Edinburgh Napier UNIVERSITY

Coursework Report

Timothy S. Sanderson
40219124@live.napier.ac.uk
Edinburgh Napier University - Computer Graphics (SET08116)

Abstract

This is a project which demonstrates the implementation of basic lighting, shadowing, and texturing effects, in a 3D graphical scene.

Keywords – Graphics, Hierarchy, Phong, Shadows, OpenGL, Mirror, Height Map, Post Processing

1 Introduction

This project demonstrates graphical concepts through the use of a 3D environment, rendered using OpenGL. In this scene the main focus is a structure made of numerous moving spheres, that is reflected in a nearby mirror. The spheres represent a hierarchical relation, as well as providing a good demonstration of the reactivity of the implemented graphical effects. Due to their rapid movement it is easy to see how efficiently the highlights are tracking across their surfaces, and how the shadows are being cast behind them.

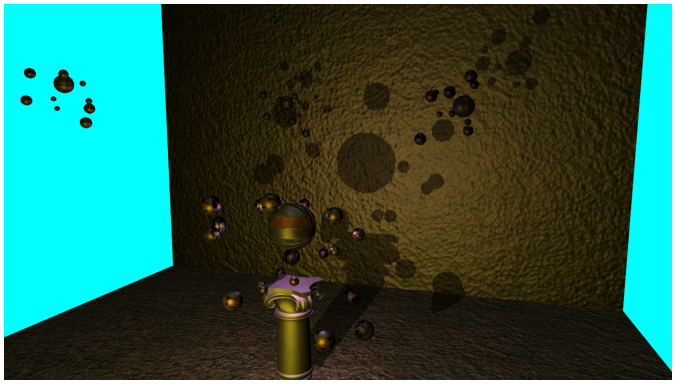

Figure 1: **Old scene overview** - A good showcase for basic techniques in the scene

2 Implementation

The following are the concepts implemented in the scene to give its current appearance.

2.1 Textures

All objects in this scene are textured. By applying a texture to an object it can be given aesthetic detail that makes it more pleasing to look at, and easier to discern from other objects.

A more complicated use of textures is the normal map. A normal map is used by a shader to manipulate how light shines on an otherwise flat surface. In Figure 1 the back wall is a completely flat plane, but thanks to normal mapping it looks like a roughly textured wall.

The final version of the project uses a floor created with a height map (Figure 2). A height map is a texture used to create geometry based on colour values. The closer to white the texture is, the higher the y-position of the corresponding vertex. Although much more dramatic terrain can be generated from a height map, simply creating a gentle hill is beneficial to this scene for two reasons. The first reason, it elevates the centre of the scene, bringing focus onto the important elements of the environment. Secondly and more simply, it prevents the ground from being a large flat plane, which is a visually unappealing thing to look at.

Figure 2: **Height mapped terrain** - A plane which has had its vertices vertically adjusted by a height map

Textures are an incredibly important thing when it comes to post-processing. When a scene is rendered it outputs to a frame buffer. By default a scene is rendered to the screen's frame buffer. However, the output can be changed to a frame buffer object, and from here the rendered scene can be saved as a texture. Once a render has become a texture, that texture can be used for post-processing.

2.2 Lighting

Objects within this scene implement Phong illumination. This lighting model implements the use of ambient, diffuse, and specular lighting.

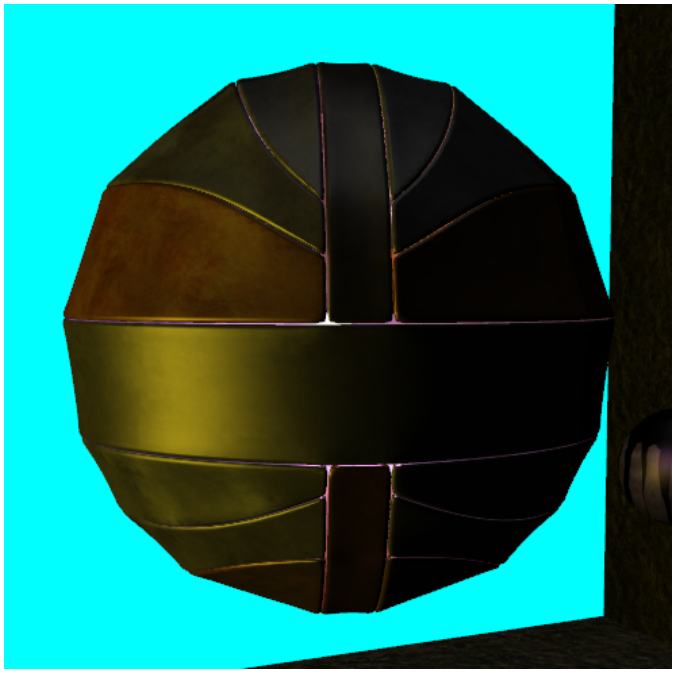

Figure 3: **Lighting** - Lighting effects demonstrated on an orb

On the right of Figure 3 can be seen the ambient light. It's set to be very low, but some details can still be seen. This lighting type is most simple of all. Ambient light is applied uniformly, and without regard to light or view direction.

