**Assignment Self Evaluation Sheet**

**Mathematics for Computer Graphics Assignment 2**

**Student’s Name**: Jack Pitman

*This self-evaluation sheet is marked only on completeness (i.e. please be honest!). The purpose is to help you reflect on your performance and to help identify features of your work.*

|  |  |  |
| --- | --- | --- |
|  | **Yes** | **No** |
| Did I complete the minimum requirements for the assignment? | **X** |  |
| Did I add any extensions (i.e. more advanced features) to the assignment? | **X** |  |
| Did I read up on the subject beyond lecture / lab contact? | **X** |  |
| Did I spend enough time on the assignment? | **X** |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Very happy | Satisfied | Disappointed | Ashamed |
| How happy am I with what I submitted? |  | X |  |  |

|  |  |
| --- | --- |
| References for sources and tutorials I used: | Bresenham’s algorithm - <https://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html>  Bezier Curve - <https://javascript.info/bezier-curve>  Matrix - <http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/>  GLM - <https://glm.g-truc.net/0.9.2/api/a00001.html> |
| Main features of my project: | My program allows for users to input their own values for all shapes except the triangle, this allows for the user to experiment with different co-ordinates and different size shapes, the Bezier curve and fake 3D cube shows the advantages of this well.  The menu system for my program also allows the user to choose the shape they want to see, and enter the values required for the shape. |
| The best part of my performance was: | Getting the initial understanding of the library and being able to plot points on the window, then learning how to advance on that and draw lines and more complex shapes |
| The worst part of my performance was: | Not being able to create the menu in a way that means the program doesn’t close after the user has chosen and drawn a shape.  Also, my lack of understanding of matrices meant I had to read up a lot before attempting transformation. |
| One way in which I could improve my submission is: | Attempt more advanced shapes and algorithms such as a more efficient filled circle and a projection 3D shape that can rotate.  Split my program up into multiple files, each drawing their own shape and being modular. |
| One thing I will do to improve my next submission is: | Make sure I use a modular approach to the program so that less code has to be rewritten.  Make sure I attempt advanced features that will possibly allow me to aim for higher grades. |

**Report**

**Introduction**

This is my report on my MCG assignment, this assignment shows off my ability to use an API I have never seen before and draw shapes using advanced techniques such as matrix and Vectors.

**Previous Work**

A common part of computer graphics is ray casting, the process of firing rays from a “Camera” that then collide with objects within the graphical space, these are then rendered to the screen as objects. Ray casting was first coined by Scott Roth in 1982 and was first used in a popular mainstream game called Wolfenstein 3D (iD Software, 1992).

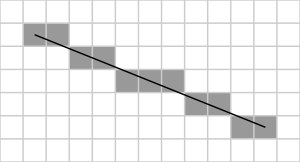
A typical Ray Cast is performed by firing rays from a central position (The camera) to every pixel of the window (A 600 x 800 window would have 480000 rays fired), for a static scene this is only done the once however can be done within a loop to handle moving objects or dynamic scenes.

The main strength of Ray Casting is that is extremely fast depending on the size of the “scene” being casted to, this allows it to be used in a variety of programs and games that can target low end hardware increasing the potential audience of the software using this method. Another strength for Ray Casting is how versatile it is, if a ray intersects with an object, consequent checks can be performed to check what that object is, if it is a collision volume then don’t render it, if it is an enemy or wall then do render it, if the Ray Cast is for a bullet then does it collide with an enemy, if so then kill the enemy and destroy the bullet, if it collides with a wall and is fired from the player, the player has collided with a wall and can no longer move in that direction.

All of these strengths make Ray Casting an extremely popular method for games and applications, however there are some weaknesses to Ray Casting, for example Ray Casting cannot be used to render complex scenes that require reflections/refractions or Global Illumination, this is due to the fact that no additional rays are created from the intersection of the first. Another weakness of Ray Casting is how complex it can become, for example a real bullet can travel through glass, however with a standard Ray Cast the ray would hit the glass and stop, making it unable to impact objects behind the glass.

Another attempt at fixing well known problems with computer graphics is the Bresenham Line Algorithm (Jack Elton Bresenham, 1962), before this algorithm was created, lines drawn diagonally were jagged and awkward to create and render, this algorithm aimed to reduce the jaggedness of diagonal lines by colouring adjacent pixels as the algorithm went further down the line, thus smoothing out the edges. This algorithm also aimed to simplify how lines were created at run time between points as multiple points could be passed into this algorithm creating lines between each, useful for fake 3D shapes or randomly generated 2D shapes.

Below is an example of a section of line drawn using the Bresenham algorithm which clearly shows the line being drawn diagonally downwards and how multiple pixels are coloured to make the line appear smooth.



**Analysis**

I am happy with how my work came out, I managed to render all of the primitive shapes and then managed to create and render a few more advanced ones such as a Bezier Curve.

I tried to program my work in such a way that made it at least slightly modular, especially when it came to the showLine method that I used often, this allowed me to not have to rewrite code I was using often allowing me to speed up development and make my code neater.

One of the strengths of my program is the neatness of the code, while not split into files, the code is split into methods and sections allowing for individual parts of the code to be singled out and reviewed without affecting other parts of the program.

A weakness of my program is the lack of a Ray Tracer, while I tried to work on one I could not get it to function to a standard I deemed acceptable for this assignment and so chose to omit it from the submission, this disappointed me as I had hoped to have a working Ray Tracer showing at least a basic lit scene.

As I spoke about in the “Previous Work” section, I used the Bresenham line algorithm in my program to draw the lines that are visible, this made it extremely easy to create a fake 3D cube based on the users inputted points and sped up development over having to create my own algorithm.

**References**

John Carmack., Wolfenstein 3D, iD Software, 1992.

*Colin Flanagan.,* [*https://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html*](https://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html)*, 1997.*