Table of Contents

Unreal Engine 4 compared to Unity

This guide is viewable on https://teunw.gitbooks.io/unity-compared-to-ue4/content.

For this school course I've investigated the differences between Unreal Engine 4 and Unity engine 5. This comparison will mostly focus on the following things.

- The graphical options of Unreal Engine and Unity.
- The development within the engines.
- Comparing the way of programming.

As a start, the graphical features of both engines will be compared.

Requirements

This guide assumes the reader has programming knowledge and a basic knowledge of game engines and their workings.

Graphics

In any game engine to graphics are arguably the most important feature, this is what the users will see. People can't play your game when they aren't shown something, simple enough. Graphics have greatly improved over the last decades, going from simple squares to huge complex worlds that look where telling the difference between simulated photos and actual reality is becoming really hard. Take a look at the picture below.

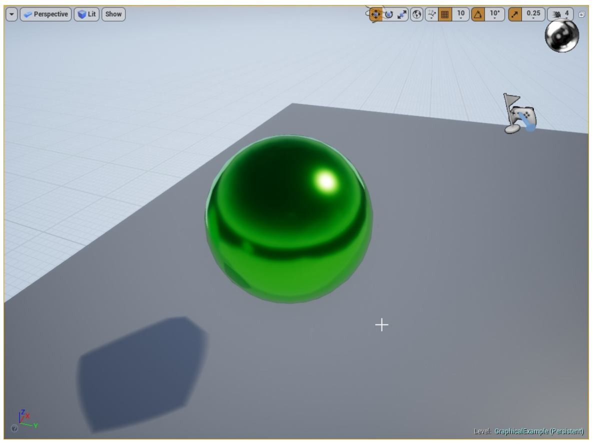
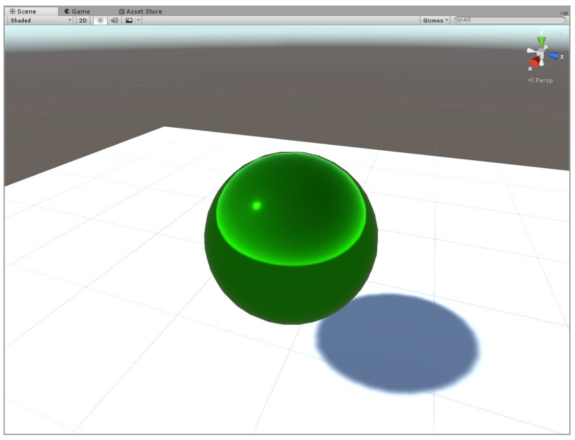
Both engines feature very advanced graphical capabilities, this is seen the examples below.

Unreal Engine	**Unity Engine**
DX12 Tech Demo LIONHEAD STUDIOS	

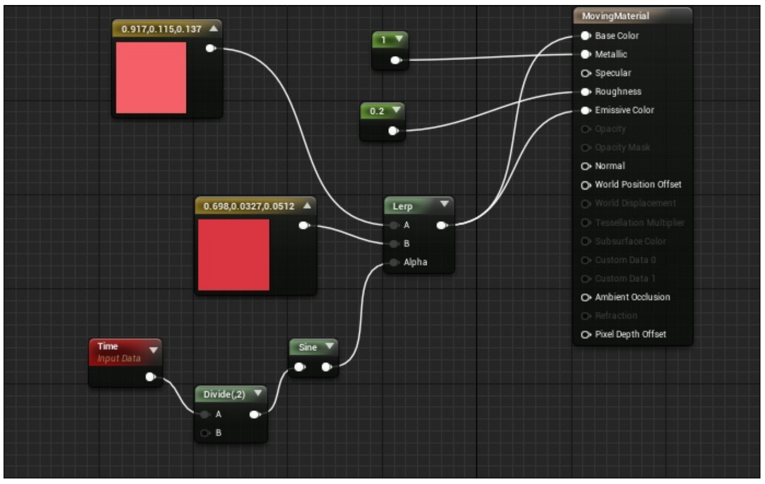
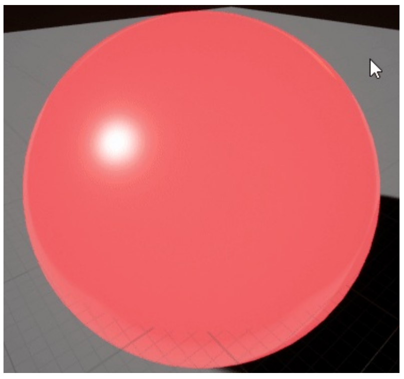
I'm not (yet) able to produce graphics like this, so will focus on more basic examples. It does show that both engines are capable of producing stunning visuals.

