

ST PAUL'S SCHOOL

# Vector Visualisation Software

---

A2 COMPUTING PROJECT

Louis de Beaumont

## Contents

Vector Visualisation Software .....	0
Glossary .....	0
1. Analysis .....	0
1.1 Background to and identification of the problem .....	1
1.2 Description of the current system .....	1
1.3 Identification of the prospective user(s) .....	1
1.4 Identification of user needs and acceptable limitations .....	2
1.5 Data sources and destinations .....	4
1.6 Analysis data dictionary and Data volumes .....	5
1.7 Data Flow Diagram .....	8
1.8 Object Analysis Diagram .....	10
1.9 Objectives for the proposed system .....	10
1.10 Realistic appraisal of the feasibility of potential systems .....	10
1.11 Justification of chosen solutions .....	11
1.12 Anaglyph type justification .....	11
1.13 Complexity of problem .....	12
2. Design .....	13
2.1 Overall system design .....	14
2.2 Description of modular system .....	16
2.3 GUI Rationale .....	20
2.4 Rendering Libraries .....	25
2.5 Class Structure .....	30
2.6 Data Validation .....	31
2.7 File Processing .....	32
2.8 Class fields and methods .....	34
2.9 Identification of Processes and Algorithms .....	102
2.10 Anaglyph Theory .....	103
2.11 Security .....	105
2.12 Test Strategy .....	105
3. System Testing .....	106
3.1 Design of test plan .....	107
3.2 Black box testing .....	107
3.3 White box testing .....	124

3.4 Stress Testing .....	134
4. System Maintenance .....	135
4.1 System Overview .....	136
4.2 Algorithm Design .....	138
5. User Manual .....	139
5.1 Introduction .....	141
5.2 Installation .....	141
5.3 Using the software .....	141
5.4 Troubleshooting .....	149
6. Appraisal.....	153
6.1 Project success .....	154
6.2 User feedback .....	155
6.3 Analysis of user feedback .....	157
6.4 Possible extensions .....	158
7. References .....	159
8. Code Listing.....	162
8.1 Angle.cs .....	163
8.2 ColorChangePanel.cs .....	165
8.3 ColoredObject.cs.....	168
8.4 InputAnglePanel.cs .....	169
8.5 InputLineFromPointsPanel.cs .....	171
8.6 InputLinePanel.cs.....	173
8.7 InputPanel.cs .....	176
8.8 InputPointPanel.cs .....	187
8.9 Line.cs.....	189
8.10 ObjectInspectorPanel.cs .....	190
8.11 Point.cs .....	193
8.12 Program.cs.....	194
8.13 Scene.cs .....	195

## Glossary

Vector: An entity containing magnitude and direction.

Position Vector: The relative (usually to the origin) position of a point.

Direction Vector: The vector defining the direction in which the line propagates.

Dot product: The function used to calculate the angle (in radians) between two lines.

Plane: A two dimensional surface with zero thickness, and infinite width and height.

Anaglyph: Producing the illusion of depth in an image by superposing two images of different colours, when used in conjunction with anaglyph glasses [sec. 1.12].

Anaglyph glasses: (Paper) Glasses with different coloured lenses which allow filtering of anaglyph images based on the colours used [sec. 1.12].

Camera: See sec. 1.6.8.

# 1. Analysis

---

### 1.1 Background to and identification of the problem

The Core 4, Further Pure 2, and Further Pure 3 AQA Maths modules all consist of problems with 3D vectors, including points, lines, and the angles between lines. Problems are largely based upon the student's ability to understand the 3D space, and to therefore produce the relevant equations from that.

However many students find it difficult to visualise 3D space and so struggle in both learning this part of the course, and in understanding and completing questions involving 3D vectors.

An example question would be:

Find the angle between the line  $(5\mathbf{i} + \mathbf{j} + 2\mathbf{k}) + s(\mathbf{i} + 2\mathbf{j} + \mathbf{k})$  and the vector between  $(3\mathbf{i}, 3\mathbf{j}, 3\mathbf{k})$  and  $(6\mathbf{i}, -2\mathbf{j}, 0\mathbf{k})$ .

Solving this question involves finding the direction vector between the two points, and then applying the dot product to the two direction vectors. This process is fairly simple when understood, but many students will not be able to see the sequence of operations immediately.

### 1.2 Description of the current system

In order to help the students visualise the vectors and angles in 3D space, the teacher will often use objects around the classroom to represent the vectors. However this means plotting single points and portraying angles proves difficult, due to physical limitations.

Current software solutions [ref. 1] include tools to plot coloured points, however the user cannot choose or change the colour of specific points. The user can also plot lines by defining starting and ending points, but cannot define the line by using a general equation of a line. The user cannot plot a line between two points which have already been defined, nor can he find or plot the angle between two lines. The software only supports points between -10 and 10 (in all directions), limiting its use.

Rotation and zoom camera [sec. 1.6.8] features are supported; however there is no panning feature. There is no support for 3D anaglyphs.

### 1.3 Identification of the prospective user(s)

Mr Rowland is a teacher at St. Paul's School who finds this a recurring problem in the course. The features of the software were determined by a series of formal meetings with Mr Rowland [sec. 1.4.1]. However not only would Mr Rowland find the program beneficial, but many other teachers throughout the department would too.

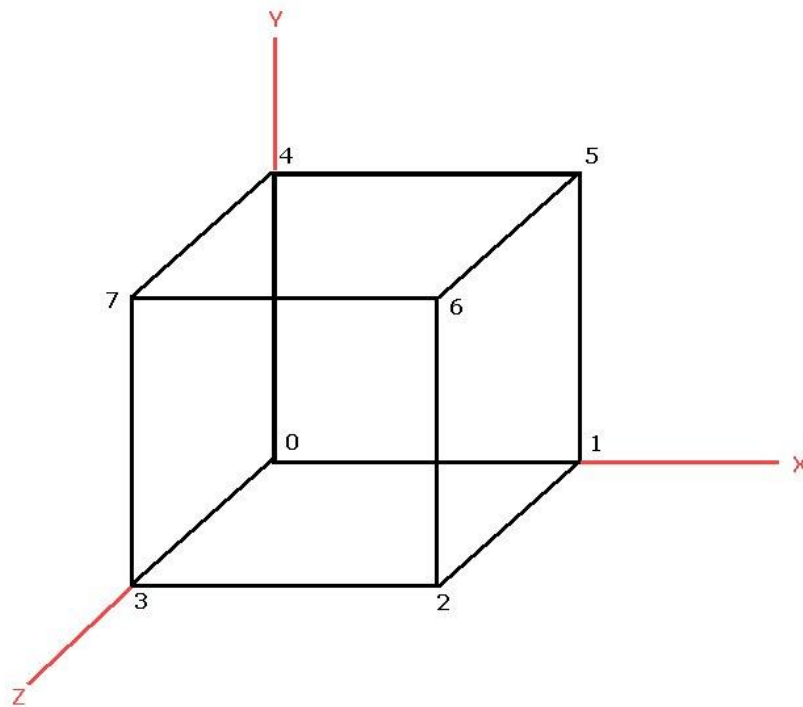
The minimum computer specification the software is to be run on matched the specifications of the computers used throughout the St Paul's School Maths department [sec. 1.4.2.9].

The teacher would also be able to display the program on the interactive whiteboard, making it easy to teach the whole class [sec. 1.4.2.9] at once.

## 1.4 Identification of user needs and acceptable limitations

### 1.4.1 Interview

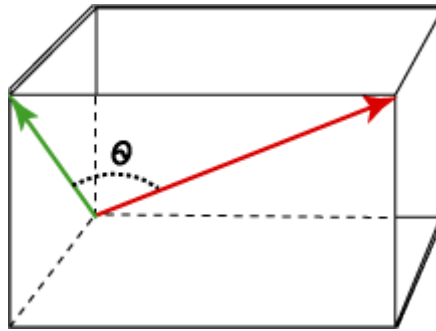
1. When asked about difficulties in teaching students the A2 Maths course Mr Rowland identified 3D vectors as an area many find difficult. He identified one of the main problems being a lack of ability to visualise 3D space, and thus not fully understanding the implications of the equations in practice (for example the dot product).
2. Mr Rowland explained that students often could not identify point coordinates relative to each other. For example:



[ref. 3]

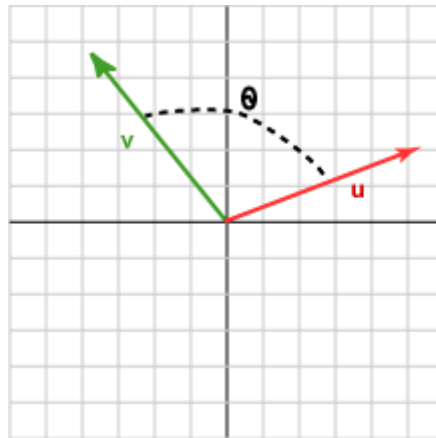
In this diagram many students would not be able to tell that points 5 and 6 have the same x and y coordinates, and only differ in z coordinate.

3. Students are required to calculate the angle between two lines. Mr Rowland explains that he is often asked how it is possible to have an angle in 3D.



[ref. 4]

He often uses a method of transposing the angle into the 2D plane to help the student see how the angle relates to the lines.



[ref. 5]

However Mr Rowland is reluctant to use this method as it encourages treating 3D vectors as 2D vectors. This causes confusion for weaker students.

4. Mr Rowland explains that problems involve multiple points, lines, and angles, but will rarely use above 15 objects in total, as the nature of questions is based around fewer objects.
5. Mr Rowland also says that a few students find it hard to use the 2D whiteboard to visualise 3D vectors.
6. Many teachers, including Mr Rowland, pre-plan lessons. This means that any software used needs to be suitable for pre-planned lessons.
7. Another important of the course is to be able to calculate the vector between two points. Mr Rowland says that this feature should be included for completeness, should a problem with it arise.
8. When asked about the maximum number of objects Mr Rowland would need to render, he said that it would be unlikely for him to have to draw more than 5 points, 5 lines, and 5 angles all at the same time.

#### 1.4.2 Identification of needs

1. The user needs a tool to help students visualise 3D vectors, points and angles in 3D space. The tool would help visualisation by rendering the objects, and allowing rotation around them. [sec. 1.4.1.1]



2. The user must be able to select two points and calculate and draw the vector between them. [sec. 1.4.1.7]
3. The software must be able to draw an angle between two pre-existing lines. This would reduce students' confusion on angles in 3D space. [sec. 1.4.1.3]
4. As multiple lines, points and angles may be rendered in a single scene, Mr Rowland requires a feature to select the colours of lines. This will stop confusion at points of intersection. [sec. 1.4.1.4]
5. Mr Rowland will need a way of managing (deleting, changing colour) objects in the scene. Therefore an inspector panel will be necessary. [sec. 1.4.1.4]
6. Mr Rowland also requires a tool to help students who find it difficult to visualise 3D space while facing a 2D whiteboard. This problem could be solved by the use of an Anaglyph. [sec. 1.4.1.5]
7. As Mr Rowland plans his lessons in advance he requires the ability to save a scene (containing 3D plots) and open it at a later stage. This would also allow problems to be solved over multiple lessons, and different classes. [sec. 1.4.1.6]
8. Mr Rowland is very unlikely to require over 20 objects to be rendered in one scene, so the program should be able to run smoothly at 20 objects.
9. The computers used throughout St Paul's School's Maths department are:
  - Windows 7 Professional [ref. 8]
  - Intel Core 2 Duo Processor [ref. 8]
  - 2.00 GB RAM [ref. 8]
  - 17" Dell Monitor [ref. 9]
  - DirectX 11 [ref. 8]
  - OpenGL 1.1 [ref. 7]

The whiteboard through which most of the class will see the software on has the following specifications:

  - 58 x 44 inch display.

The software must be able to run on these hardware specifications.

### 1.5 Data sources and destinations

Inputs with an \* after them must be manually inputted by the user.

Inputs:

1. Vector coordinates of a point in the form  $(x, y, z)$ . \*
2. Vector equation of a line in the form  $(x, y, z) + s(m, n, o)$ . \*
3. Selection of 2 lines with a button to calculate the angle between them.
4. Selection of 2 points with a button to create the line between them.
5. Selecting the colour of lines, points, and angles.
6. Key presses to allow rotation, panning and zooming of the camera.
7. A toggle for the 3D anaglyph.

Outputs:

1. Vector points, lines, and angles will be rendered on the screen.
2. Anaglyph rendering for red-cyan [sec. 1.11] glasses will be rendered when the 3D anaglyph is turned on.

## 1.6 Analysis data dictionary and Data volumes

### 1.6.1 Point

The point class, which contains all information about a point to be rendered in the scene.

#### 1. Data required from user:

- i. X, y, z coordinates of point
- ii. The user also has the ability to change the colour of the point once the point is created

#### 2. Data produced by program:

- i. Colour of point

#### 3. Validation:

- i. Inserting a non-numeric value as a coordinate will result in the program not accepting the inputted values, and not creating the point.

### 1.6.2 Line

The line class, which contains all information about a line to be rendered in the scene.

#### 1. Data required from user:

- i. Equation of line
- ii. Selection of two points and creating the line between them
- iii. The user also has the ability to change the colour of the line once the line is created

#### 2. Data produced by the program:

- i. Colour of line

#### 3. Validation:

- i. If the magnitude (in the equation of the line) is 0, the program will reject the inputted data and not create the line.
- ii. If the two selected points are the same, the program will reject the selection and not create the line.

### 1.6.3 Angle

The angle class, which contains all information about an angle to be rendered in the scene.

1. Data required from user:

- i. Selection of two lines and creating the angle between them
- ii. The user also has the ability to change the colour of the angle once the angle is created

2. Data produced by the program:

- i. Colour of angle

3. Validation:

- i. If the two selected lines are the same, the program will reject the selection and not create the angle.

#### 1.6.4 Scene

The OpenGL environment, in which all objects are rendered.

1. Data required from user:

- i. Points, lines, and angles
- ii. Whether anaglyph should be rendered
- iii. Key presses controlling the camera

2. Data produced by the program:

- i. Rendering the objects on the screen

3. Validation:

- i. None

#### 1.6.5 Main Panel

The main panel is the main user interface of the program.

Closing the main panel will result in the entire application terminating.

1. Data required from user:

- i. Selection of buttons [sec. 1.7.3]

2. Data produced by the program:

- i. None

3. Validation:

- i. None

### 1.6.6 Object Inspector

The object inspector displays a list of all objects in the scene, including information such as coordinates and colour.

1. Data required from user:

- i. Selection of objects, followed by a new colour or deleting the object

2. Data produced by the program:

- i. List of objects
- ii. List of colours

3. Validation:

- i. None

### 1.6.7 Anaglyph

1. Data required from user:

- i. Whether the anaglyph should be rendered or not

2. Data produced by the program:

- ii. New objects rendered to create perception of depth when viewed through red-cyan glasses

3. Validation:

- iii. None

### 1.6.8 Camera

The camera is the view point for the user – the objects of the scene as viewed by the camera is what is rendered on the scene.

1. Available transformations:

- i. Translation (including zooming)
- ii. Rotation

## 1.7 Data Flow Diagram

### 1.7.1 Existing Solution

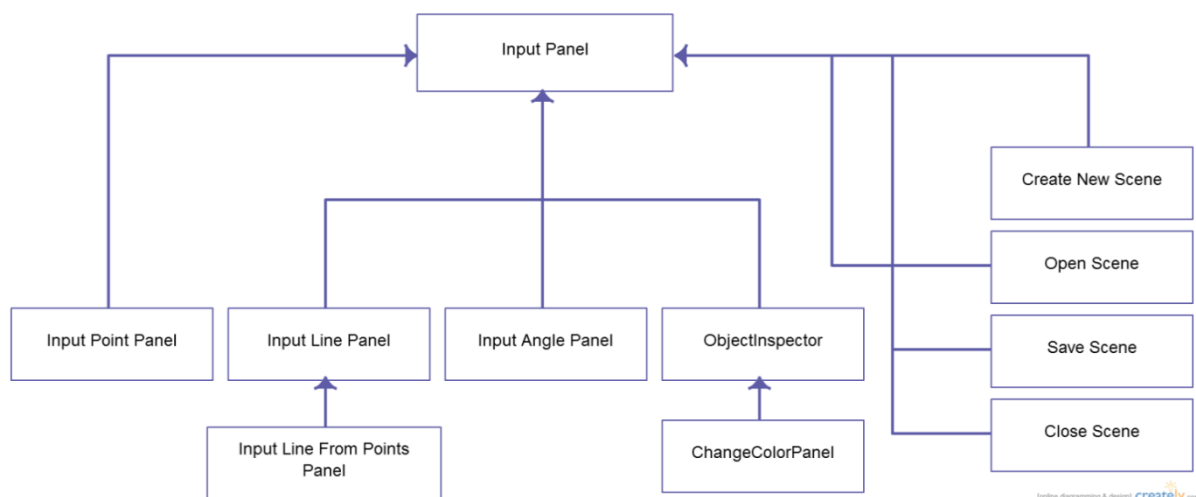
Some data flow options (inputting multiple points and vectors via a text box) have been excluded from the data flow diagram, as they caused the program to crash.



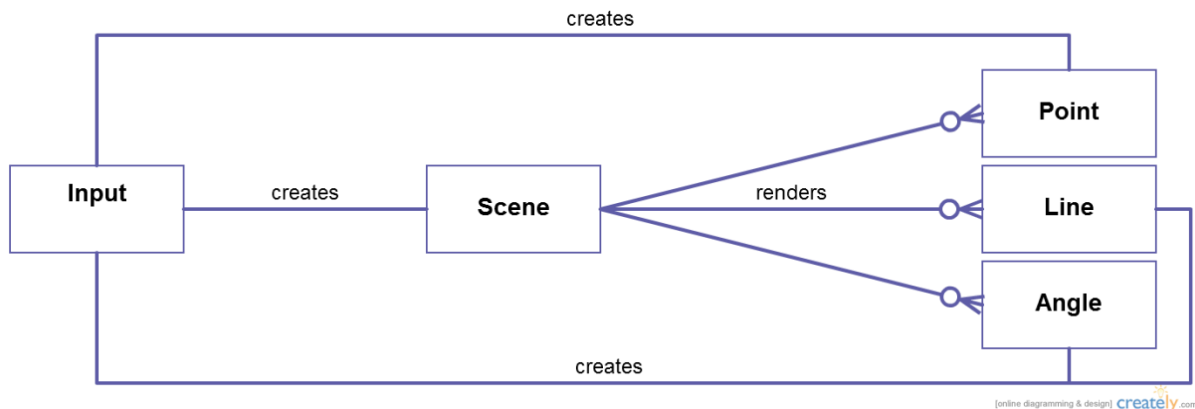
### 1.7.2 Proposed Solution



### 1.7.3 User Interface Data Flow



### 1.8 Object Analysis Diagram



### 1.9 Objectives for the proposed system

The software requirements were broken into two different categories: mathematical features, user interface features.

Mathematical features:

1. Be able to input co-ordinates,  $(x, y, z)$ , of a point in 3D space. [sec. 1.4.2.1]
2. Be able to input the equation of a 3D vector line in the form  $(a, b, c) + s(x, y, z)$ . [sec. 1.4.2.1]
3. Choose 2 different points and have a line drawn between them. [sec. 1.4.2.2]
4. Render the angle between 2 different lines. [sec. 1.4.2.3]

User interface features:

5. Separate panels to enter information for each of the above mathematical inputs.
6. The ability to select and change colours of points, lines, and angles. [sec. 1.4.2.4]
7. A panel to list all points, lines, and angles, offering the ability to delete them or change their colour. [sec. 1.4.2.5]
8. Toggle for 3D anaglyph. [sec. 1.4.2.6]
9. The ability to zoom, rotate and pan the camera, alongside the functionality to reset the camera to its original position. [sec. 1.4.2.1]
10. Options to create a new scene, close the current scene, open a new scene, and save the current scene. [sec. 1.4.2.7]

### 1.10 Realistic appraisal of the feasibility of potential systems

In order to render objects on the screen a 3D graphics library was required. I narrowed my choices down to two main options: DirectX and OpenGL. In narrowing my choices down to DirectX and OpenGL, I considered ease of installation, as the user already has both these libraries pre-installed. [sec. 1.4.2.9]

One of the problems Mr Rowland identified was that some students find 3D visualisation on a 2D whiteboard difficult [sec. 1.4.2.6]. This problem could be

solved in a number of ways. The first option, and one which already is in use, is to use classroom objects to help portray the lines. The second option is to use a 3D anaglyph [ref. 10] – a tool which renders objects multiple times, allowing a 3D effect to be created when used in conjunction with special glasses.

To code the program I was able to use my language of choice, C# - a primarily Windows only language (using the .NET framework), as all the computers the software would run on have Windows 7 installed. [sec. 1.4.2.9]

### 1.11 Justification of chosen solutions

I researched DirectX and OpenGL in more detail [ref. 6] to decide which one would be more suited to the project. DirectX is Windows only, and OpenGL is cross platform. As I am only developing for Windows, the versatility of the libraries did not matter. The performance of these libraries would also not be significant (as large amounts of data would not be rendered), and so I chose to use OpenGL based on previous experience with it. In particular I settled with FreeGlut, an open source OpenGL utility toolkit, alongside Tao, a free, open source wrapper for OpenGL. [ref. 11]

Mr Rowland already has attempted to use classroom objects to help students, however this causes multiple problems. Initially Mr Rowland may not have the objects available to him, and so this solution cannot be reliably used. On top of this, holding up two rulers (for example), does not leave Mr Rowland with a hand to point or manipulate a third object.

The use of an anaglyph would be a much better solution to the problem. The anaglyph would be rendered on screen, and built into the application, so there would be no reliability or usability issues for the user. There are some issues with an anaglyph, in that the student is still trying to view a 3D object on a 2D screen, and that also the school or teacher would need to order a set of 3D glasses.

Fortunately anaglyph are not a novel technology, and 3D glasses are widely and cheaply available, so the need to use 3D glasses is not a problem.

### 1.12 Anaglyph type justification

In designing the 3D anaglyph I had the choice of different colours to create the anaglyph from. Some of these colours included red-cyan, amber-dark blue (ColorCode 3D), and magenta-cyan.

The different colour combinations each have their pros and cons.

Red-cyan has poor red perception, but good greens.

ColorCode 3D has almost full colour perception; however it requires either a dark room or bright screen. St Paul's School uses Interactive Whiteboards which are not bright. It would also be a large inconvenience for the teacher to have to close the blinds every time they wanted to use the software. On top of this closing the blinds cannot ensure a dark room.

Magenta-cyan has better colours than red-cyan, and does not heavily rely on the brightness of the room like ColorCode 3D. Though Magenta-cyan appears to be the best choice, I chose red-cyan. This decision was based on the fact that red-



cyan is by far the most popular choice to produce 3D anaglyphs. This means that there are many more suitable glasses available on the market, meaning sourcing will be easier, and they will be cheaper. There is also the most documentation on their theory and use.

### 1.13 Complexity of problem

This project is complex because it uses 3D vector mathematics, including 3D vector manipulation, which is a part of the A level Mathematics course.

## 2. Design

---

## 2.1 Overall system design

The system can be divided into 4 sections: inputs, outputs, processes, and file manipulations.

### 2.1.1 Inputs

1. Vector information
  - a. Equation of a line
  - b. Coordinates of a point
2. Function selection [sec. 2.2.6]
  - a. Selecting two points and creating the line between them
  - b. Selecting two lines and creating the angle between them
  - c. Changing the colour of an object (line, point, angle)
  - d. Turning the anaglyph on/off
    - i. Adjusting anaglyph
3. Camera control [sec. 2.9.1]
  - a. Rotate
  - b. Pan
  - c. Zoom
  - d. Reset to original position and orientation

### 2.1.2 Outputs

1. Rendering [sec. 2.2.3]
  - a. Render a line
  - b. Render a point
  - c. Render an angle
  - d. Render anaglyph

### 2.1.3 Processes

1. Vector Calculations

- a. Calculating the two points at either end of a line [sec. 2.8.3.2]

## 2. Anaglyph Calculations

- a. Calculating parallax [sec. 2.10]

### 2.1.4 Files

#### 1. File Management

- a. Open Scene [sec. 2.7.4]
- b. Save Scene [sec. 2.7.3]

#### 2. Scene Management

- a. New Scene [sec. 2.7.1]
- b. Close Scene [sec. 2.7.2]

## 2.2 Description of modular system

Breaking the system down into modules allows each module to be tested independently.

### 2.2.1 Program

The Program module is the overarching container for everything to do with the application.

### 2.2.2 User Interface

The User Interface module contains the forms (windows), buttons, and inspectors. Using the forms the user can input all information necessary for use of the program. Outputs from the program will also be available to the user through the user interface. [sec. 2.3]

### 2.2.3 Scene

The Scene module creates, renders, and performs calculations with lines, points and angles, including anaglyph rendering and calculations. [sec. 2.8.12]

#### 1. Point Class

The Point Class contains information about a point in the Scene. An instance of the Point class can only be created by the Scene. If another module wants to create an instance of the Point class it must pass the information to the Scene.

[sec. 2.8.9]

#### 2. Line Class

The Line Class contains information about a line in the Scene. The same rules apply to the Line class as the Point class, in terms of instantiation.

[sec. 2.8.10]

#### 3. Angle Class

The Angle Class contains information about an angle between two existing lines in the Scene. Deleting one of these lines will not delete the angle. The same instantiation rules apply to the Angle class as to the Line class and the Point class.

[sec. 2.8.11]

#### 4. Anaglyph

The Anaglyph is rendered by the Scene. As such all information regarding the Anaglyph is contained within the Scene class.

[sec. 2.8.12]

### 2.2.4 File Handling

This module contains all the necessary functions for file handling and processing. The File Handling module can create, close, and save a scene. [sec. 2.7]

### 2.2.5 Object Inspector

The Object Inspector is the panel which displays a list of all objects in the Scene. As a result, the Object Inspector needs to be able to receive information about objects in the Scene, as well as pass information to the Scene such as colour changes and object deletion. [sec. 2.8.6]

#### 1. Delete Object

To delete an object the user clicks a button in the Object Inspector. This button is only available when an object is selected within the Object Inspector. Selecting the delete button will issue a prompt to the user, confirming that he wants to delete the object.

#### 2. Change Colour

The Change Colour button is accessed within the Object Inspector and is only available when an object is selected within the Object Inspector. Selecting it opens a list of all the colours available to the user, regardless of whether the colour is in use by another object or not.

[sec. 2.8.7]

#### 3. Refresh

The Object Inspector refreshes its contents after every change it makes (deleting, changing colour).

The Refresh button in the Object Inspector is a safe keep button which causes the inspector to refresh the list of items in the scene. The use of this button may be necessary if

- i. The user inputs a new object using a different form while the Object Inspector is open.
- ii. The program is running slowly enough that a change made within the Object Inspector is not updated in the Scene before the Object Inspector refreshes itself.

### 2.2.6 Input Forms

The Input Form module contains within it all the windows used to input data needed to create new objects within the Scene. Some forms within it require data from the Scene. [sec. 2.3.1]

#### 1. Input Panel

The Input Panel is the main form, containing buttons for all user functions within it. It also acts as an information route between different forms and threads.

[sec. 2.8.1]

#### 2. Point Input

Point Input is a form with fields for the coordinates of a point. It passes this information to the Scene which creates a new instance of the Point class.

[sec. 2.8.2]

#### 3. Line Input

The Line Input module contains two forms within it; both forms handle the input for information of a new line. The first form takes inputs based on the origin vector, magnitude, and direction vector of the line.

The second form takes inputs based on two pre-existing points in the Scene. This form takes information from the Scene, as it has to display a list of all instances of the Point class within it. The second form is accessed through a button in the first form.

The Line Input module passes information up to the main input panel, which is then fetched by the Scene, creating a new instance of the Line class.

[sec. 2.8.3]

#### 4. Angle Input

The Angle Input module takes information from the Scene, as it has to display two lists of all instances of the Line class. The user selects two (different) lines, and submits the form. The module then passes the data to the Scene which creates a new instance of the Angle class.

[sec. 2.8.4]

### 2.2.7 Anaglyph Control

The Anaglyph Control module handles anaglyph related processes. This mainly includes calculating new Points, Lines, and Angles based on camera position, applying new colours (red or cyan), and adjusting parallax based on where the user is standing.

[sec. 2.10]

### 2.2.8 Camera Manipulation

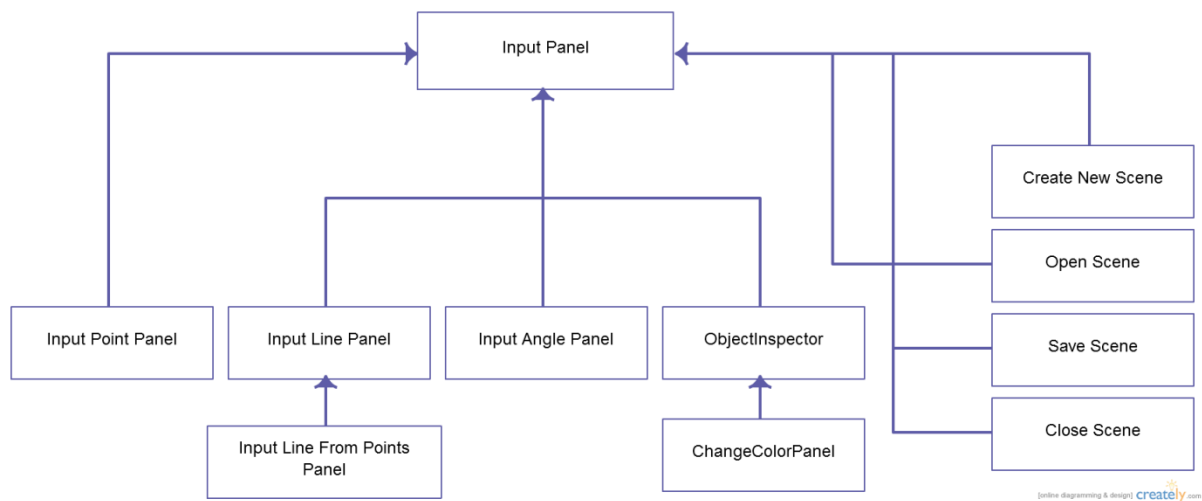
Controlled from within the Scene, the Camera Manipulation module handles button presses corresponding to rotations, translations, zooming, and a specific function to reset the camera to its original position.

[sec. 2.9.1]



## 2.3 GUI Rationale

### 2.3.1 Hierarchy

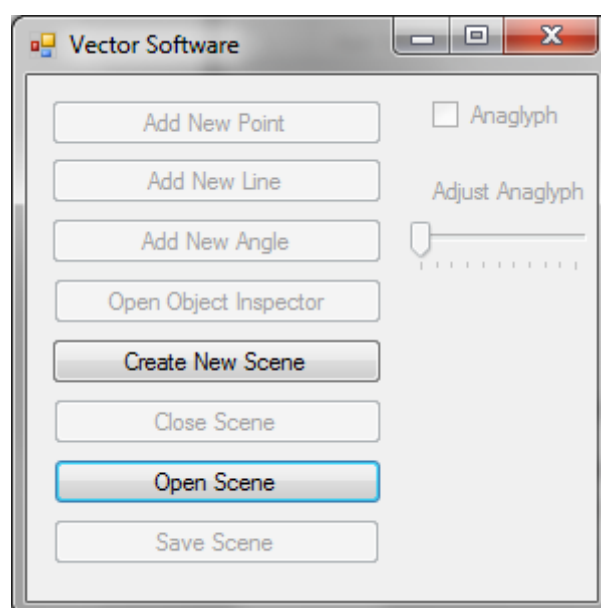


### 2.3.2 Design

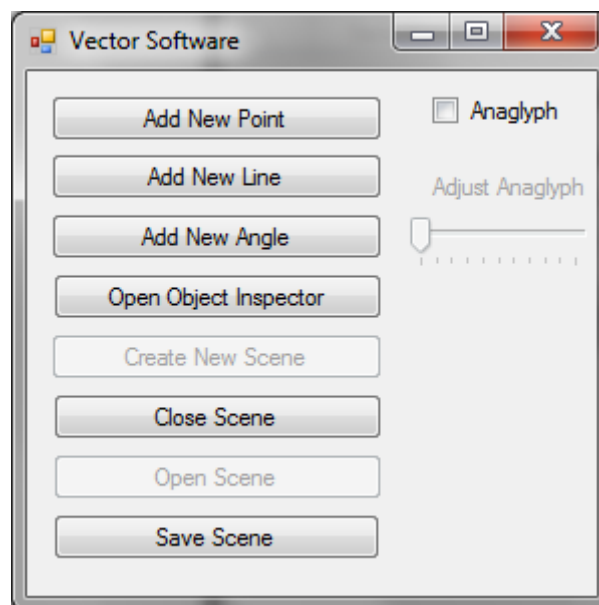
The basic Windows colour scheme and design elements were used for the project, as the user would already be familiar with them.

#### 1. Input Panel

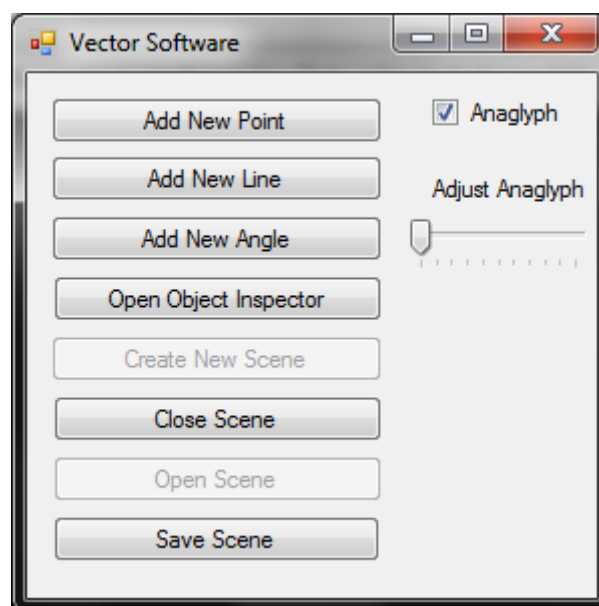
The main input panel has to allow the user access to Scene functions and object related functions. When there is no existing Scene, some functions (which would not work without a Scene) are greyed out and unavailable to the user. They are not hidden entirely, but greyed out to show the user where they would go to access a function.



Once the Scene has been created these buttons become available to the user. A check box has been used for whether the Anaglyph should be toggled, as it is a simple binary value.



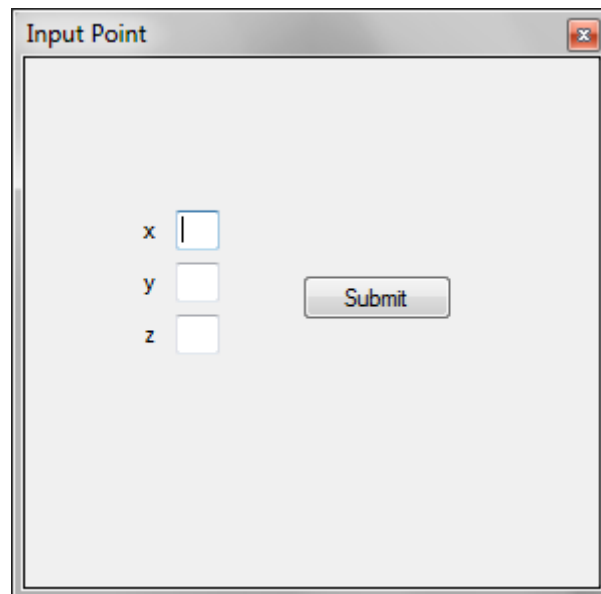
On activating the Anaglyph (checking the box), the Adjust Anaglyph slider becomes available to the user. If the Anaglyph check box is unchecked, the Adjust Anaglyph slider again becomes unavailable.



## 2. Input Point

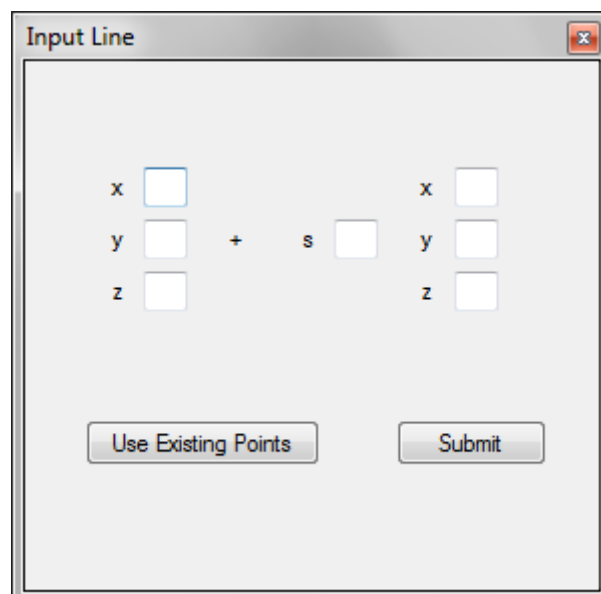
The Input Point panel takes in coordinates of a point, and submits them to the Scene. Each text box is labelled with the corresponding coordinate name, to avoid any confusion.

The x, y and z text boxes are vertically stacked, as this is the most common method of denoting a vector:  $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$

A dialog box titled "Input Point" with a close button in the top right corner. Inside the dialog, there are three vertically stacked text input boxes labeled "x", "y", and "z" on the left. To the right of these boxes is a "Submit" button.

