CS3S664
3D Scene in DirectX

Khalid Ali (15005070)

MComp Computer Games Development,
University of South Wales

2019-05-10

Contents

List of Figures

1 Introduction

DirectX is a collection of several application programming interfaces (API) and functions used to build graphical applications for Windows-based systems. It is a proprietary framework conceived and maintained by Microsoft, competing with Khronos' OpenGL and Vulkan and Apple's Metal API. It includes DirectDraw, Direct3D, DirectPlay, DirectSound and DirectMusic as subsets APIs (Techopedia, 2019). This report builds a 3D castle tower scene with DirectX 11 to demonstrate various graphical features such as texture mapping, shader features and effects.

1.1 Scene

The scene is based on a terrain built from provided deformation textures. The terrain's detail is supplied animated grass has several distinct trees placed on top. The castle is built from a single keep given with the solution, with an Excalibur-style sword planted in rocks placed in front of one of the stores. A water grid is placed across the terrain to act as a moat, with a single bridge placed between the castle's grounds and the rest of the map. The sword and the rocks are a royalty-free models from TurboSquid by elvair (2018) and ice kazim (2016) respectively.

2 Texture filtering

2.1 Bilinear

Bilinear filtering is the default variant of linear texture filtering that uses cube-mapping to wrap around a given texture to a particular scene object. The per-pixel shader used on scene objects such as the castle use linear samplers. For example, `SamplerState linearSampler : register(s0)`. Figure 1 shows an example of a scene object whose shader uses bilinear filtering.

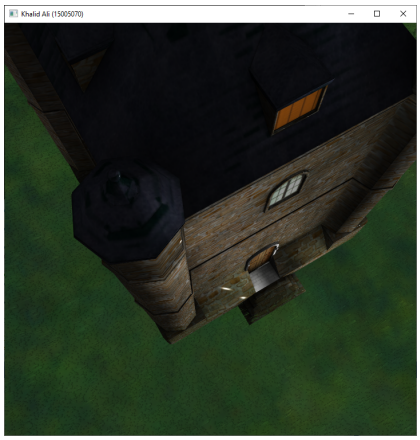

Figure 1: Example of a textured object using bilinear filtering

2.2 Anisotropic

Anisotropic filtering is a non-linear texture filtering technique that generates texture samples based on the angle the surface of an object is viewed relative to the camera. It is viewed as the best but most expensive filtering method commonly used (NVIDIA, 2012). The grass shader is an example of an effect that uses anisotropic texture filtering via sampling like `SamplerState anisotropicSampler : register(s0)`. Figure 2 shows an example of a scene object whose shader uses anisotropic filtering.

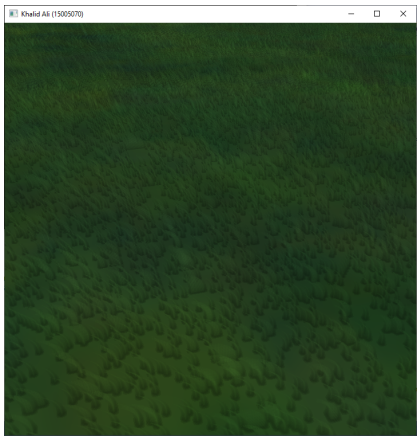

Figure 2: Example of a textured object using anisotropic filtering

3 Normal mapping

The scene's water grid was the most prominent example of a textured object that uses normal mapping. Four levels of mapping were tested in the ocean pixel shader by altering the value for `Nt`.

3.1 Absent

Normal mapping could be turned off by setting `Nt` to `float3(0, 0, 1)`. The result is shown in the **top-left** of Figure 3 and achieved an average FPS of **650**.

3.2 Coarse

A coarse level of normal mapping could be achieved by setting `Nt` to `t0.xyz`, where `t0` is equal to `gNormMap.Sample(gNormalLinearSam, IN.bumpUV0)* 2.0 - 1.0`. The result is shown in the **top-right** of Figure 3 and achieved an average FPS of **641**.

3.3 Coarse + Medium

A coarse and medium level of normal mapping could be achieved by setting `Nt` to `t0.xyz + t1.xyz`, where `t1` is equal to `gNormMap.Sample(gNormalLinearSam, IN.bumpUV1)* 2.0 - 1.0`. The result is shown in the **bottom-left** of Figure 3 and achieved an average FPS of **672**.

3.4 Course + Medium + Fine

A coarse, medium and fine level of normal mapping could be achieved by setting `Nt` to `t0.xyz + t1.xyz + t2.xyz`, where `t2` is equal to `gNormMap.Sample(gNormalLinearSam, IN.bumpUV2)* 2.0 - 1.0`. The result is shown in the **bottom-right** of Figure 3 and achieved an average FPS of **484**.

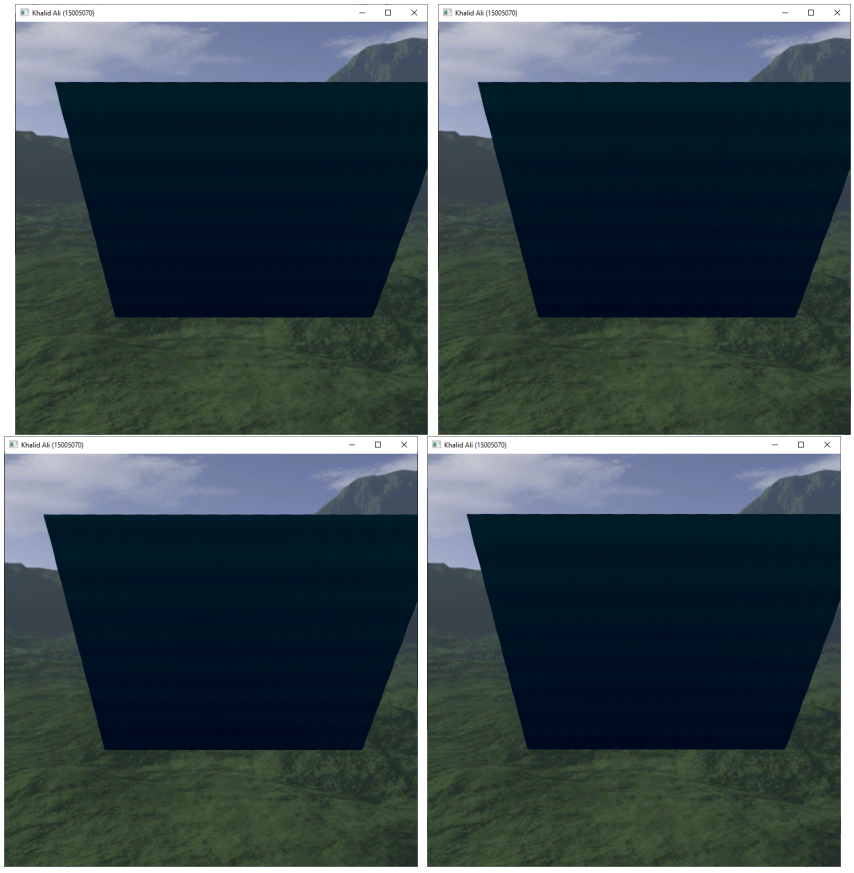

Figure 3: Visual comparison of water normal mapping levels

4 Effects

4.1 Skybox

The scene's skybox is an environmental cubemapping of the texture **grassenvmap1024.dds** around a `Box` object. The rest of the scene is positioned inside.

