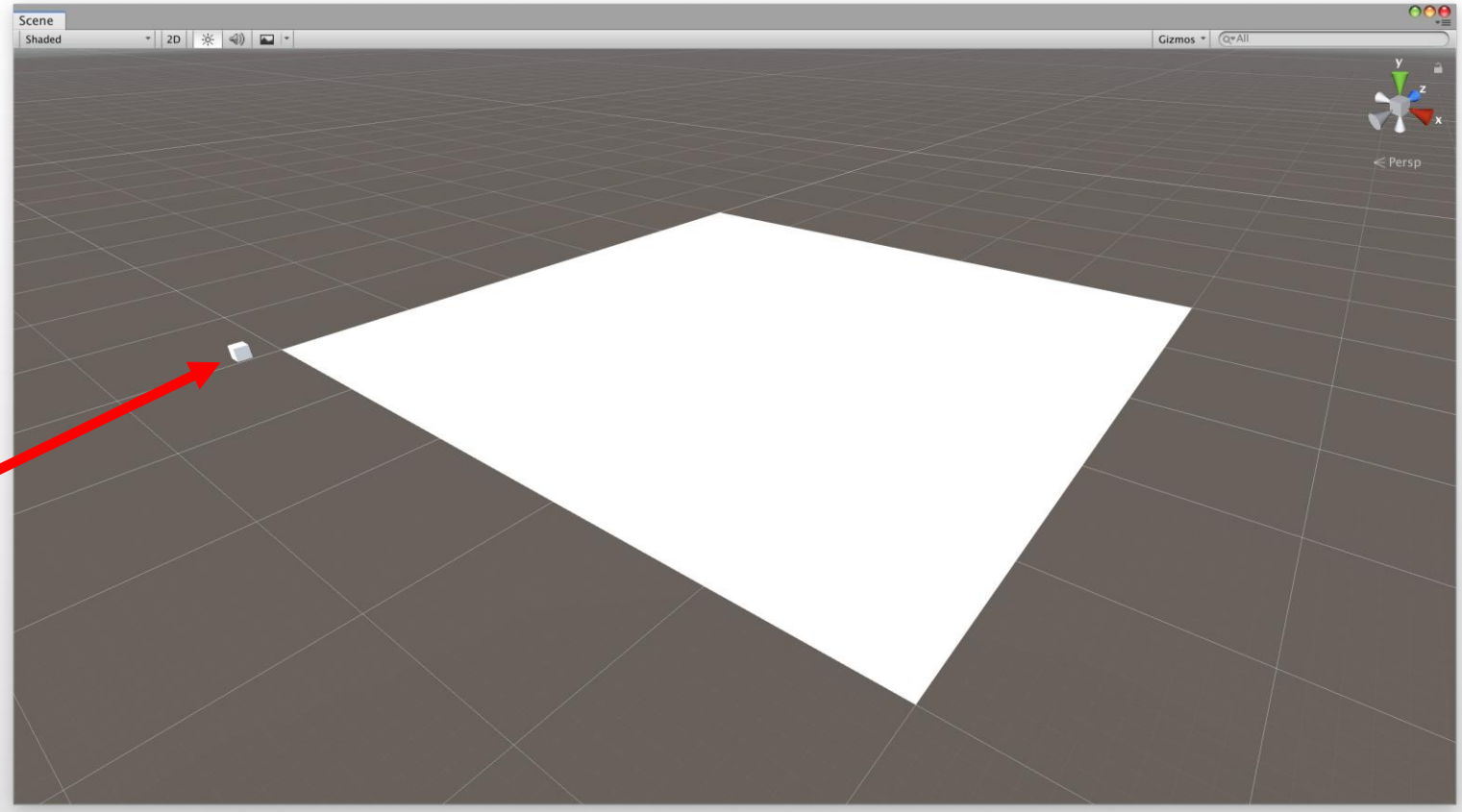
- Once added to the scene, a terrain has two peculiar components:
  - An editing tool
  - A terrain-specific collider



Beware of a Terrain

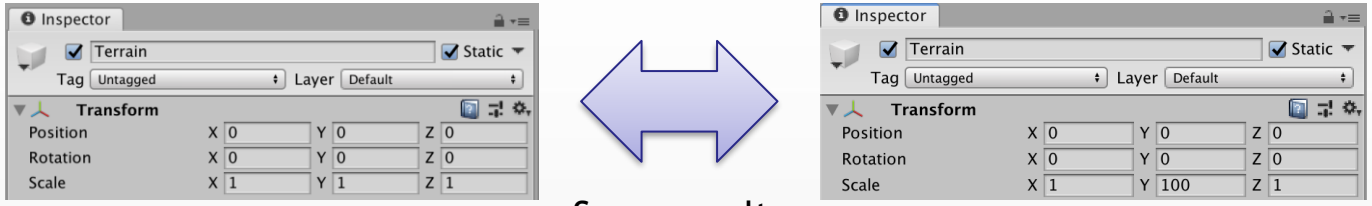
- A terrain is some kind of anomalous gameobject

1. It is extremely huge
   - By default 500x500 meters



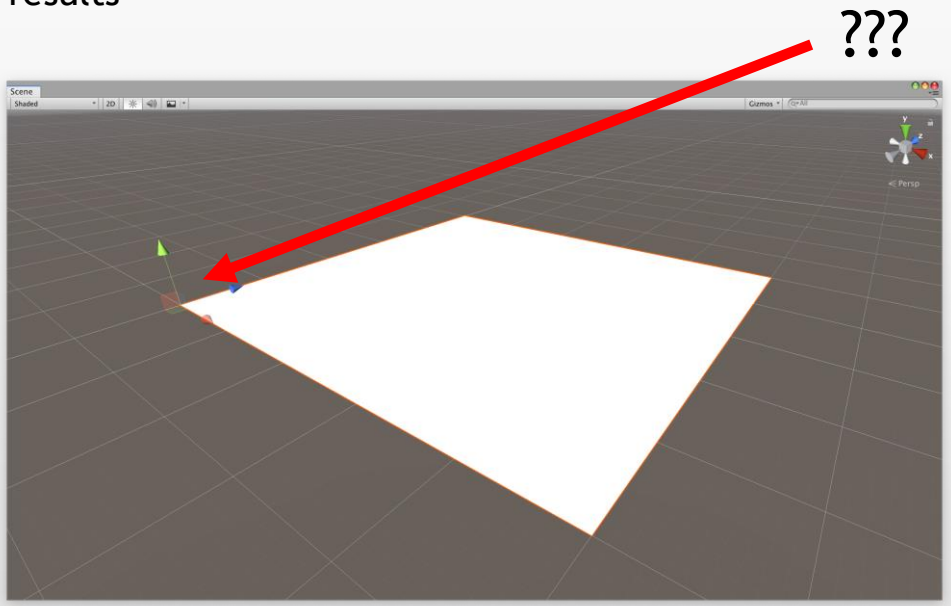
Beware of a Terrain

2. It does not scale like other objects
   - Changing the scale value in the transform component is not doing anything

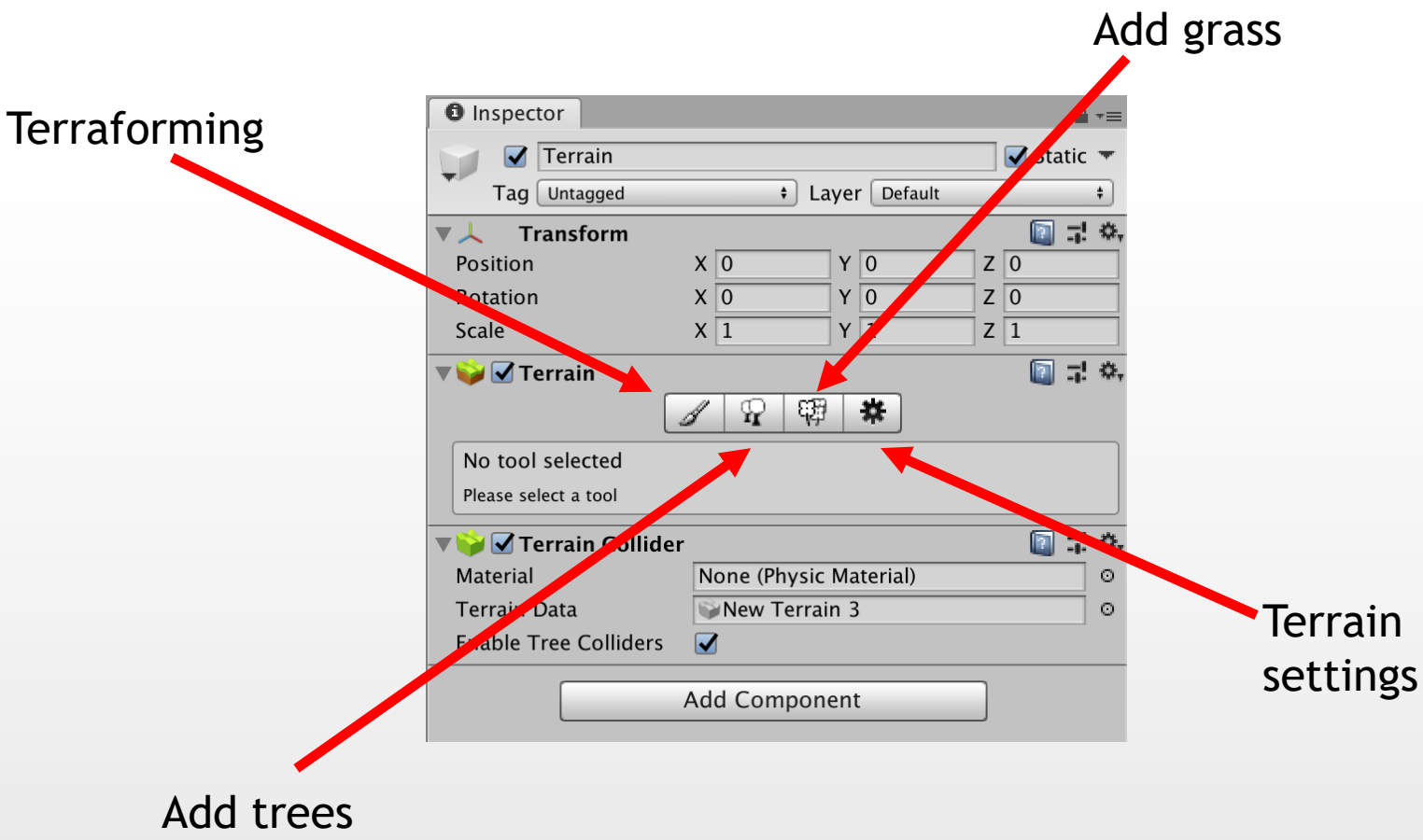


3. The geometric reference is in a corner

4. A terrain is an asset
   - Every change made to its structure will be recorder in the asset database ... and will persist between runs!

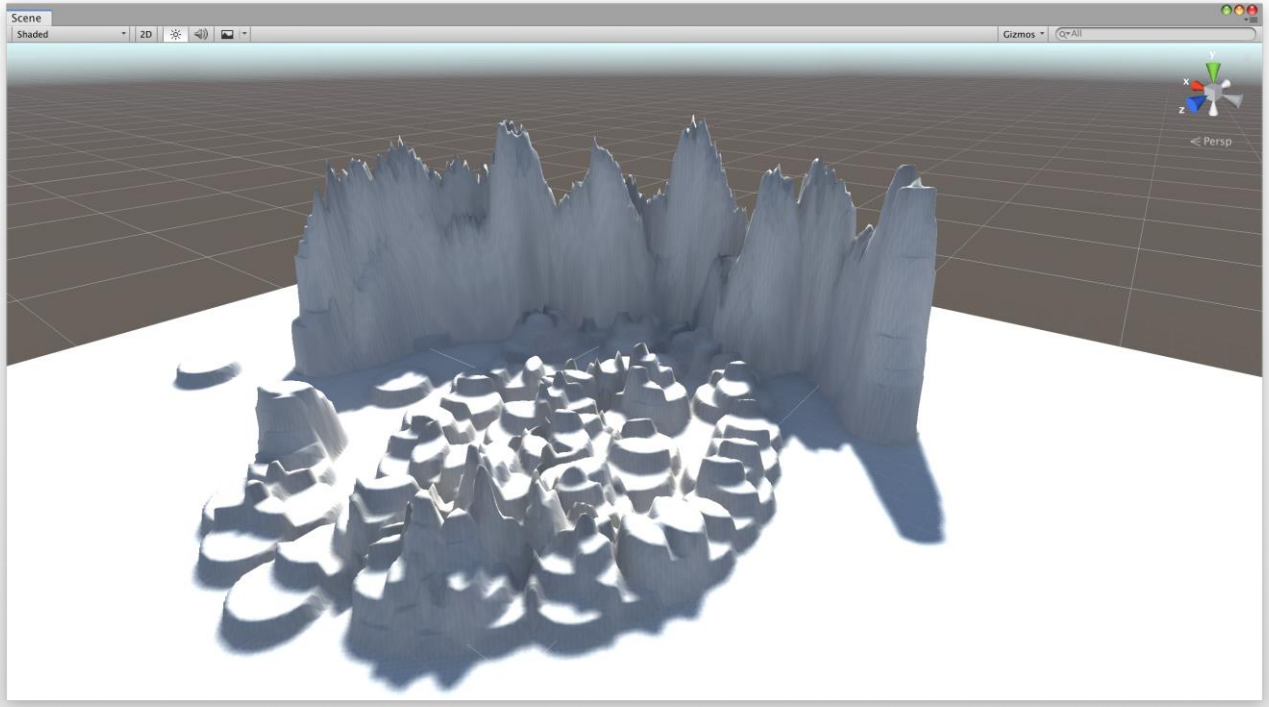


Terrain Tools



Terraforming

- You can use a brush-like tool to raise and lower the terrain
  - You can define you own brushes
  - Nevertheless, this is going to be **a painful experience** unless you are looking for something very basic
    - *"Why do you set all your games on the moon?"*



Importing Maps

- You can import maps by way of heightmaps
  - Heightmaps are grayscale images where black means height=0 and white means height=maximum

- IMPORTANT
  - Images MUST be square
    - Better if the size is a power of two
  - Format MUST be raw
    - If you don not have photoshop available use Krita (free software) and save as “R8 Heightmap”, then rename the file to .raw

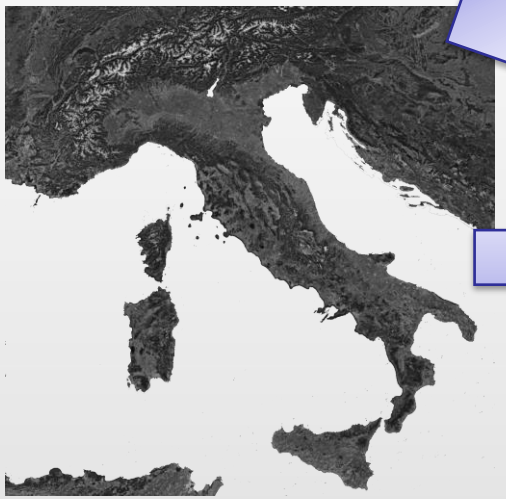
- You will be able to terraform manually after importing in order to clean artifacts

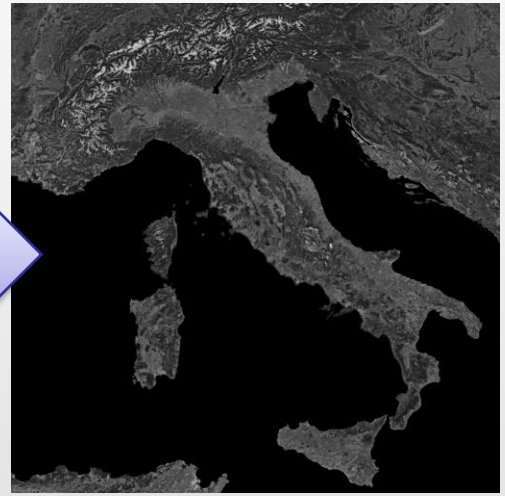
Creating an Heightmap

Step 1: get the image you need

Step 2: rip off everything not needed (it will be level zero)