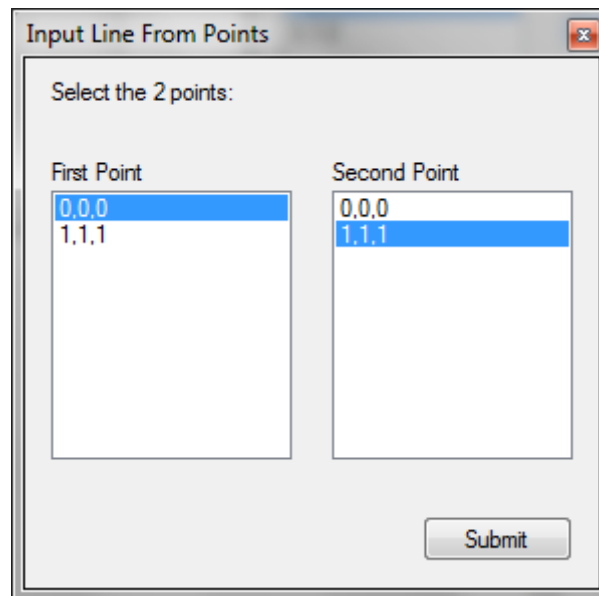
### 3. Input Line

The Input Line Panel takes in the coordinate of a Line in the form  $\begin{pmatrix} x \\ y \\ z \end{pmatrix} + s \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ . Each text box has a corresponding label to signify the data that should be inputted in it. A button is also available, within the Input Line Panel, to create a Line from existing Points.

A dialog box titled "Input Line" with a close button in the top right corner. Inside the dialog, there are two sets of vertically stacked text input boxes labeled "x", "y", and "z". Between the two sets is a "+" sign and a scalar input box labeled "s". Below the input boxes are two buttons: "Use Existing Points" and "Submit".

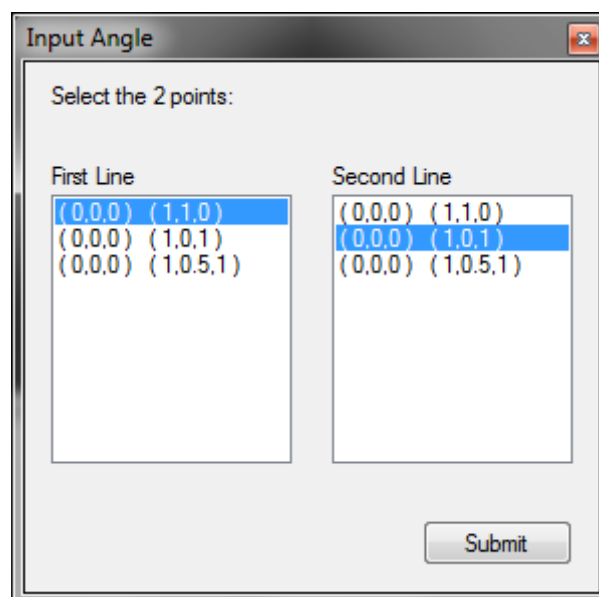
### 4. Input Lines From Points

The Input Lines From Points Panel offers two lists of existing Points, from which the user can select two Points.



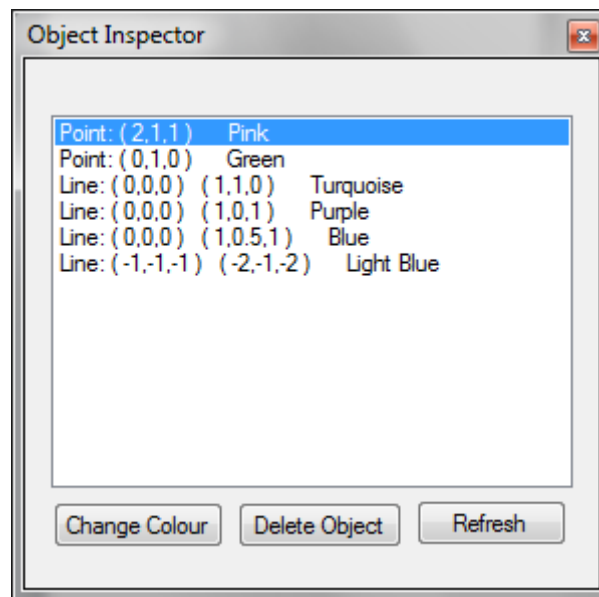
### 5. Input Angle

The Input Angle Panel consists of two lists of existing Lines, from which the user can select two Lines.



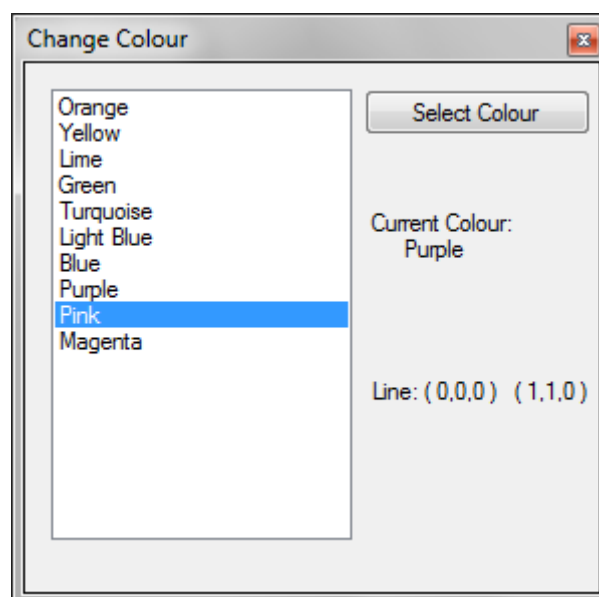
### 6. Object Inspector

The Object Inspector has to show all useful information about every object to the user. This includes the type of object, its coordinates, and its colour. Buttons to change the selected objects colour, delete the selected object, or refresh the list are available to the user.



## 7. Colour Change

The Change Colour Panel contains a list of all available colours, along with the objects current colour, and the objects coordinates.



## 2.4 Rendering Libraries

### 2.4.1 Libraries

All libraries used are included in the final published version of the application.

#### 1. OpenGL

The first library necessary to the project is OpenGL, an open source, cross platform graphics library. Within the project it is responsible for viewing objects (shaders, rendering, matrices), and for providing key data structures. The OpenGL library included is 'OpenGL.dll'.

[ref. 14]

#### 2. Tao Framework

The Tao Framework is a free, open source wrapper for OpenGL in C# and .NET. Although Tao also includes OpenAL, OpenCL, and other libraries, Tao simply acts as a wrapper for FreeGlut. The Tao library included is 'Tao.FreeGlut.dll'.

[ref. 15]

#### 3. FreeGlut

FreeGlut is an open source implementation of GLUT, the OpenGL Utility Toolkit., and will often be referred to as 'Glut'. The FreeGlut library included is 'freeglut.dll'.

Glut controls two main parts of the project: the scene window, and the render loop.

[ref. 11]

### 2.4.2 Data Types

#### 1. OpenGL

OpenGL provides us with a number of data structures necessary for rendering. View ref. 14 for the OpenGL official documentation of the following fields and methods.

#### 1. VBO

VBO stands for Vertex Buffer Object. The VBO structure allows data (position, colour) to be sent to the video card, without the data being immediately rendered. This non-

immediate mode rendering has better performance than immediate mode rendering as the data is stored in the video card's memory, rather than system memory, allowing faster access for the video card.

## 2. Vector3

The Vector3 data type holds 3 floats, commonly x, y, and z. In OpenGL, a VBO of type Vector3 can define a point or line.

Colours are also defined as Vector3 data types, where the three floats (previously x, y and z) represent red, green, and blue of RGB. Each float ranges from 0 to 1 when used to define an RGB colour.

Vector3.Up is often used when defining the camera rotation. It is an inbuilt variable for the Vector3 (0, 1, 0).

Vector3.Zero is another inbuilt variable and is (0, 0, 0). It is often used to set the camera to face the origin.

## 3. ShaderProgram

ShaderProgram is a data type for a program designed to execute one of the stages of the rendering pipeline [[http://www.opengl.org/wiki/Rendering\\_Pipeline\\_Overview](http://www.opengl.org/wiki/Rendering_Pipeline_Overview)]. This program is also responsible for the projection, view (camera), and model matrices.

## 4. VertexShader

The VertexShader takes in vertex attribute data and produces a result which is sent to an output vertex stream. In the application the VertexShader performs a transformation on the data which is then passed to the Vertex Post-Processing stage, which is set up for Primitive Assembly, the final stage in rendering a point or line.

The VertexShader is stored as a string literal. [sec. 2.8.12.1]

## 5. FragmentShader

Once the primitive object has been rasterized (converted from vector data to pixel data), the FragmentShader processes a Fragment (a collection of data produced by rasterisation) into a set of colours and a depth value.

The VertexShader is stored as a string literal. [sec . 2.8.12.1]

## 6. Matrix4

A Matrix4 data type is a 4x4 matrix containing floats. Methods contained within the Matrix4 class are used in camera and object manipulations.

### 2.4.3 Inbuilt Methods

#### 1. OpenGL

##### 1. Viewport

The viewport function sets the field of view of the camera, and has 4 parameters: x, y, width, height.

Parameters x and y represent the bottom left corner of the viewport. Width and height represent the size of the viewport. As the viewport will always be the whole scene window, x and y will be set to 0, and width and height will be set to the scene's width and height.

##### 2. Clear

The clear function removes the previous render from the screen by emptying the buffers. The two specific buffers that are emptied are the colour buffer and the depth buffer.

##### 3. ShaderProgram.SetValue

ShaderProgram.SetValue sets a new value to a matrix within the ShaderProgram. As such, every time it is called, the specific matrix you are dealing with (view, projection, model) has to be specified.

##### 4. Matrix4.LookAt

Matrix4.LookAt is used to set the camera's location, the point it's looking at (target), and its rotation.

Camera translations and resets are done using the matrix produced by Matrix4.LookAt.

##### 5. Matrix4.CreateRotation

Matrix4.CreateRotation takes in a float value representing an angle, and creates a 4x4 matrix from it representing that angle.



Three individual CreateRotation methods exist:  
CreateRotationX, CreateRotationY, CreateRotationZ.

These are used in camera manipulations by multiplying the matrices together, along with the camera's LookAt matrix.

#### 6. Matrix4.CreateTranslation

Matrix4.CreateTranslation is used when setting a model matrix's value, ensuring it is not displaced from where it is supposed to be (i.e. a translation of 0 is created).

This is used every time an object is rendered.

#### 7. BindBufferToShaderAttribute

BindBufferToShaderAttribute assigns an attribute to a buffer object, all within the ShaderProgram. It is called twice when drawing objects, first to set the position of the object, and secondly to set the colour of the object.

#### 8. BindBuffer

BindBuffer assigns a VBO to a buffer, preparing it to be rendered. The VBO stores within it information about the buffer it is to be bound to. In the draw procedures, this buffer is the ElementArrayBuffer.

#### 9. DrawElements

DrawElements is the method which is called once all the preparations for rendering have been carried out. It takes 4 parameters: BeginMode, Count, DrawElementsType, and Indices.

BeginMode is specific to which kind of object (point or line) is being drawn. Each BeginMode is a pre-defined variable in OpenGL.

Count uses the VBO defining that object to check how many vertices the object contains.

DrawElementsType specifies the type of value in the Indices.

Indices points to the location the indices are stored.

## 2. Glut

### 1. glutInit

glutInit is used to initialise the Glut library and requests the system allows a session for the Glut program. It is called at the beginning of initialising the scene.

### 2. glutInitDisplayMode

glutInitDisplayMode sets the initial display mode, and is used when creating top-level windows (specifically the scene window).

The method is passed two values, GLUT\_DOUBLE and GLUT\_DEPTH.

GLUT\_DOUBLE tells the program that it is using a double buffered display mode. This means that while one frame is being rendered, the next frame is being created in a separate buffer. Once the current frame is done displaying, the program swaps buffers and immediately displays the contents of the other buffer. This stops flickering in the display.

GLUT\_DEPTH is a bit mask to request a window with a depth buffer. This effectively declares the program as being 3D.

### 3. glutInitWindowSize

glutInitWindowSize defines the window size created by Glut, taking width and height as parameters.

### 4. glutCreateWindow

glutCreateWindow creates the window once it has been sufficiently defined by previous Glut initialisation methods. The method takes one parameter, a string representing the title of the window.

### 5. glutIdleFunc

glutIdleFunc sets the global idle callback to a specific function (taken as a parameter), allowing the Glut program to perform continuous animation, even when window system events aren't being received.

The callback function within the scene is set the the main render loop, OnRenderFrame.

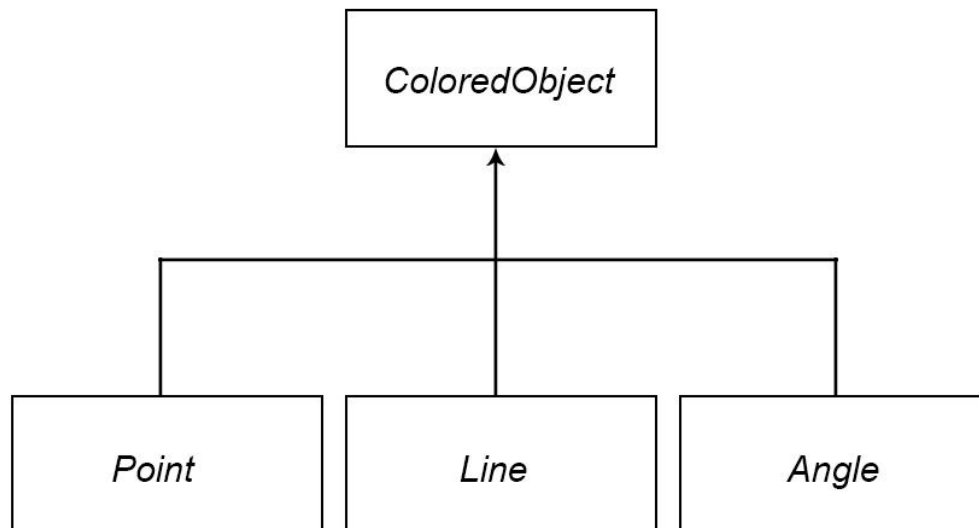
## 6. glutMainLoop

glutMainLoop causes the Glut event processing loop to be entered. glutMainLoop calls any callback functions that have been registered prior to its running (OnRenderFrame). Once this method is called it will never return.

## 7. glutSwapBuffers

glutSwapBuffers performs a buffer swap, which is necessary because the project uses double buffering. The update takes place during the vertical retrace of the monitor, rather than when the method is called, to avoid tear.

## 2.5 Class Structure



## 2.6 Data Validation

All validation takes place in the input panels.

### 2.6.1. InputPanel

#### Opening a file

When opening a file, there is no guarantee that the user hasn't manually changed any values in the csv file. As such some validation must take place here.

Firstly, if an invalid file is chosen (the file cannot be read from), a message box shows with the message "Invalid File".

If the entry in the csv does not start with 'point', 'line', or 'angle', no data from the entry will be attempted to be read in.

Within a line, if any piece of data is not a float (apart from the title at the beginning of the line), an error is thrown and the file stops trying to read that line and a message box will show with the message "Data could not be read from the file".

Another piece of data validation required is that the last Point on a Line is not equal to the first Point. This will produce a message box reading "Invalid line found and could not be imported".

The final piece of data validation required is that the two Lines defining an Angle are not the same, and that both the Lines are valid. An occurrence of this will produce a message box which reads "Invalid angle found and could not be imported".

### 2.6.2. InputPointPanel

When clicking the submit button, the program attempts to parse the contents of each text box to a float. If any of the attempts fail, the InputPointPanel will not submit the data to the InputPanel, nor will it close the form. An error message will display in the form, which reads "Invalid data inputted".

### 2.6.3 InputLinePanel

When clicking the submit button, the program attempts to parse the contents of each text box to a float. If any of the attempts fail the InputLinePanel will not submit data to the InputPanel, nor will it close the form. An error message will display, which reads "Invalid data inputted".

If the entered magnitude is 0, an error message will show with the message "Magnitude cannot be 0". This will result in the data not being submitted, and the form not being closed.

The direction vector of the line cannot be (0, 0, 0). If this data is inputted, the data will not be submitted, and the form will show an error message reading "Direction vector cannot be 0, 0, 0".

#### 2.6.4 InputLineFromPointsPanel

On clicking the submit button, if the two Points are the same, the form shows an error message which reads "The two points cannot be the same".

If one of the Points has been deleted (in the Object Inspector) after the Input Line From Points panel has been opened, and this Point is chosen to create a new line with, an error message will be shown. The error message will read "At least one of the points does not exist".

#### 2.6.5 InputAnglePanel

On clicking the submit button, if the two Lines are the same, the form shows an error message which reads "The two lines cannot be the same".

If one of the Lines has been deleted (in the Object Inspector) after the Input Angle panel has been opened, and this Line is chosen to create a new angle with, an error message will be shown. The error message will read "At least one of the lines does not exist".

### 2.7 File Processing

#### 2.7.1 Creating a new Scene

Creating a new Scene creates a new instance of a Scene class. If a Scene is already open this feature will not be available to the user.

Creating a Scene will make the function buttons, and the object inspector panel available.

#### 2.7.2 Closing a Scene

This feature will not be available to the user if a Scene does not currently exist.

Closing the Scene will disable function buttons and the object inspector panel.

#### 2.7.3 Saving a scene

This option will only be available to the user if a Scene is currently open.

Saving a Scene will create a comma-separated file (csv) containing information about the Scene, including all objects within it. The file will be saved onto the school network.

The file starts with a list of points, then a list of lines, then a list of angles. Each entry starts with the name of the object ('point', 'line', 'angle'), followed by essential coordinates defining the object, and finally by the colour of the object.

The size of the file directly relies on the number of objects it stores.

Point: 1 string, 6 floats

Line: 1 string, 9 floats

Angle: 1 string, 15 floats

However it should be noted that the .csv will store each of these as strings. The following sizes were calculated by creating a csv with 1 of the relevant objects, then creating a csv with 2 of the relevant objects, and calculating the difference.

Point: 19 bytes

Line: 28 bytes

Angle: 37 bytes

As the maximum number of objects is 20, and the largest object is the Angle, the maximum size of this file would be 740 bytes.

For pseudo code for saving a scene, see sec. 2.8.1.2, *saveScene*.

#### 2.7.4 Opening a scene

If a Scene is already open, this option won't be available to the user.

Opening a Scene creates a new instance of the Scene class, and reads all information from the comma-separated file (csv) to it.

For validation of data on inputting, see sec. 2.6.1.

## 2.8 Class fields and methods

All size calculations are based off the size of all variables within the data type. Only maximum sizes have been included, where applicable. For variables without maximum size, a value for max size when user is using 20 objects (the maximum number of objects the user will viably use).

### 2.8.1 InputPanel

#### 1. Variables

##### scene

Access Type: Private

Field Type: Scene

Initial Value: Null

Size: 8 bytes

The Scene class is created within the main input panel so that data can pass via input forms to the instance of the Scene class, and visa-versa.

##### sceneThread

Access Type: Private

Field Type: Thread

Initial Value: Null

Size: 1 megabyte

The thread the Scene class is run in.

##### pointsToBeRendered

Access Type: Private

Field Type: List of Vector3

Initial Value: Empty

Size: 240 bytes

A list checked by the Scene for new points to add to the Scene. The points have no information about their colour.

It is unlikely more than 1 Point will ever be stored in this list.

##### linesToBeRendered

Access Type: Private

Field Type: List of Vector3 Array

Initial Value: Empty

Size: 480 bytes

A list checked by the Scene for new lines to add to the Scene. The lines have no information about their colour.

It is unlikely more than 1 Line will ever be stored in this list.

#### anglesToBeRendered

Access Type: Private

Field Type: List of Vector3 Array

Initial Value: Empty

Size: 960 bytes

A list checked by the Scene for new angles to add to the Scene. The angles have no information about their colour.

It is unlikely more than 1 Angle will ever be stored in this list.

#### pointsToBeRenderedWithColor

Access Type: Private

Field Type: List of Vector3 Array

Initial Value: Empty

Size: 480 bytes

A list checked by the Scene for new points, which contain information about their colour.

It is unlikely more than 1 Point will ever be stored in this list.

#### linesToBeRenderedWithColor

Access Type: Private

Field Type: List of Vector3 Array

Initial Value: Empty

Size: 720 bytes



A list checked by the Scene for new lines, which contain information about their colour.

It is unlikely more than 1 Line will ever be stored in this list.

#### anglesToBeRenderedWithColor

Access Type: Private

Field Type: List of Vector3 Array

Initial Value: Empty

Size: 1200 bytes

A list checked by the Scene for new angles, which contain information about their colour.

It is unlikely more than 1 Angle will ever be stored in this list.

#### pointToBeDeleted

Access Type: Public

Field Type: Point

Initial Value: Null

Size: 8 bytes

A variable checked by the Scene for a Point to be deleted.

#### lineToBeDeleted

Access Type: Public

Field Type: Line

Initial Value: Null

Size: 8 bytes

A variable checked by the Scene for a Line to be deleted.

#### angleToBeDeleted

Access Type: Public

Field Type: Angle

Initial Value: Null

Size: 8 bytes

A variable checked by the Scene for an Angle to be deleted.

objectWithColorChange

Access Type: Private

Field Type: Object

Initial Value: Null

Size: 8 bytes

A variable checked by the scene for a Point, Line or Angle which is to have its colour changed by the Scene.

newColor

Access Type: Private

Field Type: Vector3

Initial Value: Null

Size: 12 bytes

A variable containing the new colour for *objectWithColorChange*.

pointsInScene

Access Type: Public

Field Type: List of Point

Initial Value: Empty

Size: 160 bytes

A list of Points to which the Scene regularly writes its local list of Points.

linesInScene

Access Type: Public

Field Type: List of Line

Initial Value: Empty

Size: 160 bytes

A list of Lines to which the Scene regularly writes its local list of Lines.

anglesInScene

Access Type: Public

Field Type: List of Angle

Initial Value: Empty

Size: 160 bytes

A list of Angles to which the Scene regularly writes its local list of Angles.

anaglyphToggle

Access Type: Public

Field Type: Bool

Initial Value: False

Size: 4 bytes

A Boolean representing whether the user has selected the Anaglyph to be rendered. This value is regularly checked by the Scene.

anaglyphAdjust

Access Type: Private

Field Type: Int

Initial Value: Null

Size: 32 bytes

The value of the Adjust Anaglyph slider in the Input Panel. This value is regularly checked by the Scene if anaglyphToggle is true.

pointLock

Access Type: Private

Field Type: Object

Initial Value: Instance of Object

Size: 12 bytes

Used to control the Points being added to the Scene at the same time as the Scene fetches the Points to be added. Applies to the *pointsToBeRendered* variable.

lineLock

Access Type: Private

Field Type: Object

Initial Value: Instance of Object

Size: 12 bytes

Used to control the Lines being added to the Scene at the same time as the Scene fetches the Lines to be added. Applies to the *linesToBeRendered* variable.

angleLock

Access Type: Private

Field Type: Object

Initial Value: Instance of Object

Size: 12 bytes

Used to control the Angles being added to the Scene at the same time as the Scene fetches the Angles to be added. Applies to the *anglesToBeRendered* variable.

colorLock

Access Type: Private

Field Type: Object

Initial Value: Instance of Object

Size: 12 bytes

Used to control the setting and getting of *objectWithColorChange* and *newColor*.

colorPointLock

Access Type: Private

Field Type: Object

Initial Value: Instance of Object

Size: 12 bytes

Used to control the Points being added to the Scene at the same time as the Scene fetches the Points to be added. Applies to the *pointsToBeRenderedWithColor* variable.

#### colorLineLock

Access Type: Private

Field Type: Object

Initial Value: Instance of Object

Size: 12 bytes

Used to control the Lines being added to the Scene at the same time as the Scene fetches the Lines to be added. Applies to the *linesToBeRenderedWithColor* variable.

#### colorAngleLock

Access Type: Private

Field Type: Object

Initial Value: Instance of Object

Size: 12 bytes

Used to control the Angles being added to the Scene at the same time as the Scene fetches the Angles to be added. Applies to the *anglesToBeRenderedWithColor* variable.

#### anaglyphLock

Access Type: Private

Field Type: Object

Initial Value: Instance of Object

Size: 12 bytes

Used to control the setting and getting of the *anaglyphToggle* variable by the Scene.

## 2. Methods

### InputPanel (constructor)

Access Type: Public

Parameters: None

Return Type: N/A

Initialises the form.

### createNewScene

Access Type: Public

Parameters: None

Return Type: Void

Creates a new instances of sceneThread, and enables buttons to control the scene.

### closeScene

Access Type: Public

Parameters: None

Return Type: Void

Destroys the current instance of the Scene, stops the Scene's thread (*sceneThread*), and disables buttons controlling the scene.

### enableComponents

Access Type: Private

Parameters: None

Return Type: Void

Enables buttons controlling the Scene, input forms, and the Object Inspector button.

### disableComponents

Access Type: Private

Parameters: None

Return Type: Void

Disables buttons controlling the Scene, input forms, and the Object Inspector button.

### initScene

Access Type: Private

Parameters: Object *obj*

Return Type: Void

Creates a Scene, passing it a reference to the Input Panel (obtained as parameter, *obj*).

### addPoint

Access Type: Public

Parameters: Float x, Float y, Float z

Return Type: Void

Used by input forms to add a Point to *pointsToBeRendered*, which will be read by the Scene.

```
Public void addPoint(float x, float y, float z)
    Lock (pointLock)
        pointsToBeRendered.Add(new Vector3(x, y, z))
    End
End
```

### addLine

Access Type: Public

Parameters: Float x, Float y, Float z, Float a, Float b, Float c

Return Type: Void

Used by input forms to add a Line to *linesToBeRendered*, which will be read by the Scene.

```
Public void addLine(float x, float y, float z, float a,
float b, float c)
    Lock (lineLock)
        //First calculate the two points
        Vector3 firstPoint = new Vector3(x, y, z)
        Vector3 secondPoint = new Vector3(a, b, c)
        //Pass the two points to be rendered as a
        line
        linesToBeRendered.Add(new Vector3[] {
            firstPoint, secondPoint } )
    End
End
```

End

End

### addAngle

Access Type: Public

Parameters: Float *l1x*, Float *l1y*, Float *l1z*, Float *l1a*, Float *l1b*, Float *l1c*, Float *l2x*, Float *l2y*, Float *l2z*, Float *l2a*, Float *l2b*, Float *l2c*

Return Type: Void

Used by input forms to add an Angle to *anglesToBeRendered*, which will be read by the Scene.

```
Public void addAngle(float l1x, float l1y, float l1z,
float l1a, float l1b, float l1c, float l2x, float l2y,
float l2z, float l2a, float l2b, float l2c)
```

Lock (angleLock)

```
//Calculate the points defining each line
```

```
Vector3 line1firstPoint = new Vector3(l1x,
l1y, l1z)
```

```
Vector3 line1secondPoint = new Vector3(l1a,
l1b, l1c)
```

```
Vector3 line2firstPoint = new Vector3(l2x,
l2y, l2x)
```

```
Vector3 line2secondPoint = new Vector3(l2a,
l2b, l2c)
```

```
//Pass the points which define the line
between which the angle lies
```

```
anglesToBeRendered.Add( new Vector3[] {
line1firstPoint, line2secondPoint,
line2firstPoint, line2secondPoint } )
```

End

End

### getPoints

Access Type: Public

Parameters: None



Return Type: List of Vector3

Returns the list of Points added by input forms, *pointsToBeRendered*, and then deletes all entries in the list.

```
Public List<Vector3> getPoints()  
  
    Lock (pointLock)  
  
        //Use a temporary value to allow the  
        original variable to be reset before  
        returning the value  
  
        List<Vector3> temp = pointsToBeRendered  
        pointsToBeRendered = new List<Vector3>()  
        return temp  
  
    End  
  
End
```

### getLines

Access Type: Public

Parameters: None

Return Type: List of Vector3 Array

Returns the list of Lines added by input forms, *linesToBeRendered*, and then deletes all entries in the list.

```
Public List<Vector3[]> getLines()  
  
    Lock (lineLock)  
  
        //Use a temporary value to allow the  
        original variable to be reset before  
        returning the value  
  
        List<Vector3[]> temp = linesToBeRendered  
        linesToBeRendered = new List<Vector3[]>()  
        return temp  
  
    End  
  
End
```

getAngles

Access Type: Public

Parameters: None

Return Type: List of Vector3 Array

Returns the list of Angles added by input forms, *anglesToBeRendered*, and then deletes all entries in the list.

```
Public List<Vector3[]> getAngles()
```

```
    Lock (angleLock)
```

```
        //Use a temporary value to allow the  
        original variable to be reset before  
        returning the value
```

```
        List<Vector3[]> temp = anglesToBeRendered
```

```
        anglesToBeRendered = new List<Vector3[]>()
```

```
        return temp
```

```
    End
```

```
End
```

getPointsWithColor

Access Type: Public

Parameters: None

Return Type: List of Vector3 Array

Returns the list of Angles with colour added by input forms, *anglesToBeRenderedWithColor*, and then deletes all entries in the list.

```
Public List<Vector3> getPointsWithColor()
```

```
    Lock (colorPointLock)
```

```
        //Use a temporary value to allow the  
        original variable to be reset before  
        returning the value
```

```
        List<Vector3> temp =  
        pointsToBeRenderedWithColor
```

```
        pointsToBeRenderedWithColor = new  
        List<Vector3>()  
  
        return temp
```

End

End

### getLinesWithColor

Access Type: Public

Parameters: None

Return Type: List of Vector3 Array

Returns the list of Lines with colour added by input forms, *linesToBeRenderedWithColor*, and then deletes all entries in the list.

```
Public List<Vector3[]> getLinesWithColor()
```

```
    Lock (colorLineLock)
```

```
        //Use a temporary value to allow the  
        original variable to be reset before  
        returning the value
```

```
        List<Vector3[]> temp =  
        linesToBeRenderedWithColor
```

```
        linesToBeRenderedWithColor = new  
        List<Vector3[]>()
```

```
        return temp
```

End

End

### getAnglesWithColor

Access Type: Public

Parameters: None

Return Type: List of Vector3 Array

Returns the list of Angles with colour added by input forms, *anglesToBeRenderedWithColor*, and then deletes all entries in the list.

```

Public List<Vector3[]> getAnglesWithColor()
    Lock (colorAngleLock)

        //Use a temporary value to allow the
        original variable to be reset before
        returning the value

        List<Vector3[]> temp =
        anglesToBeRenderedWithColor

        anglesToBeRenderedWithColor = new
        List<Vector3[]>()

        return temp

    End
End

```

### setPointsInScene

Access Type: Public

Parameters: List of Point  $p$

Return Type: Void

Sets *pointsInScene* to  $p$ . Used for Scene to pass existing data to Input Panel.

### setLinesInScene

Access Type: Public

Parameters: List of Line  $l$

Return Type: Void

Sets *linesInScene* to  $l$ . Used for Scene to pass existing data to Input Panel.

### setAnglesInScene

Access Type: Public

Parameters: List of Angle  $a$

Return Type: Void

Sets *anglesInScene* to  $a$ . Used for Scene to pass existing data to Input Panel.

setNewColor

Access Type: Public

Parameters: Object *o*, Vector3 *color*

Return Type: Void

Sets *objectWithColorChange* to *o*, and *newColor* to *color*. It is used by the Object Inspector when changing colours, and is controlled by *colorLock*.

```
Public void setNewColor(object o, Vector3 color)
```

```
    Lock (colorLock)
```

```
        objectWithColorChange = o
```

```
        newColor = color
```

```
    End
```

```
End
```

getColorChangeObject

Access Type: Public

Parameters: None

Return Type: Object

Returns *objectWithColorChange*, then sets *objectWithColorChange* to an empty value. It is called regularly by the Scene, and is controlled by *colorLock*.

```
Public object getColorChangeObject()
```

```
    Lock (colorLock)
```

```
        //Use a temporary value to allow the  
        original variable to be reset before  
        returning the value
```

```
        Object temp = objectWithColorChange
```

```
        objectWithColorChange = null
```

```
        Return temp
```

```
    End
```

```
End
```

getNewColor

Access Type: Public

Parameters: None

Return Type: Vector3

Returns *newColor*. It is called by the Scene just before the Scene calls *getColorChangeObject*, and is controlled by *colorLock*.

```
Public Vector3 getNewColor
```

```
    Lock (colorLock)
```

```
        Return newColor
```

```
    End
```

```
End
```

setAnaglyphToggled

Access Type: Private

Parameters: Bool *value*

Return Type: Void

Sets *anaglyphToggled* to *value*, and is called when the Anaglyph toggle button is clicked, or in *disableComponents*. It is controlled by *anaglyphLock*.

```
Private void setAnaglyphToggled(bool value)
```

```
    Lock (anaglyphLock)
```

```
        anaglyphToggle = value
```

```
    End
```

```
End
```

getAnaglyphToggled

Access Type: Public

Parameters: None

Return Type: Bool

Returns the value of *anaglyphToggle*, and is controlled by *anaglyphLock*.

```
Public bool getAnaglyphToggled()  
    Lock (anaglyphLock)  
        Return anaglyphToggle  
    End  
End
```

End

### saveScene

Access Type: Private

Parameters: None

Return Type: Void

Opens the .NET save file dialogue to select a file location and name, and then writes data in the scene to the file.

```
Private void saveScene()  
    saveFileDialog.ShowDialog  
    String filePath = saveFileDialog.FileName  
    //Make sure the file exists  
    If (File at filePath does not exist)  
        Create File at filePath  
    End  
  
    Write Empty string to File at filePath  
  
    //Define the separator for all the strings  
    String delimiter = ","  
  
    //Produce a list of all objects to be outputted to  
    the file  
    List<string[]> outputList = new List<string[]>()  
    //Add points to the list  
    Foreach Point p in pointsInScene
```

```
        String[] line = suitable data about p
        outputList.Add(line)
    End
    outputList.Add(empty string array)
    //Add lines to the list
    Foreach Line l in linesInScene
        String[] line = suitable data about l
        outputList.Add(line)
    End
    outputList.Add(empty string array)
    //Add angles to the list
    Foreach Angle a in anglesInScene
        String[] line = suitable data about a
        outputList.Add(line)
    End

    //Produce an array of arrays from the list
    (required for a comma separated file)
    String[][] output = outputList.ToArray()

    //A StringBuilder class is created, which places
    the ',' between each data entry in a row, and
    stores the results within the StringBuilder. The
    whole StringBuilder is then written to the file.
    Int length = output.GetLengthOfFirstDimension
    StringBuilder sb = new StringBuilder()

    //Add each output string to the string builder,
    separating them by the previously defined
    separator (delimiter)
    For (int index = 0; index < length; index++)
```



```
sb.AppendLine(output[index] joined by  
delimiter)
```

End

```
//Send the contents of the string builder to the  
file
```

```
File.AppendAllText(filePath, sb.ToString())
```

End

### openScene

Access Type: Public

Parameters: None

Return Type: Void

Opens the .NET open file dialogue.

### openFileDialog\_FileOk

Access Type: Private

Parameters: Object *sender*, EventArgs *e*

Return Type: Void

Attempts to read the file, create a new Scene, and then read data from the file into the Scene. This function is only called if *openFileDialog* determines a valid file has been selected.

```
Private void openFileDialog_FileOk(object sender,  
EventArgs e)
```

```
String filename = openFileDialog.FileName
```

```
String[] lines = new string[] { }
```

```
//If the file is valid, read in the data and  
create a new scene. Otherwise display an error  
message and don't create a new scene
```

Try

```
Lines = Read All Lines from File at filename
```

```
createNewScene()
```

Catch

```
        MessageBox.Show("Invalid File")

End

//Split the data down into individual values,
//storing them in arrays

Foreach string s in lines

    String[] data = s.Split(',')

    //If the data is of a point, deal with it
    //accordingly

    If(data[0] == "point")

        Try

            Convert each piece of data to a
            float

            Convert each float to the
            suitable Vector3

            Vector3[] passedData = new
            Vector3[] { point data, color }

            pointsToBeRenderedWithColor.Add(
            passedData)

        Catch

            MessageBox.Show("Data could not
            be read from the file")

        End

    //If the data is of a line, deal with it
    //accordingly

    Else If(data[0] == "line")

        Try

            Convert each piece of data to a
            float

            Convert each float to the
            suitable Vector3
```

```
Vector3[] passedData = new
Vector3[] { firstPoint,
secondPoint, color }

If (firstPoint == secondPoint)

    MessageBox.Show("Invalid
    line found and could not be
    imported")

Else

    linesToBeRenderedWithColor.
    Add(passedData)

End

Catch

    MessageBox.Show("Data could not
    be read from the file")

End

//If the data is of a angle, deal with it
accordingly

Else If(data[0] == "angle")

    Try

        Convert each piece of data to a
        float

        Convert each float to the
        suitable Vector3

        Vector3[] passedData = new
        Vector3[] { f1, s1, f2, s2, c }

        If both lines are not the same
        AND both lines are valid

            anglesToBeRenderedWithColor
            .Add(passedData)

        Else

            MessageBox.Show("Invalid
            angle found and could not
            be imported")

        End Try

    Catch
```

```

        MessageBox.Show("Data could not
        be read from the file")

```

```

    End

```

```

End

```

```

End

```

```

End

```

### InputPanel\_Closing

Access Type: Private

Parameters: Object *sender*, EventArgs *e*

Return Type: Void

This function is called when Input Panel is closing, and handles the case where a Scene may still be open – which would otherwise result in the Input Panel being closed and the Scene being left open. The function closes the *sceneThread*, and then exits the Application.

```

Private void InputPanel_Closing(object sender, EventArgs
e)

```

```

    //If the sceneThread is running, close it

```

```

    Try

```

```

        Abort sceneThread

```

```

    End

```

```

    Application.Exit()

```

```

End

```

## 2.8.2 InputPointPanel

### 1. Variables

#### inputPanel

Access Type: Private

Field Type: InputPanel

Initial Value: Null

Size: 64 bits

A reference to Input Panel so that the data inputted from Input Point Panel can be passed up to Input Panel, and then be fetched by the Scene.

