Graphics Engine Development

**2009**

Alex J Davison (DV003874)

Staffordshire University

1/13/2009



(Mo3del)

Dr Len Noriega

Contents

[Application Description 3](#_Toc251099325)

[Requirements of Application 4](#_Toc251099326)

[Data Structure 4](#_Toc251099327)

[Storage of the Scene Components within a Scene Graph 4](#_Toc251099328)

[The Importation of Model(s) Within the Application 10](#_Toc251099329)

[Movement 10](#_Toc251099330)

[Dynamic Configuration of Viewing Parameters 11](#_Toc251099331)

[Animation/Control of Scene Objects 11](#_Toc251099332)

[Looks 13](#_Toc251099333)

[Texturing, Lighting Shading 13](#_Toc251099334)

[Lighting and Shading 14](#_Toc251099335)

[Introduction 14](#_Toc251099336)

[Reason for lighting and shading choice 14](#_Toc251099337)

[How Gouraud works 16](#_Toc251099338)

[Conclusion 17](#_Toc251099339)

[Summary 17](#_Toc251099340)

[Objectives Achieved 17](#_Toc251099341)

[Objectives Not Achieved 17](#_Toc251099342)

[Overview 18](#_Toc251099343)

[User Guide 18](#_Toc251099344)

[Bibliography 18](#_Toc251099345)

[Appendix 19](#_Toc251099346)

[Appendix One 19](#_Toc251099347)

# Application Description

The game will be a flight simulation of low level flying through a terrain that has obstacles in the way of the flight path of the plane. The choice of the game for the assignment was a relatively easy one to make, as it is easy to implement all the requirements within a game for the assignment, rather than the one of the alterative options.

The application which has been created will be based upon the Adobe Flash game called “CUBEFIELD” (CubeField). As you can see from Appendix 1, the game is a plane which moves across on the x axis to avoid the oncoming cubes. The longer the player moves the plane through the field of cubes, the more points the player earns.

In the CubeField game the camera leans from left to right, dependent on the players input to the game. However, the game that will be created on the basis of this game will not. Instead the game that has been created will try to fulfil many of the requirements laid out in the assignment. The game will implement many of the concepts within CubeField, such as the below points:

* Player must avoid the cubes.
* Camera movement in direction which a user wishes instead of lean.
* Automation of the cubes movement towards the plane

However, the application that has been created based upon CubeField will have more intricate implementation for more effects, such as the below points:

* Texturing
* Shading
* More complex plane model

This is implemented to give that game a more realistic look and feel.

# Requirements of Application

Below are the requirements of the applications as outlined in the assignment and how they are fulfilled within the application created.

## Data Structure

Below is the description of the data structure of the application created.

### Storage of the Scene Components within a Scene Graph

The storage of components such as meshes, facets, vertex are done within a scene graph and are structured to ensure simplicity when trying to move the objects. The best example of the scene graph are the cubes within the game; each cube is made up of a mesh which in turn is made up of a facets, which then in turn is made up of vertexes.

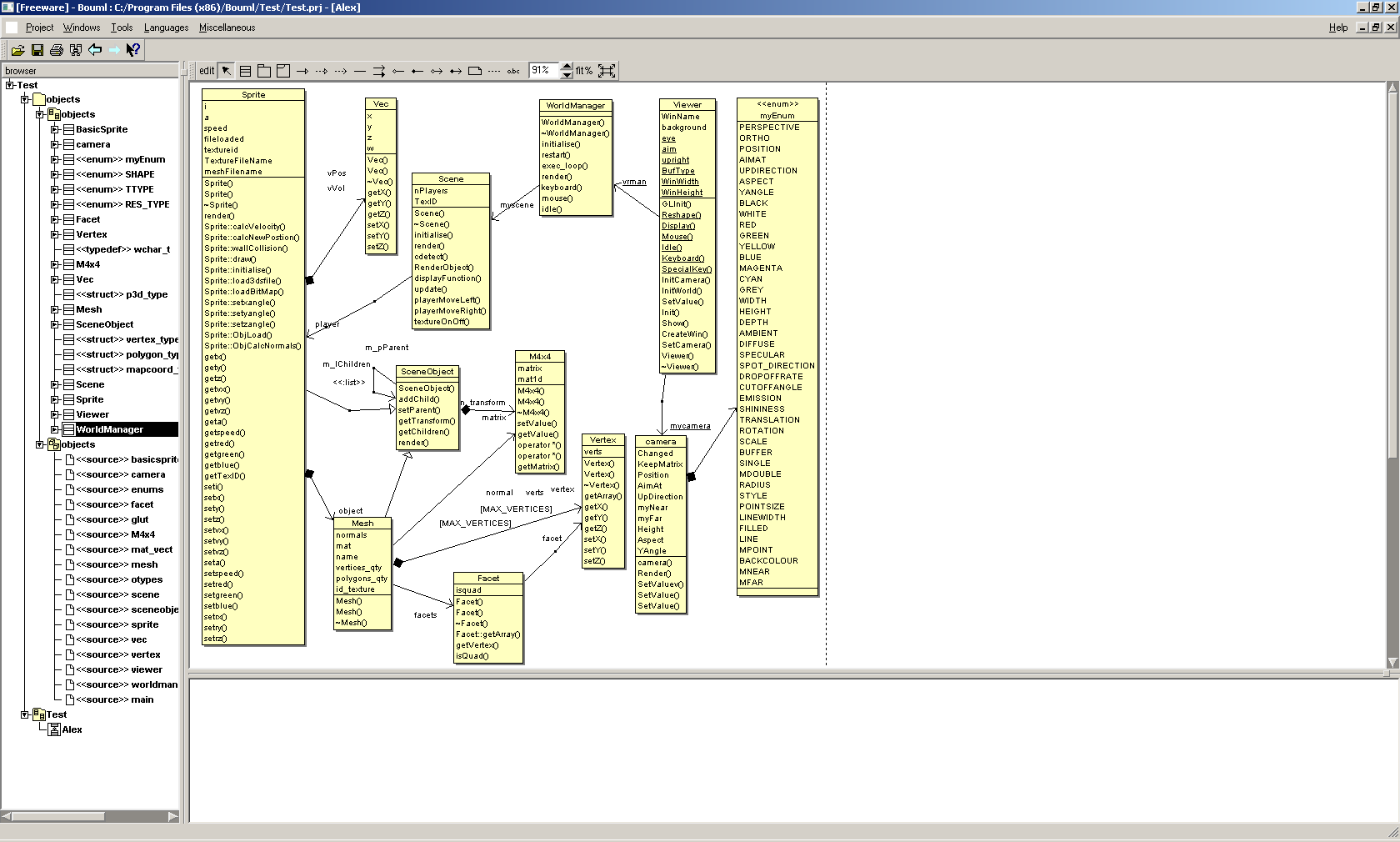
There are two main cubes within the application and then all the other cubes are children of the thoughts cubes. All the children cubes move in relevance to the two main cubes. The scene graph would on paper look like the below diagram.