Materials

Both engines use so called 'materials' to apply a look to objects. The engines share material variables, such as a base color, metallic parameter and roughness variable. Materials also can contain textures, which can be used for looks, normal maps and other effects.

Unreal Engine	Unity Engine
	

Materials have the about the same parameters in both engines. These parameters are adjustable in both editor and also in programming. Unreal Engine supports a node based system for defining how a material should be structured, similar to how it works in 3DS Max. Unity's materials can have parameters that can then be modified by C# (or Javascript) code. Personally I find Unreal's system better, because it is more easy to seperate the materials. In Unreal Engine 4 you can also add code to materials, as shown below.

The material code	Result
	

In Unity you could have the same material by changing the color parameters in C# like this. Of course you would have to create the material first.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Start() {
        Renderer rend = GetComponent<Renderer>();
        rend.material.shader = Shader.Find("Specular");
        rend.material.SetColor("_SpecColor", Color.red);
    }
}
```

Of course there is much more possible with materials, for that I suggest that you go experiment with materials in both engines yourself. You can easily change parameters to see the results instantly.

Programming

When making a VR experience, programming is an important part of the experience. Without some programming, the only thing that can be done is maybe looking around or at one object without interaction. Programming your experience can be done by either blueprint or C++ in Unreal Engine and C# in Unity.

Unreal Engine

Programming in UE4 gives you the option to use blueprint, or C++ with custom features like macros. Blueprint is made for simplicity, by connecting nodes together you can program small games and simple demo's. When programming a bigger application, you'll likely need C++ for something. C++ is also needed when you integrate other services such as online multiplayer with Steam. Blueprint is much simpler of a language because it's very visual and doesn't require much programming knowledge, C++ is more efficient and also is sometimes more easy when dealing with stuff like loops.

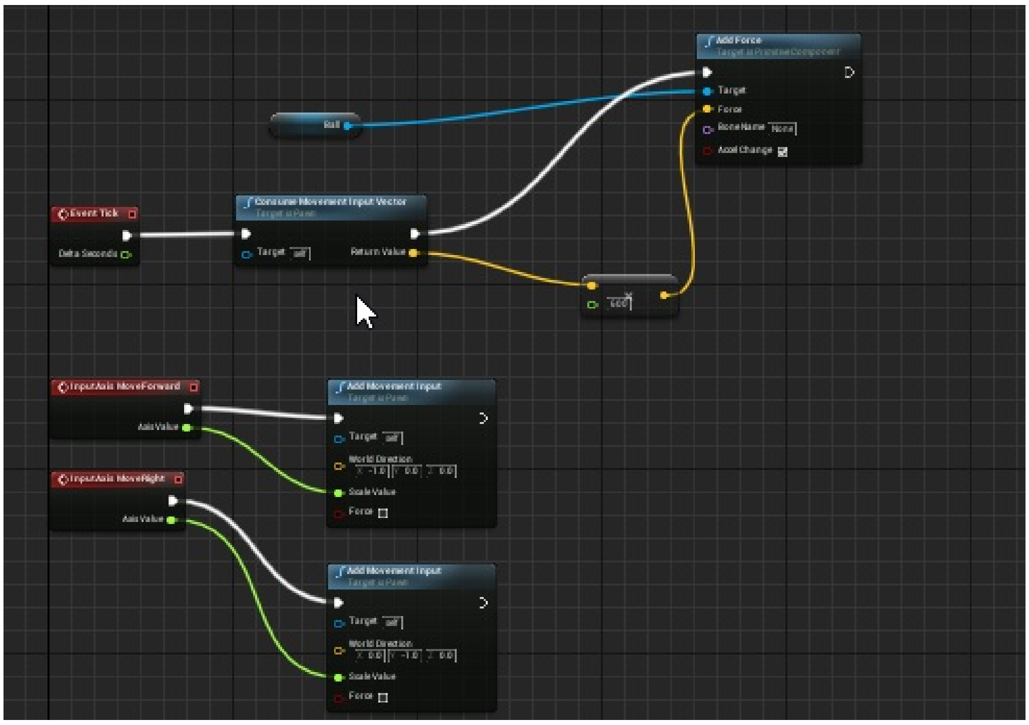
When programming in Unreal Engine in C++, you do need to use headers of the engine within your code to have Unreal work with your code, that might look like this. The `UPROPERTY` and `UCLASS` are examples of these headers.