## 2. Methods

### InputPointPanel (constructor)

Access Type: Public

Parameters: InputPanel *ip*

Return Type: N/A

Initialises the form.

### buttonSubmit\_Click

Access Type: Private

Parameters: Object *sender*, EventArgs *e*

Return Type: Void

Checks data in text boxes, and, if valid, passes them to Input Panel using *inputPanel.addPoint*. If successfully submitted the form closes.

## 2.8.3 InputLinePanel

### 1. Variables

#### inputPanel

Access Type: Private

Field Type: InputPanel

Initial Value: Null

Size: 64 bits

A reference to Input Panel so that the data inputted from Input Line Panel can be passed up to Input Panel, and then be fetched by the Scene.

### 2. Methods

#### InputLinePanel (constructor)

Access Type: Public

Parameters: InputPanel *ip*

Return Type: N/A

Initialises the form and checks whether the button to create a Line from pre-existing points should be available (whether any points exist).

#### calculateFinalPoint

Access Type: Private

Parameters: Float x, Float y, Float z, Float s, Float a, Float b, Float c

Return Type: Array of Float

Takes the equation of the Line and calculates the final Point on it, returning it in an array of floats (x, y, z).

```
Private float[] calculateFinalPoint(float x, float y,  
float z, float s, float a, float b, float c)
```

```
//Calculate the point at the other end of the line  
by adding a factor (s) of the direction vector  
(a,b,c) onto the origin vector (x,y,z)
```

```
Float fx = x + (s * a)
```

```
Float fy = y + (s * b)
```

```
Float fz = z + (s * c)
```

```
Return new float[] { fx, fy, fz }
```

End

#### buttonSubmit\_Click

Access Type: Private

Parameters: Object *sender*, EventArgs *e*

Return Type: Void

Checks data in text boxes, and, if valid, passes them to Input Panel using *inputPanel.addLine*. If successfully submitted the form closes.

### 2.8.4 InputLineFromPointsPanel

#### 1. Variables

##### inputPanel

Access Type: Private

Field Type: InputPanel

Initial Value: Null

Size: 64 bits

A reference to Input Panel so that the data inputted from Input Line From Points Panel can be passed up to Input Panel, and then be fetched by the Scene.

### pointsInScene

Access Type: Private

Field Type: Binding List of Point

Initial Value: Null

Size: 160 bytes

The list of Points in the scene, used to display information about the Points in the two list boxes.

## 2. Methods

### InputLineFromPointsPanel (constructor)

Access Type: Public

Parameters: InputPanel *ip*

Return Type: N/A

Initialises the form, calls *getPoints*, and fills the list boxes with information about the Points.

### getPoints

Access Type: Private

Parameters: None

Return Type: Binding List of Point

Fetches *inputPanel.pointsInScene* and creates a Binding List from them, returning them.

```
Private BindingList<Point> getPoints()
```

```
    //Use a temporary value to allow the original  
    variable to be reset before returning the value
```

```
List<Point> points = inputPanel.pointsInScene  
  
BindingList<Point> bindingPoints = new  
BindingList<Point>(points)  
  
Return bindingPoints
```

End

### convertPointsToStrings

Access Type: Private

Parameters: Binding List of Point *points*

Return Type: Binding List of String

Converts the Binding List of Points to a Binding List which is suitable to be displayed in the list boxes.

```
Private BindingList<string>  
convertPointsToStrings(BindingList<Point> points)  
  
    BindingList<string> stringList = new  
    BindingList<string>()  
  
    Foreach Point p in points  
  
        //Produce data ready to be displayed to the  
        user  
  
        String coordinate = p.firstPoint  
        stringList.Add(coordinate)  
  
    End  
  
    Return stringList
```

End

### submitButton\_Click

Access Type: Private

Parameters: Object *sender*, EventArgs *e*

Return Type: Void

If the two selected Points aren't the same, the function submits the Line to the Input Panel.



### 2.8.5 InputAnglePanel

#### 1. Variables

##### inputPanel

Access Type: Private

Field Type: InputPanel

Initial Value: Null

Size: 64 bits

A reference to Input Panel so that the data inputted from Input Angle Panel can be passed up to Input Panel, and then be fetched by the Scene.

##### linesInScene

Access Type: Private

Field Type: Binding List of Line

Initial Value: Null

Size: 160 bytes

The list of Lines in the scene, used to display information about the Lines in the two list boxes.

#### 2. Methods

##### InputAnglePanel

Access Type: Public

Parameters: InputPanel *ip*

Return Type: N/A

Initialises the form, calls *getLines*, and fills the list boxes with information about the Lines.

##### getLines

Access Type: Private

Parameters: None

Return Type: Binding List of Line

Fetches *inputPanel.linesInScene* and converts them from a List to a Binding List.

```
Private BindingList<Line> getLines()

    //Use a temporary value to allow the original
    //variable to be reset before returning the value

    List<Line> lines = inputPanel.linesInScene

    BindingList<Line> bindingLines = new
    BindingList<Line>(lines)

    Return bindingLines

End
```

### convertLinesToStrings

Access Type: Private

Parameters: Binding List of Line *lines*

Return Type: Binding List of String

Converts the Binding List of Lines to a Binding List which is suitable to be displayed in the list boxes.

```
Private BindingList<string>
convertLinesToStrings(BindingList<Line> lines)

    BindingList<string> stringList = new
    BindingList<string>()

    Foreach Line l in lines

        //Produce data ready to be displayed to the
        //user

        String coordinate = "( " + l.firstPoint + "
        )      ( " + l.secondPoint + " )"

        stringList.Add(coordinate)

    End

    Return stringList

End
```

### submitButton\_Click

Access Type: Private

Parameters: Object *sender*, EventArgs *e*

Return Type: Void

If the two selected Lines aren't the same, the function submits them to the Input Panel, then closes the form.

### 2.8.6 ObjectInspectorPanel

#### 1. Variables

##### inputPanel

Access Type: Private

Field Type: InputPanel

Initial Value: Null

Size: 64 bits

A reference to Input Panel so that the data can be fetched from and sent to it.

##### objectsInScene

Access Type: Private

Field Type: Binding List of Object

Initial Value: Empty

Size: 160 bytes

A list in which all objects in the Scene are stored. This list is then converted to one which displays information in the list box.

##### pointsInScene

Access Type: Private

Field Type: Binding List of Point

Initial Value: Null

Size: 160 bytes

A list in which all Points in the Scene are stored.

##### linesInScene

Access Type: Private

Field Type: Binding List of Line

Initial Value: Null

Size: 160 bytes

A list in which all Lines in the Scene are stored.

### anglesInScene

Access Type: Private

Field Type: Binding List of Angle

Initial Value: Null

Size: 160 bytes

A list in which all Angles in the Scene are stored.

## 2. Methods

### ObjectInspectorPanel (constructor)

Access Type: Public

Parameters: InputPanel *ip*

Return Type: N/A

Initialises the form, and calls *updateListBox*.

### updateListBox

Access Type: Public

Parameters: None

Return Type: Void

Fills *objectsInScene* with all the objects in the Scene. It then fills the list box with information about these objects. The function can be called by pressing the refresh button.

```
Public void updateListBox
```

```
    objectsInScene = new BindingList<object>()
```

```
    pointsInScene = getPoints()
```

```
    linesInScene = getLines()
```

```
    anglesInScene = getAngles()
```

```
Foreach Point p in pointsInScene
    objectsInScene.Add(p)
End
Foreach Line l in linesInScene
    objectsInScene.Add(l)
End
Foreach Angle a in anglesInScene
    objectsInScene.Add(a)
End
```

```
//Assign the produced list to the list box (to be
displayed to the user)
```

```
objectListBox.DataSource =
convertToString(objectsInScene)
```

End

### clearListBox

Access Type: Private

Parameters: None

Return Type: Void

Removes all items from the list box. Not calling this will cause the refresh of the list box to simply add the updated list of objects onto the end.

```
Private void clearListBox()
    objectListBox.DataSource = null
    objectListBox.Items.Clear()
```

End

### getPoints

Access Type: Private

Parameters: None

Return Type: Binding List of Point

Fetches *inputPanel.pointsInScene* and converts it into a Binding List, returning it.

The pseudo code for *getPoints* is the same as *InputPointPanel.getPoints*.

### getLines

Access Type: Private

Parameters: None

Return Type: Binding List of Line

Fetches *inputPanel.linesInScene* and converts it into a Binding List, returning it.

The pseudo code for *getLines* is the same as *InputLinePanel.getLines*.

### getAngles

Access Type: Private

Parameters: None

Return Type: Binding List of Angle

Fetches *inputPanel.anglesInScene* and converts it into a Binding List, returning it.

```
Private BindingList<Angle> getAngles()
```

```
    List<Angle> angles = inputPanel.anglesInScene
```

```
    BindingList<Angle> bindingAngles = new  
    BindingList<Angle>(angles)
```

```
    Return bindingAngles
```

```
End
```

### convertToString

Access Type: Private

Parameters: Binding List of Object *list*

Return Type: Binding List of String

Converts *objectsInScene* to useful information to be displayed in the list box.

```
Private BindingList<string>
convertToString(BindingList<object> list)

    BindingList<string> returnList = new
    BindingList<string>()

    //Produce data ready to be displayed to the user

    Foreach object o in list

        If o is Point

            Point p = (Point)o

            String message = "Point: ( " +
            p.firstPoint + " )      " +
            p.colorString

            returnList.Add(message)

        Else If o is Line

            Line l = (Line)o

            String message = "Line: ( " +
            l.firstPoint + " )      ( " +
            l.secondPoint + " )      " +
            l.colorString

            returnList.Add(message)

        Else If o is Angle

            Angle a = (Angle)o

            String message = "Angle between " +
            a.firstLine + " and " + a.secondLine +
            "      " + a.colorString

            returnList.Add(message)

        End

    End

    Return returnList

End
```

changeColorButton\_Click

Access Type: Private

Parameters: Object *sender*, EventArgs *e*

Return Type: Void

Opens the *colorChangePanel*, passing it the selected object.

deleteObjectButton\_Click

Access Type: Private

Parameters: Object *sender*, EventArgs *e*

Return Type: Void

Sets *inputPanel.objectToBeDeleted* (where *object* is the type of the object) to the selected object. The function then times-out for 10 milliseconds (to allow the Scene to delete the object), then clears and refreshes the list box.

```
Private void deleteObjectButton_Click(object sender,
EventArgs e)
```

```
    If objectListBox.Items.Count > 0
```

```
        Object objectToBeDeleted =
        objectsInScene[objectListBox.SelectedIndex]
```

```
        //For each object, tell the input panel to
        delete it, pause the current thread
        (allowing the object to be deleted before
        the list is updated), and then update the
        list
```

```
        If objectToBeDeleted is Point
```

```
            Point p = (Point)objectToBeDeleted
```

```
            inputPanel.pointToBeDeleted = p
```

```
            Thread.Sleep(10)
```

```
            clearListBox()
```

```
            updateListBox()
```

```
        Else If objectToBeDeleted is Line
```



```
        Line l = (Line)objectToBeDeleted
        inputPanel.lineToBeDeleted = l
        Thread.Sleep(10)
        clearListBox()
        updateListBox()
    Else If objectToBeDeleted is Angle
        Angle a = (Angle)objectToBeDeleted
        inputPanel.angleToBeDeleted = a
        Thread.Sleep(10)
        clearListBox()
        updateListBox()
    End
End
End
```

### 2.8.7 ColorChangePanel

#### 1. Variables

##### inputPanel

Access Type: Private

Field Type: InputPanel

Initial Value: Null

Size: 64 bits

A reference to Input Panel so that the data can be fetched from and sent to it.

##### objectInspectorPanel

Access Type: Private

Field Type: ObjectInspectorPanel

Initial Value: Null

Size: 64 bits

A reference to Object Inspector Panel so that the data can be fetched from and sent to it.

### colors

Access Type: Private

Field Type: List of Vector3

Initial Value: Empty

Size: 120 bytes

A list containing the colours in Vector3 form.

### colorStrings

Access Type: Private

Field Type: List of String

Initial Value: Empty

Size: 61 bytes

A list of names for each colour in *colors*, which is displayed in the list box.

### p

Access Type: Private

Field Type: Point

Initial Value: Null

Size: 8 bytes

The Point variable used throughout the panel if the object whose colour is being changed is a Point.

### l

Access Type: Private

Field Type: Line

Initial Value: Null

Size: 8 bytes

The Line variable used throughout the panel if the object whose colour is being changed is a Line.

a

Access Type: Private

Field Type: Angle

Initial Value: Null

Size: 8 bytes

The Angle variable used throughout the panel if the object whose colour is being changed is an Angle.

## 2. Methods

### ColorChangePanel (constructor)

Access Type: Public

Parameters: InputPanel *ip*, Object *ob*, ObjectInspectorPanel *oip*

Return Type: N/A

Initialises the form, initialises the colour lists (*colors*, *colorStrings*), and updates the list box.

### initialiseColors

Access Type: Private

Parameters: None

Return Type: Void

Initialises the colour lists: *colors*, *colorStrings*.

### updateListBox

Access Type: Private

Parameters: Object *ob*

Return Type: Void

Converts the object passed from *ObjectInspectorPanel* to displayable information, and fills the list box with *colorStrings*.

```
Private void updateListBox(object ob)
```

```
    If ob is Point
```

```
        p = (Point)ob
```

```
    Else If ob is Line
```

```
        l = (Line)ob
Else If ob is Angle
        a = (Angle)ob
End

//Produce intelligible data to display about the
//object which the user is trying to change the
//colour of.

If p != null
        Int index = colors.IndexOf(p.rawColor)
        currentRawColor = p.rawColor
        currentColorString = colorStrings[index]
        String identity = "Point: ( " + p.firstPoint
        + " )"
        identityLabel.Text = identity
Else If l != null
        Int index = colors.IndexOf(p.rawColor)
        currentRawColor = p.rawColor
        currentColorString = colorStrings[index]
        String identity = "Line: ( " + l.firstPoint
        + " )      ( " + l.secondPoint + " )"
        identityLabel.Text = identity
Else If a != null
        Int index = colors.IndexOf(p.rawColor)
        currentRawColor = p.rawColor
        currentColorString = colorStrings[index]
        String identity = "Angle between ( "
        + a.firstLine + " )      ( " + a.secondLine + "
        )"
        identityLabel.Text = identity
End
```

```
currentColorLabel.Text = currentColorString  
  
BindingList<string> displayStringList = new  
BindingList<string>(colorStrings)  
  
listBox.DataSource = displayStringList
```

End

#### selectColorButton\_Click

Access Type: Private

Parameters: Object *sender*, EventArgs *e*

Return Type: Void

Passes the object along with its new colour to Input Panel, waits 10 milliseconds and then updates *ObjectInspectorPanel*.

### 2.8.8 ColoredObject

#### 1. Variables

Total Size: 1132 bytes

##### color

Access Type: Public

Field Type: VBO of Vector3

Initial Value: Null

Size: 1 megabyte

The colour of the object in VBO form, allowing rendering by OpenGL.

##### rawColor

Access Type: Public

Field Type: Vector3

Initial Value: Null

Size: 12 bytes

The colour of the object in Vector3 form, allowing calculations to be performed on it by other parts of the program.

elements

Access Type: Public

Field Type: VBO of Int

Initial Value: Null

Size: 1 megabyte

A VBO required by OpenGL to know how many vertices it has to render for each object.

colors

Access Type: Protected

Field Type: List of Vector3

Initial Value:

(1, 0.5, 0)

(1, 1, 0)

(0.5, 1, 0)

(0, 1, 0)

(0, 1, 0.5)

(0, 0.5, 1)

(0, 0, 1)

(0.5, 0, 1)

(1, 0, 1)

(1, 0, 0.5)

[ref. 2]

Size: 120 bytes

The list of Vector3 values representing all available colours.

colorStrings

Access Type: Protected

Field Type: List of String

Initial Value:

Orange

Yellow

Lime

Green

Turquoise

Light Blue

Blue  
Purple  
Pink  
Magenta

Size: 61 bytes

The list of names of colours in *colors*, used to produce intelligible data for the user.

### 2.8.9 Point

Derives *ColoredObject*

#### 1. Variables

Total size: 2094 bytes

point

Access Type: Public

Field Type: VBO of Vector3

Initial Value: Null

Size: 1 megabytes

The VBO used, by OpenGL, to render the Point.

rawPoint

Access Type: Public

Field Type: Vector3

Initial Value: Null

Size: 12 bytes

The Vector3 coordinates of the Point, used by the program to perform calculations.

firstPoint

Access Type: Public

Field Type: String

Initial Value: Null

Size: 32 bytes

A string representing an intelligible form of the Point's coordinates, used to display the Point in various forms.

### colorString

Access Type: Public

Field Type: String

Initial Value: Null

Size: 50 bytes

The string representing the colour of the Point, used to display the colour of the Point in various forms.

## 2. Methods

### Point (constructor)

Access Type: Public

Parameters: Float x, Float y, Float z, Vector3 c

Return Type: N/A

Produces values for each of the Point's variables from the passed data.

## 2.8.10 Line

Derives *ColoredObject*

### 1. Variables

Total size: 2138 bytes

### line

Access Type: Public

Field Type: VBO of Vector3

Initial Value: Null

Size: 1 megabyte

The VBO representing the Line, used by OpenGL to render the Line.

### rawFirstPoint

Access Type: Public



Field Type: Vector3

Initial Value: Null

Size: 12 bytes

The Vector3 containing the coordinates of the first Point on the Line.

rawSecondPoint

Access Type: Public

Field Type: Vector3

Initial Value: Null

Size: 12 bytes

The Vector3 containing the coordinates of the second (final) Point on the Line.

firstPoint

Access Type: Public

Field Type: String

Initial Value: Null

Size: 32 bytes

A string representing an intelligible form of the first Point's coordinates, used to display the Point in various forms.

secondPoint

Access Type: Public

Field Type: String

Initial Value: Null

Size: 32 bytes

A string representing an intelligible form of the second Point's coordinates, used to display the Point in various forms.

colorString

Access Type: Public

Field Type: String

Initial Value: Null

Size: 50 bytes

The string representing the colour of the Line, used to display the colour of the Line in various forms.

## 2. Methods

### Line (constructor)

Access Type: Public

Parameters: Float *ox*, Float *oy*, Float *oz*, Float *fx*, Float *fy*, Float *fz*, Vector3 *c*

Return Type: N/A

Produces values for each of the Line's variables from the passed data.

### 2.8.11 Angle

Derives *ColoredObject*

#### 1. Variables

##### Line1

Access Type: Public

Field Type: Line

Initial Value: Null

Size: 8 bytes

One of the two Lines between which the Angle is drawn.

##### Line2

Access Type: Public

Field Type: Line

Initial Value: Null

Size: 8 bytes

The second of the two Lines between which the Angle is drawn.

##### point1

Access Type: Public

Field Type: Vector3

Initial Value: Null

Size: 12 bytes

The first point of two, between which the Line is drawn.

point2

Access Type: Public

Field Type: Vector3

Initial Value: Null

Size: 12 bytes

The second point of two, between which the Line is drawn.

line

Access Type: Public

Field Type: VBO of Vector3

Initial Value: Null

Size: 1 megabyte

The VBO required by OpenGL to render the Angle. The Angle is rendered as a straight line between points on each Line.

firstLine

Access Type: Public

Field Type: String

Initial Value: Null

Size: 32 bytes

A string representing an intelligible form of the first Line's coordinates, used to display the Line in various forms.

secondLine

Access Type: Public

Field Type: String

Initial Value: Null

Size: 32 bytes

A string representing an intelligible form of the second Line's coordinates, used to display the Line in various forms.

### colorString

Access Type: Public

Field Type: String

Initial Value: Null

Size: 50 bytes

The string representing the colour of the Angle, used to display the colour of the Angle in various forms.

## 2. Methods

### Angle (constructor)

Access Type: Public

Parameters: Line *l1*, Line *l2*, Vector3 *c*

Return Type: N/A

The constructor of the Angle class. It calculates the two points between which the Line representing the Angle is drawn.

### NormalizedDirection

Access Type: Private

Parameters: Line *l*

Return Type: Vector3

Calculates the unit vector (normalised vector) of a Line.

```
Private Vector3 NormalizedDirection(Line l)
```

```
    Vector3 direction = l.rawSecondPoint -  
    l.rawFirstPoint
```

```
    Double mod = Mod(direction)
```

```
    Float x = direction.x / (float)mod
```

```
    Float y = direction.y / (float)mod
```

```
Float z = direction.z / (float)mod  
Return new Vector3(x, y, z)  
End  
Mod  
Access Type: Private  
Parameters: Vector3 dVector  
Return Type: Double  
Calculates the modulus of a vector.  
Private double Mod(Vector3 dVector)  
    //Perform a 3D Pythagoras calculation  
    Double mod = SquareRoot(dVector.x ^ 2 + dVector.y  
        ^ 2 + dVector.z ^ 2)  
    Return mod  
End
```

### 2.8.12 Scene

#### 1. Variables

##### inputPanel

Access Type: Private

Field Type: InputPanel

Initial Value: Null

Size: 8 bytes

A reference to Input Panel so that the data inputted can be got from and sent to it.

##### width

Access Type: Private

Field Type: Int

Initial Value: 1000

Size: 32 bytes

The width of the window.

### Height

Access Type: Private

Field Type: Int

Initial Value: 500

Size: 32 bytes

The height of the window.

### program

Access Type: Private

Field Type: ShaderProgram

Initial Value: Null

The OpenGL variable responsible for part of the rendering pipeline which handles projection, view, and model matrices.

### points

Access Type: Private

Field Type: List of Point

Initial Value: Empty

Size: 160 bytes

A list of all the Points in the Scene.

### lines

Access Type: Private

Field Type: List of Line

Initial Value: Empty

Size: 160 bytes

A list of all the Lines in the Scene.

### angles

Access Type: Private

Field Type: List of Angle

Initial Value: Empty

Size: 160 bytes

A list of all the Angles in the Scene.

### colors

Access Type: Private

Field Type: Array of Vector3

Initial Value: Empty

Size: 120 bytes

An array of all the colours with which objects can be rendered.

### unusedColors

Access Type: Private

Field Type: List of Vector3

Initial Value: Empty

Size: 120 bytes

A list of the colours in *colors* which are not currently in use by another object.

### watch

Access Type: Private

Field Type: Stopwatch

Initial Value: Null

A stopwatch used to calculate the time elapsed over every loop of Glut's main loop. This is used to calculate constant changes in angles and positions when a key is held down.

### angleX

Access Type: Private

Field Type: Float

Initial Value: 0

Size: 32 bytes

The angle of rotation of the camera in the X axis.

angleY

Access Type: Private

Field Type: Float

Initial Value: 0

Size: 32 bytes

The angle of rotation of the camera in the Y axis.

angleZ

Access Type: Private

Field Type: Float

Initial Value: 0

Size: 32 bytes

The angle of rotation of the camera in the Z axis.

cameraX

Access Type: Private

Field Type: Float

Initial Value: 0

Size: 32 bytes

The X coordinate of the position of the camera.

cameraY

Access Type: Private

Field Type: Float

Initial Value: 0

Size: 32 bytes

The Y coordinate of the position of the camera.

cameraZ

Access Type: Private

Field Type: Float

Initial Value: 0



Size: 32 bytes

The Z coordinate of the position of the camera.

anaglyphToggled

Access Type: Private

Field Type: Bool

Initial Value: False

Size: 8 bytes

A Boolean determining whether the user has selected the Anaglyph to be rendered.

anaglyphCyan

Access Type: Private

Field Type: Vector3

Initial Value: (0, 1, 1)

Size: 12 bytes

The colour of cyan objects produced by the Anaglyph.

anaglyphRed

Access Type: Private

Field Type: Vector3

Initial Value: (1, 0, 0)

Size: 12 bytes

The colour of red objects produced by the Anaglyph.

VertexShader

Access Type: Private

Field Type: (Literal) String

Initial Value:

@”

//Takes vertex position and colour as inputs,  
producing a colour as an output

```
in vec3 vertexPosition;
in vec3 vertexColor;
out vec3 color;

//Defines the matrices for rendering
uniform mat4 projection_matrix;
uniform mat4 view_matrix;
uniform mat4 model_matrix;

void main(void)
{
    color = vertexColor

    //Calculates the position of the point in
    relation to the camera

    gl_Position = projection_matrix *
    view_matrix * model_matrix *
    vec4(vertexPosition, 1);
}
```

Size: 500 bytes

For a description of the VertexShader, see sec. 2.4.2.1.4.

### FragmentShader

Access Type: Private

Field Type: (Literal) String

Initial Value:

```
@
in vec3 color;
out vec4 fragment;

//Produces a fragment (a primitive object to be
rendered)
```

```

void main(void)
{
    //The Vector4 defines the colour alongside
    an alpha (transparency) value

    fragment = vec4(color, 1);
}”

```

Size: 150 bytes

For a description of the FragmentShader, see sec. 2.4.2.1.5.

## 2. Methods

### Scene (constructor)

Access Type: Public

Parameters: InputPanel *ip*

Return Type: N/A

Initialises Glut (if not already initialised – occurs from closing and re-opening a Scene), creates the Scene window, and starts the main loop.

Public Scene (InputPanel ip)

```

//If Glut is already initialised, don't try to re-
initialise it.

Int glutTime =
Glut.glutGet(Glut.GLUT_ELAPSED_TIME)

If glutTime <= 0
    Glut.glutInit()

End

//Initialise data for the window.

Glut.glutInitDisplayMode(Glut.GLUT_DOUBLE |
Glut.GLUT_DEPTH)

Glut.glutInitWindowSize(width, height)

Glut.glutCreateWindow("Scene")

```

```
Glut.glutSetOption(Glut.GLUT_ACTION_ON_WINDOW_CLOSE, 1)

Glut.glutCloseFunc(closeFunc)

Glut.glutIdleFunc(OnRenderFrame)

//Creates the Shader Program

program = new ShaderProgram(VertexShader,
FragmentShader)

program.Use()

//Define the camera (projection matrix)

program["projection_matrix"].SetValue(Matrix4.CreatePerspectiveFieldOfView(0.45f, (float)width / height, 0.1f, 1000f))

resetCamera()

//Initialise the stopwatch

watch = System.Diagnostics.Stopwatch.StartNew()

inputPanel = ip

//Start the main loop

Glut.glutMainLoop()

End
```

### Camera Manipulations

Contained within the *OnRenderFrame* function. Certain key presses correspond to changing the values of different variables which are then used in manipulating the camera. A list of corresponding keys and actions can be found at sec. 2.9.1.

The structure of the code is as follows:

```
If (Keyboard.IsKeyDown(Relevant Key))
```

```
Variable += deltaTime
```

```
//or Variable -= deltaTime depending on the key  
pressed
```

```
End
```

### OnRenderFrame

Access Type: Private

Parameters: None

Return Type: Void

Glut's main loop, which checks for key presses (corresponding to camera control), manipulates the camera, fetches inputted data from Input Panel, sends data to Input Panel, draws the Anaglyph (if *anaglyphToggled* is true), and draws all objects in the Scene.

```
Private void OnRenderFrame()
```

```
//Pause the stopwatch while time calculations are  
made
```

```
watch.Stop()
```

```
float deltaTime = (float)watch.ElapsedTicks /  
System.Diagnostics.Stopwatch.Frequency
```

```
watch.Restart()
```

```
//Restart the stopwatch once time calculations  
have been made
```

```
//Check for keypresses at this point. See sec. 8
```

```
//Update the view of the scene in case the window  
has been resized
```

```
Gl.Viewport(0, 0,  
Glut.glutGet(Glut.GLUT_WINDOW_WIDTH),  
Glut.glutGet(Glut.GLUT_WINDOW_HEIGHT))
```

```
//Clear the previous frame buffers
```

```
Gl.Clear(ClearBufferMask.ColorBufferBit |
ClearBufferMask.DepthBufferBit)

//Get all inputted points

List<Vector3> inputtedPointsRaw =
inputPanel.getPoints()

Foreach Vector3 v in inputtedPointsRaw

    Point p = new Point(v.x, v.y, v.z,
getColor())

    points.Add(p)

End

//Get all inputted lines

List<Vector3[]> inputtedLinesRaw =
inputPanel.getLines()

Foreach Vector3[] in inputtedLinesRaw

    Line l = new Line(v[0].x, v[0].y, v[0].z,
v[1].x, v[1].y, v[1].z, getColor())

    lines.Add(l)

End

//Get all inputted angles

List<Vector3[]> inputtedAnglesRaw =
inputPanel.getAngles()

Foreach Vector3[] v in inputtedAnglesRaw

    Line line1 = new Line(v[0].x, v[0].y,
v[0].z, v[1].x, v[1].y, v[1].z, colors[0])

    Line line2 = new Line(v[2].x, v[2].y,
v[2].z, v[3].x, v[3].y, v[3].z, colors[0])

    Angle a = new Angle(line1, line2,
getColor())

    angles.Add(a)
```

End

```
//Get all inputted points where their colour is predefined
```

```
List<Vector3[]> inputtedPoints =  
inputPanel.getPointsWithColor()
```

```
Foreach Vector3[] v in inputtedPoints
```

```
    Point p = new Point(v[0].x, v[0].y, v[0].z,  
        v[1])
```

```
    points.Add(p)
```

End

```
//Get all inputted lines where their colour is predefined
```

```
List<Vector3[]> inputtedLines =  
inputPanel.getLinesWithColor()
```

```
Foreach Vector3[] v in inputtedLines
```

```
    Line l = new Line(v[0].x, v[0].y, v[0].z,  
        v[1].x, v[1].y, v[1].z, v[2])
```

```
    lines.Add(l)
```

End

```
//Get all inputted angles where their colour is predefined
```

```
List<Vector3[]> inputtedAngles =  
inputPanel.getAnglesWithColor()
```

```
Foreach Vector3[] v in inputtedAngles
```

```
    Line line1 = new Line(v[0].x, v[0].y,  
        v[0].z, v[1].x, v[1].y, v[1].z, colors[0])
```

```
    Line line2 = new Line(v[2].x, v[2].y,  
        v[2].z, v[3].x, v[3].y, v[3].z, colors[0])
```

```
    Angle a = new Angle(line1, line2, v[4])
```

```
        angles.Add(a)
    End

    //Check for any point to be deleted
    If inputPanel.pointToBeDeleted != null
        Point pointToBeDeleted =
            inputPanel.pointToBeDeleted

        //If the point still exists, remove it from
        the list of points (delete the point)

        If points contains pointToBeDeleted
            Remove pointToBeDeleted from points
        End

        inputPanel.pointToBeDeleted = null
    End

    //Check for any line to be deleted
    If inputPanel.lineToBeDeleted != null
        Line lineToBeDeleted =
            inputPanel.lineToBeDeleted

        //If the line still exists, remove it from
        the list of lines (delete the line)

        If lines contains lineToBeDeleted
            Remove lineToBeDeleted from lines
        End

        inputPanel.lineToBeDeleted = null
    End

    //Check for any angle to be deleted
    If inputPanel.angleToBeDeleted != null
```



```
Angle angleToBeDeleted =
inputPanel.angleToBeDeleted

//If the angle still exists, remove it from
the list of angles (delete the angle)

If angles contains angleToBeDeleted
    Remove angleToBeDeleted from angles
End

inputPanel.angleToBeDeleted = null

End

//Check if any existing object needs to have its
colour changed

Object colorChangeObject =
inputPanel.getColorChangeObject()

If colorChangeObject != null
    Vector3 newColor = inputPanel.getNewColor()

    //Find the type of object and remove it from
    the list of current objects

    If colorChangeObject is Point
        Point p = (Point)colorChangeObject
        If p is in points
            Remove p from points

            Create p as a new point with the
            same coordinates but a new
            colour, newColor

            Add p to points
        End
    Else If colorChangeObject is Line
        Line l = (Line)colorChangeObject
        If l is in lines
```

```
        Remove l from lines

        Create l as a new line with the
        same coordinates but a new
        colour, newColor

        Add l to lines

    End

Else If colorChangeObject is Angle

    Angle a = (Angle)colorChangeObject

    If a is in angles

        Remove a from angles

        Create a as a new angle with the
        same coordinates but a new
        colour, newColor

        Add a to angles

    End

End

End

//Pass current objects in the scene to inputPanel
inputPanel.setPointsInScene(points)
inputPanel.setLinesInScene(lines)
inputPanel.setAnglesInScene(angles)

//Perform Anaglyph calculations and draws here
anaglyphToggled = inputPanel.getAnaglyphToggled()
if(anaglyphToggled == true)

    //Calculate the true camera position (not
    normally stored - the camera position is
    normally stored as a position and angle)

    Vector3 cameraPos = new Vector3(
```

```
        cameraX - (Sin(angleY) * (10 -
        cameraZ)),

        cameraY + (Sin(angleX) * (10 - cameraZ
        )),

        10 - cameraZ)

Double convergenceDepth = 10

//Calcualte the distance between eyes

Double interocularDistance =
((convergenceDepth / 2) +
inputPanel.getAnaglyphAdjust()) / 30

//Calculate positions for each eye
Vector3 leftEye = new Vector3(cameraPos.x -
interocularDistance, cameraPos.y,
cameraPos.z)

Vector3 rightEye = new Vector3(cameraPos.x +
interocularDistance, cameraPos.y,
cameraPos.z)

Vector3 leftPoint

Vector3 rightPoint

//Calculate the anaglyph points for a point
and render them

Foreach Point p in points

    leftPoint =
    calculateAnaglyphPoint(p.rawPoint,
    leftEye)

    rightPoint =
    calculateAnaglyphPoint(p.rawPoint,
    rightEye)

    drawPoint(new Point(leftPoint,
    anaglyphCyan))

    drawPoint(new Point(rightPoint,
    anaglyphRed))
```

End

```
//Calculate the anaglyph lines for a line  
and render them
```

```
Foreach Line l in lines
```

```
    leftPoint =  
    calculateAnaglyphPoint(l.rawFirstPoint  
    , leftEye)
```

```
    rightPoint =  
    calculateAnaglyphPoint(l.rawSecondPoin  
    t, leftEye)
```

```
    Line leftLine = new Line(leftPoint,  
    rightPoint, anaglyphCyan)
```

```
    leftPoint =  
    calculateAnaglyphPoint(l.rawFirstPoint  
    , rightEye)
```

```
    rightPoint =  
    calculateAnaglyphPoint(l.rawSecondPoin  
    t, rightEye)
```

```
    Line rightLine = new Line(leftPoint,  
    rightPoint, anaglyphRed)
```

```
    drawLine(leftLine)
```

```
    drawLine(rightLine)
```

End

```
//Calculate the anaglyph angle line for an  
angle and render them
```

```
Foreach Angle a in angles
```

```
    leftPoint =  
    calculateAnaglyphPoint(a.point1,  
    leftEye)
```

```
        rightPoint =  
        calculateAnaglyphPoint(a.point2,  
        leftEye)  
  
        Line leftLine = new Line(leftPoint,  
        rightPoint, anaglyphCyan)  
  
        leftPoint =  
        calculateAnaglyphPoint(a.point1,  
        rightEye)  
  
        rightPoint =  
        calculateAnaglyphPoint(a.point2,  
        rightEye)  
  
        Line rightLine = new Line(leftPoint,  
        rightPoint, anaglyphRed)  
  
        drawLine(leftLine)  
  
        drawLine(rightLine)  
  
    End  
Else  
    //Draw all objects if the anaglyph isn't  
    being rendered  
  
    Foreach Point p in points  
        drawPoint(p)  
    End  
  
    Foreach Line l in lines  
        drawLine(l)  
    End  
  
    Foreach Angle a in angles  
        drawAngle(a)  
    End  
End
```

```
//Perform manipulations on the camera based off
key presses
```

```
program["view_matrix"].SetValue(Matrix4.CreateTranslation(new Vector3(0, 0, cameraZ)) *
Matrix4.CreateRotationX(angleX) *
Matrix4.CreateRotationY(angleY) *
Matrix4.CreateRotationZ(angleZ) *
Matrix4.LookAt(new Vector3(0, 0, 10), new
Vector3(cameraX, cameraY, 0), Vector3.Up));
```

```
Glut.glutSwapBuffers()
```

End

### closeFunc

Access Type: Private

Parameters: None

Return Type: Void

Handles the event when the Scene is closed. It tells the Input Panel that the Scene is closing, so that it can perform its own relevant functions.

```
Private void closeFunc()
    inputPanel.closeScene()
    Glut.glutLeaveMainLoop()
```

End

### resetCamera

Access Type: Private

Parameters: None

Return Type: Void

Resets both the view matrix, and the camera's position and angle variables.

```
Private void resetCamera()
    Program["view_matrix"].SetValue(Matrix4.LookAt(new
Vector3(0, 0, 10), Vector3.Zero, Vector3.Up))
```

```
angleX = 0
angleY = 0
angleZ = 0
cameraX = 0
cameraY = 0
cameraZ = 0
```

End

### getColor

Access Type: Private

Parameters: None

Return Type: Vector3

Calls *setUpUnusedColors*, then returns a random colour from *unusedColors*. If *unusedColors* is empty, it will return a random colour from *colors* instead.

```
Private Vector3 getColor()
    SetUpUnusedColors()

    //If there are unused colours, chose a random
    number and use that number colour

    If unusedColors.Count > 0

        Random r = new Random()

        Int randInt = r.Next(0, unusedColors.Count -
        1)

        Return unusedColors[randInt]

    Else

        //Otherwise chose a random colour out of all
        the colours

        Random r = new Random()

        Int randInt = r.Next(0, 9)

        Return colors[randInt]

    End
```

End

### setUpUnusedColors

Access Type: Private

Parameters: None

Return Type: Void

Checks the colours of every object in the Scene, producing a list (*unusedColors*) of colours that are currently not in use.

```
Private void setUpUnusedColors()
```

```
    //Add every colour to the list of unused colours
```

```
    Foreach Vector3 c in colors
```

```
        If NOT unusedColors.Contains(c)
```

```
            unusedColors.Add(c)
```

```
        End
```

```
    End
```

```
    //For each currently used colour, remove it from  
    the list of unused colours
```

```
    Foreach Point p in points
```

```
        If unusedColors.Contains(p.rawColor)
```

```
            unusedColors.Remove(p.rawColor)
```

```
        End
```

```
    End
```

```
    Foreach Line l in lines
```

```
        If unusedColors.Contains(l.rawColor)
```

```
            unusedColors.Remove(l.rawColor)
```

```
        End
```

```
    End
```

```
End
```

### resetModelMatrix



Access Type: Private

Parameters: None

Return Type: Void

Resets the Shader Program's model matrix (sets it to zero).

```
Private void resetModelMatrix

    program["model_matrix"].SetValue(Matrix4.CreateTranslation(Vector3.Zero))

End
```

### drawPoint

Access Type: Private

Parameters: Point *point*

Return Type: Void

Resets the model matrix, then draws the Point on the screen.

```
Private void drawPoint(Point point)

    resetModelMatrix()

    //Set up the buffers to render the point

    Gl.BindBufferToShaderAttribute(point.point,
    program, "vertexPosition")

    Gl.BindBufferToShaderAttribute(point.color,
    program, "vertexColor")

    Gl.BindBuffer(point.elements)

    //Render the point

    Gl.DrawElements(BeginMode.Points,
    point.elements.Count,
    DrawElementsType.UnsignedInt, IntPtr.Zero)
```

End

### drawLine

Access Type: Private

Parameters: Line *line*

Return Type: Void

Resets the model matrix, then draws the Line on the screen.

```
Private void drawLine(Line line)
```

```
    resetModelMatrix()
```

```
    //Set up the buffers to render the line
```

```
    Gl.BindBufferToShaderAttribute(line.line, program,  
    "vertexPosition")
```

```
    Gl.BindBufferToShaderAttribute(line.color,  
    program, "vertexColor")
```

```
    Gl.BindBuffer(line.elements)
```

```
    //Render the line
```

```
    Gl.DrawElements(BeginMode.Lines,  
    line.elements.Count, DrawElementsType.UnsignedInt,  
    IntPtr.Zero)
```

End

### drawAngle

Access Type: Private

Parameters: Angle *angle*

Return Type: Void

Resets the model matrix, then draws the Angle on the screen. An angle is rendered as a line.

```
Private void drawAngle(Angle angle)
```

```
    resetModelMatrix()
```

```
    //Set up the buffers to render the angle
```

```
    Gl.BindBufferToShaderAttribute(angle.angle,  
    program, "vertexPosition")
```

```
Gl.BindBufferToShaderAttribute(angle.color,  
program, "vertexColor")  
  
Gl.BindBuffer(angle.elements)  
  
//Render the angle  
  
Gl.DrawElements(BeginMode.LineStrip,  
angleElements.Count, DrawElementsType.UnsignedInt,  
IntPtr.Zero)
```

End

## 2.9 Identification of Processes and Algorithms

For descriptions and pseudo code of algorithms, see sec. 2.8.

