The base idea was to implement normal mapping on a previously created model, then add in another technique that would merge well with it. After debating between bloom, geometry shaders and parallax mapping, I decided to implement parallax mapping on another shape to seamlessly merge it with the existing scene by using it as a floor or wall. Implementing either bloom or geometry shaders would have required a more complicated model to be used rather then just the quad for the floor, as well as it would have been difficult to seamlessly integrate them into the current scene. The result is a previously created demon cat model with normal mapping implemented to show cracks on the wings and fur on the body, standing in front of a brick wall with a parallax shader applied to it to create the illusion of depth between the bricks.

Normal mapping is a technique used to increase the amount of detail rendered on a flat plane. Normally, the light gets reflected along the per-surface normal which is the same for each fragment. This creates a flat looking texture with little detail. Normal mapping allows the use of per-fragment normals, meaning that light is reflected in different directions per fragment, instead of the one direction based on the surface. The X, Y and Z axis of a normal vector are stored as RGB factors of a 2D texture. Red represents X, green Y, and blue Z. This results in most of a normal map being tinged blue for the normals are pointing outwards along the positive Z-axis. The RGB data can then be translated back to XYZ coordinates to be used by the shader to calculate normals.

Parallax mapping is like normal mapping in the sense it increases a textures detail without altering the mesh, in this case it conveys a sense of depth. It is often combined with normal mapping to generate realistic geometry. It is similar to displacement mapping where it offsets the vertices based on a height map, however it doesn’t require extra vertex data but is instead done as an illusion. Parallax mapping is done using a displacement map which is the inverse to a height map, this is for it is easier to fake depth then height on a surface. There is a chance that at the edges of the plane the displaced texture coordinates can be outside the range, resulting in an unrealistic looking texture. However, this can easily be fixed by discarding the fragments which lie outside the texture coordinate range.

The main resource I used to create the shaders used in this project was the tutorials on LearnOpenGL[1]. I also used their free to use diffuse[2], normal[3] and displacement[4] textures to create the brick wall. The demon cat model[5] was one I had previously created and the normal map it uses[6] is a standard fur map which I edited to fit the models UVs. The model itself was created and edited in 3DS Max and Maya, which is where it was exported from to have the .mtl file linked. This file contains all detail about the textures and materials applied to the object, which the shader programme accesses to apply the correct materials to the model. The normal map was edited and created using GIMP. The tutorials provided on LearnOpenGL go through the different aspects of shaders in detail, as well as the basics of model loading and lighting. These where used to aid the development of the project and understanding the calculations needed to create the shaders. The project and shader .vert and .frag files where created and edited in Visual Studio.

In terms of the model, it is loaded in using its .mtl file, the meshes are identified and their data stored. Information such as vertex position, texture coordinates, normal vector and the tangent vector are obtained from the vertices while the number of indices is obtained from the number of faces. The texture data is also obtained from the material and stored to be used by the shader. This data is used to draw the model, and some is passed onto the shader files to apply the correct textures and mapping.

For normal mapping, the shader uses the stored normal map from the model to generate the per-fragment normals. This is done in the fragment shader[7] file along with the light attributes. The vertex file[8] converts these coordinates from world space to tangent space which allows the normal vectors to be in the direction relevant to the model transformation.

The wall is created using a GL quad. The position of the vertices, texture coordinates and the normal vector are statically defined. The set up and binding of the quad is done in the main file using a function called setupQuadVAO(). This generates a standard quad by rendering two triangles with the statically defined variables and binds it using OpenGL, creating the plane that the wall is drawn on.

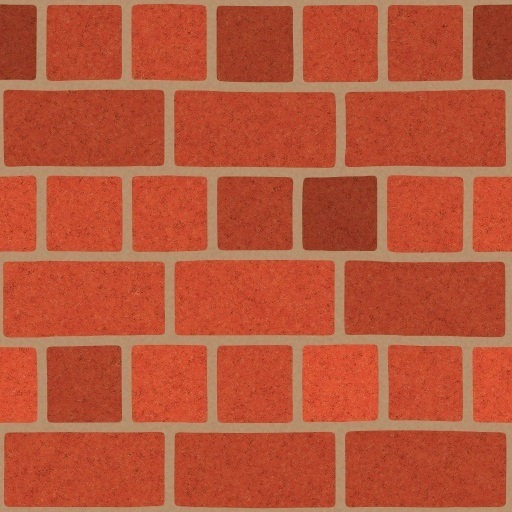
The fragment shader[9] for parallax gets passed the images for the diffuse, normal and height/displacement maps and translates them onto the quad. It applies the diffuse and normal maps as standard, and when parallax mapping is turned on, creates the illusion of depth added to the bricks and discards the out of bounds coordinates. It also sets up the light attributes to be used by the shader. The vertex shader[10] for parallax is similar to the one used for normal for it also translates the coordinates from the world coordinate system to the tangent coordinate system.

