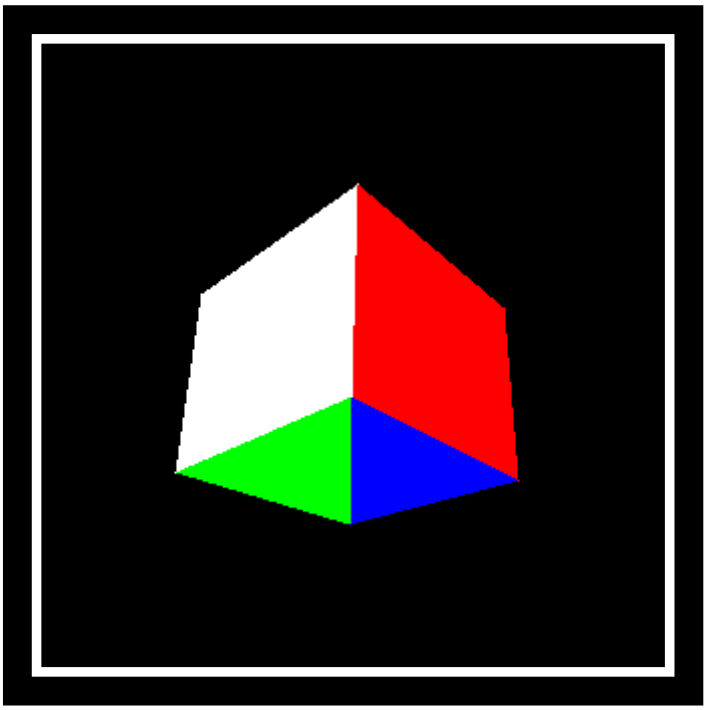
GEngine

A computer science project

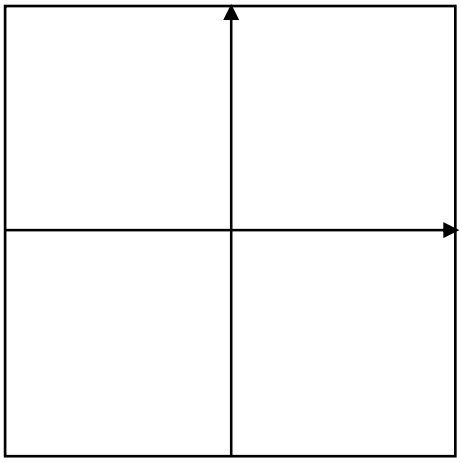
By George Beales



Contents

Mathematical Foundations

To start understanding the basic concepts in my project a bit of understanding of the maths behind computer graphics had to be understood. My experience with some of the concepts used was already decent enough to know I could start the project but here is an in-depth explanation of the methods used.



In this figure there is a circle at the geometric co-ordinates (1,1) I am going to imagine in this example this is just a point rather than a circle for simplicity.

Notation and matrices ● (1,1)

In my project there is a lot of matrix usage. There a lot of co-ordinates as well. To understand how I use the matrices I keep my co-ordinates in a column notation

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Using the figure, the x value would be 1 and so would the y value. The point is a point and not a vector and so therefore the w value would be considered 1.

Translation

The method of translation is moving the object to a different set of co-ordinates. This could be simply changing its x value by 1 and therefore moving it one to the right. You can also change multiple of these variables at once. When thinking about translation in my project the use of vector notation is used as well as matrices to do the calculations.

If I have a vector $\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ and I want to move it right along the x axis 1 and the y axis 1 I must apply a translation matrix. For translations the general matrix used is $\begin{bmatrix} 1 & 0 & x\ translation \\ 0 & 1 & y\ translation \\ 0 & 0 & 1 \end{bmatrix}$.

So, if I want to move the point 1 in both directions the y and x translation value would both be 1.

The final calculations would be $\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$.

To understand what happened in this multiplication I have created a cheat diagram to understand what is being multiplied and added.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} * \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{bmatrix}$$

Using this almost all the matrix maths in this project should be understandable with the extra part of the z axis which I will get onto explaining.

Rotation

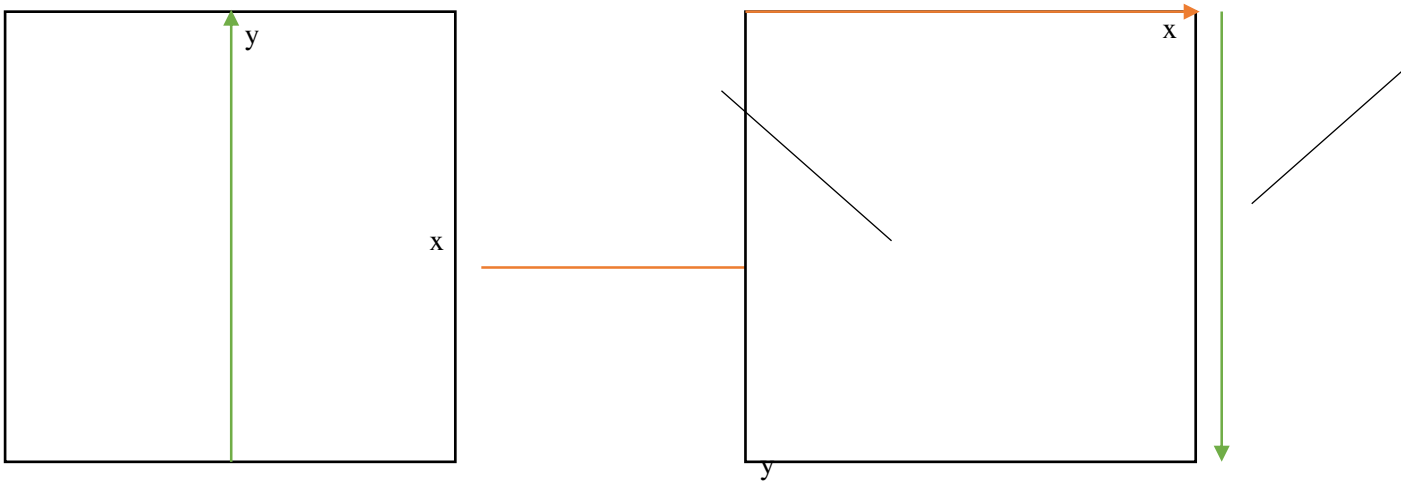
Rotation uses the same principle as translation but just by using different matrices and the fact is that it rotates around a certain point $\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

This is a basic rotation matrix for the 2d plane but set of more complicated ones are used to rotate around the axis in 3d space which I will get on to in a later section.

Drawing lines

In my project one of the first milestones was drawing lines. Although it may seem simple it unlocks a world of possibilities. At the start of my project I decided I wanted my project to use as few libraries as possible and try and keep to the complete basics. To do this I created my own class that inherited the JPanel class which contains a method for painting in the panel. The object that I could use to call painting methods was called 'g.' There were plenty of built in methods for drawing certain objects and painting certain objects but I knew they weren't something I wanted to use however in this section I first experimented with the drawline method for the simplicity of testing. This meant I could specify start and ending co-ordinates to draw a line. When I first did this, it created some very weird lines which I couldn't really understand that well, this was down to the fact that the axis worked in a very weird way and wasn't what I was used to or even explained in an earlier sketch at all.

To understand how the co-ordinate system worked in the paint class I have created a sketch.



Regular co-ordinate system Raster space

What these sketches illustrate is the fact that the centre of the co-ordinates or (0,0) is at the top left and the y axis is flipped in positivity. This means the higher the y value the lower down the object. Weird right? This is done for a very good reason but at the start of my project I initially thought this was stupid and made no sense but that was an idea I'd soon realise I'd have to lose.

This system is very important for a later section on perspective as this was a huge part of creating the project.

Drawing meshes

As with drawing lines there were plenty of pre-made methods for drawing pretty much any object ranging from triangles to multiple sided polygons. This was something I wanted to avoid as it would void most of my project, so with this knowledge I knew I needed to create what I already had but with my own methods.

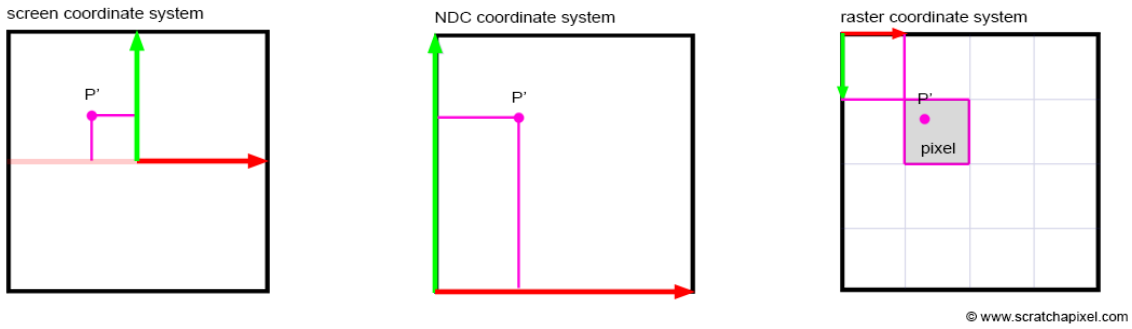
Drawing every pixel individually on the panel means I have full control over what I am showing. If I know which pixels to colour, I can create any shape I want it is just a case of knowing which pixels I needed to shade in. With lines this is quite simple its just a case of knowing the gradients and creating a certain equation that can be followed with different values of y or x but what if I wanted to colour in a whole object maybe even a triangle?

This uses a technique called rasterization and I will get on to how that whole algorithm works in a later section but for now I am just going to talk about how I created the drawings with the information of what to colour. If I know how big my panel is, I can gather that information to know how many pixels there will be by multiplying the width by height (in pixels).

Once I have this, I can create a 2d array with this information. In java I could create an array and go through this process but there is a much easier thing to do which would cut down on process time and this is called a buffered image. This is basically an image which can have its x and y co-ordinated pixels changed in colour to create images. This image can then be drawn by the java library through the method I just specified. This removes processing time and methods which would be unnecessary to include.

Converting world space to screen space and normalisation

As talked about in an earlier section I was having problems understanding the co-ordinate system in place and was trying to get to grips with how it worked. In this section I am going to try and explain how I converted my regular co-ordinates to a proper place on the screen. To do this we have to understand the pipeline.



In the above image you can see that we start with the normal screen co-ordinate system which is what most people will know and use in maths where the centre is the origin and points in the negative planes are shown where they should be. The graphics module I use in java takes points only in positive values. The top left is considered some sort of origin and no negative values can be displayed. What if I wanted to display some negative value on the screen and make the origin the centre of the screen rather than the top left? This can be done using simple conversions to make use of the way the graphics module works. First we need to convert our co-ordinate into Normalised Device Co-ordinate system. The normalised device system is like taking our maximum value in any direction to be 1 and every other value has to be a decimal between -1 and 1 to exist. If it's higher than one it can't exist in the screen as this is our viewport size.

$$P_{normalised}.x = \frac{P.x + viewport.width/2}{viewport.width}$$

Next we need to convert our position into the raster co-ordinate system which means giving it a value what make the center of the screen the origin. This means all negative values will now technically be all positive. This can then be rounded down to give us the closes pixel to the specified place in the panel.

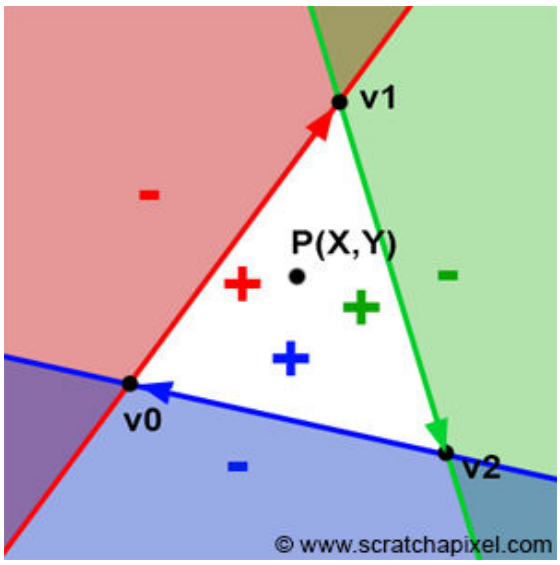
$$P_{raster}.x = \lfloor P_{normalised}.x * window.width \rfloor$$

$$P_{raster}.y = \lfloor (1 - P_{normalised}.y) * window.width \rfloor$$

After we have our raster point, we can define which pixel we want to colour as at this point it will be an integer assigning to a pixel.

Rasterization and Z-buffer algorithm

Perhaps one of the most difficult parts of my entire project was the rasterization algorithm as it encompasses a part of maths that is quite difficult for me as well as putting it into the context of a computer. To achieve my rasterization in my program I went with a style of rasterization that uses a system called a barycentric co-ordinate system. I didn't truly understand the little individual parts of the system that make it work on a small base level but what I can explain is how I used the knowledge in my program. What rasterization is, is figuring out which pixels are inside and outside of a triangle. This is extremely important if I want to colour in the shapes in my 3D engine rather than just have lines connecting the points. To figure out whether a pixel lies within the range of the triangle we have to work out what are known as edge functions. For the sake of my understanding and simplicity I will name the edge functions b1, b2 and b3. These edge functions are almost like equations of lines that specify a boundary. The below image shows how these boundaries can be used to determine whether a pixel lies within the triangle.



These lines represent an edge function which is a number that can be used to compare against a value to determine whether it lies within the triangle.

$$E_{red}(P) = (P.x - v0.x) * (v1.y - v0.y) - (P.y - v0.y) * (v1.x - v0.x)$$

The equation above is the value you would get when comparing the point with the triangle to check whether it is on the correct side of the red boundary. There are two more edge functions for the other boundaries and the values fetched from these functions need to equal greater than or smaller than 0 depending on which edge function it is. This method however is more specific to this example image I have. The method I used in my code had to take advantage of the way my pixel system works. In my method I will treat the three points of the triangle as p1, p2 and p3. My edge functions will be b1, b2 and b3.

$$b1 = \frac{(y - p3.y) * (p2.x - p3.x) + (p2.y - p3.y) * (p3.x - x)}{triangle\ area}$$

$$b2 = \frac{(y - p1.y) * (p3.x - p1.x) + (p3.y - p1.y) * (p1.x - x)}{triangle\ area}$$

$$b3 = \frac{(y - p2.y) * (p1.x - p2.x) + (p1.y - p2.y) * (p2.x - x)}{triangle\ area}$$

This is where the triangle area can be worked out using this equation.

$$triangle\ area = (p1.y - p3.y) * (p2.x - p3.x) + (p2.y - p3.y) * (p3.x - p1.x)$$

Using these formulae, we can work out whether the x and y are in the triangle in question. To do this we can check these edge formulae against values of 1 or 0.

$$If\ b1 \geq 0\ and\ b1 \leq 1\ and\ b2 \geq 0\ and\ b2 \leq 1\ and\ b3 \geq 0\ and\ b3 \leq 1$$

If all these conditions are met, then the point does lie within the triangle's boundaries. The next thing we need to do is check whether the points we are colouring in is the closest point on the screen in relation to the z-axis. To do this we need to calculate the Z-buffer.

Z-buffer

In my program I actually implemented quite an easy system for the Z-buffer. The method I used involved creating an array which contained the same number of pixels as elements in the panel. This array would have all the elements set to a value of negative infinity to define a really far distance. This means that pretty much everything in the scene would be rendered. When the rasterization algorithm was running there would be another check to see whether a certain pixel should be coloured for said triangle. This check would involve check if the pixel's depth was in fact the smallest depth compared to every other object's pixel depth for the x and y co-ordinate. To work out the depth of the pixel in the rasterization algorithm the below equation can be used.

$$depth = b1 * p1.z + b2 * p2.z + b3 * p3.z$$

If this depth was the smallest for a certain pixel the depth value would be stored in the array as the closest object. If another object had a closer depth, then it would replace the value stored in the array.

