**IMAT3906 – Advanced Shader Programming Report**

by Joshua Jackson (p16179167)

# Introduction

This project covers the research and implementation of various advanced shading techniques within graphics programming using OpenGL. I will be discussing the techniques in general as well as a more detailed description of the implementation process for each technique as well as a discussion around what benefits each technique brings and *why* modern games might decide to implement them over other alternatives.

These advanced techniques are widely used within modern games to achieve further graphical fidelity without the addition of further vertices to a model, discussions around and examples of such games can be found in the appendices of this document.

The project also makes use of Blinn-Phong lighting which isn’t covered in a specific section as it isn’t an advanced shading technique, however the GLSL code attached with this project clearly demonstrates with comments how that has been implemented.

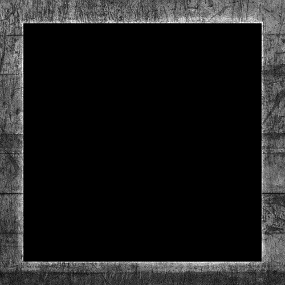
#### Specular and Diffuse Mapping

Specular and Diffuse mapping allow us to set the respective colour for each individual fragment/pixel of an object by utilizing an individual texture for each proponent. Each model will have a diffuse map, often the same as the base/default texture, this determines the diffuse colour of the object and then it may have a specular texture file that determines how bright the specular component of the lighting component is on specific areas of the diffuse texture.

An example would be for a wooden crate. The following image shows the diffuse texture/map:



And then the following image shows the specular texture/map:

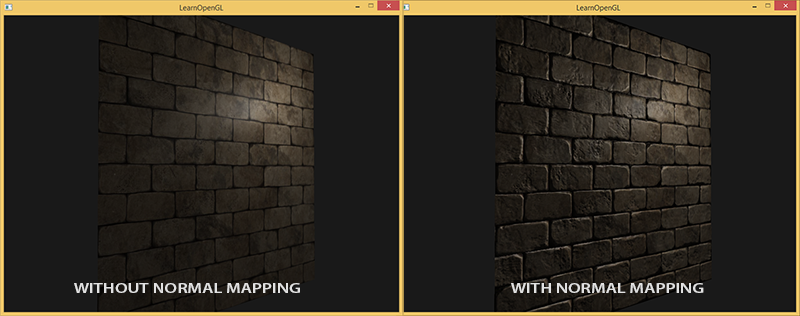


This tells us that the outsides of crate (ie. the metal bits) should have a brighter specular component which make sense as the wooden material wouldn’t be as shiny as metal when a light is pointed at it.

#### Normal Mapping

Normal mapping allows us to specify a texture that can provide more detail to a model’s surface without the need for extra vertices. It does this by using a texture to determine which areas of the surface reflect more light.

An example of this would be:

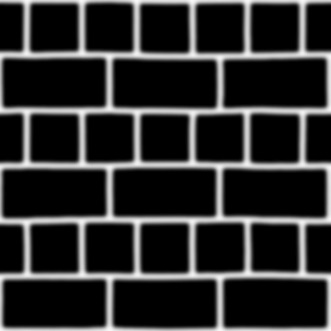


[From learnopengl.com]

Where the model looks more detailed without the use of extra vertices.

#### Parallax/Displacement Mapping

Parallax mapping, sometimes know as displacement/depth mapping, is a technique similar to normal mapping that boosts a textured surface’s detail by giving it a sense of depth without the need for extra vertices. It does this by using a texture that looks like the following:



That will be used to displace the texture coordinates giving the illusion of depth. This technique combined with normal mapping can lead to some very realistic results.

# Resources

Most of the literary content used within this project/report has come from Joey de Vries at <https://learnopengl.com/> but also from <http://ogldev.atspace.co.uk/>

I have made use of many models from <https://www.turbosquid.com/> and <https://www.cgtrader.com/> as they come with both an obj file as well as each type of texture file needed to render the advanced techniques.