### 2.9.1 Camera Manipulations

The user controls the camera using key presses. The presses are handled by the *OnRenderFrame* function, within the Scene. The camera manipulations include rotation, translation, zooming, and resetting the camera to its original position.

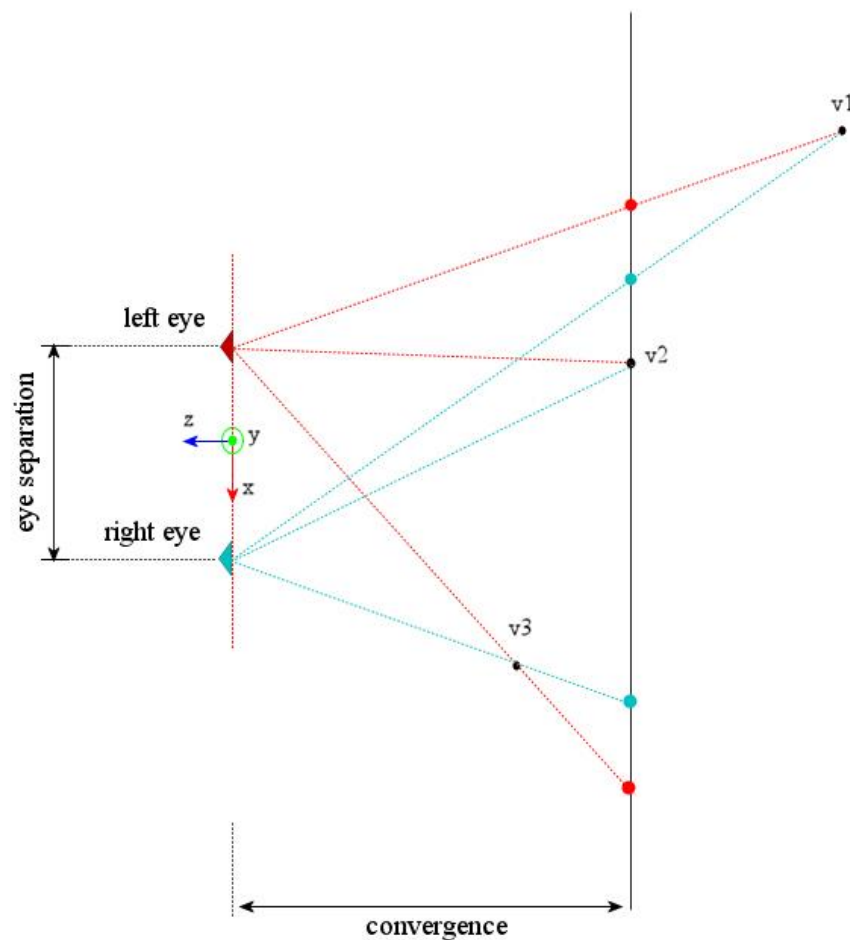
Manipulation	Key Press
Translation in x direction	Right arrow key, Left arrow key
Translation in y direction	Up arrow key, Down arrow key
Rotation in horizontal plane	NumPad 4, NumPad 6
Rotation in vertical plane	NumPad 8, NumPad 2
Rotation in diagonal plane	NumPad 7, NumPad 9
Zoom	PageUp, PageDown
Reset Camera	Home

## 2.10 Anaglyph Theory

An anaglyph uses stereoscopy to create the illusion of depth in an image. It involves creating two offset two dimensional images, one for the left eye, and one for the right eye. The parallax effect from the two images deceives the brain into thinking the object is three dimensional. [ref. 13] [ref. 14]

Because the anaglyph works on the basis that different images are rendered for each eye, we have to define two cameras. The distance between these two cameras is the *interocular distance*. Because we need to have a depth control, we define the perpendicular distance from the cameras to the plane at which objects will appear to have the same depth as the *convergence distance*. From this point we calculate the amount of offset each render (red, cyan) has. Because of this the convergence distance is defined as in the viewing direction.

As the user's distance from the screen affects the interocular distance, the user is provided with a slide to adjust the interocular distance. This slide is contained within the Input Panel, and is available to the user only when the Anaglyph is rendered.

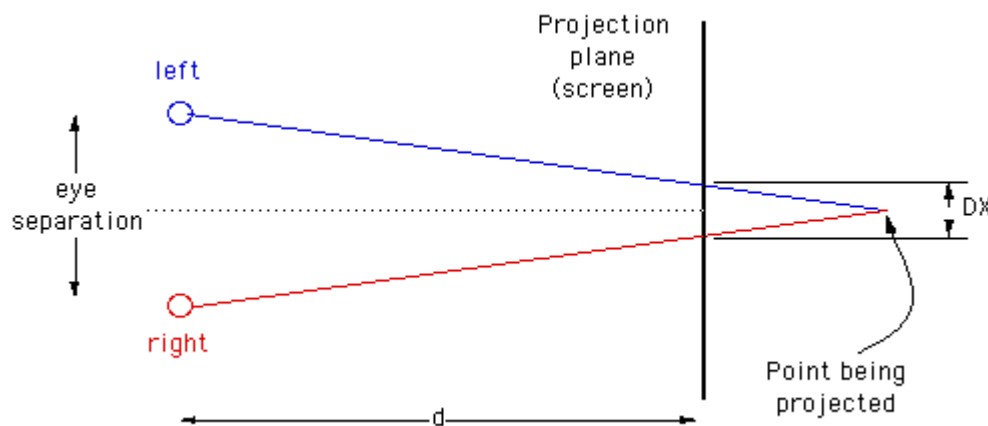


In the above image,  $v_1$ ,  $v_2$  and  $v_3$  are points of parallax. Because we are taking our convergence line at  $v_2$ ,  $v_1$  is defined as negative, and  $v_3$  as positive vertices.

The distance between where vertices from both eyes to the same parallax point meet the convergence line is the measure of parallax. For points further away the intersection points (between each eye and the parallax points) are the same way round as the eyes. For example the lines joining the eyes to  $v_1$ : the red dot is above the cyan dot (just as the left eye – red – is above the right eye – cyan). This is known as positive parallax. Vertex  $v_2$  has zero parallax, as the red and cyan dots are in the same place. Vertex  $v_3$  has negative parallax, as the red and cyan dots have flipped position (in relation to the eyes).

Points rendered with positive parallax will appear deeper than points rendered with zero or negative parallax. Points rendered with negative parallax will appear to come out of the screen. [ref. 5]

In order to measure the parallax (the distance between the points on the convergence line which join the same parallax point), or offset, for a parallax point we use similar triangles.



[ref. 6]

Once we have calculated the parallax in each direction ( $x$ ,  $y$ ,  $z$ ), we can calculate the position of each anaglyph point based on the position from the left and right eyes. The point is then immediately rendered.

In order to produce an anaglyph line we perform the above method to the points and the end of each line, and then draw the line between each corresponding (left eye, right eye) anaglyph point.

For implementation of anaglyph algorithms, including pseudo code, see sec. 2.8.

The actual code used to calculate and render the anaglyph is found in sec. 8.13.

## 2.11 Security

The data used by the program is of no use to anyone but the user, so system security is not an issue.

Should the teacher rely on using the same Scene for multiple lessons, they can save it onto the school's network, which is regularly backed up.

## 2.12 Test Strategy

### 2.12.1 Black-box Testing

Black-box testing is the best way of testing the main purpose of the program – visualisation. A range of values for each object will be inputted and their render will be checked to make sure it is correct. Camera manipulations will also be tested intuitively.

### 2.12.2 White-box Testing

Due to a number of different forms, problems may arise in forms being used when they aren't fully up to date with the Scene. White-box testing (finding issues with object instance continuity throughout the project) is the best way of identifying these issues.

### 2.12.3 Stress Testing

The maximum number of objects to be rendered at one time by the user is 20. It is important to make sure the program can run smoothly with this number of objects in the Scene.

## 3. System Testing

---

### 3.1 Design of test plan

Three kinds of test strategy are to be carried out on the software:

- i. Black box testing
  - a. Button functionality
  - b. Render functionality
  - c. Object Inspector functionality
  - d. Anaglyph rendering and adjustment
  - e. File Handling functionality
  - f. Camera functionality
- ii. White box testing
  - a. Erroneous Data
    - i. Input Forms
    - ii. Files
  - b. Extreme Data
    - i. Normal render
    - ii. Anaglyph render
    - iii. Input Forms
  - c. Object existence continuity across forms
- iii. Stress testing
  - a. Normal render
  - b. Anaglyph render

### 3.2 Black box testing

Black box testing involves inputting data, and viewing the output, and considering whether that input should lead to that output.

#### Test 1

[sec. 3.1.i.a]

Description: Pressing the 'Create New Scene' button

Expected result: A new scene will be created

Pass/Fail: Pass

#### Test 2

[sec. 3.1.i.a]

Description: Pressing 'Add New Point'

Expected result: The Input Point panel will be created

Pass/Fail: Pass



### Test 3

[sec. 3.1.i.a]

Description: Pressing 'Add New Line'

Expected result: The Input Line panel will be created

Pass/Fail: Pass

### Test 4

[sec. 3.1.i.a]

Description: Pressing 'Use Existing Points' when at least two points exist in the scene

Expected result: The Input Line From Points panel will be created

Pass/Fail: Pass

### Test 5

[sec. 3.1.i.a]

Description: Pressing 'Add New Angle'

Expected result: The Input Angle panel will be created

Pass/Fail: Pass

### Test 6

[sec. 3.1.i.a]

Description: Pressing 'Open Object Inspector'

Expected result: The Object Inspector panel will be created

Pass/Fail: Pass

### Test 7

[sec. 3.1.i.a]

Description: Selecting an object in the Object Inspector and pressing 'Change Colour'

Expected result: The Colour Change panel will be created

Pass/Fail: Pass

Test 8

[sec. 3.1.i.a]

Description: Pressing 'Close Scene'

Expected result: The current scene will be closed

Pass/Fail: Pass

Test 9

[sec. 3.1.i.a]

Description: Pressing 'Open Scene'

Expected result: The open file window will be created

Pass/Fail: Pass

Test 10

[sec. 3.1.i.a]

Description: Pressing 'Save Scene'

Expected result: The save file window will be created

Pass/Fail: Pass

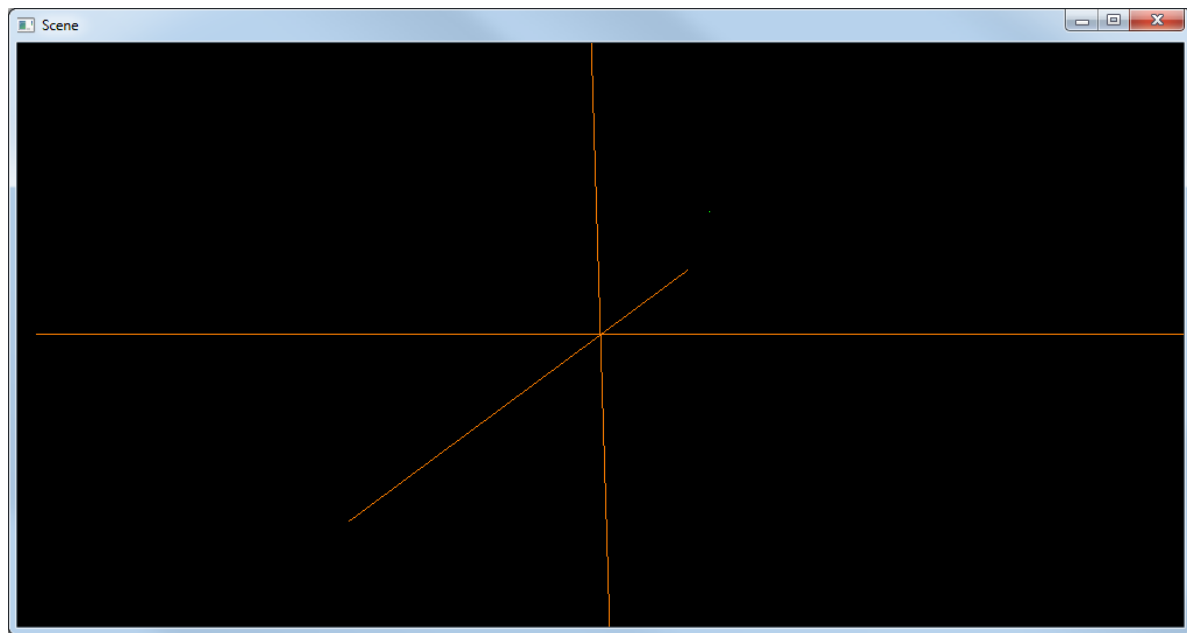
Test 11

[sec. 3.1.i.b]

Description: Input a point within the default view frame

Expected result: The point is rendered at the coordinates inputted (axis lines already drawn as a reference)

Pass/Fail: Pass



The point in this test had coordinate  $\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ .

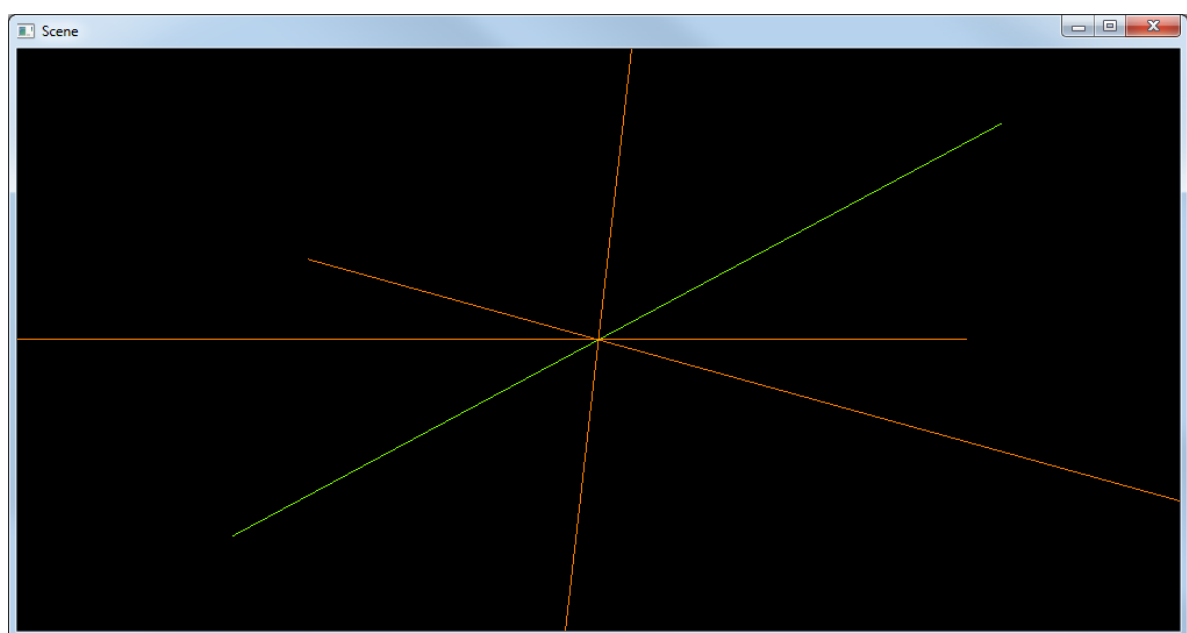
### Test 12

[sec. 3.1.i.b]

Description: Input the equation of a line within the default view frame

Expected result: The line is rendered correctly (axis lines already drawn as a reference)

Pass/Fail: Pass



The line in this test has equation  $\begin{pmatrix} -2 \\ -2 \\ -2 \end{pmatrix} + 4 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$  ( $y = x = z$  between -2 and 2).

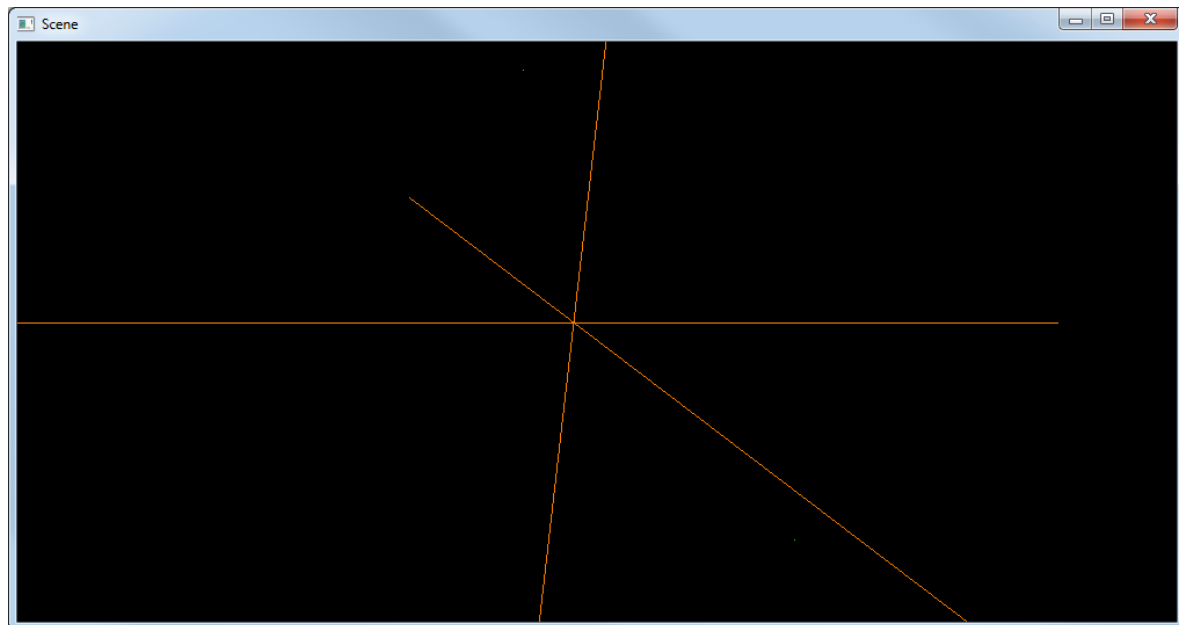
### Test 13

[sec. 3.1.i.b]

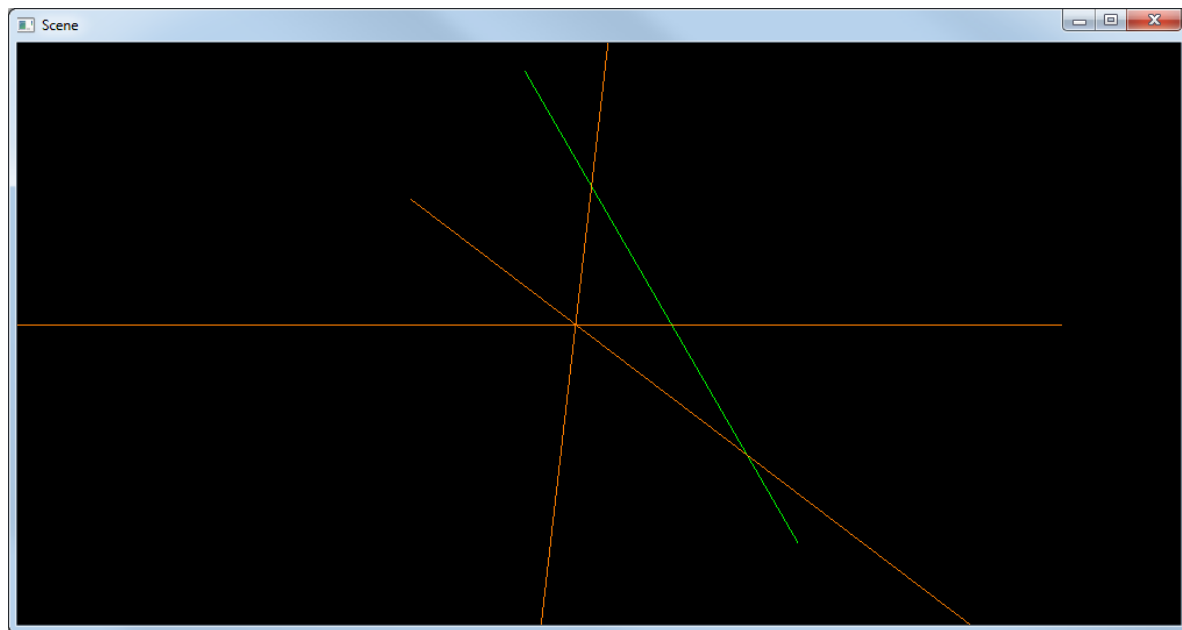
Description: Input two points within the default view frame, and select them to have the line between them drawn

Expected result: The line is drawn between the two points (axis lines already drawn as a reference)

Pass/Fail: Pass



The scene with only the two points added.



The scene with the line added through the Input Line From Points panel.

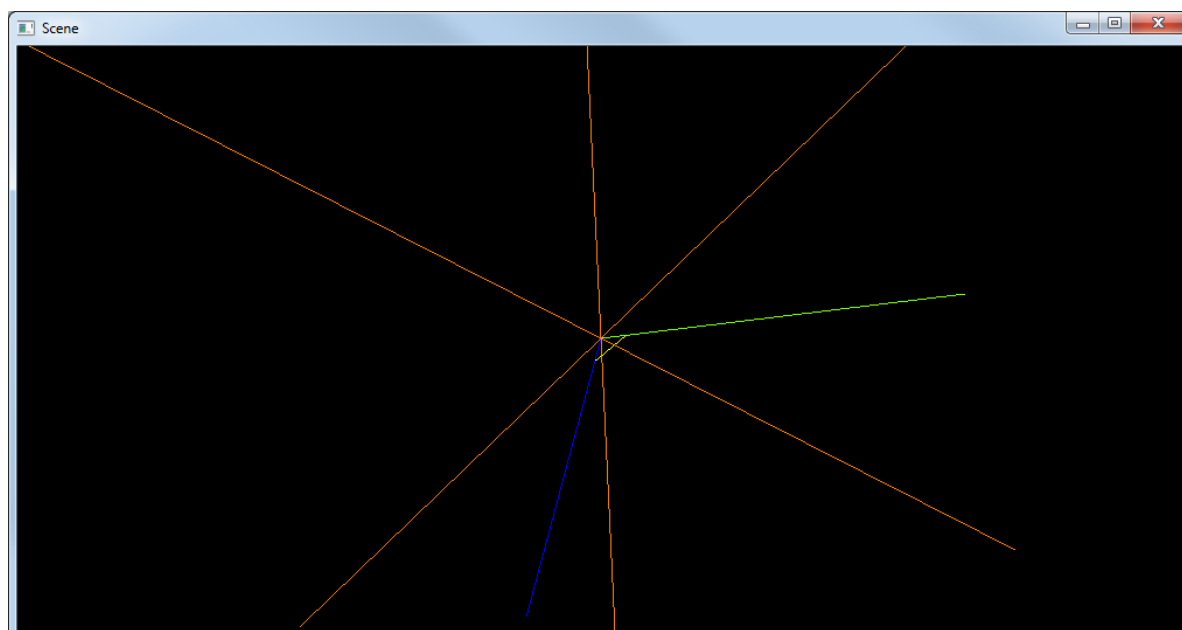
#### Test 14

[sec. 3.1.i.b]

Description: Add two lines and have the angle between them rendered

Expected result: A straight line will be drawn from points on each line to the other line.

Pass/Fail: Pass



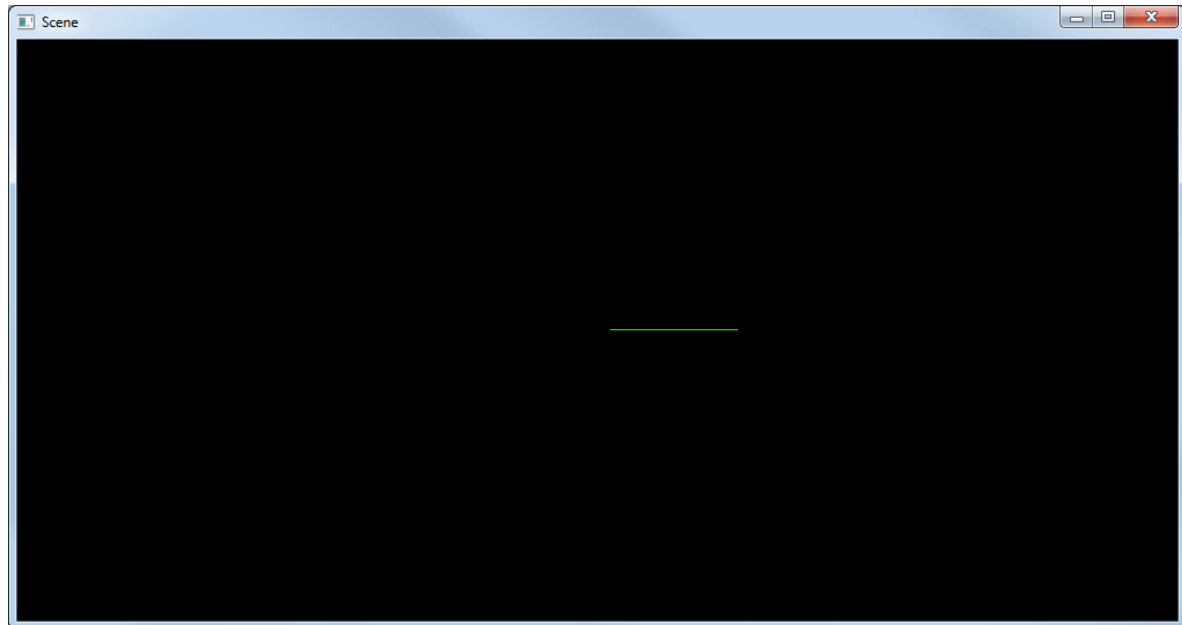
### Test 15

[sec. 3.1.i.c]

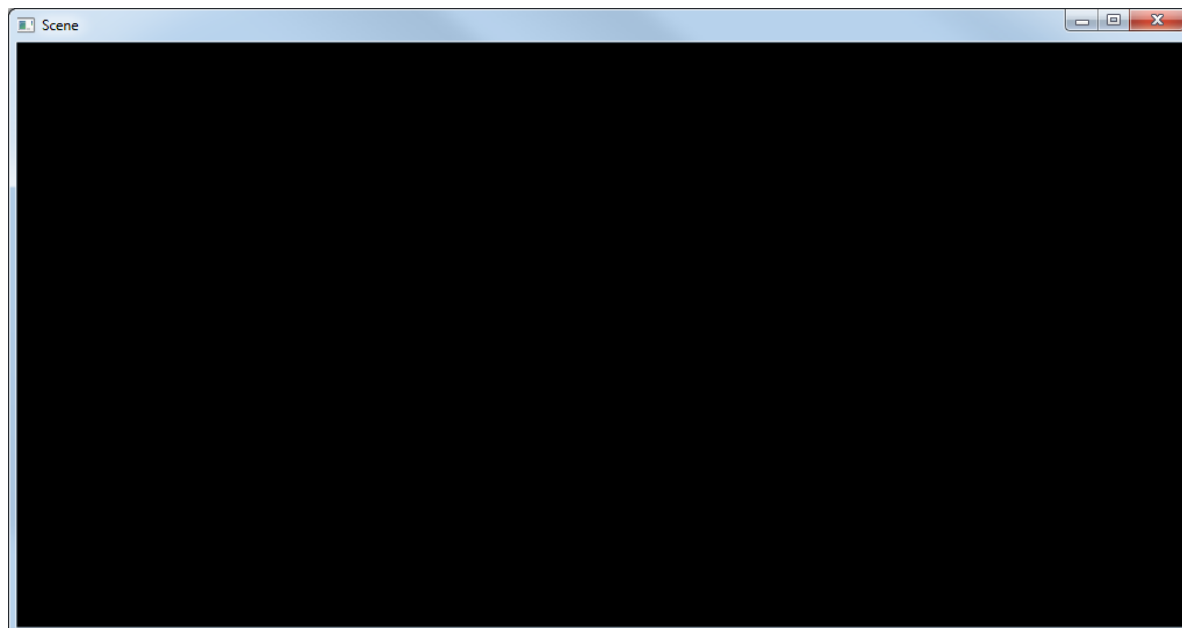
Description: Pressing 'Delete Object' with a selected object in the Object Inspector

Expected result: The object will be deleted

Pass/Fail: Pass



A simple inputted line.



The line is removed when selected and 'Delete Object' is pressed.

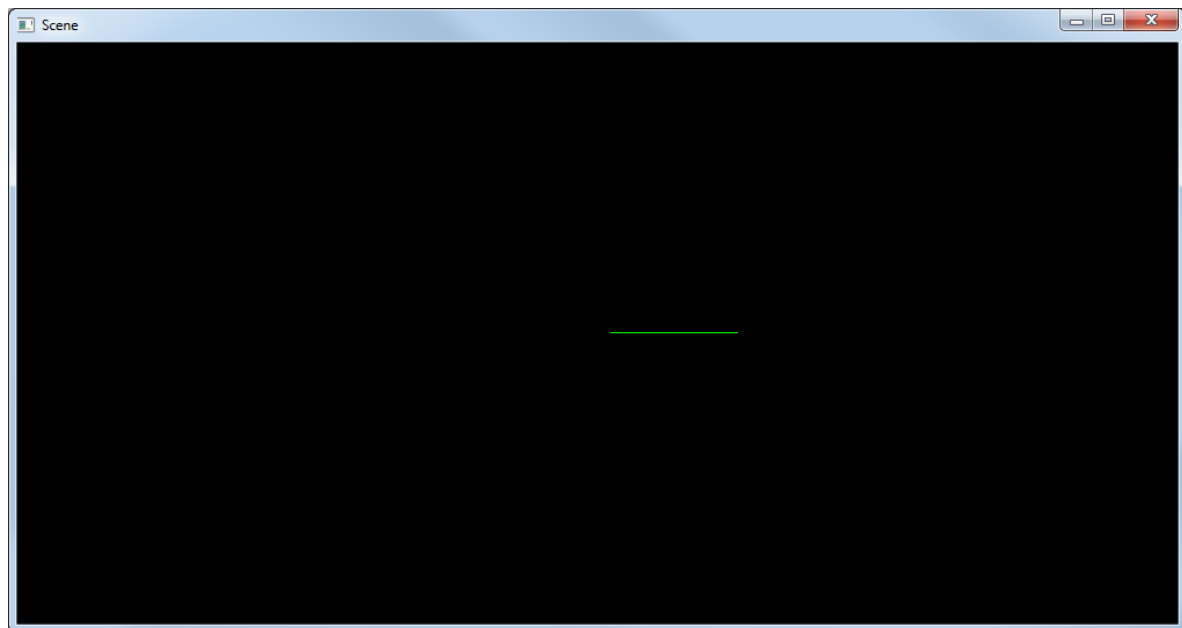
### Test 16

[sec. 3.1.i.c]

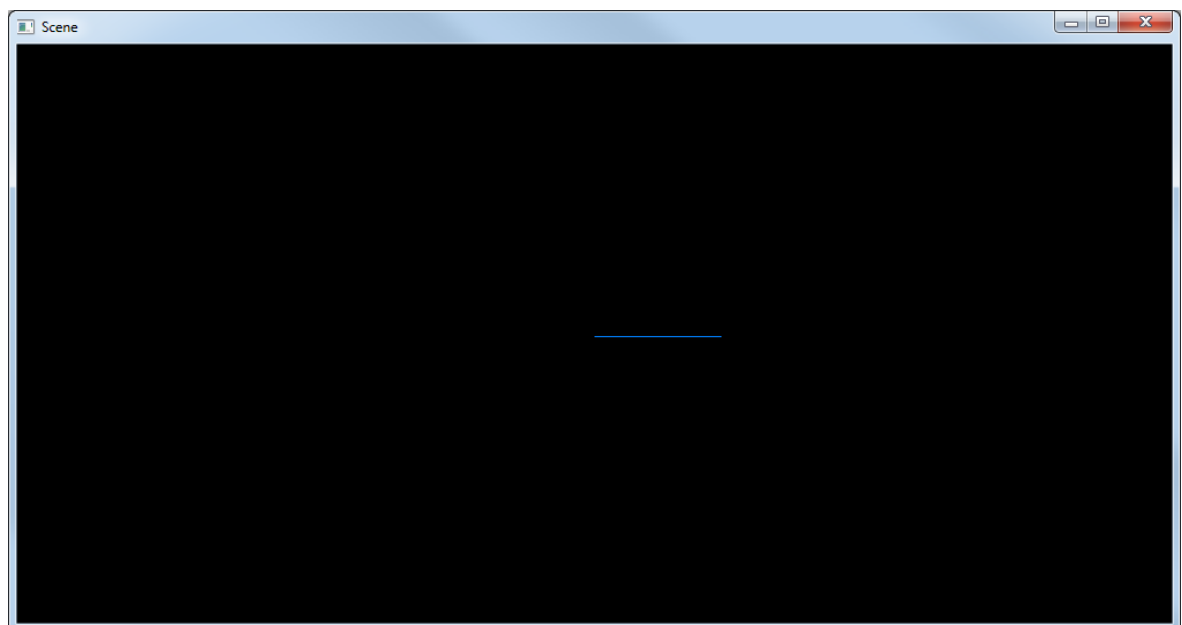
Description: Selecting and submitting a new colour for an object

Expected result: The colour of that object is changed to the new selected colour

Pass/Fail: Pass



The scene prior to changing the colour.



The scene after the colour of the object was changed to Light Blue.