Step 3: Make it grayscale



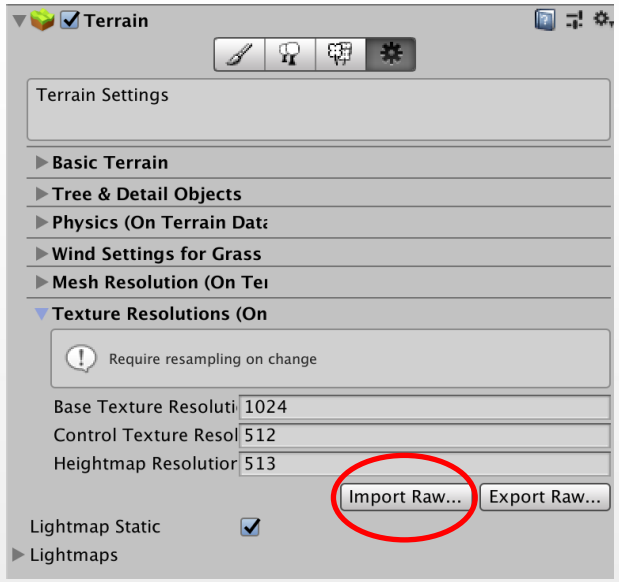


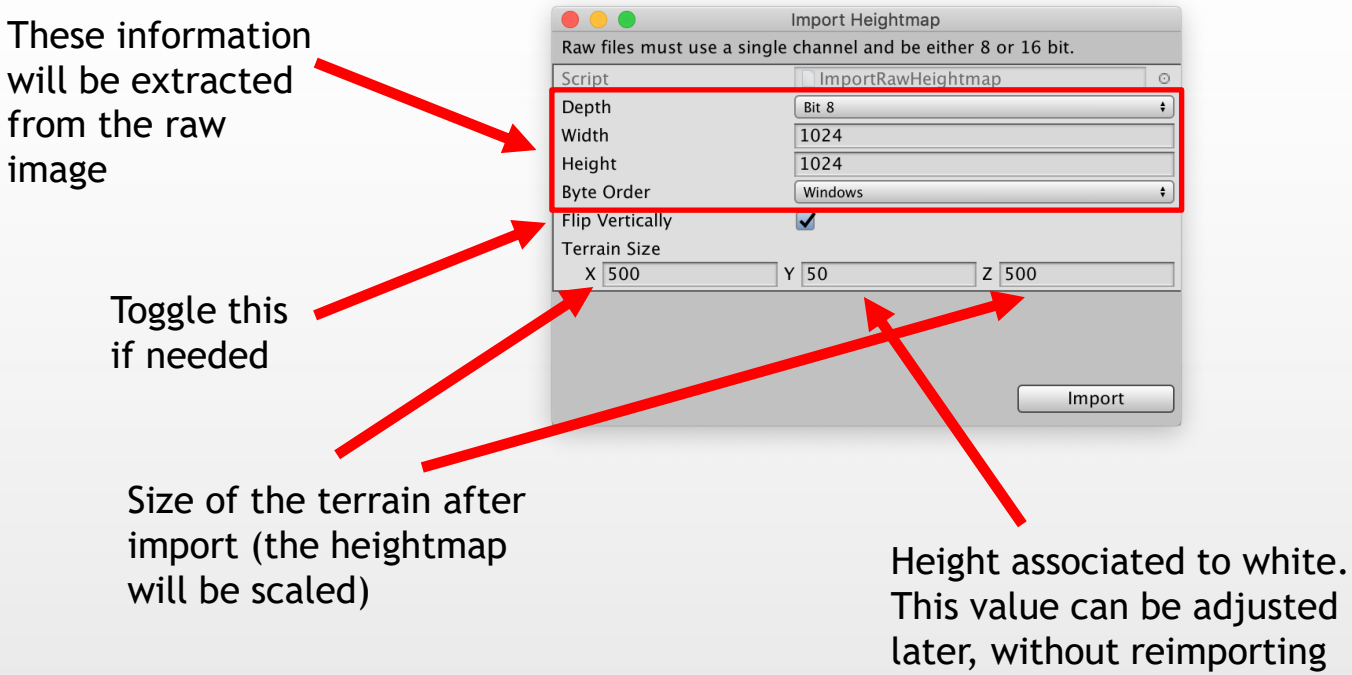
Step 4: Clean up and fill every hole with black.

Cleaning is important because even a small white dot will create visible artifacts (spikes) in your scene

Importing an Heightmap

- Open the terrain component and select import raw
  - A file selection dialog will appear

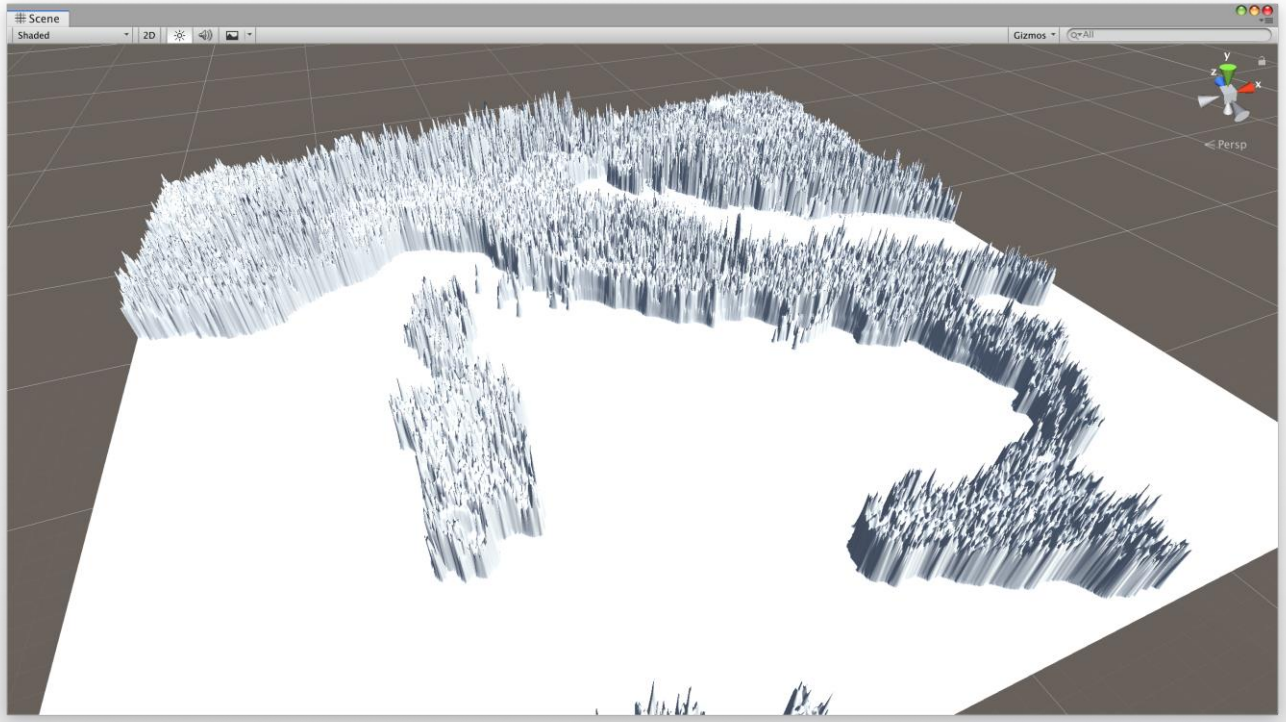




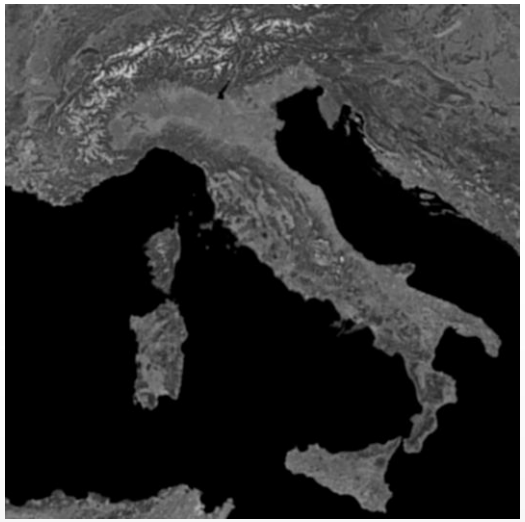
- After selecting the file, an import dialog will pop up

Importing Italy

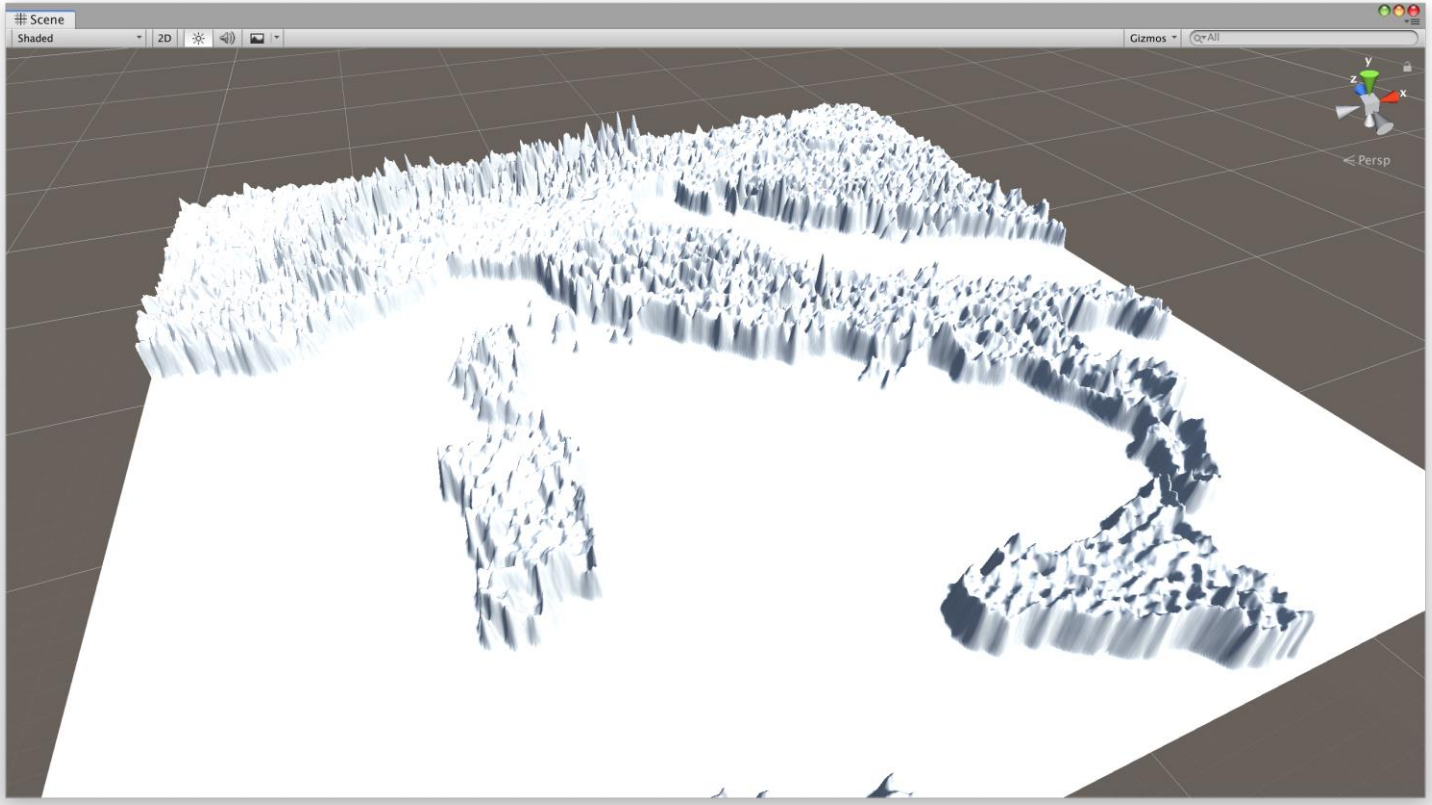
- Not bad, for a first try, but the map is too spiky
- Each pixel is associated to a map sample, and we have too much high frequency components in the image
- This can be easily fixed blurring the image and re-importing



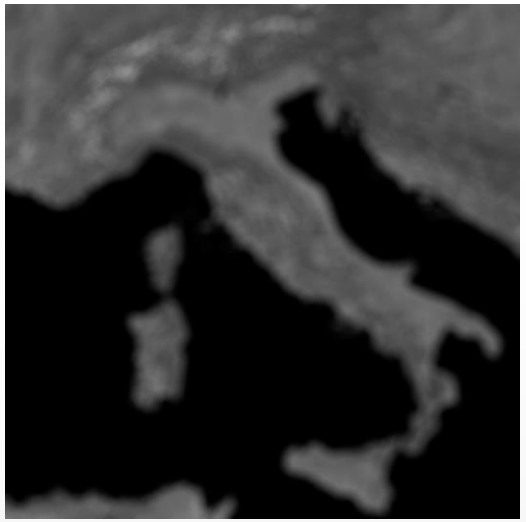
Blurring With Radius 3



Better, but still a bit spiky

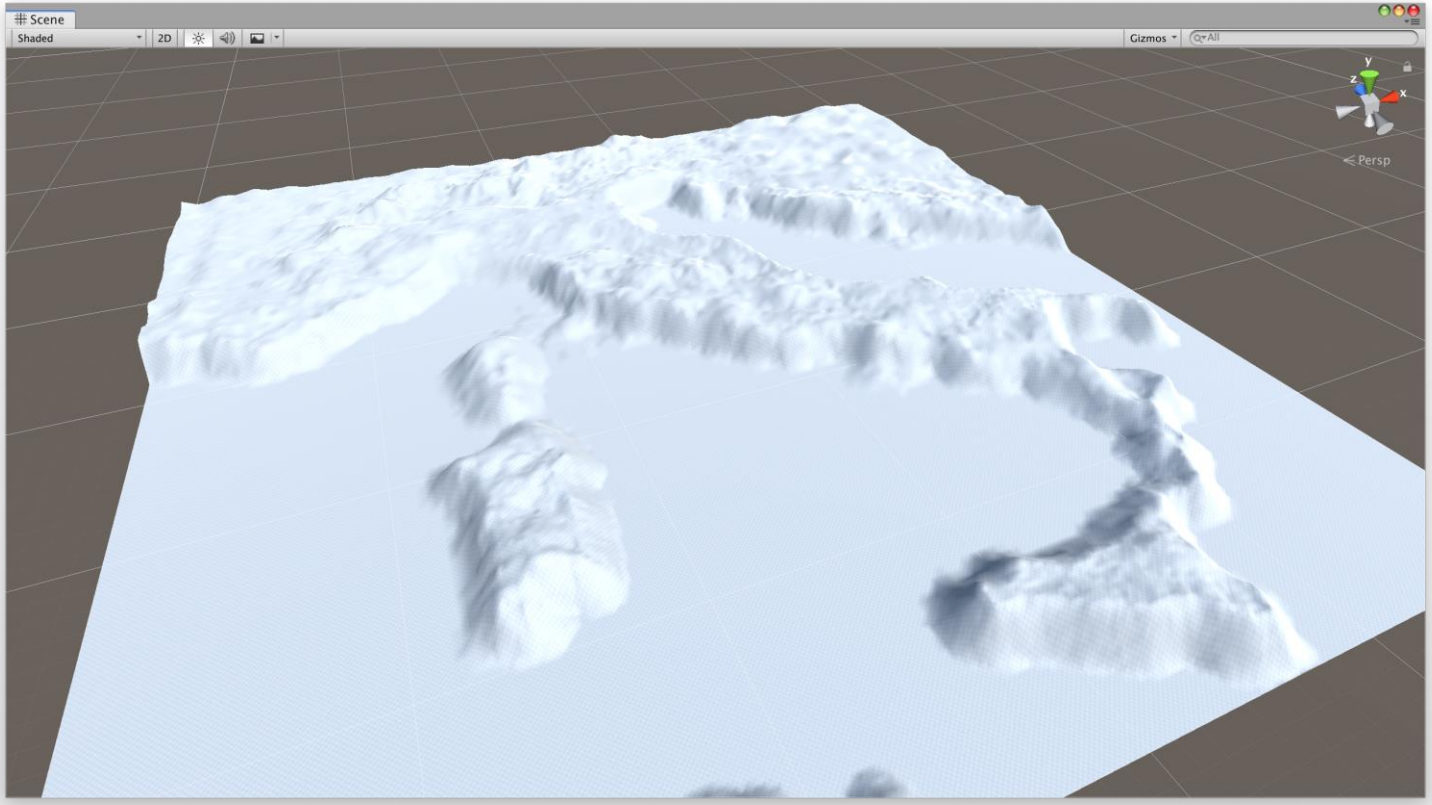


Blurring With Radius 16

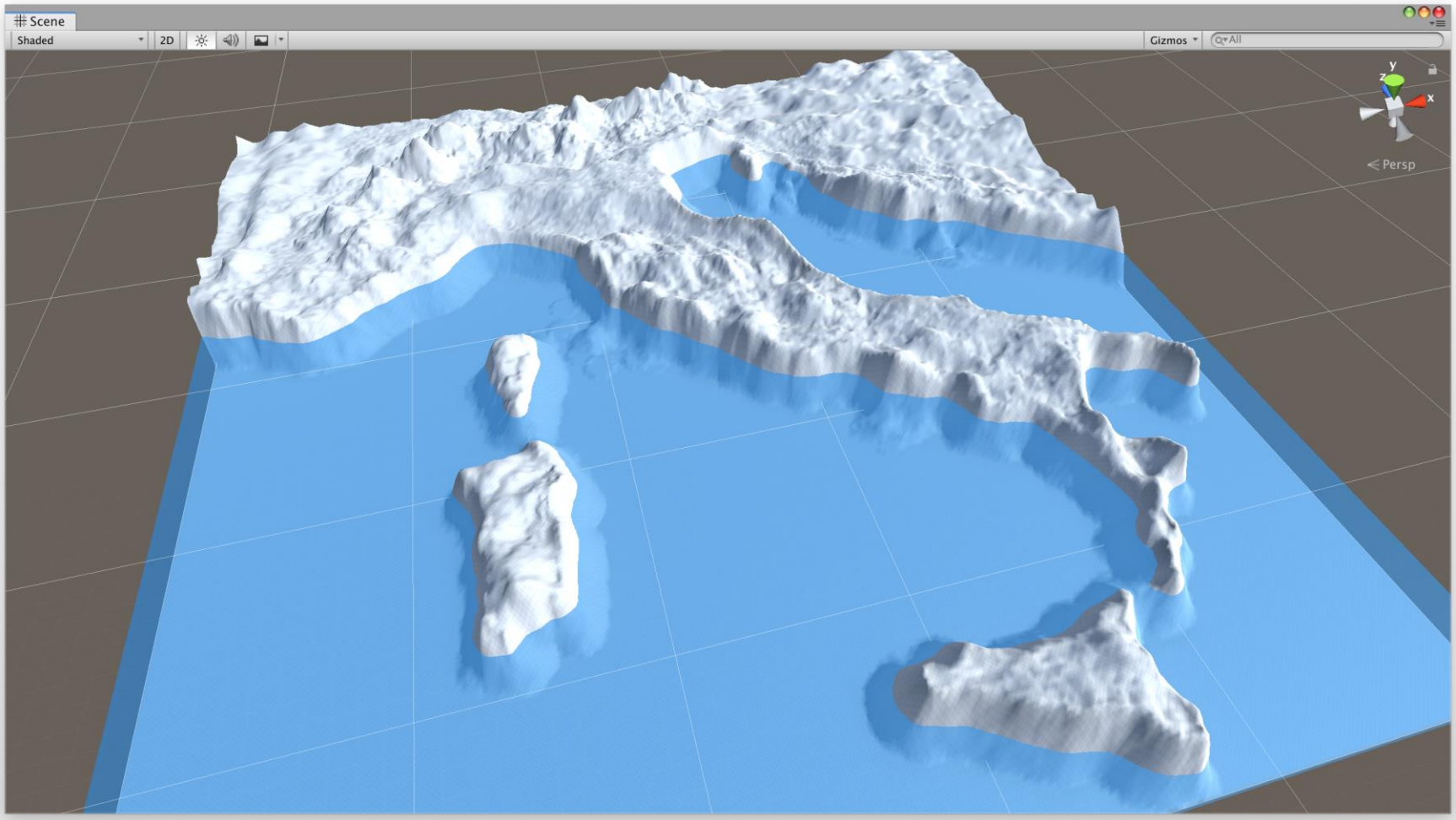


The heightmap image looks awful, but the result is great.