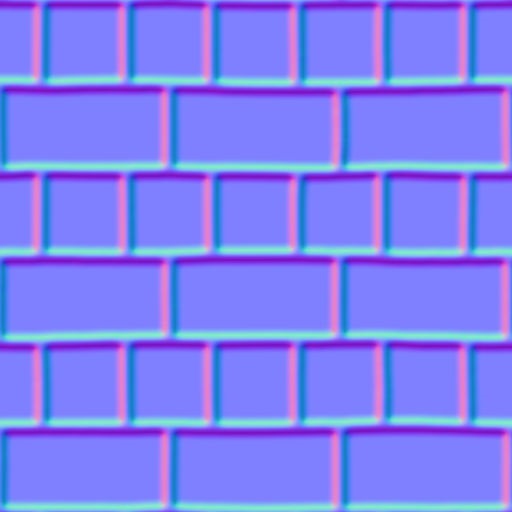
Overall, the project turned out well, however it was not everything I had wanted to achieve. The techniques are well implemented and are demonstrated effectively on the chosen models, you can clearly see the effect normal mapping has on the cat’s wings and parallax mapping has on the wall[11]. Despite this, there is still more that I could have added to the project. I had originally wanted to implement either bloom or geometry shaders but had ultimately decided against this. If I had more time on the project, I would have loved to have implemented an exploding glass using geometry shaders. This idea stems from the original source of the cat model; it was created to be used in an animation in which the cat knocks a glass off a table. Although animation was not the goal of this project, the idea still could have been transported over by having a glass explode in front of the cat. Bloom also could have been implemented by adding in coloured shapes around the scene.

Alongside this, the code heavily relies on the tutorials on LearnOpenGL. In hindsight, the implementation of the code could have been done much better with proper separation of classes into headers and .cpp’s. it also could have deviated more from the tutorials and explored other implementations. The current implementation is fine for a small project however it would not upscale well to a larger one. This is another area of the project I would have liked to improve on with more time as well as the variety and complexity of techniques added.

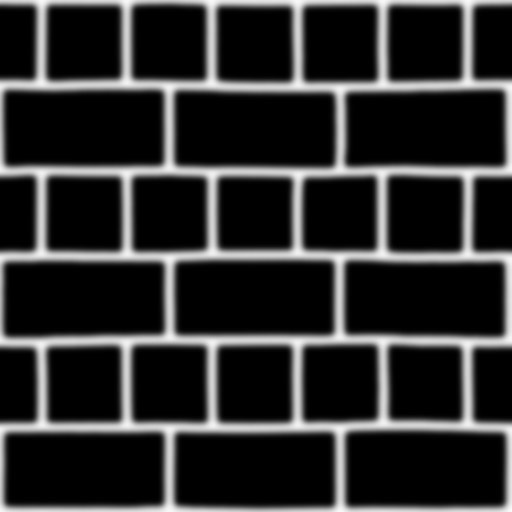
References

[1] <https://learnopengl.com/Advanced-Lighting/Parallax-Mapping>

[2]

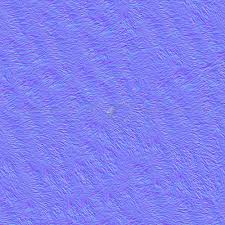


[3]



[4]

