**Advanced Shader Report**

**By Sam Knight – P16184152**

**Computer Games Programming**

**De Montfort University**

# Introduction

Shading provides us with a variety of techniques to create realistic visualisations of scenes. These techniques are used by the games and film industry to create stunning visual effects. This report will look at the implementation of 3 advanced shader techniques: Blinn-Phong, Normal mapping and Parallax mapping.

# Shading techniques

## Blinn-Phong shader

Blinn-Phong shading is a technique that provides smooth and accurate looking ambient, diffuse and specular lighting and material shading. This results in the ability to control the colours that an object can absorb/reflect and the colour of the light. Before Blinn-Phong there was Phong and Gouraud shading: Gouraud shading calculates lighting on a per vertex basis and interpolates the in-between colours, however, this results in triangular highlights on low-poly models; Phong shading improves on Gouraud shading by doing the lighting calculations per fragment resulting in smoother lighting. However, its specular highlight still falls short with low exponent/shininess values as you can see a sharp edge to the highlight. Blinn-Phong deals with the problem as makes the highlight look more diffuse at low exponents, resulting in natural looking lighting.

## Normal mapping shader

Normal mapping is a technique used to make flat surfaces look 3D by giving a surface custom normal vectors in the form of a texture (Normal map), therefore light reflects as if the surface had varying height. Normal maps store normal values in the form of RGB values resulting in a texture that looks like a strange combination of red, green and blue colours.

## Parallax mapping shader

Parallax mapping is technique that is used to give a surface the illusion of actual depth, therefore when view from different angles certain parts of a surface could appear to be hidden behind another portion of the surface, even though the surface is in fact flat. Parallax mapping uses a height/depth map that determines the depth at which the actual texture should be rendered, however, this technique isn’t very accurate and relies on estimations. Say we had a brick texture and the edges of those bricks are sloped, to calculate the visible displacement that the texture should have we need to calculate an intersection point between the view direction and the height map value. The problem is that we don’t have enough information to calculate this intersection point accurately. Therefore, a trick is used to a position that close most of the time, we do this by used the height value at the fragment we are currently at and scaling the view direction vector by that height value. This should result in a fragment position that is roughly near the intersection point that we need.

Steep Parallax Mapping

So far, the parallax mapping would result in some strange artefacts when the surface is view at a low angle and if there is a sharp change in depth. To deal with these artefacts we improve upon the parallax mapping shader by sampling the height map in layers until the layer-depth value is greater than the sampled depth. This does improve the accuracy of the shader; however, it does result in the surface looking like it has been sliced into many layers.

Parallax Occlusion Mapping

To fix this problem parallax occlusion mapping interpolates between texture coordinates of the depth layer above an below where we think the view direction collides with the depth map. Therefore, rather than just taking the texture coordinate for the first layer, with a depth greater than its sampled depth, we also take the texture coordinate of the layer that came before it. The results in a much smoother effect (No more visible layers).

# Development

The first step to getting anything work with OpenGL is to create a window, this is done by setting up the GLFW and GLAD libraries and calling the appropriate initialising functions. This provides us with a window to render to by using functions provided by GLFW and GLAD. To render anything, we also need to tell OpenGL how it should render and how to deal with any data we want rendering. This is done using a Shader using OpenGL Shading Language (GLSL). To simplify the preparation and usage of a shader we create a class called ShaderProgram, this handle setting up and compilation of shaders as well as providing some error checking, in case we do anything wrong in the shader code. The ShaderProgram class tells OpenGL which shader the vertex and fragment shaders are, then links them to a shader program and we record the ID of that program so that we can easily pass data to the shaders later. Now that we have all the requirements to start rendering to the window, we also want to be able to easily view anything we render. Therefore, we make a camera class that does all the necessary vector and view matrix calculations for us to traverse a 3D environment.

The first shading technique to implement was lighting, this started out as calculating how an object would absorb and reflect ambient colour e.g. if we had a white light and a red object, then we should see a red object or if we had a blue light and a red object we should see a purple object. This results in a shape that it all one colour and doesn’t look very realistic, we need a way to differentiate between sides facing toward a light source and away. This is called diffuse lighting, the closer surface normal (direction the surface is facing) is to facing the light source, the brighter that surface’s colour will be. This is an improvement but let’s say the object was very shiny, we would expect to see a bright spot when viewing the object from certain angles. This is the specular highlight that acts as a reflection of the light itself and depending on a how shiny the object is will determine how spread-out/sharp the highlight is. These calculates are done in the fragment shader and result in the Phong shading technique. We take this one step further and implement Blinn-Phong shading which deals with the problem of low shiny/exponent values.

The next technique to add is normal mapping. To do this we take a colour texture and a normal map texture. Rather than using one normal for an entire surface and sample the normal map and calculate light accordingly.

The final technique, parallax mapping, builds on normal mapping by also adding a depth map. As explained earlier in the report we now sample the depth map to and physically render the colour texture according to the calculated depths.

# Conclusion

Overall, I think I have demonstrated an understanding of the 3 advanced shader techniques. To improve I would want to try and get these techniques to work on models rather than simple geometry since most games work with artist made models.

# Bibliography

Vries, J. d., 2014. *https://learnopengl.com/.* [Online]   
Available at: https://learnopengl.com/  
[Accessed 18 4 2019].