TIP: Lower intensity of the directional light to reduce the glare



Final Result

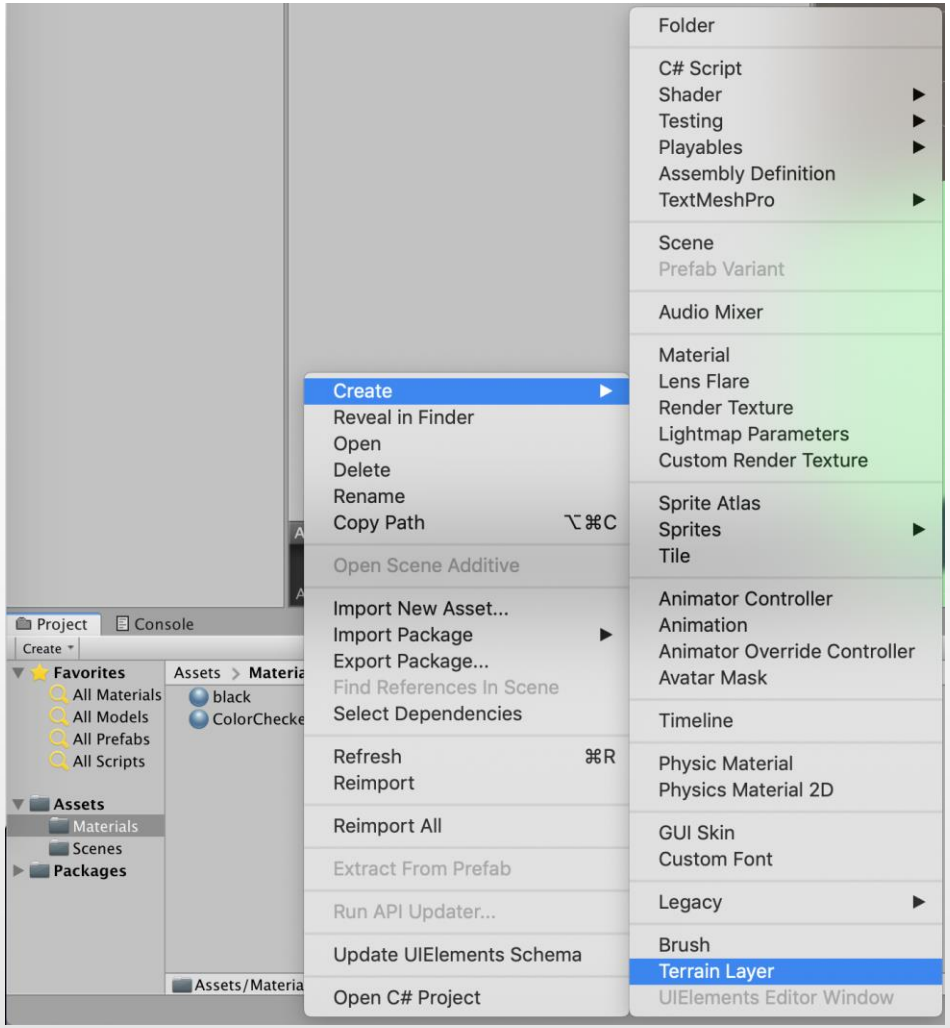


Adding some water and adjusting the light can do wonders to your map

Painting Terrain

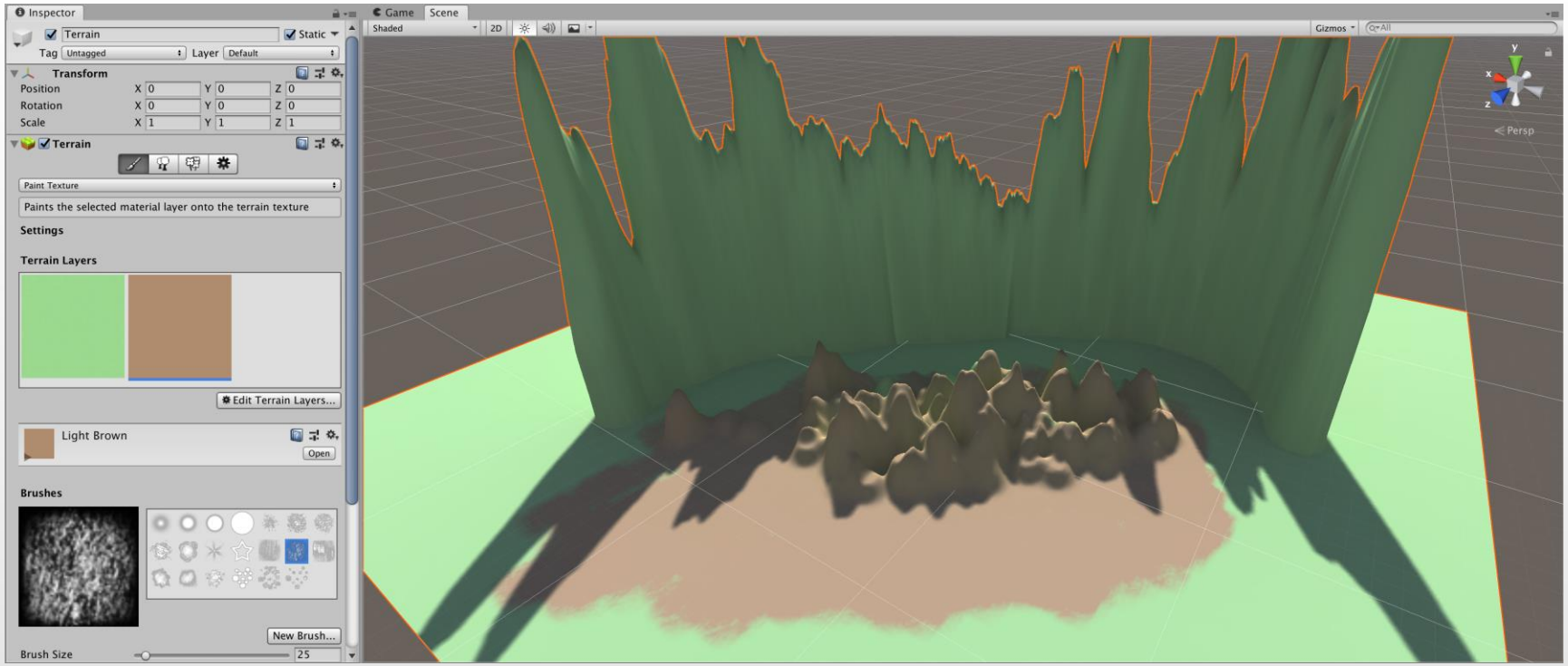
- This operation is part of the terraforming process
  - Select “paint texture” in the dropdown

- You must define layers for the terrain
  - Each layer is characterized by a diffuse texture and (optionally) a normal map and a mask map
  - Create a layer by right-clicking in the asset database and selecting create and then terrain layer



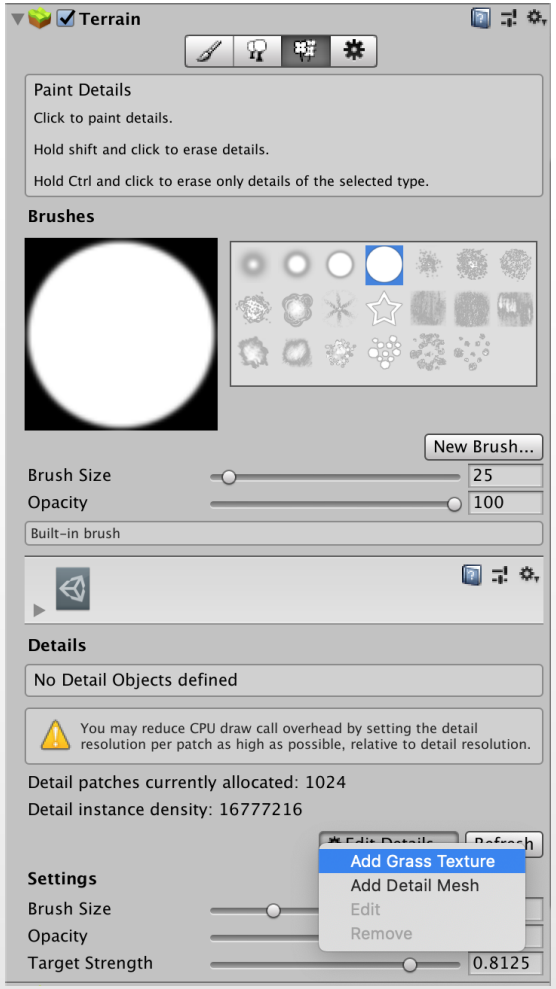
Painting Terrain

- The first layer you add will flood the terrain
- From the second layer on, it will be selectable to add color patches on the terrain following the shape of your brush



Trees and Grass

- To add trees, you need at least a tree prefab
  - There are none by default. Create your own, or hunt for them in the asset store
- To add grass, you paint it on the terrain
  - You must add a grass texture to the terrain
- IMPORTANT!
  - Trees and grass are rendered based on distance. If you do not see them, move closer to the terrain or raise “detail distance” in the terrain settings



Sky Boxes

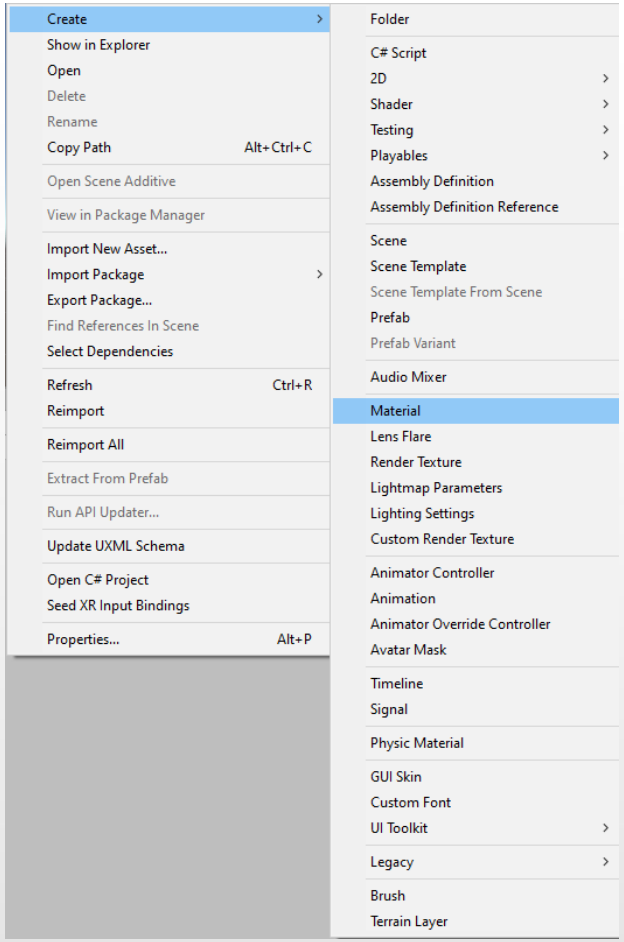
- Skyboxes are basically the opposite on a terrain
- A skybox is a rendering container wrapping the environment and giving the impression of the horizon

- In Unity there are two kinds of skyboxes:
  - Cubemap
  - 6-sided

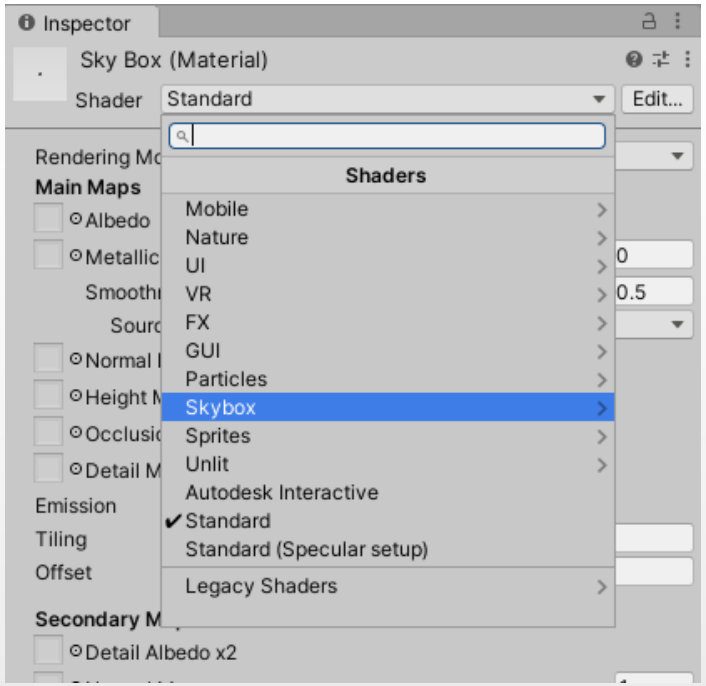
- Both are defining a material to be applied to the scene

Creating a Sky Box

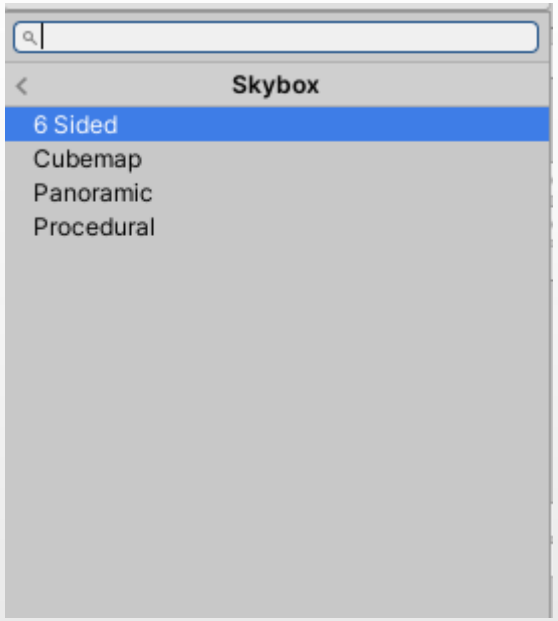
Step 1: create a new material in the asset database



Step 2: in the inspector set the shader type to skybox



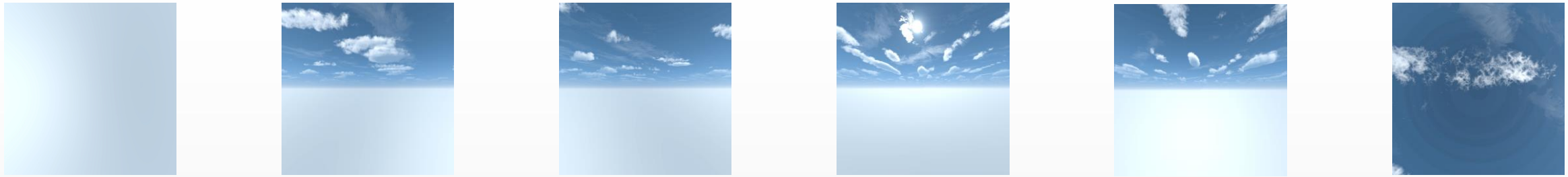
Step 3: select the type to skybox you want



