**Technische Universität Wien**

**Realtime Rendering**

**Submission 2**

**Group 7**

Vasco Magalhães Pereira - 11833146

**Brief Description**

This project involved the development of different real-time rendering effects. To achieve this an engine had to be developed first. The engine's job was to render any meshes into the screen using the developed shaders, but also handle transformations to the objects for manipulating the scene during runtime. The objects in the scene are also separated into static and dynamic objects, and this was already done early in development because shadow maps were already planned to be one of the wanted effects.

After the engine supported these basic functionalities the effects were ordered by priority according to the feedback from the submission 0. The effects that ended up on this final submission were the dynamic directional shadow maps, static omnidirectional shadow maps, dynamic environment mapping that was used for the reflections and refractions. The refractions are affected by chromatic aberration and, using Fresnel, when the normals on the surface of the object start to become increasingly more parallel to the view direction, the reflection becomes stronger and the refraction less visible.

After the effects were implemented the camera movement and, also, other objects movement were defined. To more easily determine the points on the path of the camera a simple behavior was implemented which printed the position and rotation of an object using the space key. Having the wanted points and orientation, the camera is then moved by linearly interpolating between the last keyframe and the next one over time. The same is done to the reflecting and refracting spheres to showcase dynamic environment mapping.

In the end, all it took was to check if the demo was still running at 60fps at 1920x1080 on the VisLab computers. Since it was, a few parameters were adjusted to improve the look of the scene while keeping everything within the requirements.

**Additional Libraries**

Some external libraries were used for the development of this application:

* Assimp (<http://www.assimp.org/>) is an open source asset importer library that was used to parse the external model files.
* OpenGL mathematics (<https://glm.g-truc.net/0.9.9/index.html>) is an open source C++ mathematics library based on the OpenGL shading language specification.
* nlohmann/json (<https://github.com/nlohmann/json>) is a library that facilitates working with json in C++. This library was proposed to me by a student from previous years because I needed a way to load and store the scene into a file. Unfortunately, this part of the project is not fully developed, and it still does not write to and read from a file.

**Effects Implemented**

The first step was trying to render a simple triangle on the screen using vertex buffers. Having that working an optimization was made to use indexing for vertex identification instead of repeating the same vertex multiple times. Support for other parameters, like texture coordinates, was also added.

After that, the **Blinn-Phong model**[1] was used to calculate lighting in the scene. The directional light was the first to be implemented, but support for point lights and spotlights was also added.

Using an external library, it was easy to support models of different formats and very little was changed to the existing code to handle the information coming in from the models.

The next step was to implement **directional shadow maps**. A camera is positioned at a given point (LightPoint) and, using an orthographic projection, the scene is rendered into a depth texture. On the rendering shader (default shader), the depth buffer is used to calculate if a fragment’s depth to the LightPoint is less of higher than the one on the directional shadow map[2]. 2 shadow maps are used for directional lights, one for static objects and another for dynamic objects. An optimization to reduce the number of textures consulted for each fragment would be to, before doing the rendering pass, join both textures into one using shaders and a quad.

Then the **omnidirectional shadow maps** were added using the geometry shader to take advantage of only requiring one single pass[3]. The same process as the one above is used to calculate whether a fragment is behind an object that is blocking the light, instead, it’s using a cubemap instead of a 2D texture. Omnidirectional shadow maps are only calculated one time at the start of the program and only works for static objects. This is done because, even using a geometry shader instead of 6 passes, this process is very expensive on the GPU. Some other optimizations like, only updating the shadow map when close enough to the light, or not updating each face per frame.

The shadow mapping artifact related to **incorrect self-shadowing** was solved by rendering the object using back-face culling into the shadow map. At first, the bias was used to try and solve this issue, but specular highlights, when the angle between the light direction and the camera direction to the fragment were reaching 180º, still appeared the artifact. This led to some investigation and in the class slides[3] it was found a second, the one the end up on the final application.

**Environment mapping** was the last effect to be implemented. The same process of rendering the 6 faces of a cubemap in one pass is used[4]. The refractions use **chromatic aberration** for a more realistic effect [5]. And the **Fresnel effect** is used to determine whether it should **refract** or **reflect** the light (or both) [6]. The resolution of the environment map changes according to the distance to the camera, and it is only updated if at a certain angle from the view direction.

**Models**

Skybox:

* <http://www.custommapmakers.org/skyboxes.php>

Helicopter:

* <https://free3d.com/3d-model/uh60-helicopter-47194.html>

Trees:

* <https://free3d.com/3d-model/tree-74556.html>
* <https://free3d.com/3d-model/tree02-35663.html>

Mountain:

* <https://free3d.com/3d-model/everest-mountain-930871.html>

**Play Modes and Controls**

**Cinematic Mode** – the camera moves by itself on a pre-defined path that attempts to show all implemented effects.

**Free Roam** – the user is free to manipulate the camera using the WASD keys and a mouse.

The escape key is used to exit the application, and the numbers from 1 to 9 turn on and off the point lights. Since the scene only has two point lights, only the 1 and 2 keys do something.

**Graphics Card**

The project was tested on a VisLab computer with a GTX 1060 6GB.

**References**

[1] Blinn-Phong Model

<https://learnopengl.com/Advanced-Lighting/Advanced-Lighting>

[2] Directional Shadow Maps

<http://cseweb.ucsd.edu/~ravir/274/15/papers/p270-williams.pdf>

[3] Incorrect Self-shadowing solution - <https://tuwel.tuwien.ac.at/mod/resource/view.php?id=571162>

[4] Single pass omni-directional shadow mapping - <https://users.cg.tuwien.ac.at/husky/RTR/OmnidirShadows-whyCaps.pdf>

[5] Reflection, Refractions and Fresnel

<https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel>

[6] Chromatic Aberration

<http://www.paulsprojects.net/opengl/refract/refract.html>

<http://developer.download.nvidia.com/assets/gamedev/docs/ChromaticAberration_CEDEC_E.pdf>