The areas where details can easily be seen are the ones demonstrating diffuse lighting. This is the light which is directly hitting, and illuminating a vertex. Diffuse lighting changes its position on an object based on the direction of the light source. As such a vertex on a spinning object will not always be illuminated, and the maths needs to account for this.

Bright yellow highlights on the left are an example of a specular reflection from the yellow light source. Specular highlights are special because their location changes based on viewing angle, the position of the vertex, and the location of the light.

To test the relation of a vertex to light, the vertex's normal is used. When combined with the direction to the light in a dot product this gives information about the angle between the two and, from there, the degree of lighting can be calculated.

2.3 Shadows

Shadows (Figure 4) are a relatively complicated and mathematically intense technique. To know if something is in shadow it needs to be known if there is an object between it and the light. In order to find this out the scene first has to be viewed from the position of the light. This data is saved to a texture which allows the red value to be used to determine how far away that vertex was from the camera. Due to the nature of colours holding whole values to up to 255, this means there is a very limited capacity for detail. In this scene the shadows are calculated from objects up to 300 arbitrary units away, so that shadows are far reaching, but do not lose too much precision. Then, when rendering the scene to have shadows implemented, the vertex needs to be operated on with matrices from both the camera's view and the light's view, in order to see whether it lies within the cast of a shadow.

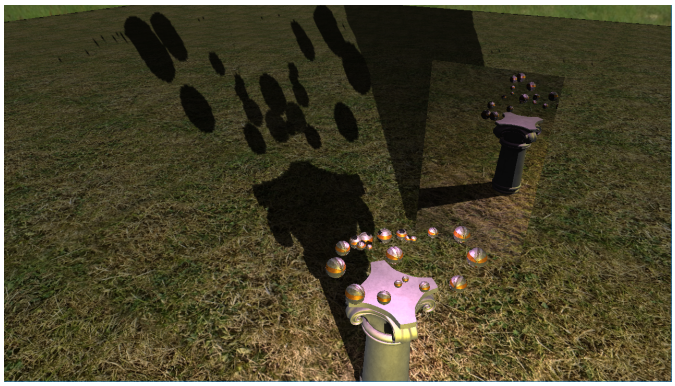

Figure 4: **Shadows** - Shadows being cast across the scene

2.4 Post-processing

Visual effects applied after the rendering of the main scene are known as post-processing. To apply these visual effects the scene is first rendered as a texture, and then this texture is manipulated to give a certain output. Post-processing is a powerful tool allowing pixels to be read from a screen space they didn't originally belong to, among other things. Figure 5 shows off this concept clearly as the pixels in a vertical line are instead actually displaying pixels to either the left, or right, of themselves based on a sin wave.

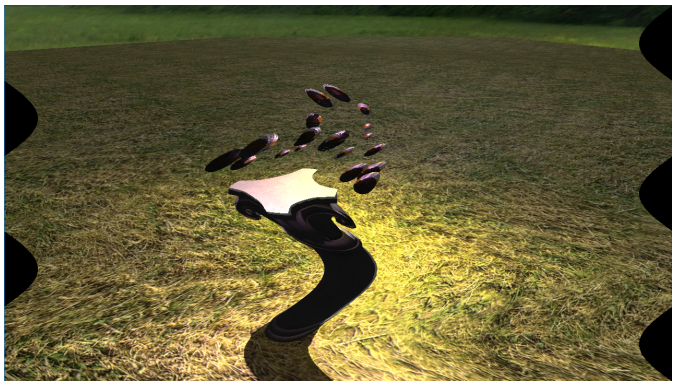

Figure 5: **House of mirrors** - A post processing effect to turn the regular screen output into multiple waves

A more common visual effect is the use of blurring. Variations of blurring can used for things such as depth of field. However, this scene implements a very simple blur, which has a result somewhere between anti-aliasing, and looking out of focus (Figure 6).