As you can see from the diagram, you can see clearly that as the first cube is translated, scaled or rotated the other cubes that follow the main cube will in turn be changed according to the first cube. This is a simple but effective design that has been used to achieve the cube track for the game.

#### UML

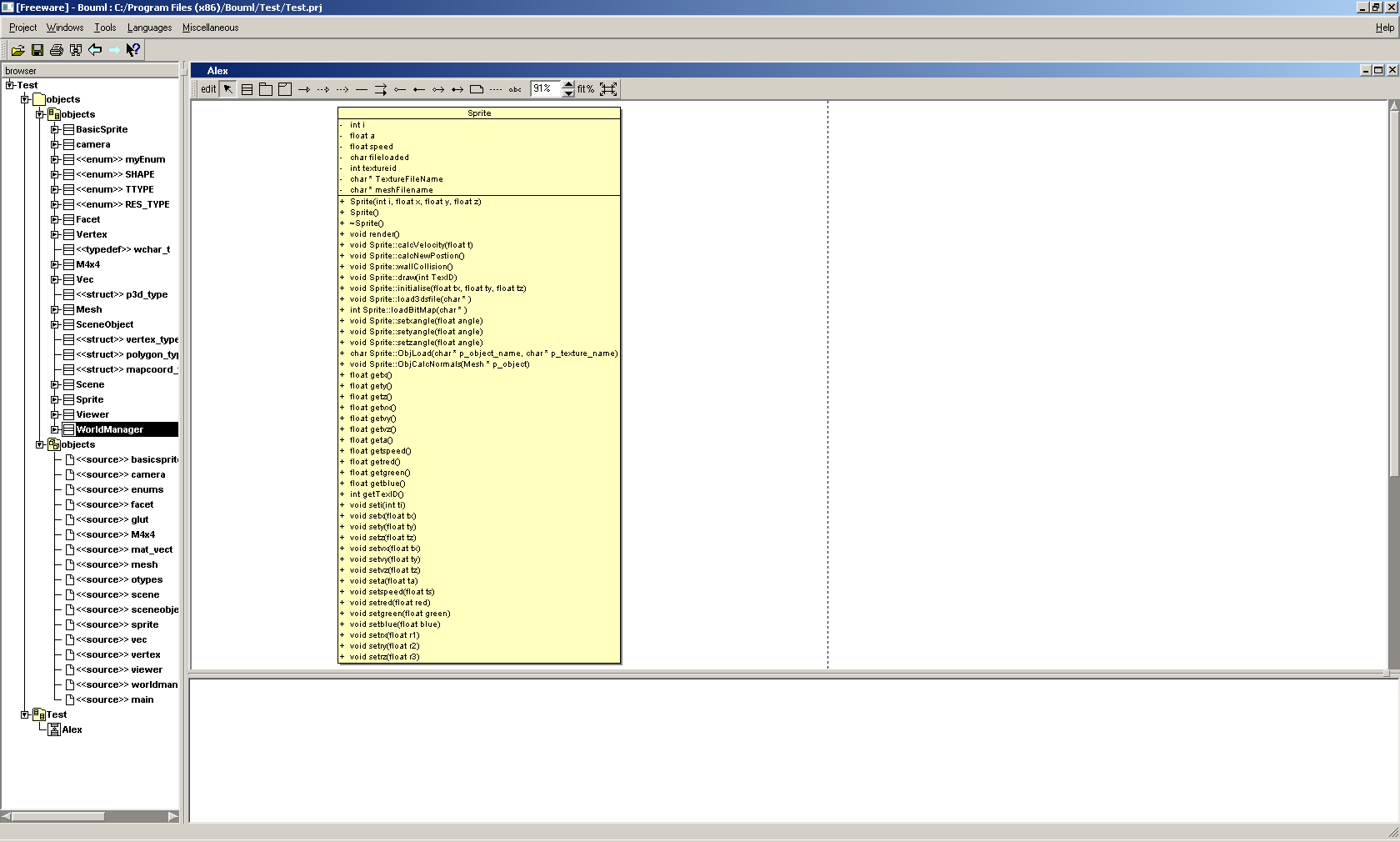
Below is the UML diagram showing the structure of the classes within the application.



#### Class Diagrams

Below are the class diagrams for some of the class in the above UML.