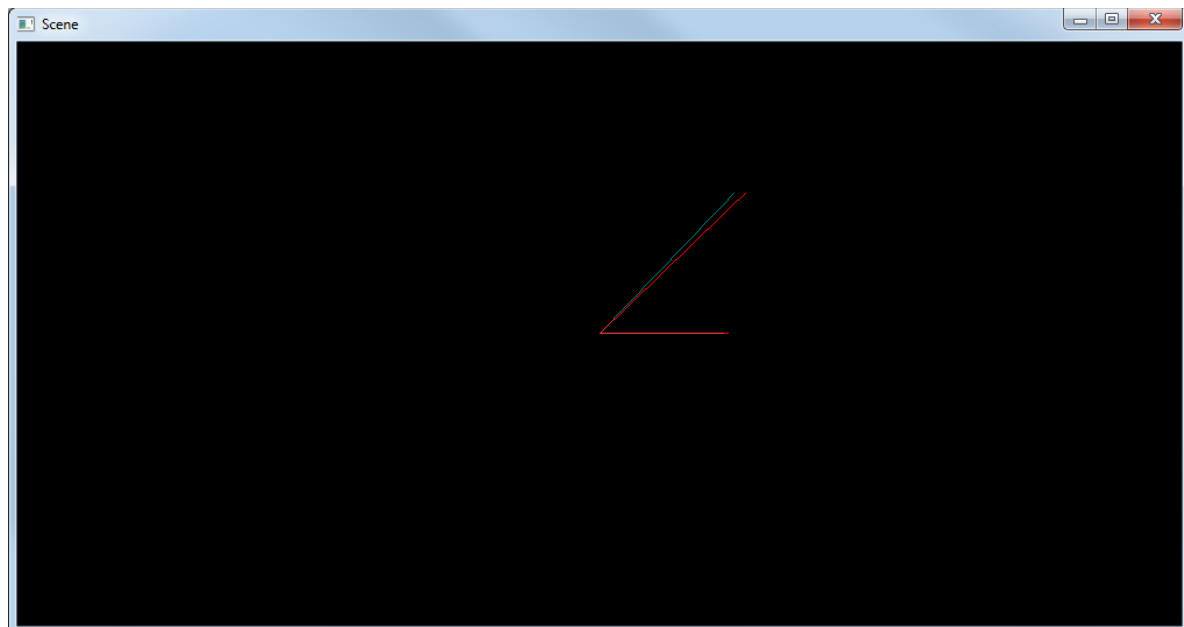
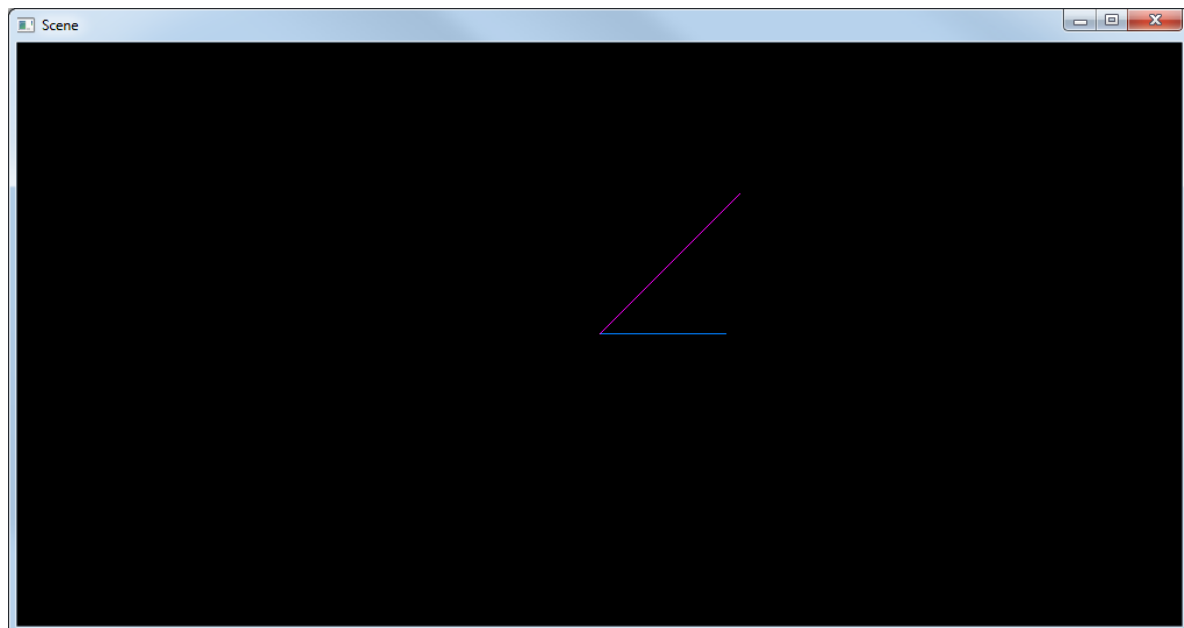
Test 17

[sec. 3.1.i.d]

Description: Checking the anaglyph check box

Expected result: The previous object disappear, and anaglyph objects are rendered.

Pass/Fail: Pass





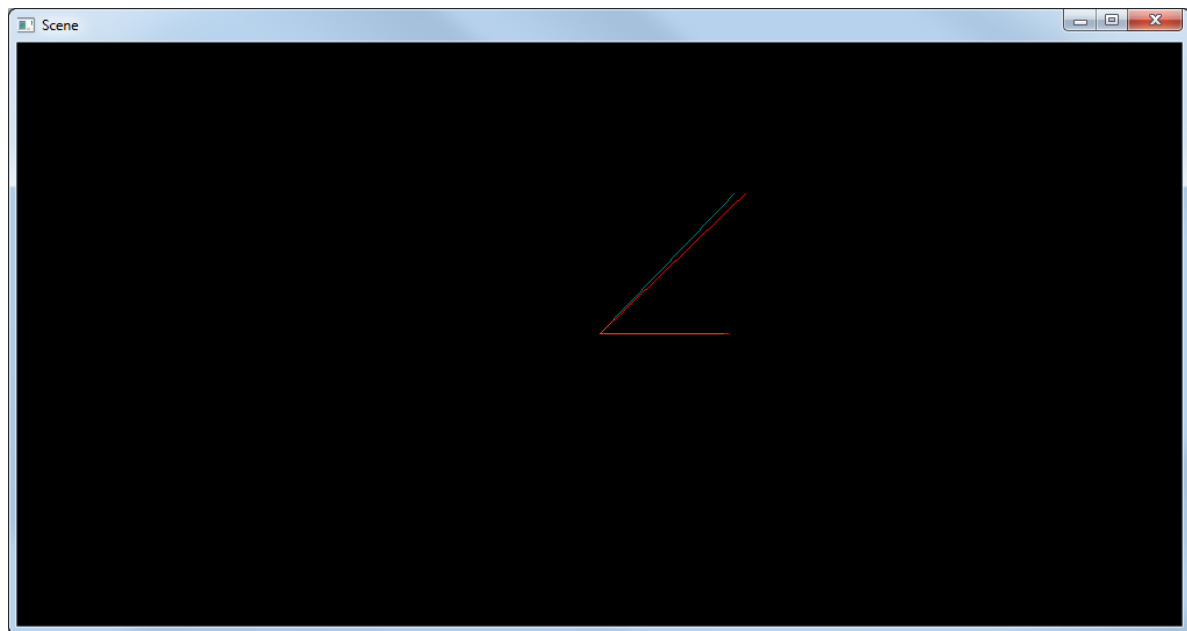
Test 18

[sec. 3.1.i.d]

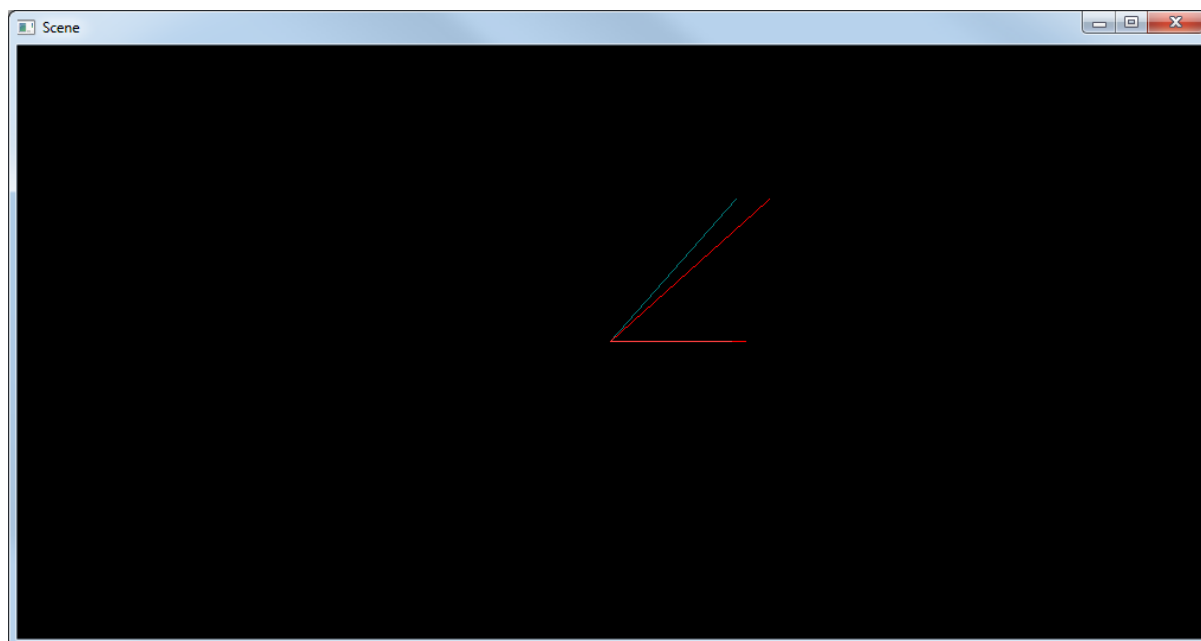
Description: Sliding the anaglyph adjust slider

Expected result: Separation of objects increase as the slider value increases

Pass/Fail: Pass



The slider at the minimal position.



The slider at the maximum position.

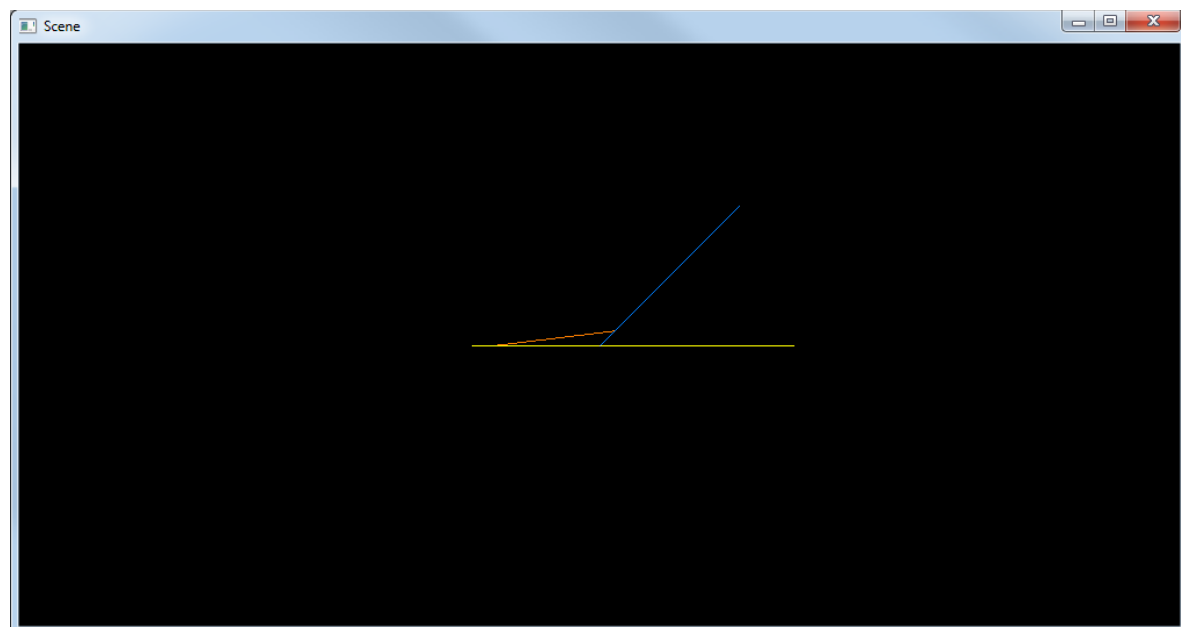
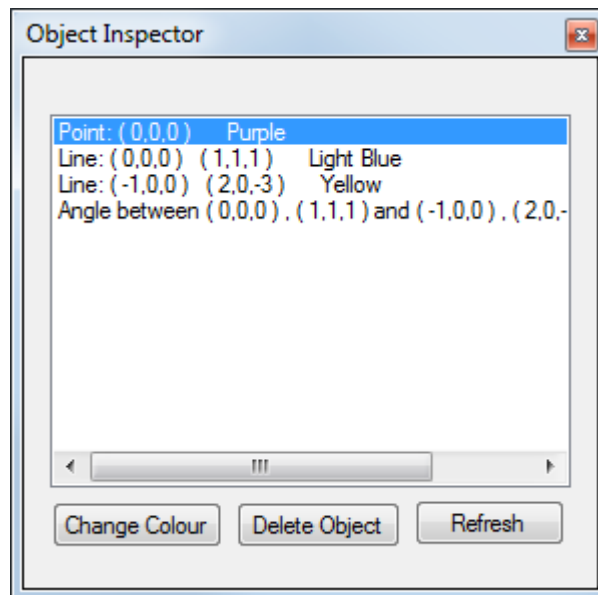
### Test 19

[sec. 3.1.i.e]

Description: Saving a scene and inspecting the csv file produced

Expected result: A csv containing correct information about every object is produced

Pass/Fail: Pass



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	point	0	0	0	0.5	0	1										
2																	
3	line	0	0	0	1	1	1	0	0.5	1							
4	line	-1	0	0	2	0	-3	1	1	0							
5																	
6	angle	0	0	0	1	1	1	-1	0	0	2	0	-3	1	0.5	0	
7																	
8																	

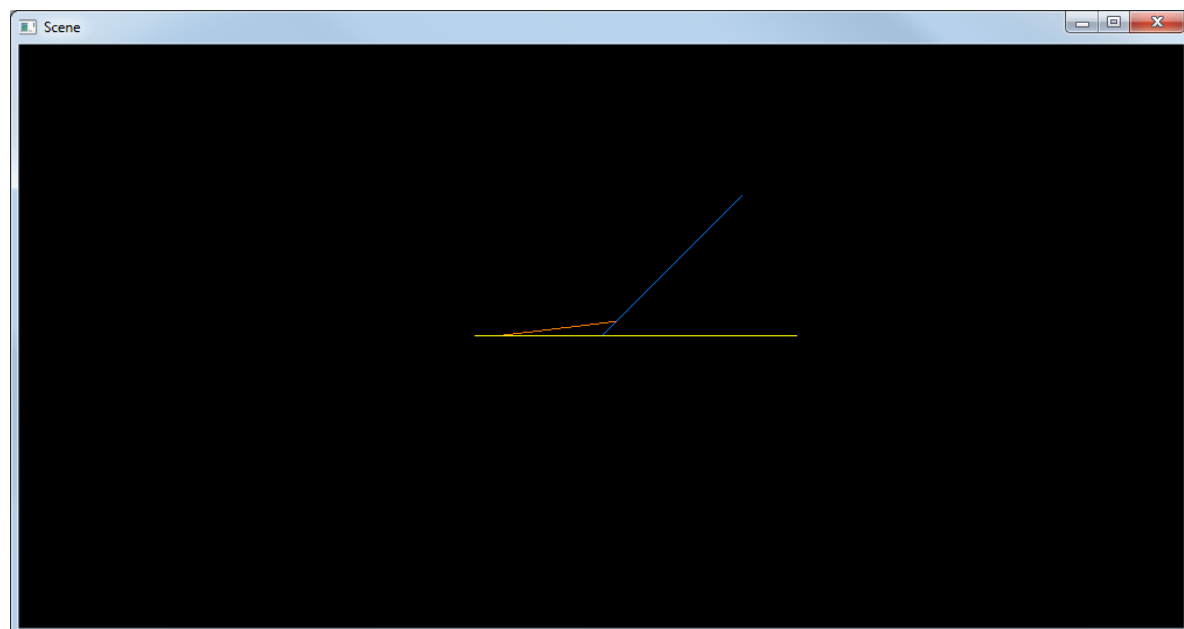
### Test 20

[sec. 3.1.i.e]

Description: Opening a scene from a previously saved scene

Expected result: The scene is opened with all data accurately imported

Pass/Fail: Pass



### Test 21

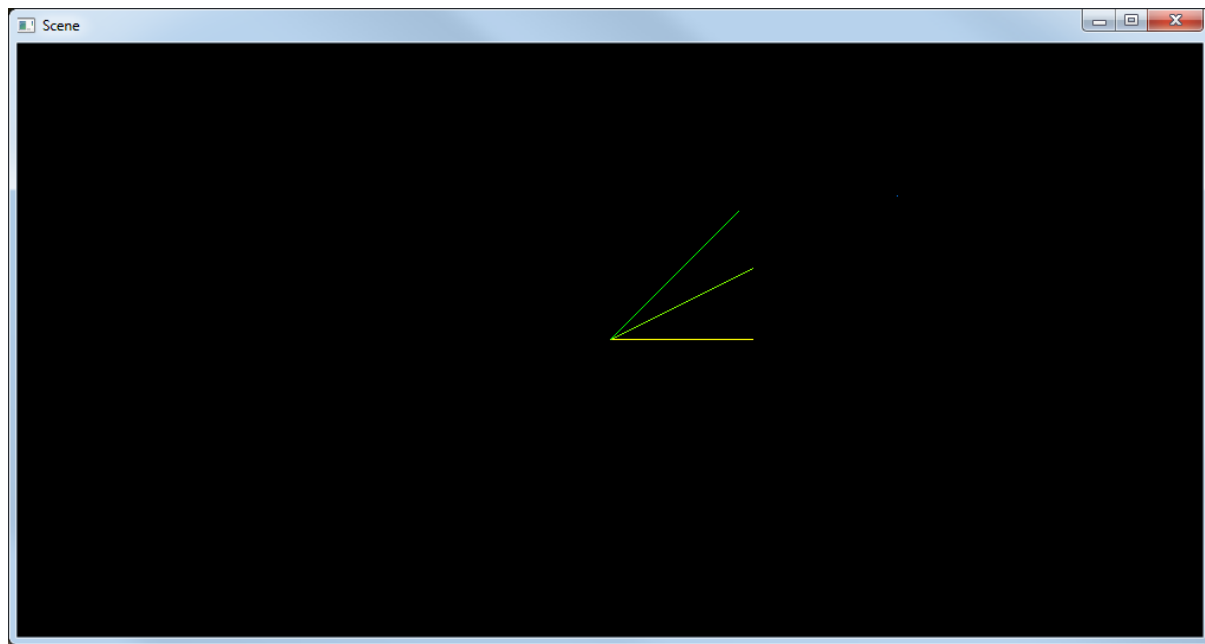
[sec. 3.1.i.f]

Description: Pressing arrow keys while the scene is selected

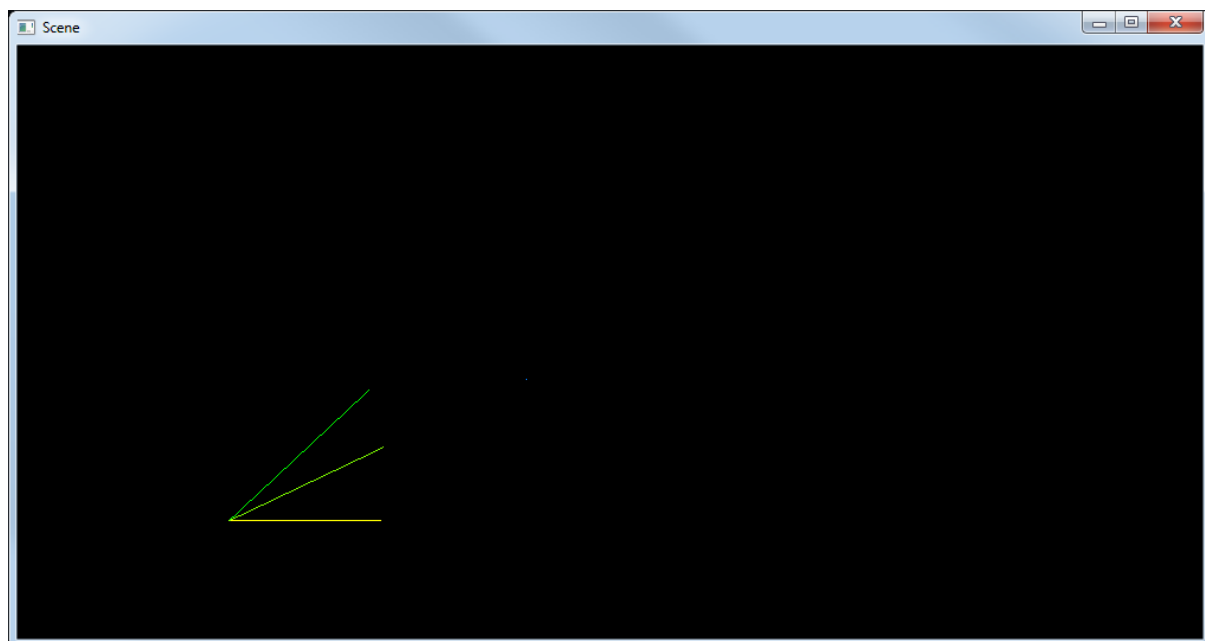
Expected result: Camera translates, moving objects in the scene with respect to the camera

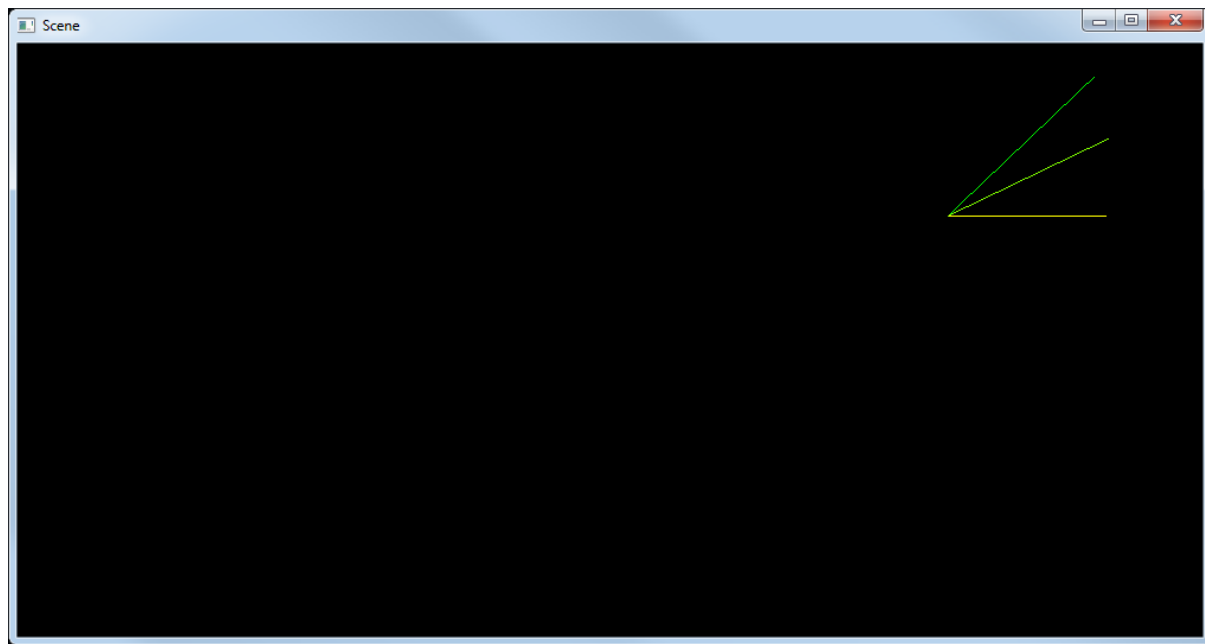
Pass/Fail: Pass

The original scene:



The scene after the arrow keys are pressed:





Translation in all directions works fully.

### Test 22

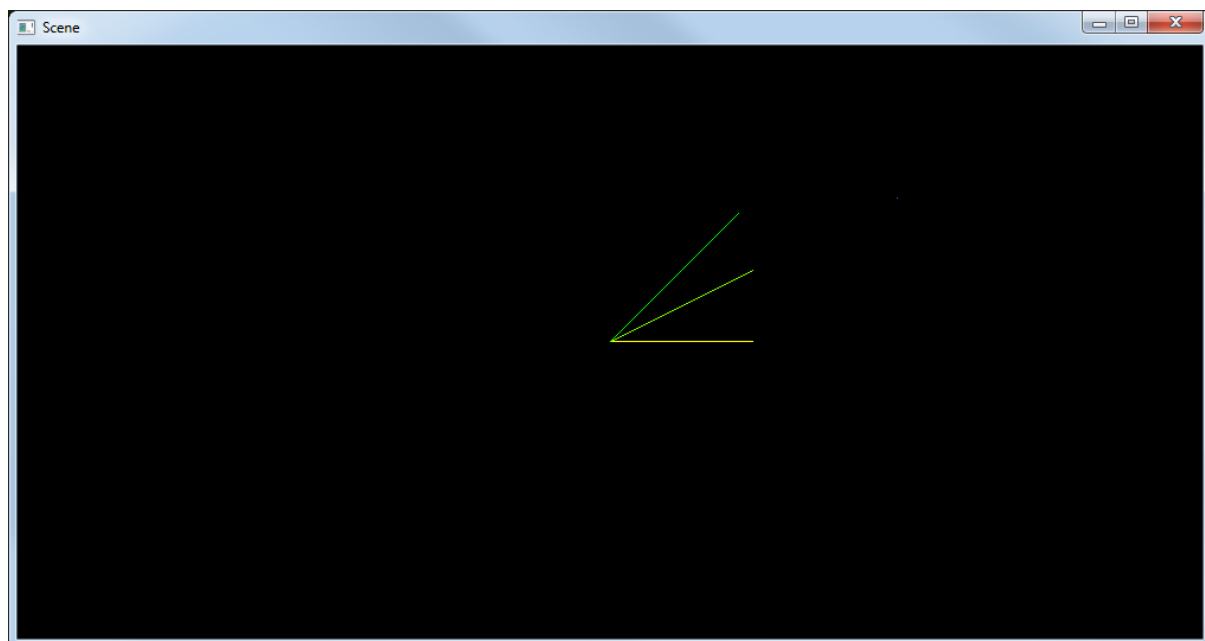
[sec. 3.1.i.f]

Description: Pressing 4 and 6 on the num pad

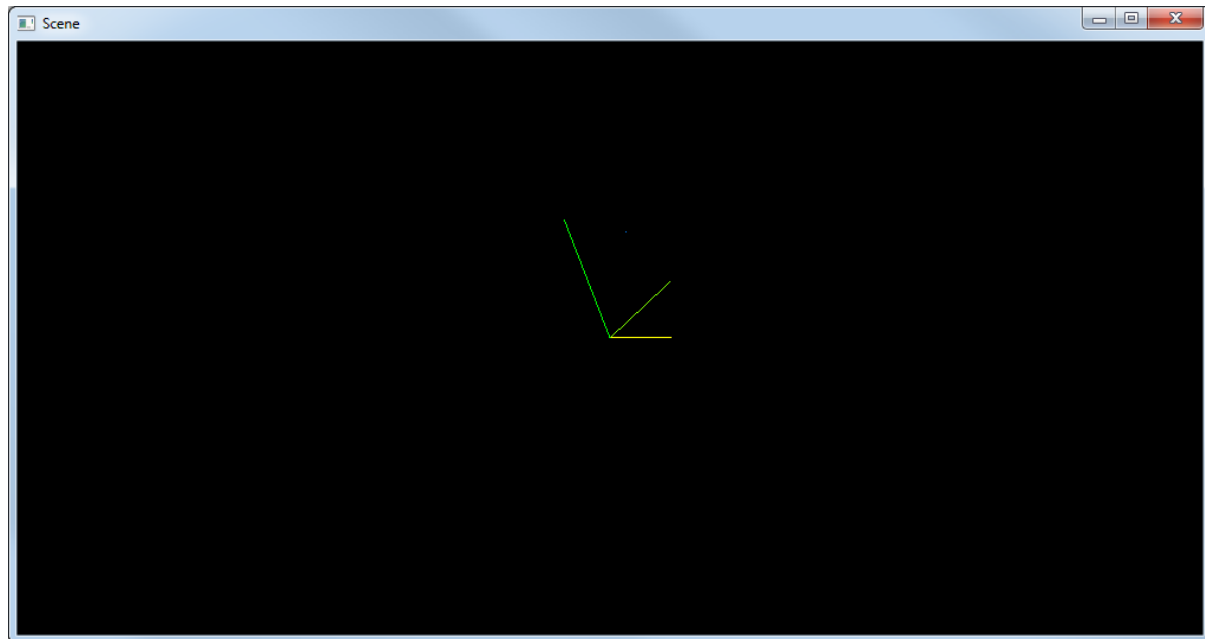
Expected result: Camera rotates around the y axis

Pass/Fail: Pass

The scene before any rotation:



The scene after the camera has been rotated around the y axis (by pressing 4 and/or 6):



### Test 23

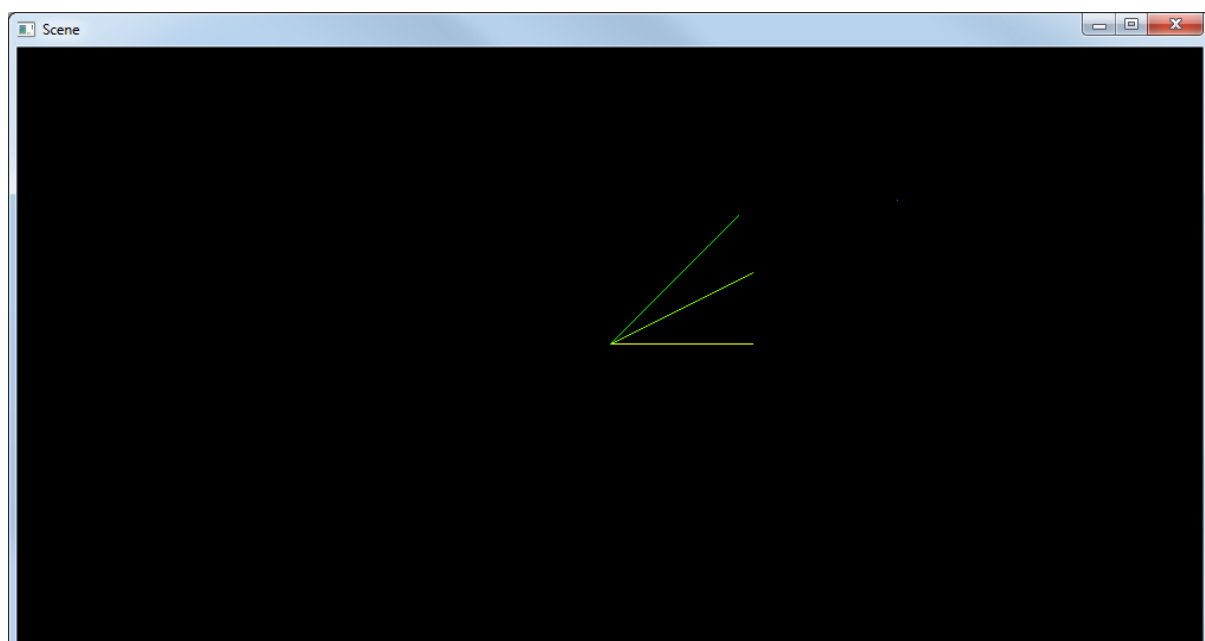
[sec. 3.1.i.f]

Description: Pressing 2 and 8 on the num pad

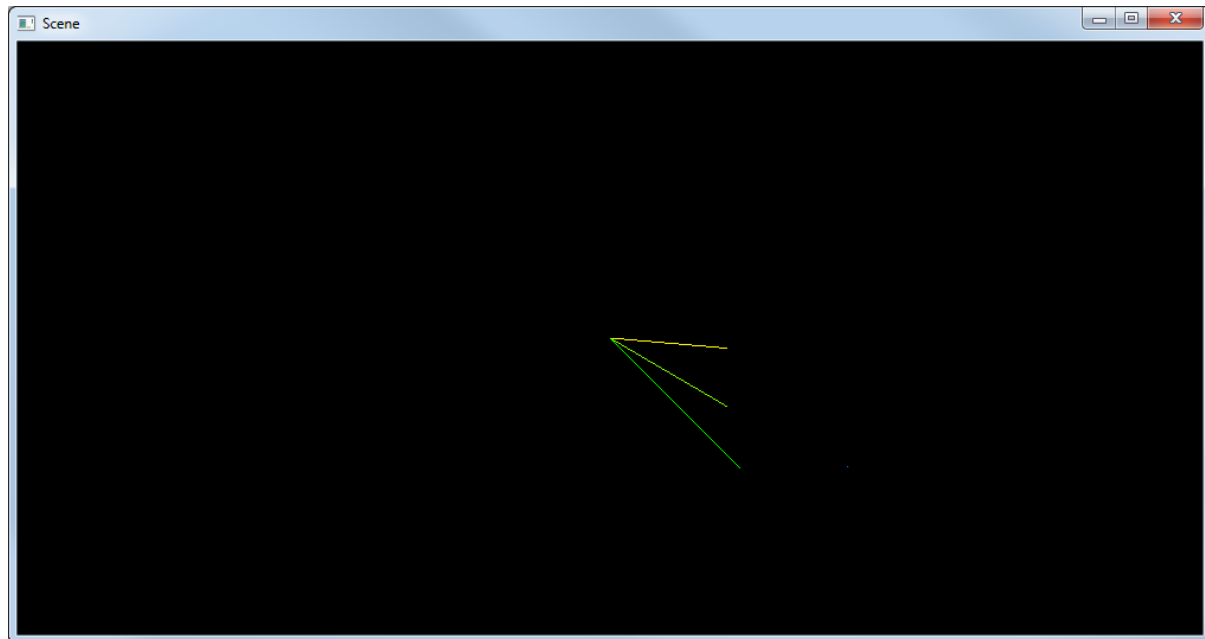
Expected result: The camera rotates around the x axis

Pass/Fail: Pass

The scene before any rotation:



The scene after the camera has been rotated around the x axis (by pressing 2 and/or 8):



### Test 24

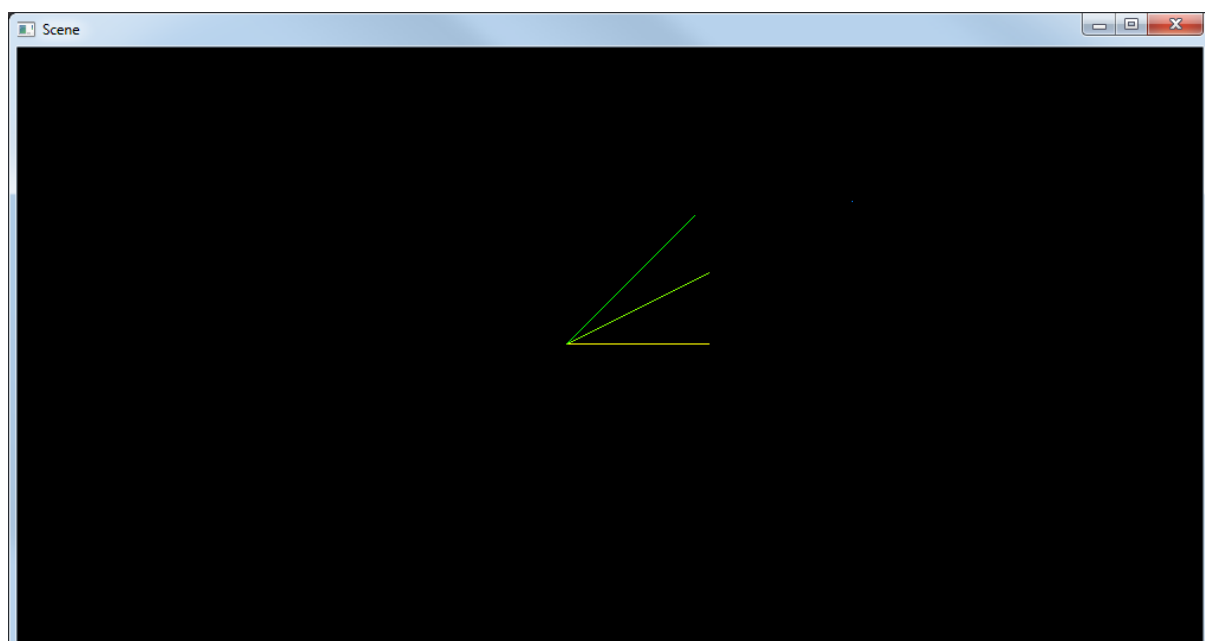
[sec. 3.1.i.f]

Description: Pressing 7 and 9 on the num pad

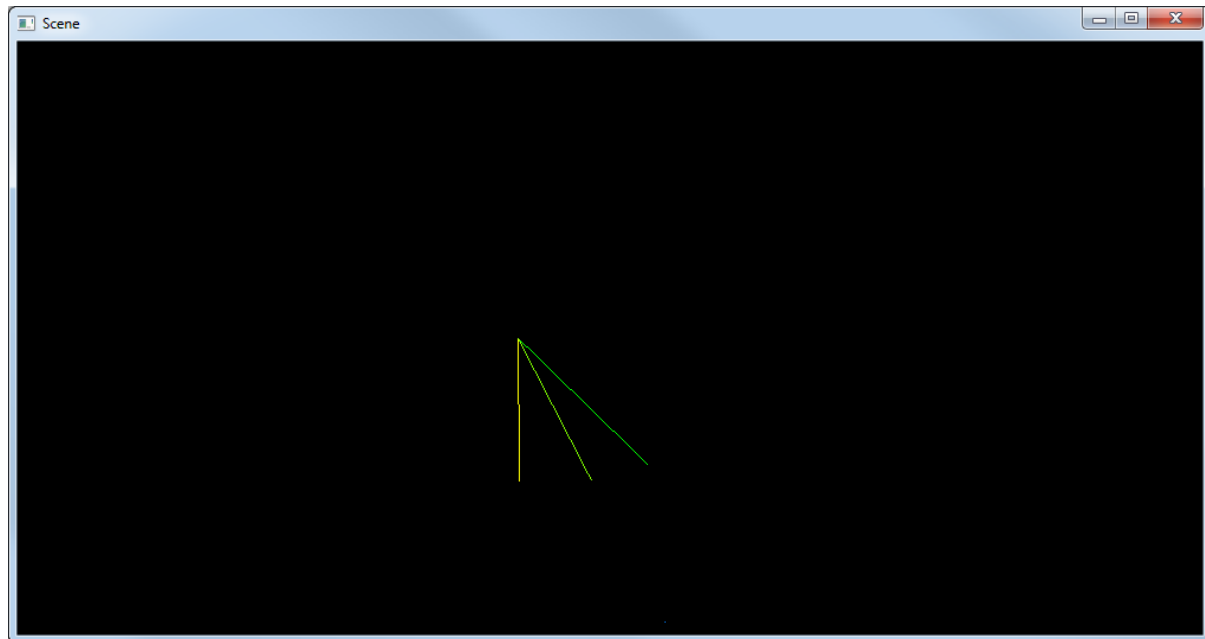
Expected result: The camera rotates around the z axis