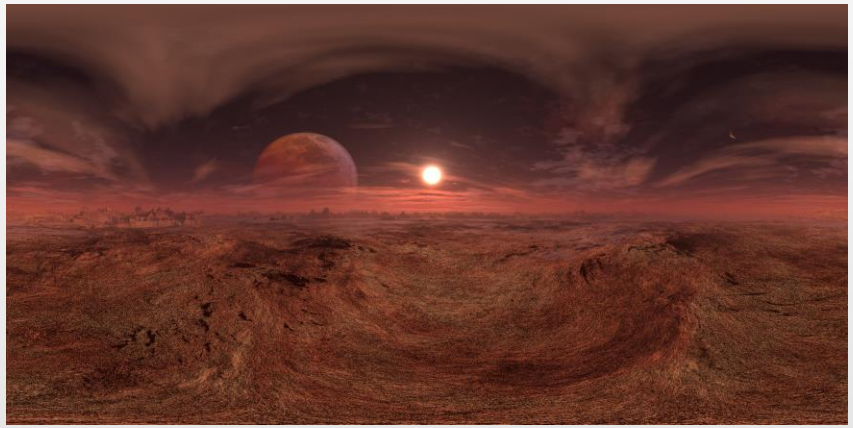
6-sided vs Cubemap

- In a 6-sided sky box you have a texture for each side of the cube

Textures must be square

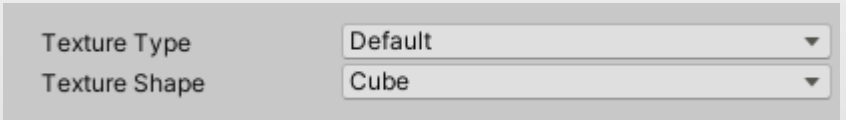


- In a cubemap sky box you have a single texture wrapping all around the box

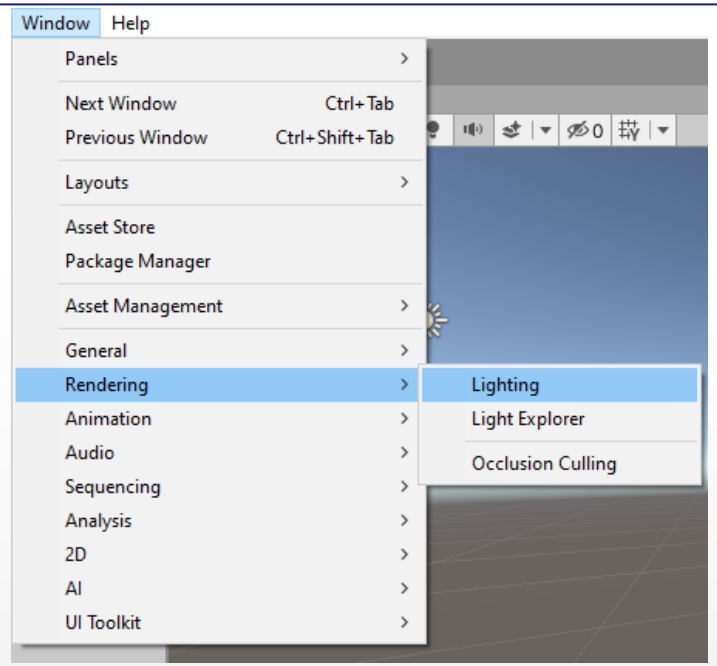


Texture aspect must be 2:1

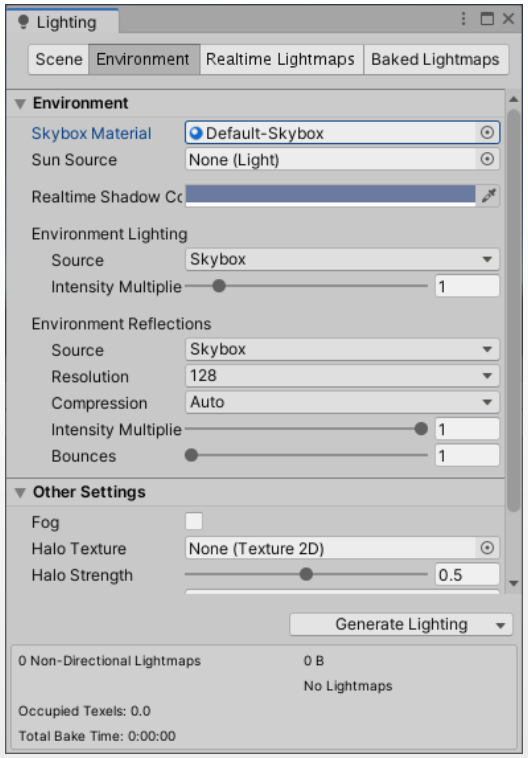
Make sure the texture shape in the inspector is set to “Cube”



Applying a Sky Box



Step 1: Open the lighting window



Step 2: Select " Environment" and change the skybox material

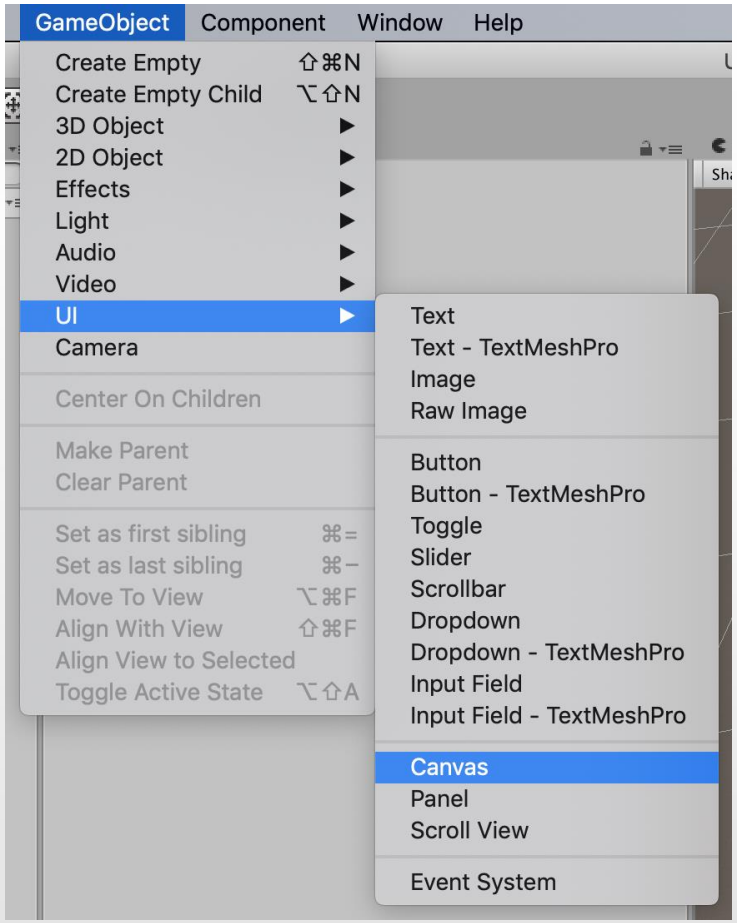
Step 3: Use the editor as usual

User Interface

- The UI is a canvas with 2D child objects showing in front of the camera, no matter its position

- The canvas is part of the scene
  - Can be added from the object menu
  - And that is … kind of weird

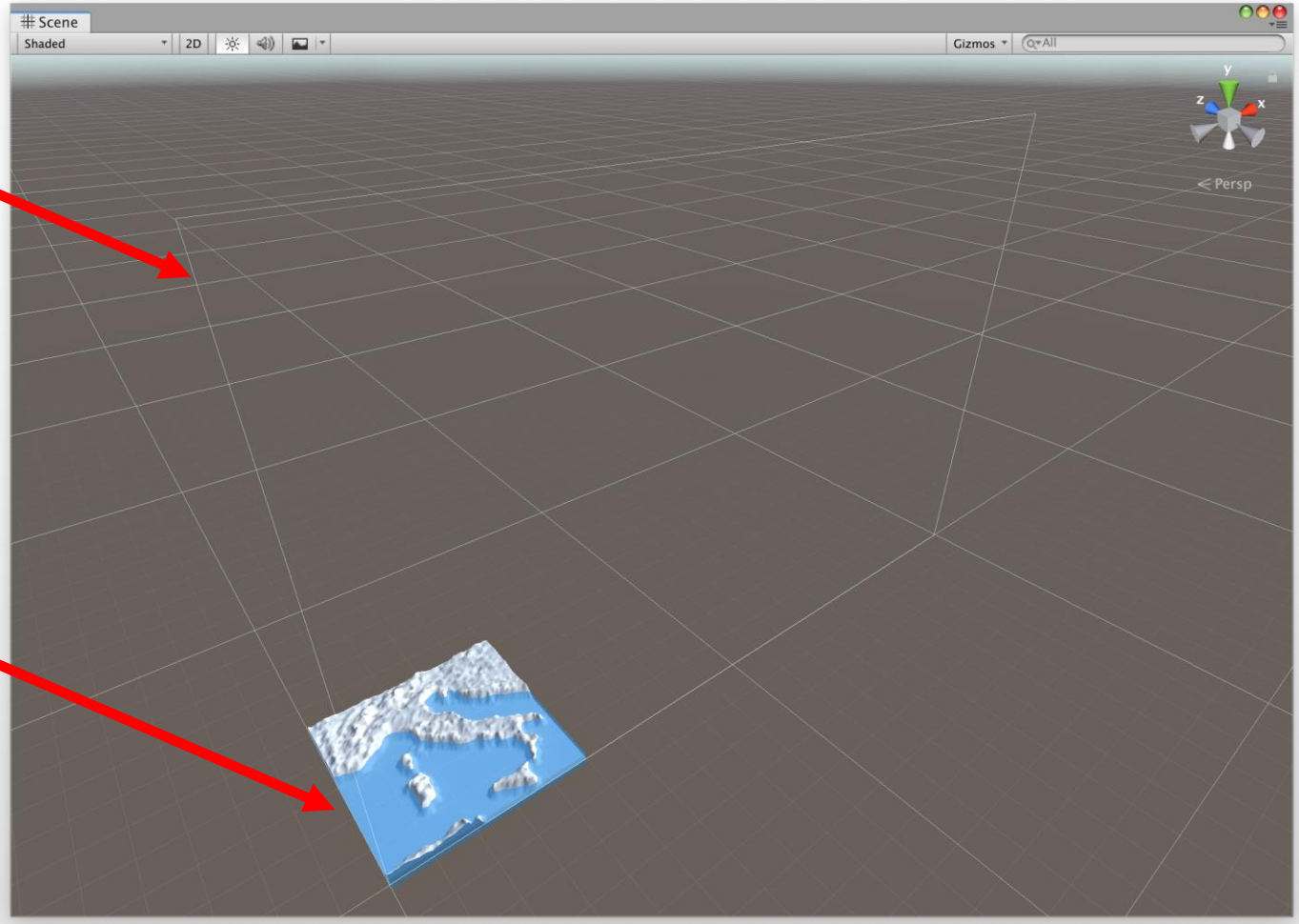
- UI gameobjects management is the same as for in-scene objects
  - But, in this case, the Z position controls the overlap



A Canvas

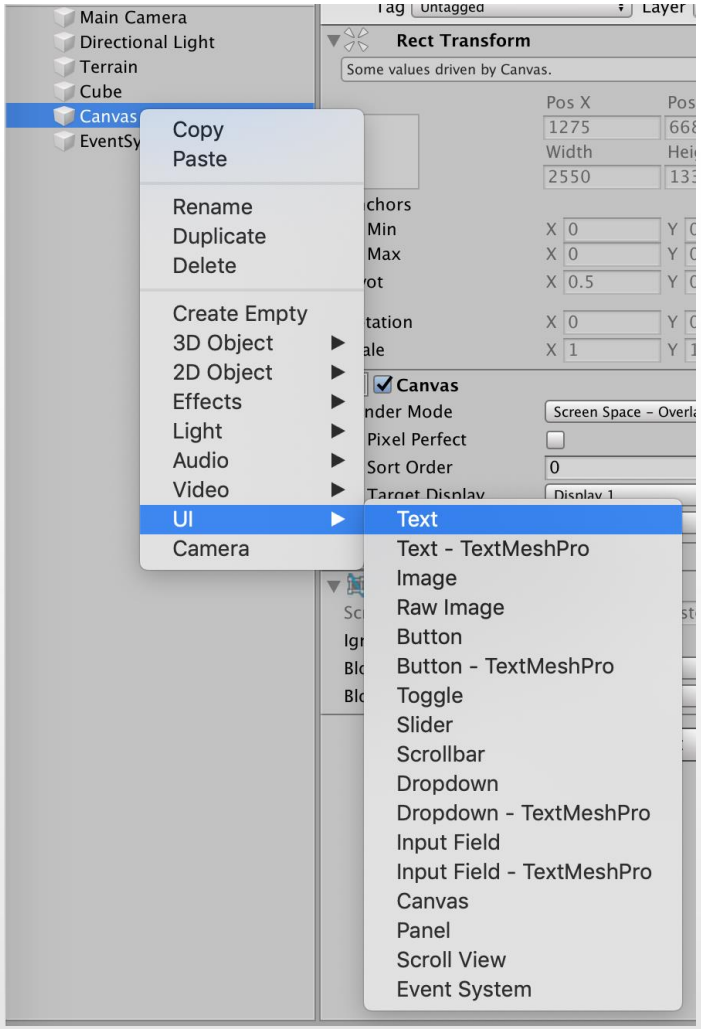
This is a canvas outline

And this is 500x500!



Adding Element to a Canvas

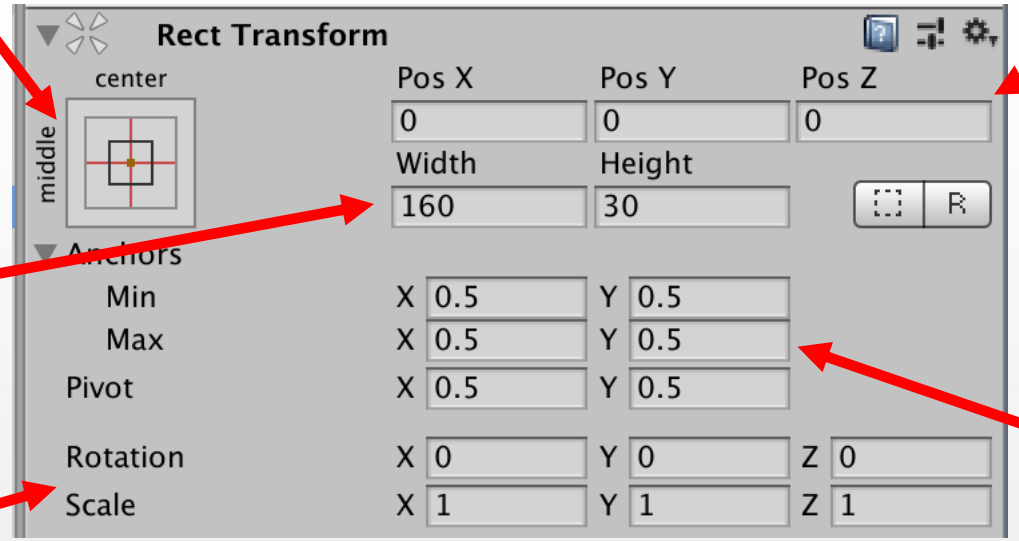
- You just need to add 2D child objects to the canvas
- Then, adjust the parameters of the child object
- Beware! The 2D transform for UI objects is kind of tricky to manage



UI Rect Transform

Reference point
(origin with respect to the canvas)

Gameobject position
(from reference point)



Gameobject size
(unscaled)

Usual stuff

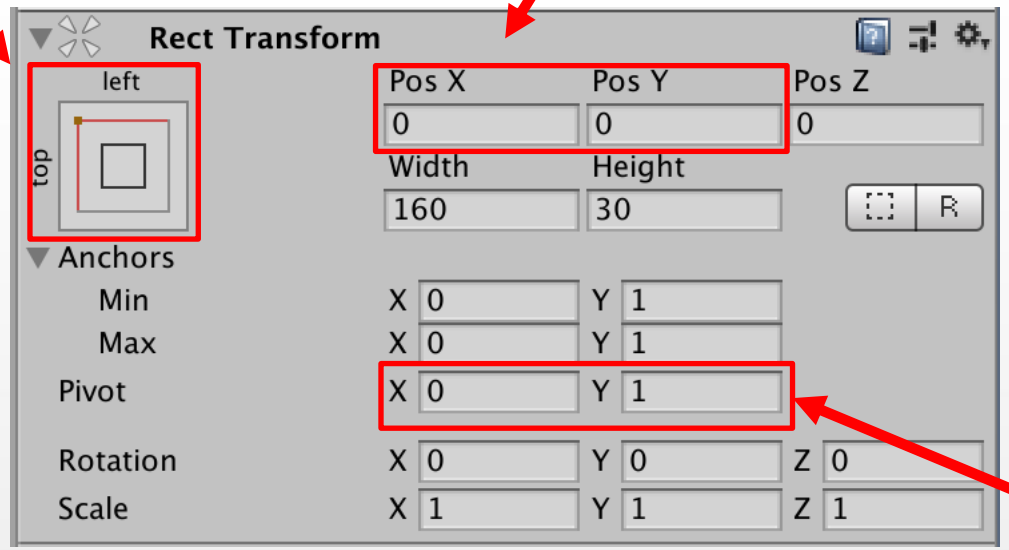
Where to put the
reference point of
the gameobject
(relative to object size)

TIP: If a text object looks pixelized, increase the object size (and the font size) and decrease the scale

Putting Something in the Upper Left Corner

Reference position
top left

Gameobject position is (0, 0) regardless of size

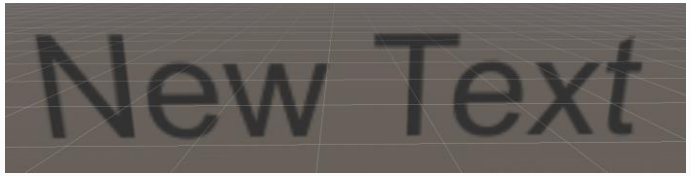


Gameobject reference point in its upper left corner

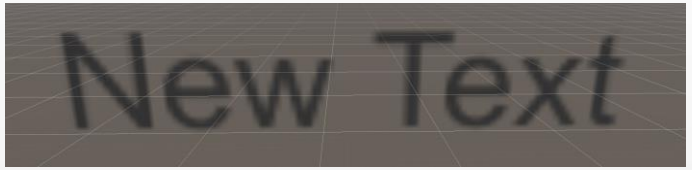
Size vs Scale for Text

Scaling operated on an image while size is pixel-wise and will leverage on vector graphics (and true type fonts)