##### Sprite

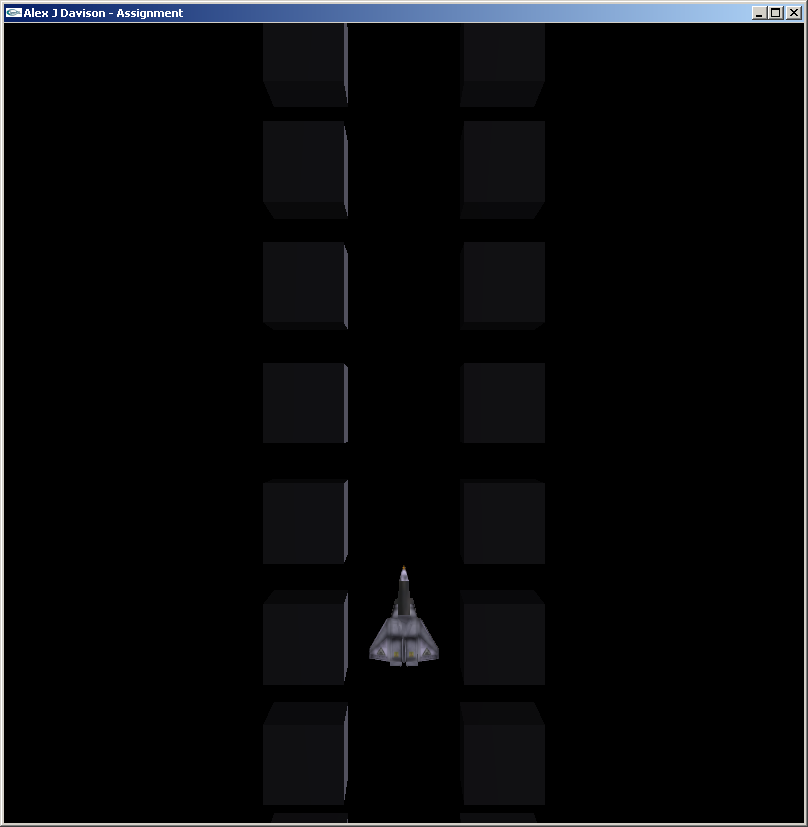
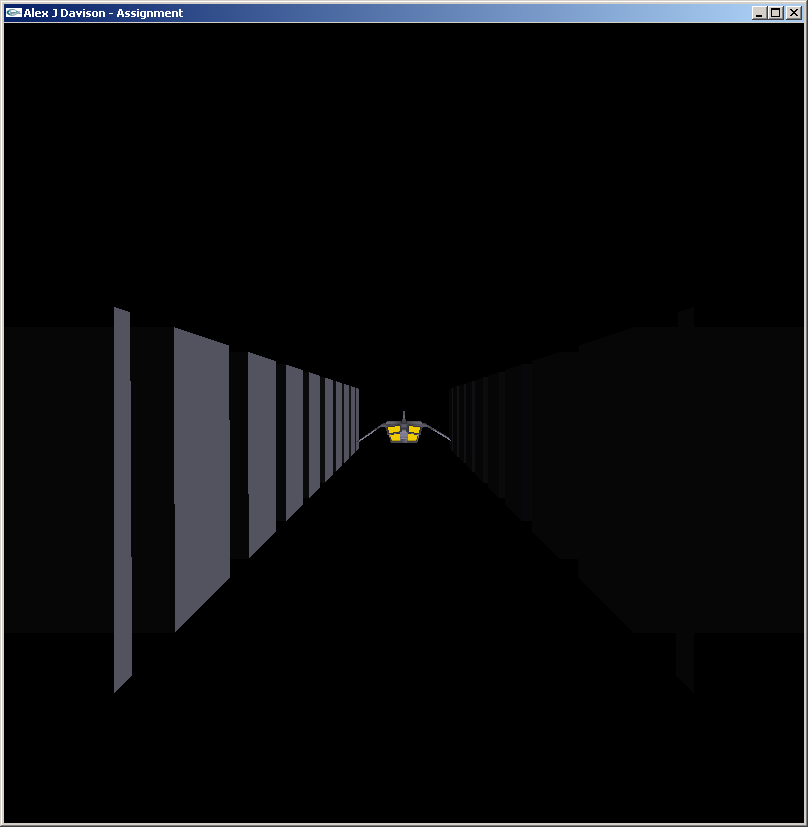


|  |  |
| --- | --- |
| Facet | World Manger |
|  |  |
| Scene | Viewer |
|  |  |
| M4x4 | Scene Object |
|  |  |
| Basic Sprite | Vec |
|  |  |
| Enum | Struct |
|  |  |
| Typedef | Camera |
|  |  |

### The Importation of Model(s) Within the Application

The application does read in two models; one which is a text file and the other is of a plane made from third party software called 3D Max. Both models are read in to the structure that has been created. However, they are imported at different levels within the structure.

The cubes are imported at the mesh level of the object orientated structure. The model of the plane is imported at the sprite level of the data structure. Due to the plane being imported higher up in the structure means that the plane is easier to control and manipulate within the application.

**Figure 1 Figure 2**

Figure 1 is a top down view of the plane model and the cubes. Figure 2 is rear to front view of the plane model and the cubes.