I originally used my own model loader but hit many stumbling blocks that prevented me from focusing on the shading techniques so I opted to use Assimp to load in my models and the stb\_image library to load textures. (<http://www.assimp.org/>) – (<https://github.com/nothings/stb>)

Specific links to resources can be found in the appendices.

# Process

In this section I will be outlining the implementation process involved with each advanced shading technique, covering the process of taking data from the CPU to the GPU and how it is then used on the GPU via the shader to create the desired effect[s].

#### Specular and Diffuse Mapping

The process of getting specular and diffuse mapping up and running is fairly simple. We load the texture file[s] and then bind it:

//Bind diffuse texture map

glActiveTexture(GL\_TEXTURE0);

glBindTexture(GL\_TEXTURE\_2D, m\_diffuseTextureDataID);

if (m\_specularTextureDataID != -1) {

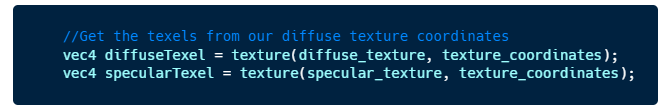
//Bind specular texture map

glActiveTexture(GL\_TEXTURE1);

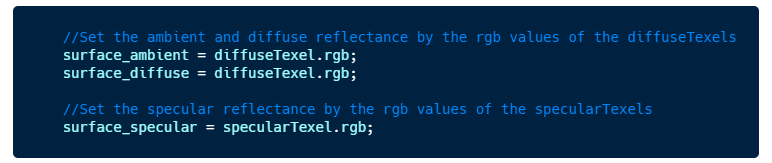
glBindTexture(GL\_TEXTURE\_2D, m\_specularTextureDataID);

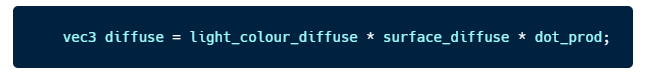
}

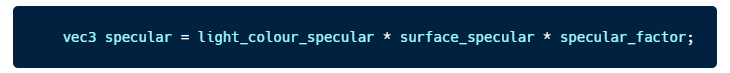
We can then, inside of our fragment shader, grab the texels using the specific sampler2D texture we bound above and the coordinates of the texture (grabbed during model loading):



Then we get our surface\_diffuse and surface\_specular components from the texels grabbed from the texture file to be used in our lighting calculations:



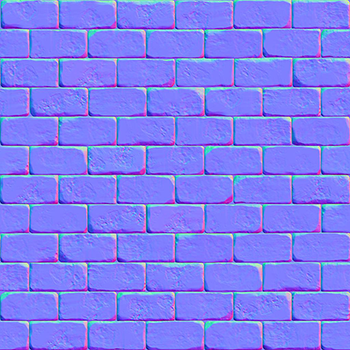




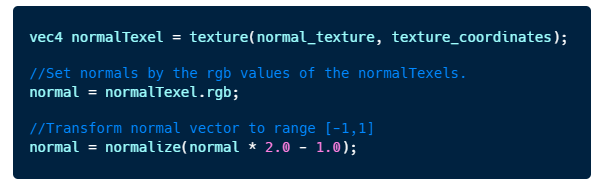
#### Normal Mapping

Normal mapping can be achieved in much the same way as diffuse/specular mapping whereby we pull values from a specific texture file but instead of pulling RGB values and using them directly, we will instead pull RGB values out of the texture and use them to determine the normal of each fragment. This then will allow the light to emulate the detail on the surface instead of the surface itself.

