

# NEA Documentation

Jack Doughty

# Contents

## Analysis

Description

Clients

Prototyping

Research

Relating Existing Solutions

Objectives

## Design

User Interface Specification

Algorithms

Data Structures

File Organisation

## Technical Solution

[Link to PDF](#)

## Testing and Development

Testing Done in Development

Issues in Development

Final Testing

## Evaluation

Objectives

Evidence

# Analysis

## Description

I intend to complete an investigation into rendering three-dimensional graphics, and thus develop an educational platform to aid in teaching Key Stage 2 and 3 geometry. Written in C++; it will take advantage of OpenGL, a cross platform Application Programming Interface (API) for rendering 2D and 3D vector graphics, enabling the potential use of Hardware Accelerated Rendering to allow for greater performance. The GUI will be comparable to the Microsoft 3D Viewer application. (Figure 1)

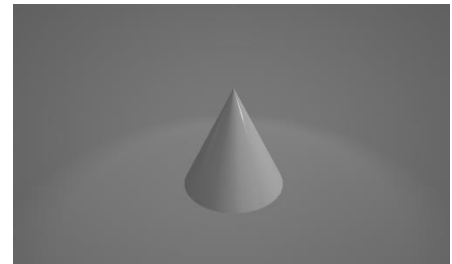


Figure 1 – A cone in Microsoft 3D Viewer

This problem is suited to a computational resolve because, as of beginning the development, there is a lack of effective teaching methods that are directly fitted for KS2 and 3 classes in respect to accessibility, complexity, and efficacy. This topic is most often taught through interacting with physical shapes, objects, and drawing diagrams - so by providing a computational medium to alternatively teach this, whilst providing a more engaging and 'in depth' insight into the topic, less resources are required and, potentially, a calmer learning environment is achieved.

## Clients

The program will be of use to all GCSE Mathematics and primary school students and teachers on Windows x32/x64/x86, Linux and MacOS platforms. It will be easily available as it will be freeware and will come as an executable file, so no dependencies will be required after compilation. Furthermore, given that the core consumers will be younger students, the layout of information on the display should be understandable, digestible, and gripping as much as it can be.

Referring to the AQA GCSE specification, sections G12 and G13 state that students should be able to '*identify properties of the faces, surfaces, edges and vertices of: cubes, cuboids, prisms, cylinders, pyramids, cones and spheres*' and '*...interpret plans and elevations of 3D shapes*'. Additionally, G16 and G20 state that the students should be able to calculate the '*volume of cuboids and other right prisms (including cylinders)*' and '*... find angles and lengths in right-angled triangles and, where possible, general triangles in two and three dimensional figures*'. My program will need to be of use to students in some of these respects.

I shared an in-person discussion with one of my maths teachers, who also oversees younger classes, and I picked up on some feedback for my ideas: -

- As it stands, the Maths department uses plastic models of shapes to teach the topic. So, this relies on passing these models around each class, and there may not always be enough, therefore justifying a computational solution.
- Features including transforming shapes and viewing plans/elevations would be useful, so this will be added to my objectives.
- An implementation of planes of symmetry for each shape and labelling coordinates vertices would be helpful, so I will add this to my extension objectives.

## Prototyping

I have developed a prototype of my program that demonstrates how polygons can be rendered in OpenGL. (Figure 2) Whilst the current build serves little purpose, the backend of the program is well made, robust and can be expanded on with ease, having abstracted OpenGL into multiple classes.

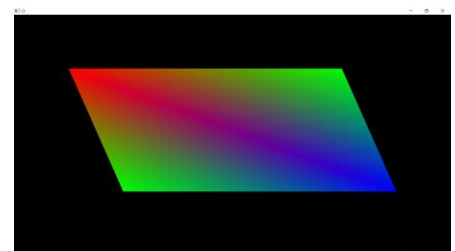


Figure 2 – My first prototype

Currently, the program has the following functions: -

- Renders a 2D quadrilateral displaying a spectrum of colours.
- Can toggle full screen using the F11 key.
- Close the program using either the ESC key or the built-in exit button.

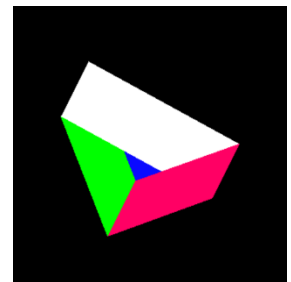


Figure 3 – My Python prototype

Furthermore, I have been using a demo project, made in Python, to fully understand the problem. It demonstrates how 4D matrices can be used in rendering 3D shapes, and how the transformations are carried out. The code below defines an algorithm of how to multiply, two, 4 by 4 matrices. For example, it can be used to multiply the transformation matrix ( $M1$ ) and the coordinates of the shape ( $M2$ ) to find the new coordinates ( $Result$ ).

```
def Multiply(M1,M2):  
    Result = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]  
    for i in range(0,4):  
        for j in range(0,4):  
            Result[i][j] = M1[i][0] * M2[0][j] + M1[i][1] * ...  
            M2[1][j] + M1[i][2] * M2[2][j] + M1[i][3] * M2[3][j]  
    return(Result)
```

The prototype code can be found under [./Jack Doughty 2022 NEA/Prototypes/Prototype.pdf](#)

## Research

Three-dimensional rendering is easier achieved by breaking down the illusion into different parts, such that a given shape can be composed of several two-dimensional polygons that link at vertices to create a network of edges. The surfaces of the object are given by the colour of each polygon. (Figure 3) Therefore, in reference to my program, a newly instantiated 3D shape object could have properties including an array of polygon objects, a colour, an ID or name and potentially a value that defines how matte the surfaces are, as to govern how much light is reflected.

To perform transformations on the shape, the dot product of a given matrix, depending on the type of transformation, and 3-Vector coordinates of each vertex of one of the polygons is calculated to define the vertices of the new shape. However, this only pertains for transformations about the origin, so to successfully transform the shape on the spot, it should first be translated to the origin, then the desired transformation applied, and translated back to the shapes original position.

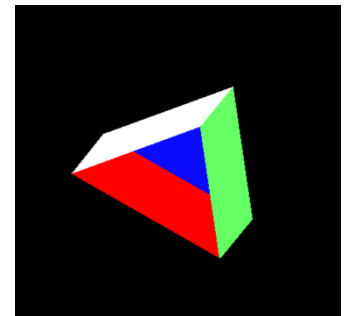
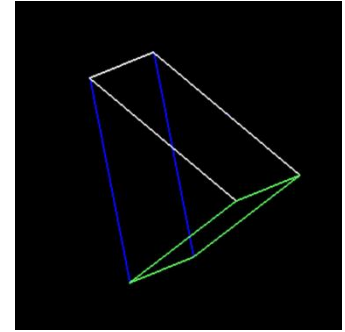


Figure 4 - Triangular prisms consisting of polygons

### Example 1)

To enlarge a vertex at point (1, 2, 3) from the origin, with a scalar ( $\lambda$ ) of 2: -

$$\begin{aligned} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} &= \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \\ &= \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \\ &= \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} \end{aligned}$$

### Example 2)

To rotate a vertex at point (1, 2, 3) around the  $x$  axis, by  $\frac{\pi}{4}$  radians ( $\theta$ ): -

$$\begin{aligned} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \frac{\pi}{4} & -\sin \frac{\pi}{4} \\ 0 & \sin \frac{\pi}{4} & \cos \frac{\pi}{4} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \\ &= \begin{bmatrix} 1 \\ -\frac{\sqrt{2}}{2} \\ \frac{5\sqrt{2}}{2} \end{bmatrix} \end{aligned}$$

However, in most cases, this will not be enough information to render a 3D shape. An extra column and row can be added to the transformation matrix that enables the ability to change perspective, scale the object, rotate, and translate in one operation, therefore increasing overall efficiency and lightening the load on the CPU / GPU, however I will further explain this in my Design section.

### Example 3)

A square has vertices (0, 0, 0), (1, 0, 0), (1, 1, 0) and (0, 1, 0). The shape is to be rotated by  $\frac{\pi}{2}$  radians around the  $x$  axis, on the spot. To do so, the shape must first be translated to the origin, the transformation applied, and the shape returned to its original position as follows.

let  $A \leftarrow$  Translation of the shape to the origin,

let  $B \leftarrow$  Rotation matrix,

let  $C \leftarrow$  Vertex coordinates.

$$C' = ((AB) \cdot A^{-1}) \cdot C$$

$$\begin{aligned}
 & \begin{array}{cccc} \text{Translation to origin} & \text{Rotation by } \frac{\pi}{2} \text{ radians} & \text{Translation from origin} & \text{Original coordinates} \end{array} \\
 & = \left( \left( \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & -1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \frac{\pi}{2} & -\sin \frac{\pi}{4} & 0 \\ 0 & \sin \frac{\pi}{4} & \cos \frac{\pi}{4} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \right) \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \right) \cdot \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \\
 & = \begin{bmatrix} 0 & 2 & 2 & 1 \\ 1 & 2 & 2 & 1 \\ 1 & 1 & 2 & 1 \\ 0 & 1 & 2 & 1 \end{bmatrix}
 \end{aligned}$$

The new coordinates of the square are (0, 2, 2), (1, 2, 2), (1, 1, 2), (0, 1, 2).

A chosen shape will have a colour assigned to it – however, if all the sides of the object are the same shade, the user will not be able to differentiate one side from another so it will not be clear which direction the shape is facing. This can be patched by implementing lighting algorithms. One of which, diffuse lighting, calculates the intensity of light that reflects off each side of the object and reaches the camera depending on the position of the light source. The intensity of light is inversely proportional to the angle between the light source and the normal to the incident surface. (Figure 4) I will again refer to this in my Design section.

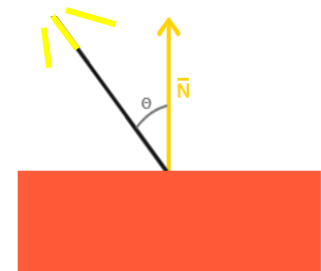


Figure 5 - Ray casting  
(learnopengl.com)

# Related Existing Solutions

Inspiration can be taken from existing services, such as Microsoft 3D Viewer, Blender, Google SketchUp, Unreal Engine and Paint 3D.

**All Programs** – Can rotate and enlarge objects in an environment with a light source.

**GeoGebra** – This is a website that functions as a graphical calculator. It has a set of axes that rotate depending on the position of the camera and the ability to plot functions.