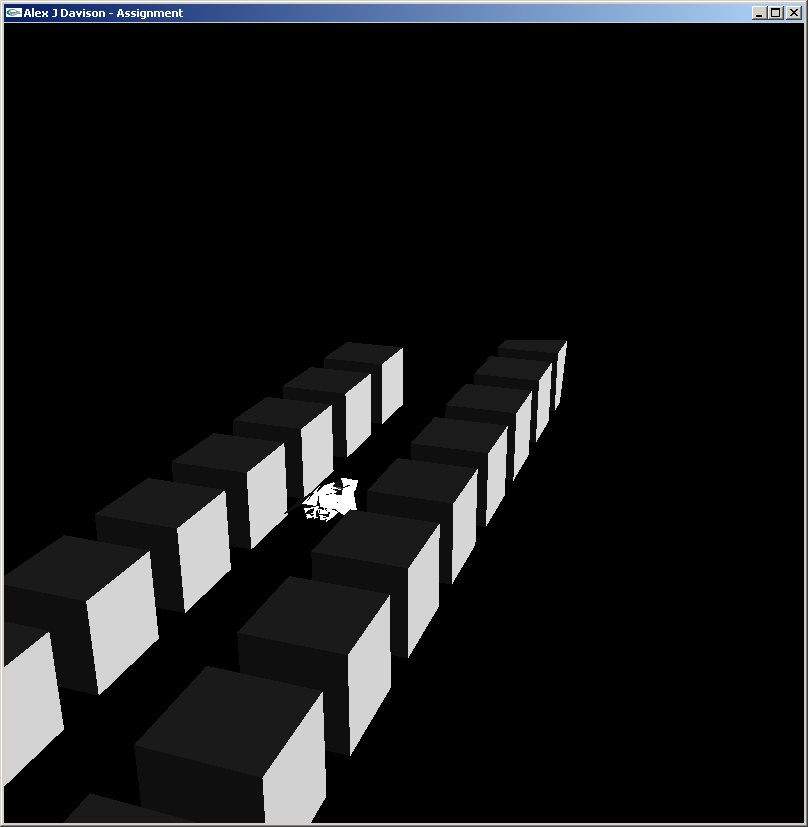
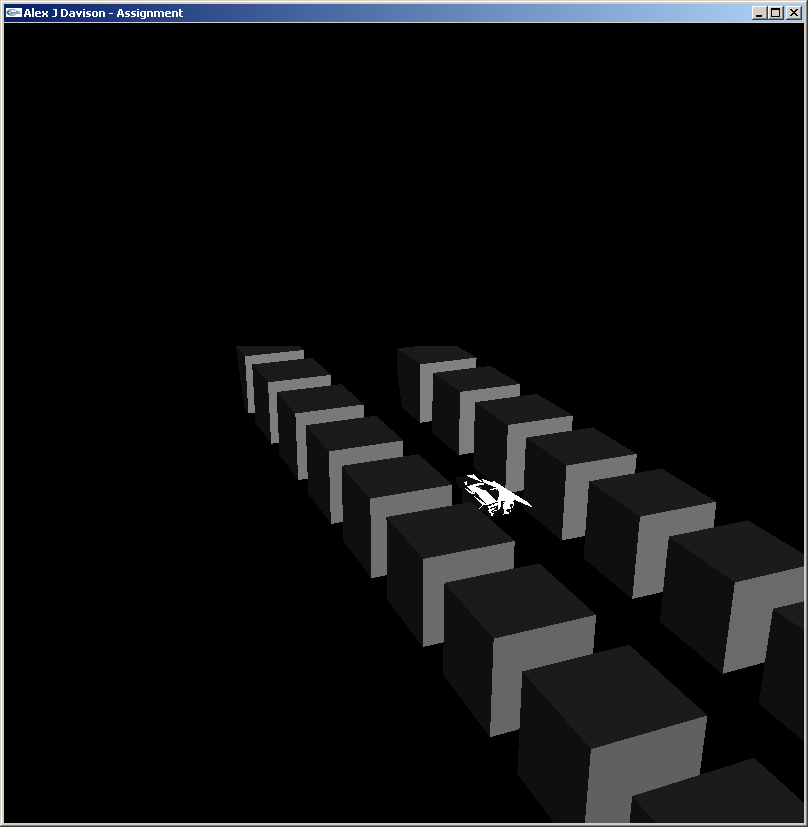
## Movement

Below is the description of the movement implemented within the application created.

### Dynamic Configuration of Viewing Parameters

Due to the structure of the application being object orientated, the dynamic configuration of the viewing parameters was relatively easy to achieve. The viewer is defined as a “viewer” within the application. The “viewer” object has three arrays of floats. Each array contains three float values which give the position of the viewer, where the viewer is looking and also which way is up for the viewer. The viewer then in turn updates the camera object within the application which controls the camera for the view within the 3D world.

To change the position of the camera meant that when the user inputted one of the relevant key stokes (see user guide section) for the camera, the camera would move in the relevant position to the increase in value for the camera position. The camera’s new position is then update within the update loop.

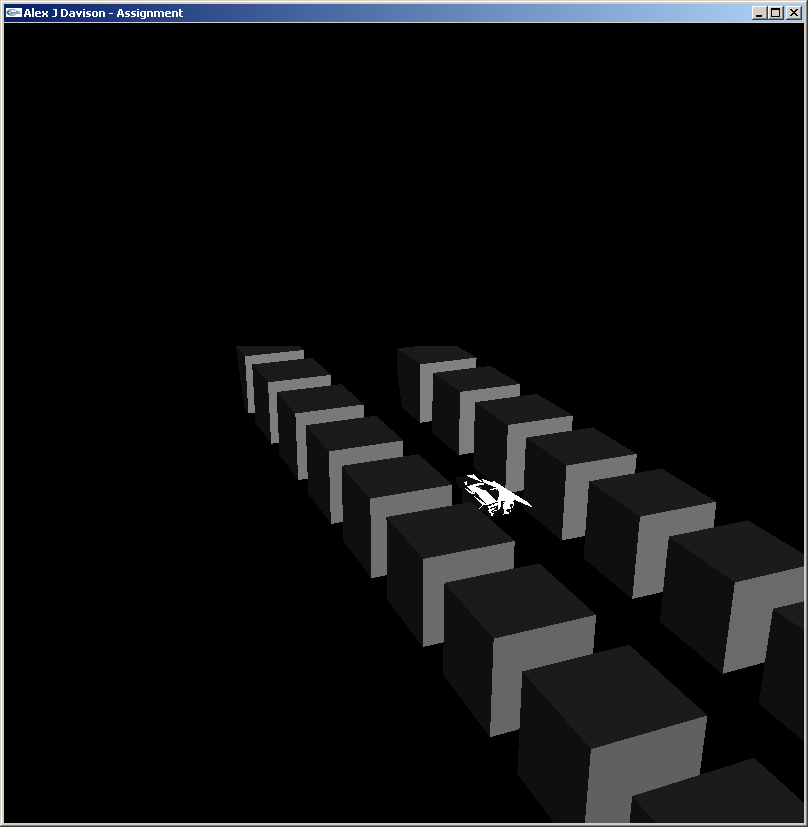
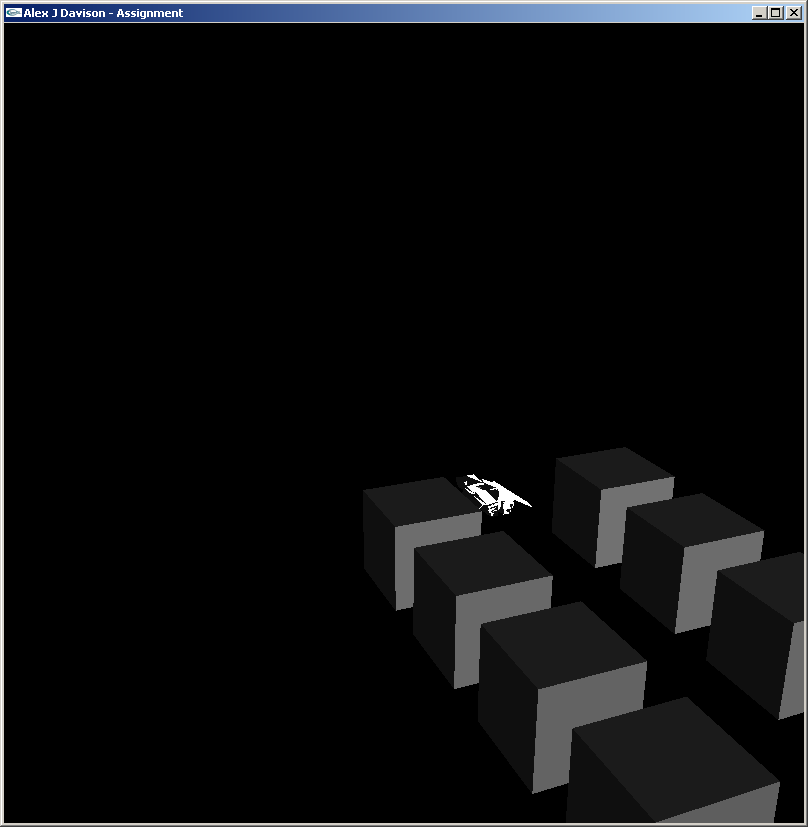
 