Pass/Fail: Pass

The scene before any rotation:



The scene after the camera has been rotated around the z axis (by pressing 7 and/or 9):



### Test 25

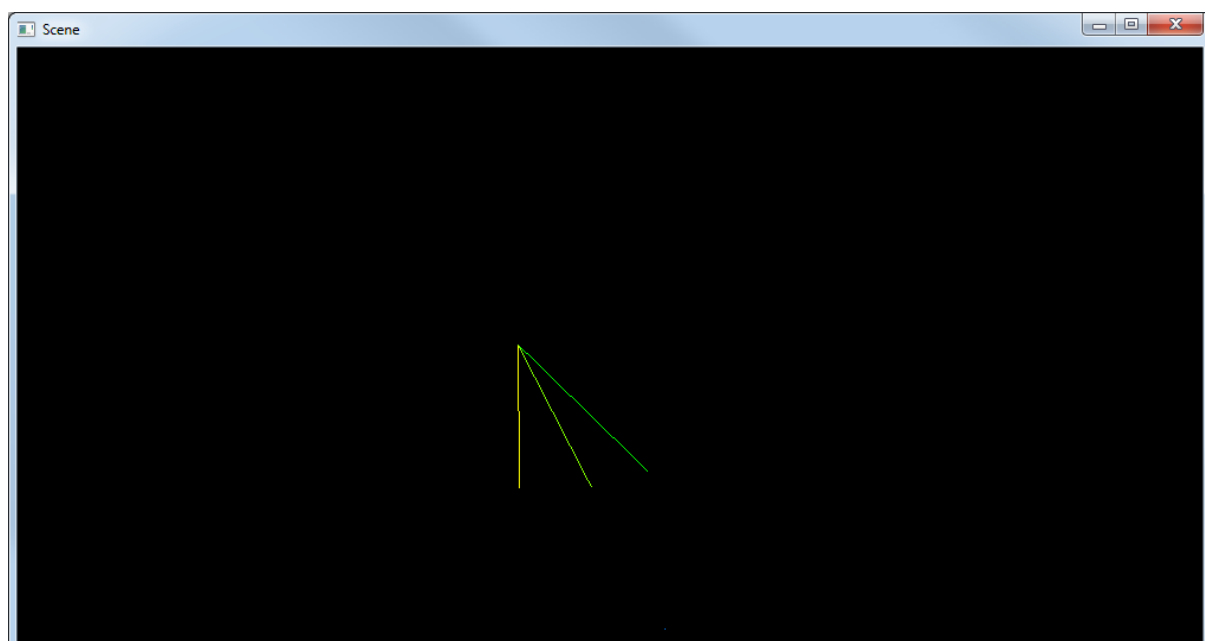
[sec. 3.1.i.f]

Description: Pressing the 'Home' button on the keyboard

Expected result: The camera returns to the original position

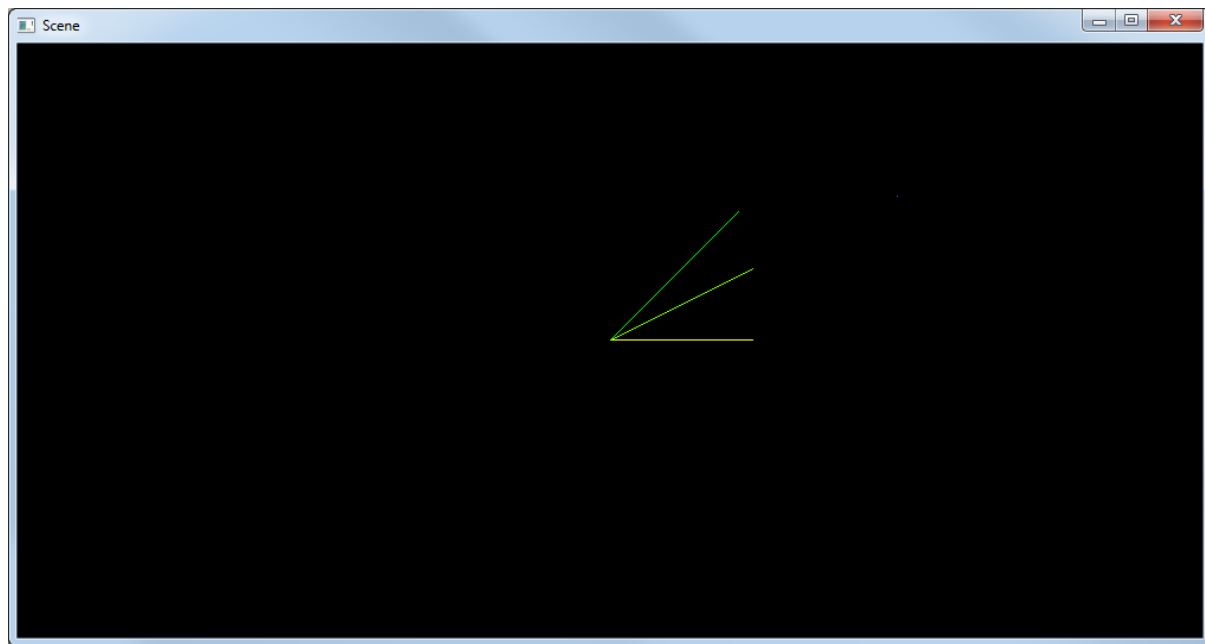
Pass/Fail: Pass

The scene with the camera altered:





The scene after the home button has been pressed:



### 3.3 White box testing

White box testing involves knowledge of the code behind the software, and includes inputting invalid data, and using the program in a work flow that it was not originally intended for.

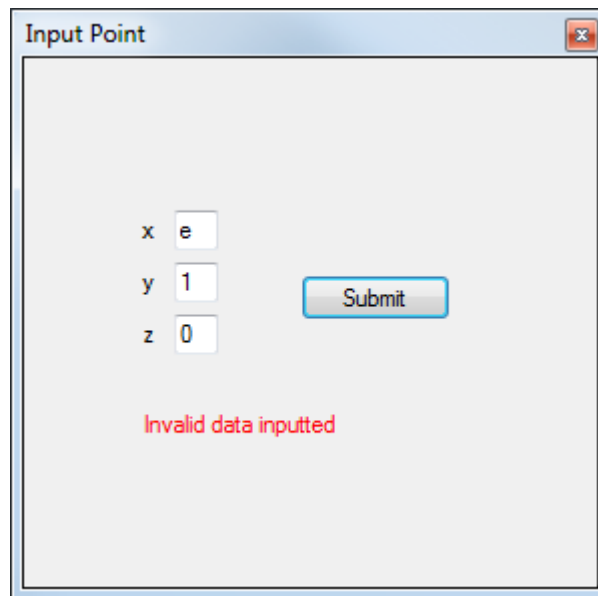
#### Test 1

[sec. 3.1.ii.a.i]

Description: Inputting erroneous (non-number) data into the Input Point form

Expected result: The form does not submit the data and an error message shows

Pass/Fail: Pass



Input Point

x e

y 1

z 0

Submit

Invalid data inputted

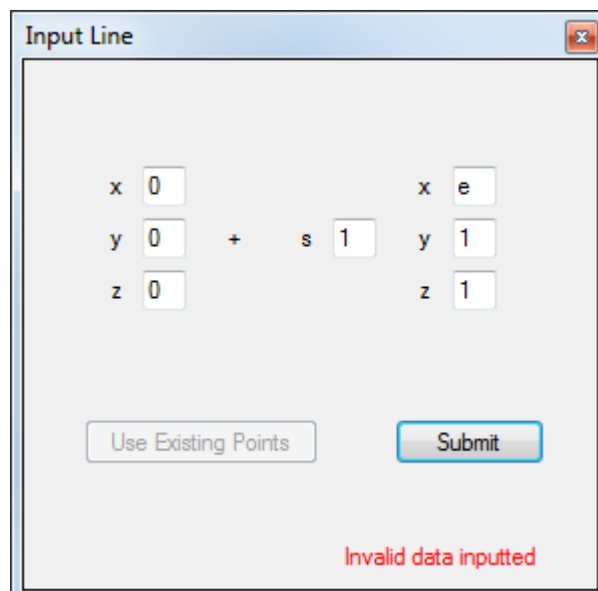
### Test 2

[sec. 3.1.ii.a.i]

Description: Inputting erroneous (non-number) data into the Input Line form

Expected result: The form does not submit the data and an error message shows

Pass/Fail: Pass



Input Line

x 0      x e

y 0      +      s 1      y 1

z 0                     z 1

Use Existing Points      Submit

Invalid data inputted

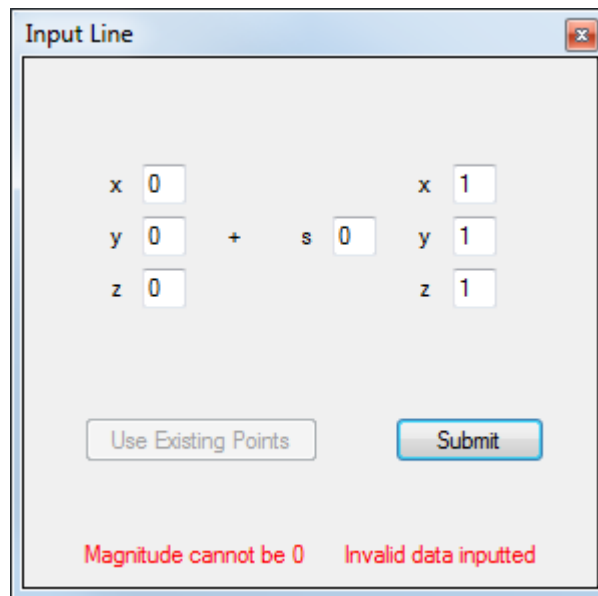
### Test 3

[sec. 3.1.ii.a.i]

Description: Inputting erroneous (magnitude of 0) data into the Input Line form

Expected result: The form does not submit the data and an error message shows

Pass/Fail: Pass



The screenshot shows a window titled "Input Line" with a close button in the top right corner. Inside the window, there are two sets of input fields for vectors. The left set has fields for x, y, and z, all containing the value 0. The right set has fields for x, y, and z, all containing the value 1. Between these two sets is a plus sign and a field for 's' containing the value 0. Below the input fields are two buttons: "Use Existing Points" and "Submit". At the bottom of the window, there is a red error message that reads "Magnitude cannot be 0 Invalid data inputted".

#### Test 4

[sec. 3.1.ii.a.i]

Description: Inputting erroneous (direction of 0, 0, 0) data into the Input Line form

Expected result: The form does not submit the data and an error message shows

Pass/Fail: Pass

Input Line

x 0 y 0 z 0 + s 1 x 0 y 0 z 0

Use Existing Points Submit

Direction vector cannot be 0, 0, 0

Invalid data inputted

### Test 5

[sec. 3.1.ii.a.i]

Description: Inputting erroneous (same point) data into the Input Line From Points form

Expected result: The form does not submit the data and an error message shows

Pass/Fail: Pass

Input Line From Points

Select the 2 points:

First Point Second Point

0,0,0 0,0,0

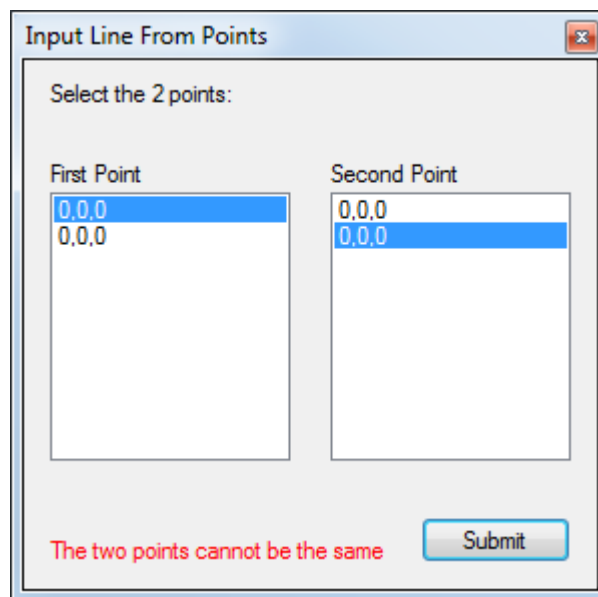
1,1,1 1,1,1

The two points cannot be the same Submit

Description: Inputting erroneous (different point object instance, same coordinate) data into the Input Line From Points form

Expected result: The form does not submit the data and an error message shows

Pass/Fail: Pass



Input Line From Points

Select the 2 points:

First Point

0,0,0  
0,0,0

Second Point

0,0,0  
0,0,0

The two points cannot be the same

Submit

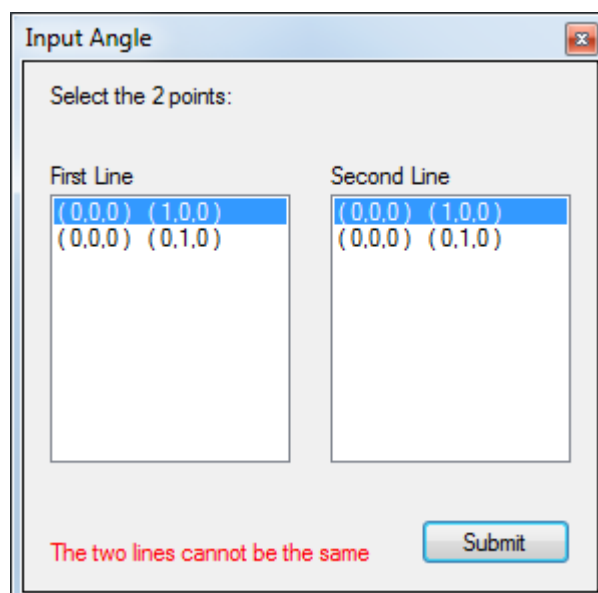
### Test 6

[sec. 3.1.ii.a.i]

Description: Inputting erroneous (same line) data into the Input Angle form

Expected result: The form does not submit the data and an error message shows

Pass/Fail: Pass



Input Angle

Select the 2 points:

First Line

(0,0,0) (1,0,0)  
(0,0,0) (0,1,0)

Second Line

(0,0,0) (1,0,0)  
(0,0,0) (0,1,0)

The two lines cannot be the same

Submit

### Test 7

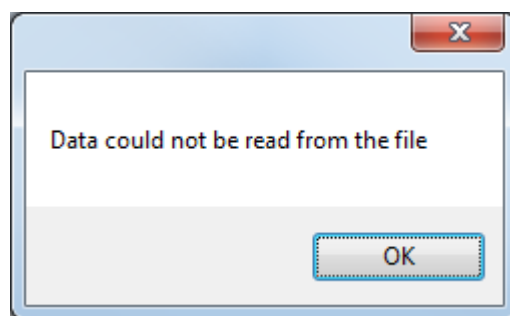
[sec. 3.1.ii.a.ii]

Description: Inputting erroneous (non-number) data into a save file, and then attempting to open it

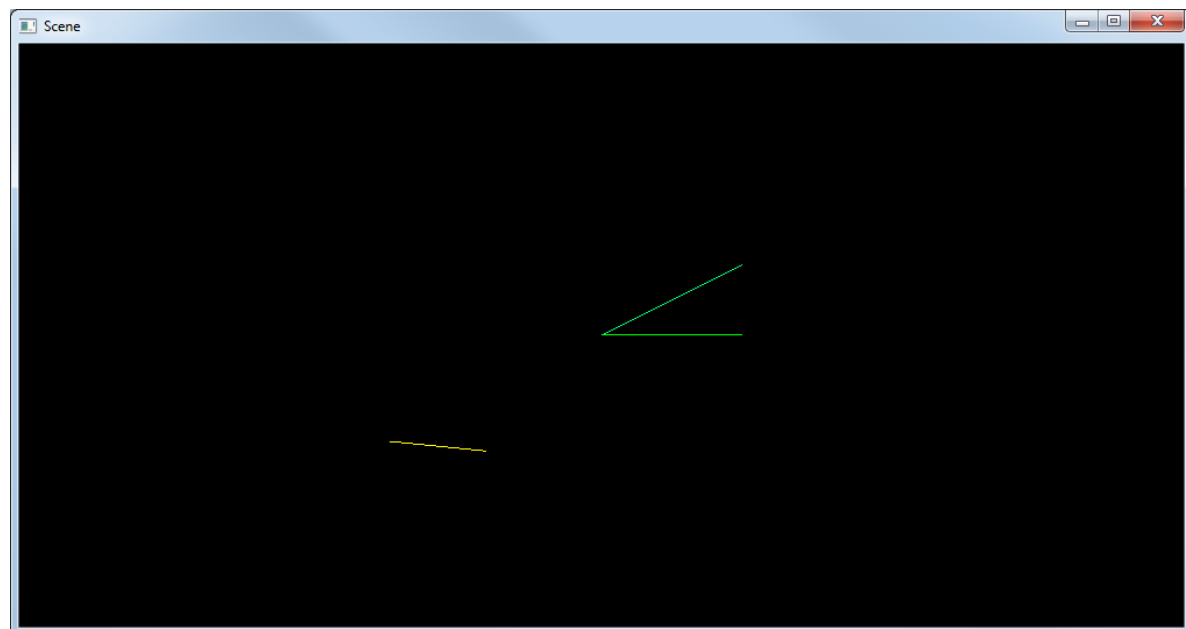
Expected result: An error box shows, reporting the error. The rest of the objects in the file are successfully read in.

Pass/Fail: Pass

	A	B	C	D	E	F	G	H	I	J	K
1											
2	line	0	0	0	1	e	0	0.5	1	0	
3	line	0	0	0	1	0	1	0	1	0	
4	line	0	0	0	1	0.5	1	0	1	0.5	
5	line	-1	-1	-1	-2	-1	-2	1	1	0	
6											



An error box shows as the scene is created.



The other objects in the file are still successfully read in.

### Test 8

[sec. 3.1.ii.c]

Description: Opening Input Line From Points panel, deleting one of the points in the Object Inspector, and then attempting to create the line between one existing and one non-existing point.

Expected result: An error message will show, and the form will not attempt to submit the line

Pass/Fail: Pass

Input Line From Points

Select the 2 points:

First Point	Second Point
0,0,0	0,0,0
1,1,1	1,1,1

At least one of the points does not exist

Submit

Object Inspector

Point: (0,0,0) Orange

Change Colour Delete Object Refresh

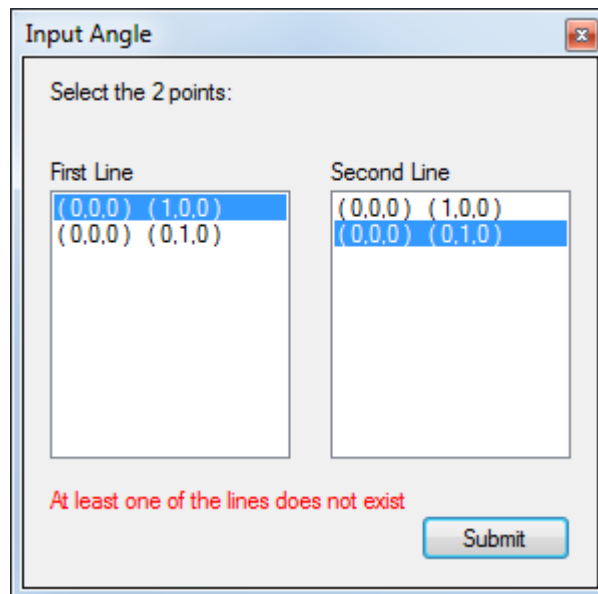
## Test 9

[sec. 3.1.ii.c]

Description: Opening Input Angle panel, deleting one of the lines in the scene through the Object Inspector, and then attempting to create an angle using the non-existing line.

Expected result: An error message will show, and the form will not attempt to submit the angle

Pass/Fail: Pass



Input Angle

Select the 2 points:

First Line

(0,0,0) (1,0,0)  
(0,0,0) (0,1,0)

Second Line

(0,0,0) (1,0,0)  
(0,0,0) (0,1,0)

At least one of the lines does not exist

Submit

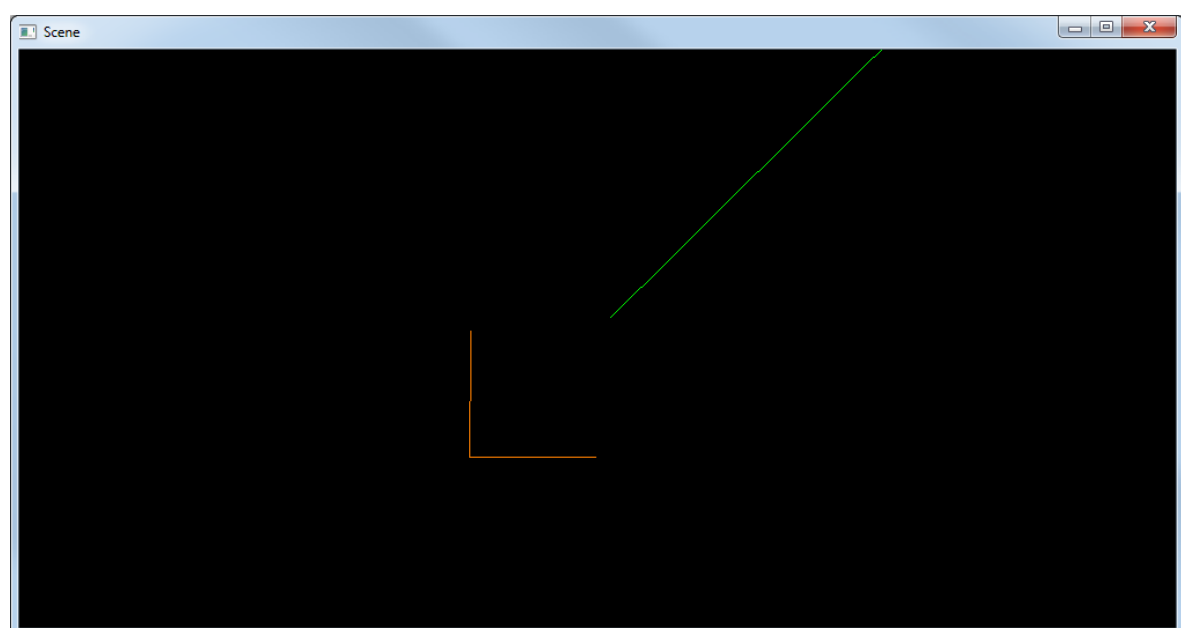
### Test 10

[sec. 3.1.ii.b.i]

Description: Rendering objects outside the initial view frame (extreme data)

Expected result: Objects still render as usual

Pass/Fail: Pass





The orange lines are the axis, and join at the origin. The camera has been translate to the right and up, here.

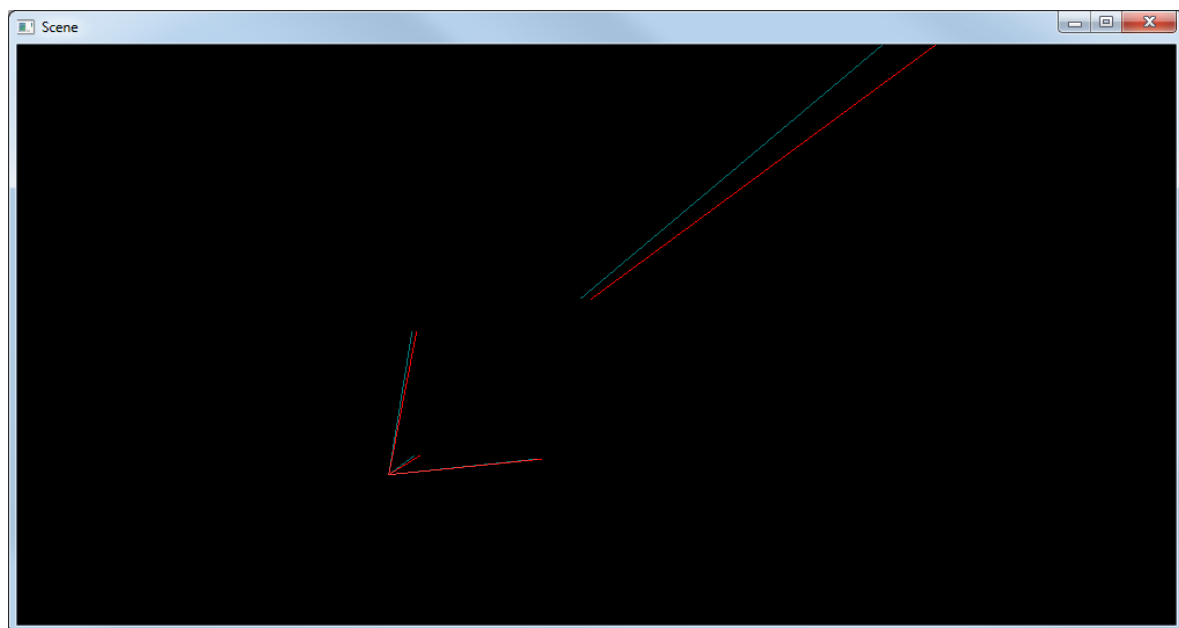
### Test 11

[sec. 3.1.ii.b.ii]

Description: Rendering anaglyphs for objects outside the initial view frame (extreme data)

Expected result: Anaglyph still renders as usual

Pass/Fail: Pass



However, rendering the anaglyph on some computers resulted in all objects in the scene not being rendered after about 20 seconds. The issue was only found on some computers, all of which it would always occur on. The cause of the issue is unknown, and is an area for further testing.

### Test 12

[sec. 3.1.ii.b.iii]

Description: Inputting points with large positive values (far outside the initial view frame)

Data Inputted:

x = 999999999999

y = 999999999999

z = 999999999999

Expected result: The program does not crash (testing whether the point is rendered is impractical as it would involve moving the camera a large distance)

Pass/Fail: Pass

### Test 13

[sec. 3.1.ii.b.iii]

Description: Inputting points with large negative values (far outside the initial view frame)

Data Inputted:

x = -999999999999

y = -999999999999

z = -999999999999

Expected result: The program does not crash (testing whether the point is rendered is impractical as it would involve moving the camera a large distance)

Pass/Fail: Pass

### Test 14

[sec. 3.1.ii.b.iii]

Description: Inputting lines with large values (far outside the initial view frame)

Expected result: The program does not crash (testing whether the point is rendered is impractical as it would involve moving the camera a large distance)

Pass/Fail: Pass

### 3.4 Stress Testing

One of the objectives of the software was to be able to render 20 objects at once. The computers on which the stress tests were carried on were the same model as that which the user would use [sec. 1.4.2.9].

#### Test 1

[sec. 3.1.iii.a]

Description: Rendering 20 objects (a mix of points, lines and angles) and observing lag

Expected result: No lag

Pass/Fail: Pass

#### Test 2

[sec. 3.1.iii.b]

Description: Rendering 20 objects, selecting the anaglyph to be rendered and observing lag

Expected result: No lag

Pass/Fail: Pass

## 4. System Maintenance

---

## 4.1 System Overview

### 4.1.1 Forms and Data Flow

#### Input Panel

[sec. 2.8.1]

The Input Panel is the main form of the program. The application is closed only when the Input Panel is closed. The Input Panel provides access to file and scene handling buttons, anaglyph control features, alongside access to input forms (below).

The scene is created in the Input Panel. As a result, the scene only has a reference to the Input Panel, and so must fetch all data from there.

All data inputted from forms is passed to the Input Panel, where it is then fetched by the scene.

#### Input Point Panel

[sec. 2.8.2]

The Input Point Panel takes in the coordinates of one point, and passes the data up to the Input Panel, where it is then fetched and dealt with by the scene.

#### Input Line Panel

[sec. 2.8.3]

The Input Line Panel takes in the equation of one line, and passes the data up to the Input Panel, where it is then fetched and dealt with by the scene.

A button is also available, in the Input Line Panel, to open the Input Line From Points Panel. Pressing the button will close the Input Line Panel, and open the Input Line From Points Panel.

#### Input Line From Points Panel

[sec. 2.8.4]

The Input Line From Points Panel displays the same list of points (obtained through the Input Panel) twice. The user selects a point from each and the form submits the selected points to the Input Panel. The line is then fetched by the scene, from the Input Panel.

#### Input Angle Panel

[sec. 2.8.5]

The Input Angle Panel displays the same list of lines (obtained through the Input Panel) twice. The user selects a line from each and the form submits the selected lines to the Input Panel. The angle is then fetched by the scene, from the Input Panel.

### Object Inspector

[sec. 2.8.6]

The Object Inspector displays a list of all objects in the scene, including their type, their coordinates, and their colour. The user can select an object and chose to change its colour (opening the Colour Change Panel), or delete it. Deleting an object will pass the object to the Input Panel where it will be fetched by the scene, and then deleted by the scene.

### Colour Change Panel

[sec. 2.8.7]

The Colour Change Panel displays a list of colours available for objects, alongside details about the object selected to have its colour changed. When a colour is selected and the submit button is pressed, the form passes the data about the object and the colour to the Input Panel, where it is then fetched by the scene.

## 4.1.2 Form Relations

For form relations, view sec. 2.3.1.

## 4.1.3 Classes

### ColoredObject

[sec. 2.8.8]

The ColoredObject class is the parent class for Point, Line and Angle. It contains the possible colours, and names of these colours, that a coloured object can take.

### Point

[sec. 2.8.9]

The Point class contains information about a point to be rendered in the scene. This information includes colour (inherited from ColoredObject) and coordinates.

### Line

[sec. 2.8.10]

The Line class contains information about a line to be rendered in the scene. This information includes the colour (inherited from ColoredObject) and the coordinates of points at either end of the line (defining the line).

### Angle

[sec. 2.8.11]

The Angle class contains information about an angle to be rendered in the scene. This information includes the colour (inherited from ColoredObject), the points defining the lines between which the angle is rendered, and the points defining the line drawn to represent the angle.

### Scene

[sec. 2.8.12]

The Scene class is the class which, when created, creates the render environment window using GLUT, and then renders objects using OpenGL.

The main loop is run in this class. Within the loop, objects are rendered, objects are fetched from the Input Panel and updated, current objects are passed to the Input Panel, key presses are detected and the camera is updated as necessary.

## 4.1.4 Class Inheritance

For class inheritance, view sec. 2.5.

## 4.1.5 Libraries

[sec. 2.4]

OpenGL, freeglut and Tao's freeglut are included in the release. The OpenGL version included is not the same version as that which is installed on the computer by default. The packaging of the OpenGL library with the final program ensures that the program will run on any computer whether it has OpenGL installed by default or not.

## 4.2 Algorithm Design

For algorithm descriptions and pseudo code, view sec. 2.8.

# 5. User Manual

---



User Manual Contents

5. User Manual .....	<b>Error! Bookmark not defined.</b>
5.1 Introduction .....	<b>Error! Bookmark not defined.</b>
5.2 Installation .....	<b>Error! Bookmark not defined.</b>
5.3 Using the software .....	<b>Error! Bookmark not defined.</b>
5.4 Troubleshooting .....	<b>Error! Bookmark not defined.</b>

## 5.1 Introduction

The Vector Visualisation Software is a teaching application for Windows which aids visualisation of 3D. The program also offers use of a 3D anaglyph, which is used in conjunction with red-cyan glasses.

## 5.2 Installation

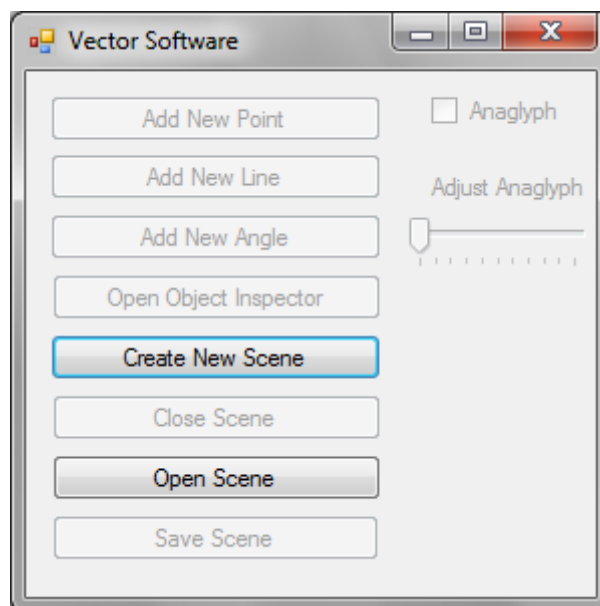
The application, VectorSoftware.exe, comes in a .zip file. Extract this file by right clicking and selecting 'extract'. To run the program, select VectorSoftware.exe.

The software requires Windows to run, with a .NET version of 4.5 or higher. The newest version of .NET can be found at <http://microsoft.com/net>.

For anaglyph use, red-cyan glasses can be found online.

## 5.3 Using the software

On opening the software the following window should appear:

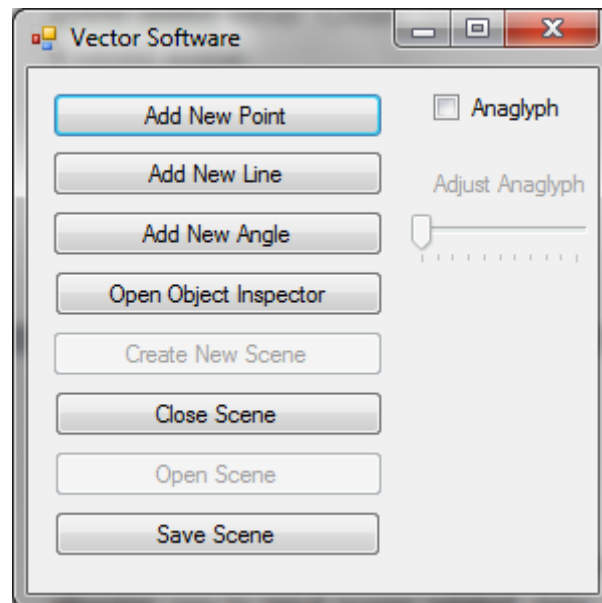


This is the main window from which you can control the program. Closing this window will close the application.

### 5.3.1 Creating a scene

The scene is the window which renders the objects in 3D. To create a new scene select either 'Create New Scene' or 'Open Scene', if you can have a saved scene.

On creating a scene, the main window will change to the following, allowing you to input scene related data:



### 5.3.2 Closing a scene

To close a scene, select the 'Close Scene' button, or press the red cross in the top right of the scene window.

Closing the main application will close the scene.

### 5.3.3 Saving a scene

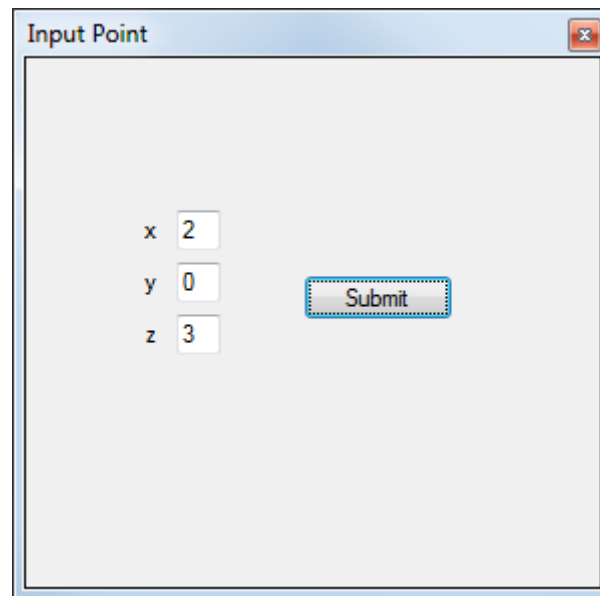
To save a scene, select the 'Save Scene' button, input a name for the save file, and select 'Save'.

### 5.3.4 Inputting objects

Objects can only be inputted when a scene exists.

#### 1. Inputting a point

To input a point, select 'Add New Point'. A new window will open in which the coordinates of the point, x, y, z, can be inputted. Once the coordinates have been inputted, select 'Submit' to send the point to the scene.



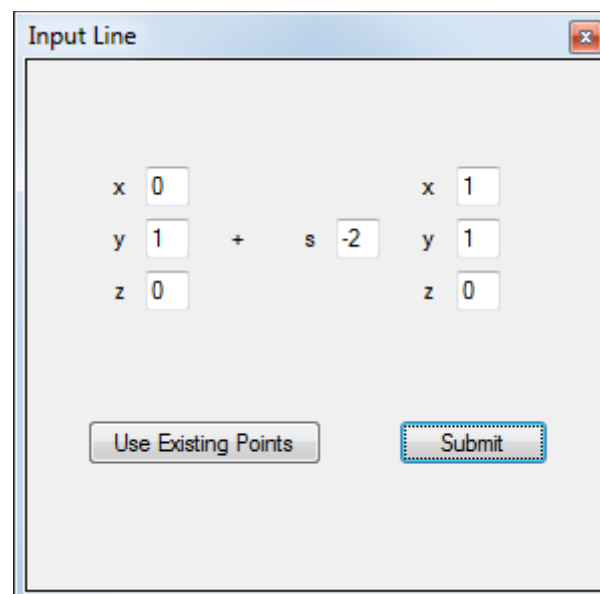
A dialog box titled "Input Point" with a close button (X) in the top right corner. Inside the dialog, there are three input fields for coordinates: "x" with the value "2", "y" with the value "0", and "z" with the value "3". To the right of these fields is a "Submit" button.