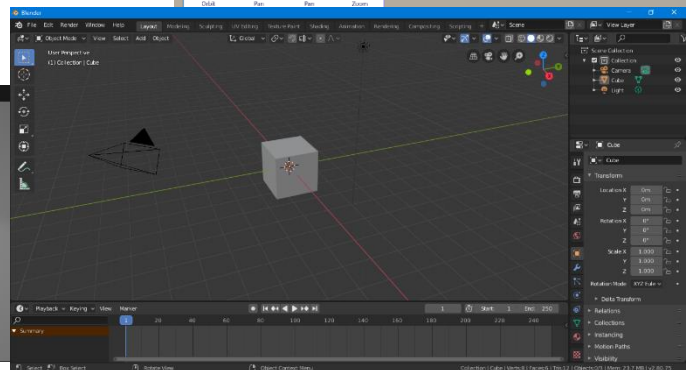
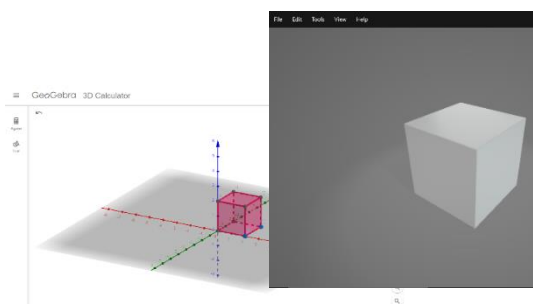
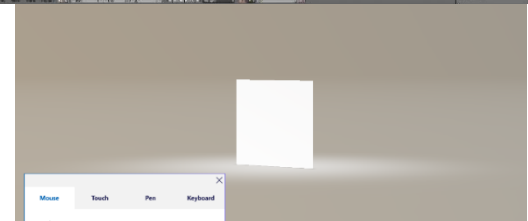
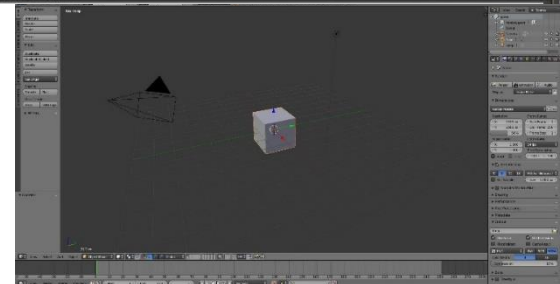
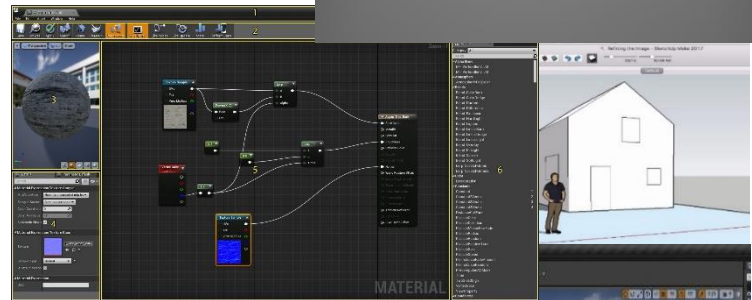
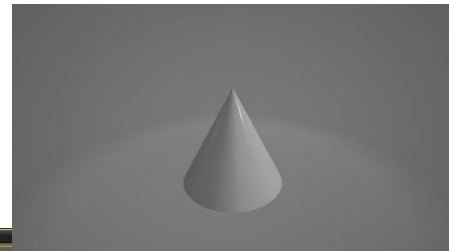
**3D Viewer** – This is a program solely for viewing 3D objects and files. It has very responsive and appropriate controls and a concise GUI.

**SketchUp** – This is a program built for Computer Aided Design (CAD). Has a clear GUI, most notably the menu to the left.

**Unreal Engine** – This is a game engine; designed for the development of video games. It features a 3D environment that allows you to add objects from file into the scene and alter their properties. It also showcases an in-depth material editor, allowing you to change all the properties of a given surface, such as texture, amount of reflected light and tessellation.

**Paint 3D** – This software is for drawing raster graphics and 3D modelling. It is similar to 3D viewer except it allowed the amending of objects. It, too, has practical controls and a simple GUI.

**Blender** – Blender is freeware for 3D modelling, animation, simulation, motion tracking and many other purposes. It features a set of axes, lighting, gyroscope, and controls that are easy to pick up.



# Objectives

It is my intention that the application should;

- Be of use to students and teachers, relevant to their curricula as specified prior. The application should allow the transformation of a 3D object, allowing the counting of edges, faces and vertices in a maths lesson.
- Be able to generate and render 3D objects in real time. Users should be able to see immediately the effect of manipulating the object or making changes to the environment.
- Be able to manipulate each object e.g. Rotate using mouse controls, change colour using a colour wheel and zoom in/out.
- Use perspective to give an accurate simulation of a real life object as it is transformed, allowing the user to recognise near and far edges or faces.
- Be able to accurately simulate a light source, and how it behaves in a 3D environment with the chosen object and its material properties. Also be able to specify the location and quality of the light source (such as brightness).
- Provide an intuitive and responsive Graphical User Interface (GUI), drawing features from similar programs. Assessing this objective will require some feedback from prospective users.
- Exhibit robust Object-Oriented Programming (OOP) and smooth/consistent performance. The application should be robust and function without error, or ceasing to function due to run time errors.
- Exhibit complex algorithms such as ray casting, projection, and linear / matrix transformations.

## Extension

- Implement specular lighting.
- Implement planes of symmetry and an algorithm to calculate where they should be displayed. This will be useful in teaching planes of symmetry to GCSE classes.
- Label coordinates for each vertex. Again, to demonstrate how coordinates change after rotation.
- Import custom shapes from file or other methods. This will allow the user to render and manipulate other non standard shapes.
- Allow graphing including generating volumes of revolution and elongating a graph into a prism.
- Add an undo function to allow the user to return to the previous position after a rotation or enlargement. This will involve the use of a stack data structure.

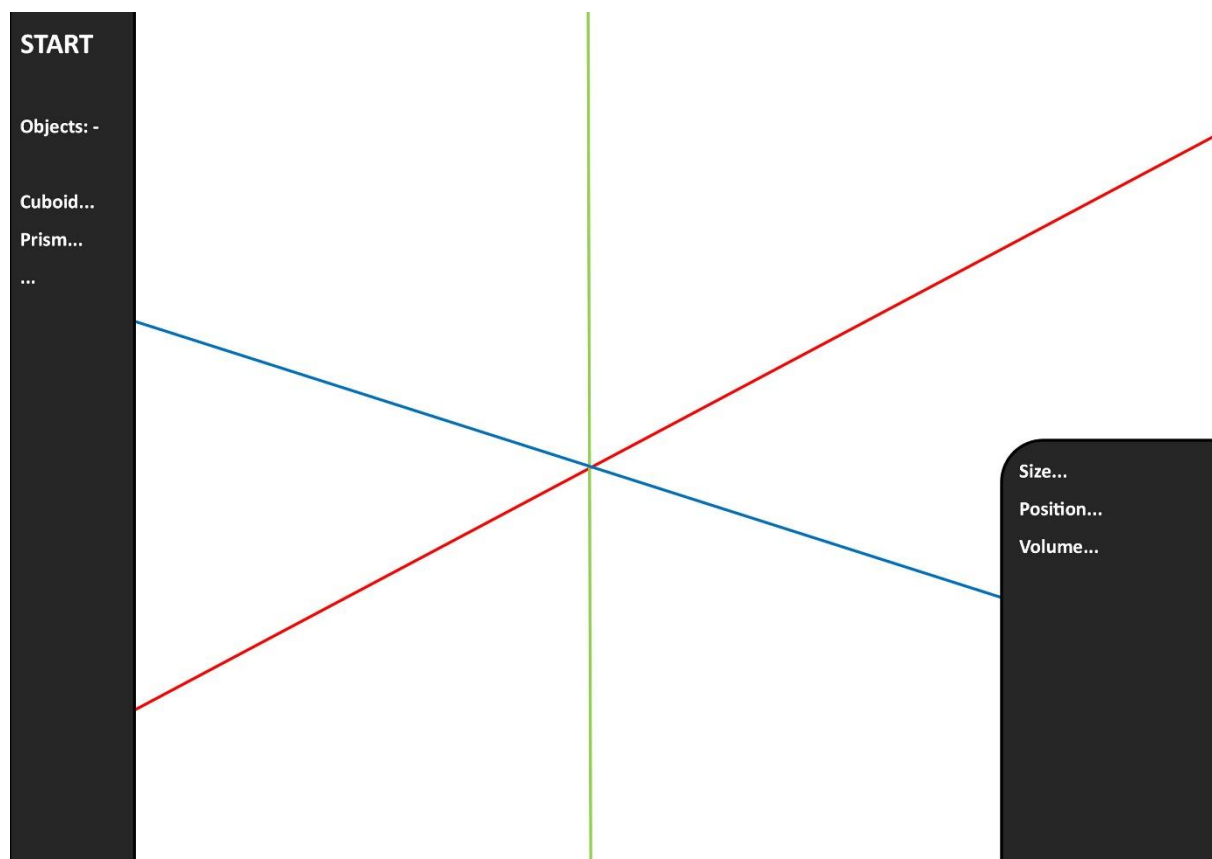


# Design

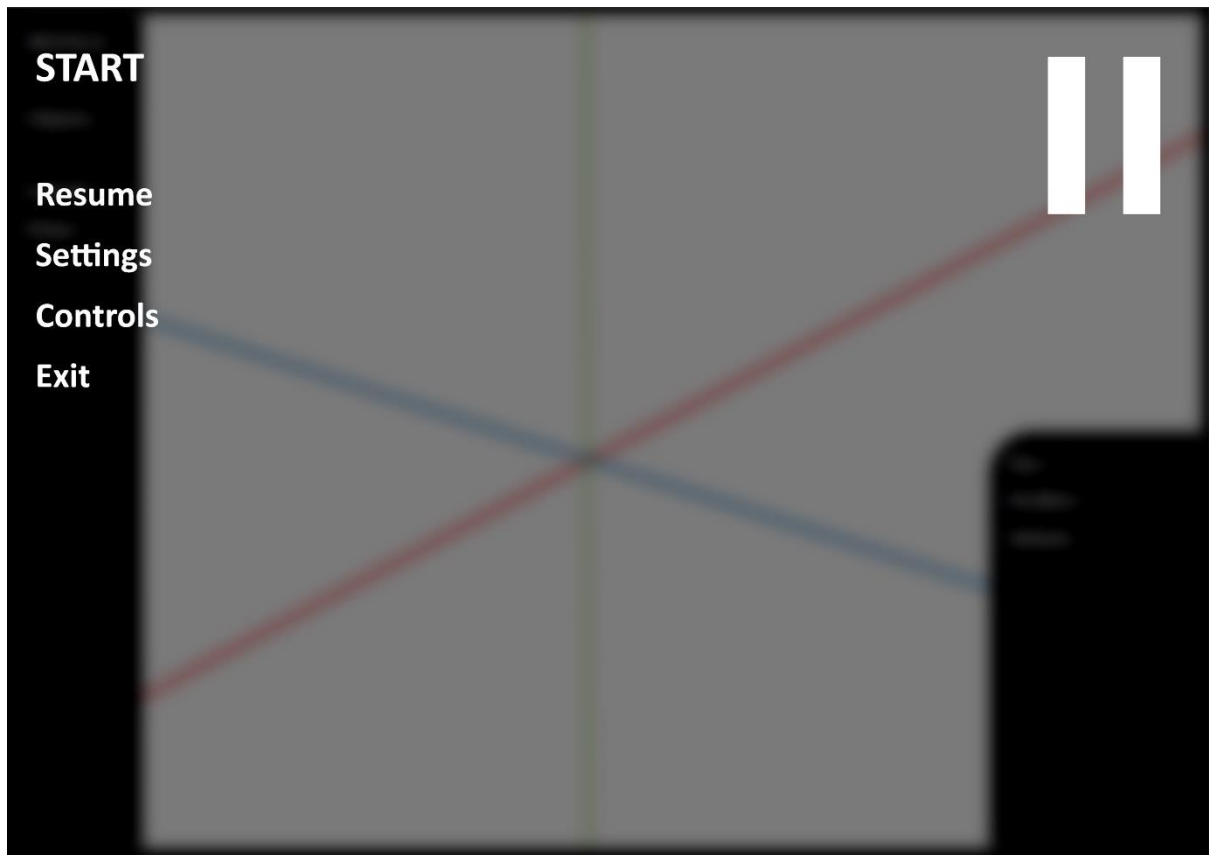
## User Interface Specification

*Fitts's law is a predictive model of human movement primarily used in human–computer interaction and ergonomics. This scientific law predicts that the time required to rapidly move to a target area is a function of the ratio between the distance to the target and the width of the target.* (Wikipedia)

Keeping Fitts's Law, similar programs, and my objectives in mind, I have designed example GUIs for my software, although my final designs and implementations will likely differ.