**Figure 3 Figure 4**

[[1]](#footnote-1)Figure 3 is of the camera moved using the q key and figure 4 is of the camera being moved with the e key.

### Animation/Control of Scene Objects

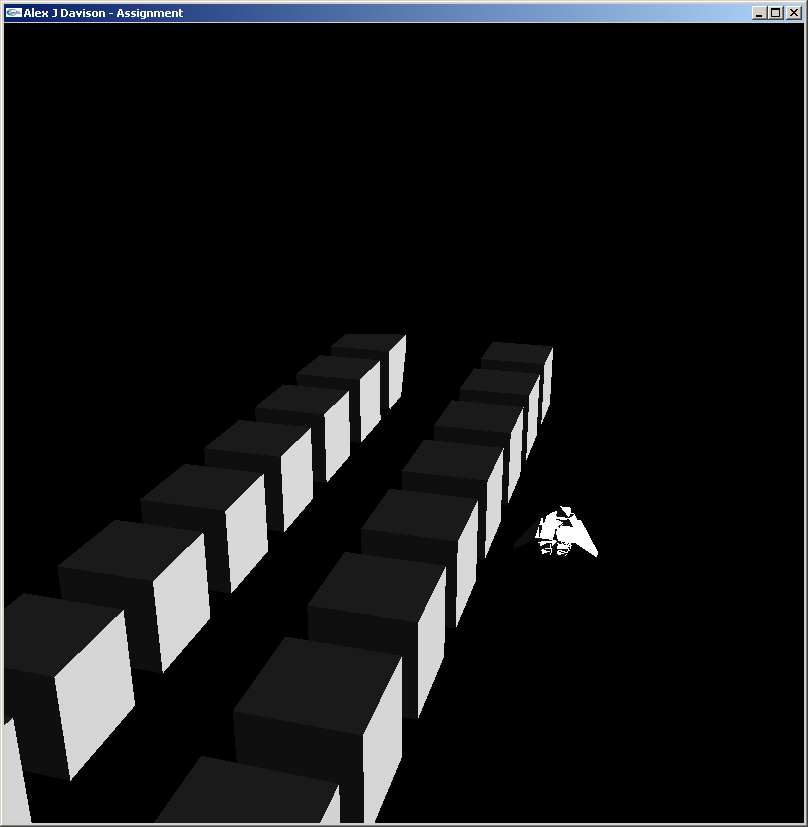
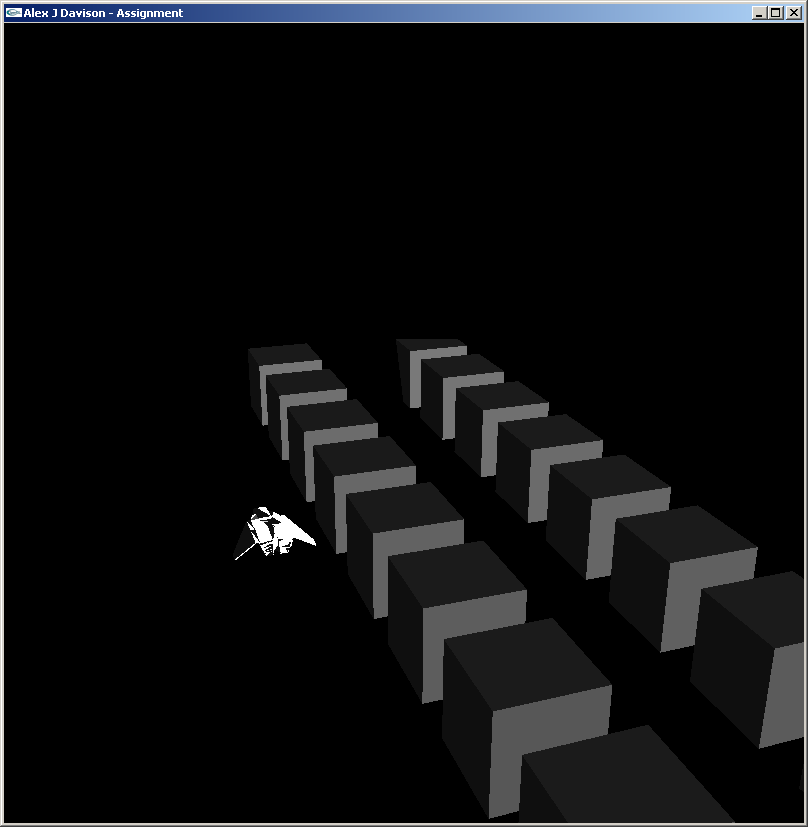
In the application the animation of the cubes is automated, so that the cubes move negatively in the z axis, so that the cubes move towards the plane. The cubes move in a linear linked list of the scene graph, so the first moves forward in accordance with the structure of the application. The cubes move within the position of the vertex for the corners of the cubes to be updated every frame.

**Figure 5 Figure 6**

[[2]](#footnote-2)Figure 5 shows the plane moving through the cubes. Figure 6 shows the plane coming to the end of the cubes, showing the motion of the cubes.

Due the plane model being a sprite object within the game, the model has an x, y and z total coordinate that moves the whole model. To change the position of the model you only need to change the position of the totality of the model. The model has a velocity which enables the model to move across continuously, until the user inputs another direction. If the user inputs the same direction more than once after the model has been stationary, then the model moves faster and faster within the given direction that was repeatedly inputted.

**Figure 7 Figure 8**

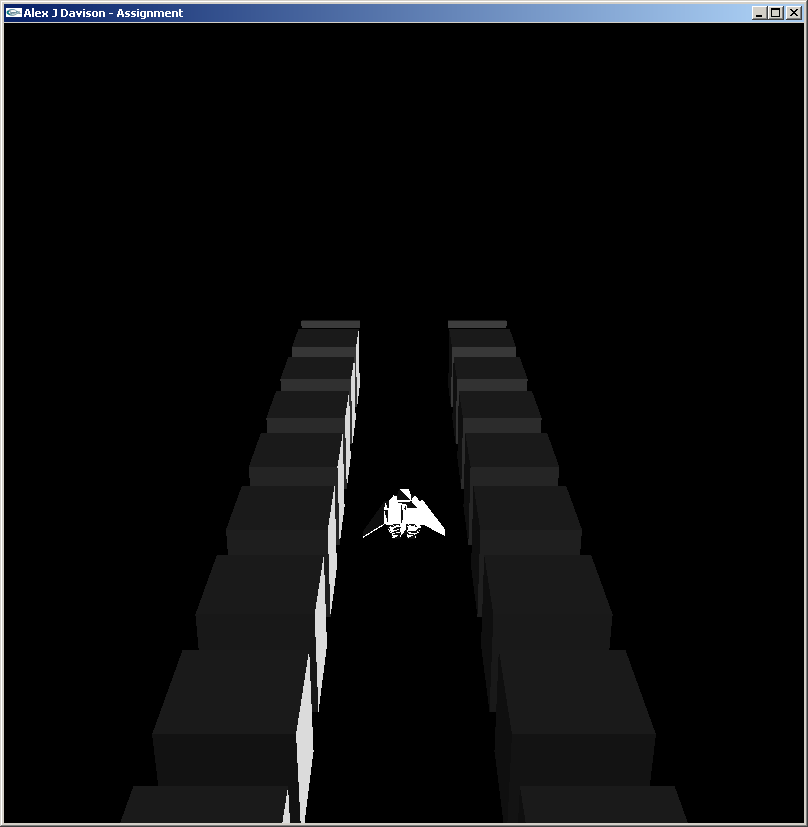
[[3]](#footnote-3)Figure 7 shows that the plane has moved to the right of the oncoming cubes. Figure 8 shows the plane has moved to the left of the oncoming cubes.

## Looks

Below is the description of the looks implemented within the application created.

### Texturing, Lighting Shading

Within the application there is light at the right of the scene to give the scene some light. The light within the scene has been placed to the right of all the objects, so that you can clearly see that there is light within the scene itself. The texts found the OpenGL Super Bible (Richard S. Wright 2005) help to accomplish the lighting within the scene.

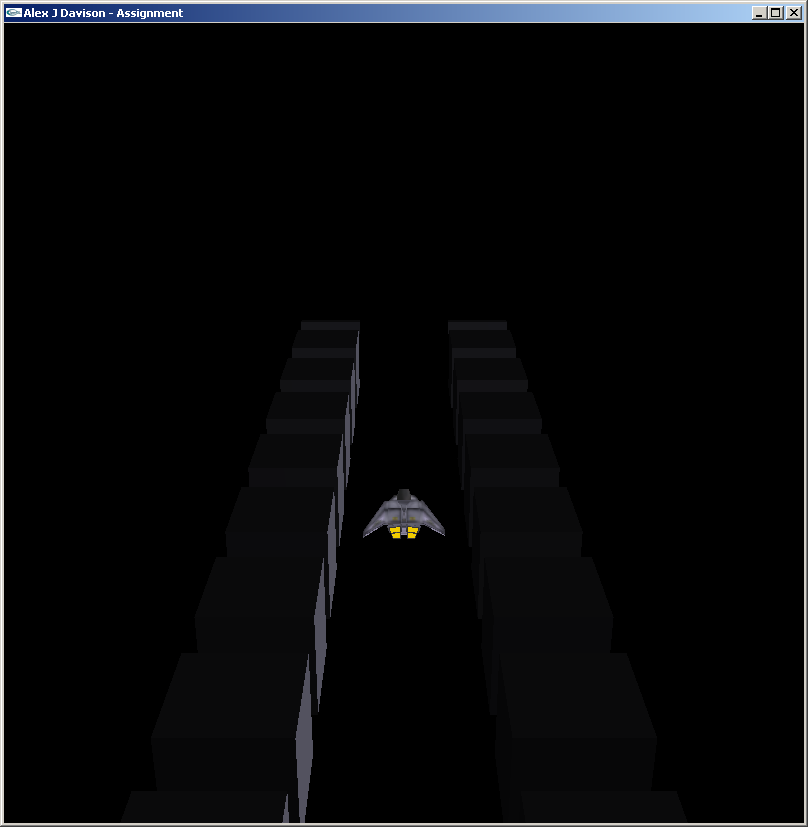


**Figure 9**

Figure 9 shows the smooth shading model used within the application.

