|  |
| --- |
| Staffordshire university |
| 3D Computer Graphics CE003640-6 |
| Development of the Graphics Engine ‘iCare 3D’ |
|  |
| **By Alex J Davison (DV003874)** |
| **8-Jan-12** |

**Word Count: 2008**

|  |
| --- |
|  |

Contents

[Introduction 3](#_Toc313795501)

[Block Diagram of Pipeline 3](#_Toc313795502)

[Data Structures, Algorithm and Techniques 3](#_Toc313795503)

[Texture 3](#_Toc313795504)

[Specular lighting 4](#_Toc313795505)

[Bump map 5](#_Toc313795506)

[Object Scaling 7](#_Toc313795507)

[Object Translation 7](#_Toc313795508)

[Object Rotation 7](#_Toc313795509)

[Rotation in x axis 7](#_Toc313795510)

[Rotation in y axis 8](#_Toc313795511)

[Rotation in z axis 8](#_Toc313795512)

[Z-Buffer 8](#_Toc313795513)

[Lighting 9](#_Toc313795514)

[Parallel light 9](#_Toc313795515)

[Point light 10](#_Toc313795516)

[Spotlight 12](#_Toc313795517)

[Camera movement 12](#_Toc313795518)

[Data structure 14](#_Toc313795519)

[Critical Appraisal 14](#_Toc313795520)

[Conclusion 15](#_Toc313795521)

[Bibliography 16](#_Toc313795522)

[Appendix 16](#_Toc313795523)

[Appendix One: User Guide 16](#_Toc313795524)

# Introduction

This report outline how a 3D virtual environment has been created us HLSL (High Level Shader Luggage) in conjunction with DirectX 10. This report will out with use of block diagram and short description how, the graphics pipeline was utilized to create the 3D environment. This report will out the data structures that were used to create the application as well as any algorithms and/or techniques used. This report also contains a critical appraisal of the work that has been carried and a conclusion as to how well the project went.

# Block Diagram of Pipeline

Figure 1 - Pipeline diagram shows a block diagram of the applications flow of processing the data to render the application final 2D picture.



Figure 1 - Pipeline diagram

# Data Structures, Algorithm and Techniques

In this section of the report we cover the data structures, algorithm and techniques used with in the application.

## Texture

Within the 3D scene are cubes that are made up of polygons that are none shared vertexes. Figure 2 - Screen shot of application wire frame with back face culling shows the rasterizer state being set to wireframe and back face culling switched on.