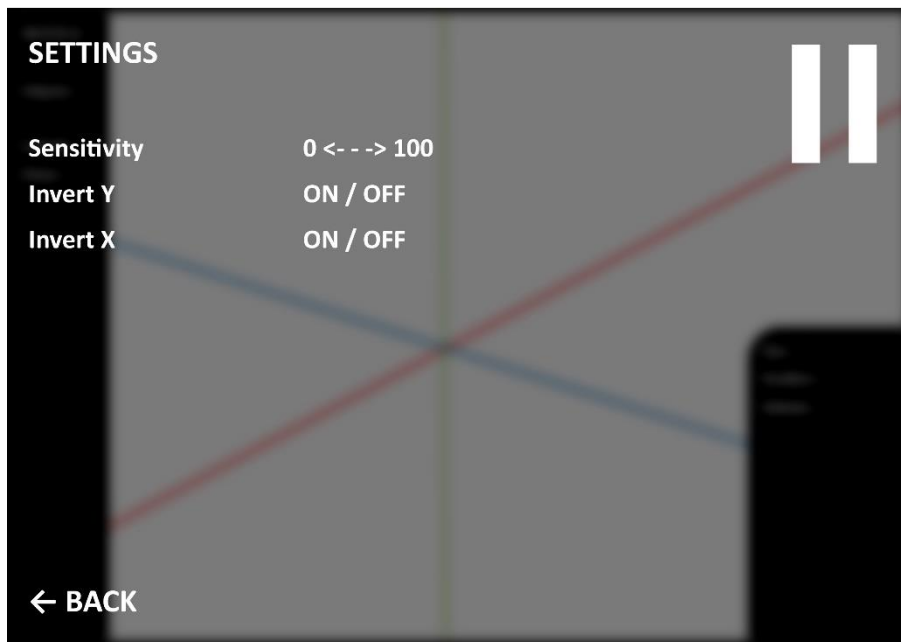


This could be the layout of the main screen of the program. Of all the screens, this will have the most use. It displays the shape, axes, a list of shapes, shape properties and allows the user to perform any supported action on the object and access all other screens. The rounded edges and dark surfaces are easier on the eyes whilst providing contrast to the brightly lit environment; ensuring that the user can differentiate between the elements onscreen. I may later wish to implement previews or icons for each shape, along with their respective names, to make them easier identifiable.



The Start screen, or Start menu, can either be accessed from pressing the ESC key, or by pressing the START button on the main screen. It will grant access to the two other screens, Settings and Controls, and allow the user to close the application or return to the main screen. It will also cause all keystrokes, that would previously affect the shape, to be ineffective, and the previous GUI to be dimmed and blurred, as to give the impression that the program has been paused. The pause symbol, whilst being the largest target onscreen, will serve no computational purpose other than reassuring the user that the program has been paused.

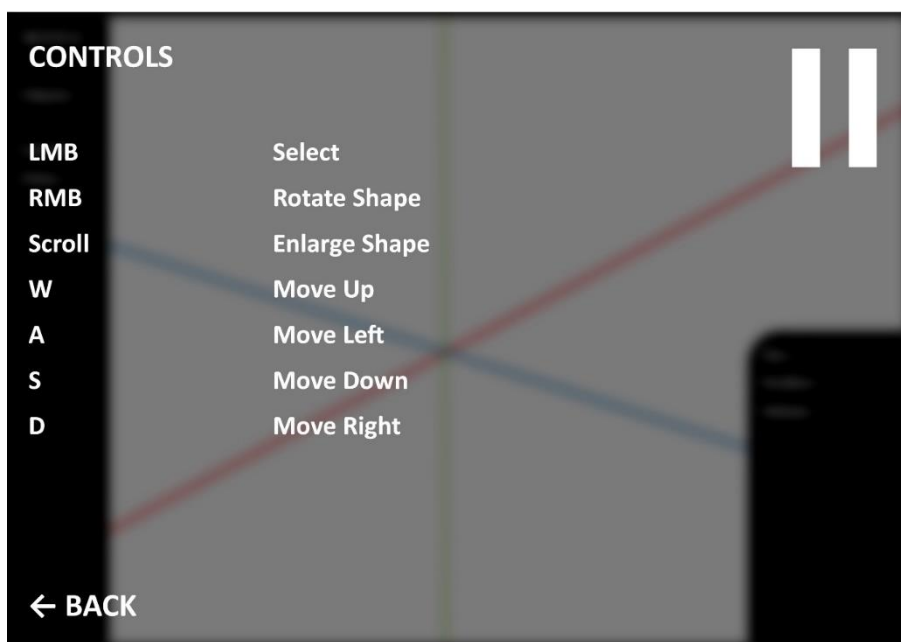
An issue I may run into is that a blurred background could be difficult to implement as, natively, OpenGL does not have GUI support, so I will need an additional module to achieve this, which has no guarantee of supporting a blur effect. As an alternative to this, instead of implementing a pause menu, I place the Settings, Controls and Exit buttons on the main screen, possibly at the bottom-left or top-left hand corner as to not be too much of an obstruction onscreen.



- + Mouse acceleration
- + Keyboard sensitivity
- + Keyboard acceleration
- + Shape rotary acceleration

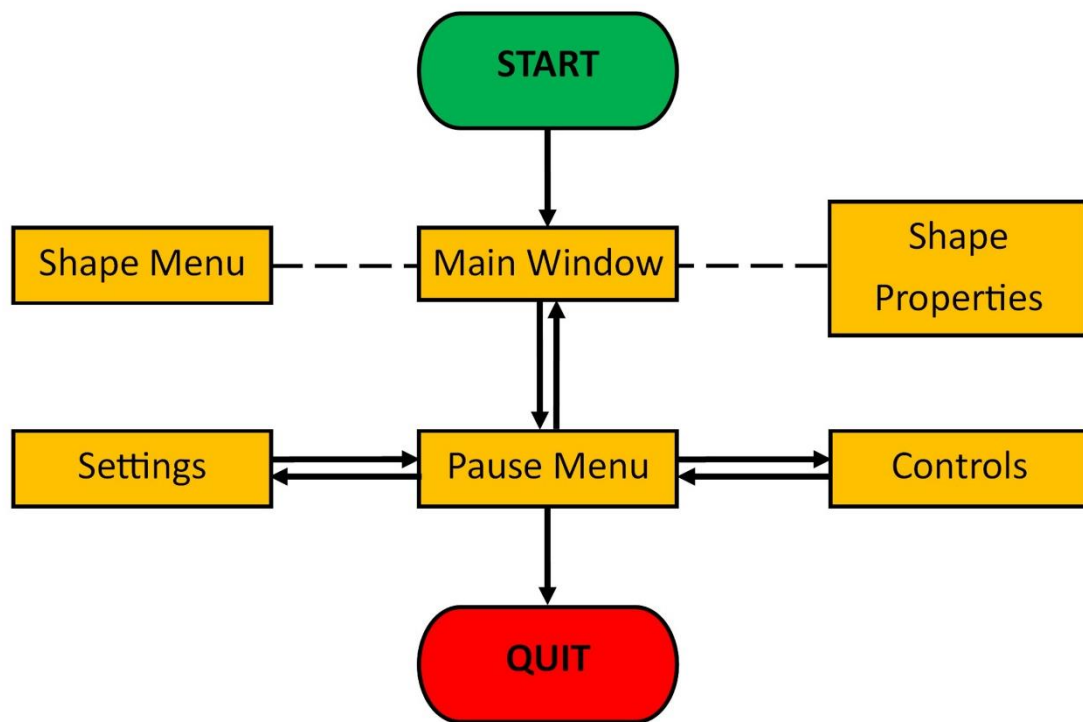
Figure 1

The Settings screen will allow the user to make changes to the program depending on how they're own personal preferences. As it stands there are three settings that can be changed, the sensitivity of the mouse, inverting all movements made by the user in the X direction and in the Y direction. However I provided a small list of additional settings I may still implement. (Figure 1)



The Controls menu, being aesthetically similar to the Start menu, serves little purpose other than displaying the controls to interact with the software. I will not be implementing a way to change the controls as deciding on the best controls to manipulate a 3D object is often convoluted, as the shape and camera can be transformed in more ways than one, and the user experience should be developed to be intuitive and linear as described in my objectives.

The program can be broken down into a flow-diagram to show how the user can navigate the GUIs. The dashed lines indicate that the two conjoined elements are part of the same screen but are separated from one another.



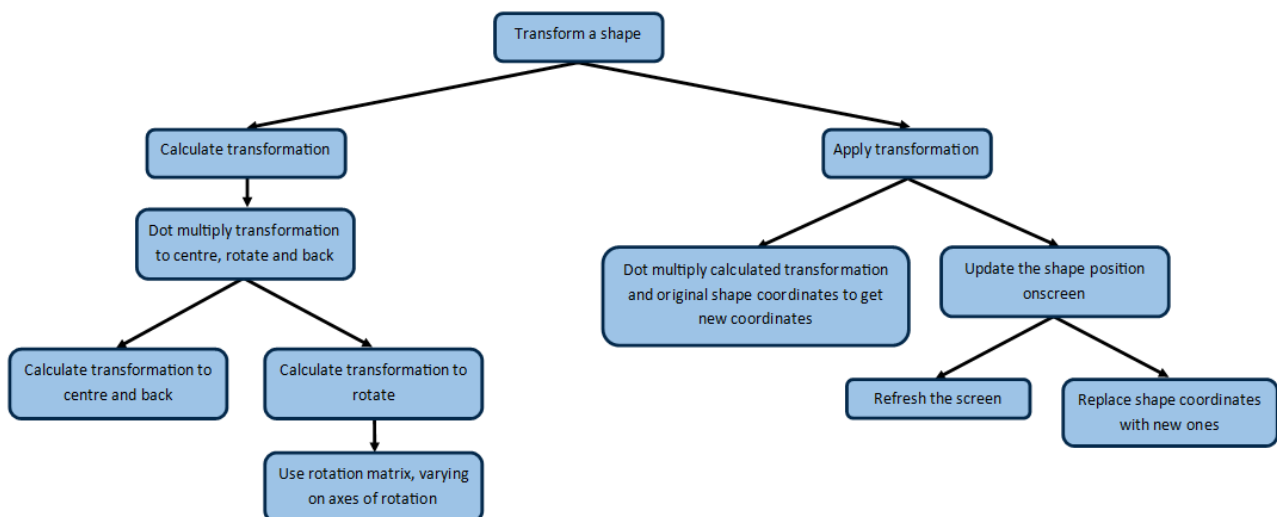
# Algorithms

## Primary Graphics Pipeline

The main priority in the development of this solution will be the rendering and transforming of 3D objects. This process can be written as an input, process, and an output.



This process can further be decomposed into smaller problems to individually program, making development, overall, easier. However, this can be separated into two categories, calculating and applying the transformation.



Calculating the transformation should require two subroutines, one which finds which axis to rotate the shape in from which button was pressed, and one which dot multiplies two 4x4 matrices. Applying the transformation should require three subroutines, however, it will share the dot multiplication subroutine with the other category. The last two subroutines will be to change the shape properties, including its coordinates, and to refresh the screen after this has been done. Furthermore, if the shape is situated at (0, 0, 0), then it will not have to be transformed before and after being rotated, lowering the time required to complete the entire transformation.