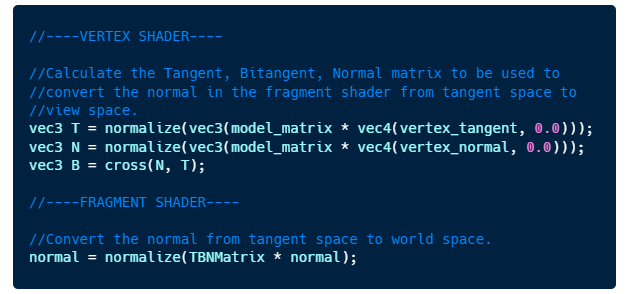
We start by sending our normal map to the shader which looks something like the following:



Inside of the fragment shader we can then get the rgb values from the texture file and then set the normal to be the normalized version of the rgb value like so:



We then need to ensure that the normal are always facing the correct direction, we do this by converting them into world space from tangent space by using a TBN matrix and then we can leave the lighting calculations as they are:

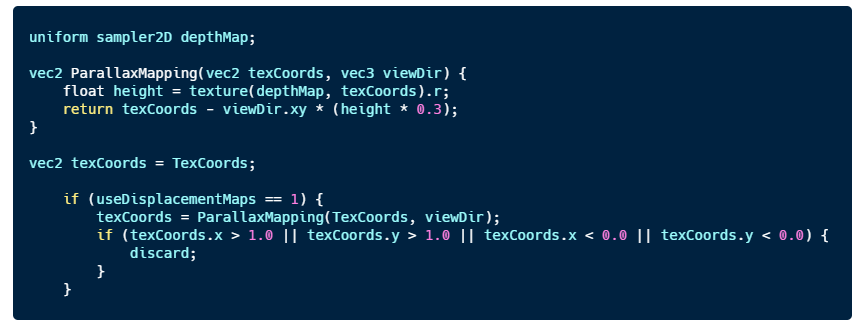


An alternative way of doing this that I am using within my latest shader is to convert the light direction and view direction into tangent space. As long as the normals are in the same space as when doing the lighting calculations, it will work as expected.

#### Parallax/Displacement Mapping

Although my implementation isn’t fully complete, it still demonstrates how parallax mapping works at the shader level.

First we start by loading a displacement map texture much like we have with the other textures and then in our fragment shader we pull in that displacement texture much like with the other textures, we then adjust the texture coordinates that the other textures utilize when loading their rgb values. We do this by creating a function that takes the original texture coordinates and the view direction and then slightly shifts them by the displacement texture multiplied by a height scale.



# Reflection

This project was a very good learning experience especially leading on from the introduction to graphics and 3d modelling module. I found displacement mapping to be the hardest advanced technique to try and implement as I had many problems along the way, the end result is therefore missing fully featured displacement mapping, however it does demonstrate the process along the way and how a potential result would look like (albeit not the desired one).

If I could tackle the project again I would start off by using Assimp as the default model loader as I lost a lot of time with loading models. I would also take a closer look at other video games and try and create models as simple shapes that replicate their models and where the textures would be the main point of detail on any model.

# Appendix

##### Bibliography

1. LearnOpenGL from Joey de Vries - <https://learnopengl.com/>
2. House Model from Turbo Squid - <https://www.turbosquid.com/3d-models/free-scotish-house-3d-model/464428>
3. Rock 1 Model - <https://www.cgtrader.com/free-3d-models/plant/other/rock-01-98f8a3df-4cba-4bb7-8452-f1c36bc35fa7>
4. Rock 2 Model - <https://www.cgtrader.com/free-3d-models/exterior/landscape/8k-mountain-rock-scan-a>
5. Archway Model - <https://www.cgtrader.com/free-3d-models/architectural/other/archway-3>
6. JPG to BMP converter for textures - <https://image.online-convert.com/convert-to-bmp>
7. MeshLab to triangulate obj models before the use of assimp - <http://www.meshlab.net/>

##### Games making use of some of the advanced techniques

Verdun make good use of some of the advanced techniques such as Normal/Bump Mapping and as well as advanced lighting to achieve good performance without increasing the number of vertices on models:



Another good example would be the Uncharted series that makes extensive use of the techniques, especially on lower end console such as the PS3 which can only really achieve the level of detail through shading and not higher poly models:

