Real Time Rendering – DirectX Scene

The aim of this assignment is to create a 3D Castle Scene, which implements 2 features, all using DirectX 11. Them being:

* Texture mapping filtering, normal mapping and environment mapping
* Water Effect

In addition to those effects, we need to include one of the following features:

* Foliage Effect
* Particle System
* Lighting and Glow effect.

For this assignment, I have decided to go with a Foliage effect, as well as a Particle System.

# Texture Mapping

Texture mapping is the process of placing a texture from a 2D image and placing it upon a 3D model, or on a 2D quad.

In our scene, we use texture mapping on every 3D model to produce a textured item to place within our scene.

To texture an object within our scene, we need to load in an image using the “Texture” class, then use a function called “getShaderResourceView”, which we use to create an array that stores the texture. Later, when we attempt to apply a texture to an object, regardless of if it is 2D or 3D, then we need to have it stored as an array. The line below shows off us creating this array.

Using the castle as our example, we call the Model constructor to load in our model, and attach the texture array to it, therefore attaching the texture to the object.

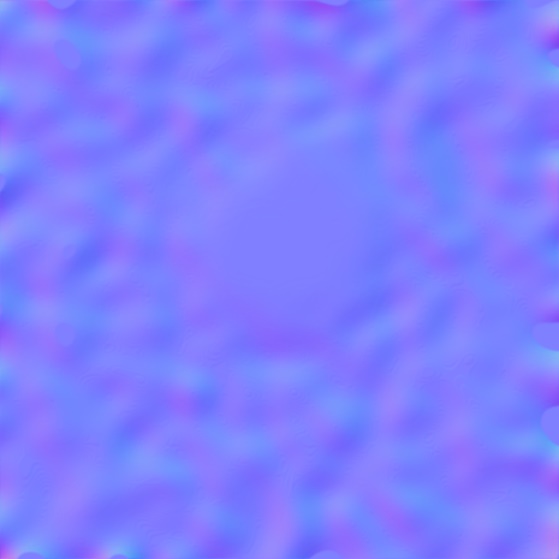
By then rendering the castle to the scene, the ending result is a castle that is fully textured.

Applying texture to the object can be done in multiple “layers”. This is known as multitexturing and can be used to apply different effects to the same object.

This texture is the texture we apply to the castle mentioned earlier. Each section on this image is mapped to a certain area on the 3D model itself. For example, the section roughly in the middle would map itself onto the roof of the castle.

This texture on it’s own provides a fairly basic style effect.

For our terrain, we use 2 other texturing styles, known as height and normal mapping.

This texture is known as a height map. This style of image is used in elevation styled maps. Each pixel in this image stores a value which is used to elevate our terrain. Generally, the whiter the area, the higher up that part of the terrain is. As shown in the example on the left, the black spot in the middle would be much lower than different areas within our terrain. If we were to place a few white pixels in the middle of this, then the middle would then be elevated up, leaving us with a donut-like hole.

This purple style map is what’s known as a normal map. In 3D graphics use normal mapping to provide a bump like effect that “tricks” lighting into providing more detail than what is actually there. The map on the left is our normal map for our terrain, and as a result, from the correct angle it would then make it look like there is more detail in the ground than there actually is.

# Water Effect

A water effect in 3D graphics is one of the harder effects to perform, since to perform realistic lighting, we need to simulate 2 techniques, refraction and reflection. For our 3D scene, we can take a shortcut to simulating these by using environment mapping. This is performed by applying the scene to the inside of a cube. Then, using a normal vector, the water can then see what colour is seen at certain directions. Although this method has issues with constant changing scenes and reflecting objects that are close together, it is however a fast method, and as a result, used within games.

Here is how we have our water within our 3D scene. The aim is to have is as a moat, surrounding the castle, which can be crossed over using a bridge.

As shown, the water provides a surface that reflects off of it, depending on which way the camera is facing. In addition to this, the water is not a direct reflection, but instead providing a reflection that has a filter over it, as if the water itself is not still, but being moved by another force, such as wind.