## 2. Inputting a line

To input a line select 'Add New Line'. A new window will open.

### 1. By equation

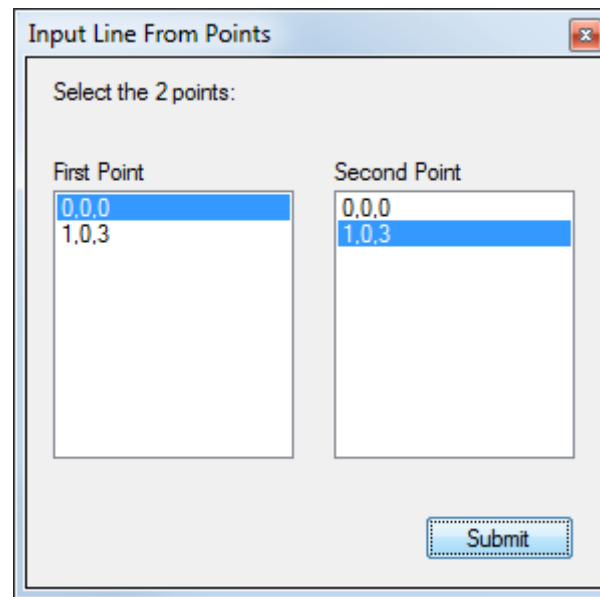
To input the line by equation, input the data into the field marked  $\begin{pmatrix} x \\ y \\ z \end{pmatrix} + s \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ . Click the 'Submit' button, and the line will be sent to the scene.



A dialog box titled "Input Line" with a close button (X) in the top right corner. Inside the dialog, there are two sets of input fields for coordinates. The first set has "x" (0), "y" (1), and "z" (0). To the right of these is a "+" sign, followed by "s" and a field containing "-2". To the right of that is another set of input fields: "x" (1), "y" (1), and "z" (0). At the bottom of the dialog, there are two buttons: "Use Existing Points" and "Submit".

### 2. By existing points

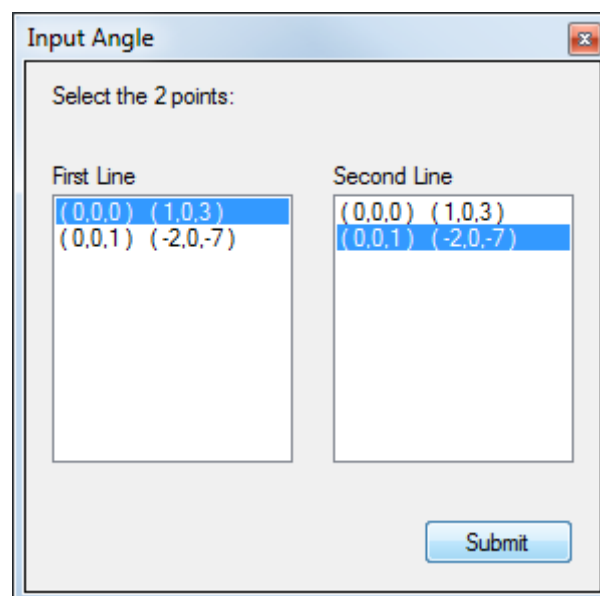
To input a line between two existing points, select 'Use Existing Points'. Then select a point from each list and click the 'Submit' button. The line will then be rendered in the scene.



### 3. Inputting an angle

To input an angle, select 'Add New Angle'. A new window will appear with two lists of the lines in the scene.

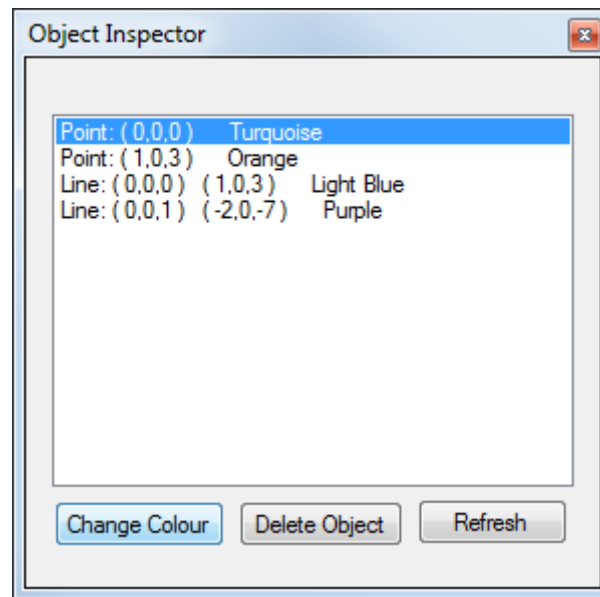
Select a line from each list, and click 'Submit'. The angle between these lines will then be rendered.



#### 5.3.5 Changing objects colours

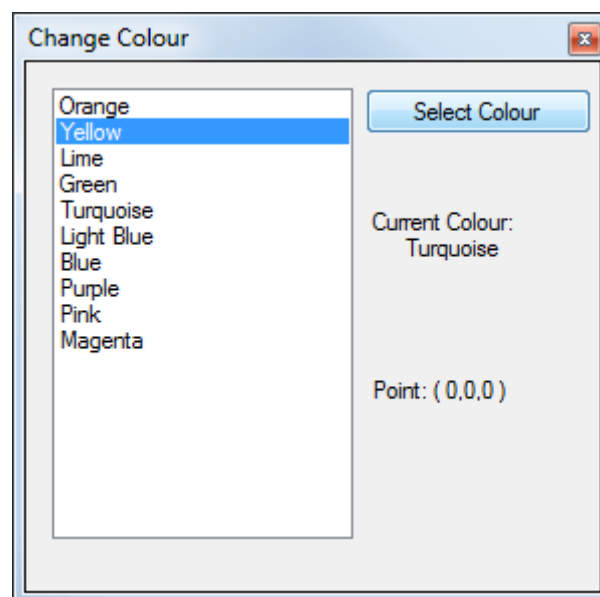
To change the colour of an object, open the Object Inspector by clicking 'Open Object Inspector'.

Then select the object you want to change the colour of from the list of objects. Click 'Change Colour'.



A window with a list of colours will open. Select the desired colour for the object and click 'Select Colour'.

The colour of the object in the scene should now be changed.

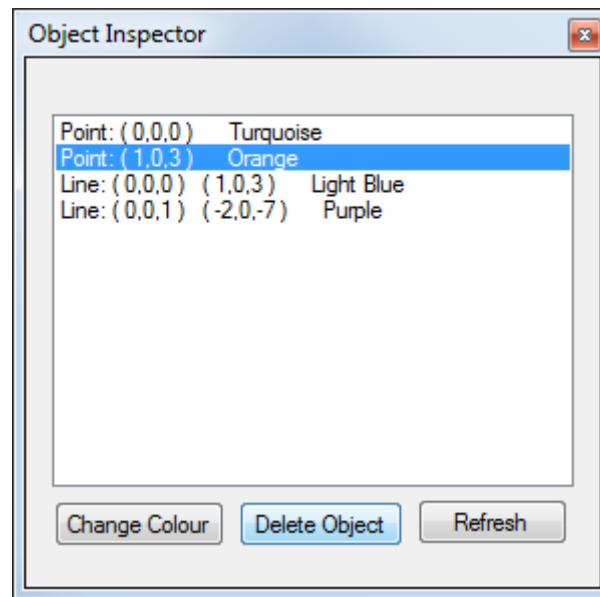


### 5.3.6 Deleting objects

To delete an object, open the Object Inspector by clicking 'Open Object Inspector'.

Then select the object you want to delete from the list of objects. Click 'Delete Object'.

The object should now be removed from the list, and deleted from the scene. If the object is not removed from the list, but is removed from the scene, click 'Refresh'.



### 5.3.7 Controlling the camera

#### 1. Translation

Translation along the x axis:

Left and Right arrows

Translation along the y axis:

Up and Down arrows

#### 2. Zooming

Zooming in:

Page Up

Zooming out:

Page Down

#### 3. Rotation

To rotate, the **num lock** button must be turned on.

Rotation around the y axis:

4 and 6 numpad keys

Rotation around the x axis:

2 and 8 numpad keys

Rotation around the z axis:

7 and 9 numpad keys

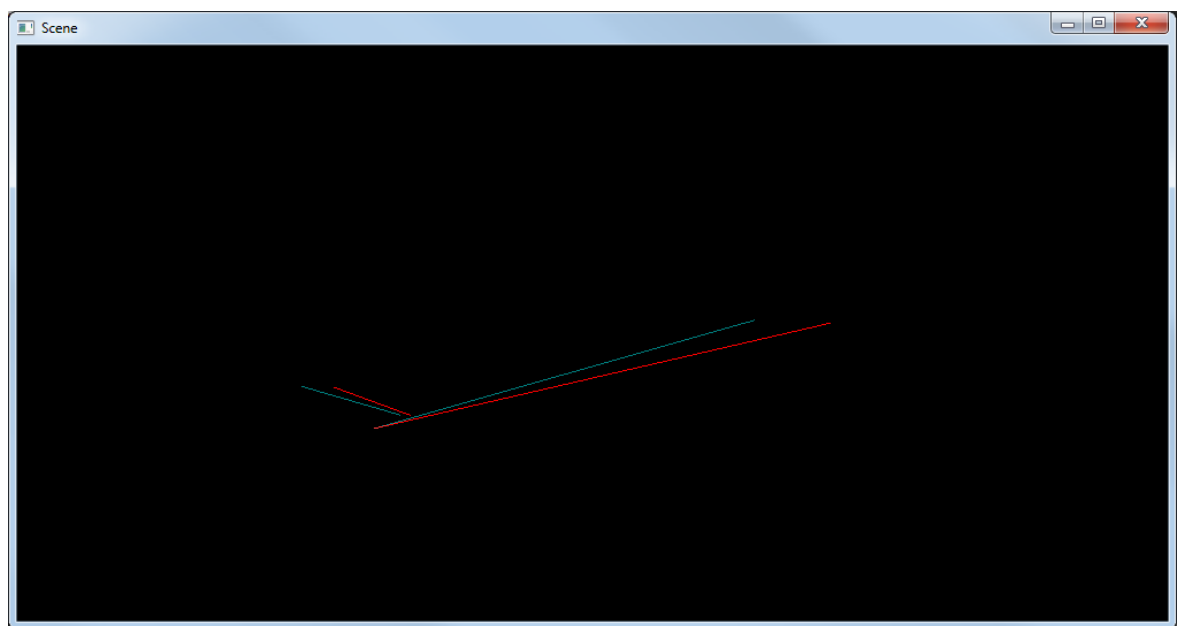
#### 4. Resetting the camera

Home button

#### 5.3.8 Using the anaglyph

##### 1. Rendering the anaglyph

To render the anaglyph, check the Anaglyph check box. The normal coloured objects should no longer be rendered, and only red and cyan objects should be rendered.

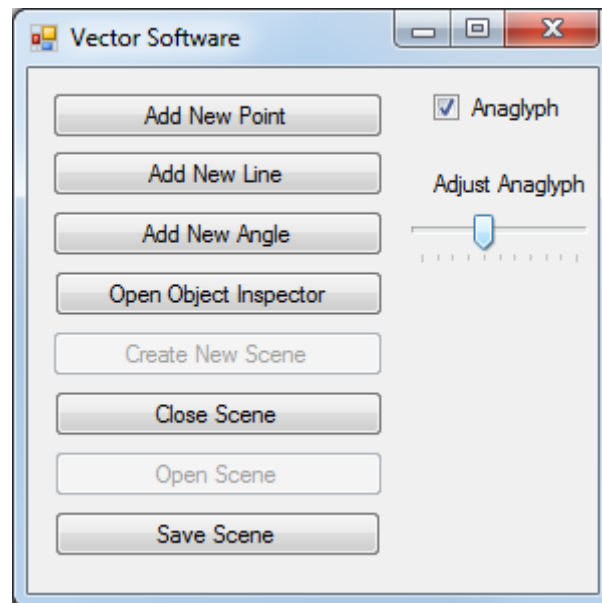


##### 2. Positioning yourself to view the scene

Put on the red-cyan glasses, face the scene front on at first, about an arm's length from the screen.

Adjust the anaglyph using the Adjust Anaglyph slider until the 3D effect is seen.





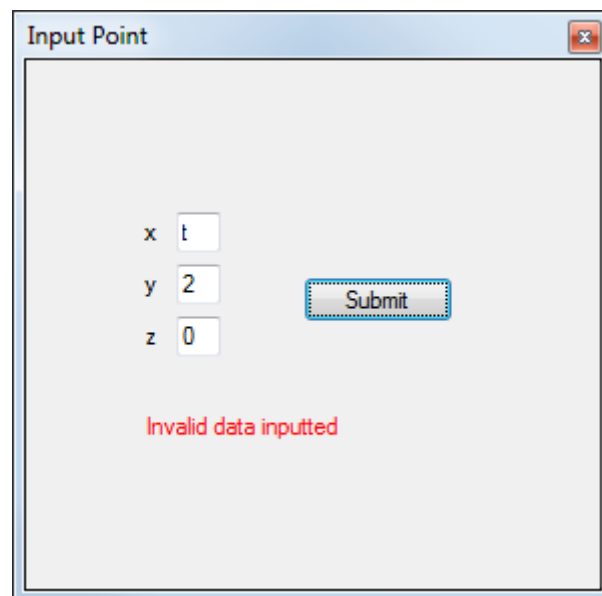
It may help to move your head from side to side while adjusting the anaglyph, as this should give you a better perspective of the 3D effect at that Adjust Anaglyph setting.

## 5.4 Troubleshooting

### 5.4.1 Inputting a point

#### 'Invalid data inputted'

A non-number value was entered (e.g. a letter). Make sure each x, y, and z value have only number entries.

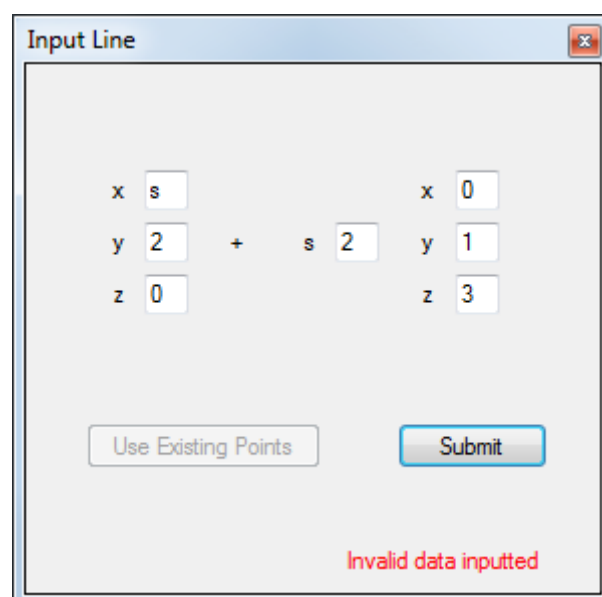


The 'Input Point' dialog box has a title bar with a close button. Inside, there are three input fields labeled 'x', 'y', and 'z'. The 'x' field contains the letter 't', the 'y' field contains '2', and the 'z' field contains '0'. To the right of these fields is a 'Submit' button. Below the input fields, the text 'Invalid data inputted' is displayed in red.

### 5.4.2 Inputting a line by equation

#### 'Invalid data inputted'

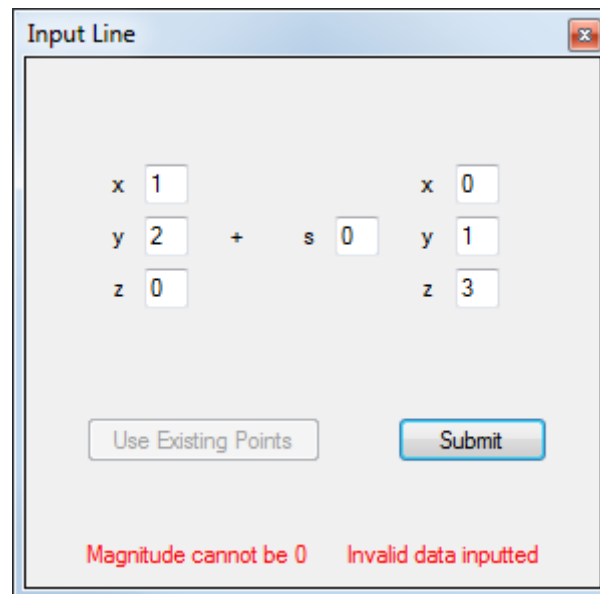
A non-number value was entered (e.g. a letter). Make sure each x, y, z and s value have only number entries.



The 'Input Line' dialog box has a title bar with a close button. Inside, there are two sets of input fields. The first set has fields for 'x', 'y', and 'z' with values 's', '2', and '0' respectively. The second set has fields for 'x', 'y', and 'z' with values '0', '1', and '3' respectively. Between these two sets is a '+' sign and an 's' label with a value of '2'. At the bottom, there are two buttons: 'Use Existing Points' and 'Submit'. Below the buttons, the text 'Invalid data inputted' is displayed in red.

'Magnitude cannot be 0'

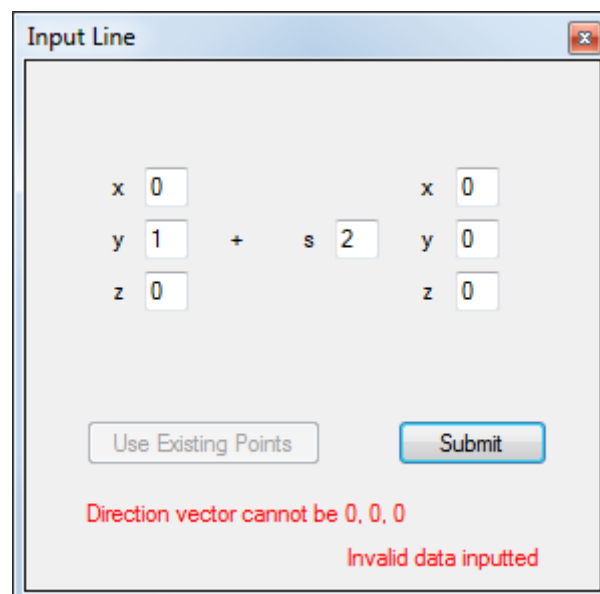
The s entry is 0, meaning the magnitude of the line is 0. Replace it with any number other than 0.



The screenshot shows a dialog box titled "Input Line" with a close button (X) in the top right corner. Inside the dialog, there are two sets of input fields for x, y, and z coordinates, separated by a plus sign and a scalar s. The first set has x=1, y=2, z=0. The second set has x=0, y=1, z=3. The scalar s is set to 0. Below the input fields are two buttons: "Use Existing Points" and "Submit". At the bottom of the dialog, there is a red error message that reads "Magnitude cannot be 0 Invalid data inputted".

'Direction vector cannot be 0, 0, 0'

The second set of x, y, z values (the direction vector) is 0, 0, 0. Replace any of the numbers with any number other than 0.

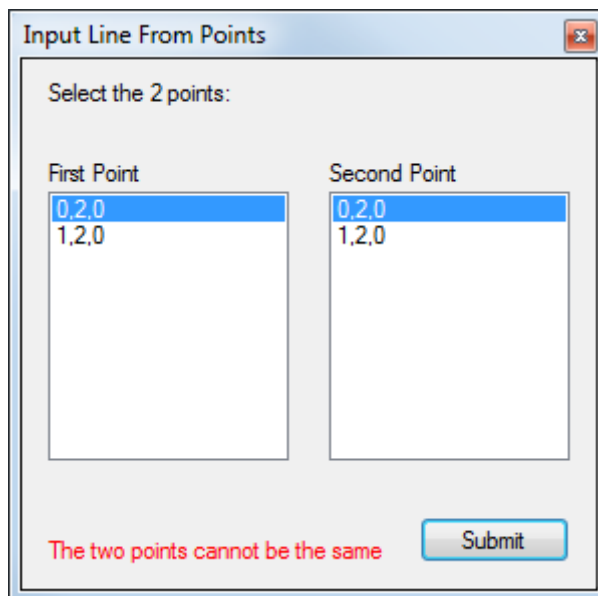


The screenshot shows a dialog box titled "Input Line" with a close button (X) in the top right corner. Inside the dialog, there are two sets of input fields for x, y, and z coordinates, separated by a plus sign and a scalar s. The first set has x=0, y=1, z=0. The second set has x=0, y=0, z=0. The scalar s is set to 2. Below the input fields are two buttons: "Use Existing Points" and "Submit". At the bottom of the dialog, there is a red error message that reads "Direction vector cannot be 0, 0, 0 Invalid data inputted".

### 5.4.3 Inputting a line by existing points

#### 'The two points cannot be the same'

The point selected in each list have the same coordinates, which would produce a line of 0 magnitude. This is therefore an invalid input. Chose two points with different coordinates to render the line between them.



Input Line From Points

Select the 2 points:

First Point

- 0,2,0
- 1,2,0

Second Point

- 0,2,0
- 1,2,0

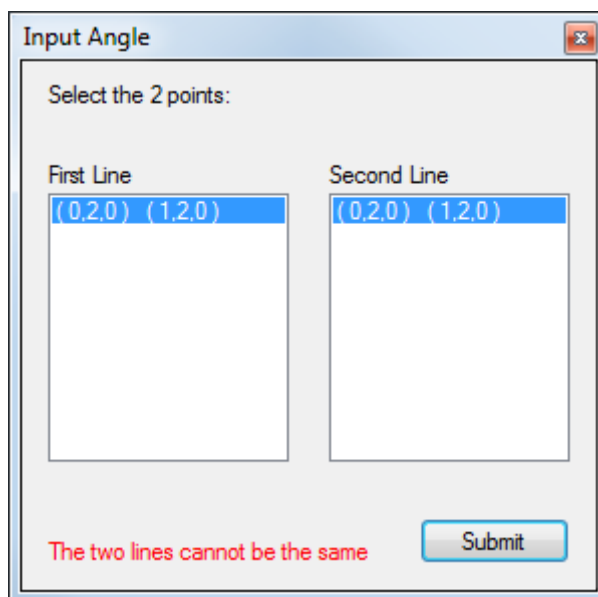
The two points cannot be the same

Submit

### 5.4.4 Inputting an angle

#### 'The two points cannot be the same'

The line selected in each list have the same equations, and so the angle between the lines is 0. Chose two lines with different equations to render the angle between them.



Input Angle

Select the 2 points:

First Line

- (0,2,0) (1,2,0)

Second Line

- (0,2,0) (1,2,0)

The two lines cannot be the same

Submit

#### 5.4.5 Rotating the camera

##### The camera does not rotate when the keys are pressed

Make sure the Num Lock button is turned on.

Make sure the keys you are using are on the numpad and not on the main section of the keyboard.

#### 5.4.6 Rendering objects

##### Inputted objects are not rendering

Your object may be outside of the original view frame. Start by rendering points close to the origin (no more than 2 units away in all dimensions).

If your points still aren't rendering you may need to update your .NET framework.

##### A single dot appears at the origin when an object is inputted

Your .NET framework needs updating. The newest version of .NET can be found online at <http://microsoft.com/net>.

# 6. Appraisal

---

## 6.1 Project success

The success of the project is based upon how many of the features outlined in the Analysis document (and listed below) are present in the final project.

### 6.1.1 The objectives

For objectives of the software, view sec. 1.9.

### 6.1.2 Analysis of project

Objective 1 [sec 1.9.1]:

The user could input the co-ordinates of a point, with full functionality and validation. This objective was met.

Objective 2 [sec 1.9.2]:

The user could input the equation of a line, with full functionality and validation. This objective was met.

Objective 3 [sec 1.9.3]:

The user could select two different existing points and have the line between them drawn, with full functionality and validation. This objective was met.

Objective 4 [sec 1.9.4]:

The user could select two different existing lines and have the angle between them drawn. Full validation existed, though the angle was drawn as a line instead of a circle segment. The form of the angle was not important, so the objective was met, though the user experience involved could have been improved.

Objective 5 [sec 1.9.5]:

User interface forms existed for each input. This objective was met.

Objective 6 [sec 1.9.6]:

The user could select an object, followed by selecting a colour from a list, and the object would be changed to that colour. This objective was met.

Objective 7 [sec 1.9.7]:

The Object Inspector displayed all objects in the scene, allowing deletion and colour changing functionality. This objective was met.

Objective 8 [sec 1.9.8]:

The user could use a check button (when a scene existed) to toggle whether the anaglyph was rendered or not. This objective was met.

Objective 9 [sec 1.9.9]:

The user could use keyboard presses to translate, rotate and zoom the camera, along with an extra key to reset the camera in to its original position. This objective was met.

Objective 10 [sec 1.9.10]:

The user could create a scene, close a scene, open a saved scene, and save a scene, with all functionality and validation included. This objective was met.

## 6.2 User feedback

The following feedback was delivered via email.

"I found the program easy to run since it did not require any installation, and was up and functioning within seconds. I also did not need to look at the manual at all to enter points or lines and found the process of creating a scene entirely intuitive. Once I clicked on the Object Inspector, it was also obvious how to delete objects or change their colours. Saving a scene and opening it later was also straightforward, as was toggling and adjusting the anaglyph.

When I tried to adjust the camera position, I also found the translation options to be easy and intuitive.

I was initially unsure how to rotate the scene, and educated guessing failed to discover the controls. However, this was because I did not have the Num Lock key pressed, and I was delighted to discover that this issue was addressed explicitly in a clear and well laid out troubleshooting section of the manual.

I am completely confident, that having played around with the program a little, I could now pull it out the next time I am introducing 3D geometry to a class, and use it effectively without having to consult the manual.

Overall, I was impressed by the ease of use of the program, and feel that it did all that I had originally wanted it to do.

The only slight problem with the program as it currently stands is the "add angle feature".

It can be used effectively, but to do so requires great care on the part of the user.

In particular, if there are four points A,B, C and D in the scene. Then angle ABC is effectively rendered as the angle between the directed segments BA and BC. However, the angle between AB and BC is rendered rather differently since angle



marker then joins a point near A on AB with a point near B on BC. Also, if you ask the program to draw the angle between AB and CD it will add an angle marker, even though it might be nonsensical to talk about the angle between two non intersecting segments.

Another thing I found slightly irritating was that adding each point took a fairly large number of clicks, if the "add point" window did not close automatically after each point it would be easier to add the, usually large, number of points needed for a typical scene.

Separating the display of points, lines and angles in the object inspector would also be easier on the eye in complex scenes.

There are also a number of ways the project could be usefully extended, some easy to implement, and others less so.

I would like to see future versions of the program come with a few pre-saved example scenes, this would make adjusting the anaglyph to its optimal level easier, and give an instant sense of the potential of the program.

Optional x,y and z axes displayed in-scene would also be beneficial, and the ability to label points and lines in scene so they can be referred to in questions, would also be of great help in a classroom.

It is certainly useful to be able to switch the camera back to its default position at the touch of a button, and this functionality could also be extended. For example, the ability to save the camera settings at some advantageous view point, and the switch back and forth between that view and the home view might prove useful.

The program could also be adjusted to make it useful in another context: the scenes it currently creates are ideal for introducing 3D geometry problems in the run up to GCSE, however, at A - level the study on an infinite line in three dimensions, rather than just a line segment, is introduced. A option to add lines which were "infinitely long" might therefore be helpful. In the meantime, adding long line segments is a manageable work around.

Overall, the project achieved all that I had hoped it would and I am sure the finished program will prove useful." (sic)

### 6.3 Analysis of user feedback

1. Mr Rowland feels the project has succeeded in the following areas:

- i. Ease of installation and functionality.
- ii. Intuitive point and line input, and scene management.
- iii. Intuitive object management using the object inspector.
- iv. Intuitive use of the anaglyph.
- v. The aide to 3D vector visualisation the program provided.
- vi. The use of the anaglyph to further aide 3D vector visualisation.
- vii. Full technical functionality.

2. Mr Rowland feels the project identified problems in the following areas:

- i. Adding angles – the program draws the angle from a calculation based on the origin vectors of each line, rather than an intersection or closest points. To use the angle feature effectively, Mr Rowland feels that the user must carefully set up the lines to suit the programs method of drawing the angle.
- ii. Adding angles – the program does not reject angles between non-intersecting lines, a condition Mr Rowland feels to be unhelpful to learning.
- iii. Inputting multiple objects, in-particular points, took a large amount of time since the input point panel closes itself once the point is submitted.
- iv. In scenes with a large number of objects, the list of objects in the object inspector became dense and difficult to read.

3. Mr Rowland identified the following extensions to the project that would be helpful:

- i. The downloaded file containing some premade scenes to act as examples and as a calibration aide for the anaglyph.
- ii. A toggle button enabling a set of axes at the origin to be rendered, a feature that would help with visualisation of objects in relation to the origin.
- iii. The ability to name and label objects in the scene, allowing Mr Rowland to set classroom tasks by easily identifying the objects to the class. For example asking the class to calculate the angle between lines 'A' and 'B'.
- iv. Mr Rowland feels it would be helpful to have a custom camera position management system, allowing the user to create, delete, and switch between custom camera positions (including rotation).
- v. Mr Rowland suggests that adding 'infinitely long' line functionality would be helpful to A level students (as they have to study lines with no fixed length). However Mr Rowland does note this program can currently be overcome by adding large magnitude lines.

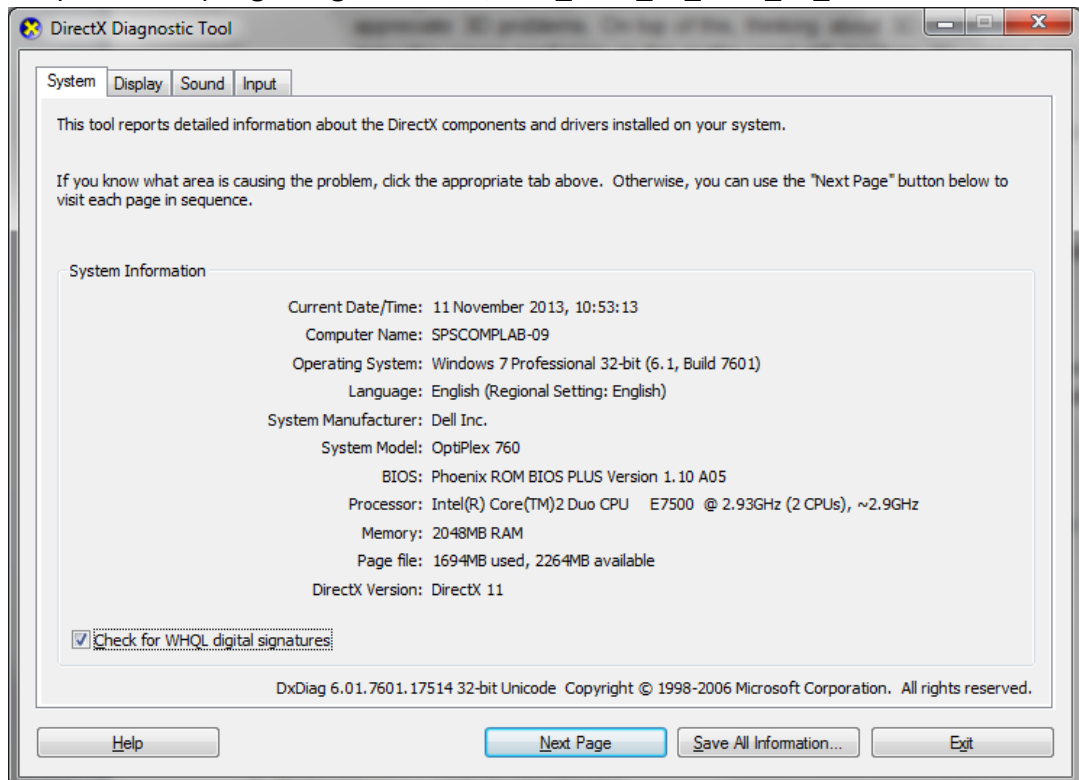
## 6.4 Possible extensions

1. Angle functionality could be improved by calculating the point of intersection and rendering the angle close to that, instead of rendering the angle close to the origin vectors of each line. [sec. 6.3.2.i]
2. Angle functionality could be improved by rejecting the input of angles between non-intersecting lines. [sec. 6.3.2.ii]
3. Input functionality could be improved by not closing input forms when an object has been inputted. Alternatively a text box where the user could type in the coordinates of multiple objects at once would improve input speed greatly. [sec. 6.3.2.iii]
4. Separators between object types in the object inspector panel would be useful in scenes with large numbers of objects. [sec. 6.3.2.iv]
5. [sec. 6.3.3.i]
6. [sec. 6.3.3.ii]
7. [sec. 6.3.3.iii]
8. [sec. 6.3.3.iv]
9. [sec. 6.3.3.v]

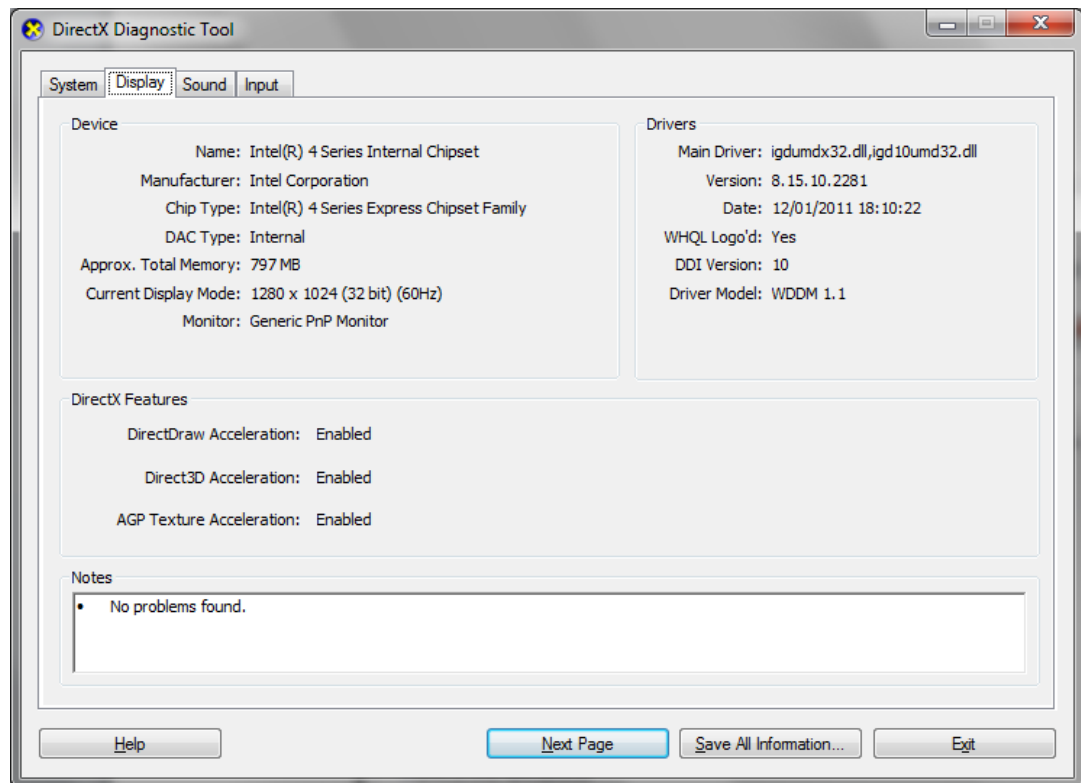
## 7. References

---

1. <http://www.bodurov.com/VectorVisualizer/>
2. [http://www.rapidtables.com/web/color/RGB\\_Color.htm](http://www.rapidtables.com/web/color/RGB_Color.htm)
3. <http://www.mkasolutions.eu/getelement.aspx?E=96>
4. <http://www.access2science.com/rbg/GAME2302/KjellVectorTutorial/vch07/dotprod2.gif>
5. <http://chortle.ccsu.edu/vectorlessons/vch07/dotprod1.gif>
6. [http://en.wikipedia.org/wiki/Comparison\\_of\\_OpenGL\\_and\\_Direct3D](http://en.wikipedia.org/wiki/Comparison_of_OpenGL_and_Direct3D)
7. [http://www.opengl.org/wiki/FAQ#How\\_Does\\_It\\_Work\\_On\\_Windows.3F](http://www.opengl.org/wiki/FAQ#How_Does_It_Work_On_Windows.3F)



8.