Perspective

So far, my program is really starting to take shape. So far it can draw points and shade in with an active Z-buffer that makes objects look like they are in the correct place. There is one problem however that really does make it all look like its for nothing though and that is perspective. At this point there is no way to tell how far away something is and this is because it is being what's called "projected orthographically." This means if something is really far away from the camera it still looks the same size as something really close to the camera. Now there are multiple ways of achieving a perspective projection and this was what probably cause the most issues during development in my project. The two main methods are doing something called a perspective divide or creating a perspective manipulation matrix. These are two good ways of achieving this effect however each have their advantages and disadvantages. The main problem with a perspective divide is it doesn't use a changeable fov. This means we can't alter the perspective effect. The perspective matrix method however is very difficult to achieve and at the minimum understand. I personally couldn't get the matrix method to work however but I could get the perspective divide to work. Below is a comparison of how each method is achieved.

To achieve a perspective divide the original x and y values of each vertex has to be altered using the below equation.

$$P_{perspective}.x = \frac{P.x}{-P.z}$$

$$P_{perspective}.y = \frac{P.y}{-P.z}$$

Its quite easy to see why it is called a perspective divide. Next, we have the matrix that we could use for a perspective divide with the angle of fov included.

$$\begin{bmatrix} \frac{1}{\arctan(\frac{\alpha}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{\alpha}{2})} & 0 & 0 \\ 0 & 0 & \frac{-NearZ - FarZ}{NearZ - FarZ} & \frac{2 * FarZ * NearZ}{NearZ - FarZ} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

To truly understand this matrix, I would have to have a good understanding of how it works but as far as I could get with this matrix well, was not very far. I don't have a good enough understanding of this maths topic to truly get to grips with it but it said to be more efficient and obviously include an fov angle.

Movement and world transformation around all axis

In this 3d engine the movement system used is not the typical system because I don't really truly know what the typical system is because I had to figure it out completely by myself. It might seem like something quite simple but actually creates huge difficulties when using rotation at the same time. Say you want to move forward in the world what you are actually doing is moving the whole world in the opposite direction to the direction you want to go. This could be done quite simply by say moving everything in the z axis in the opposite direction to where you want to travel. When rotation is added into the equation you will need to rotate the world the same angle in the opposite direction. When these two are added together it can seem like it works quite well until you try looking around once you have move from the origin. This issue exists because whenever we rotate the objects in the world space, we rotate them around the origin and if you have moved away from the origin it can look quite weird when rotating once you have moved. To fix this we must introduce the system of a camera object to handle all information related to this. Once we have a camera we need to figure out where every object is in relation to the camera's co-ordinates and rotation. To calculate rotations, we need to use the same rotation matrices in the previous sections only after we have translated all the meshes in the world. Once all the meshes are translated everything can be rotated using rotation matrices in the exact same normal way we did before. Below is an example of how these are calculated.

Say the camera has the co-ordinates $\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$ first we need to translate every object in the world by this amount in the opposite direction. The matrix $\begin{bmatrix} 1 & 0 & 0 & -x\ translation \\ 0 & 1 & 0 & -y\ translation \\ 0 & 0 & 1 & -z\ translation \\ 0 & 0 & 0 & 1 \end{bmatrix}$ can be used for working out the new position vectors. First, we apply this translation matrix and next we apply all the rotation matrices for every axis which are:

$$x\ rotation = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(x\ rotation) & -\sin(x\ rotation) & 0 \\ 0 & \sin(x\ rotation) & \cos(x\ rotation) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$y\ rotation = \begin{bmatrix} \cos(y\ rotation) & 0 & \sin(y\ rotation) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(y\ rotation) & 0 & \cos(y\ rotation) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$z\ rotation = \begin{bmatrix} \cos(z\ rotation) & -\sin(z\ rotation) & 0 & 0 \\ \sin(z\ rotation) & \cos(z\ rotation) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Once all these matrices have been applied the world should be in the correct position to give the illusion of correct movement. The only problem remains is the fact that if I press the forward button the program will just shift everything in the z axis in the opposite direction which is good if I want to move the world but what if I am looking in a different direction to down the axis and I want to move forward at that angle rather than just along the axis. The way to overcome this is to use the angles at which the camera is rotation and adjust the x and z values accordingly. The way I did this in my code is show below.

If I press the forward button and I am at a rotation the values of x,y,z I need to change are:

$$x = \sin(y\ rotation) * forward\ value$$

$$y = \sin(x\ rotation) * -forward\ value$$

$$z = \cos(y\ rotation) * -forward\ value$$

This would once again give the illusion of traveling in the direction the camera is facing. I have included the other directional buttons below.

$$Backwards: x = \sin(y\ rotation) * -forward\ value$$

$$y = \sin(x\ rotation) * forward\ value$$

$$z = \cos(y\ rotation) * forward\ value$$

$$Left: x = \cos(y\ rotation) * -forward\ value$$

$$y = 0$$

$$z = \sin(y\ rotation) * -forward\ value$$

$$Right: x = \cos(y\ rotation) * forward\ value$$

$$y = 0$$

$$z = \sin(y\ rotation) * forward\ value$$

Software Implementation

So far, I have covered all the ideas that I want to implement into my code and in this section, I am going to explain how I will go around implementing these ideas.

Storing information

The information required for this project is heavy and requires extreme efficiency to work correctly or at all. I already know that for a project like this everything must be object oriented in order to work. To store the object information, I will create the following classes:

- A vertex information class
- A triangle information class
- A mesh information class

In the vertex information class, it will hold the most basic information about every point in an object. Each vertex class will hold an x, y and z co-ordinate as well as a getter and setter for each. This means that I can keep my vertex class as simple as possible as this will mean more efficiency.

In the triangle's information class, I will store information about the triangle such as the colour. I will also keep a vertex list that will contain three vertex classes inside. This means that each triangle will have three vertices just like it should. As before I will have getters and setters for both attributes.

In the mesh's information class, I will store a name for the mesh as well as a dynamic triangle list that can contain any number of triangles. This will mean I can create a full object made from triangles all stored in one class.

Another piece of information I will need to store are matrices. To do this I can create a class that contains a 2d array of numerical values. This will also contain getters and setters to set and retrieve information from the individual objects. I will use this for both 3x3 and 4x4 matrices.

A crucial amount of information I need to store is all about the camera and where the camera is and how it interacts with the renderer. In the camera class I will need to store a lot of attributes. Most of these attributes will mainly be matrices containing the information from the previous sections on rotation and translation. The camera will also need to contain information about its position and rotation in each axis. To do this I will simply store these as float values. One last thing I need to store in the camera class is all the triangle information that it can see so when the renderer needs to render it can fetch this information from the camera. To do this I will have two triangles class lists that contain both the information about where the triangles are in the world nondependent of the camera and where the triangles are in the world after the camera has moved and rotated.

Rendering

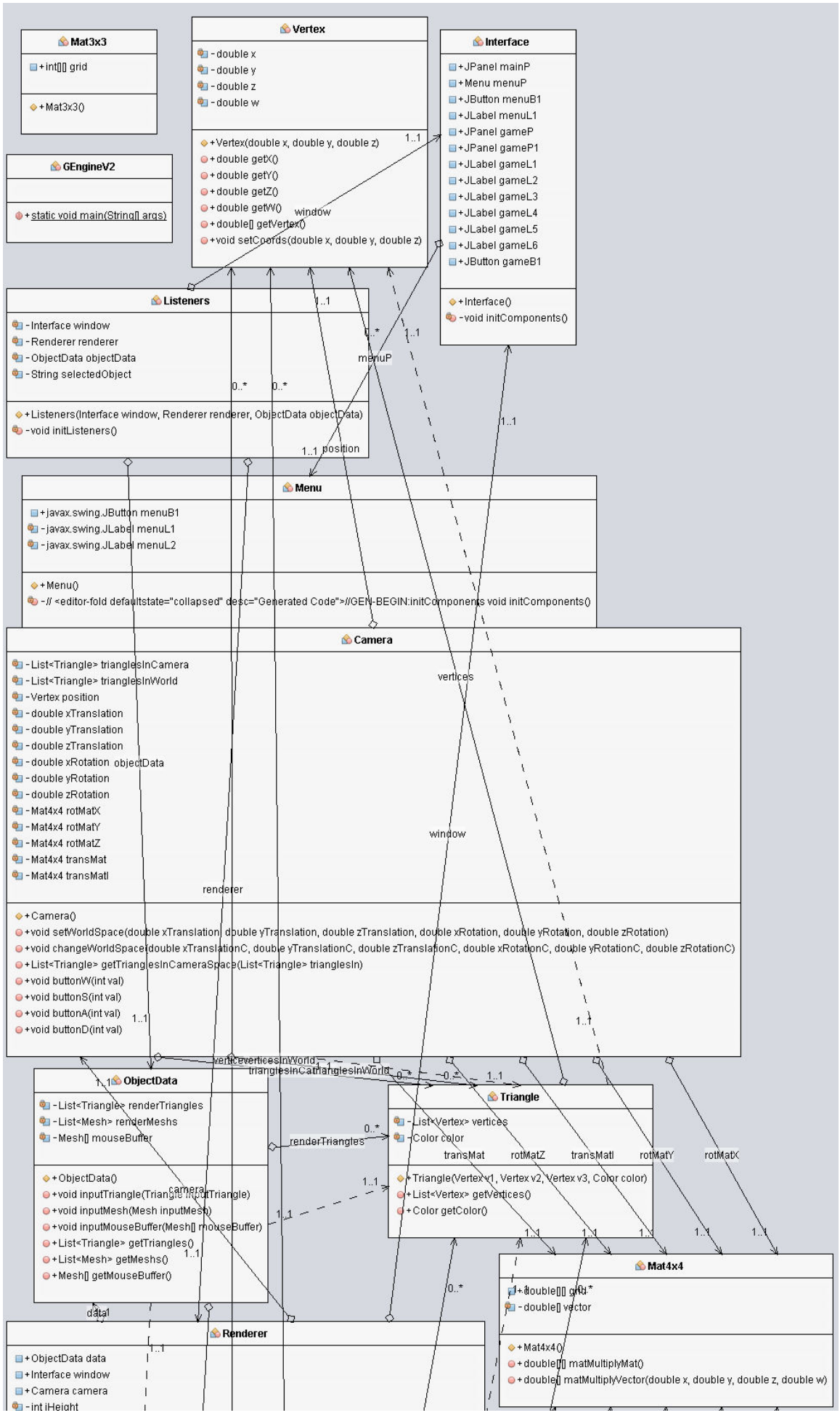
In my program I am quite solid on how I am going to implement rendering, and this is how I am going to go about doing it. I will create a class that inherits from the JPanel class in awt to use polymorphism to override the function paint(). This means I can create a panel which can have its own effect when the paint method is called on it. This method also effects how the repaint method works. To refresh the scene, I will call the repaint method which basically calls the original overridden method paint. What this means for my project is I will have a customizable class for a panel which contains all the render information within it.

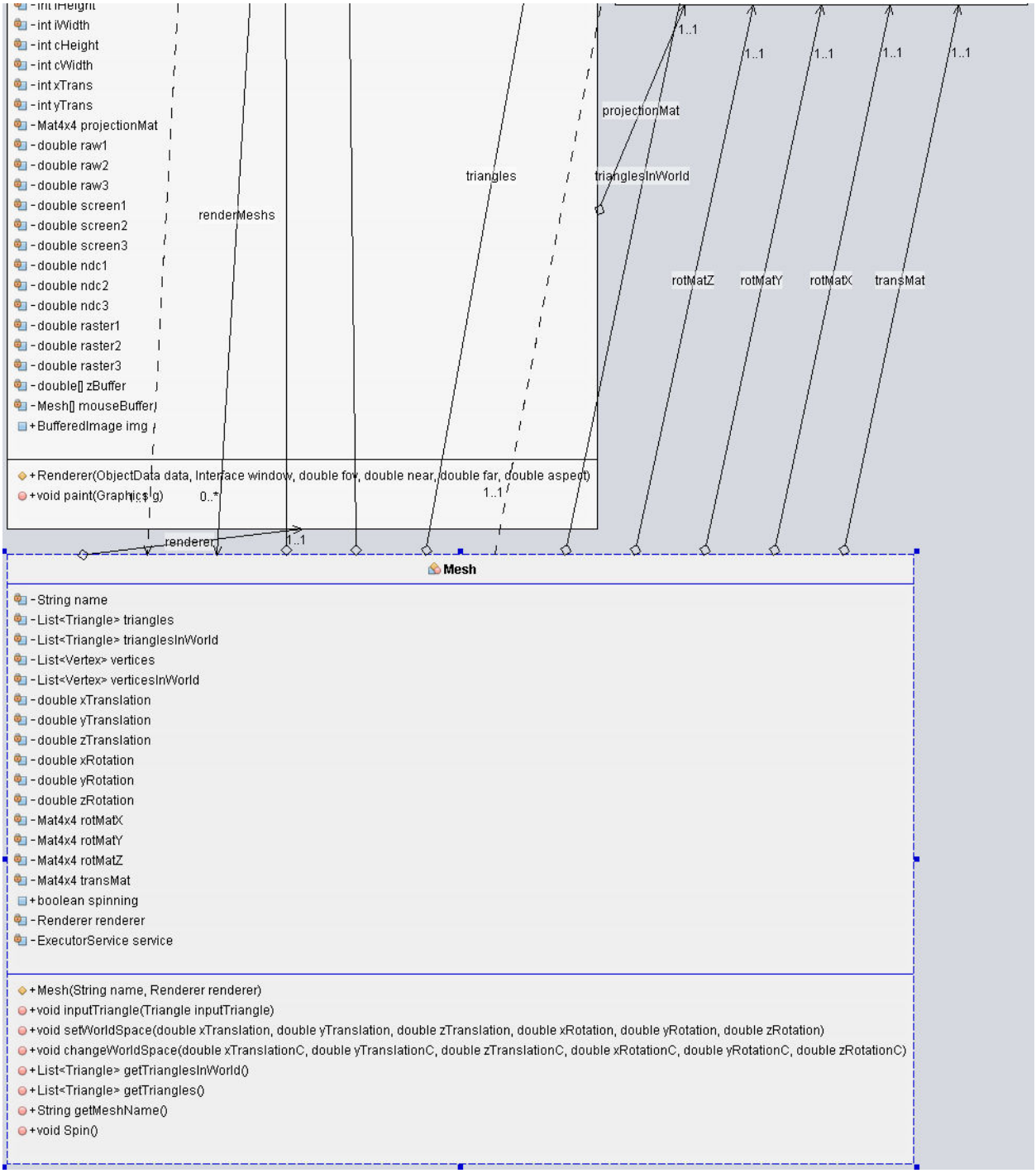
Threads

I have also decided in this project I will separate the listeners from the class that they will act on. To do this I will create a class for the listeners with every other object created in the code being available from within. This means that I can add a listener for any button for any class. This adds lots of flexibilities and makes it a lot easier to handle listeners if they are all in the same place.

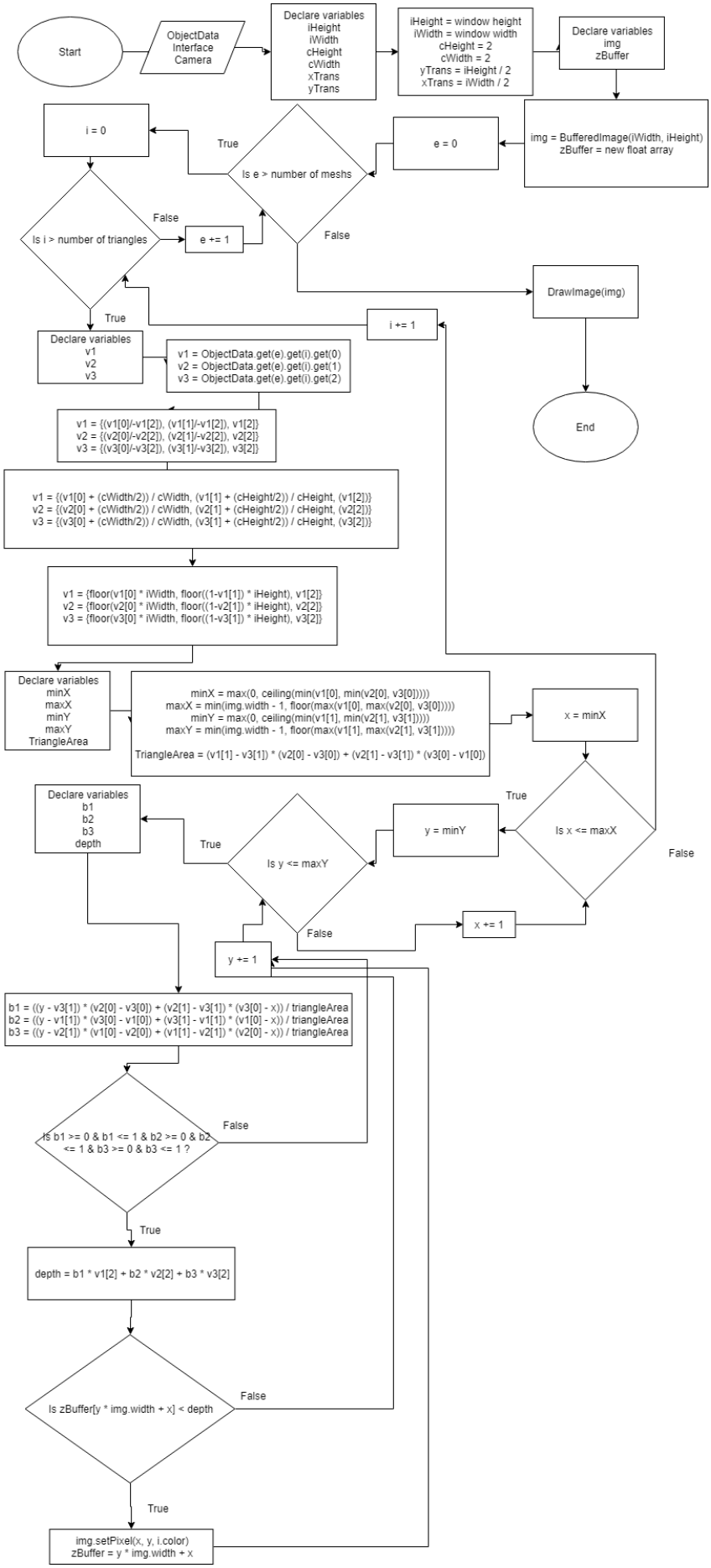
Class diagram

To better describe my class system, I have included a class diagram.


Mat3x3
+int[][] grid
+Mat3x3()
Vertex
-double x
-double y
-double z
-double w
+Vertex(double x, double y, double z)
+double getX()
+double getY()
+double getZ()
+double getW()
+double[] getVertex()
+void setCoords(double x, double y, double z)
Interface
+JPanel mainP
+Menu menuP
+JButton menuB1
+JLabel menuL1
+JPanel gameP
+JPanel gameP1
+JLabel gameL1
+JLabel gameL2
+JLabel gameL3
+JLabel gameL4
+JLabel gameL5
+JLabel gameL6
+JButton gameB1
+Interface()
-void initComponents()
GEngineV2
+static void main(String[] args)
window
Listeners
-Interface window
-Renderer renderer
-ObjectData objectData
-String selectedObject
+Listeners(Interface window, Renderer renderer, ObjectData objectData)
-void initListeners()
menuP
position
Menu
+javax.swing.JButton menuB1
-javax.swing.JLabel menuL1
-javax.swing.JLabel menuL2
+Menu()
-// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents void initComponents()
Camera
-List<Triangle> trianglesInCamera
-List<Triangle> trianglesInWorld
-Vertex position
-double xTranslation
-double yTranslation
-double zTranslation
-double xRotation
-double yRotation
-double zRotation
-Mat4x4 rotMatX
-Mat4x4 rotMatY
-Mat4x4 rotMatZ
-Mat4x4 transMat
-Mat4x4 transMatI
+Camera()
+void setWorldSpace(double xTranslation, double yTranslation, double zTranslation, double xRotation, double yRotation, double zRotation)
+void changeWorldSpace(double xTranslationC, double yTranslationC, double zTranslationC, double xRotationC, double yRotationC, double zRotationC)
+List<Triangle> getTrianglesInCameraSpace(List<Triangle> trianglesIn)
+void buttonW(int val)
+void buttonS(int val)
+void buttonA(int val)
+void buttonD(int val)
vertices
objectData
window
renderer
vertexverticesInWorld
trianglesInCatrianglesInWorld
ObjectData
-List<Triangle> renderTriangles
-List<Mesh> renderMeshs
-Mesh[] mouseBuffer
+ObjectData()
+void inputTriangle(Triangle inputTriangle)
+void inputMesh(Mesh inputMesh)
+void inputMouseBuffer(Mesh[] mouseBuffer)
+List<Triangle> getTriangles()
+List<Mesh> getMeshs()
+Mesh[] getMouseBuffer()
renderTriangles
camera
Triangle
-List<Vertex> vertices
-Color color
+Triangle(Vertex v1, Vertex v2, Vertex v3, Color color)
+List<Vertex> getVertices()
+Color getColor()
transMat
rotMatZ
transMatI
rotMatY
rotMatX
Mat4x4
-double[][] grid
-double[] vector
+Mat4x4()
+double[][] matMultiplyMat()
+double[] matMultiplyVector(double x, double y, double z, double w)
data1
Renderer
+ObjectData data
+Interface window
+Camera camera
-int iHeight



Rasterizer flow chart


Start
ObjectData
Interface
Camera
Declare variables
iHeight
iWidth
cHeight
cWidth
xTrans
yTrans
iHeight = window height
iWidth = window width
cHeight = 2
cWidth = 2
yTrans = iHeight / 2
xTrans = iWidth / 2
Declare variables
img
zBuffer
img = BufferedImage(iWidth, iHeight)
zBuffer = new float array
e = 0
Is e > number of meshs
True
False
i = 0
Is i > number of triangles
False
e += 1
True
DrawImage(img)
End
i += 1
Declare variables
v1
v2
v3
v1 = ObjectData.get(e).get(i).get(0)
v2 = ObjectData.get(e).get(i).get(1)
v3 = ObjectData.get(e).get(i).get(2)
v1 = {(v1[0]/-v1[2]), (v1[1]/-v1[2]), v1[2]}
v2 = {(v2[0]/-v2[2]), (v2[1]/-v2[2]), v2[2]}
v3 = {(v3[0]/-v3[2]), (v3[1]/-v3[2]), v3[2]}
v1 = {(v1[0] + (cWidth/2)) / cWidth, (v1[1] + (cHeight/2)) / cHeight, (v1[2])}
v2 = {(v2[0] + (cWidth/2)) / cWidth, (v2[1] + (cHeight/2)) / cHeight, (v2[2])}
v3 = {(v3[0] + (cWidth/2)) / cWidth, (v3[1] + (cHeight/2)) / cHeight, (v3[2])}
v1 = {floor(v1[0] * iWidth, floor((1-v1[1]) * iHeight), v1[2]}
v2 = {floor(v2[0] * iWidth, floor((1-v2[1]) * iHeight), v2[2]}
v3 = {floor(v3[0] * iWidth, floor((1-v3[1]) * iHeight), v3[2]}
Declare variables
minX
maxX
minY
maxY
TriangleArea
minX = max(0, ceiling(min(v1[0], min(v2[0], v3[0]))))
maxX = min(img.width - 1, floor(max(v1[0], max(v2[0], v3[0]))))
minY = max(0, ceiling(min(v1[1], min(v2[1], v3[1]))))
maxY = min(img.width - 1, floor(max(v1[1], max(v2[1], v3[1]))))
TriangleArea = (v1[1] - v3[1]) * (v2[0] - v3[0]) + (v2[1] - v3[1]) * (v3[0] - v1[0])
x = minX
Is x <= maxX
True
False
y = minY
Is y <= maxY
True
False
x += 1
y += 1
Declare variables
b1
b2
b3
depth
b1 = ((y - v3[1]) * (v2[0] - v3[0]) + (v2[1] - v3[1]) * (v3[0] - x)) / triangleArea
b2 = ((y - v1[1]) * (v3[0] - v1[0]) + (v3[1] - v1[1]) * (v1[0] - x)) / triangleArea
b3 = ((y - v2[1]) * (v1[0] - v2[0]) + (v1[1] - v2[1]) * (v2[0] - x)) / triangleArea
Is b1 >= 0 & b1 <= 1 & b2 >= 0 & b2 <= 1 & b3 >= 0 & b3 <= 1 ?
False
True
depth = b1 * v1[2] + b2 * v2[2] + b3 * v3[2]
Is zBuffer[y * img.width + x] < depth
False
True
img.setPixel(x, y, i.color)
zBuffer = y * img.width + x

Developing and testing

Testing during development

Here I am using the test tables from my design section to separate the stages of development and show different tests for mid development. I will include screenshots for each test and the end of this section.

Stage 1

Test Number	Description	Type	Test data	Expected result	Result
1	Does the window open once the code executes?	Valid	Opening the program.	The window appears.	Pass
2	Does the program application stop once the window is closed?	Valid	Closing the program and using task manager to check if the process has gone.	The program and the window both close.	Pass
3	Does the window have any scaling or sizing issues?	Invalid	Comparing the window size with the relevant objects or other programs on the computer.	The program can be scaled properly and have the correct dimensions.	Pass
4	Are there any objects misplaced or seem glitched or bugged?	Valid	Looking at the program once executed to see if any of the objects are misplaced.	The objects are in the correct place and they are not misplaced.	Pass

Stage 2

Test Number	Description	Type	Test data	Expected result	Result
1	Does the object render properly?	Valid	Once executing the program looking at the main window of the program where the 3d objects will be located.	The triangular object will be rendered properly, and this will be easily clear.	Pass.
2	Does the Z-Buffer work in the program?	Valid	Once executing the program looking at the main window of the program where the 3d objects will be located. Does the triangular object have any side which are displaying on top of other sides when they shouldn't?	The triangular object will have the sides rendered behind each other appropriately so that the object looks normal and not 4 dimensional.	Pass.
3	Is there any perspective at all?	Invalid	Once executing the program looking at the main window of the program where the 3d	The further away elements should seem smaller and this can be seen by looking at a linear object and	Pass.

			objects will be located. Will the further away elements seem smaller even if they are the same size as the closer ones	seeing if it looks smaller on the further away parts.	
4	Can the objects move or rotate?	Valid	Once executing the program looking at the main window of the program where the 3d objects will be located.	The objects will be able to be rotated or moved and this will be done by a slider or button.	Pass.

Stage 3

Test Number	Description	Type	Test data	Expected result	Result
1	Does the physics demo work?	Valid	Launch the physics demo from the main menu	The physics demo works as planned and does not crash.	Fail.
2	Does the lighting demo work?	Valid	Launch the physics demo from the main menu	The lighting demo works as planned and does not crash.	Fail.
3	Does the collision demo work?	Valid	Launch the physics demo from the main menu	The collision demo works as planned and does not crash.	Fail.
4	Does the object interaction demo work?	Valid	Launch the physics demo from the main menu	The object interaction demo works as planned and does not crash.	Fail.
5	Does the particle demo work?	Valid	Launch the physics demo from the main menu	The particle demo works as planned and does not crash.	Fail.

Stage 4

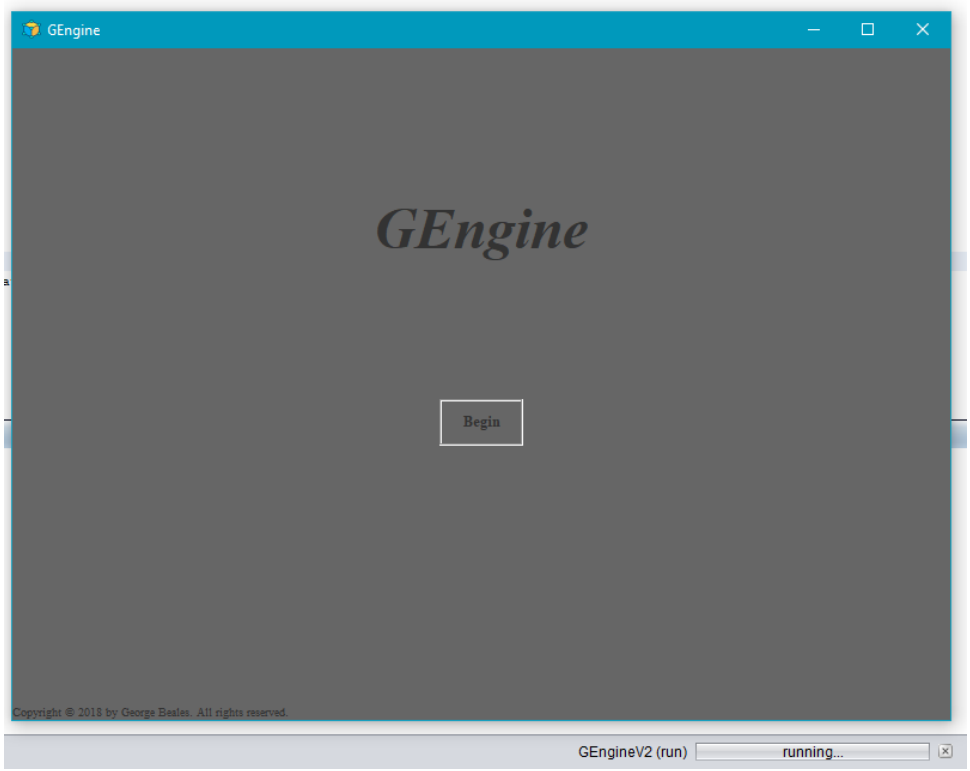
Test Number	Description	Type	Test data	Expected result	Result
1	Is the button's placement correct?	Valid	When the program is executed.	The buttons are in the correct place.	Pass.
2	Do the buttons navigate you around the menu?	Valid	When the program is executed. The buttons are on the menus.	The buttons navigate you around the menu.	Pass.
3	Does the menu work with no bugs or glitches when pressing certain option or select buttons?	Invalid	When the program is executed. The buttons are on the menus.	The buttons are fluid and work perfectly.	Pass with condition.
4	Is the background 3d model camera pan working?	Valid	When the program is executed. The background should be visible.	The background 3d model camera works and does not interfere with the rest of the	Fail.

				code.	

Stage 5

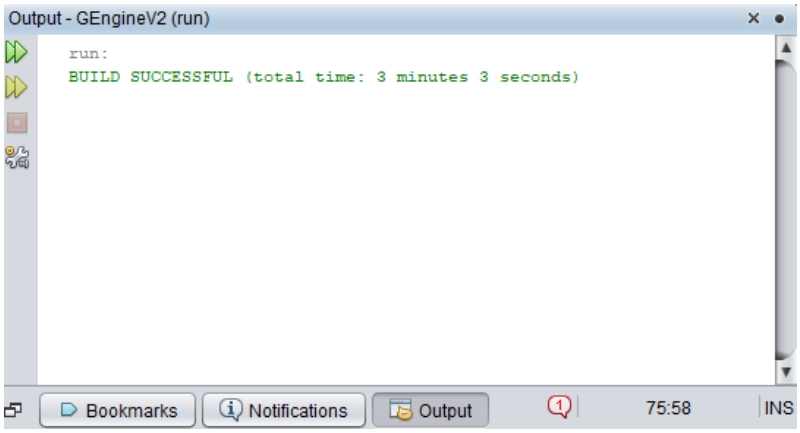
Test Number	Description	Type	Test data	Expected result	Result
1	Do the buttons lead to the demos?	Valid	When the program is executed navigate to the demo's menu	The buttons open up the demos with ease.	Pass with condition.
2	Do the options in the options menu change what happens in the programs?	Valid	When the program is executed navigate to the options menu and then to one of the demos	The demos have the effects of one of the options being toggled.	Fail with condition.

Test 1.1



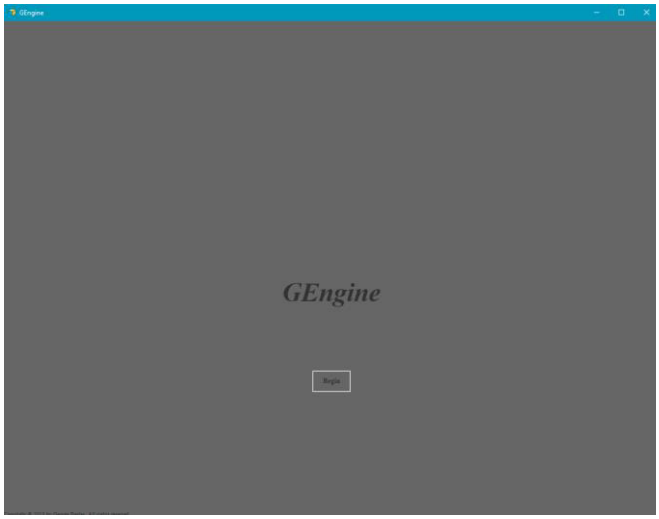
The window opens once the code has been executed.

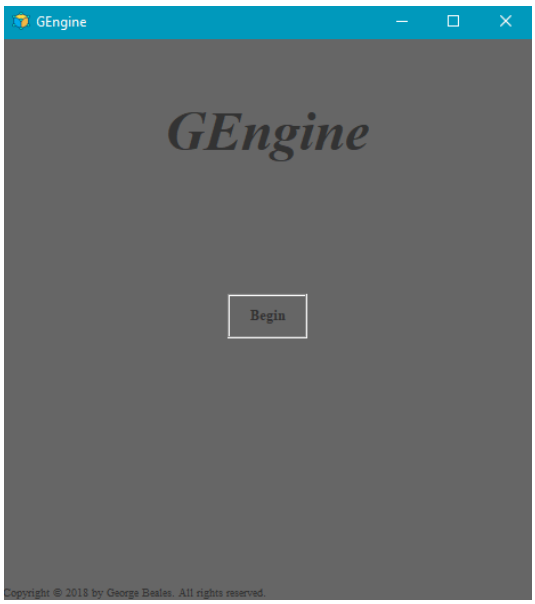
Test 1.2



Here you can see the program has run and when closed it builds the project and makes me aware that it has closed, this is hard to screenshot.

Test 1.3



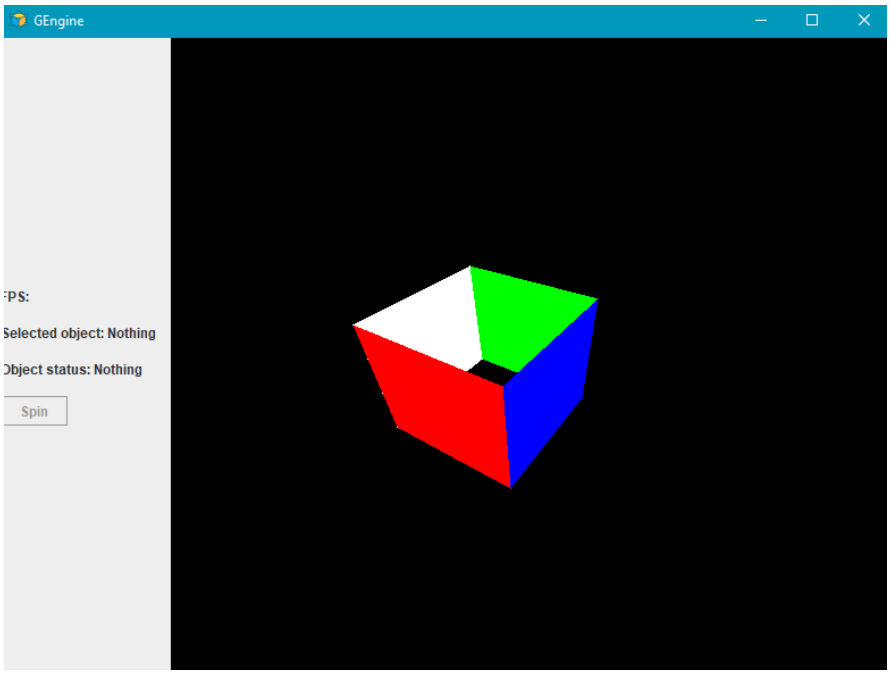


Here is the window being resized to two different sizes and it shows that the program remains intact and working

Test 1.4

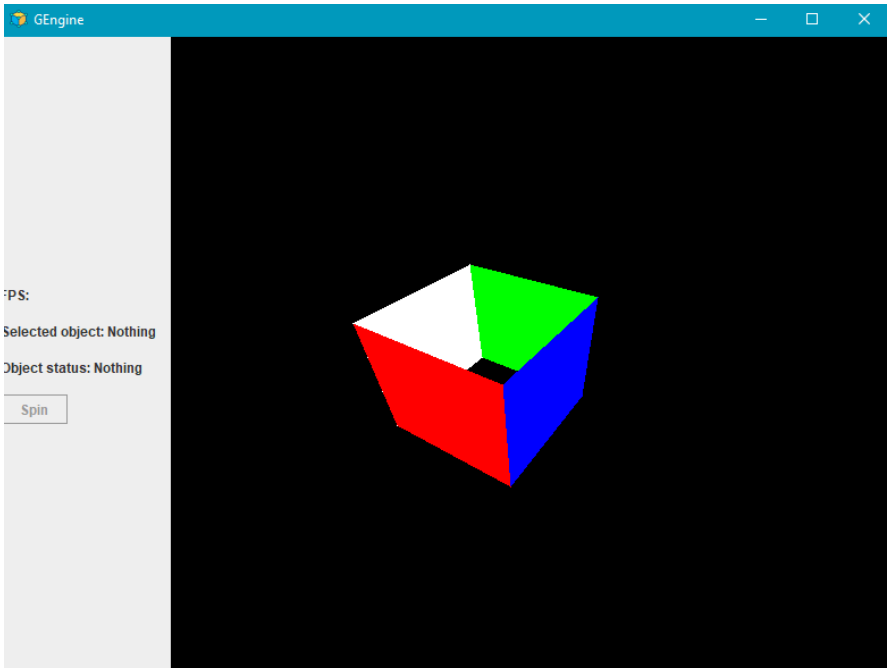
There appears to be no visual anomalies or bugs with placement or existence of objects in the panel.

Test 2.1



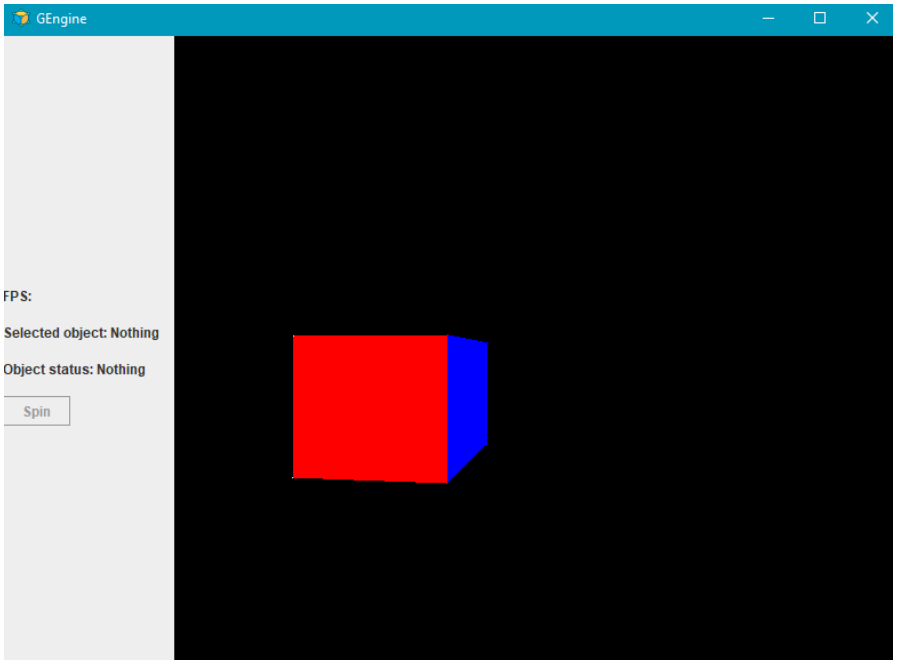
The object although not triangular looking does render properly in 3d space.

Test 2.2



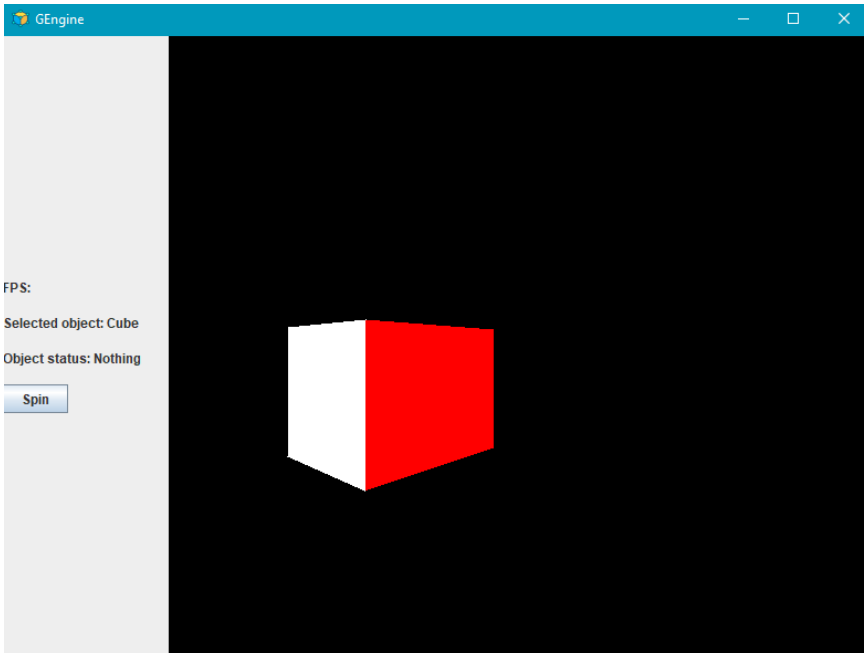
The Z-buffer does work because the depth of the coloured sides looks to be correctly placed based on where the camera is in the world.

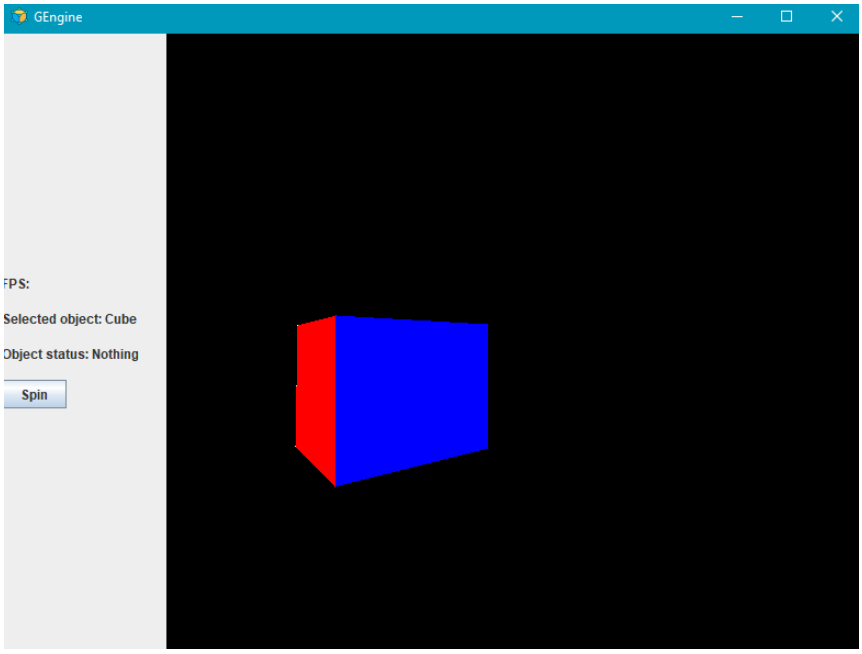
Test 2.3



Here the object's blue side does appear to get smaller in size close in the x viewport plane to look as if it is 3D and gives off the illusion of depth.

Test 2.4





In the screenshots the camera remains in the same position, but the cube is spinning, and this is visible due to the fact that you can see different sides after time.

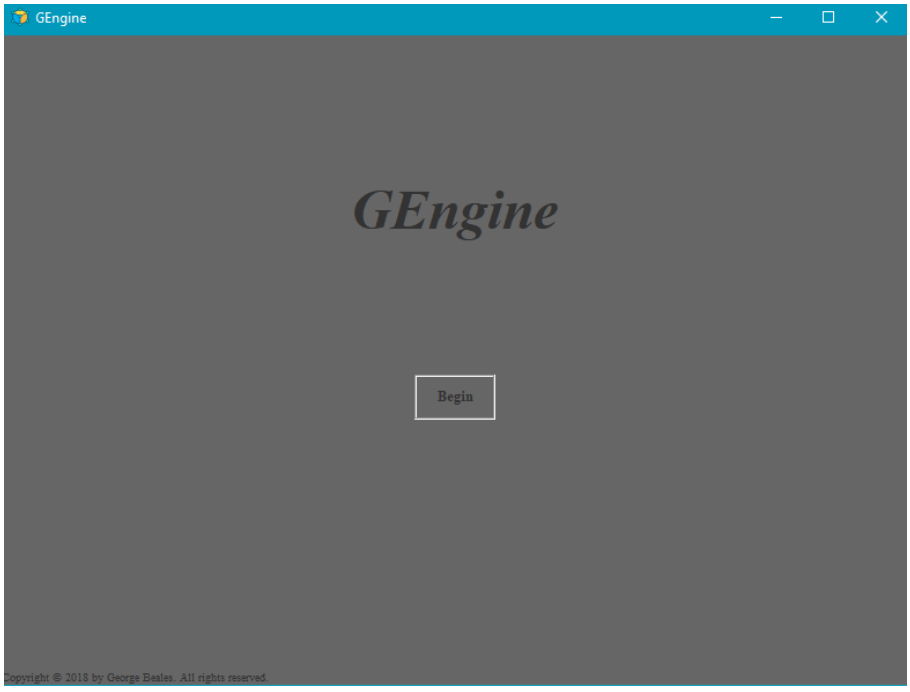
Tests 3.1 – 3.5

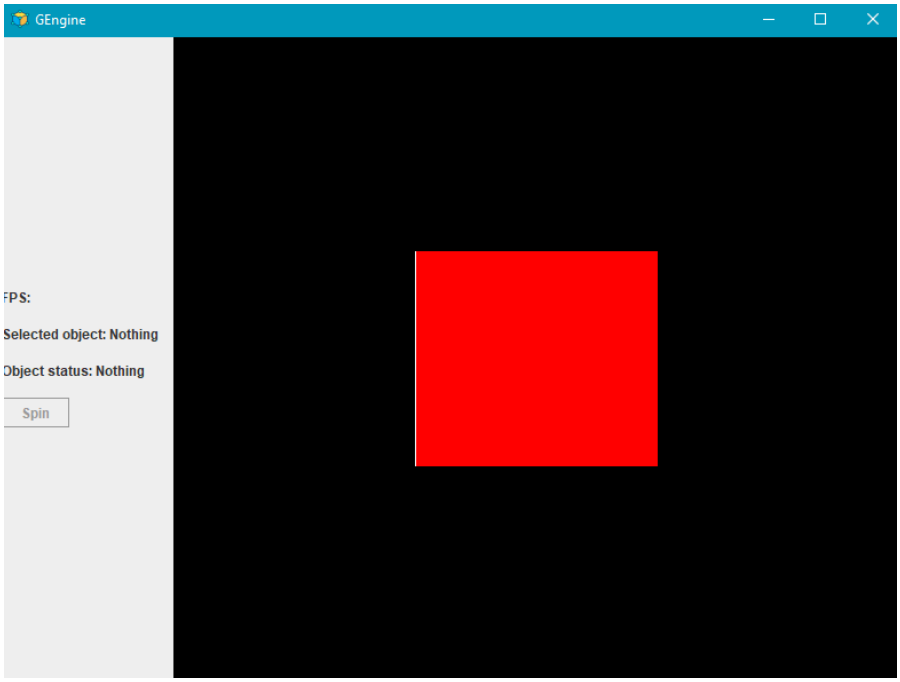
There is no visual aid for these tests not only because they fail but because they were never able to be implemented into the final code. I am to mention this in the evaluation in more detail.

Test 4.1

Although the buttons are not placed in the way of the original design choices the button placement is what I chose after I altered from the original design.

Test 4.2





Although there is only one menu navigation button it does work correctly, screenshot 1 being the menu and screenshot 2 being where the button leads to.

Test 4.3

There is no visual aid for this test because the test can't take place without the settings menu existing, which it doesn't so it technically doesn't fail but all the buttons that are in its place work so I am using that as reference.

Test 4.4

This shows how I opted to use a simple grey background instead of a 3d panning panel as the technicalities could not be worked out in time for the project finalisation.

Test 5.1 and 5.2

There once again is no visual aid for this test as there are no other menus or settings menus as due to design changes later during the project which mean I have nulled the tests. I'd say that the buttons work that are taking their place, so I give it a fail with consideration.

Post-development testing

Earlier on I mentioned that after my program was finalised, I would get a random person interested to test it for me. I have now decided that it is better and easier for me to test the program as I can explain all the occurrences and what I did instead of and so on.

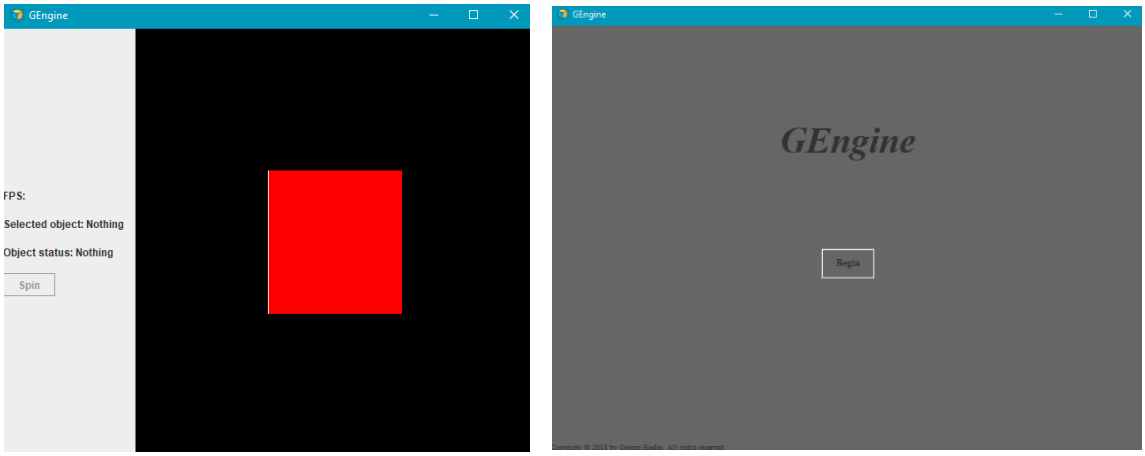
Here is what I will check:

I. Go through the program and go to the options menu to see if there is anything I would like to change.
II. Go to the demos menu and try out all the demos.
III. Make sure to try every demo at least once.
IV. Buttons test
V. Keyboard spam program break test
VI. Demos test
VII. Not doing what the demos say test

Test I, II and III

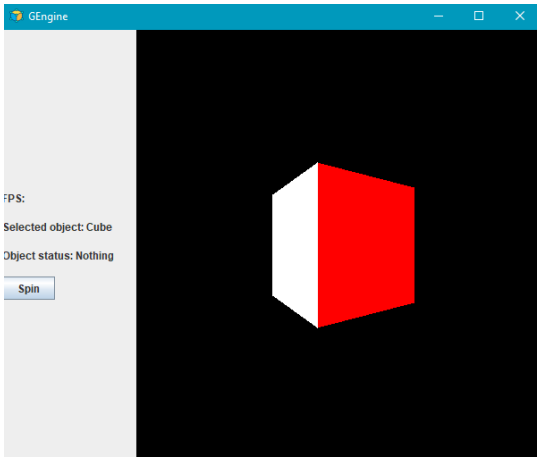
There is no visual aid for this test as these tests can not take place because the feature was either not added or cannot be completed without the prior.

Test IV



In the two screenshots above I have two screenshots of the program after (left) and before (right) for when I click the begin button.

Result: The begin button works and takes the user to the correct menu. The spin button also works because the spin button is greyed out when no object is selected and once it is selected it enables and spins and like in this picture below.



Test V

As you can see on the image, I am on the main menu. I have tried spamming lots of keys on my keyboard and in fact most keys on the keyboard to check for any reaction from the program.

Result: The program did not react in any way to random buttons being pressed by the user. This is good because it means that the program is considered stable and no mistakes can occur when using the program.

Test VI and VII

These tests once again contain no visual aid as this feature was not added to the program and can therefore not be tested on.

Code

Program structure

- genginev2
  - GEnginev2.java
  - Interface.java
  - Listeners.java
  - Menu.java
  - Renderer.java
- genginev2.ObjectInformation
  - Camera.java
  - Mat3x3.java
  - Mat4x4.java
  - Mesh.java
  - ObjectData.java
  - Triangle.java
  - Vertex.java

Program code

GEnginev2.java

```
1.  package genginev2;
2.
3.  import genginev2.ObjectInformation.*;
4.  import java.awt.Color;
5.  import java.util.concurrent.ExecutorService;
6.  import java.util.concurrent.Executors;
7.
8.  /*
9.   * @author George
10.  */
11. public class GEngineV2 {
12.
13.     public static void main(String[] args) {
14.         //Main GUI class initialisation.
15.         Interface window = new Interface();
16.         //Object data class initialisation for holding information about all
17.         //objects.
18.         ObjectData objectData = new ObjectData();
19.         //Renderer class initialisation for handling the object information and
20.         //rendering with now useless fov information.
21.         Renderer renderer = new Renderer(objectData, window, 45, 1, 150, 1);
22.         //Putting the renderer panel in the main interface window in the game
23.         //panel.
24.         window.gameP.add(renderer);
25.         //Button and key listener class initialisation for handling the key and
26.         //button inputs.
27.         Listeners input = new Listeners(window, renderer, objectData);
28.         //Creating a new thread to repeatededly cause focus in the renderer
29.         //window to make sure multiple inputs can be detected.
30.         ExecutorService service = Executors.newFixedThreadPool(1);
31.         service.submit(new Runnable() {
32.             //Infinite loop to focus the renderer window at all times.
33.             public void run() {
34.                 while(1>0){
35.                     renderer.requestFocusInWindow();
36.                     renderer.repaint();
```

```
37.                 }
38.
39.             }
40.         });
41.
42.         //Test code.
43.
44.         //Manual creation of objects inputting raw co-ordinates for creation.
45.         //Cube example.
46.         objectData.inputMesh(new Mesh("Cube", renderer));
47.         objectData.getMeshs().get(0).inputTriangle(new Triangle(
48.             new Vertex(-25, -25, -25),
49.             new Vertex(-25, -25, 25),
50.             new Vertex(-25, 25, 25),
51.             Color.WHITE));
52.         objectData.getMeshs().get(0).inputTriangle(new Triangle(
53.             new Vertex(-25, 25, 25),
54.             new Vertex(-25, 25, -25),
55.             new Vertex(-25, -25, -25),
56.             Color.WHITE));
57.         objectData.getMeshs().get(0).inputTriangle(new Triangle(
58.             new Vertex(-25, -25, 25),
59.             new Vertex(25, -25, 25),
60.             new Vertex(25, 25, 25),
61.             Color.RED));
62.         objectData.getMeshs().get(0).inputTriangle(new Triangle(
63.             new Vertex(25, 25, 25),
64.             new Vertex(-25, 25, 25),
65.             new Vertex(-25, -25, 25),
66.             Color.RED));
67.         objectData.getMeshs().get(0).inputTriangle(new Triangle(
68.             new Vertex(25, -25, -25),
69.             new Vertex(25, -25, 25),
70.             new Vertex(25, 25, 25),
71.             Color.BLUE));
72.         objectData.getMeshs().get(0).inputTriangle(new Triangle(
73.             new Vertex(25, 25, 25),
74.             new Vertex(25, 25, -25),
75.             new Vertex(25, -25, -25),
76.             Color.BLUE));
77.         objectData.getMeshs().get(0).inputTriangle(new Triangle(
78.             new Vertex(-25, -25, -25),
79.             new Vertex(25, -25, -25),
80.             new Vertex(25, 25, -25),
81.             Color.GREEN));
82.         objectData.getMeshs().get(0).inputTriangle(new Triangle(
83.             new Vertex(25, 25, -25),
84.             new Vertex(-25, 25, -25),
85.             new Vertex(-25, -25, -25),
86.             Color.GREEN));
87.         objectData.getMeshs().get(0).setWorldSpace(0, 0, -100, 0, 0, 0);
88.
89.         //Pyramid example
90.         /*objectData.inputMesh(new Mesh("Pyramid", renderer));
91.         objectData.getMeshs().get(0).inputTriangle(new Triangle(
92.             new Vertex(-25, -25, -25),
93.             new Vertex(0, 25, 0),
94.             new Vertex(25, -25, -25),
95.             Color.WHITE));
96.         objectData.getMeshs().get(0).inputTriangle(new Triangle(
97.             new Vertex(-25, -25, 25),
98.             new Vertex(0, 25, 0),
99.             new Vertex(25, -25, 25),
100.                Color.RED));
101.            objectData.getMeshs().get(0).inputTriangle(new Triangle(
102.                new Vertex(-25, -25, -25),
```

```
                    new Vertex(0, 25, 0),
                    new Vertex(-25, -25, 25),
                    Color.BLUE));
                objectData.getMeshs().get(0).inputTriangle(new Triangle(
                    new Vertex(25, -25, -25),
                    new Vertex(0, 25, 0),
                    new Vertex(25, -25, 25),
                    Color.GREEN));
                objectData.getMeshs().get(0).setWorldSpace(0, 0, -100, 0, 0, 0);*/
            }
        }
```

Interface.java

```
package genginev2;

import java.awt.*;
import javax.swing.*;

/**
 *
 * @author George
 */
public class Interface extends javax.swing.JFrame{

    //Component declaration.
    public JPanel mainP;
    public Menu menuP;
    public JButton menuB1;
    public JLabel menuL1;
    public JPanel gameP;
    public JPanel gameP1;
    public JLabel gameL1;
    public JLabel gameL2;
    public JLabel gameL3;
    public JLabel gameL4;
    public JLabel gameL5;
    public JLabel gameL6;
    public JButton gameB1;

    public Interface(){
        initComponents();
    }

    private void initComponents(){
        //Component creation.
        mainP = new JPanel();
        menuP = new Menu();
        gameP = new JPanel();
        gameP1 = new JPanel();
        gameL1 = new JLabel();
        gameL2 = new JLabel();
        gameL3 = new JLabel();
        gameL4 = new JLabel();
        gameL5 = new JLabel();
        gameL6 = new JLabel();
        gameB1 = new JButton();

        //Main window properties.
        setSize(800,600);
        setVisible(true);
        setTitle("GEngine");
        setIconImage(new ImageIcon("cube.png").getImage());
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

```

```
52.         //Main window sorting.
53.         add(mainP);
54.         
55.         //Component layouts.
56.         mainP.setLayout(new CardLayout());
57.         gameP.setLayout(new BoxLayout(gameP,BoxLayout.X_AXIS));
58.         gameP1.setLayout(new BoxLayout(gameP1,BoxLayout.Y_AXIS));
59.         
60.         //Constraints.
61.         
62.         //Component sorting.
63.         mainP.add(menuP, "Menu");
64.         mainP.add(gameP, "Game");
65.         gameP.add(gameP1);
66.         gameP1.add(gameL1);
67.         gameP1.add(gameL2);
68.         gameP1.add(gameL3);
69.         gameP1.add(gameL4);
70.         gameP1.add(gameL5);
71.         gameP1.add(gameL6);
72.         gameP1.add(gameB1);
73.         
74.         //Component properties.
75.         mainP.setVisible(true);
76.         mainP.setSize(getContentPane().getSize());
77.         gameP1.setPreferredSize(new Dimension(150,600));
78.         gameP1.setVisible(true);
79.         gameL1.setText("FPS: ");
80.         gameL2.setText(" ");
81.         gameL3.setText("Selected object: Nothing");
82.         gameL4.setText(" ");
83.         gameL5.setText("Object status: Nothing");
84.         gameL6.setText(" ");
85.         gameB1.setText("Spin");
86.         gameB1.setEnabled(false);
87.     }
88. }
```

Listeners.java

```
1.  package genginev2;
2.  
3.  import genginev2.ObjectInformation.*;
4.  import java.awt.*;
5.  import java.awt.event.*;
6.  
7.  /**
8.   *
9.   * @author George
10.  */
11. public class Listeners {
12.  
13.     //All objects that require a listener or need listener information.
14.     private Interface window;
15.     private Renderer renderer;
16.     private ObjectData objectData;
17.     private String selectedObject;
18.     //Constructor takes all the windows and their component's information.
19.     public Listeners(Interface window, Renderer renderer, ObjectData objectData){
20.         this.window = window;
21.         this.renderer = renderer;
22.         this.objectData = objectData;
23.         //Call initListeners method that starts all the listeners.
24.         initListeners();
25.     }
```

```
26.
27.     private void initListeners(){
28.         //Menu button listener.
29.         window.menuP.menuB1.addActionListener(new ActionListener(){
30.             //Overriding the actionPerformed method of ActionListener to add my
31.             //own commands for when the actionPerformed command is triggered by
32.             //the listener.
33.             @Override
34.             public void actionPerformed(ActionEvent e){
35.                 //Changing the current shown panel to the main panel instead of
36.                 //the menu panel.
37.                 CardLayout card = (CardLayout) window.mainP.getLayout();
38.                 card.show(window.mainP, "Game");
39.             }
40.         });
41.
42.         window.gameB1.addActionListener(new ActionListener(){
43.             //Overriding the actionPerformed method of ActionListener to add my
44.             //own commands for when the actionPerformed command is triggered by
45.             //the listener.
46.             @Override
47.             public void actionPerformed(ActionEvent e){
48.                 //For loop that goes through all the meshes in the mesh array
49.                 //to look for the selected object.
50.                 for(Mesh i : objectData.getMeshs()){
51.                     if(i.getMeshName() == selectedObject){
52.                         //If the selected object is found the object has a spin
53.                         //method which spins the object on its axis.
54.                         i.Spin();
55.                     }
56.                 }
57.             }
58.         });
59.
60.         renderer.addMouseListener(new MouseListener(){
61.             //Overriding the mouse listener for the renderer panel to add mouse
62.             //actions as listeners with my own functions.
63.             @Override
64.             public void mouseClicked(MouseEvent e){
65.                 //When the mouse is clicked information about where on the panel
66.                 //the mouse clicked is given to me in co-ordinates.
67.
68.                 //x and y co-ordinates.
69.                 int x = e.getX();
70.                 int y = e.getY();
71.                 //If the cursor co-ordinates are taken up by an object rather
72.                 //than null then it give true. The array that contains this data
73.                 //is contained in the objectData object which stores this
74.                 //information from the rasterizer.
75.                 if(objectData.getMouseBuffer()[y * renderer.getSize().width + x]
76.                         != null){
77.                     //Getting the name of the selected object in the array.
78.                     selectedObject =
79.                         objectData.getMouseBuffer()
80.                         [y * renderer.getSize().width + x].getMeshName();
81.                     //Setting the textbox next to the selected object text to
82.                     //the object that is selected.
83.                     window.gameL3.setText("Selected object: " +
84.                             objectData.getMouseBuffer()
85.                             [y * renderer.getSize().width + x].getMeshName());
86.                     //Making the spin button in the game panel clickable because
87.                     //an object is selected.
88.                     window.gameB1.setEnabled(true);
89.                 }
90.                 else{
91.                     //If no object is selected the button is not clickable and
```

```
92.                     //the selected object text is set to nothing.
93.                     window.gameB1.setEnabled(false);
94.                     window.gameL3.setText("Selected object: " + "Nothing");
95.                 }
96.             }
97.             //Overriding the other actions performed by the mouse but not giving
98.             //them commands to make the actions do nothing rather than
99.             //something unpredictable.
100.                    @Override
101.                    public void mousePressed(MouseEvent e) {
102.                    }
103.                    @Override
104.                    public void mouseReleased(MouseEvent e) {
105.                    }
106.                    @Override
107.                    public void mouseEntered(MouseEvent e) {
108.                    }
109.                    @Override
110.                    public void mouseExited(MouseEvent e) {
111.                    }
112.                });
113.
114.                //Adding a key listener to the renderer to take keyboard input whils
    t
115.                //the renderer panel is in focus.
116.                renderer.addKeyListener(new KeyListener(){
117.                    //Overriding the actions performed when pressing a button but on
    ly
118.                    //using the keyPressed action because this is the only one I wan
    t to
119.                    //use. This is to make the useless actions not to anything
120.                    //unpredictable.
121.                    @Override
122.                    public void keyTyped(KeyEvent e) {}
123.                    @Override
124.                    public void keyReleased(KeyEvent e) {}
125.                    @Override
126.                    public void keyPressed(KeyEvent e) {
127.                        //Getting the code of the key pressed.
128.                        int keyCode = e.getKeyCode();
129.                        //Switch statement for quick comparison and use of what butt
    on
130.                        //is pressed by the user.
131.                        switch( keyCode ) {
132.                            case KeyEvent.VK_W:
133.                                //If the W key is pressed the command buttonW is use
    d on
134.                                //the camera. This is the move forward button for th
    e 3D
135.                                //engine.
136.                                renderer.camera.buttonW(1);
137.                                break;
138.                            case KeyEvent.VK_S:
139.                                //If the S key is pressed the command buttonS is use
    d on
140.                                //the camera. This is the move backward button for t
    he
141.                                //3D engine.
142.                                renderer.camera.buttonS(1);
143.                                break;
144.                            case KeyEvent.VK_A:
145.                                //If the A key is pressed the command buttonA is use
    d on
146.                                //the camera. This is the move to the left button fo
    r
147.                                //the 3D engine.
```

```
148.                                    renderer.camera.buttonA(1);
149.                                    break;
150.                                case KeyEvent.VK_D :
151.                                    //If the D key is pressed the command buttonD is use
    d on
152.                                    //the camera. This is the move to the right button f
    or
153.                                    //the 3D engine.
154.                                    renderer.camera.buttonD(1);
155.                                    break;
156.                                case KeyEvent.VK_Z :
157.                                    //If the Z button is pressed it moves the camera dow
    n in
158.                                    //the 3D engine.
159.                                    renderer.camera.changeWorldSpace(0, -
    1, 0, 0, 0, 0);
160.                                    break;
161.                                case KeyEvent.VK_Q :
162.                                    //If the Q button is pressed it moves the camera up
    in
163.                                    //3D engine.
164.                                    renderer.camera.changeWorldSpace(0, 1, 0, 0, 0, 0);

165.                                    break;
166.                                case KeyEvent.VK_LEFT:
167.                                    //If the left arrow key button is pressed the camera

168.                                    //turns left in the y axis to look left.
169.                                    renderer.camera.changeWorldSpace(0, 0, 0, 0, -
    1, 0);
170.                                    break;
171.                                case KeyEvent.VK_RIGHT:
172.                                    //If the right arrow key button is pressed the camer
    a
173.                                    //turns right in the y axis to look right.
174.                                    renderer.camera.changeWorldSpace(0, 0, 0, 0, 1, 0);

175.                                    break;
176.                                case KeyEvent.VK_UP:
177.                                    //If the up arrow key button is pressed the camera t
    urns
178.                                    //up in the x axis.
179.                                    renderer.camera.changeWorldSpace(0, 0, 0, -
    1, 0, 0);
180.                                    break;
181.                                case KeyEvent.VK_DOWN:
182.                                    //If the down arrow key button is pressed the camera

183.                                    //turns down in the x axis.
184.                                    renderer.camera.changeWorldSpace(0, 0, 0, 1, 0, 0);

185.                                    break;
186.                                }
187.                            renderer.repaint();
188.                        }
189.                    });
190.            }
191.        }
```

Menu.java

```
1.  package genginev2;
2.
3.  /**
4.   *
```

```
 * @author George
 */
public class Menu extends javax.swing.JPanel {

    /**
     * Creates new form Menu
     */
    //Contructor for the main panel which is the main menu in the program. I
    //used the built in GUI editor in NETBEANS to achieve this class so most of
    //the things generated in this class is by the IDE.
    public Menu() {
        //Initialising the components in the class.
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-
fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        menuB1 = new javax.swing.JButton();
        menuL1 = new javax.swing.JLabel();
        menuL2 = new javax.swing.JLabel();

        setBackground(new java.awt.Color(102, 102, 102));

        menuB1.setBackground(new java.awt.Color(204, 204, 204));
        menuB1.setFont(new java.awt.Font("Times New Roman", 1, 12)); // NOI18N
        menuB1.setText("Begin");
        menuB1.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        menuB1.setContentAreaFilled(false);
        menuB1.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));

        menuL1.setFont(new java.awt.Font("Times New Roman", 3, 48)); // NOI18N
        menuL1.setText("GEngine");

        menuL2.setFont(new java.awt.Font("Times New Roman", 0, 10)); // NOI18N
        menuL2.setText("Copyright © 2018 by George Beales. All rights reserved.");

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addComponent(menuL2)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VA
LUE))
            .addGroup(layout.createSequentialGroup()
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VA
LUE)
                .addComponent(menuB1, javax.swing.GroupLayout.PREFERRED_SIZE, 70, j
avax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VA
LUE))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequ
entialGroup()
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VA
LUE)
                .addComponent(menuL1)
```

```
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addContainerGap(162, Short.MAX_VALUE)
                .addComponent(menuL1)
                .addGap(114, 114, 114)
                .addComponent(menuB1, javax.swing.GroupLayout.PREFERRED_SIZE, 39, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(216, 216, 216)
                .addComponent(menuL2))
        );

        menuB1.setFocusable(false);
    }// </editor-fold>


    // Variables declaration - do not modify
    public javax.swing.JButton menuB1;
    private javax.swing.JLabel menuL1;
    private javax.swing.JLabel menuL2;
    // End of variables declaration
}
```

Renderer.java

```
package genginev2;

import genginev2.ObjectInformation.*;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.util.List;
import javax.swing.JPanel;

/**
 *
 * @author George
 */
public class Renderer extends JPanel{
    //All the attributes of the renderer class with their uses described later
    //in the code when they are used.
    public ObjectData data;
    public Interface window;
    public Camera camera;
    private int iHeight;
    private int iWidth;
    private int cHeight;
    private int cWidth;
    private int xTrans;
    private int yTrans;
    private Mat4x4 projectionMat;
    private double raw1[];
    private double raw2[];
    private double raw3[];
    private double screen1[];
    private double screen2[];
    private double screen3[];
    private double ndc1[];
    private double ndc2[];
    private double ndc3[];
```

```
37.     private double raster1[];
38.     private double raster2[];
39.     private double raster3[];
40.     private double[] zBuffer;
41.     private Mesh[] mouseBuffer;
42.     public BufferedImage img;
43.
44.     //Constructor of the renderer class.
45.     public Renderer(ObjectData data, Interface window, double fov, double near, dou
    ble far, double aspect){
46.         //The objectData object and Window object are taken from the
47.         //initialisation in the main class to use and access information about
48.         //them. The other attributes aren't actually used in the final version
49.         //but are information about how the camera and render should work if the
50.         //projection matrix was ever added.
51.         this.data = data;
52.         this.window = window;
53.         //Creating and unused projection matrix that contains the matrix for
54.         //projection.
55.         //projectionMat = new Mat4x4();
56.         //Creating the camera object in the renderer class.
57.         camera = new Camera();
58.         //setting the size of the panel to be 500 wide and 500 tall, this
59.         //information will be used lator on.
60.         setSize(500,500);
61.         //Setting the values for an unused projection matrix.
62.         //setProjectionMat(fov, near, far, aspect);
63.     }
64.
65.     //An unused piece of code on creating a 4x4 matrix for projection.
66.     /*private void setProjectionMat(double fov, double near, double far, double asp
    ect){
67.         double q = 1 / Math.tan(Math.toRadians(fov)/2);
68.         double a = q / aspect;
69.         double b = (far+near)/(near-far);
70.         double c = (2*far*near)/(near-far);
71.         projectionMat.grid[0][0] = a; projectionMat.grid[0][1] = 0; projectionMat.g
    rid[0][2] = 0; projectionMat.grid[0][3] = 0;
72.         projectionMat.grid[1][0] = 0; projectionMat.grid[1][1] = q; projectionMat.g
    rid[1][2] = 0; projectionMat.grid[1][3] = 0;
73.         projectionMat.grid[2][0] = 0; projectionMat.grid[2][1] = 0; projectionMat.g
    rid[2][2] = b; projectionMat.grid[2][3] = c;
74.         projectionMat.grid[3][0] = 0; projectionMat.grid[3][1] = 0; projectionMat.g
    rid[3][2] = -1; projectionMat.grid[3][3] = 0;
75.     }*/
76.     //Overriding the paint method to change it and add my own commands for
77.     //graphical manipulation.
78.     @Override
79.     public void paint (Graphics g) {
80.         super.paintComponent(g);
81.         //Creating the Graphics2D object for use when drawing to the panel.
82.         Graphics2D g2 = (Graphics2D) g;
83.         //Setting the background colour to black.
84.         setBackground(Color.black);
85.         //Getting accurate height information about the window when the window
86.         //is open on the computer.
87.         iHeight = getSize().height;
88.         iWidth = getSize().width;
89.         //Getting the co-ordinates of the origins by division.
90.         xTrans = iWidth / 2;
91.         yTrans = iHeight / 2;
92.         //Setting the vieport window size.
93.         cHeight = 2;
94.         cWidth = 2;
95.         //Creating the image buffer with the size of the window for later use
96.         //with the z-buffer and object selection.
```

```
97.         BufferedImage img = new BufferedImage(iWidth, iHeight,
98.                 BufferedImage.TYPE_INT_ARGB);
99.         //Creating an array with size the same amount as the amount of pixels.
100.                double[] zBuffer = new double[iWidth * iHeight];
101.                //Creating an array with size the same amount as the amount of pixel
    s.
102.                Mesh[] mouseBuffer = new Mesh[iWidth * iHeight];
103.                for(int e=0; e < zBuffer.length; e++){
104.                    //Setting every value in the array to be equal to as much as
105.                    //negative infinity for use with depth.
106.                    zBuffer[e] = Double.NEGATIVE_INFINITY;
107.                }
108.                for(int e=0; e < mouseBuffer.length; e++){
109.                    //Setting every value in the array to be equal to as much as
110.                    //negative infinity for use with object selection.
111.                    mouseBuffer[e] = null;
112.                }
113.
114.                //For loop that renders every mesh available from the objectData cla
    ss.
115.                for(Mesh e : data.getMeshs()){
116.                    //For loop that renders every triangle in the mesh array.
117.                    for(Triangle i : camera.getTrianglesInCameraSpace(
118.                            e.getTrianglesInWorld())){
119.                        //Creating a local variable equal to the vertices list of th
    e
120.                        //specific triangle.
121.                        List<Vertex> vertices = i.getVertices();
122.                        Vertex v1 = vertices.get(0);
123.                        Vertex v2 = vertices.get(1);
124.                        Vertex v3 = vertices.get(2);
125.                        //Setting the vertices to local variables as the raw vertice
    s of
126.                        //the triangle.
127.                        raw1 = v1.getVertex();
128.                        raw2 = v2.getVertex();
129.                        raw3 = v3.getVertex();
130.
131.                        //Convert points to projected screen space.
132.                        double[] screen1 = {(raw1[0]/-raw1[2]), (raw1[1]/-
    raw1[2]),
133.                            raw1[2]};
134.                        double[] screen2 = {(raw2[0]/-raw2[2]), (raw2[1]/-
    raw2[2]),
135.                            raw2[2]};
136.                        double[] screen3 = {(raw3[0]/-raw3[2]), (raw3[1]/-
    raw3[2]),
137.                            raw3[2]};
138.
139.                        //Screen space to NDC space.
140.                        double[] ndc1 = {((screen1[0] + (cWidth/2)) / cWidth),
141.                            ((screen1[1] + (cHeight/2)) / cHeight), screen1[2]};
142.                        double[] ndc2 = {((screen2[0] + (cWidth/2)) / cWidth),
143.                            ((screen2[1] + (cHeight/2)) / cHeight), screen2[2]};
144.                        double[] ndc3 = {((screen3[0] + (cWidth/2)) / cWidth),
145.                            ((screen3[1] + (cHeight/2)) / cHeight), screen3[2]};
146.
147.                        //NDC space to raster space.
148.                        double[] raster1 = {Math.floor(ndc1[0] * iWidth),
149.                            Math.floor((1-ndc1[1]) * iHeight), ndc1[2]};
150.                        double[] raster2 = {Math.floor(ndc2[0] * iWidth),
151.                            Math.floor((1-ndc2[1]) * iHeight), ndc2[2]};
152.                        double[] raster3 = {Math.floor(ndc3[0] * iWidth),
153.                            Math.floor((1-ndc3[1]) * iHeight), ndc3[2]};
154.
155.                        //Rasterizer.
```

```
156.                        //Boxing the triangle to save of processing power of the min
    imum
157.                        //area the triangle could be located.
158.                        int minX = (int) Math.max(0, Math.ceil(
159.                                Math.min(raster1[0], Math.min(raster2[0], raster3[0]
    ))));
160.                        int maxX = (int) Math.min(img.getWidth() - 1,
161.                                Math.floor(Math.max(raster1[0],
162.                                Math.max(raster2[0], raster3[0]))));
163.                        int minY = (int) Math.max(0, Math.ceil(
164.                                Math.min(raster1[1], Math.min(raster2[1], raster3[1]
    ))));
165.                        int maxY = (int) Math.min(img.getHeight() - 1,
166.                                Math.floor(Math.max(raster1[1],
167.                                Math.max(raster2[1], raster3[1]))));
168.
169.                        //Computing the triangle area using the triangle vertices.
170.                        double triangleArea = (raster1[1] - raster3[1]) *
171.                                (raster2[0] - raster3[0]) + (raster2[1] -
     raster3[1]) *
172.                                (raster3[0] - raster1[0]);
173.
174.                        //Going through every pixel in the box that the triangle can
     be
175.                        //in.
176.                        for(int x = minX; x<= maxX; x++){
177.                            for(int y = minY; y<= maxY; y++){
178.                                //Barymetric co-
    ordinate system to figure out which
179.                                //pixels are in the triangle and which are not.
180.                                double b1 = ((y - raster3[1]) *
181.                                        (raster2[0] - raster3[0]) +
182.                                        (raster2[1] - raster3[1]) * (raster3[0] -
     x)) /
183.                                        triangleArea;
184.                                double b2 = ((y - raster1[1]) *
185.                                        (raster3[0] - raster1[0]) +
186.                                        (raster3[1] - raster1[1]) * (raster1[0] -
     x)) /
187.                                        triangleArea;
188.                                double b3 = ((y - raster2[1]) *
189.                                        (raster1[0] - raster2[0]) +
190.                                        (raster1[1] - raster2[1]) * (raster2[0] -
     x)) /
191.                                        triangleArea;
192.
193.                                if (b1 >= 0 && b1 <= 1 && b2 >= 0 && b2 <= 1 &&
194.                                        b3 >= 0 && b3 <= 1) {
195.                                    //If the pixel is in the triangle it sets the de
    pth
196.                                    //variable equal to where the pixel is in 3D spa
    ce.
197.                                    double depth = b1 * raster1[2] + b2 * raster2[2]
     +
198.                                            b3 * raster3[2];
199.                                    //If the pixel is closer than something else in
    the
200.                                    //z-buffer array it replaces it.
201.                                    if(zBuffer[y * img.getWidth() + x] < depth){
202.                                        //Sets the pixel in the buffered image to be
     at
203.                                        //the computed co-
    ordinates with the triangle's
204.                                        //colour.
205.                                        img.setRGB((x), (y), i.getColor().getRGB());
```

```
206.                                        //Setting the new z-buffer value for the
207.                                        //z-buffer array.
208.                                        zBuffer[y * img.getWidth() + x] = depth;
209.                                        //Ssetting the new mouse buffer value for th
    e
210.                                        //mouse buffer array.
211.                                        mouseBuffer[y * img.getWidth() + x] = e;
212.
213.                                    }
214.                                }
215.                            }
216.                        }
217.                        //Gives the new information into the objectData object.
218.                        data.inputMouseBuffer(mouseBuffer);
219.                        //Draws the image to the panel after all the objects have be
    en
220.                        //rendered.
221.                        g2.drawImage(img, 0, 0, null);
222.                    }
223.                }
224.            }
225.        }
```