# Foliage

What this effect does is that it gives certain items within a scene a rustling effect. An example of this would be when a gust of wind hits a field of grass and the grass moves a bit along with it.

In our scene, we implement foliage effects in for our trees, as well as the grass on the ground of our scene. Both of these use the same styled effect, which sways back and forth, simulating wind blowing across it.

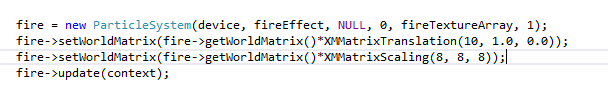
The image on the right shows what the trees look like in our scene:



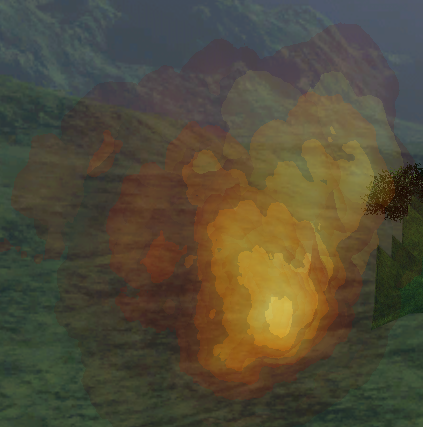
Each blade of grass, and each “branch” on the tree is loaded onto a single quad. Each quad is then attached the tree, then made transparent through the use of alpha blending. If we didn’t enable alpha blending, or set it up incorrectly, then we would see each quad, and therefore have a black background behind each branch.

# Particle Effect

A particle effect within graphics is a technique that uses many small 3D textures that when grouped together, produce a more filling graphic. An example would be a pile of smoke, where each texture would have a small texture of smoke, and when all grouped together, would result in a smoke like effect.

To create a particle effect in our 3D scene, we need a library included in our project called “ParticleSystem”.In addition with a respective shader given to us, this performs our necessary calculations, and therefore produces an object that “spews” out multiple textures. 

This here is the fire particle system we have in our scene being created, and the image below shows off the fire within our scene.



Much like the foliage, the fire uses alpha blending to make the textures on the fire semi transparent. If alpha blending was not enabled, then we would see every quad with the same fire texture being shown in the scene, then disappearing a second or so later. Since we have alpha blending enabled and set up correctly, we get the result above.

# Performance Testing

After each element has been added into our scene, I compared the performance slowdown in 2 methods. Both in terms of SPF (Seconds Per Frame), and FPS (Frames Per Second). The difference between the two is that SPF times how long it takes for 1 frame of our scene to be rendered, and FPS counts how many frames in total have been outputted in the span of 1 second.

Each test was performed after each feature has been implemented, in the order of Terrain, 3D Models, Trees, Water, and then Particle Effects. Each scene lasted for 60 seconds, and the camera in each scene was moved around to focus on different areas.

## Terrain

Max FPS = 311.964

Min FPS = 76.3

Average FPS = 185.261

Max SPF = 0.0131062

Min SPF = 0.0032053

Average SPF = 0.0062962

## 3D Models

Max FPS = 419.708

Min FPS = 76.244

Average FPS = 244.66

Max SPF = 0.0131158

Min SPF = 0.00238261

Average SPF = 0.00493604

## Foliage

Max FPS = 296.326

Min FPS = 92.7626

Average FPS = 194.858

Max SPF = 0.0107767

Min SPF = 0.00337466

Average SPF = 0.00589328

## Water

Max FPS = 270.42

Min FPS = 103.566

Average FPS = 228.947

Max SPF = 0.00965564

Min SPF = 0.00369796

Average SPF = 0.00458879

## Particle Effect

Max FPS = 272.348

Min FPS = 87.879

Average FPS = 114.764

Max SPF = 0.0113793

Min SPF = 0.00367177

Average SPF = 0.00911996

The results shown here that in terms of performance, the max FPS drops almost 100 FPS once we load in our Tree Foliage. The Max FPS drops again once we load in our water. Then, although the Max FPS does increase slightly once we have our particle effects loaded in, the average FPS has dropped from 229 FPS all the way down to 115 FPS, a drop just under 50%.