Texture was implemented on to the 3Ds max model that was read in. The texture was meant to be read in once and then displayed every frame thereafter. However, it was not possible to render the texture every frame without first loading the bitmap image every frame. This was due to the buffer seemingly being empty every frame.

The texture is affected by lights defined within the scene, making the left hand side of the plane darker than that of the right hand side of the plane, where the light source emanates from.



**Figure 10**

[[4]](#footnote-4)Figure 10 shows the textures being used within the application.

# Lighting and Shading

Below is a description of the lighting and shading that was chosen for the application.

## Introduction

Lighting and shading within the application were chosen for the looks as well as the performance that the lighting and shading model offered. This is why the GL\_SMOOTH option was chosen.

To quote (Richard S. Wright 2005) “Smooth shading causes the colours along a line to vary”; this is an exactly what smooth shading is.

GL\_SMOOTH is based upon the Gouraud shading. The basic principal of Gouraud shading is that Gouraud creates a differing lighting effect of colour across a surface of a given object. Gouraud shading is mainly found within 3D applications that have a low number of facets. Gouraud is used in applications with low facets numbers because it doesn’t require that comprehensive calculation be carried out.

## Reason for lighting and shading choice

Gouraud shading gives a realistic look to 3D models, however it does not computationally take long to calculate. Had the project been to create a film quality scene such as “Monster vs Aliens” (as shown in figure 11), Gouraud shading would have been ineffective in creating the complex lighting effects to give the sense of realism. For a film quality image such as the one in figure 11, for the computation to work out the lights within that one frame could take hours. Hours to calculate one frame for a game is not acceptable as this is not real time information for the player.

Figure 11 was most likely rendered using ray tracing. Ray tracing takes a light ray and tracks its path from object to object. The ray has an effect on each pixel that the ray hits. If there are a lot of rays to trace, then it will take a long time to change all the pixels in the scene.



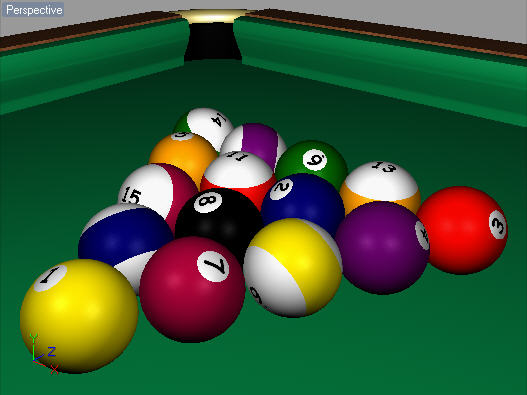
**[[5]](#footnote-5)Figure 11**

Flat shading like the one shown in figure 13 does not give the realism that has become expected of the current games market. However, it does give a simplistic approach to applying shading. The calculation time to work out the lighting within the scene is small in comparison to the ray tracing shading.



**[[6]](#footnote-6)Figure 13**

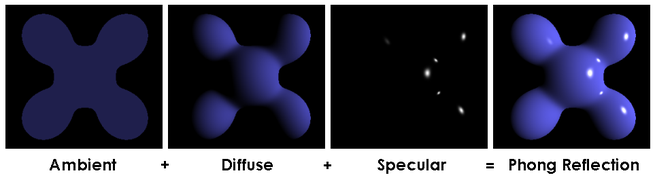
Gouraud shading makes a compromise between the simplistic shading and the complex shading, to offer a shading model that allows the realism of the ray tracing and the speed of the simplistic shading. Figure 12 shows a scene that may be found in a game using Gouraud shading.



**[[7]](#footnote-7)Figure 12**

## How Gouraud works

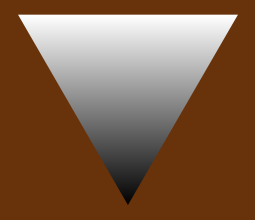
The principle behind the method of the Gouraud shading is to calculate the estimated normal of the facet at each vertex of the facet. Using the calculated estimate of the normal for the facet, then can you calculate the lighting computation based on the Phong reflection model which then produces colour intensities at the vertices.



[[8]](#footnote-8)**Figure 14**

Figure 14 shows an illustration of the Phong equation (Phong reflection model).

Gouraud shading has the great strength of interpolating the colour values by only using the three samples. The three samples make the calculation much less expensive to work out the intensivity of the facet. Figure 15 show an example facet that uses the calculation to work the different intensities at the different vertices.



**[[9]](#footnote-9)Figure 15**

## Conclusion

The reason for the use of Gouraud shading is mainly due to the realism that it can give for the low computation of the lighting effect. Unlike ray tracing, Gouraud can be calculated per frame, whereas flat shading can only change the colour of an entire facet. This is why Gouraud was chosen for the application.

# Summary

Below is a list of all the objectives achieved, not achieved and an overview of the overall goal.

## Objectives Achieved

* Texturing.- Plane is textured
* Reading a 3D model from third party software – Plane is a third party model
* Reading a model from a text file – Cubes are read in from a text file
* Scene graph of objects – Cubes are part of a scene graph
* Personal created translation matrix node
* Shading – Cubes and plane have a shading implemented on them
* Automation movement of cubes – Cubes move forward towards plane
* User controlled plane – Plane moves left and right when user gives input via keyboard
* Camera movable – Camera moves according to the input given via the keyboard.

## Objectives Not Achieved

* Collection detection – Collection detection was achieved in the tutorials. However, it was not achieved in the assignment, as the collection detection was meant to be implemented using the customised cubes.
* Statistics – Statistics were not implemented as collection detection was not achieved. Statistics of the file being read in was achieved but not game statistics.
* Looping cubes
* Cubes moving left and right – Was not achieved as attention was focused on the implementation of collection detection.

## Overview

Overall the application was not the greatest of successes when it comes to the aspect, as no game loop was implemented or game state. However, a lot was learned from the structure of the code and the implementation of the framework to get the application as far as it did. The application would have been complete to the full extent of the CubeField game that it was based upon. Overall a lot was learnt in the structure of the code and the implementation of the different aspects of the application.