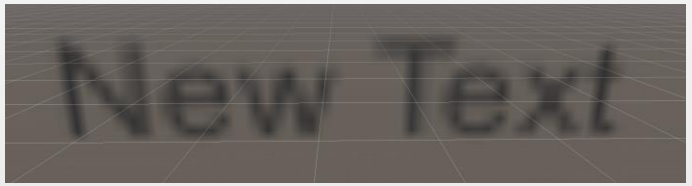
Size: 320x60
Font size: 50
Scale: 1



Size: 160x30
Font size: 25
Scale: 2

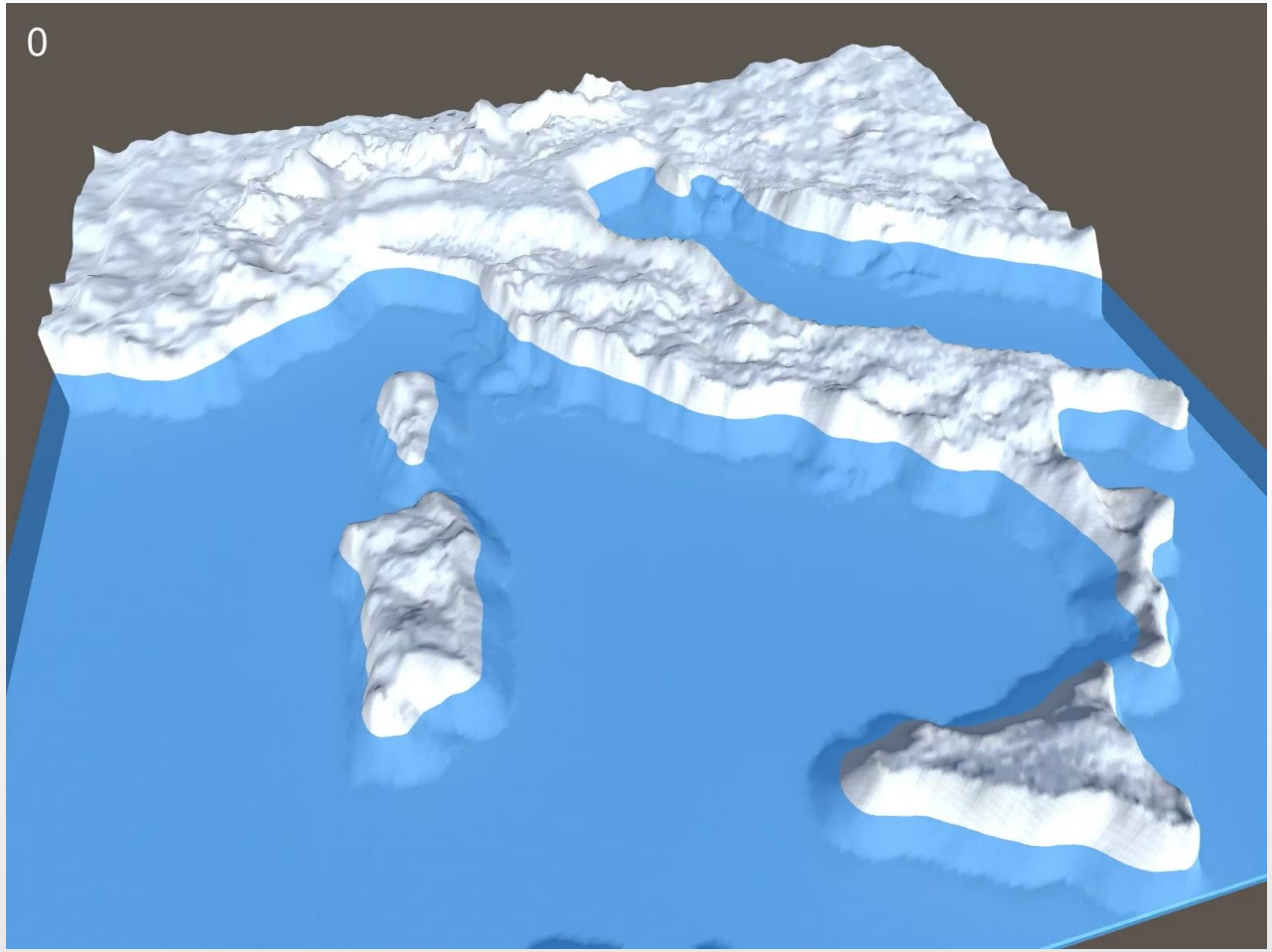


Size: 80x15
Font size: 12
Scale: 4



Keep this in mind!

Display Information on a Text Label



Display Information on a Text Label

Option 1: use a script in the pivot with a reference to the text label

```
using UnityEngine;
using UnityEngine.UI;

public class DisplayOrientation : MonoBehaviour {

    public Text UItag = null;

    void Update() {
        if (UItag != null)
            UItag.text = ((int) transform.rotation.eulerAngles.y).ToString ();
    }
}
```

Reference to the UI element

The y component in degrees of the orientation of the current transform is cast to int to suppress decimals

Display Information on a Text Label

Option 2: use a script in the text label with a reference to the pivot

```
using UnityEngine;
using UnityEngine.UI;

[RequireComponent( typeof(Text) )]

public class GetOrientation : MonoBehaviour {

    public Transform reference = null;

    void Update() {
        if (reference != null)
            GetComponent<Text>().text = ((int) reference.rotation.eulerAngles.y).ToString ();
    }
}
```

This will make sure there is a Text component in the same gameobject

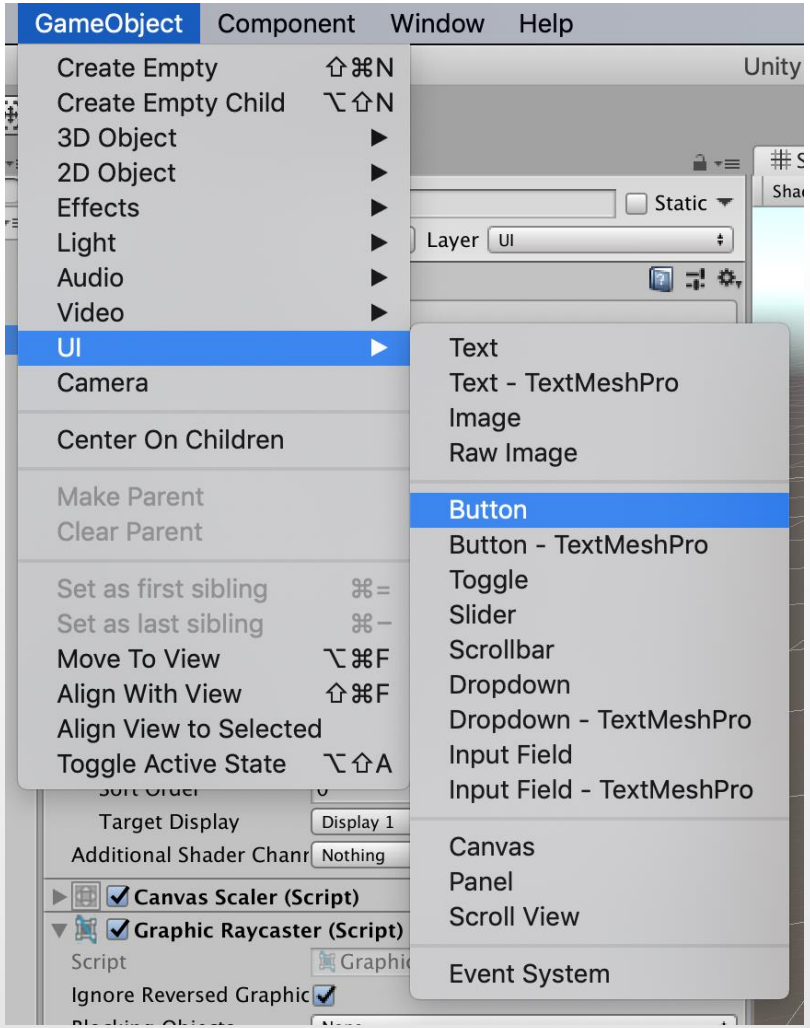
Reference to the pivot

We set the text field in the Text component

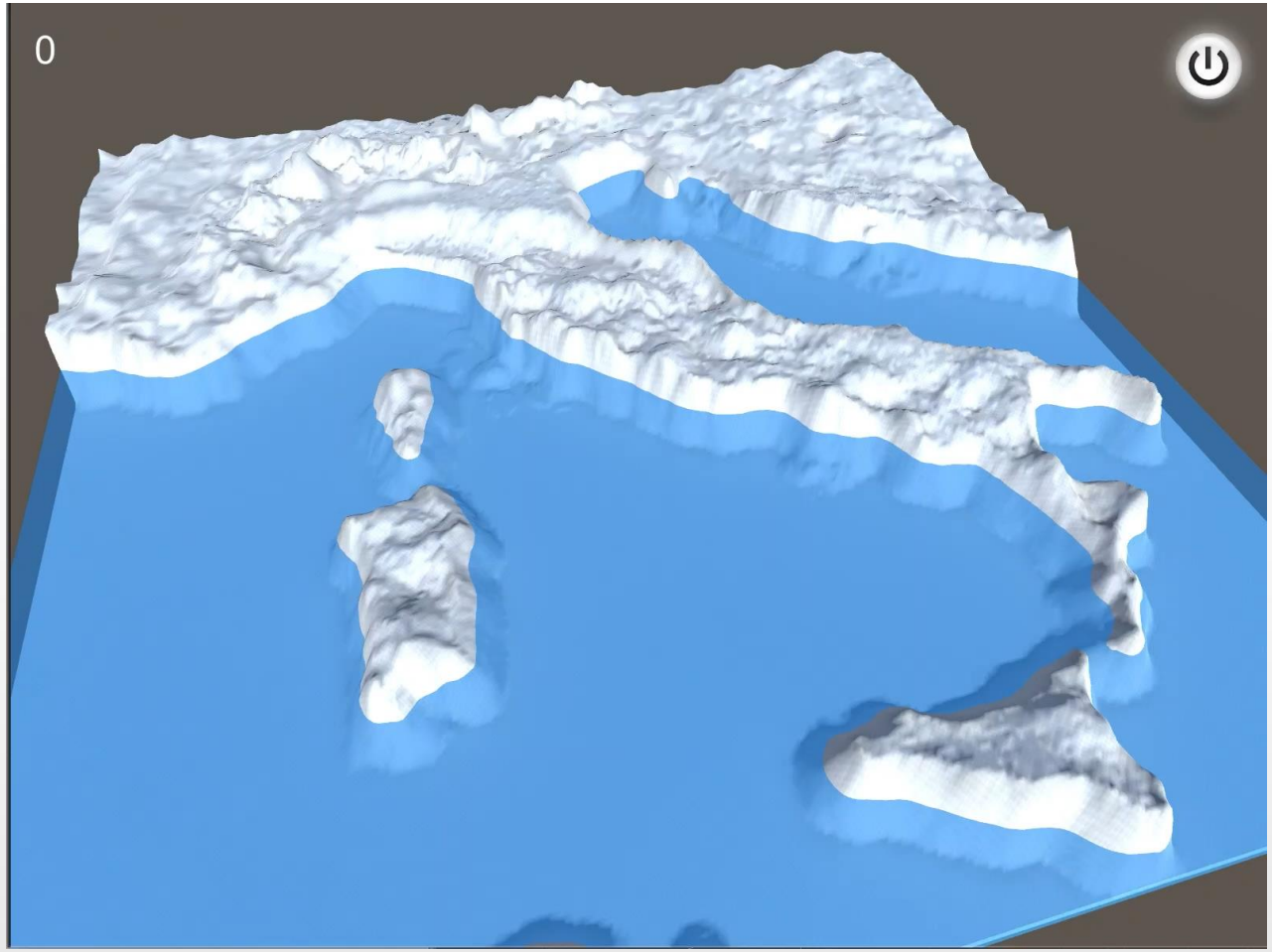
This time we are not using our own transform, but the transform of the pivot

Buttons

- A button is a texture-based element reacting to mouse clicks
- Differently from mouse clicks on scene objects, we cannot (should not) use a component implementing an OnClick method
- Buttons are events generators; we must subscribe to the event and provide it a delegate to call
- This is because multiple actions may be triggered by the same button or the same action may be triggered by multiple buttons



Start and Stop Button



Subscribing to Events

- It can be done in two ways:

1. From a script, using the AddListener method
   - Code will be more complex, but also self-contained
2. From the editor interface
   - Code will be simpler, but half of your game logic will be hidden in the interface

- IMPORTANT
  - An EventSystem gameobject must be in the scene for this to work
    - You had one automatically added when you creating the canvas; if deleted by accident, just instantiate a new one
  - NEVER do both, or your delegate will be called twice

Subscribing From a Script

```
using UnityEngine;
using UnityEngine.UI;

[RequireComponent(typeof (Rotation))]

public class OnOffButton : MonoBehaviour {

    public Button UIElement = null;

    void Start () {
        if (UIElement != null)
            UIElement.onClick.AddListener (ReactToClick);
    }

    public void ReactToClick () {
        Rotation r = GetComponent<Rotation> ();
        if (r != null)
            r.enabled = !r.enabled;
    }
}
```

We run this script on the pivot.
So we must be sure the Rotate component is there

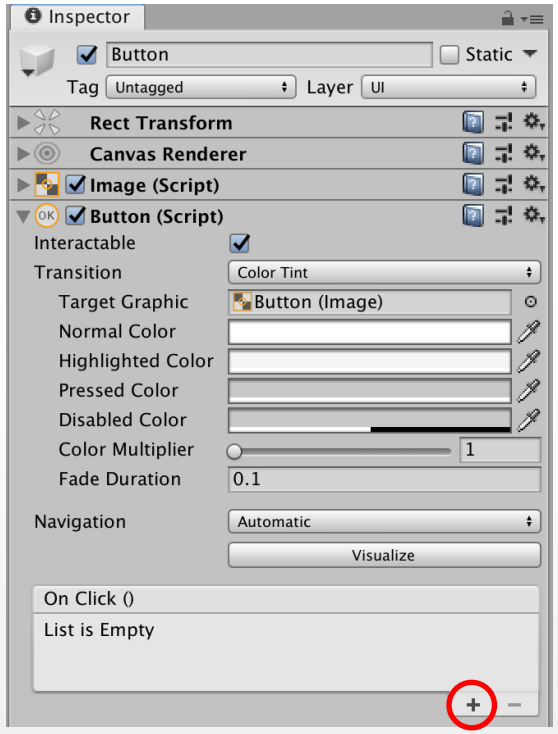
Reference to the Button

Ask the button to call ReactToClick when activated

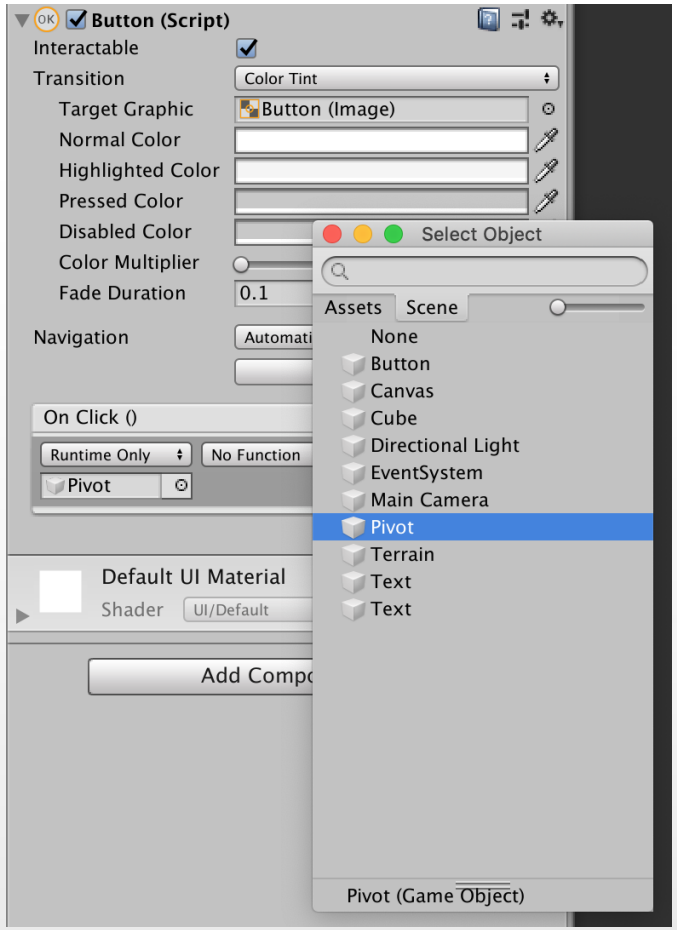
ReactToClick does NOT need to be declared public here, but it will turn useful in the next slide

When the button is pressed, we toggle the enabled flag for the rotation script

Subscribing From the Editor



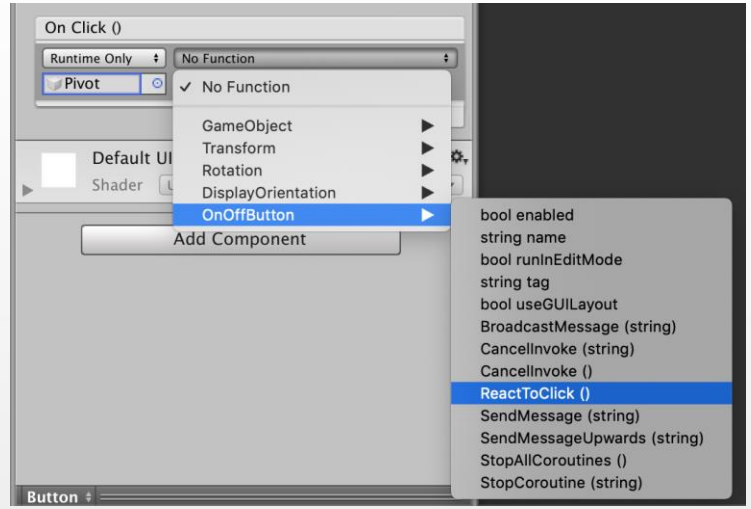
Press “+” in the Button component to add a callback



Then select the required gameobject from the scene

And then, inside the gameobject, the component and the callback.