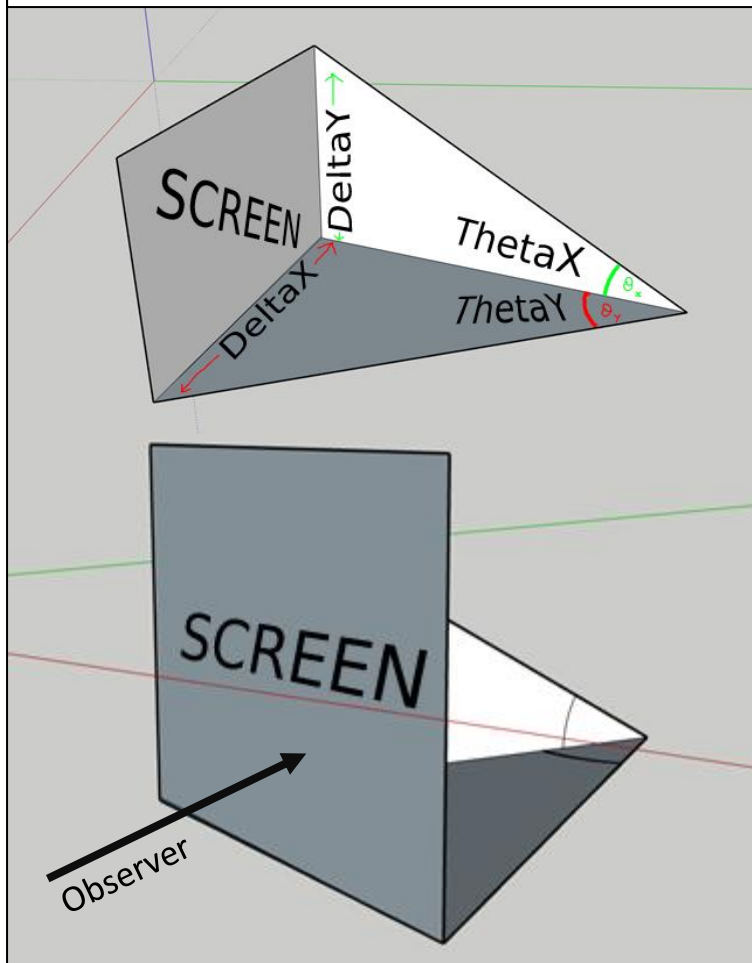
## Mouse Input

Another frequently used algorithm is the method of taking user mouse input and controlling the circular motion of the shape, such that it gradually slows down after the user releases the shape after a rotation. I have put together an example algorithm to show this-

```
MainLoop() {  
    AccelerationFlag = true;  
    Mouse.Sensitivity = 250;  
    t = 10;  
    if (keyPressed(LEFT_MOUSE_BUTTON)) { // 1  
        Mouse.lastMousePositionX = Mouse.MousePositionX;  
        Mouse.lastMousePositionY = Mouse.MousePositionY;  
    }  
    else if (keyDown(LEFT_MOUSE_BUTTON)) { // 2  
        DeltaX = Mouse.MousePositionX - Mouse.lastMousePositionX;  
        DeltaY = Mouse.MousePositionY - Mouse.lastMousePositionY;  
        DeltaZ = Mouse.Sensitivity * abs(Shape.PositionZ - Camera.PositionZ);  
        // See Testing section to see why this is a bug in some scenarios  
        // 3  
        ThetaX = atan(DeltaY / DeltaZ); // 4  
        ThetaY = atan(DeltaX / DeltaZ);  
    };  
    Shape.Model = RotationMat(mat4(1.0f), ThetaX, vec3(1.0f, 0.0f, 0.0f)); // 5  
    Shape.Model = RotationMat(Shape.Model, ThetaY, vec3(0.0f, 1.0f, 0.0f));  
  
    if (AccelerationFlag) {  
        DecelerationX = -ThetaX / t;  
        DecelerationY = -ThetaY / t;  
        ThetaX += DecelerationX;  
        ThetaY += DecelerationY;  
    }  
    else {  
        ThetaX = ThetaY = 0;  
    }  
    Mouse.lastMousePositionX = Mouse.MousePositionX;  
    Mouse.lastMousePositionY = Mouse.MousePositionY;  
    Shader.setMat4("model", Shape.Model); // 6  
};  
  
Shader.setMat4("model", Shape.Model);
```

DeltaX / DeltaY – Change in mouse position between frames, and while mouse button down.

ThetaX / ThetaY – Angles of rotation.



### Pre-Defined Functions

- 1) 'keyPressed' checks if the provided key has been pressed previously.
- 2) 'keyDown' checks if the provided key is currently being held down.
- 3) 'abs' takes the absolute value of the given value.
- 4) 'RotationMat' creates a 4x4 matrix using the angle and axes of rotation provided.
- 5) 'atan' takes the inverse tangent of the value provided.
- 6) 'Shader.setMat4' gives the renderer the transformation matrix that was previously calculated.

*Figure 2 – Algorithm and diagram depicting how angles of rotation for the shape are calculated from mouse movement.*

### Further Transformations

Commonly in computer graphics, 4-Variable, homogenous, coordinates are used to more efficiently store points and vectors. What homogeneous coordinates do is give points and vectors the same form by adding an extra coordinate, homogenising them. Most notably, it allows matrices to now operate on points, so the same type of object can be sent through the graphics pipeline, streamlining the process. This fourth coordinate is given by a variable  $w$ , and if  $w = 1$ , the point or vector is said to be normalised. I will be designing my own Matrix and Vector classes.

In addition to rotations, there are other transformations that are useful in the development of my software. One of such causes a scaling effect, in that a scalar is applied to each coordinate per each vertex-

Where  $\lambda_x, \lambda_y, \lambda_z$  are scalars,

$$\begin{bmatrix} \lambda_x & 0 & 0 & 0 \\ 0 & \lambda_y & 0 & 0 \\ 0 & 0 & \lambda_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} \lambda_x x \\ \lambda_y y \\ \lambda_z z \\ w \end{bmatrix}$$

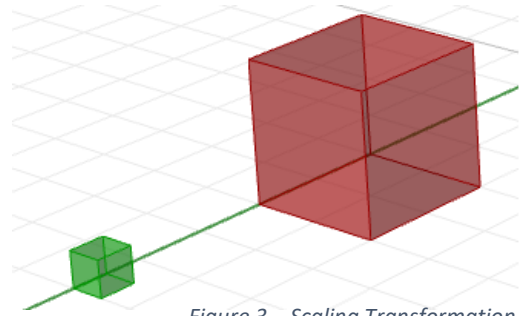


Figure 3 – Scaling Transformation

Furthermore, I make use of perspective transformations applied to the world space, such that objects further away appear smaller. The way this is done is a matrix is applied to each vertex of each shape, such that the cube world space is mapped to a frustum. This row-major matrix transformation is formed as follows-

Where the following variables define the dimensions of a frustum,

$r \leftarrow \text{right } (x)$   
 $b \leftarrow \text{bottom } (y)$   
 $l \leftarrow \text{left } (x)$   
 $n \leftarrow \text{near } (z)$   
 $t \leftarrow \text{top } (y)$   
 $f \leftarrow \text{far } (z)$

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ \frac{r+l}{r-l} & \frac{t+b}{t-b} & -\frac{f+n}{f-n} & -1 \\ 0 & 0 & -\frac{2fn}{f-n} & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

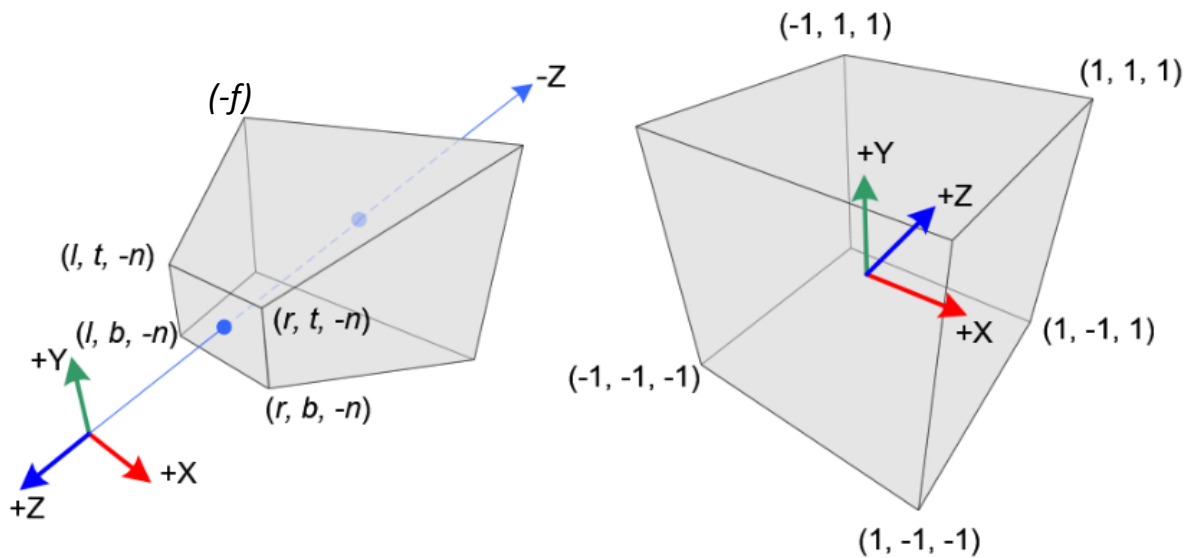


Figure 4 – Perspective Transformation (www.songho.ca)



The final version of the rotation algorithm is slightly different to the one I have already specified in this write-up. This is because the shape must be rotated in both the x and y axes simultaneously. However, this cannot be done with one rotation matrix that I have shown earlier, there must be two rotations done successively, with different angles. The way I go about this is through effectively using the Euler-Rodrigues Formula, such that a transformation from one 3D point to another can be described with an angle and a unit vector. The matrix is constructed by completing rotation transformations in the three axes with angles varying on the unit vector, or axes of rotation. I will demonstrate how this works below.

Given a rotation angle  $\theta$  and axes of rotation  $\hat{k} = \begin{bmatrix} k_x \\ k_y \\ k_z \end{bmatrix}$ , a transformation matrix  $M$  is defined as follows.

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(k_x\theta) & -\sin(k_x\theta) & 0 \\ 0 & \sin(k_x\theta) & \cos(k_x\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(k_y\theta) & 0 & -\sin(k_y\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(k_y\theta) & 0 & \cos(k_y\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(k_z\theta) & -\sin(k_z\theta) & 0 & 0 \\ -\sin(k_z\theta) & \cos(k_z\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(k_y\theta)\cos(k_z\theta) & -\cos(k_y\theta)\sin(k_z\theta) & -\sin(k_y\theta) & 0 \\ \sin(k_x\theta)\sin(k_y\theta)\cos(k_z\theta) + \sin(k_x\theta)\sin(k_z\theta) & \cos(k_x\theta)\cos(k_z\theta) - \sin(k_x\theta)\sin(k_y\theta)\sin(k_z\theta) & \sin(k_x\theta)\cos(k_y\theta) & 0 \\ \cos(k_x\theta)\sin(k_y\theta)\cos(k_z\theta) - \cos(k_x\theta)\sin(k_z\theta) & -\sin(k_x\theta)\cos(k_z\theta) - \cos(k_x\theta)\sin(k_y\theta)\sin(k_z\theta) & \cos(k_x\theta)\cos(k_y\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This calculation can be seen in my programming for the Matrix class.

## Lighting

As mentioned prior, a light source must be employed in order for the user to recognise the object on screen as a 3D shape. In my software I have implemented an algorithm, Phong Shading, to accurately model how lighting would react with a shape of varying colour and material. The Phong Lighting Model consists of three components, Ambient, Diffuse and Specular lighting. Below I will describe and explain each of the three individual models.