# User Guide

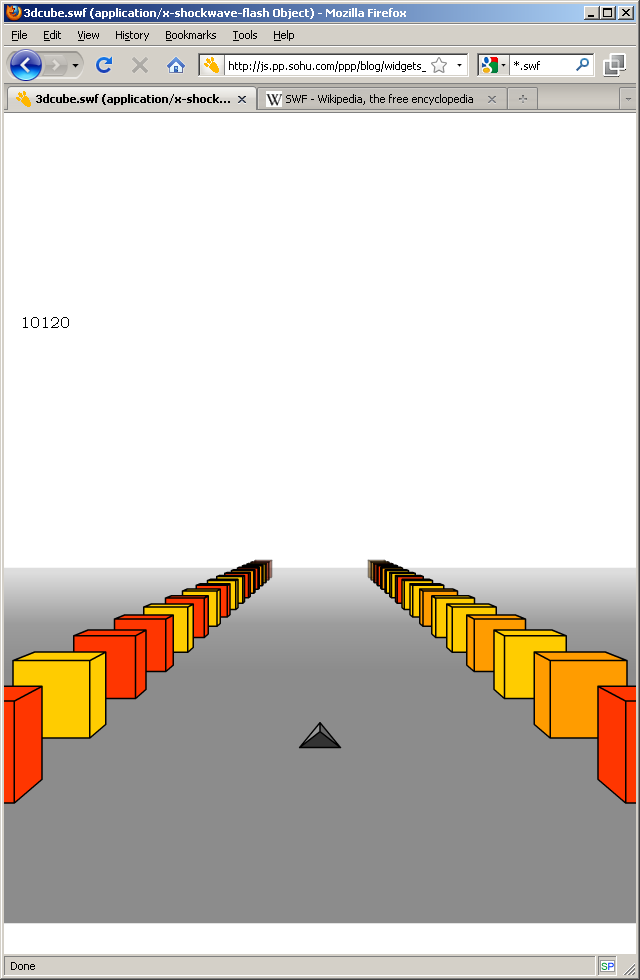
|  |  |
| --- | --- |
| **Control** | **Outcome** |
| q | Move camera right in a circular motion around the point of origin in the x axis. |
| e | Move camera left in a circular motion around the point of origin in the x axis. |
| r | Move the camera up in a circular motion around the point of origin in the y axis. |
| f | Move the camera down in a circular motion around the point of origin in the y axis. |
| t | Move the camera forward in a circular motion around the point of origin in the z axis. |
| g | Move the camera backwards in a circular motion around the point of origin in the z axis. |
| a | Moves the camera right in a linear motion in the x axis. |
| d | Moves the camera left in a linear motion in the x axis. |
| w | Moves the camera forward in a linear motion in the z axis. |
| s | Moves the camera backwards in a linear motion in the z axis. |
| F1 | Moves the camera above the origin of the world. (Looking from top to the bottom, Top of the plane/cubes) |
| F2 | Moves the camera flat on the x axis so it looks in to the world. (Looking from back to front. Back of the plane/cubes) |
| Page Up | Move the camera up in a linear motion in the y axis. |
| Page Down | Move the camera down in a linear motion in the y axis. |
| Left Arrow Key | Moves the plane to the left in the x axis. |
| Right Arrow Key | Moves the plane to the right in the x axis. |
| Up Arrow Key | Turns textures on and off. |

# Bibliography

|  |
| --- |
| C4Cafe (Unknown) Pool Balls [Online]. <http://www.c4dcafe.com/reviews/r10/images/enhanced_gouraud.jpg> [Accessed:12/01/10] |
| Computer Weekly (Unknown) Monster Vs Aliens [Online]. <http://www.computerweekly.com/PhotoGalleries/235560/917_30_Monsters-Vs-Aliens-Top-ten-CGI-films.jpg> [Accessed:12/01/10] |
| CubeField (Unknown) Game Called CubeField [Online] <http://js.pp.sohu.com/ppp/blog/widgets_0914/game/3dcube.swf> [Accessed:12/01/10] |
| Giant Bomb (Unknown) Money For Nothing <http://media.giantbomb.com/uploads/1/19148/680634-money_for_nothing_small_large.jpg> [Accessed:12/01/10] |
| Mo3del (Unknown) Picture of Bumble Bee from transformers [Online]. <http://mo3del.ru/uploads/posts/2008-05/1211819749_bumblbee_537_orig.jpg> [Accessed:12/01/10] |
| Richard S. Wright Jr & Benjamin Lipchak (2005)OpenGL Super Bible Third Edition Sam’s Publishing USA |
| Wiki Media (Unknown) Gouraud triangle [Online]. <http://upload.wikimedia.org/wikipedia/commons/thumb/2/2e/Gouraud_triangle.svg/120px-Gouraud_triangle.svg.png> [Accessed:12/01/10] |
| Wiki Media (Unknown) Phong Equation [Online] <http://upload.wikimedia.org/wikipedia/commons/6/6b/Phong_components_version_4.png> [Accessed:12/01/10] |

# Appendix

# [[10]](#footnote-10)Appendix One



1. Many other camera positions are achievable using the keyboard input stated in the user guide. [↑](#footnote-ref-1)
2. Note that the loop for the cubes has been disabled for figure 4 screenshot. [↑](#footnote-ref-2)
3. Please note that collision detection has not been enabled in these screen shoots. [↑](#footnote-ref-3)
4. Texturing was accomplished. However it creates a memory leak for known reasons, but there was not enough time to work out the fix for the problem. [↑](#footnote-ref-4)
5. (Computer Weekly) [↑](#footnote-ref-5)
6. (Giant Bomb) [↑](#footnote-ref-6)
7. (C4Cafe) [↑](#footnote-ref-7)
8. (Wiki Media) [↑](#footnote-ref-8)
9. (Wiki Media) [↑](#footnote-ref-9)
10. (CubeField) [↑](#footnote-ref-10)