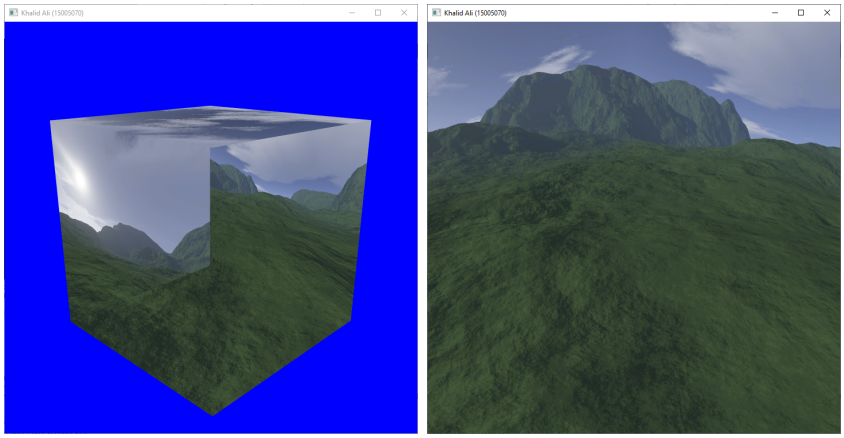

Figure 4: Outside versus inside the skybox cube

4.2 Terrain & foliage

4.2.1 Grass

The grass effect is achieved by using the grass vertex and pixel shaders to generate a tile-based texture of animated fur. Inside the grass vertex shader (**grass_vs.hlsl**), the scale factor can be altered to change how big all the grass blades are. The wind animation of the blades are also computed at the end of this shader. Inside the grass pixel shader (**grass_ps.hlsl**), the grass' tile repeat can be modified to alter the density of grass within the terrain. The effect uses alpha blending. In **Scene.h**, the quality of the grass render can be altered by changing `numGrassPasses`.

4.2.2 Tree

A single tree object is rendered, which uses the per-pixel lighting effect to wrap the solution-given texture onto the also given tree model. The pixel shader (**tree_ps.hlsl**) uses a **phong** reflection calculation to calculate the colour of the object whilst factoring in light direction and eye direction.

4.2.3 Terrain

The terrain distinctively does not have an effect, but is instead an environmental mapping for the grass tiles onto a surface modified by a given normal and height map.

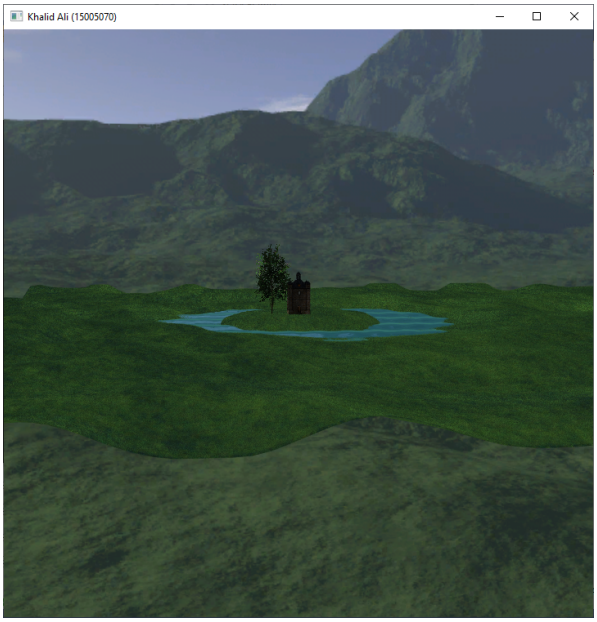

Figure 5: The complete terrain

4.3 Water

The water effect is achieved by spreading a given waves model across a `Grid` object. The ocean shaders' reflection model is Fresnel-based, which is used to calculate how strong a reflection is based on eye angle. Inside the ocean pixel shader (**ocean_ps.hlsl**), the Fresnel's bias can be altered to increase the strength of the reflection (Scratchapixel, 2019). As shown in Figure 6, the bias is a critical variable in the effect. The top-left image is a bias of 0, top-right is 0.25, bottom-left is 0.5, and bottom-right is 1.