Camera.java

```
1.  package genginev2.ObjectInformation;
2.
3.  import java.util.ArrayList;
4.  import java.util.List;
5.
6.  /**
7.   *
8.   * @author George
9.   */
10. public class Camera {
11.     //All the attributes of the camera class with their uses described later in
12.     //the code when they are used.
13.     private List<Triangle> trianglesInCamera;
14.     private List<Triangle> trianglesInWorld;
15.     private Vertex position;
16.     private double xTranslation;
17.     private double yTranslation;
18.     private double zTranslation;
19.     private double xRotation;
20.     private double yRotation;
21.     private double zRotation;
22.     private Mat4x4 rotMatX;
23.     private Mat4x4 rotMatY;
24.     private Mat4x4 rotMatZ;
25.     private Mat4x4 transMat;
26.     private Mat4x4 transMatI;
27.
28.     //Camera class constructor.
29.     public Camera(){
30.         //Setting the default position of the camera to the origin.
31.         position = new Vertex(0,0,0);
32.         //Creating a bunch of 4x4 matrices to compute the calculations but as
33.         //empty matrices that will later be added to.
34.         trianglesInCamera = new ArrayList<>();
35.         rotMatX = new Mat4x4();
36.         rotMatY = new Mat4x4();
37.         rotMatZ = new Mat4x4();
38.         transMat = new Mat4x4();
39.         transMatI = new Mat4x4();
40.         //Setting the defaults translations and rotation of the camera to
```

```
41.         //nothing.
42.         xTranslation = 0;
43.         yTranslation = 0;
44.         zTranslation = 0;
45.         xRotation = 0;
46.         yRotation = 0;
47.         zRotation = 0;
48.     }
49.     //Method to set the default position of an object.
50.     public void setWorldSpace(double xTranslation, double yTranslation, double zTra
    nslation, double xRotation, double yRotation, double zRotation){
51.         this.xTranslation = xTranslation;
52.         this.yTranslation = yTranslation;
53.         this.zTranslation = zTranslation;
54.         //Changing the values if they are changed in the called method and
55.         //converting them to radians which is the used angle measurement in
56.         //java.
57.         if(xRotation != 0){
58.             this.xRotation = xRotation;
59.             xRotation = Math.toRadians(xRotation);
60.         }
61.         if(yRotation != 0){
62.             this.yRotation = yRotation;
63.             yRotation = Math.toRadians(yRotation);
64.         }
65.         if(zRotation != 0){
66.             this.zRotation = zRotation;
67.             zRotation = Math.toRadians(zRotation);
68.         }
69.     }
70.     //Method to change the default position of an object.
71.     public void changeWorldSpace(double xTranslationC, double yTranslationC, double
     zTranslationC, double xRotationC, double yRotationC, double zRotationC){
72.         //Adding the changed values to the original set values.
73.         xTranslation += xTranslationC;
74.         yTranslation += yTranslationC;
75.         zTranslation += zTranslationC;
76.         //If the values are higher than a full rotation of 2 pi then the angles
77.         //are changed to be new value minus 2 pi.
78.         if(xRotationC != 0){
79.             xRotation += Math.toRadians(xRotationC);
80.             if(xRotation > 2 * Math.PI){
81.                 xRotation -= 2 * Math.PI;
82.             }
83.         }
84.         if(yRotationC != 0){
85.             yRotation += Math.toRadians(yRotationC);
86.             if(yRotation > 2 * Math.PI){
87.                 yRotation -= 2 * Math.PI;
88.             }
89.         }
90.         if(zRotationC != 0){
91.             zRotation += Math.toRadians(zRotationC);
92.             if(zRotation > 2 * Math.PI){
93.                 zRotation -= 2 * Math.PI;
94.             }
95.         }
96.     }
97.     //Method to fetch the triangles are in relation to the camera after the
98.     //camera has moved.
99.     public List<Triangle> getTrianglesInCameraSpace(List<Triangle> trianglesIn){
100.             //Fetching the triangles vertex values from the triangles array in t
    he
101.             //objectData class.
102.             trianglesInWorld = trianglesIn;
103.
```

```
104.                //Creating matrices with the rotation values that have been set or
105.                //changed in the previous methods. See reference in the documentatio
    n
106.                //for explanation into how these matrices work.
107.                rotMatX.grid[0][0] = 1; rotMatX.grid[0][1] = 0; rotMatX.grid[0][2] =
     0; rotMatX.grid[0][3] = 0;
108.                rotMatX.grid[1][0] = 0; rotMatX.grid[1][1] = Math.cos(xRotation); ro
    tMatX.grid[1][2] = -Math.sin(xRotation); rotMatX.grid[1][3] = 0;
109.                rotMatX.grid[2][0] = 0; rotMatX.grid[2][1] = Math.sin(xRotation); ro
    tMatX.grid[2][2] = Math.cos(xRotation); rotMatX.grid[2][3] = 0;
110.                rotMatX.grid[3][0] = 0; rotMatX.grid[3][1] = 0; rotMatX.grid[3][2] =
     0; rotMatX.grid[3][3] = 1;
111.
112.                rotMatY.grid[0][0] = Math.cos(yRotation); rotMatY.grid[0][1] = 0; ro
    tMatY.grid[0][2] = Math.sin(yRotation); rotMatY.grid[0][3] = 0;
113.                rotMatY.grid[1][0] = 0; rotMatY.grid[1][1] = 1; rotMatY.grid[1][2] =
     0; rotMatY.grid[1][3] = 0;
114.                rotMatY.grid[2][0] = -
    Math.sin(yRotation); rotMatY.grid[2][1] = 0; rotMatY.grid[2][2] = Math.cos(yRotatio
    n); rotMatY.grid[2][3] = 0;
115.                rotMatY.grid[3][0] = 0; rotMatY.grid[3][1] = 0; rotMatY.grid[3][2] =
     0; rotMatY.grid[3][3] = 1;
116.
117.                rotMatZ.grid[0][0] = Math.cos(zRotation); rotMatZ.grid[0][1] = -
    Math.sin(zRotation); rotMatZ.grid[0][2] = 0; rotMatZ.grid[0][3] = 0;
118.                rotMatZ.grid[1][0] = Math.sin(zRotation); rotMatZ.grid[1][1] = Math.
    cos(zRotation); rotMatZ.grid[1][2] = 0; rotMatZ.grid[1][3] = 0;
119.                rotMatZ.grid[2][0] = 0; rotMatZ.grid[2][1] = 0; rotMatZ.grid[2][2] =
     1; rotMatZ.grid[2][3] = 0;
120.                rotMatZ.grid[3][0] = 0; rotMatZ.grid[3][1] = 0; rotMatZ.grid[3][2] =
     0; rotMatZ.grid[3][3] = 1;
121.
122.                transMat.grid[0][0] = 1; transMat.grid[0][1] = 0; transMat.grid[0][2
    ] = 0; transMat.grid[0][3] = xTranslation;
123.                transMat.grid[1][0] = 0; transMat.grid[1][1] = 1; transMat.grid[1][2
    ] = 0; transMat.grid[1][3] = yTranslation;
124.                transMat.grid[2][0] = 0; transMat.grid[2][1] = 0; transMat.grid[2][2
    ] = 1; transMat.grid[2][3] = zTranslation;
125.                transMat.grid[3][0] = 0; transMat.grid[3][1] = 0; transMat.grid[3][2
    ] = 0; transMat.grid[3][3] = 1;
126.
127.                transMatI.grid[0][0] = 1; transMatI.grid[0][1] = 0; transMatI.grid[0
    ][2] = 0; transMatI.grid[0][3] = -xTranslation;
128.                transMatI.grid[1][0] = 0; transMatI.grid[1][1] = 1; transMatI.grid[1
    ][2] = 0; transMatI.grid[1][3] = -yTranslation;
129.                transMatI.grid[2][0] = 0; transMatI.grid[2][1] = 0; transMatI.grid[2
    ][2] = 1; transMatI.grid[2][3] = -zTranslation;
130.                transMatI.grid[3][0] = 0; transMatI.grid[3][1] = 0; transMatI.grid[3
    ][2] = 0; transMatI.grid[3][3] = 1;
131.
132.                //Resetting the trianglesInCamera array for every time this method i
    s
133.                //called.
134.                trianglesInCamera = new ArrayList<>();
135.                //For loop that goes through every triangles in the meshes and adds
    them
136.                //to a new variable.
137.                for(Triangle i : trianglesInWorld){
138.                    //Adding the triangles to a new array for copying purposes.
139.                    trianglesInCamera.add(new Triangle(
140.                            new Vertex(i.getVertices().get(0).getX(),
141.                                    i.getVertices().get(0).getY(),
142.                                    i.getVertices().get(0).getZ()),
143.                            new Vertex(i.getVertices().get(1).getX(),
144.                                    i.getVertices().get(1).getY(),
145.                                    i.getVertices().get(1).getZ()),
```

```
                            new Vertex(i.getVertices().get(2).getX(),
                                    i.getVertices().get(2).getY(),
                                    i.getVertices().get(2).getZ()),
                            i.getColor()));
                }
                //For loop that goes through the new copied array of triangles.
                for(Triangle i : trianglesInCamera){
                    //For loop that goes through every vertex in each triangle.
                    for(Vertex e : i.getVertices()){
                        //Applies all the matrix multiplications to each vertex.
                        double[] tI = transMatI.matMultiplyVector(e.getX(), e.getY()
    , e.getZ(), 1);
                        double[] rY = rotMatY.matMultiplyVector(tI[0], tI[1], tI[2],
     1);
                        double[] rX = rotMatX.matMultiplyVector(rY[0], rY[1], rY[2],
     1);
                        double[] rZ = rotMatZ.matMultiplyVector(rX[0], rX[1], rX[2],
     1);
                        //Sets the new co-ordinates of the vertices after all the
                        //transformations and translations and so on.
                        e.setCoords(rZ[0], rZ[1], rZ[2]);

                    }
                }
                return trianglesInCamera;
            }
            //All the key listener methods for moving the camera around in the world

            //whilst moving in a vector method. Each method moves in the direction
            //dependent on the rotation of the camera. This is explained in more det
    ail
            //in the documentation.
            public void buttonW(int val){
                changeWorldSpace(Math.sin(yRotation) * val, Math.sin(xRotation) * -
    val,
                        Math.cos(yRotation) * -val, 0,0,0);
            }
            public void buttonS(int val){
                changeWorldSpace(Math.sin(yRotation) * -
    val, Math.sin(xRotation) * val,
                        Math.cos(yRotation) * val, 0,0,0);
            }
            public void buttonA(int val){
                changeWorldSpace(Math.cos(yRotation) * -val, 0,
                        Math.sin(yRotation) * -val, 0,0,0);
            }
            public void buttonD(int val){
                changeWorldSpace(Math.cos(yRotation) * val, 0,
                        Math.sin(yRotation) * val, 0,0,0);
            }
        }
```

Mat3x3.java

```
package genginev2.ObjectInformation;

/**
 *
 * @author George
 */
public class Mat3x3 {
    //Attribute for the values of the matrix in a 2d array for easier
    //understanding.
    public int[][] grid;
    //Constructor for 3x3 matrix.
```

```
    public Mat3x3(){
        grid = new int[3][3];
    }
}
```

Mat4x4.java

```
package genginev2.ObjectInformation;

/**
 *
 * @author George
 */
public class Mat4x4 {
    //Attribute for the values of the matrix in a 2d array for easier
    //understanding.
    public double[][] grid;
    //Attribute for easier vector notation when multiplying the matrix by a
    //vector.
    private double[] vector;
    //Constructor for the 4x4 matrix.
    public Mat4x4(){
        grid = new double[4][4];
        vector = new double[4];
    }
    //Method for returning the matrix values.
    public double[][] matMultiplyMat(){
        return grid;
    }
    //Method for multiplying the matrix by a vector and returning in vector
    //form.
    public double[] matMultiplyVector(double x, double y, double z, double w){
        //Simple matrix/vector multiplication explained more in the
        //documentation.
        vector[0] = (x * grid[0][0]) + (y * grid[0][1]) + (z * grid[0][2]) + (w * g
rid[0][3]);
        vector[1] = (x * grid[1][0]) + (y * grid[1][1]) + (z * grid[1][2]) + (w * g
rid[1][3]);
        vector[2] = (x * grid[2][0]) + (y * grid[2][1]) + (z * grid[2][2]) + (w * g
rid[2][3]);
        vector[3] = (x * grid[3][0]) + (y * grid[3][1]) + (z * grid[3][2]) + (w * g
rid[3][3]);
        return vector;
    }
}
```