```
UCLASS()
class VC2_PROJECT_API ARoughnessVariationExample : public AStaticMeshActor
{
GENERATED_BODY()

public:
    // Sets default values for this actor's properties
    ARoughnessVariationExample();

    UPROPERTY(BlueprintReadWrite, EditAnywhere)
    float CurrentRoughness = 0.00f;

    UPROPERTY(BlueprintReadWrite, EditAnywhere)
    float RoughnessIncrement = 0.001f;

    UPROPERTY(BlueprintReadWrite)
    bool RoughnessGoingUp = true;

    UPROPERTY(BlueprintReadWrite, EditAnywhere)
    FName Property;

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;

public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;

};
```

I found it quite tricky to use these macro's at first, because I forgot how to use them quite easily. And since these contain critical codewords within them I quite often had to figure out what I did wrong as the error messages didn't help either. Also, compiling C++ takes a long time (about 20s) every time you want to test your code.

Blueprints

What I did like about the Unreal Engine is the use of blueprints, blueprint is a visual scripting language specifically made for the Unreal Engine. The language works by connecting nodes together as seen in the picture below.



I found the language quite easy to work with, but it took some getting used to as you have to know the engine as well. In the end it still uses the same methods you would use when writing C++, only that it's a lot simpler in my opionon.

Unity Engine

In Unity the only option for programming is done through an actual programming language, but in Unity you have 3 choices.

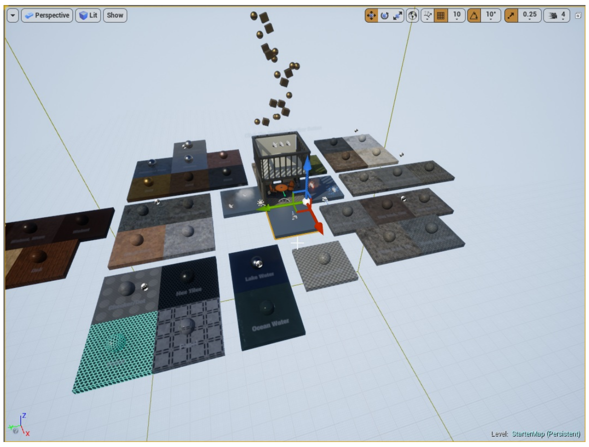
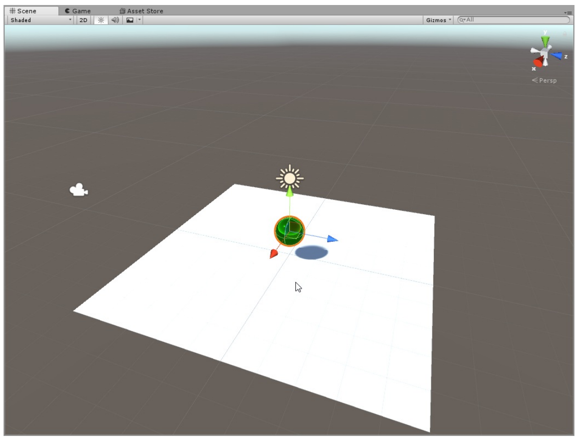
- Boo, a language developed by Unity Technologies.
- Javascript/Unityscript, a language originating from the web.
- C#, a language developed by Microsoft.

Most developers use C# as their programming language of choice, I also find it to be the most complete and versatile language, containing a strong typing system and a great set of standard methods. Unity also uses the standard C# types in the editor, in Unreal Engine you have to use the custom `FTEXT` of `FSTRING` classes sometimes.

There are also more examples available for Unity as it's a bit more popular and also has been around longer. The fact that it has been around longer also gives it more room for deprecation of old code which has happended a lot (Mainly with old variables and GetComponent).

Level Editor

Both engines feature a 3D level editor in which you can edit your levels and immediatly see results about things you did.

Unreal Engine	Unity Engine
	

Both editors feature full control over objects in your scene, in Unreal Engine you also have different setting possibilties for your scenes. This makes it so that changing gamemodes and other settings can be done with relative ease. Unity gives you full control over the scene, where a completely empty scene won't do anything, it doesn't give the player anything to work with.

In both editors, objects are able to rotated, translated and scaled to your liking. In both editors you can also set variables for different objects. You can also create animations and look at the different content that you have in your project.

Conclusion

I liked the overall feel of the Unreal Engine, but Unity's engine is very easy to get started with. Unity's wide support also gives it an edge. But that doesn't mean I think either engine is better.

If you want to get started making a game as a beginner, I would say that you should pick Unity because it is a easier engine to get started with. There are also much more resources and tutorials available for the engine.

Unreal is in my opionon a better engine if you want to do some more heavy lifting with regards to graphics, since you can modify a lot about the engine itself (since it's open source). The wild variety of editors is also a strongpoint.