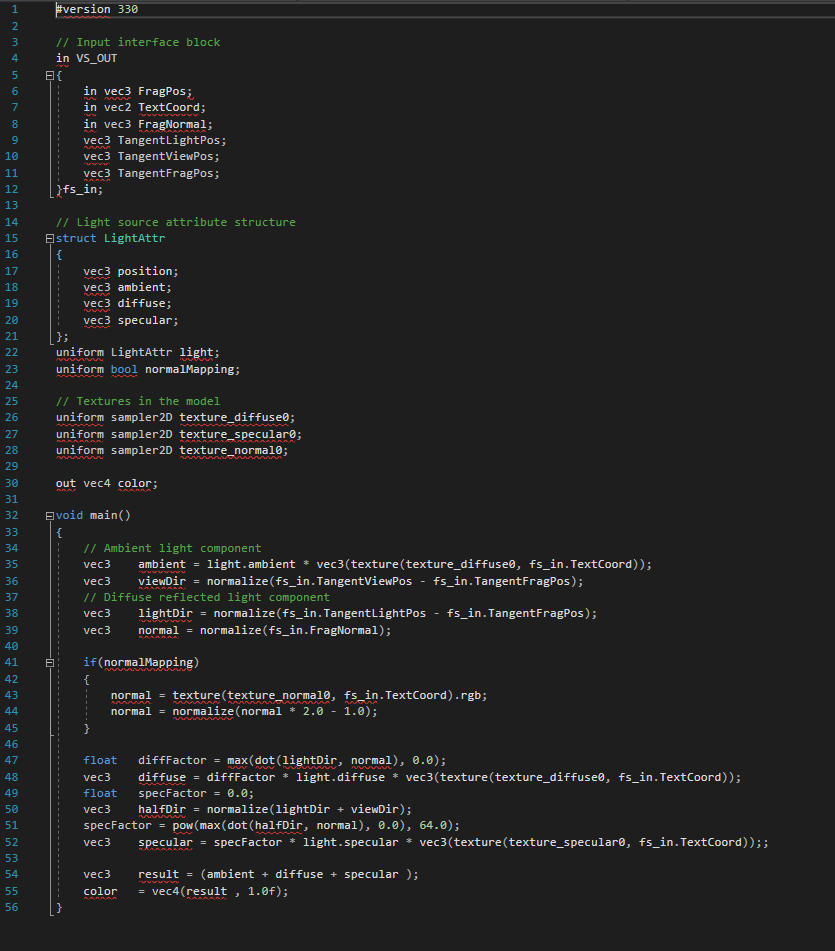
[5]



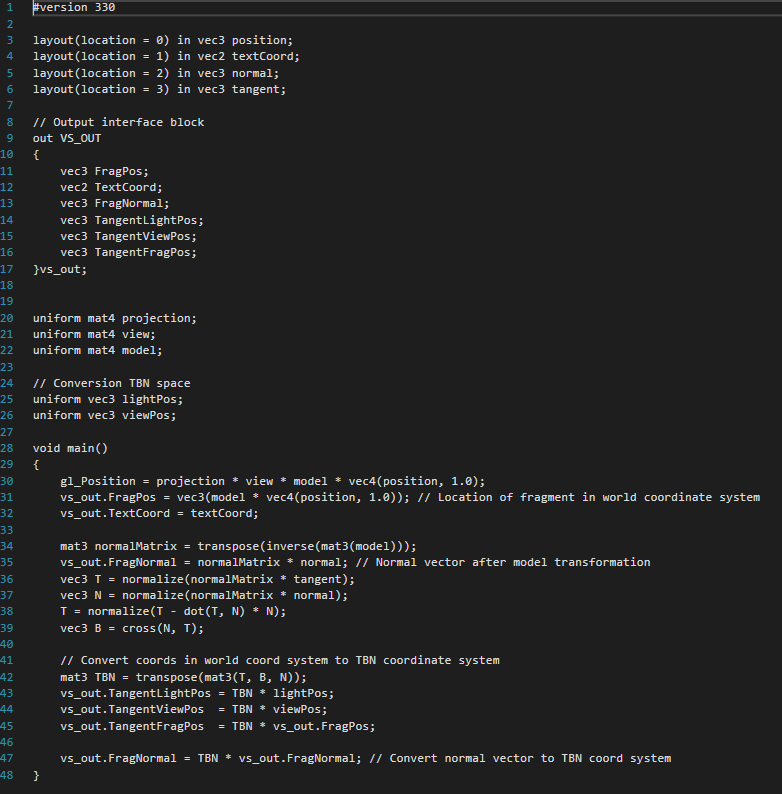
[6]