To do this, ReactToClick MUST be declared public



You can do this using the same script as before, but remember to leave the UI Element field empty in OnOffButton, otherwise ReactToClick will be called twice!

Sliders

- Sliders are selectable linear input elements
- All setup strategies and recommendations from buttons still holds
- The only difference is that the delegate will now receive a float parameter representing the slider value
  - Min and max values as well as direction and sensibility for the slider can be configured
- Let’s try to add a slider to control the light orientation

Slider for Light Orientation

```
using UnityEngine;
using UnityEngine.UI;

public class SliderMovement : MonoBehaviour {

    public Slider UIElement;

    void Start() {
        if (UIElement != null)
            UIElement.onValueChanged.AddListener (ReactToSlide);
    }

    public void ReactToSlide (float value) {
        Vector3 v = transform.rotation.eulerAngles;
        v.y = 360f * value;
        transform.rotation = Quaternion.Euler(v);
    }

}
```

We run also this script on the pivot.
But this time we do not need the rotation script

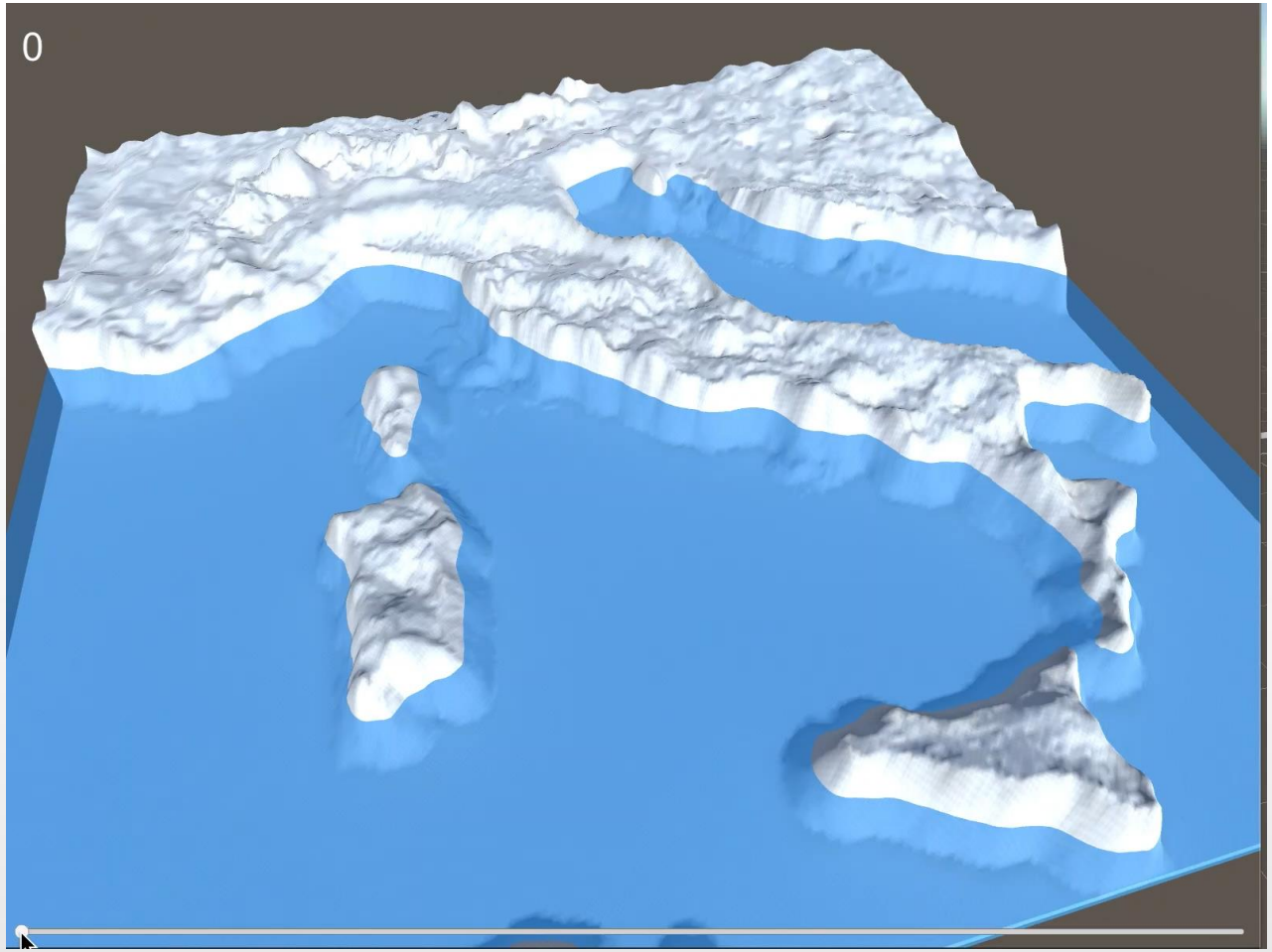
Reference to the Slider

Ask the slider to call ReactToSlide when its value changes

We change the y component of the rotation of the pivot based on the slider value.

We are assuming the slider will report values in the range [0,1]

Slider for Light Orientation



Gizmos

- Gizmos are visual clues inside the scene providing useful information for debugging
  - They are not visible in the game panel and will purged from the executable file

- While in the editor window, every component can draw gizmos by implementing the following methods
  - OnDrawGizmos
    - The code will be executed at every frame to draw the visual clues
  - OnDrawGizmosSelected
    - The code will be executed to draw the visual clues only if the object is selected

Gizmos

- Available gizmos include:
  - Geometric shapes
    - Lines, rays, cubes, and spheres
  - Images
    - Icons and textures
      - Texture are projected onto the UI
  - Complex shapers
    - Camera frustum and meshes

What Gizmos Cannot Do

- As we can see, the list is quite limited
  - Moreover, lines and ray are very thin and difficult to spot
- If we need to write text or use more complex shapes, it is also possible to use UnityEditor.Handles
  - Handles are provided to implement additional ways to manipulate objects, but using some of their draw methods will do no harm
- Another option is to use DrawMesh, but this will require additional assets

A Simple Gizmo

The coordinates of the gameobject are drawn as text one meter above the shape. When the object is selected, an eye icon will appear in front of it (so we will know where it is facing) and the shape will be outlined

```
using UnityEngine;
using UnityEditor;

public class VisualClues : MonoBehaviour {

    public Color wires = Color.yellow;

    void OnDrawGizmos () {
        Handles.Label (transform.position + transform.up * 1f, transform.position.ToString ());
    }

    void OnDrawGizmosSelected () {
        Gizmos.DrawIcon (transform.position + transform.forward * 1.5f, "eye.png");
        Gizmos.color = wires;
        Gizmos.DrawWireMesh (GetComponent<MeshFilter> ().sharedMesh, transform.position,
                             transform.rotation, transform.localScale * 1.01f);
    }
}
```

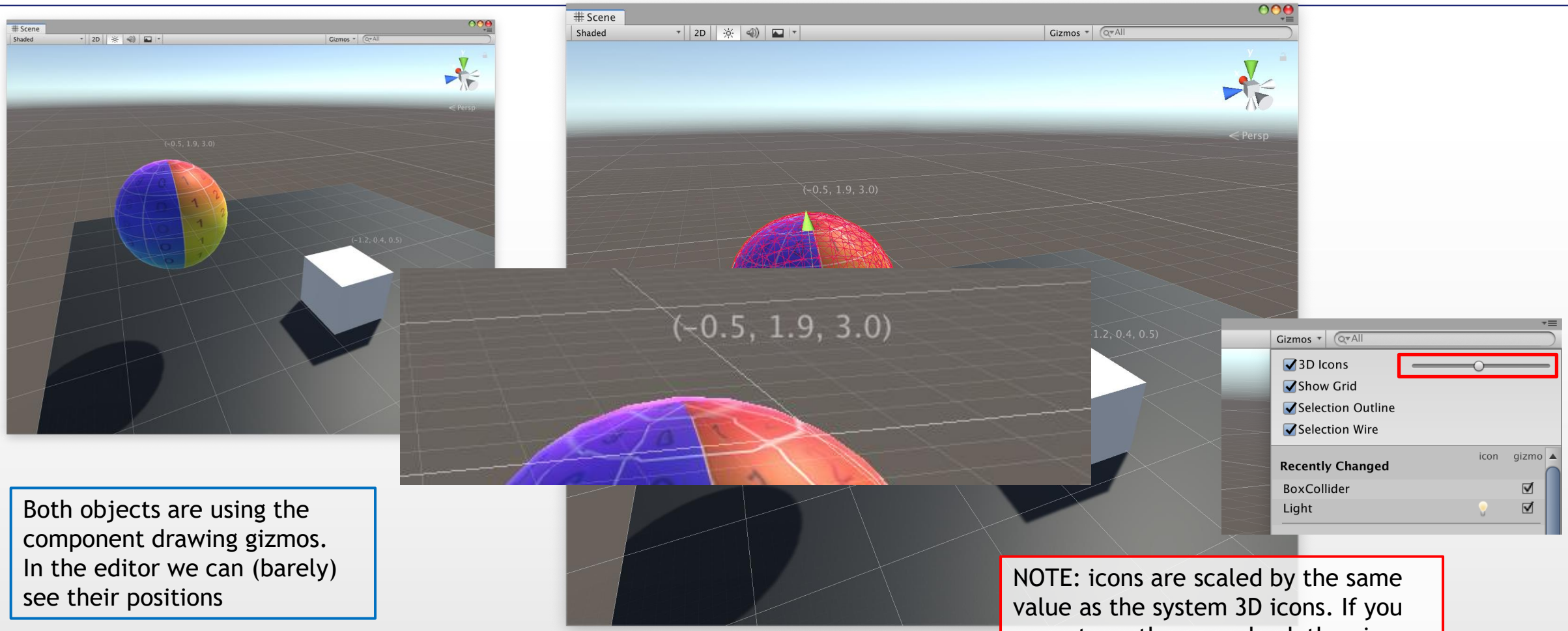
Wires is the color to use for the mesh outline

Handles.label is providing the text above the gameobject

eye.png MUST be in a directory named "Gizmos". This directory can be anywhere, but must have the right name

The outline is obtained by drawing the object mesh in the same location, with the same rotation, but 1% larger than the object itself

Gizmos in the Editor Window

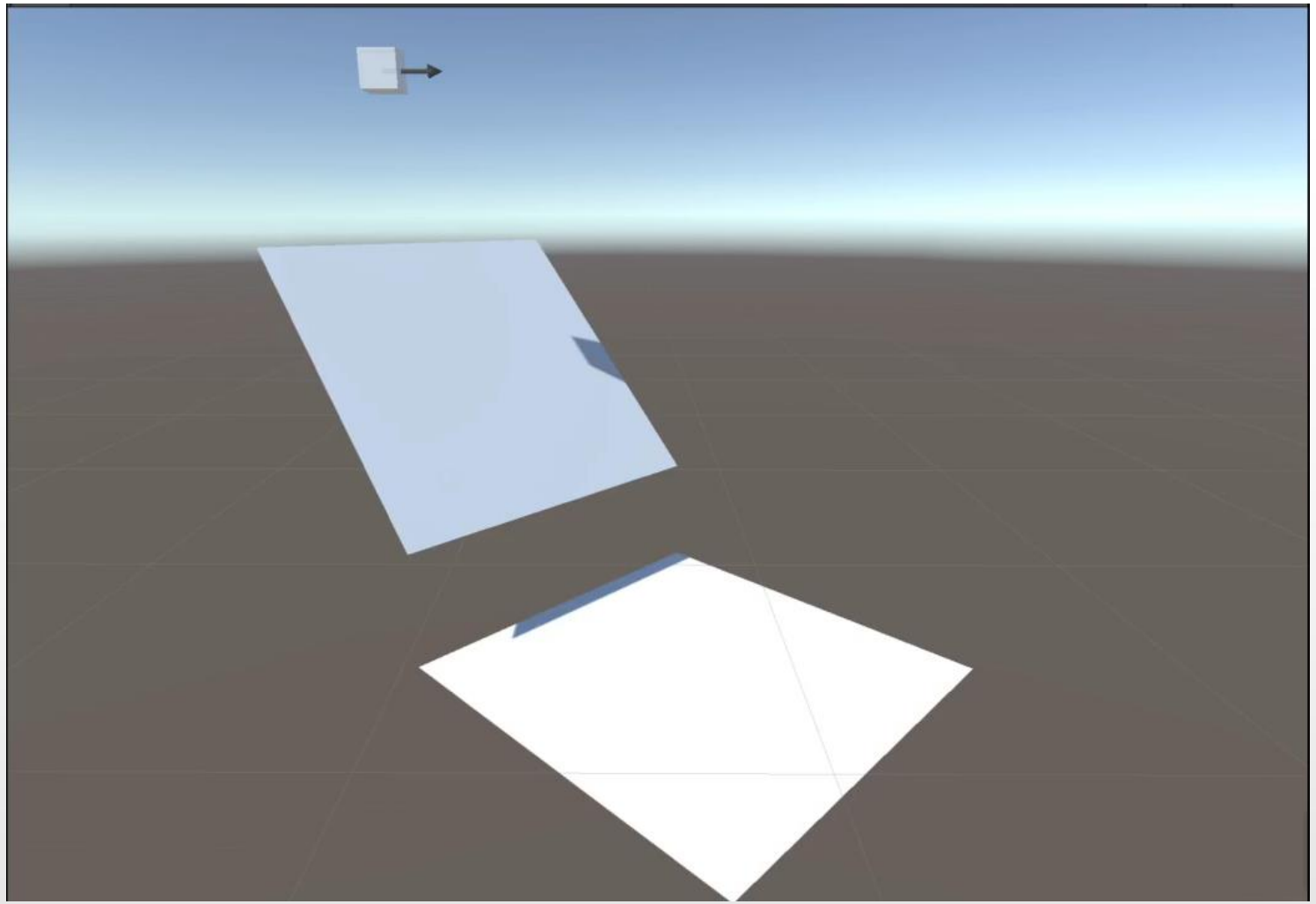


Both objects are using the component drawing gizmos. In the editor we can (barely) see their positions

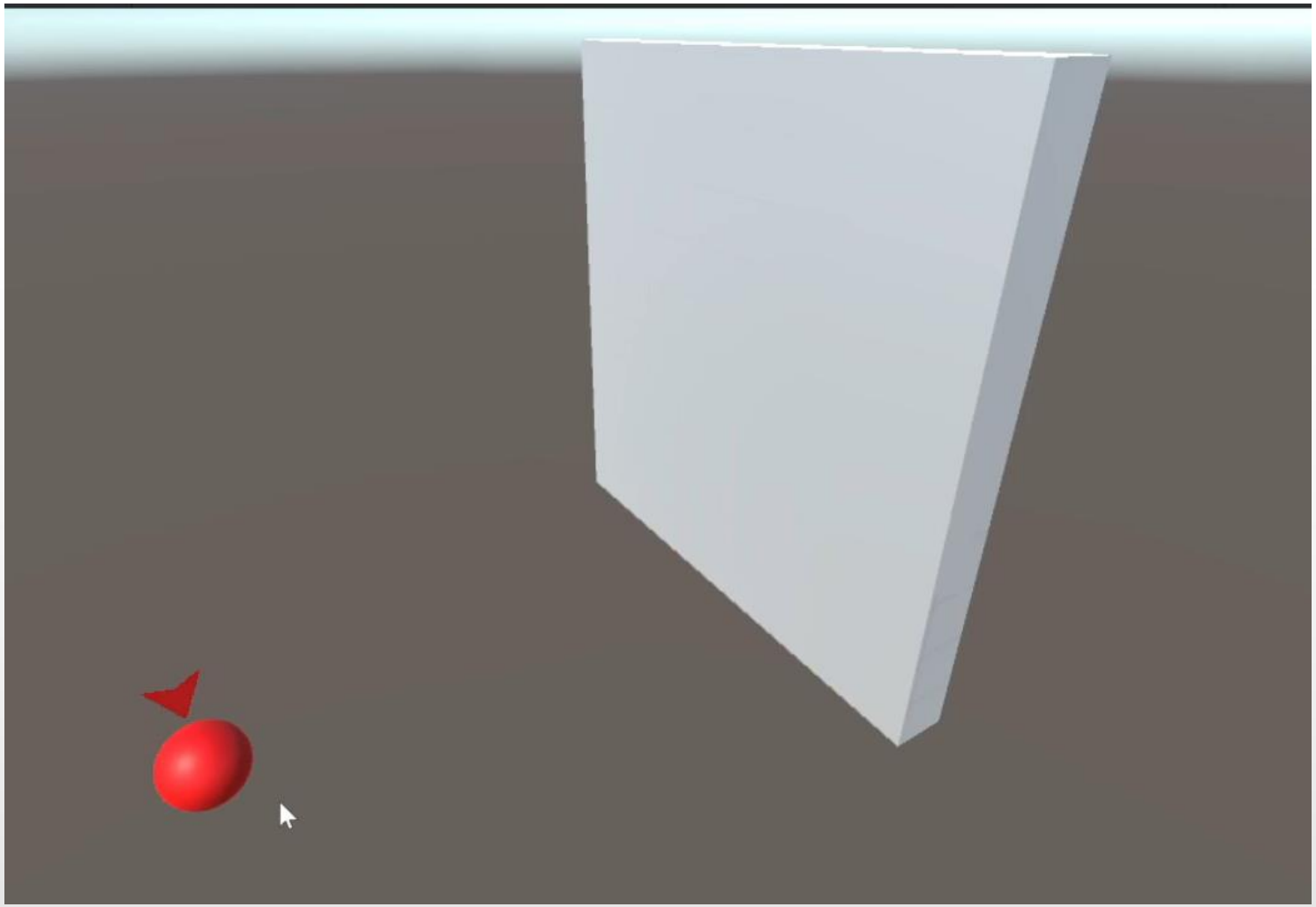
When one object is selected, outline and facing direction are displayed

NOTE: icons are scaled by the same value as the system 3D icons. If you cannot see the eye, check the gizmos drop-down menu un in scene panel

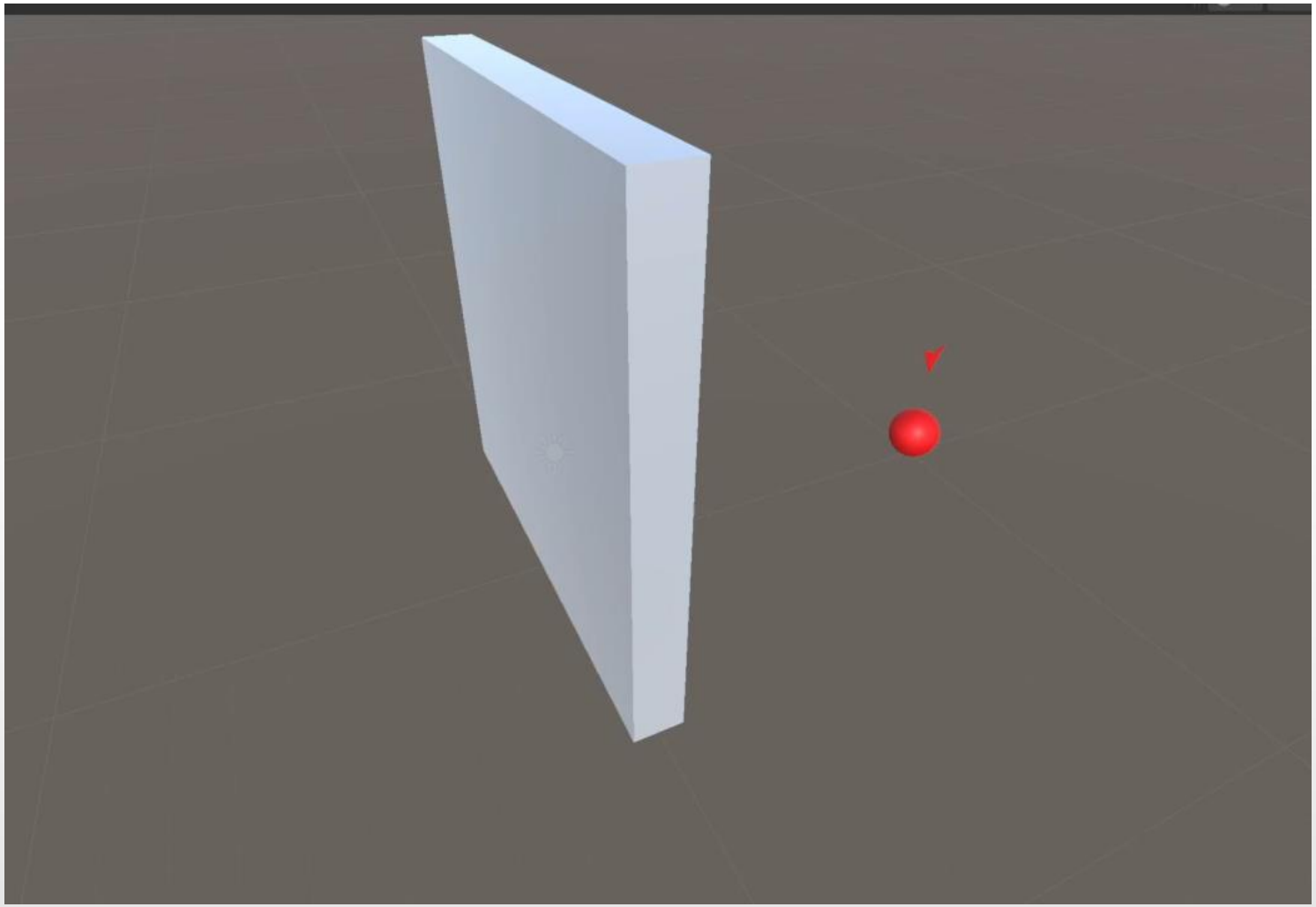
A More Complex Gizmo



An Even More Complex Gizmo (Scene)



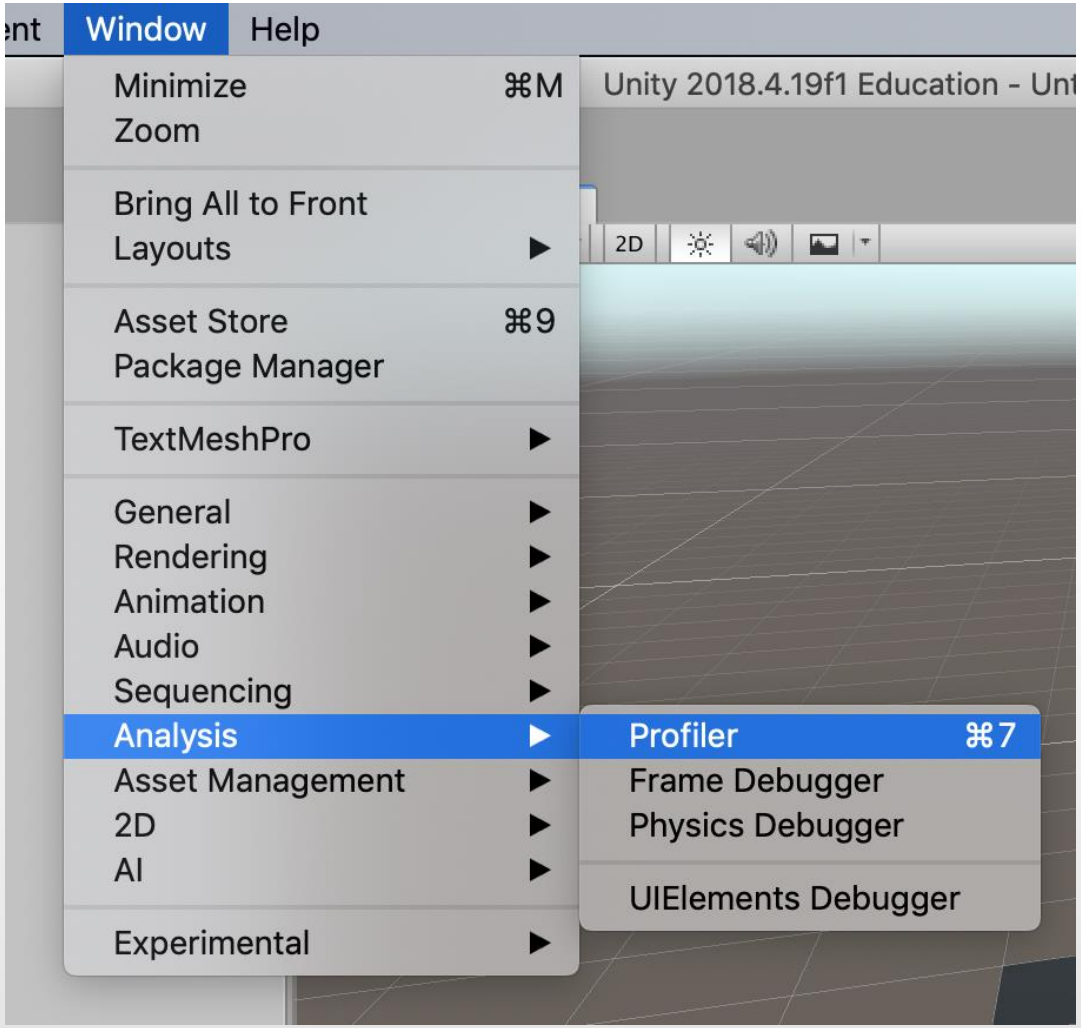
An Even More Complex Gizmo (Editor)



Performance Monitor

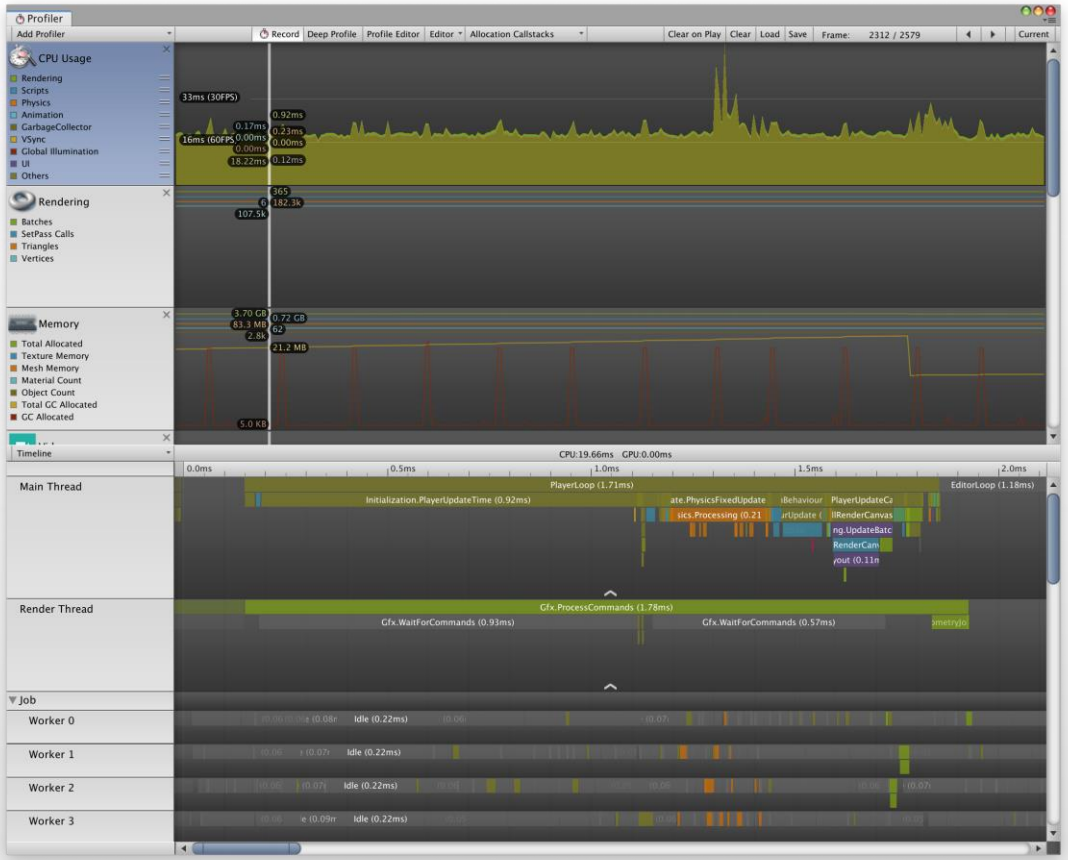
- Many times, we experience a low framerate with no apparent reason
  - Finding the piece of code slowing everything down is strategic to guarantee the user experience

- Unity provides us a way to check workload distribution in real time: it is the profiler panel from the window menu



Unity Profiler

- The profiler is keeping track of EVERYTHING
  - ... Maybe even a bit too much
- In the upper part there are sections for every element
  in the system: cpu, memory, graphycs
  ...
- Selecting a section will provide details in the lower part of the panel
- Clicking along the timeline will give information about a specific point in time



Profiling Your Own Code

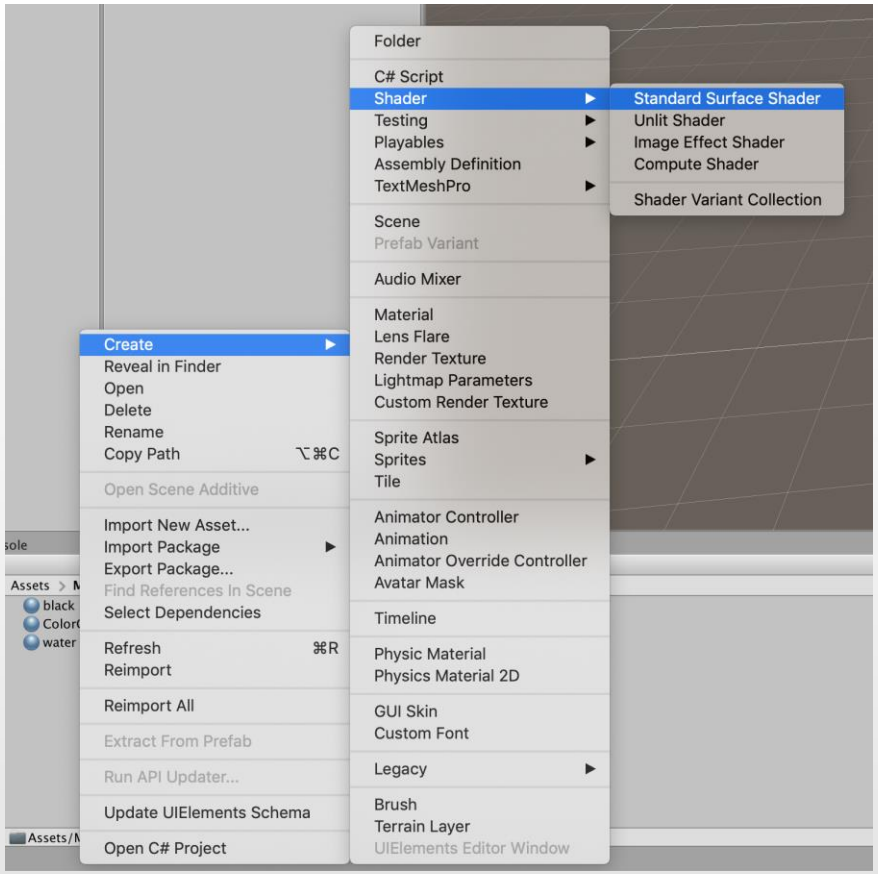
- It is possible to delimitate a specific section of your code
  - Because you might want to analyze an activity spanning several methods
- Profiling.Profiler.BeginSample(myTag)
  will set the beginning of the section to profile
- Profiling.Profiler.EndSample()
  will close the section opened by the last active BeginSample
- Results well be reported in the profiler under the “myTag” label

Shaders and Camera Filters

- As already said, shaders are programs for the graphics card describing how to draw an object
- Objects to be visualized have a mesh renderer. Each renderer has a reference to at least one material, and the material is a data structure holding parameters for the shader
  - And there are many shaders out there
- Sorry, we are not going to learn how to implement a shader in Unity here. We will just see what options are available if you want to do it

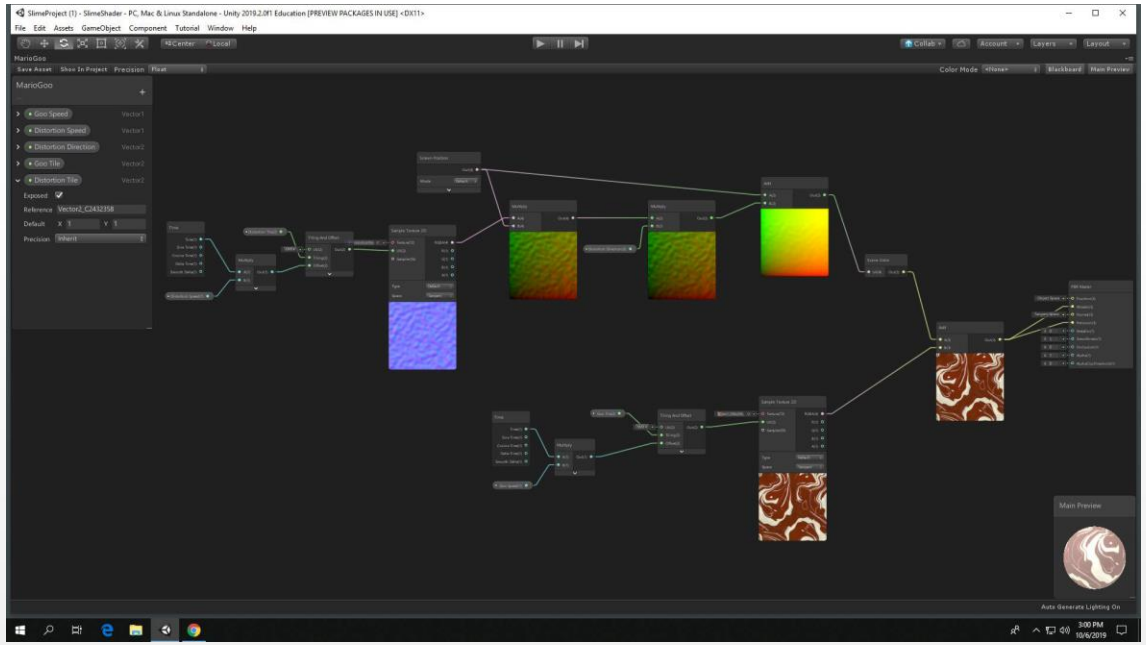
Shaders for Men: the Editor

- You can create a new shader in the asset database and then open it with the editor
- That is going to be a source file using a variant of the HLSL shading language
- PRO:
  - As soon as you save you will see the results in the editor
  - Quite (?) documented
- CONS:
  - There is absolutely no way to debug your code