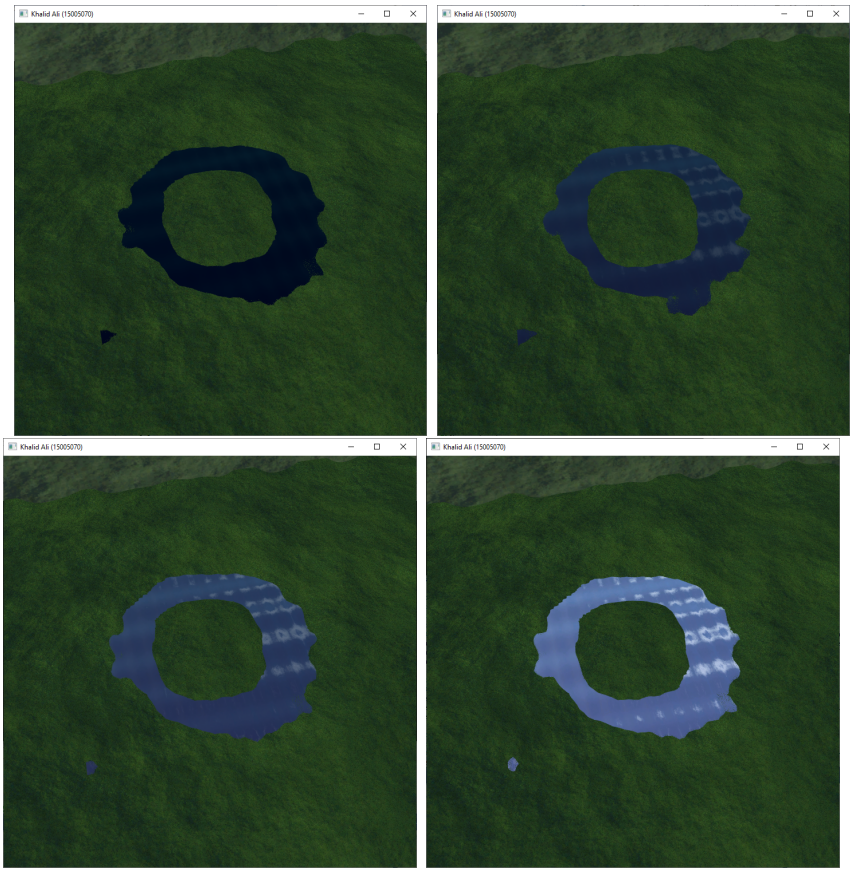

Figure 6: Visual comparison of different water Fresnel bias

4.4 Flares

The lens flare effect is achieved by placing six flares in a line from light source to camera, and is only visible when looking towards its origin. The amount of flares could potentially be altered by changing `numFlares` in **Scene.h**.

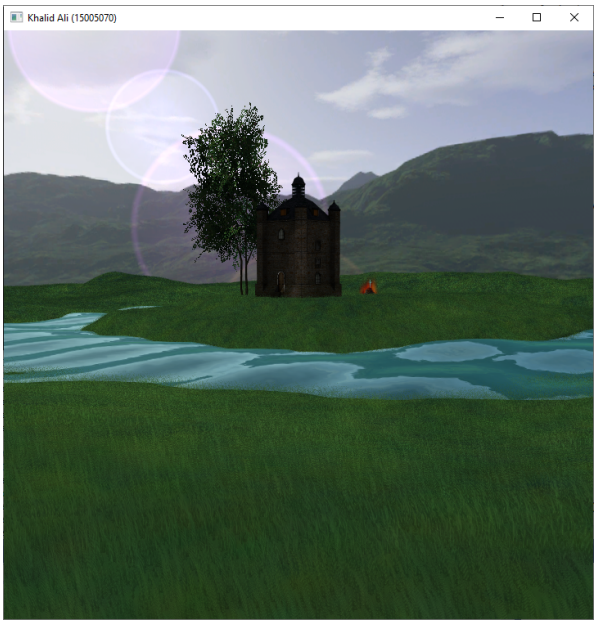

Figure 7: Scene's flares

4.5 Particle systems

4.6 Glow

The glow effect uses a combination of the per-pixel lighting vertex shader (**per_pixel_lighting_vs.hlsl**) and the emissive pixel shader (**emissive_ps.hlsl**). Additionally, the `BlurUtility` class is used to apply a blur to the render output. Unfortunately, the final effect is rendered on top of all other scene objects.

Figure 8: Scene's glow and fire particle system effect

5 References

elvair (2018). *Sword model*. URL: https://www.turbosquid.com/FullPreview/Index.cfm/ID/1283673 (visited on 05/10/2019).

ice kazim (2016). *rock 06*. URL: https://www.turbosquid.com/3d-models/free-obj-mode-rock/1016934 (visited on 05/10/2019).

NVIDIA (2012). *Antialiasing and Anisotropic Filtering*. URL: https://www.geforce.com/whats-new/guides/aa-af-guide#1 (visited on 05/10/2019).

Scratchapixel (2019). *Introduction to Shading (Reflection, Refraction and Fresnel)*. URL: https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel (visited on 05/10/2019).

Techopedia (2019). *What is DirectX?* URL: https://www.techopedia.com/definition/4971/directx (visited on 05/10/2019).