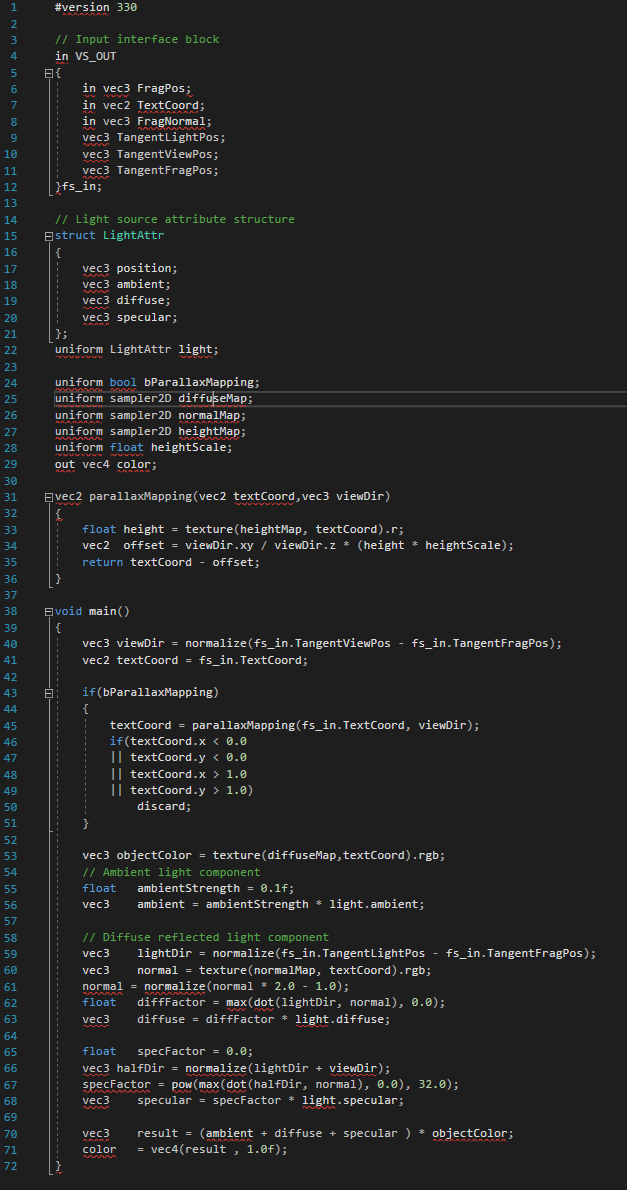
Original normal map Edited to fit UVs

[7]