Shaders for  : Shader Graph

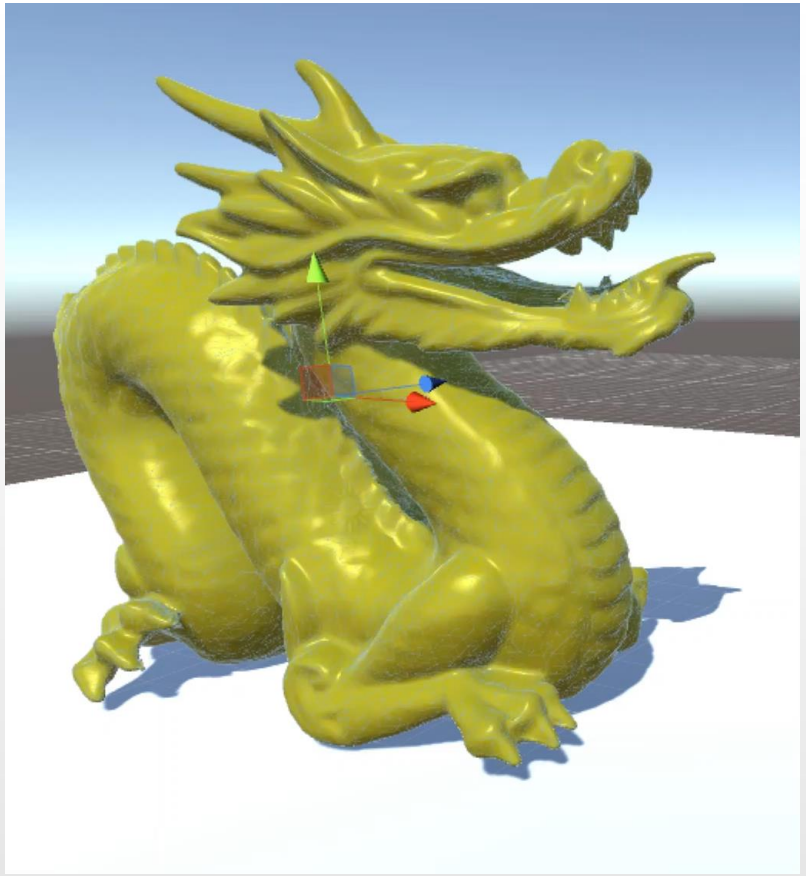
- Starting from version 2018.1, unity introduced shader graph: a visual tool to create shaders



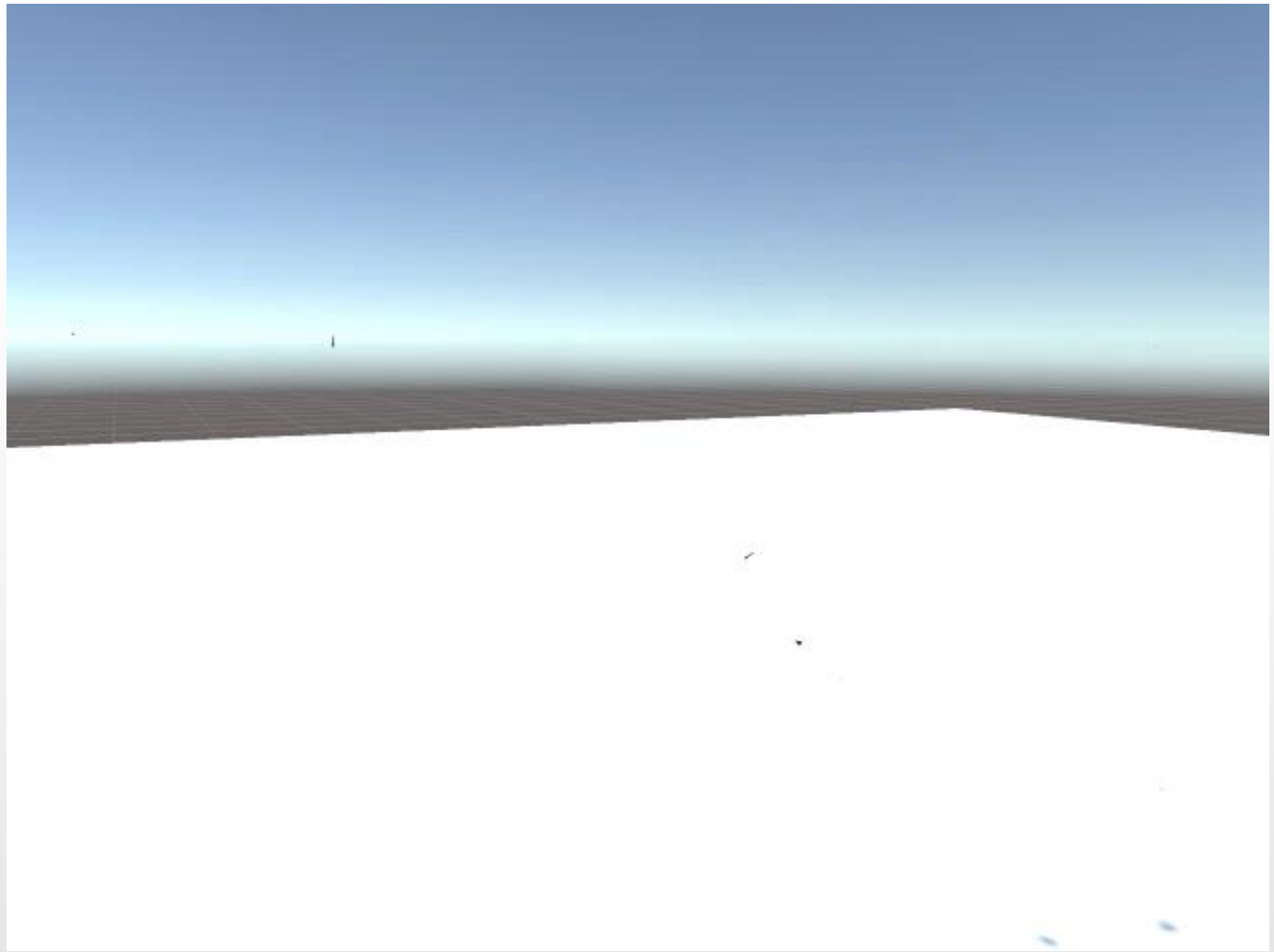
- Shader graph let you define a visual processing sequence by concatenation of simpler units such as color displacement or noise generation
- On export, shader graph will write the HLSL code for you
- PRO:
  - Easy to use
  - Gives more control on the processing pipeline
- CONS:
  - Less powerful than writing your own code, if you have special requirements

Why “for  ”?

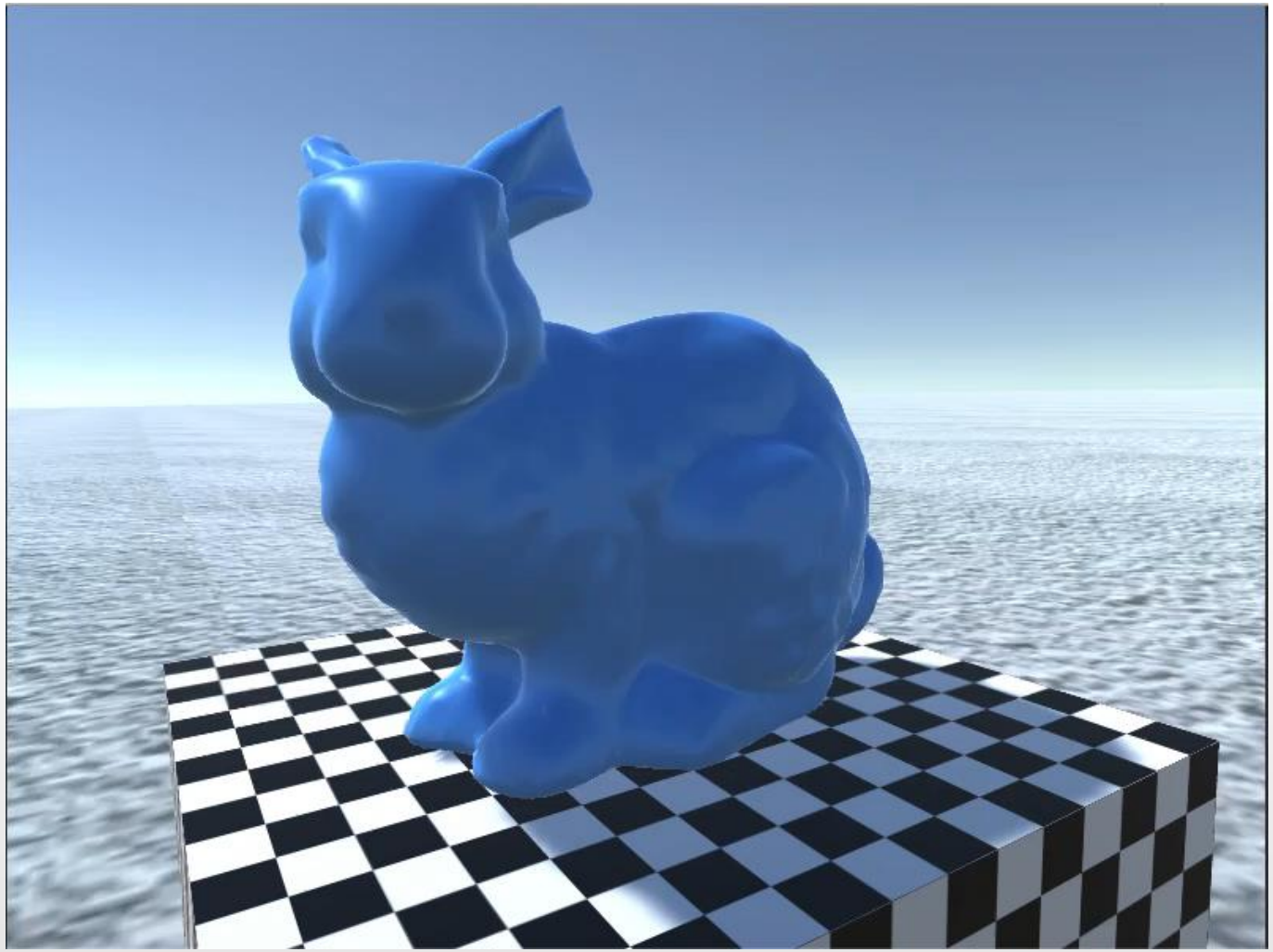
- Because ShaderGraph can be convenient to change what you see in term of texture and color
  - If something more complex (read powerfull) is required … then you need to write your own code



A More Complex Example



An Even More Complex Example



Camera Filters

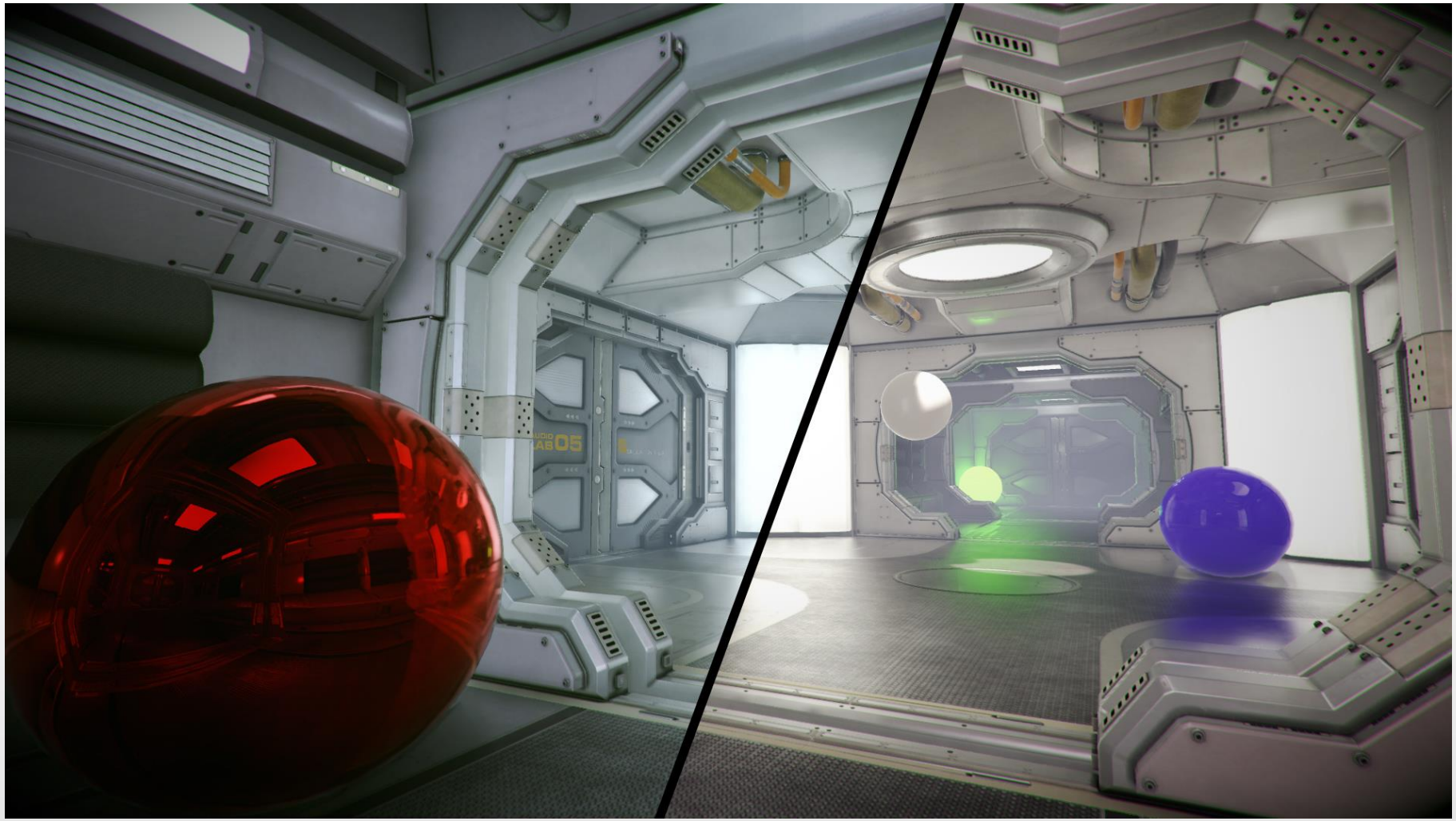
- Camera filters are post-processing operations applied to the camera buffer before the image appears on the screen
  - Post-processing effects can simulate film properties or add additional layers, such as rain or fog

- Unfortunately, post processing depends on the Rendering Pipeline in use
  - Built-in RP does not provide its own post processing
    - You must use the Post-Processing Version 2 package from the asset store
  - Universal RP has its own post-processing solution
  - High Definition RP has its own post-processing solution NOT COMPATIBLE with HDRP

- Unity is including in each RP some filters you can use out of the box to improve the look of your game

- The creation of a custom camera filter is very like the creation of a custom shader

Example of a Camera Filter



Assets: Early and Late Binding

- Bad news: running your game in the editor and as a standalone executable IS NOT the same
- In the editor, all the asset database is still there, so you can load whatever you need at runtime
  - Might be useful to create new assets at runtime and for PCG
  - This is the equivalent of late binding taking place in programming languages
- In an executable, only the subset of the asset database referenced in the scene will be bundled. Trying to loading an asset unreferenced at build time will produce and error
  - Referencing is not the equivalent of early binding in languages, but, in this way, you might remember it

This is NOT the only difference, to be honest.
But it is one you could easily bump into

Forcing Early Binding

- For many assets, it can be useful to force unity to bundle them in the executable regardless if they are referenced or not

- Usually, those are resources retrieve by name (a string) rather than by ID
  - Textures and icons
  - Modules
  - Shaders
  - Multimedia

- Luckily enough, there is a simple solution: put everything in a folder named “Resources”
  - They will be bundled regardless of references
  - It is not important where is this folder; only its name matters
  - You can have more than one resource folders in different locations

Building

- Building your game is the last step of your production process
  - This is a lie! There is still a looooong way to debug when you start building

- You must select a platform, and any platform will provide a huge array of specific options
  - Thus, no build is the same as the other
  - **There is no way to write all your code and then deploy on many platform seamlessly!**
    You will have to take the platform into consideration in your scripts

Suggestions About the Building Process

- Read all the possible flags
  - You will be surprised
- Remember to set icons, cursors, and splash screen (when applicable)
- Never underestimate the size of your resources (especially on mobile platforms)

- PC and Mac; you must take a decision about windowed or full screen, because your UI will depend on that

- Android and IOS
  - You must take a decision about portrait, landscape, or both. Because it will influence you user experience
  - Be prepared to suffer a bit for testing on real devices; sideloading may get time consuming
  - Always read about best practices from the vendor

Lots of Things are Left for You to Read

- https://docs.unity3d.com/Manual/index.html
  - Your jumpstart
- https://docs.unity3d.com/ScriptReference/index.html
  - Scripting APIs *with examples*

- https://www.youtube.com/channel/UCifiUB82IZ6kCkjNXN8dwsQ
  - Nice tutorials (**the character controller one is a must see!**)
- http://catlikecoding.com/
  - Tutorials on advanced topics
- https://www.alanzucconi.com/
  - Shaders

- https://forum.unity.com/
- https://stackoverflow.com/
- https://www.reddit.com/r/Unity3D/
  - Many examples and a lot of help … but use them with (a lot of) caution!

Happy Hacking

Dario Maggiorini (dario@di.unimi.it)