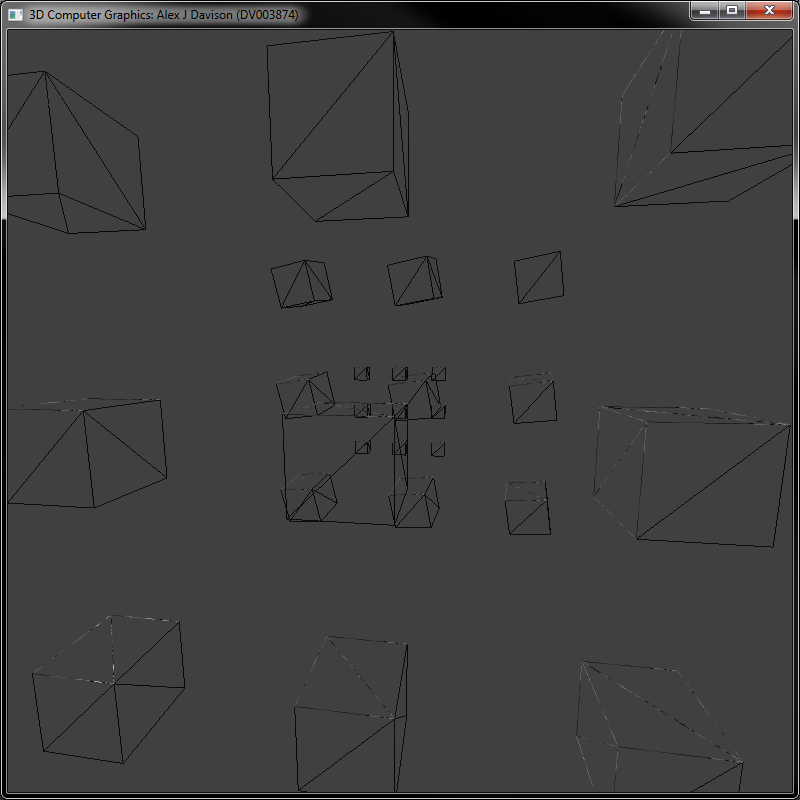


Figure 2 - Screen shot of application wire frame with back face culling

The texture as show in Figure 3 – Diffused texture this texture coordinates system is mapped to match the coordinates of that of the cube. This texture is used for the diffused aspect of the lighting within the 3D scene.



Figure 3 – Diffused texture

The texture is read in by the shader using the sampling technique as shown in Figure 4 - Texture sampling technique.

Figure 4 - Texture sampling technique

//Texture sampling information

SamplerState gTriLinearSam

{

Filter = MIN\_MAG\_MIP\_LINEAR;

AddressU = Wrap;

AddressV = Wrap;

};

## Specular lighting

The specular texture is also sampled using the same method in Figure 4 - Texture sampling technique and also translated to cube coordinate. This is then used to calculate the specular light component, this covered later on in the report.



Figure 5 - Specular texture

## Bump map

Figure 6 - Normal texture shows the normal texture that is used to achieve the bump mapping for the application. This was achieved by reading the normal from the texture using the sampler in the Figure 4 - Texture sampling technique. We also provide the facet a normal and tangent. We do not need a bi-normal as we can work this out using the normal and tangent



Figure 6 - Normal texture

The normal from the texture is then uncompressed from the texture by using the following calculation:

Next we need work out the bi-normal this achieved by the following equation:

Next we create tangent, bi-normal and normal matrix, this achieved by normalizing the normal throughout the shader to ensure that the entreaty of the vectors are maintained. For the tangent we carry out the following equation. being the tangent in world space, being the normal in world space.

Then we normalize the result. For the bi-normal we must work it out using the tangent and the normal. This is achieved by the following equation.

Figure 7 - Alterative to normal texture (Bump height texture) shows an alternative texture that could have been used for the bump mapping. However the used of this texture does not have as much information as the Figure 6 - Normal texture.



Figure 7 - Alterative to normal texture (Bump height texture)

The lack of information is due to the use of grey channel which only allows for the two seemingly channel of depth, whereas Figure 6 - Normal texture used three channel, red, green and blue or the bump/depth mapping of the surface.

## Object Scaling

To scale the object we add a vector to the objects world matrix as shown below. The matrix is an example of how you would scale an object.

## Object Translation

To translate/move an object within the game world we must apply it to the world object which will be carried into the world matrix of the world object.

## Object Rotation

The rotations of the objects within the 3D world are calculated using the three different calculations for each axis.

### Rotation in x axis

For the rotation of the object within the x axis we uses the below calculation on the object world matrix.

### Rotation in y axis

For the rotation of the object within the y axis we uses the below calculation on the object world matrix.

### Rotation in z axis

For the rotation of the object within the z axis we uses the below calculation on the object world matrix.

## Z-Buffer

Z-buffering is the method of ensuring that right objects are viewable in the correct order rather than the order in which they are drawn. This was achieved by using the following depth stencil as show in Figure 8 - Depth stencil descriptio.