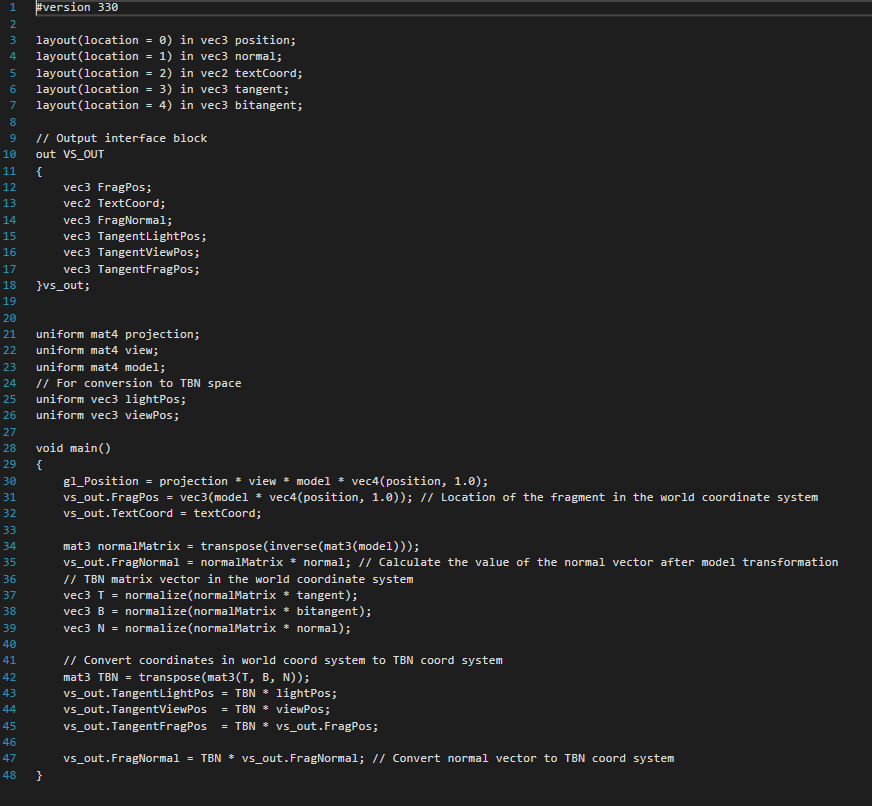


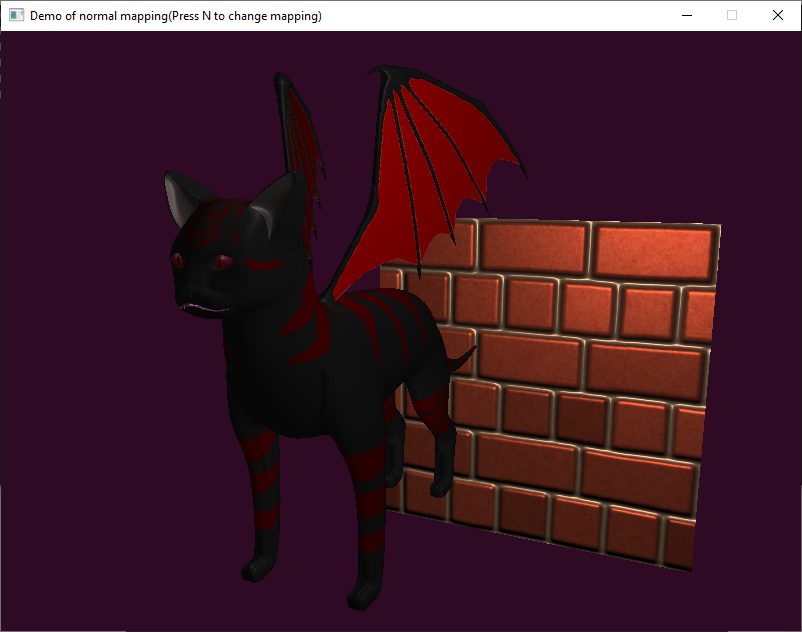
[8]



[9]

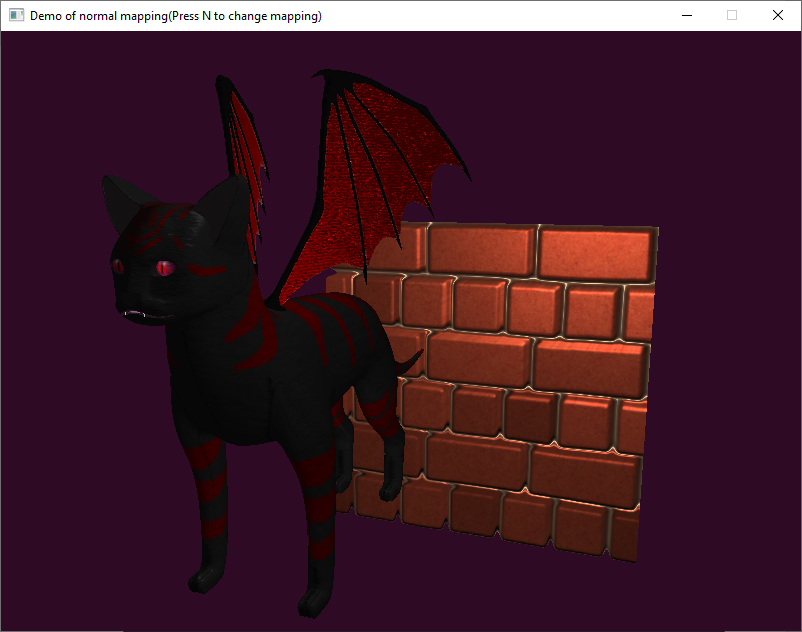
[10]





[11]

Parallax and normal mapping off



Parallax and normal mapping on