Mesh.java

```
package genginev2.ObjectInformation;

import java.awt.Color;
import java.util.ArrayList;
import java.util.List;
import genginev2.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

/*
 * @author George
 */
public class Mesh {
    //All the attributes of the camera class with their uses described later in
    //the code when they are used.
    private String name;
```

```
17.     private List<Triangle> triangles;
18.     private List<Triangle> trianglesInWorld;
19.     private List<Vertex> vertices;
20.     private List<Vertex> verticesInWorld;
21.     private double xTranslation;
22.     private double yTranslation;
23.     private double zTranslation;
24.     private double xRotation;
25.     private double yRotation;
26.     private double zRotation;
27.     private Mat4x4 rotMatX;
28.     private Mat4x4 rotMatY;
29.     private Mat4x4 rotMatZ;
30.     private Mat4x4 transMat;
31.     public boolean spinning;
32.     private Renderer renderer;
33.     private ExecutorService service;
34.
35.     //Constructor for the Mesh class.
36.     public Mesh(String name, Renderer renderer){
37.         //Taking in a name and the renderer panel the mesh is in to create the
38.         //mesh.
39.         this.name = name;
40.         this.renderer = renderer;
41.         //Creating the lists which contain the triangles and vertices of each
42.         //mesh.
43.         triangles = new ArrayList<>();
44.         vertices = new ArrayList<>();
45.         verticesInWorld = new ArrayList<>();
46.         //Creating blank matrices for the rotation matrix calculations and
47.         //translation matrixs.
48.         rotMatX = new Mat4x4();
49.         rotMatY = new Mat4x4();
50.         rotMatZ = new Mat4x4();
51.         transMat = new Mat4x4();
52.         //Setting all the default translation and rotation values to 0.
53.         xTranslation = 0;
54.         yTranslation = 0;
55.         zTranslation = 0;
56.         xRotation = 0;
57.         yRotation = 0;
58.         zRotation = 0;
59.         //Adding blank vertices to the verticesInWorld list to help with later
60.         //calculations.
61.         verticesInWorld.add(new Vertex(0,0,0));
62.         verticesInWorld.add(new Vertex(0,0,0));
63.         verticesInWorld.add(new Vertex(0,0,0));
64.         //Setting the default spinning method to 0 so it does not spin by
65.         //default.
66.         spinning = false;
67.
68.     }
69.     //Method for adding triangles to the mesh to create an object. Each mesh has
70.     //multiple triangles making up a full object.
71.     public void inputTriangle(Triangle inputTriangle){
72.         //Adding the triangle to the triangle list.
73.         triangles.add(inputTriangle);
74.     }
75.     //Method for setting the default world space of the object.
76.     public void setWorldSpace(double xTranslation, double yTranslation, double zTra
    nslation, double xRotation, double yRotation, double zRotation){
77.         //Take the values and set them to the values in the class.
78.         this.xTranslation = xTranslation;
79.         this.yTranslation = yTranslation;
80.         this.zTranslation = zTranslation;
81.         //If the value is unchanged do not alter the default values set in the
```

```
82.         //constructor. Also convesion to radians as this is how java handles
83.         //angle units.
84.         if(xRotation != 0){
85.             this.xRotation = xRotation;
86.             xRotation = Math.toRadians(xRotation);
87.         }
88.         if(yRotation != 0){
89.             this.yRotation = yRotation;
90.             yRotation = Math.toRadians(yRotation);
91.         }
92.         if(zRotation != 0){
93.             this.zRotation = zRotation;
94.             zRotation = Math.toRadians(zRotation);
95.         }
96.     }
97.     //Method for changing the position and rotation of the mesh in the world.
98.     public void changeWorldSpace(double xTranslationC, double yTranslationC, double
     zTranslationC, double xRotationC, double yRotationC, double zRotationC){
99.         xTranslation += xTranslationC;
100.                yTranslation += yTranslationC;
101.                zTranslation += zTranslationC;
102.                //If the value is unchanged do not alter the default values set in t
    he
103.                //set function. Also convesion to radians as this is how java handle
    s
104.                //angle units.
105.                if(xRotationC != 0){
106.                    xRotation += Math.toRadians(xRotationC);
107.                    if(xRotation > 2 * Math.PI){
108.                        xRotation -= 2 * Math.PI;
109.                    }
110.                }
111.                if(yRotationC != 0){
112.                    yRotation += Math.toRadians(yRotationC);
113.                    if(yRotation > 2 * Math.PI){
114.                        yRotation -= 2 * Math.PI;
115.                    }
116.                }
117.                if(zRotationC != 0){
118.                    zRotation += Math.toRadians(zRotationC);
119.                    if(zRotation > 2 * Math.PI){
120.                        zRotation -= 2 * Math.PI;
121.                    }
122.                }
123.            }
124.            //Method for fetching the triangles' information after transformations a
    nd
125.            //rotations. This is called by the camera to receieve information on whe
    re
126.            //every triangle is when it needs to render.
127.            public List<Triangle> getTrianglesInWorld(){
128.                //Creating matrices with the rotation values that have been set or
129.                //changed in the previous methods. See reference in the documentatio
    n
130.                //for explanation into how these matrices work.
131.                rotMatX.grid[0][0] = 1; rotMatX.grid[0][1] = 0; rotMatX.grid[0][2] =
     0; rotMatX.grid[0][3] = 0;
132.                rotMatX.grid[1][0] = 0; rotMatX.grid[1][1] = Math.cos(xRotation); ro
    tMatX.grid[1][2] = -Math.sin(xRotation); rotMatX.grid[1][3] = 0;
133.                rotMatX.grid[2][0] = 0; rotMatX.grid[2][1] = Math.sin(xRotation); ro
    tMatX.grid[2][2] = Math.cos(xRotation); rotMatX.grid[2][3] = 0;
134.                rotMatX.grid[3][0] = 0; rotMatX.grid[3][1] = 0; rotMatX.grid[3][2] =
     0; rotMatX.grid[3][3] = 1;
135.
136.                rotMatY.grid[0][0] = Math.cos(yRotation); rotMatY.grid[0][1] = 0; ro
    tMatY.grid[0][2] = Math.sin(yRotation); rotMatY.grid[0][3] = 0;
```

```
137.                rotMatY.grid[1][0] = 0; rotMatY.grid[1][1] = 1; rotMatY.grid[1][2] =
    0; rotMatY.grid[1][3] = 0;
138.                rotMatY.grid[2][0] = -
   Math.sin(yRotation); rotMatY.grid[2][1] = 0; rotMatY.grid[2][2] = Math.cos(yRotatio
   n); rotMatY.grid[2][3] = 0;
139.                rotMatY.grid[3][0] = 0; rotMatY.grid[3][1] = 0; rotMatY.grid[3][2] =
    0; rotMatY.grid[3][3] = 1;
140.
141.                rotMatZ.grid[0][0] = Math.cos(zRotation); rotMatZ.grid[0][1] = -
   Math.sin(zRotation); rotMatZ.grid[0][2] = 0; rotMatZ.grid[0][3] = 0;
142.                rotMatZ.grid[1][0] = Math.sin(zRotation); rotMatZ.grid[1][1] = Math.
   cos(zRotation); rotMatZ.grid[1][2] = 0; rotMatZ.grid[1][3] = 0;
143.                rotMatZ.grid[2][0] = 0; rotMatZ.grid[2][1] = 0; rotMatZ.grid[2][2] =
    1; rotMatZ.grid[2][3] = 0;
144.                rotMatZ.grid[3][0] = 0; rotMatZ.grid[3][1] = 0; rotMatZ.grid[3][2] =
    0; rotMatZ.grid[3][3] = 1;
145.
146.                transMat.grid[0][0] = 1; transMat.grid[0][1] = 0; transMat.grid[0][2
   ] = 0; transMat.grid[0][3] = xTranslation;
147.                transMat.grid[1][0] = 0; transMat.grid[1][1] = 1; transMat.grid[1][2
   ] = 0; transMat.grid[1][3] = yTranslation;
148.                transMat.grid[2][0] = 0; transMat.grid[2][1] = 0; transMat.grid[2][2
   ] = 1; transMat.grid[2][3] = zTranslation;
149.                transMat.grid[3][0] = 0; transMat.grid[3][1] = 0; transMat.grid[3][2
   ] = 0; transMat.grid[3][3] = 1;
150.                //Resetting the trianglesInWorld array for every time this method is

151.                //called.
152.                trianglesInWorld = new ArrayList<>();
153.                //For loop that goes through every triangles in the meshes and adds
   them
154.                //to a new variable.
155.                for(Triangle i : triangles){
156.                    trianglesInWorld.add(new Triangle(
157.                            new Vertex(i.getVertices().get(0).getX(),
158.                                    i.getVertices().get(0).getY(),
159.                                    i.getVertices().get(0).getZ()),
160.                            new Vertex(i.getVertices().get(1).getX(),
161.                                    i.getVertices().get(1).getY(),
162.                                    i.getVertices().get(1).getZ()),
163.                            new Vertex(i.getVertices().get(2).getX(),
164.                                    i.getVertices().get(2).getY(),
165.                                    i.getVertices().get(2).getZ()),
166.                            i.getColor()));
167.                }
168.                //For loop that goes through the new copied array of triangles.
169.                for(Triangle i : trianglesInWorld){
170.                    //For loop that goes through every vertex in each triangle.
171.                    for(Vertex e : i.getVertices()){
172.                        //Applies all the matrix multiplications to each vertex.
173.                        double[] rX = rotMatX.matMultiplyVector(e.getX(), e.getY(),

174.                                e.getZ(), 1);
175.                        double[] rY = rotMatY.matMultiplyVector(rX[0], rX[1], rX[2],
    1);
176.                        double[] rZ = rotMatZ.matMultiplyVector(rY[0], rY[1], rY[2],
    1);
177.                        double[] t = transMat.matMultiplyVector(rZ[0], rZ[1], rZ[2],
    1);
178.                        //Sets the new co-ordinates of the vertices after all the
179.                        //transformations and translations and so on.
180.                        e.setCoords(t[0], t[1], t[2]);
181.                    }
182.                }
183.                return trianglesInWorld;
184.            }
```

```
185.            //Method for fetching the triangles' array for every triangle in the mes
    h.
186.            public List<Triangle> getTriangles(){
187.                return triangles;
188.            }
189.            //Method for fetching the mesh name.
190.            public String getMeshName(){
191.                return name;
192.            }
193.            //Method for spinning the mesh. This is only used for demonstration
194.            //purposes.
195.            public void Spin(){
196.                //Creating a new service which will run on a seperate thread.
197.                ExecutorService service = Executors.newFixedThreadPool(1);
198.                //Checks if the object is already spinning because this method is a

199.                //toggle.
200.                if(spinning == false){
201.                    spinning = true;
202.                    //Starts the thread to spin the mesh.
203.                    service.submit(new Runnable() {
204.                        public void run() {
205.                            //Loop that keeps changing the mesh's rotation value eve
    ry
206.                            //50 milliseconds.
207.                            while(service.isTerminated() == false){
208.                                try{
209.                                    Thread.sleep(50);
210.                                }
211.                                catch(InterruptedException e){
212.                                    e.printStackTrace();
213.                                }
214.                                changeWorldSpace(0, 0, 0, 0, 1, 0);
215.                            }
216.                        }
217.                    });
218.                }
219.                else{
220.                    //Confirmation for the spinning to stop.
221.                    spinning = false;
222.                    System.out.println("Stopping spin");
223.                    service.shutdownNow();
224.                }
225.            }
226.        }
```

ObjectData.java

```
1.  package genginev2.ObjectInformation;
2.
3.  import java.util.ArrayList;
4.  import java.util.List;
5.
6.  /**
7.   *
8.   * @author George
9.   */
10. public class ObjectData {
11.     //List that contains all the triangles in the scene.
12.     private List<Triangle> renderTriangles;
13.     //List that contains all the meshes in the scene.
14.     private List<Mesh> renderMeshs;
15.     //Buffer for the mouse to help with location of mouse clicks in the
16.     //listeners class.
17.     private Mesh[] mouseBuffer;
```

```
18.
19.     //Constructor that creates the lists for the objectData class.
20.     public ObjectData(){
21.         //List of all triangle objects and their information.
22.         renderTriangles = new ArrayList<>();
23.         //List of all mesh objects and their information.
24.         renderMeshs = new ArrayList<>();
25.         //Mouse buffer that contain information about where the mouse has
26.         //clicked on the screen.
27.         mouseBuffer = new Mesh[0];
28.     }
29.     //Method for inputting triangles into the objectData class and then into the
30.     //renderTriangles list.
31.     public void inputTriangle(Triangle inputTriangle){
32.         renderTriangles.add(inputTriangle);
33.     }
34.
35.     //Method for inputting the meshes into the objectData class and then into
36.     //the renderMeshs list.
37.     public void inputMesh(Mesh inputMesh){
38.         renderMeshs.add(inputMesh);
39.     }
40.
41.     //Method for inputting the mousebuffer variable information from the
42.     //renderer panel.
43.     public void inputMouseBuffer(Mesh[] mouseBuffer){
44.         this.mouseBuffer = mouseBuffer;
45.     }
46.
47.     //Method for fetching the triangles from the objectData class.
48.     public List<Triangle> getTriangles(){
49.         return renderTriangles;
50.     }
51.     //Method for fetching the meshs from the objectData class.
52.     public List<Mesh> getMeshs(){
53.         return renderMeshs;
54.     }
55.
56.     //Method for fetching the mousebuffer variable information from the render
57.     //panel.
58.     public Mesh[] getMouseBuffer(){
59.         return mouseBuffer;
60.     }
61. }
```

Triangle.java

```
1.  package genginev2.ObjectInformation;
2.
3.  import java.awt.Color;
4.  import java.util.ArrayList;
5.  import java.util.List;
6.  /**
7.   *
8.   * @author George
9.   */
10. public class Triangle {
11.     //Attribute for the vertices that this triangle class contains and their
12.     //information.
13.     private List<Vertex> vertices;
14.     //The colour of individual triangle.
15.     private Color color;
16.
17.     //Constructor for the triangle class which takes in 3 vertex information and
18.     //the desired colour for the triangle.
```

```
    public Triangle(Vertex v1,Vertex v2,Vertex v3, Color color){
        //Creating a blank list of vertices.
        vertices = new ArrayList<>();
        vertices.add(v1);
        vertices.add(v2);
        vertices.add(v3);
        this.color = color;
    }

    //Method for fetching the vertices list in this triangle class.
    public List<Vertex> getVertices(){
        return vertices;
    }

    //Method for fetching the colour of the triangle in this triangle class.
    public Color getColor(){
        return color;
    }
}
```

Vertex.java

```
package genginev2.ObjectInformation;

/**
 *
 * @author George
 */
public class Vertex {
    //Attributes for values for all of the co-ordinates of the vertex.
    private double x;
    private double y;
    private double z;
    private double w;

    //Constructor of the vertex that takes in all the co-ordinates of its
    //position and sets a default value of w to 1.
    public Vertex(double x, double y, double z){
        this.x = x;
        this.y = y;
        this.z = z;
        w = 1;
    }
    //Methods for fetching the individual vertices.
    public double getX(){
        return x;
    }
    public double getY(){
        return y;
    }
    public double getZ(){
        return z;
    }
    public double getW(){
        return w;
    }
    //Method for fetching all the vertex information in a double array.
    public double[] getVertex(){
        double[] localCoords = {x,y,z};
        return localCoords;
    }
    //Method for changing the vertex's co-ordinate values.
    public void setCoords(double x, double y, double z){
        this.x = x;
        this.y = y;
```

```
44.         this.z = z;
45.     }
46. }
```

Evaluation

Criteria comparison

Criteria	Were the criteria met?	Any further notes?
Make the basic window.	Yes	The basic window structure was achieved with use of built in java libraries.
Give the window a title.	Yes	
Give the window an icon.	Yes	I used and icon from a website that provides free icon images.
Size the window correctly.	Yes	I sized the window for best use on most monitors.
Create a working vertex system that can handle vertex changes.	Yes	I created a working vertex class system using OOP and it can handle vertex changes very easily with getters and setters.
Create a transformation matrix system that works with the vertex system.	Yes	My transformation matrix system seems to work exactly as I planned and without any issues. It did however have to issues a long the way when developing it.
Render simple triangular objects.	Yes	Rendering a simple triangular image was achieved easily and it does a good job of it. I was also able to render multiple triangular objects like I talked about in my design section.
Add a working Z-buffer that hides the appropriate objects.	Yes	Adding a working z-buffer was one of the tougher parts of the project but it works as intended and designed. The Z-buffer can successfully hide objects when its supposed to.
Make the system render in a perspective projection rather than orthographic.	Yes	Making the system render in a perspective projection was the hardest aspect of the entire project and required the most research. I did settle on an easier approach that required less matrix maths and seemed to do the job just as good for a simple project like this.
Add a working camera manipulation system.	Yes	The camera system was a system that took months to create and a bit longer to perfect. It now does work almost seamlessly and exactly how any new user would expect it to when using the program for the first time. Like I described earlier in the documentation I have set the keys as regular movement keys like in any other first person 3d game.
Make the physics demo	No	I couldn't achieve any of the demos in this project due to time constraints and difficulty. I originally planned to as a nice extra on top of the engine and a way to turn my engine into an educational tool. I decided later on in the development of my project that adding these demos would take away from the finishing touches on the main parts of the project that would have the most attention of the user.
Make the lighting demo	No	
Make the collision demo	No	
Make the object interaction demo	No	
Make the particle demo	No	

		Therefore, I believe these demos would have been a nice addition but unfortunately, I couldn't make them in time.
Add working buttons that lead to separate menus and panels.	Yes	The buttons in the project were some of the easier parts but getting them to be placed on the screen correctly was not. I eventually managed to place them in their correct planned places and gave them the correct functionality.
Add an options menu that can be accessed with the escape key.	No	As I said earlier about the demos the same applies to these objectives. These objects were simply too difficult to achieve in a reasonable timeframe and/ or I lacked the knowledge in order to create them. These are heavily link in with the demos so I will talk about all these combined criteria later.
Add a working demos menu.	No	
Create a working background 3d model camera pan in the menu.	No	

Further improvements

After finishing my project, I have some final ideas that I wish I could have added to my code. I also have some ideas that I already had but didn't have enough time or research in order to add to my code.

Clipping

A major part of game engines or 3d engines in general is that they can dynamically clip. What this means is to hide objects from being rendered when the camera can't see them. You can either do this by completely removed the objects that are not completely on the screen or cutting down objects into smaller triangles that do fit on the screen. This would have been a very nice addition to my engine but requires knowledge that either I can't obtain or wouldn't understand.

Demos

Another improvement I could make is to actually add the demos that I suggested I should earlier in the project. These demos could have elevated my project from simple object rendering to a game engine. Things like collision and lighting are so common in today's game that the average person has no real idea how hard it is to make let alone understand the theory behind it. Having demos would have not only made my engine more game worthy but it would have been a lot better for teaching people how the engine itself works. I feel that this would have been a very nice addition but its unfortunate I didn't have time to add it.

Mouse camera movement

Using the mouse to move the camera is considered "the way" to move a camera in a game. Although it is very common in 3d games in my code it would have been a very difficult task to add. The camera movement currently in place in my code is controlled with the arrow keys. This means that both hands have to be on the keyboard and it also means that it is hard to control the camera. Adding mouse support would make the user experience a lot better and make testing a lot easier.

3D Engine camera pan on the main menu

Adding a camera pan to the main menu was something I added to the design section of my project because I thought it would make a very good impression on the user and show the abilities of my engine before the user had even launched it properly. I also think this would look very visually appealing for a menu screen and would be like games that have a panning camera menu.

Conclusion

Overall, I think my project turned out to be something I like and something I found very useful for teaching me programming techniques and mathematical ideas. I may not have done everything I planned for or designed for, but I still think what I ended up creating was very visually appealing. I have mentioned what I should improve on and what didn't go well but to wrap up my project I am going to say that I think it was generally a success.

Bibliography

https://www.scratchapixel.com/ Accessed on 20/06/2018

https://netbeans.org/ Accessed on 20/06/2018

https://www.draw.io/ Accessed on 10/05/2019

https://www.flaticon.com/ Accessed 26/04/2019