Ambient lighting is the simplest of the three models. It requires a value, less than one and greater than zero, to scale the whole colour of the shape. In effect, this value represents the intensity of the light source. (Figure 5)

```
float Ambient = 0.5;
```

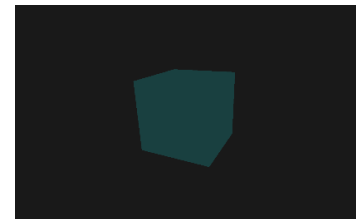


Figure 5 – Ambient Lighting

Diffuse lighting takes three parameters, the position of the light source, the exact position of a fragment or pixel on the shape after undergoing the desired transformation, and the normal vector to the plane belonging to that fragment. It will calculate the normalised direction vector from the fragment position to the light source, then take the dot product of

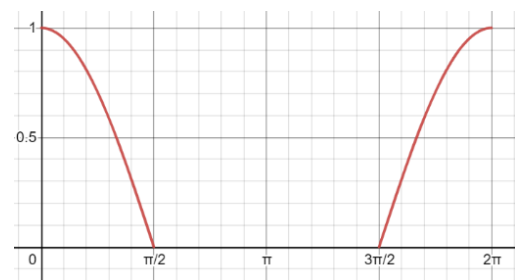


Figure 6 – Graph of  $\max(\cos\theta, 0)$

this vector and the normal vector, to find the cosine of the angle subtended by the two vectors. The aim is to calculate an intensity for light on that particular fragment, so the co-domain cosine of the angle, -1 to 1, must be changed to 0 to 1, which can be done using the builtin max() function (Figure 6), giving the cosine of the angle and 0 as arguments, such that if the cosine of the angle is less than one, the intensity will take a value of 0. This means that if the angle subtended between the two vectors is 90°, there will be no light on that fragment. (Figure 7)

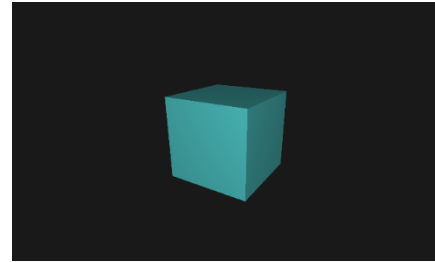


Figure 7 – Diffuse Lighting

```
vec3 lightDirection = normalize(lightPosition - fragmentPosition);
float Diffuse = max(dot(Normal, lightDirection), 0.0);
```

Specular lighting requires a further three parameters compared to diffuse lighting, the position of the observer or camera, a value for how reflective the shape surface

```
//Where S = Shininess value and R = reflectiveness value

vec3 lightDirection = normalize(lightPosition - fragmentPosition);
vec3 viewDirection = normalize(viewPosition - fragmentPosition);
vec3 reflectDirection = reflect(-lightDirection, Normal);
float Specular = R * pow(max(dot(viewDirection, reflectDirection), 0.0), S);
```

is, and a shininess value of the highlight. The method for finding a specular value is through calculating the normalised direction vector from the light source to the fragment position, then substituting this and the normal vector into a builtin function, reflect(), which, for a given incident vector and surface normal, returns the reflection direction. Calculating the dot product of this vector and the normalised direction vector from the fragment position to the observer, gives the cosine of the angle subtended between the two vectors. Taking the maximum value between this and 0, taking it to the power of the shininess value, and scaling it by the reflectiveness value gives the specular value.

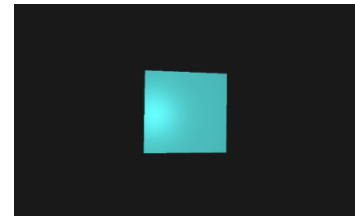
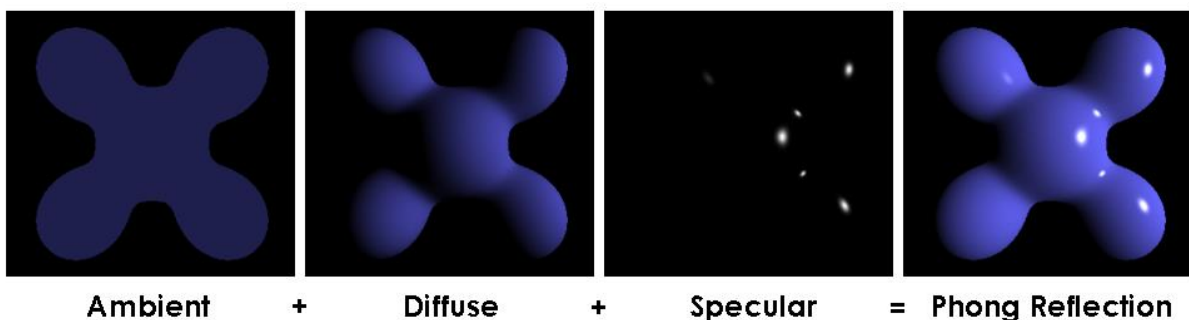


Figure 8 – Phong Lighting

Combining all three models, provided the objects' colour and intensity of light, gives a realistic simulation for how a light source would behave with a 3D object of varying material, colour, position and dimensions, the Phong Reflection Model. (Figure 8)



By Brad Smith - Own work, CC BY-SA 3.0

# Data Structures

In developing a 3D renderer, various data types can become heavily depended on. Here, I will summarise a few such structures.

## Pointers

C++, unlike most other languages, offers a method of getting hold of the memory address of a given value. For example, the program in Figure 3 would return a value in a hexadecimal format akin to '0x0000'.

```
string coolperson = "jack";  
return &coolperson;
```

Figure 3

To further this, a pointer is a data structure that stores the memory address to another piece of data, and can be declared as shown in Figure 4. They prove to be valuable when requiring a function or method to return more than one variable – something that was previously not possible in base C++ (See Figure 5.1).

```
string coolperson = "jack";  
string* coolpointer = &coolperson;  
  
// coolpointer = 0x0000  
// *coolpointer = "jack"
```

Figure 4

Overall, pointers and references are used to dynamically allocate, de-allocate and manipulate data within main memory, which often reduces the time complexity and improves the performance of code. For example, performance can be improved through passing variables by reference (See Figure 5.2), because all other common data structures allocate more data space than a single memory address.

```
void swap(int* a, int* b) {  
    int temp = *a;  
    *a = *b; [Figure 5.1]  
    *b = temp;  
};  
  
int one = 1;  
int two = 2;  
  
swap(&one, &two); [Figure 5.2]  
// one = 2, two = 1
```

Figure 5

## Vectors and Matrices

A vector is a geometric object with both magnitude and direction. Respective to the purposes of my program, they are most commonly used as coordinates, but also represent light rays and colours. Natively, C++ does not treat them as data types, but with either a class definition or library they can be implemented. I will outline a possible class definition for a vector with 3 float components. (Figure 7)

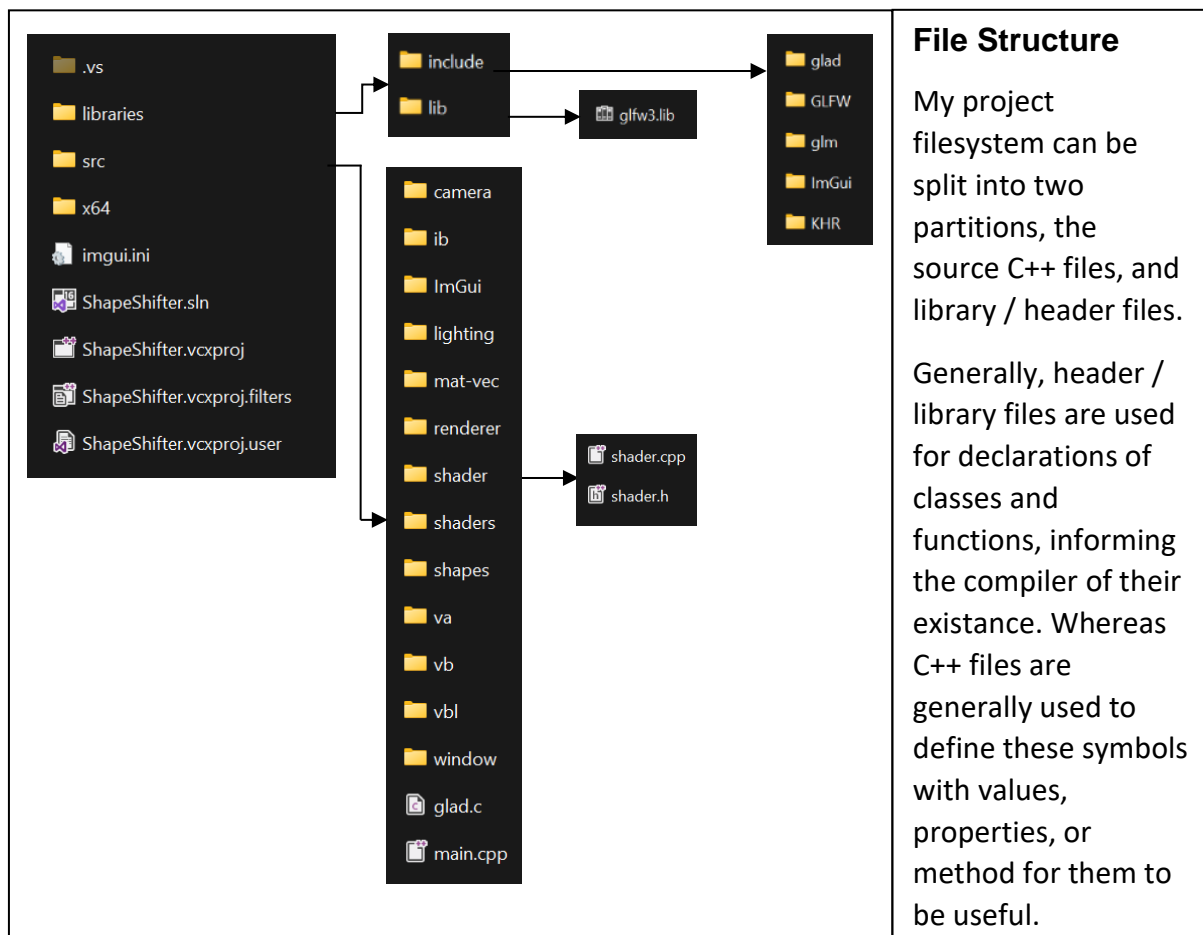
Similar to vectors, matrices can be used to represent geometric objects and transformations and are not natively supported by C++. To save time and resources vectors can in fact be implemented as a polymorphic class from the matrix class, in that an n-vector is a  $n * 1$  matrix. (See Figure 6 for possible class definition)

<pre> class Mat3f { public:     Mat3f (float[3][3]);     Vec3f multiplyVec(Vec3f);     Mat3f transposeMat();     Mat3f invertMat();     Mat3f rotateMat(float, Vec3f);  private:     float m_Self[3][3]; }; </pre>	<pre> class Vec3f : Mat3f { public:     Vec3f(float[3]);     Vec3f normaliseVec();     Vec3f scaleVec(float);     float getMagnitude();     float dotMultiply(Vecf);  private:     float m_Self[3]; }; </pre>
--	---

Figure 6

Figure 7

## File Organisation



While my software does not require a specific file structure to run – it is important to keep it in an ordered manner to make including new files easier, such that it becomes a more streamlined process for the programmer, while rendering the project easier understandable for a potential client or user, after downloading the software.

