CGP600 Advanced Games Programming

# Overview

This report will detail the process of how I created a 3D game in Microsoft DirectX 11 and C++, Following the design documentation found in the previous assignment.

# Technology Implemented

## Lighting

### Ambient:

Ambient lighting is a lighting technique in which every object in a scene is lit up equally and evenly across of surfaces, with the same colour and magnitude. In the current build, it has been implemented via the vertex shader; also known as Gouraud lighting. There were plans to implement ambient lighting via the pixel shader, which is known as the Phong reflection model.

### Directional Lighting

Also known as diffuse lighting, this is a lighting technique that places a light source in the world space with a direction it is facing. It “produces” light in the direction that is then used to calculate how much the surface is lit by the light. Taking into account the angle of the surface relative to the light direction, as well as how intense the light source is. This has been implemented via the vertex shader, with future plans to implement it via the pixel shader.

## Model Loading

The model loading class allows the automated creation of vertex and pixel shaders based on a filepath to a .obj file in the assets folder. In addition to this it also allows for uniform methods to manipulate the object. It also contains the sphere collision methods.

## Bounding Sphere Collision

The collision check has several parts. The first part is at the creation of the object, we work out the centre point of the object, as well as the radius of the object. To work out the centre we find the smallest and largest values of each axis’ vertices to get the Minimum value of X and the Maximum value of X; as well as the same for the Y and Z axis. Then we half each of these values to find the midpoint of each, thus finding the co-ordinates of the exact centre point of the object. To find the radius we check the distance to the centre point from each vertex using 3D Pythagoras. From here we then have everything we need to check if two spheres are colliding.

The following formula is used to check the distance between two spheres based off of their centres.

distance = (s1.x -s2.x)² + (s1.y -s2.y)²+ (s1.z -s2.z)²

Then if the distance is less than the two radii combined, a collision as occurred.

# Problems

There have been several problems throughout the project. The biggest of which are the deprecation of various DirectX 11 libraries, this led to a lot of tutorials online being borderline useless to use as they haven’t been updated to take into account API changes. Secondly Visual Studio doesn’t error check HLSL files or offer any autofill functionality, this has led to several periods of long debugging sessions to find an error that was caused by a tiny typo in a HLSL file.

# Solutions

The most useful solution to a lot of the issues I had would be a brand-new set of tutorials that take into account all of the changes to DirectX 11 that have happened over the years. This would help smooth out the steep learning curve. For the HLSL issue there is an addon available that adds in the functionality, but it seems like a large oversight and in my opinion, should be added in the base install of Visual Studio.

# Conclusion

In conclusion, this project has been moderately successful in regards to my learning of the DirectX 11 API. It has mostly failed as a game but as a proof of concept/technical demonstration it has a decent amount of value. Several techniques have been implemented successfully and has shown what can be done with more time and learning.