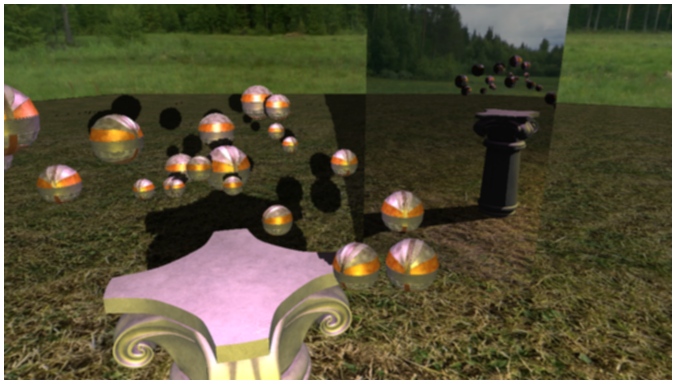

Figure 6: **Out of focus** - This effect blurs the scene

2.5 Mirroring

A fusion between post-processing, and scene rendering is the recipe for the highlight of this scene, mirroring. In this scene the mirror is simply a plane with a texture on it. Using texture coordinates based on screen space instead of the object's position created the appearance of it reflecting a three dimensional space. Getting the texture is what requires most of the work, and processing power.

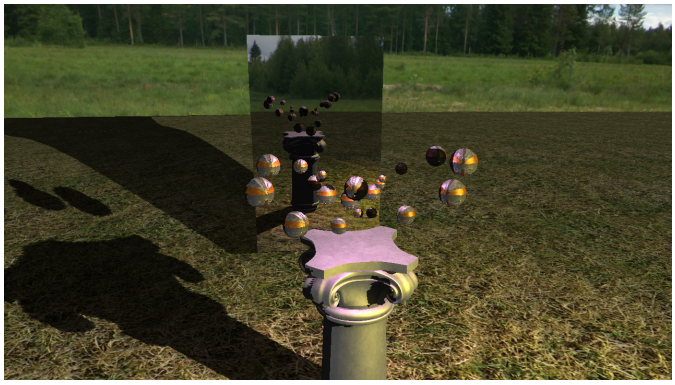

Figure 7: **Reflection** - A mirror reflecting the scene

To get the texture for the mirror required doing a render of the scene before the scene was properly rendered. The difference was that this first scene was rendered from a camera, that existed as a counterpart to the active camera. This "counterpart" existed behind the mirror, mirroring the movement and view of the first camera. Of course this requires two renderings of the scene instead of one, which greatly increases GPU load.

3 Optimisation

These scene has a both good and bad points when it comes to optimisation. There are many areas where an effort was made to streamline processes, however there are many known taxes on processing power as well.

3.1 Positive optimisation

In openGL the "bind" function is a particularly taxing one. As such it is best to avoid it where possible. To accomplish this textures are only bound to the renderer if they are different from the previously rendered mesh. With the structure of orbs featured this is particularly important as the repetitions would be high.

Further optimisations were made with relation to the cameras view and projection matrices. These were multiplied together once at the start of the rendering process, instead of every time an object was rendered. Although individually insignificant it is important to avoid even minor processes when the amount they are done is high.

When calling functions in c++ the typical method is to use pass by value. This process creates a copy of the data to hand to the function, which when processing large maps, can be a problem for efficiency. Consequently many of the functions in this program utilise pointers or references to speed up function calls.

When behind a mirror it's reflection can not be seen. With this in mind the mirrors reflection is only rendered should the camera be positioned in front of it. Due to issues with field of view, the mirror does not stop rendering when you simply look away, but the positional optimisation alone sees significant performance boosts when behind the mirror.

3.2 Negative optimisation

The polygon count of a model is a good way to tell the effect it will have on a GPU. Although the specifics are unknown, the model that was imported for the orbs has a very high poly count and does cause problems on weaker GPU's. Adjusting level of detail would help to improve the performance impacts of the orbs.

Optimising camera matrix multiplication was discussed in the positive section, but it belongs here as well. The matrices are multiplied every render cycle, but it would be even more optimal to calculate them only when the camera has moved.

A better mirror than the one implemented here would be one which didn't render the areas of the scene it didn't need to show. As it stands the mirror camera renders an entire scene and then only a small portion of this is actually used.

4 Future Work

The elements discussed in the negative optimisation section are of course things that need to be worked on. They range from simple for ignoring some matrix multiplication, to very complicated maths for the mirrors. Many other areas of code could benefit from being cleaner too, as many segments of the render function could be outsourced into methods.

The scene could also do with a more cohesive setting/theme, whereas it currently is visually like a tech-demo. Although functional, there is much that could be

improved to give the scene more personality while still demonstrating the same concepts.

Something that would greatly improve the scene from a functionality stand point is a GUI to tell a person how to control it. There are a great many buttons that can be pushed in this scene and a visual reminder would be a big help.

5 Conclusion

Overall, this scene has met the requirements made for it. Although it might lack the personality that would make it more interesting to look at, it is still an effective exercise in graphical effect processing.