Advanced Graphics Programming

Student name: Wasili Prattis
Student number: 500689732

Date: May 9th 2016

1. Primitives

By using a custom method for generating an octagonal prism that can be resized and redrawn with more/less vertices, I have fulfilled the third column of the assessment rubric. The mesh is dynamically adjusted on the CPU by using the `D3D11_CPU_ACCESS_WRITE` and the `D3D11_USAGE_DYNAMIC` flags when creating the vertex buffer.

The mesh can be resized vertically by using the ‘W’ and ‘S’ keys and horizontally by using the ‘A’ and ‘D’ keys. The vertex count can be adjusted by using the ‘Q’ and ‘E’ keys. This behaviour is handled in the DrawScene method.

The prism is generated by taking a centre position and using a radius and slice count to generate the vertices. This is done twice, once for the top cap and once for the bottom cap. Indices are then linked between the top vertices and the bottom vertices to form the prism.

This assignment was completed in the *Chapter 6 Box* project of the Luna solution.

2. Models

To complete this assignment, I have imported assimp (Open Asset Import Library) into my project in order to draw different kinds of mesh formats commonly found in the industry. This means I should be in the 'Advanced' section of the assessment rubric. To import the library, I had to make the library project using CMake, as described in the instruction on the assimp website. After making the library project and compiling it using Visual Studio 2015, I had to link the dependencies with my project and include the right header files.

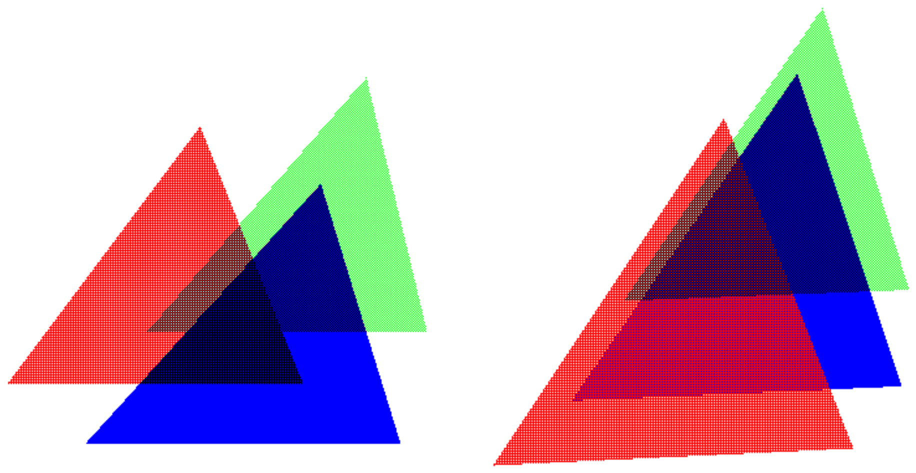
After reading the mesh file (tested successfully with *.fbx* and *.blend* formats) I convert the vertex and index data to a format that is usable by the Luna project and then the mesh is generated as expected after passing the data to the vertex and index buffers.

The models assignment was implemented in the *Chapter 6 Hills* project of the Luna solution.

3. Blending and Stencilling

Both colour and alpha blending have been applied to the project in order to create three triangles that blend together in the scene, which means I should be in the Intermediate section of the assessment rubric.
When changing the order of the triangles in multiplicative blending mode, I noticed that the triangle closest to the camera seems to be the dominant colour when calculating the resulting colour, despite the one in the front having a higher transparency value than the one behind it.

The green triangle has an alpha value of 0.5f, the red triangle has an alpha value of 0.8f and the blue one has an alpha value of 1.0f. When drawn in different distances from the camera, the resulting colours seem to change in multiplicative mode.
Left: drawn in order of alpha value, smallest value is further from the camera.
Right: red is drawn closer to the camera, then blue and finally green. Alpha values are still the same.



In additive mode, the resulting colours of the triangles did not change when the triangle positions were changed.

The blending modes were set as described in chapter 9 of the Directx11 book for this course: the values of the states were changed in the RenderStates.cpp file to create the desired effect. This state is then applied to the direct3d context inside the DrawScene method by calling the OMSetBendState method.

The code for this assignment can be found in the *Chapter 9 BlendDemo* project of the Luna solution.

4. Lighting

The specular components of the spotlight and pointlight can be changed by pressing the 1, 2 and 3 keys to increase the RGB values and the Z, X and C keys to decrease the RGB values. The diffuse components can be changed by pressing the 4, 5 and 6 keys for increasing the RGB values and the V, B and N keys for decreasing the RGB values. The Q and E keys can be used to change the range of the pointlight and the A and D keys to change the range of the spotlight. T is used to turn on toon lighting mode and U is used to turn toon lighting mode off.

Seeing as I've implemented changing values of lights by using the keyboard and the settings needed to apply toon lighting to the project, I should be in the Advanced section of the assessment rubric.

Toon lighting has been applied by implementing the description given in the book:

$$k'_d = f(k_d) = \begin{cases} 0.4 & \text{if} \quad -\infty < k_d \leq 0.0 \\ 0.6 & \text{if} \quad 0.0 < k_d \leq 0.5 \\ 1.0 & \text{if} \quad 0.5 < k_d \leq 1.0 \end{cases}$$

$$k'_s = g(k_s) = \begin{cases} 0.0 & \text{if} \quad 0.0 \leq k_s \leq 0.1 \\ 0.5 & \text{if} \quad 0.1 < k_s \leq 0.8 \\ 0.8 & \text{if} \quad 0.8 < k_s \leq 1.0 \end{cases}$$

These rules have been applied to the diffuse and specular components of both the spotlight and the pointlight.

The implementation of this assignment can be found in the *Chapter 7 Lighting* project of the Luna solution.

5. Texturing

A mobile phone is drawn on the screen, with a picture taken from my own mobile phone in its display. This means I should be in the intermediate section of the assessment rubric.

The mobile phone has a front side and a backside, just like a real mobile phone. Even though the front and back side of the phone look different, they are part of the same mesh. This is accomplished by selecting which portion of the texture should be drawn on each vertex of the mesh. The front side takes the right half of the texture and the back side takes the left portion of the texture.

To draw the screen, as separate set of vertices (a plane) is generated on top of the mobile phone, with its values adjusted so that it fits the display of the mobile phone perfectly. The texture containing the photo I took is then applied onto this plane in order to make it seem as if the picture was taken using the rendered mobile phone.

The results of this assignment can be found in the *Chapter 8 Crate* project of the Luna solution.

6. Shading

For this assignment, I used a shader I found on shadertoy.com with over 100 lines of code. This means I should be in the Advanced section of the assessment rubric.
The shader is used can be found here: https://www.shadertoy.com/view/4ddXDs

To implement this shader, I created a simple quad (2 triangles) and imported the shader I found on shadertoy by converting the code from GLSL to HLSL inside the color.fx file of the *Chapter 6 Shapes* project. The .fx file is read inside the BuildFX method and the shader is then rendered onto the quad.

GLSL keeps track of the time internally, so I had to find a way around this issue in order to convert this shader to HLSL. This was done by keeping track of the time in the main ShapesDemo.cpp code and then passing the global time value onto the color.fx code. Converting the rest of the code was a matter of following conversion guides I found on the internet:
https://alastaira.wordpress.com/2015/08/07/unity-shadertoys-a-k-a-converting-glsl-shaders-to-cghlsl/
and
https://msdn.microsoft.com/en-us/library/windows/apps/mt187142.aspx#compare

The code for this assignment can be found in the *Chapter 6 Shapes* project of the Luna solution.