# Technical Solution

[See ShapeShifter.pdf](#)

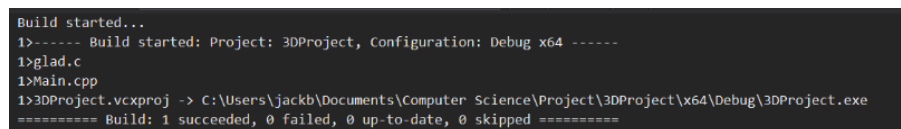
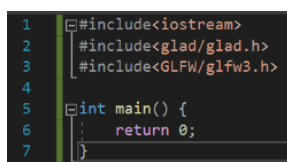
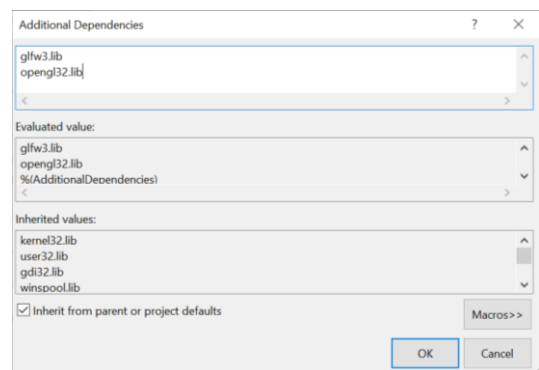
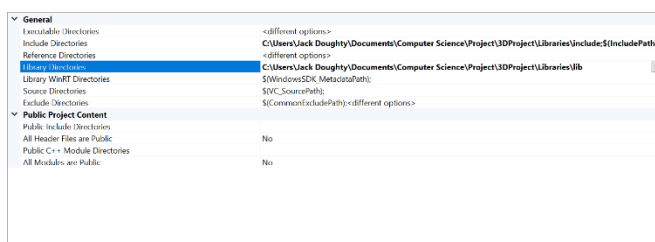
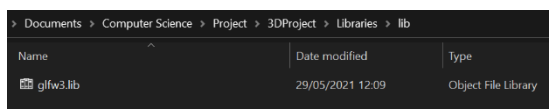
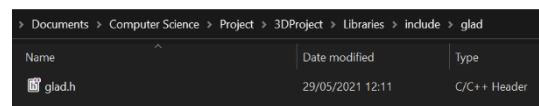
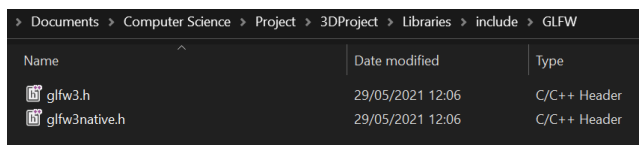
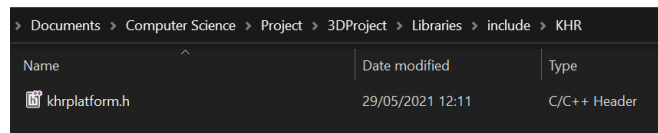
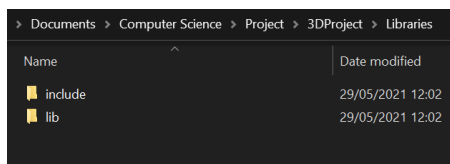
If the link is not working see "Jack Doughty 2022  
NEA\Documentation\ShapeShifter.pdf".

# Testing and Development

## Tests Done in Development

### 1<sup>st</sup> Test – Project Setup

I began with creating a libraries and include folder for my project, copying in the first required modules. I then used the Visual Studio Linker to include them in my project, specifying their paths relative to the project solution. I then wrote a basic project that included the modules and built it. I had expected the program to execute, not display anything, then close, as there was no previous errors being highlighted to me and I had provided no code to create a window. This turned out to be correct, if anything in the process of linking the dependancies had gone wrong, the program would not have built and executed.



## 2<sup>nd</sup> Test – Creation of a Window class

I noticed that after developing a lot of the core program that I was referencing the OpenGL window object, and values relating to it, often, and this code either took up a lot of room, or was repeating itself. So I chose to abstract the OpenGL window to make my own class, defining only the methods and properties I needed. I expected the program to build and execute, displaying a window with a blue background. The screenshot I provide of the window proves I was successful in this.

```
int main() {  
    //initialise GLFW  
    glfwInit();  
  
    //specify which version of OpenGL to use (3.3)  
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);  
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);  
    //using the core profile  
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);  
  
    //create a window, and setting 'D' as its caption  
    GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, ":D", NULL, NULL);  
  
    if (window == NULL) {  
        //Error catching  
        std::cout << "Failed to open window";  
        return -1;  
    }  
  
    //assign the window to the current context  
    glfwMakeContextCurrent(window);  
  
    glfwSetKeyCallback(window, KEY_CALLBACK);  
  
    //load GLAD to configure OpenGL  
    gladLoadGL();  
  
    //set up the viewport such that 0 >= x >= WIDTH and 0 >= y >= HEIGHT  
    glViewport(0, 0, WIDTH, HEIGHT);  
}
```

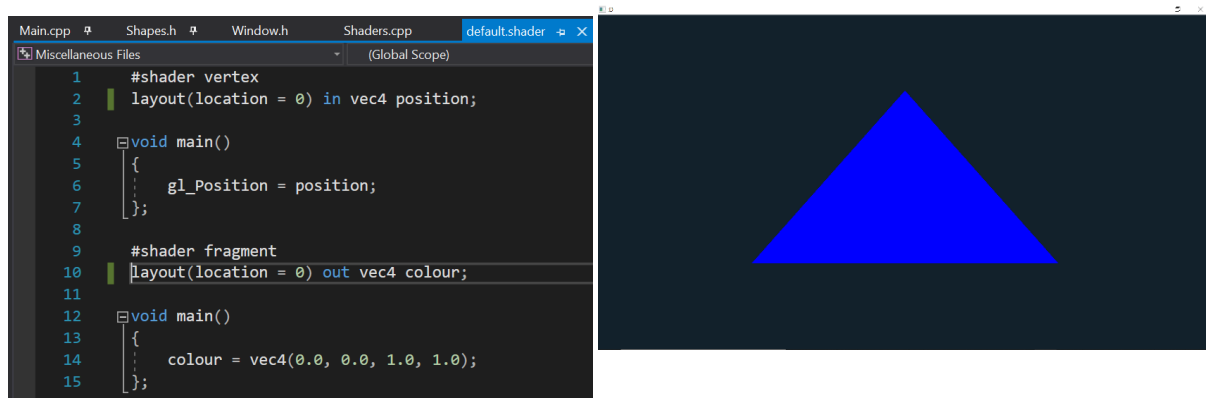
```
1 #include<GLFW/glfw3.h>  
2  
3 class Window {  
4 private:  
5     GLFWwindow* self;  
6     unsigned int WindowWidth;  
7     unsigned int WindowHeight;  
8     unsigned int MaxWindowWidth;  
9     unsigned int MaxWindowHeight;  
10    const char * WindowTitle;  
11    bool Fullscreen = false;  
12    GLFWmonitor* Monitor;  
13  
14 public:  
15  
16    void WindowInit(int Width, int Height, const char* Title, GLFWmonitor* Mon) {  
17  
18        WindowWidth = Width;  
19        WindowHeight = Height;  
20        WindowTitle = Title;  
21        Monitor = Mon;  
22  
23        self = glfwCreateWindow(Width, Height, Title, NULL, NULL);  
24        const GLFWvidmode* mode = glfwGetVideoMode(Monitor);  
25        MaxWindowWidth = mode->width;  
26        MaxWindowHeight = mode->height;  
27        MaximiseWindow();  
28    };  
29  
30  
31    void MaximiseWindow() {  
32        glfwMaximizeWindow(self);  
33        WindowWidth = MaxWindowWidth;
```

```
28  
29    };  
30  
31    void MaximiseWindow() {  
32        glfwMaximizeWindow(self);  
33        WindowWidth = MaxWindowWidth;  
34        WindowHeight = MaxWindowHeight;  
35    };  
36    void CloseWindow() {  
37        glfwWindowShouldClose(self);  
38    };  
39    void SetWindowTitle(const char * string) {  
40        WindowInit(WindowWidth, WindowHeight, string, Monitor);  
41    }  
42  
43    unsigned int GetWindowWidth() {  
44        return WindowWidth;  
45    };  
46    unsigned int GetWindowHeight() {  
47        return WindowHeight;  
48    };  
49    void ToggleFullscreen() {  
50        glfwSetWindowMonitor(self, Fullscreen ? NULL : Monitor, 0, 0, Fullscreen ? WindowWidth : MaxWindowWidth, Fullscreen ? WindowHeight : MaxWindowHeight, -1);  
51        Fullscreen = !Fullscreen;  
52    };  
53    GLFWwindow* GetWindowInstance() {  
54        return self;  
55    };  
56  
57
```

## 3<sup>rd</sup> Test – Hello Triangle!

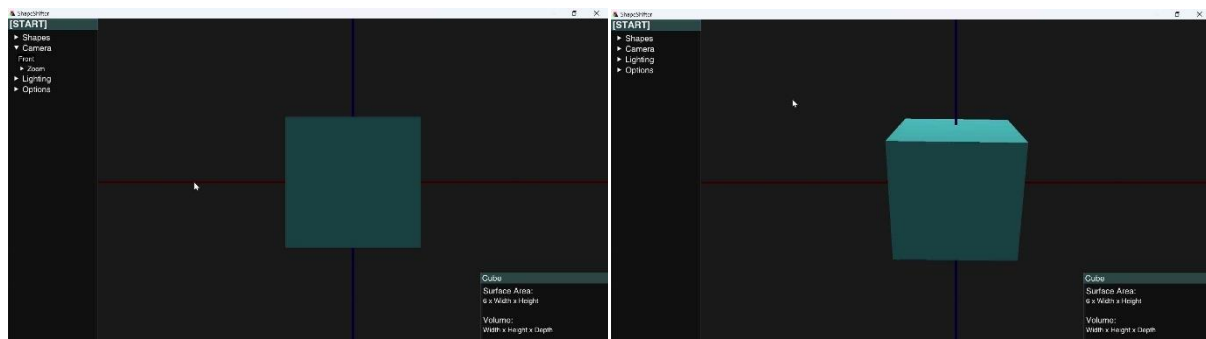
One of the first tasks to do in graphical programming, with OpenGL especially, is build a 'Hello Triangle' program. Similar to that of a 'Hello World' program when we first learn to program. What this program does is display a triangle onscreen, having written algorithms to, for example, take in a shader, coordinates and an index list. I expected this program to do just that, in a window with a blue background. This is

shown in provided screenshot. The image beside this shows the vertex and fragment shader used to render the triangle.



#### 4<sup>th</sup> Test – Testing Motion of Shape

I tested the motion of the shape under mouse movement to see if it visibly behaved correctly. What I expect to happen is, if the mouse is moved up, the shape would rotate up, in the x-axis, and if the mouse is moved right, the shape would move right, in the y-axis. At first, everything was as I expected, my expectations being true. However, after changing the camera perspective it begins to move in a different manner. I will explain this issue further in the 'Current Issues' section, but it meant that from facing the shape from behind, and moving the mouse upwards, the shape would rotate downwards in the x-axis, as shown in the following images.



## Issues in Development

- The first larger Issue I encountered was when passing data to an OpenGL buffer, using `glBufferData`. To calculate the size of the buffer I had to work out the size of a float, multiplied by how many there are. However these two values were of different data types, so they occupied different amounts of memory. The first was a 4 bit number and the other an 8 bit number. By multiplying the first by the second caused an overflow as the result, an 8 bit number atleast, had to be stored in a 4 bit space which was not possible. This meant that the function was unable to run, and no data could be passed to OpenGL. This was fixed by casting the 1<sup>st</sup> number to a data type with a size of 8 bits or greater.