Figure 8 - Depth stencil description

depthStencilDesc.Width = 800;

depthStencilDesc.Height = 800;

depthStencilDesc.MipLevels = 1;

depthStencilDesc.ArraySize = 1;

depthStencilDesc.Format = DXGI\_FORMAT\_D24\_UNORM\_S8\_UINT;

depthStencilDesc.SampleDesc.Count = 1; // multisampling must match

depthStencilDesc.SampleDesc.Quality = 0; // swap chain values.

depthStencilDesc.Usage = D3D10\_USAGE\_DEFAULT;

depthStencilDesc.BindFlags = D3D10\_BIND\_DEPTH\_STENCIL;

depthStencilDesc.CPUAccessFlags = 0;

depthStencilDesc.MiscFlags = 0;

m\_pd3dDevice->CreateTexture2D(&depthStencilDesc, 0, &pBuffer);

m\_pd3dDevice->CreateDepthStencilView(pBuffer, 0, &mDepthStencilView);

## Lighting

The light calculation are were inspired by (Luna, 2008). The calculation have been optimised to ensure that any unnecessary code is omitted from the shader.

### Parallel light

For the a parallel light we calculate light vector which aims the opposite direction the light rays travel this is achieved by the following equations:

Then we work out the ambient and diffuse components and add them to final component. This achieved as follows, where is diffused light and where is ambient:

Next we work out the diffuse factor. We do this by find the dot product of using the light vector and surface normal:

Next we check that the diffuse factor is greater than zero. If it is then we work the following otherwise we return the pixel colour.

Next we work the vector

Now we work out the reflective vector

Now we work out the specular factor, as follows:

Once we have calculated the specular factor we the combine all the components together to as follows:

This calculates the pixel’s colour with specular lighting.

### Point light

For a point light we first must calculate the light vector using the lighting position and the surface point as follows:

Next we work out the magnitude of the light vector to determine the distance between the two. As follows:

Next we check to see if the distance between the surface and the light is going to actually hit the surface by the light’s range. If the distance is too great then we set the pixel to zero. Next we normalize the light vector as follows:

We then multiple the diffuse and ambient components as follows:

Next work out the diffuse factor, we work this out by using the surface normal and the light vector as follows:

Next we check to ensure that the diffuse factor is greater than zero if it less or equal to zero then pixel is black else we calculate the specular power as follows, being the specular alpha channel:

Next we work out the normalized vector from the eye position and the surface position and work out the reflection once we have worked this out we work out the specular factor as follows:

Once we have calculated the specular factor we the combine all the components together to as follows:

Finally we take in to attenuation as follows:

### Spotlight

First for calculating the spotlight we must firstly calculate the point light. Once we have completed we normalise the vector between the surface position and light position.

Once we have specular component of the lighting we can then add it to the other components. This is achieved by time current light component by the specular as follows:

This calculates the pixel’s colour with specular lighting.

## Camera movement

For the camera movement, we need ,the number of viewing points and the radius of the circle that the camera must move around. We do the following calculation to for , the increments.

This give us in degrees. Once we have we simple loop though each point until we reach the number of point. This is to workout each incremental step. The calculation is as follows:

Calculate degrees to radon:

Calculate increment:

Calculate x coordinate:

Y stays as constant:

Calculate z coordinate:

## Data structure

Figure 9 - Application UML show a UML (Universe Modelling Language) this shows the structure of the application.