- A small problem, which however took me an unreasonable amount of time to find and fix, was with passing values as uniforms to the shader. The OpenGL function to pass uniforms to the shader requires the value as a 'constant pointer'; a pointer, as mentioned earlier, that points to a constant value in memory. The function wrapper that I had defined for this function, did not have parameters of this type, so values were being passed to the wrapper which did not match the data type of the OpenGL function, so the uniform was not passed. What made this problem especially hard is that in this case it meant that no error was given to me, and the screen was blank, so it required me to search through my code and make changes as a method of trial and error until I found the issue.

- The next major issue I had was when calculating the transformations for the shape. OpenGL interprets its matrices in column order (Figure 1), such that when all the items are indexed, an algorithm would travel down down the first column, then the second. Whereas I was interpreting matrices in row order, where the same applies however I traverse the first row, then second row. By confusing the two, unexpected issues would occur (Figure 2). I fixed this with either transposing a matrix, in that I would flip either side of the matrix down its diagonal top left to bottom right, or I would stick to one order entirely, often row order.

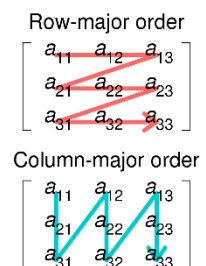


Figure 1 - By  
Cmglee - Own  
work, CC BY-SA 4.0

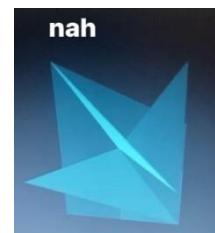


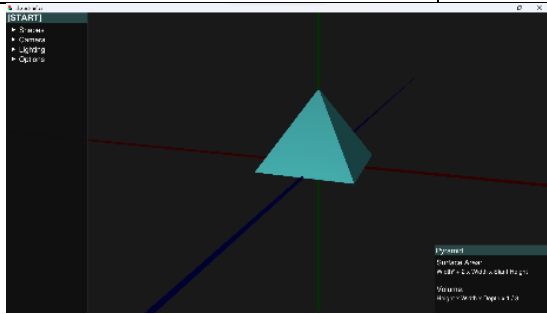
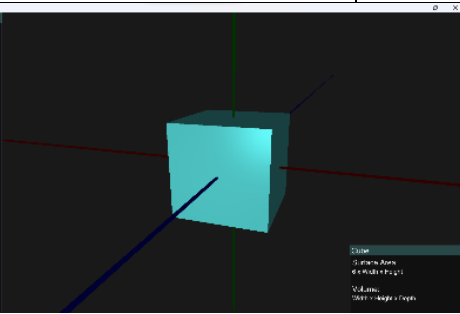
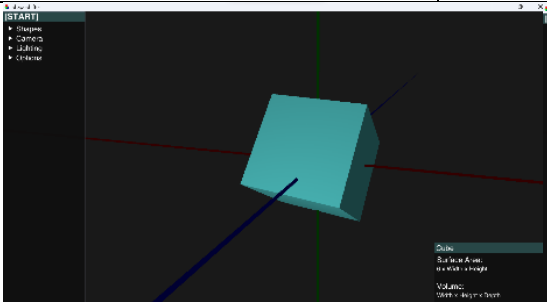
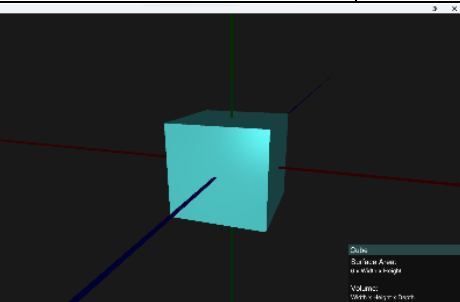
Figure 2

- A later problem I encountered was when calculating shape movements, as the calculation depended on the difference in Z coordinates between the shape and camera positions, so when I implemented variable camera position, it could mean that the transformation would go wrong and potentially cause the shape to not be rendered at all. The way this was fixed was through changing the difference in Z coordinates to the distance between the two points, using Pythagoras' Theorem.
- The last issue I had, affecting the user experience, was with interacting with the ImGui menu, because while you could be interacting with a slider, for example, my section of code would work independantly to that and take this input to rotate the shape. This was fixed through using an ImGui method I was not previously aware of which checks for mouse input on its windows and returns true if the mouse is held down. I then used this to say when the mouse input should be used to control the shape by checking for when this function returns False.

- A bug that, to date, I have not been able to solve is related to the transformation of the shape, such that if the camera position changes, mouse movement will cause unexpected movement of the shape. This is because the algorithm makes an assumption that you are facing the shape from the front view, so by dragging the mouse upwards on the shape, while facing it from behind, it will appear as if the shape is moving downwards, the opposite direction to the mouse movement. Logically this issue makes sense and can be expected, however patching it appears to me to be difficult as it would either include performing successive rotation transformations on the shape, or inverse transformations.

## Final Testing

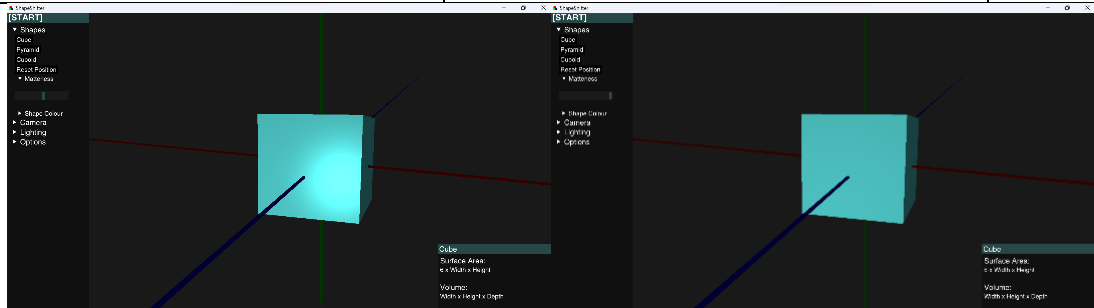
My program consists of one screen only so I will test the project as a whole. It is important to check that each feature of the program interacts as it should with every other. Below I will provide a list of each feature in my program, where in each case every other feature in the program has been changed to check if it makes any difference to it. I expect each test to pass, besides rotating the shape, due to the bug described prior.

<ul style="list-style-type: none"> <li>• Change Shape</li> </ul>	Performs as expected, without error.	✓
		
<ul style="list-style-type: none"> <li>• Reset Shape Position</li> </ul>	Performs as expected, without error.	✓
		

- Matteness

Performs as expected, without error. I could set a maximum value for matteness as putting the slider to the furthest position right causes the shape's 'shine' to disappear.

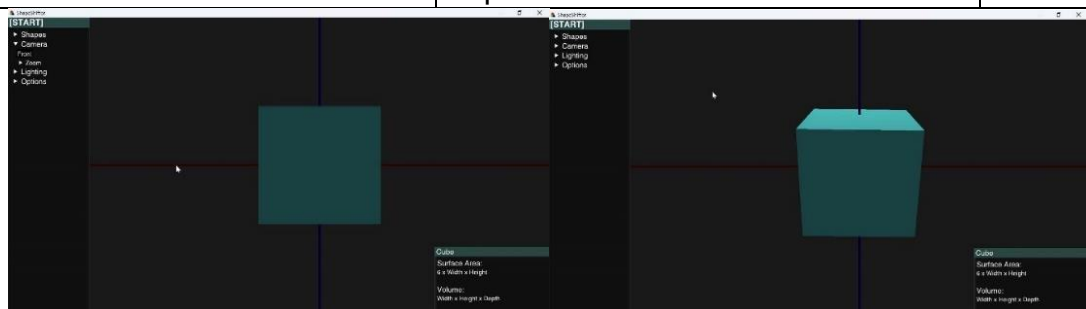
✓



- Rotate Shape

Performs roughly as expected, without error. Though depending on the camera angle, the rotation may move in a different way than expected.

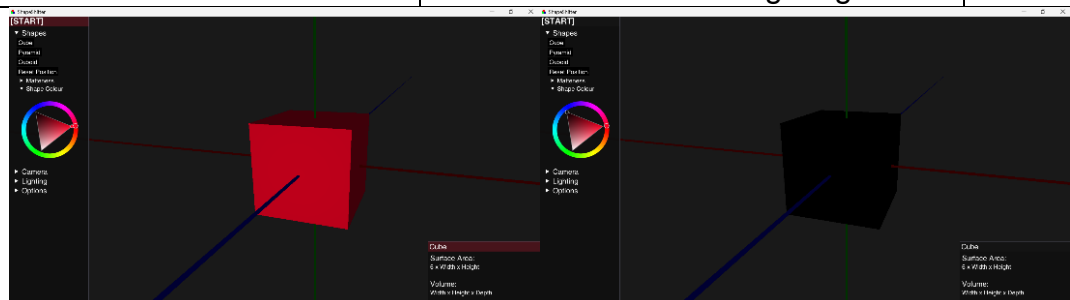
✗



- Shape Colour

Performs as expected, without error. Though I could set a minimum colour as setting the shape to black causes there to be no lighting.

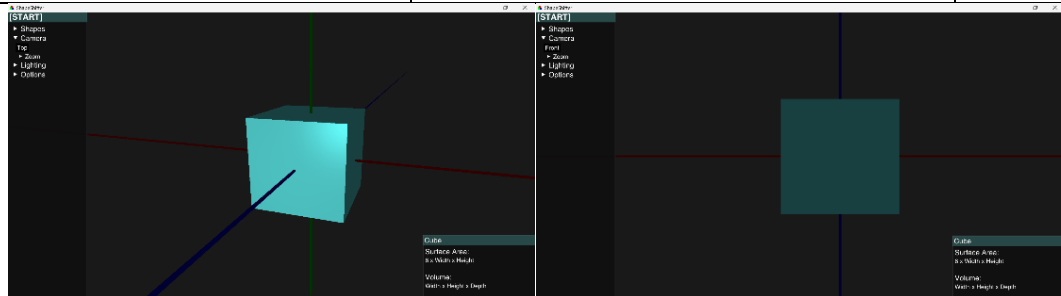
✓



- Camera Position

Performs as expected, without error. However the label only displays the name of the next camera position, instead of that of the current one.

✓



- Camera Zoom

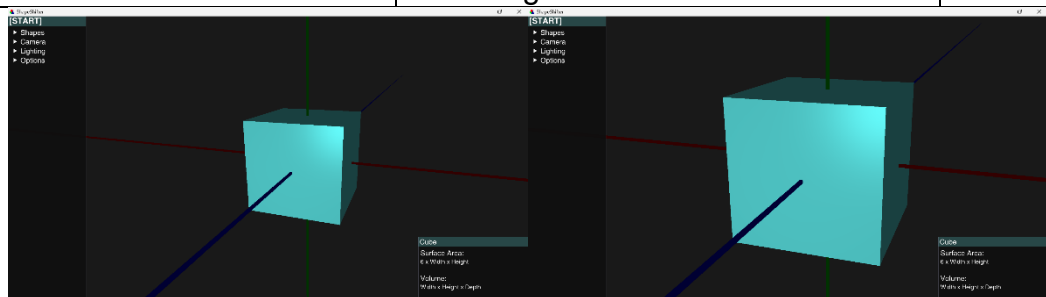
Performs as expected, without error.

✓

- Camera Zoom (Scrollwheel)

Performs as expected, without error. However the motion of the shape changing size looks slightly jittery and might need a different zoom algorithm to smooth it.

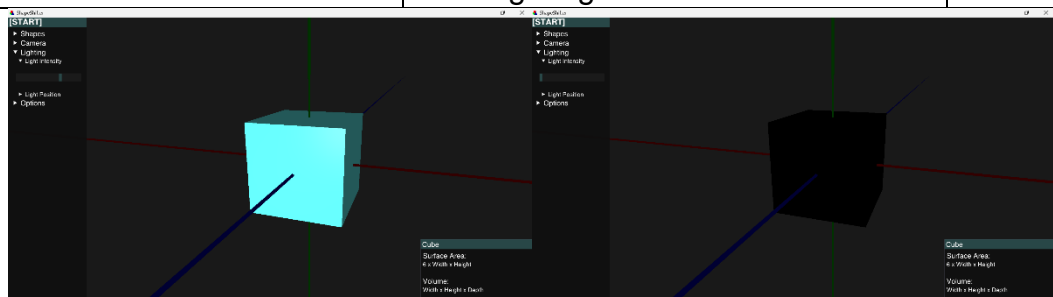
✓



- Light Intensity

Performs as expected, without error. Though there should be a minimum light intensity set, as when the slider is set to the far left, there is no lighting.

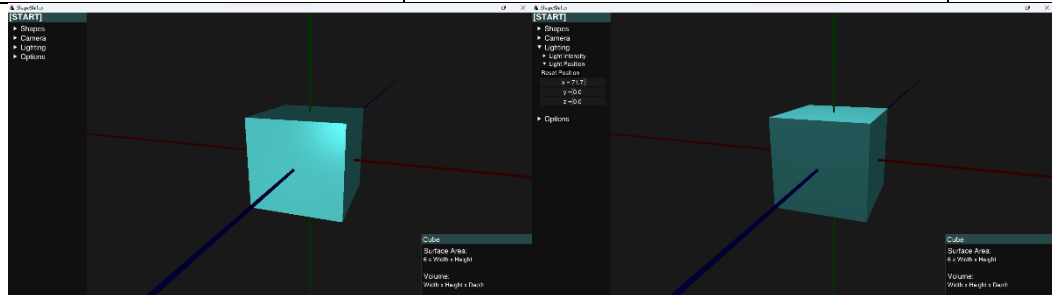
✓



- Light Position

Performs as expected, without error. However a rotation in the z axes has no effect, since the light source is at position (0, 0, -1).

✓



- Sensitivity

Works as expected without error.

✓

- Toggle Acceleration

Works as expected without error.

✓

- Shape Description

Works as expected without error.

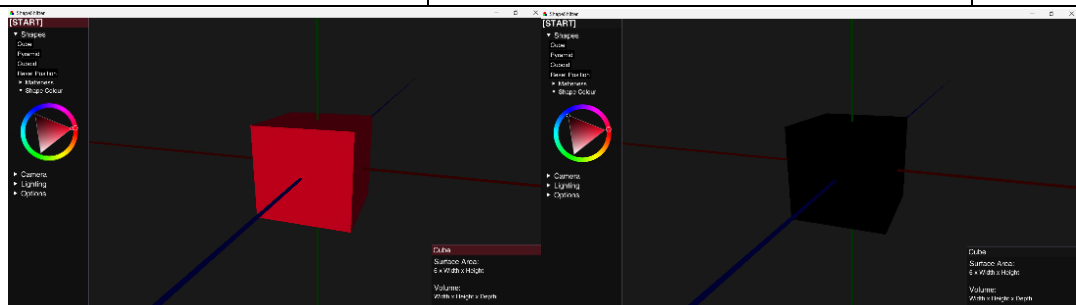
✓

Cube	Pyramid
<b>Surface Area:</b> $6 \times \text{Width} \times \text{Height}$	<b>Surface Area:</b> $\text{Width}^2 + 2 \times \text{Width} \times \text{Slant Height}$
<b>Volume:</b> $\text{Width} \times \text{Height} \times \text{Depth}$	<b>Volume:</b> $\text{Height} \times \text{Width} \times \text{Depth} \times 1 / 3$

- ImGui Colour Change

Works as expected, without error.

✓



- Exit (Cross)

Works as expected, without error.

✓

- Exit (Esc Key)

Works as expected, without error.

✓

Generally speaking, these outcomes were expected, and the main issue previously described does not impact too much on the user experience. So, I will continue to evaluate, distribute and share this version with my client despite this. Evidence for the above tests can be found in the evaluation, where I provide videos to prove also that I have met my objectives.

# Evaluation

## Client Feedback

I shared another discussion with my client, my maths teacher, and shared with them my software. I received some feedback, and I will divide this into positive feedback and criticisms.

### Positive Feedback

- The plans and elevations are a good feature, and will be useful in teaching Maths GCSE classes.
- The ability to freely rotate and enlarge the shape feels intuitive and could allow students to better learn from the program.
- The user interface is clear, and layed out well, though if all the tabs are open some of the lower ones become hidden by the edge of the screen. But this doesn't affect the experience too much as it is not often that they are all open.

### Points of Criticism

- To properly teach a GCSE class, more shapes that are relevant to the course would have to be available. Especially cylindrical shapes.
- A feature which would introduce planes of symmetry on each shape would be useful.

I am in agreement with all the points of criticism, as the GCSE Maths success criteria states that students should be able to calculate properties of cones, spheres and cyclinders, all of which are shapes that I have not been able to implement, and be able to identify lines of symmetry. I believe the issue with the GUI hiding tabs would not be to bad of an issue to fix, so if I were to complete the project again, this would be the first problem to fix.

# Objectives Met

Green = Completed Grey = Partially Completed Red = Not Completed

\* = Evidenced in the videos below

\*\* = Evidenced in my project code

## Objectives

- Be of use to students and teachers, relevant to their curricula as specified prior. The application should allow the transformation of a 3D object, allowing the counting of edges, faces and vertices in a maths lesson.

*This objective is partly met, as the final application does fulfil many of the desired functions, such as the ability to manipulate common 3D objects in the classroom. However, it did not meet all the requirements of the user, such as planes of symmetry.*

- \*Be able to generate and render 3D objects in real time. Users should be able to see immediately the effect of manipulating the object or making changes to the environment.

*This objective is completely met as shown in the videos. The user can drag the mouse to manipulate the object and results can be seen immediately.*

- \*Be able to manipulate each object e.g. Rotate using mouse controls, change colour using a colour wheel and zoom in/out.

*Again, the videos show that the mouse slider controls can be used to change colour and zoom, among other functions implemented.*

- \*Use perspective to give an accurate simulation of a real life object as it is transformed, allowing the user to recognise near and far edges or faces.

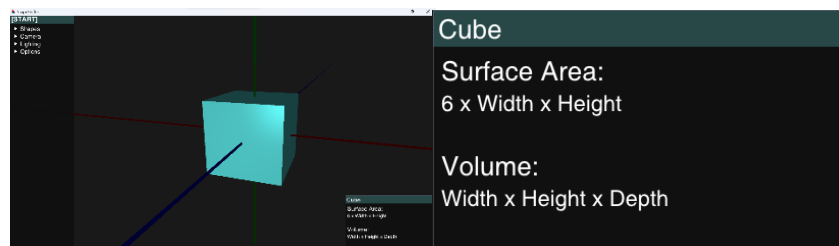
*Evidenced in the video, as the object edges recede they can be seen to be smaller with the perspective effect.*

- \*Be able to accurately simulate a light source, and how it behaves in a 3D environment with the chosen object and its material properties. Also be able to specify the location and quality of the light source (such as brightness).

*Evidenced in the videos, showing how the coordinates and quality of lighting can be manipulated.*

- Provide an intuitive and responsive Graphical User Interface (GUI), drawing features from similar programs. Assessing this objective will require some feedback from prospective users.

*This is evidenced in my client feedback, as my client positively commented on the usability of the user interface.*



- \*\*\*Exhibit robust Object-Oriented Programming (OOP) and smooth/consistent performance. The application should be robust and function without error, or ceasing to function due to run time errors.

*While I do not have a Frames per Second (FPS) counter, the program does not slow or jitter often.*

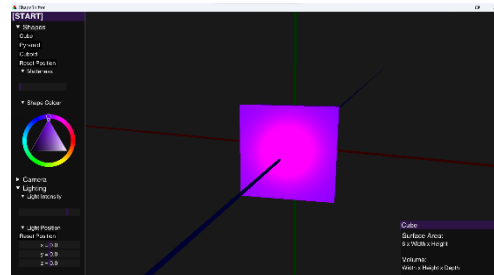
- \*\*Exhibit complex algorithms such as ray casting, projection, and linear / matrix transformations.

*This can be evidenced further in my documentation.*

## Extension

- \*\*Implement specular lighting.

*This can be evidenced further in my documentation*





- Implement planes of symmetry and an algorithm to calculate where they should be displayed. This will be useful in teaching planes of symmetry to GCSE classes.
- Label coordinates for each vertex. Again, to demonstrate how coordinates change after rotation.
- Import custom shapes from file or other methods. This will allow the user to render and manipulate other non standard shapes.
- Allow graphing including generating volumes of revolution and elongating a graph into a prism.
- Add an undo function to allow the user to return to the previous position after a rotation or enlargement. This will involve the use of a stack data structure.

I have purposefully set the first objective to be partially completed as it may be up to the client as to whether this has been fulfilled or not. The remaining incomplete objectives are due to time constraints, as they all require further complex algorithms, and are not necessarily required for the primary function of the software, however if I had included them, it would have improved my project. Furthermore, I could improve the project by implementing more shapes, specifically shapes that have a spherical or circular element to them, as this would be more relevant to the curriculum for the students of my client. However, this would be incredibly difficult to achieve at this point in time. This is because the way I have set OpenGL to render shapes is through using triangles, such that every shape that I draw and want to draw is comprised of small triangles. So, the way to make a circle would involve rendering a large number of tiny triangles until the edge of the resultant shape appears smooth and curved. I would need to design an algorithm to calculate the coordinates for each triangle, which while being hard to develop, would be costly on performance.

If I were to attempt this type of project again, or if I had more time for development, I would first try to patch or implement the following features:-

- The bug relating to the rotation of the shape. It might involve, successive rotation transformations, inverse transformations, or infact going about creating a new algorithm altogether and looking at the problem from a different perspective to solve it. By solving this it would reduce confusion on the user end and allow for easier learning, which links back with my first objective, suiting more towards younger students.
- I would try to implement my extension objectives, add more shapes including spherical/circular types and a method of rendering planes of symmetry, due to the points of criticism from my client feedback. I could perhaps also program a way of allowing the user to input custom shapes either from file, or by specifying the location on vertices. This would give the user more freedom to investigate about the shapes of their choice rather than being restricted to the discrete set of shapes I have provided.

My project coupled as an investigation into 3D rendering aswell, from starting it in May of 2021 to finishing it in April of 2022 I have learnt many skills and gained a lot of knowledge that I can transfer into my learning in university and potentially in my later career. One area I have grown a lot in is matrix transformations, we touch on it as a topic in my further mathematics course, but it was not until completing this project that I had realised its depth and importance of the subject in the world of graphics rendering. I now have much more experience in C++ and OpenGL programming, which I believe to be a powerful skill to have in this digital age, with almost all commercial software having a graphical user interface, or graphics related elements, such as video games. And with this language becoming less known, despite most high-end programs being comprised of it, there is a chance that demand in it will grow.

## Evidence

For simplicity, I have divided the program into four parts respective to the four menu options. Each video demonstrates each feature under that menu option and provides evidence to support my list of completed objectives. If the links are not working, see “Jack Doughty 2022 NEA\Documentation\Resources”.



Shapes.mp4



Camera.mp4



Lighting.mp4



Options.mp4