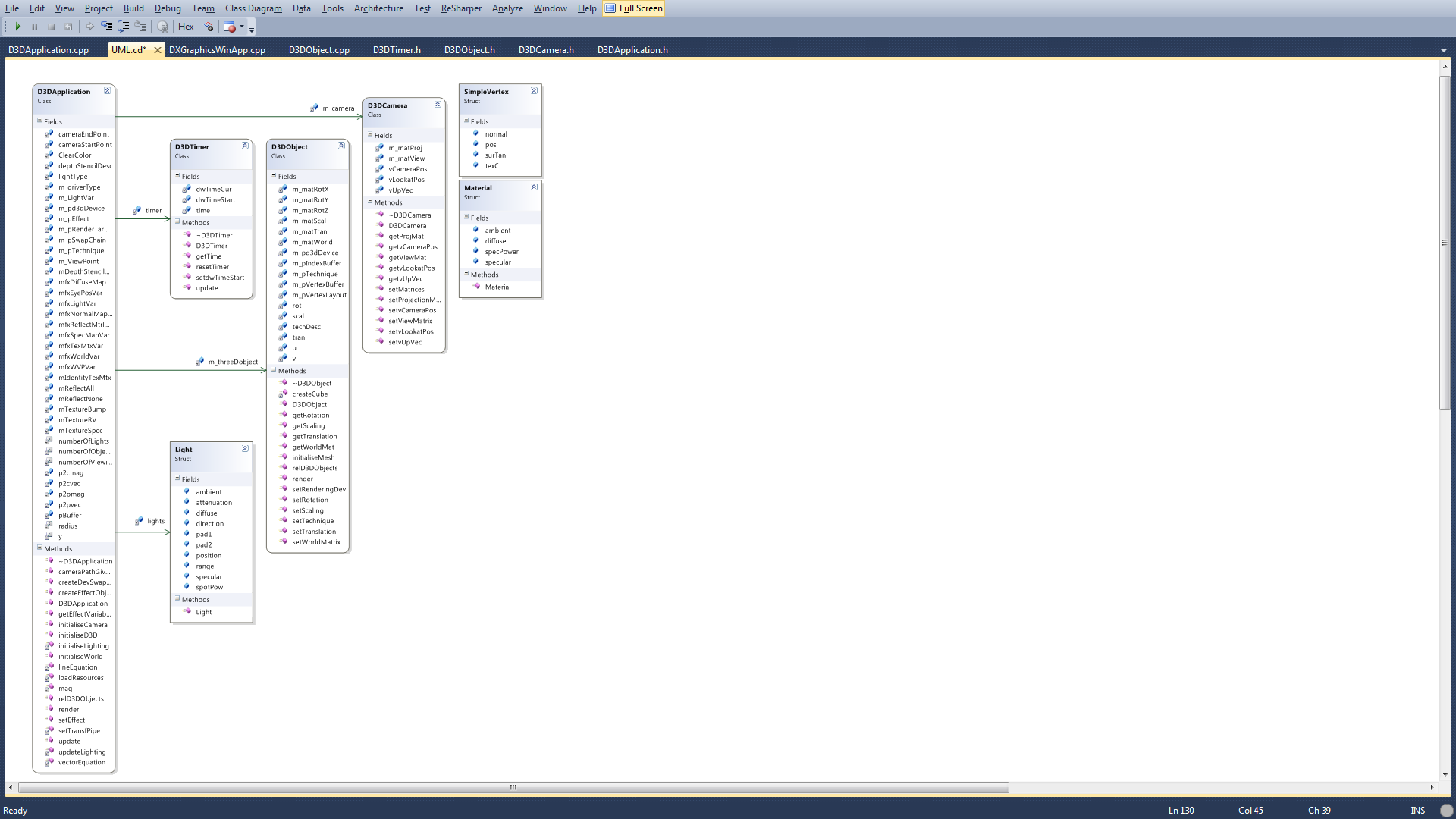


Figure 9 - Application UML

# Critical Appraisal

The project was a relative success in demonstrating some of the 3D graphical effects that are achievable with DirectX 10. However there is room for improvement. Such as different methods of executing some of the effect used in the 3D environment. Some of criteria were not achieved that was within the scope of the project.

Deformation of vector(s) was not achieved with computation of the deformation using force(s). This would have been a demonstration of car crumple zone with in a vehicle simulation.

Piecewise linear approximation was achieve however, this is not a very smooth as spline curves. Spline curves are more aesthetically pleasing however, a spline curve requires more calculation and more CPU usage.

In the application there is no use of curved surfaces. This not a very good demonstration of how an automobile is created. All automotive vehicles have a curved surface of some description. It would have been a good demonstration of the lighting model used if there was a curved surface within the 3D world scene.

# Conclusion

As you can see from Figure 10 - Screenshot a lot of the effects and criteria of the application however there is room for improvement. The subject and content of this is relative difficult one. It is difficult one as there was in the introduction of HLSL this was exacerbated by the introduction of the flexible graphics pipeline of DirectX 10. The use of the (Luna, 2008) and (Lengyel, 2011) however helped with the developed of the application. Using the (Luna, 2008) method for lighting and normal helped. Changing the code to suit the application helped with the applications development.

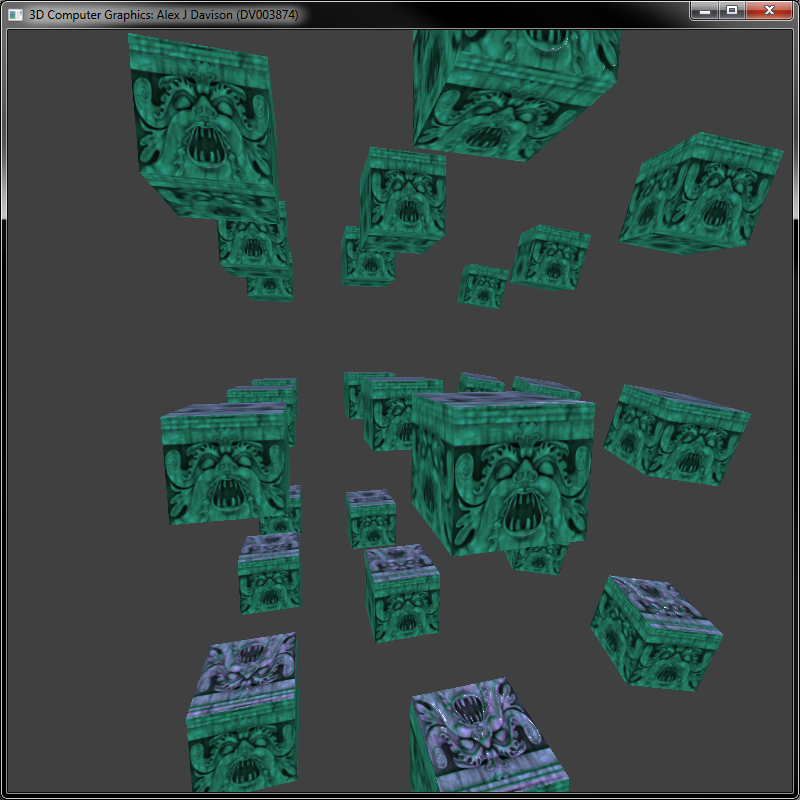


Figure 10 - Screenshot

# Bibliography

Lengyel, E. (2011) *Mathematics for 3D Game Programming and Computer Graphics*, Boston MA: Course Technology PTR.

Luna, F.D. (2008) *Introduction To 3D Game Porgramming With DirectX 10*, Sudbury, MA, USA: Wordware Publishing, Inc.

# Appendix

# Appendix One: User Guide

|  |  |
| --- | --- |
| Input | Action |
| 1 | Parallel light |
| 2 | Point light |
| 3 | Spotlight |
| 4 | All lights |
| Z | Zero ambient, diffused and specular light |
| W | Increase ambient red channel by 0.0003f |
| S | Decrease ambient red channel by 0.0003f |
| E | Increase ambient blue channel by 0.0003f |
| D | Decrease ambient blue channel by 0.0003f |
| R | Increase ambient green channel by 0.0003f |
| F | Decrease ambient green channel by 0.0003f |
| T | Increase diffused red channel by 0.0003f |
| G | Decrease diffused red channel by 0.0003f |
| Y | Increase diffused blue channel by 0.0003f |
| H | Decrease diffused blue channel by 0.0003f |
| U | Increase diffused green channel by 0.0003f |
| J | Decrease diffused green channel by 0.0003f |
| I | Increase specular red channel by 0.0003f |
| K | Decrease specular red channel by 0.0003f |
| O | Increase specular blue channel by 0.0003f |
| L | Decrease specular blue channel by 0.0003f |
| P | Increase specular green channel by 0.0003f |
| ; | Decrease specular green channel by 0.0003f |