- 9.
10. [http://en.wikipedia.org/wiki/Anaglyph\\_3D](http://en.wikipedia.org/wiki/Anaglyph_3D)
11. <http://freeglut.sourceforge.net/>
12. <http://www.zeuscmd.com/tutorials/opengles/19-blending.php>
13. <http://paulbourke.net/stereographics/anaglyph/>
14. <http://paulbourke.net/stereographics/stereorender/>
15. <http://www.opengl.org/>
16. <http://sourceforge.net/projects/taoframework/>

## 8. Code Listing

---

## 8.1 Angle.cs

```

using System;
using Tao.FreeGlut;
using OpenGL;
using System.Collections.Generic;
using System.Windows.Forms;

namespace VectorSoftware
{
    public class Angle : ColoredObject
    {
        public Line Line1;
        public Line Line2;

        public Vector3 point1;
        public Vector3 point2;
        public VBO<Vector3> line;

        public string firstLine;
        public string secondLine;

        public string colorString;

        public Angle(Line l1, Line l2, Vector3 c)
        {
            //Initialises elements for the angle
            elements = new VBO<int>(new int[] { 0, 1 },
BufferTarget.ElementArrayBuffer);

            Line1 = l1; //Line1 is bottom line
            Line2 = l2; //Line2 is top line

            rawColor = c;

            color = new VBO<Vector3>(new Vector3[] { c, c });

            //Calculate normalised direction
            Vector3 direction = NormalizedDirection(Line1);

            //Calculate the amount the line needs to be travelled down to find the
first point of the line
            Vector3 step = new Vector3(direction.x * 0.2, direction.y * 0.2,
direction.z * 0.2);

            //Calculate the point 'step' distance down the line
            point1 = Line1.rawFirstPoint + step;

            //Repeat the above process for the second line
            direction = NormalizedDirection(Line2);
            step = new Vector3(direction.x * 0.2, direction.y * 0.2, direction.z *
0.2);
            point2 = Line2.rawFirstPoint + step;

            line = new VBO<Vector3>(new Vector3[] { point1, point2 });

            firstLine = "( " + Line1.firstPoint + " ) , ( " + Line1.secondPoint + "
)";
            secondLine = "( " + Line2.firstPoint + " ) , ( " + Line2.secondPoint + "
)";
        }
    }
}

```



```
        int index = colors.IndexOf(rowColor);
        if (index > -1)
        {
            colorString = colorStrings[index];
        }
    }

    private Vector3 NormalizedDirection(Line l)
    {
        //Find the direction vector of the line
        Vector3 direction = l.rawSecondPoint - l.rawFirstPoint;

        //Find the modulus of the line
        double mod = Mod(direction);

        //Calculate the unit vector
        float x = direction.x / (float)mod;
        float y = direction.y / (float)mod;
        float z = direction.z / (float)mod;

        return new Vector3(x, y, z);
    }

    private double Mod(Vector3 dVector)
    {
        //Calculate the modulus of the line
        double mod = Math.Sqrt(Math.Pow(dVector.x, 2) + Math.Pow(dVector.y, 2) +
Math.Pow(dVector.z, 2));
        return mod;
    }
}
```

## 8.2 ColorChangePanel.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using OpenGL;
using System.Threading;

namespace VectorSoftware
{
    public partial class ColorChangePanel : Form
    {
        private InputPanel inputPanel;
        private ObjectInspectorPanel objectInspectorPanel;
        private List<Vector3> colors = new List<Vector3>();
        private List<string> colorStrings = new List<string>();
        private Point p;
        private Line l;
        private Angle a;
        private string currentColorString;
        private Vector3 currentRawColor;

        public ColorChangePanel(InputPanel ip, object ob, ObjectInspectorPanel oip)
        {
            inputPanel = ip;
            objectInspectorPanel = oip;
            InitializeComponent();
            initialiseColors();
            updateListBox(ob);
        }

        private void initialiseColors()
        {
            colors.Add(new Vector3(1, 0.5f, 0));
            colors.Add(new Vector3(1, 1, 0));
            colors.Add(new Vector3(0.5f, 1, 0));
            colors.Add(new Vector3(0, 1, 0));
            colors.Add(new Vector3(0, 1, 0.5f));
            colors.Add(new Vector3(0, 0.5f, 1));
            colors.Add(new Vector3(0, 0, 1));
            colors.Add(new Vector3(0.5f, 0, 1));
            colors.Add(new Vector3(1, 0, 1));
            colors.Add(new Vector3(1, 0, 0.5f));

            colorStrings.Add("Orange");
            colorStrings.Add("Yellow");
            colorStrings.Add("Lime");
            colorStrings.Add("Green");
            colorStrings.Add("Turquoise");
            colorStrings.Add("Light Blue");
            colorStrings.Add("Blue");
            colorStrings.Add("Purple");
            colorStrings.Add("Pink");
            colorStrings.Add("Magenta");
        }

        private void updateListBox(object ob)
        {
            if (ob is Point)
            {

```

```

        p = (Point)ob;
    }
    else if (ob is Line)
    {
        l = (Line)ob;
    }
    else if (ob is Angle)
    {
        a = (Angle)ob;
    }

    //Convert each object into information about the object, along side its
colour
    if (p != null)
    {
        int index = colors.IndexOf(p.rawColor);
        currentRawColor = p.rawColor;
        currentColorString = colorStrings[index];

        string identity = "Point: ( " + p.firstPoint + " )";
        identityLabel.Text = identity;
    }
    else if (l != null)
    {
        int index = colors.IndexOf(l.rawColor);
        currentRawColor = l.rawColor;
        currentColorString = colorStrings[index];

        string identity = "Line: ( " + l.firstPoint + " ) ( " +
l.secondPoint + " )";
        identityLabel.Text = identity;
    }
    else if (a != null)
    {
        int index = colors.IndexOf(a.rawColor);
        currentRawColor = a.rawColor;
        currentColorString = colorStrings[index];

        string identity = "Angle between ( " + a.firstLine + " ) ( " +
a.secondLine + " )";
        identityLabel.Text = identity;
    }

    currentColorLabel.Text = currentColorString;

    //Show the list in the list box
    BindingList<string> displayStringList = new
BindingList<string>(colorStrings);
    listBox.DataSource = displayStringList;
}

private void selectColorButton_Click(object sender, EventArgs e)
{
    int index = listBox.SelectedIndex;
    Vector3 newColor = colors[index];

    //If the object still exists, pass it and the new colour to input panel
    if (p != null)
    {
        inputPanel.setNewColor(p, newColor);
    }
    else if (l != null)

```

```
        {
            inputPanel.setNewColor(1, newColor);
        }
        else if (a != null)
        {
            inputPanel.setNewColor(a, newColor);
        }

        Thread.Sleep(10);
        objectInspectorPanel.updateListBox();
        this.Close();
    }
}
```

### 8.3 ColoredObject.cs

```
using System;
using System.Collections.Generic;
using Tao.FreeGlut;
using OpenGL;

namespace VectorSoftware
{
    public abstract class ColoredObject
    {
        public VBO<Vector3> color;
        public Vector3 rawColor;

        public VBO<int> elements;

        //The list of colours a ColoredObject can take
        protected List<Vector3> colors = new List<Vector3>
        {
            new Vector3(1, 0.5f, 0),
            new Vector3(1, 1, 0),
            new Vector3(0.5f, 1, 0),
            new Vector3(0, 1, 0),
            new Vector3(0, 1, 0.5f),
            new Vector3(0, 0.5f, 1),
            new Vector3(0, 0, 1),
            new Vector3(0.5f, 0, 1),
            new Vector3(1, 0, 1),
            new Vector3(1, 0, 0.5f)
        };

        //The names of each colour, in the same order as the list, colors
        protected List<string> colorStrings = new List<string>
        {
            "Orange",
            "Yellow",
            "Lime",
            "Green",
            "Turquoise",
            "Light Blue",
            "Blue",
            "Purple",
            "Pink",
            "Magenta"
        };
    }
}
```

## 8.4 InputAnglePanel.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;

namespace VectorSoftware
{
    public partial class InputAnglePanel : Form
    {
        private InputPanel inputPanel;
        private BindingList<Line> linesInScene;

        public InputAnglePanel(InputPanel ip)
        {
            inputPanel = ip;
            InitializeComponent();

            linesInScene = getLines();

            //If there are no lines in the scene, don't enable the submit button
            if (linesInScene.Count <= 0)
            {
                submitButton.Enabled = false;
            }

            //Create two appropriate lists to be displayed to the user
            BindingList<string> firstDisplayList =
convertLinesToStrings(linesInScene);
            BindingList<string> secondDisplayList =
convertLinesToStrings(linesInScene);

            //Assign the two above lists to the list boxes
            firstLineListBox.DataSource = firstDisplayList;
            secondLineListBox.DataSource = secondDisplayList;
        }

        //Gets a list of the lines in the scene
        private BindingList<Line> getLines()
        {
            List<Line> lines = inputPanel.linesInScene;
            BindingList<Line> bindingLines = new BindingList<Line>(lines);
            return bindingLines;
        }

        //Turn the list of lines into intelligible data to be displayed to the user
        private BindingList<string> convertLinesToStrings(BindingList<Line> lines)
        {
            BindingList<string> stringList = new BindingList<string>();

            foreach (Line l in lines)
            {
                string coordinate = "( " + l.firstPoint + " ) ( " + l.secondPoint +
" )";
                stringList.Add(coordinate);
            }

            return stringList;
        }
    }
}

```

```
private void submitButton_Click(object sender, EventArgs e)
{
    try
    {
        Line firstSelectedLine = linesInScene[firstLineListBox.SelectedIndex];
        Line secondSelectedLine =
linesInScene[secondLineListBox.SelectedIndex];

        //If the two selected lines are the same, display an error label
        if (firstSelectedLine.line == secondSelectedLine.line)
        {
            errorLabel.Visible = true;
        }
        else
        {
            //If the two selected lines are different, submit the angle and
close the panel
            inputPanel.addAngle(
                firstSelectedLine.rawFirstPoint.x,
firstSelectedLine.rawFirstPoint.y, firstSelectedLine.rawFirstPoint.z,
                firstSelectedLine.rawSecondPoint.x,
firstSelectedLine.rawSecondPoint.y, firstSelectedLine.rawSecondPoint.z,
                secondSelectedLine.rawFirstPoint.x,
secondSelectedLine.rawFirstPoint.y, secondSelectedLine.rawFirstPoint.z,
                secondSelectedLine.rawSecondPoint.x,
secondSelectedLine.rawSecondPoint.y, secondSelectedLine.rawSecondPoint.z);

            this.Close();
        }
    }
    catch
    {
        existLabel.Visible = true;
    }
}
}
```

## 8.5 InputLineFromPointsPanel.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;

namespace VectorSoftware
{
    public partial class InputLineFromPointsPanel : Form
    {
        private InputPanel inputPanel;
        private BindingList<Point> pointsInScene;

        public InputLineFromPointsPanel(InputPanel ip)
        {
            inputPanel = ip;
            InitializeComponent();

            pointsInScene = getPoints();

            //Create two appropriate lists to be displayed to the user
            BindingList<string> firstPointsToBeDisplayed =
convertPointsToStrings(pointsInScene);
            BindingList<string> secondPointsToBeDisplayed =
convertPointsToStrings(pointsInScene); ;

            //Assign the two above lists to the list boxes
            firstPointListBox.DataSource = firstPointsToBeDisplayed;
            secondPointListBox.DataSource = secondPointsToBeDisplayed;

            this.Show();
        }

        //Get a list of points in the scene from the input panel
        private BindingList<Point> getPoints()
        {
            List<Point> points = inputPanel.pointsInScene;
            BindingList<Point> bindingPoints = new BindingList<Point>(points);
            return bindingPoints;
        }

        //Turn the list of points into intelligible data to be displayed to the user
        private BindingList<string> convertPointsToStrings(BindingList<Point> points)
        {
            BindingList<string> stringList = new BindingList<string>();

            foreach (Point p in points)
            {
                string coordinate = p.firstPoint;
                stringList.Add(coordinate);
            }

            return stringList;
        }

        private void submitButton_Click(object sender, EventArgs e)
        {
            try
            {
                //Get the two selected points
            }
        }
    }
}
```



```
        Point firstSelectedPoint =
pointsInScene[firstPointListBox.SelectedIndex];
        Point secondSelectedPoint =
pointsInScene[secondPointListBox.SelectedIndex];

        //If the two points are the same, display an error label and don't
submit the points
        if (firstSelectedPoint.rawPoint == secondSelectedPoint.rawPoint)
        {
            errorLabel.Visible = true;
        }
        else
        {
            //If the points are different, submit the points to input panel
and close the form
            inputPanel.addLine(firstSelectedPoint.rawPoint.x,
firstSelectedPoint.rawPoint.y, firstSelectedPoint.rawPoint.z,
secondSelectedPoint.rawPoint.x, secondSelectedPoint.rawPoint.y,
secondSelectedPoint.rawPoint.z);
            this.Close();
        }
    }
    catch
    {
        //Show error label concerning one of the points not existing
        existLabel.Visible = true;
    }
}
}
```

## 8.6 InputLinePanel.cs

```

using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace VectorSoftware
{
    public partial class InputLinePanel : Form
    {
        private InputPanel inputPanel;

        public InputLinePanel(InputPanel ip)
        {
            inputPanel = ip;
            List<Point> points = inputPanel.pointsInScene;
            InitializeComponent();

            //Don't allow the functionality of choosing two points to create the line
            //between if there are no existing points
            if (points.Count <= 0)
            {
                buttonExistingPoints.Enabled = false;
            }

            //Calculates the final point on the line from the inputted equation of the
            //line
            private float[] calculateFinalPoint(float x, float y, float z, float s, float
a, float b, float c)
            {
                float fx = x + (s * a);
                float fy = y + (s * b);
                float fz = z + (s * c);

                return new float[] { fx, fy, fz };
            }

            private void buttonSubmit_Click(object sender, EventArgs e)
            {
                float x = 0;
                float y = 0;
                float z = 0;
                float s = 0;
                float a = 0;
                float b = 0;
                float c = 0;
                bool valid = true;

                //Attempt to read in data from each of the text boxes
                try
                {
                    x = float.Parse(textBoxX.Text);
                }
                catch
                {
                    valid = false;
                }

                try
                {

```

```
        y = float.Parse(textBoxY.Text);
    }
    catch
    {
        valid = false;
    }

    try
    {
        z = float.Parse(textBoxZ.Text);
    }
    catch
    {
        valid = false;
    }

    try
    {
        s = float.Parse(textBoxS.Text);
    }
    catch
    {
        valid = false;
    }

    if (s == 0)
    {
        valid = false;
        labelMagnitude.Visible = true;
    }

    try
    {
        a = float.Parse(textBoxA.Text);
    }
    catch
    {
        valid = false;
    }

    try
    {
        b = float.Parse(textBoxB.Text);
    }
    catch
    {
        valid = false;
    }

    try
    {
        c = float.Parse(textBoxC.Text);
    }
    catch
    {
        valid = false;
    }

    if ((a == 0) && (b == 0) && (c == 0))
    {
        valid = false;
        directionLabel.Visible = true;
    }
}
```

```
    }

    if (valid)
    {
        //If all of the data is successfully read in, pass the line data to
the input panel
        float[] finalPoint = calculateFinalPoint(x, y, z, s, a, b, c);
        inputPanel.addLine(x, y, z, finalPoint[0], finalPoint[1],
finalPoint[2]);

        //Close this panel
        this.Close();
    }

    else
    {
        //If any of the data is invalid, show to error label
        labelInvalid.Visible = true;
    }
}

private void buttonExistingPoints_Click(object sender, EventArgs e)
{
    //Open inputLineFromPointsPanel, and then close this panel
    InputLineFromPointsPanel inputLineFromPointsPanel = new
InputLineFromPointsPanel(inputPanel);
    this.Close();
}

}
}
```

### 8.7 InputPanel.cs

```

using System;
using System.Collections.Generic;
using System.Threading;
using System.Windows.Forms;

using System.ComponentModel; //objectInspector

using System.IO; //file handling
using System.Text;

using Tao.FreeGlut;
using OpenGL;

namespace VectorSoftware
{
    public partial class InputPanel : Form
    {
        private Scene scene;
        public Thread sceneThread;

        private List<Vector3> pointsToBeRendered = new List<Vector3>();
        private List<Vector3[]> linesToBeRendered = new List<Vector3[]>();
        private List<Vector3[]> anglesToBeRendered = new List<Vector3[]>();

        private List<Vector3[]> pointsToBeRenderedWithColor = new List<Vector3[]>();
        private List<Vector3[]> linesToBeRenderedWithColor = new List<Vector3[]>();
        private List<Vector3[]> anglesToBeRenderedWithColor = new List<Vector3[]>();

        public Point pointToBeDeleted;
        public Line lineToBeDeleted;
        public Angle angleToBeDeleted;

        private object objectWithColorChange;
        private Vector3 newColor;

        public List<Point> pointsInScene = new List<Point>();
        public List<Line> linesInScene = new List<Line>();
        public List<Angle> anglesInScene = new List<Angle>();

        private bool anaglyphToggle = false;
        private int anaglyphAdjust;

        public InputPanel()
        {
            InitializeComponent();
        }

        public void createNewScene()
        {
            //Initialise the thread
            sceneThread = new Thread(new ParameterizedThreadStart(initScene));
            sceneThread.SetApartmentState(ApartmentState.STA);

            //Start the thread, passing it a reference to input panel (to allow object
fetches) sceneThread.Start(this);

            enableComponents();
        }
    }
}

```

```
public void closeScene()
{
    //If the attempt at calling this function was flagged as invalid due to a
    cross-thread call, allow the call
    if (InvokeRequired)
    {
        Invoke(new MethodInvoker(closeScene));
    }

    scene = null;

    disableComponents();

    //Try to close the sceneThread
    try
    {
        sceneThread.Abort();
    }
    catch
    {
        //If the sceneThread can't be closed, it already is closed
    }
}

private void enableComponents()
{
    //Enable input buttons
    newPointButton.Enabled = true;
    newLineButton.Enabled = true;
    newAngleButton.Enabled = true;

    //Enable objectInspector button
    objectInspectorButton.Enabled = true;

    //Disable create scene control buttons
    createNewSceneButton.Enabled = false;
    openSceneButton.Enabled = false;

    //Enable current scene control buttons
    saveSceneButton.Enabled = true;
    closeSceneButton.Enabled = true;

    //Enable anaglyph check box
    anaglyphCheckBox.Enabled = true;
}

private void disableComponents()
{
    //Disable input buttons
    newPointButton.Enabled = false;
    newLineButton.Enabled = false;
    newAngleButton.Enabled = false;

    //Disable objectInspector button
    objectInspectorButton.Enabled = false;

    //Enable create scene control buttons
    createNewSceneButton.Enabled = true;
    openSceneButton.Enabled = true;

    //Disable current scene control buttons
}
```

```

        saveSceneButton.Enabled = false;
        closeSceneButton.Enabled = false;

        //Set anaglyph render to false, and disable the check box
        anaglyphCheckBox.Checked = false;
        anaglyphCheckBox.Enabled = false;
    }

    private void initScene(object obj)
    {
        InputPanel ip = (InputPanel)obj;
        scene = new Scene(ip);
    }

    private readonly object pointLock = new object();
    private readonly object lineLock = new object();
    private readonly object angleLock = new object();
    private readonly object colorLock = new object();
    private readonly object colorPointLock = new object();
    private readonly object colorLineLock = new object();
    private readonly object colorAngleLock = new object();
    private readonly object anaglyphLock = new object();
    private readonly object anaglyphAdjustLock = new object();

    public void addPoint(float x, float y, float z)
    {
        lock (pointLock)
        {
            pointsToBeRendered.Add(new Vector3(x, y, z));
        }
    }

    public void addLine(float x, float y, float z, float a, float b, float c)
    {
        lock (lineLock)
        {
            linesToBeRendered.Add(new Vector3[] { new Vector3(x, y, z), new
Vector3(a, b, c) });
        }
    }

    public void addAngle(float l1x, float l1y, float l1z, float l1a, float l1b,
float l1c, float l2x, float l2y, float l2z, float l2a, float l2b, float l2c)
    {
        lock (angleLock)
        {
            Vector3 line1FirstPoint = new Vector3(l1x, l1y, l1z);
            Vector3 line1SecondPoint = new Vector3(l1a, l1b, l1c);
            Vector3 line2FirstPoint = new Vector3(l2x, l2y, l2z);
            Vector3 line2SecondPoint = new Vector3(l2a, l2b, l2c);

            anglesToBeRendered.Add(new Vector3[] { line1FirstPoint,
line1SecondPoint, line2FirstPoint, line2SecondPoint });
        }
    }

    public List<Vector3> getPoints()
    {
        lock (pointLock)
        {
            List<Vector3> temp = pointsToBeRendered;
            pointsToBeRendered = new List<Vector3>();
        }
    }

```

```
        return temp;
    }
}

public List<Vector3[]> getLines()
{
    lock (lineLock)
    {
        List<Vector3[]> temp = linesToBeRendered;
        linesToBeRendered = new List<Vector3[]>();
        return temp;
    }
}

public List<Vector3[]> getAngles()
{
    lock (angleLock)
    {
        List<Vector3[]> temp = anglesToBeRendered;
        anglesToBeRendered = new List<Vector3[]>();
        return temp;
    }
}

public List<Vector3[]> getPointsWithColor()
{
    lock (colorPointLock)
    {
        List<Vector3[]> temp = pointsToBeRenderedWithColor;
        pointsToBeRenderedWithColor = new List<Vector3[]>();
        return temp;
    }
}

public List<Vector3[]> getLinesWithColor()
{
    lock (colorLineLock)
    {
        List<Vector3[]> temp = linesToBeRenderedWithColor;
        linesToBeRenderedWithColor = new List<Vector3[]>();
        return temp;
    }
}

public List<Vector3[]> getAnglesWithColor()
{
    lock (colorAngleLock)
    {
        List<Vector3[]> temp = anglesToBeRenderedWithColor;
        anglesToBeRenderedWithColor = new List<Vector3[]>();
        return temp;
    }
}

public void setPointsInScene(List<Point> p)
{
    pointsInScene = p;
}

public void setLinesInScene(List<Line> l)
{
    linesInScene = l;
}
```



```
}

public void setAnglesInScene(List<Angle> a)
{
    anglesInScene = a;
}

public void setNewColor(object o, Vector3 color)
{
    lock (colorLock)
    {
        objectWithColorChange = o;
        newColor = color;
    }
}

public object getColorChangeObject()
{
    lock (colorLock)
    {
        object temp = objectWithColorChange;
        objectWithColorChange = null;
        return temp;
    }
}

public Vector3 getNewColor()
{
    lock (colorLock)
    {
        return newColor;
    }
}

private void setAnaglyphToggled(bool value)
{
    lock (anaglyphLock)
    {
        anaglyphToggle = value;
    }
}

public bool getAnaglyphToggled()
{
    lock (anaglyphLock)
    {
        return anaglyphToggle;
    }
}

private void setAnaglyphAdjust(int value)
{
    lock (anaglyphAdjustLock)
    {
        anaglyphAdjust = value;
    }
}

public int getAnaglyphAdjust()
{
    lock (anaglyphAdjustLock)
    {
```

```
        return anaglyphAdjust;
    }
}

private void newPointButton_Click(object sender, EventArgs e)
{
    InputPointPanel inputPointPanel = new InputPointPanel(this);
    inputPointPanel.Show();
}

private void newLineButton_Click(object sender, EventArgs e)
{
    InputLinePanel inputLinePanel = new InputLinePanel(this);
    inputLinePanel.Show();
}

private void newAngleButton_Click(object sender, EventArgs e)
{
    InputAnglePanel inputAnglePanel = new InputAnglePanel(this);
    inputAnglePanel.Show();
}

private void objectInspectorButton_Click(object sender, EventArgs e)
{
    ObjectInspectorPanel objectInspectorPanel = new
ObjectInspectorPanel(this);
    objectInspectorPanel.Show();
}

private void createNewSceneButton_Click(object sender, EventArgs e)
{
    createNewScene();
}

private void closeSceneButton_Click(object sender, EventArgs e)
{
    closeScene();
}

private void openSceneButton_Click(object sender, EventArgs e)
{
    openScene();
}

private void saveSceneButton_Click(object sender, EventArgs e)
{
    saveScene();
}

private void anaglyphCheckBox_CheckedChanged(object sender, EventArgs e)
{
    //Set the value of the check box to the global variable, anaglyphToggled
    setAnaglyphToggled(anaglyphCheckBox.Checked);
    if (anaglyphCheckBox.Checked)
    {
        anaglyphLabel.Enabled = true;
        trackBarAnaglyph.Enabled = true;
        setAnaglyphAdjust(trackBarAnaglyph.Value);
    }
    else
    {
        anaglyphLabel.Enabled = false;
    }
}
```

```

        trackBarAnaglyph.Enabled = false;
    }
}

private void trackBarAnaglyph_ValueChanged(object sender, EventArgs e)
{
    setAnaglyphAdjust(trackBarAnaglyph.Value);
}

private void saveScene()
{
    //Open the windows save file window
    saveFileDialog.ShowDialog();
    string filePath = saveFileDialog.FileName;

    if (!File.Exists(filePath))
    {
        File.Create(filePath).Close();
    }

    //Remove all previous data from file
    File.WriteAllText(filePath, string.Empty);

    string delimiter = ",";

    List<string[]> outputList = new List<string[]>();

    //Take each object and convert it into a line which is ready to be written
into the file
    foreach (Point p in pointsInScene)
    {
        string x = p.rawPoint.x.ToString();
        string y = p.rawPoint.y.ToString();
        string z = p.rawPoint.z.ToString();
        string colorX = p.rawColor.x.ToString();
        string colorY = p.rawColor.y.ToString();
        string colorZ = p.rawColor.z.ToString();

        string[] line = {"point", x, y, z, colorX, colorY, colorZ };
        outputList.Add(line);
    }
    outputList.Add(new string[] { "" });
    foreach (Line l in linesInScene)
    {
        string fx = l.rawFirstPoint.x.ToString();
        string fy = l.rawFirstPoint.y.ToString();
        string fz = l.rawFirstPoint.z.ToString();
        string sx = l.rawSecondPoint.x.ToString();
        string sy = l.rawSecondPoint.y.ToString();
        string sz = l.rawSecondPoint.z.ToString();
        string colorX = l.rawColor.x.ToString();
        string colorY = l.rawColor.y.ToString();
        string colorZ = l.rawColor.z.ToString();

        string[] line = {"line", fx, fy, fz, sx, sy, sz, colorX, colorY,
colorZ };
        outputList.Add(line);
    }
    outputList.Add(new string[] { "" });
    foreach (Angle a in anglesInScene)
    {
        string f1x = a.Line1.rawFirstPoint.x.ToString();

```

```

        string f1y = a.Line1.rawFirstPoint.y.ToString();
        string f1z = a.Line1.rawFirstPoint.z.ToString();
        string s1x = a.Line1.rawSecondPoint.x.ToString();
        string s1y = a.Line1.rawSecondPoint.y.ToString();
        string s1z = a.Line1.rawSecondPoint.z.ToString();

        string f2x = a.Line2.rawFirstPoint.x.ToString();
        string f2y = a.Line2.rawFirstPoint.y.ToString();
        string f2z = a.Line2.rawFirstPoint.z.ToString();
        string s2x = a.Line2.rawSecondPoint.x.ToString();
        string s2y = a.Line2.rawSecondPoint.y.ToString();
        string s2z = a.Line2.rawSecondPoint.z.ToString();

        string colorX = a.rawColor.x.ToString();
        string colorY = a.rawColor.y.ToString();
        string colorZ = a.rawColor.z.ToString();

        string[] line = {"angle", f1x, f1y, f1z, s1x, s1y, s1z,
f2x, f2y, f2z, s2x, s2y, s2z, colorX, colorY, colorZ };
        outputList.Add(line);
    }

    //Convert the lines into a table of all lines
    string[][] output = outputList.ToArray();

    //Get the max number of elements in a row
    int length = output.GetLength(0);
    StringBuilder sb = new StringBuilder();
    for (int index = 0; index < length; index++)
    {
        //Join every element with a comma (required for the csv file)
        sb.AppendLine(string.Join(delimiter, output[index]));
    }

    //Write all data from the StringBuilder to the file
    File.AppendAllText(filePath, sb.ToString());
}

public void openScene()
{
    //Show the windows open file window
    openFileDialog.ShowDialog();
}

//Called when the user has selected a valid file in the open file window
private void openFileDialog_FileOk(object sender, CancelEventArgs e)
{
    //Get the file selected by the user in the open file window
    string filename = openFileDialog.FileName;

    string[] lines = new string[] { };

    try
    {
        //Attempt to read in all data from the file
        var reader = new StreamReader(File.OpenRead(filename));
        lines = File.ReadAllLines(filename);

        //If all data is successfully read, create a new scene
        createNewScene();
    }
    catch

```

```

{
    //If there is a problem in reading the file, show an error message
    MessageBox.Show("Invalid File");
}

foreach (string s in lines)
{
    //Create an array of values by separating values where a comma exists
    string[] data = s.Split(',');

    if (data[0] == "point")
    {
        //Attempt to read in point data
        try
        {
            float x = float.Parse(data[1]);
            float y = float.Parse(data[2]);
            float z = float.Parse(data[3]);
            float cx = float.Parse(data[4]);
            float cy = float.Parse(data[5]);
            float cz = float.Parse(data[6]);

            Vector3 point = new Vector3(x, y, z);
            Vector3 color = new Vector3(cx, cy, cz);
            Vector3[] passedData = new Vector3[] { point, color };

            //Pass the data to a global variable within input panel, which
            will be fetched by the scene
            pointsToBeRenderedWithColor.Add(passedData);
        }
        catch
        {
            //If some data is invalid, don't read in that line of the
            file, but carry on with other lines
            MessageBox.Show("Data could not be read from the file");
        }
    }
    else if (data[0] == "line")
    {
        //Attempt to read in line data
        try
        {
            float fx = float.Parse(data[1]);
            float fy = float.Parse(data[2]);
            float fz = float.Parse(data[3]);
            float sx = float.Parse(data[4]);
            float sy = float.Parse(data[5]);
            float sz = float.Parse(data[6]);
            float cx = float.Parse(data[7]);
            float cy = float.Parse(data[8]);
            float cz = float.Parse(data[9]);

            Vector3 firstPoint = new Vector3(fx, fy, fz);
            Vector3 secondPoint = new Vector3(sx, sy, sz);
            Vector3 color = new Vector3(cx, cy, cz);
            Vector3[] passedData = new Vector3[] { firstPoint,
secondPoint, color };

            //If the first point on the line is the same as the second
            point on the line, show an error message and don't pass the data to the scene
            if(firstPoint == secondPoint)

```

```

        {
            MessageBox.Show("Invalid line found
and could not be imported");
        }
        else
        {
            linesToBeRenderedWithColor.Add(passedData);
        }
    }
    catch
    {
        //If some data is invalid, don't read in that line of the
file, but carry on with other lines
        MessageBox.Show("Data could not be read from the file");
    }
}

else if (data[0] == "angle")
{
    //Attempt to read in angle data
    try
    {
        float f1x = float.Parse(data[1]);
        float f1y = float.Parse(data[2]);
        float f1z = float.Parse(data[3]);
        float s1x = float.Parse(data[4]);
        float s1y = float.Parse(data[5]);
        float s1z = float.Parse(data[6]);
        float f2x = float.Parse(data[7]);
        float f2y = float.Parse(data[8]);
        float f2z = float.Parse(data[9]);
        float s2x = float.Parse(data[10]);
        float s2y = float.Parse(data[11]);
        float s2z = float.Parse(data[12]);
        float cx = float.Parse(data[13]);
        float cy = float.Parse(data[14]);
        float cz = float.Parse(data[15]);

        Vector3 f1 = new Vector3(f1x, f1y, f1z);
        Vector3 s1 = new Vector3(s1x, s1y, s1z);
        Vector3 f2 = new Vector3(f2x, f2y, f2z);
        Vector3 s2 = new Vector3(s2x, s2y, s2z);
        Vector3 c = new Vector3(cx, cy, cz);

        //Make sure the two lines aren't the same, and make sure the
two lines are both valid (the first and second point on each line are not the same)
        if (((f1 == f2) && (s1 == s2)) || ((s1 == f2) && (s2 == f1))
|| (f1 == s1) || (f2 == s2))
        {
            MessageBox.Show("Invalid angle found and could not be
imported");
        }
        else
        {
            //Pass the data to the input panel, where it will be
fetched by the scene
            anglesToBeRenderedWithColor.Add(new Vector3[] { f1, s1,
f2, s2, c });
        }
    }
    catch
    {

```

```
        //If some data is invalid, don't read in that line of the
file, but carry on with other lines
        MessageBox.Show("Data could not be read from
the file");
    }
}

private void InputPanel_Closing(object sender, EventArgs e)
{
    //Close the sceneThread, if it is running
    try
    {
        sceneThread.Abort();
    }
    catch { }
    //Make sure the whole application closes
    Application.Exit();
}
}
```

## 8.8 InputPointPanel.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;

using OpenGL;

namespace VectorSoftware
{
    public partial class InputPointPanel : Form
    {
        private InputPanel inputPanel;

        public InputPointPanel(InputPanel ip)
        {
            InitializeComponent();
            inputPanel = ip;
        }

        private void buttonSubmit_Click(object sender, EventArgs e)
        {
            float x = 0;
            float y = 0;
            float z = 0;
            bool valid = true;

            //Attempt to get a float from each test box
            //If the data isn't valid, set valid to false

            try
            {
                x = float.Parse(textBoxX.Text);
            }
            catch
            {
                valid = false;
            }

            try
            {
                y = float.Parse(textBoxY.Text);
            }
            catch
            {
                valid = false;
            }

            try
            {
                z = float.Parse(textBoxZ.Text);
            }
            catch
            {
                valid = false;
            }

            //If all data was successfully obtained as floats
            if (valid)
            {

```



```
        //Send the data to the input panel
        inputPanel.addPoint(x, y, z);
        this.Close();
    }

    else
    {
        //Display the error message if the data inputted isn't valid
        labelInvalid.Visible = true;
    }
}
}
```

## 8.9 Line.cs

```

using System;
using System.Collections.Generic;
using Tao.FreeGlut;
using OpenGL;

namespace VectorSoftware
{
    public class Line : ColoredObject
    {
        public VBO<Vector3> line;
        public Vector3 rawFirstPoint;
        public Vector3 rawSecondPoint;

        public string firstPoint;
        public string secondPoint;
        public string colorString;

        public Line(float ox, float oy, float oz, float fx, float fy, float fz,
Vector3 c)
        {
            //Initialises elements for the line
            elements = new VBO<int>(new int[] { 0, 1 },
BufferTarget.ElementArrayBuffer);

            line = new VBO<Vector3>(new Vector3[] { new Vector3(ox, oy, oz), new
Vector3(fx, fy, fz) });
            color = new VBO<Vector3>(new Vector3[] { c, c });

            //Stores raw data about the line
            rawColor = c;
            rawFirstPoint = new Vector3(ox, oy, oz);
            rawSecondPoint = new Vector3(fx, fy, fz);

            firstPoint = ox.ToString() + "," + oy.ToString() + "," + oz.ToString();
            secondPoint = fx.ToString() + "," + fy.ToString() + "," + fz.ToString();

            //Find the appropriate name of the lines colour
            int index = colors.IndexOf(rawColor);
            if (index > -1)
            {
                colorString = colorStrings[index];
            }
        }
    }
}

```

### 8.10 ObjectInspectorPanel.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Threading;
using System.Windows.Forms;

namespace VectorSoftware
{
    public partial class ObjectInspectorPanel : Form
    {
        private InputPanel inputPanel;

        private BindingList<object> objectsInScene = new BindingList<object>();
        private BindingList<Point> pointsInScene;
        private BindingList<Line> linesInScene;
        private BindingList<Angle> anglesInScene;

        public ObjectInspectorPanel(InputPanel ip)
        {
            inputPanel = ip;
            InitializeComponent();
            updateListBox();
        }

        //Input data into the list box
        public void updateListBox()
        {
            objectsInScene = new BindingList<object>();

            //Gets all objects from the input panel
            pointsInScene = getPoints();
            linesInScene = getLines();
            anglesInScene = getAngles();

            //Adds each object to one Binding List
            foreach (Point p in pointsInScene)
            {
                objectsInScene.Add(p);
            }
            foreach (Line l in linesInScene)
            {
                objectsInScene.Add(l);
            }
            foreach (Angle a in anglesInScene)
            {
                objectsInScene.Add(a);
            }

            //Converts the Binding List to appropriate strings, and puts this data in
            the list for the user
            objectListBox.DataSource = convertToString(objectsInScene);
        }

        private void clearListBox()
        {
            objectListBox.DataSource = null;
        }
    }
}
```

```

        objectListBox.Items.Clear();
    }

    private BindingList<Point> getPoints()
    {
        List<Point> points = inputPanel.pointsInScene;
        BindingList<Point> bindingPoints = new BindingList<Point>(points);
        return bindingPoints;
    }

    private BindingList<Line> getLines()
    {
        List<Line> lines = inputPanel.linesInScene;
        BindingList<Line> bindingLines = new BindingList<Line>(lines);
        return bindingLines;
    }

    private BindingList<Angle> getAngles()
    {
        List<Angle> angles = inputPanel.anglesInScene;
        BindingList<Angle> bindingAngles = new BindingList<Angle>(angles);
        return bindingAngles;
    }

    //Converts each object to intelligible data
    private BindingList<string> convertToString(BindingList<object> list)
    {
        BindingList<string> returnList = new BindingList<string>();

        foreach (object o in list)
        {
            if (o is Point)
            {
                Point p = (Point)o;
                string message = "Point: ( " + p.firstPoint + " )      " +
p.colorString;
                returnList.Add(message);
            }

            else if (o is Line)
            {
                Line l = (Line)o;
                string message = "Line: ( " + l.firstPoint + " )      ( " +
l.secondPoint + " )      " + l.colorString;
                returnList.Add(message);
            }

            else if (o is Angle)
            {
                Angle a = (Angle)o;
                string message = "Angle between " + a.firstLine + " and " +
a.secondLine + "      " + a.colorString;
                returnList.Add(message);
            }
        }

        return returnList;
    }

    //If the changeColorButton is clicked
    private void changeColorButton_Click(object sender, EventArgs e)
    {

```

```

        //If there's a valid item selected
        if (objectListBox.Items.Count > 0)
        {
            //Opens the colourChangePanel, passing the data about the selected
object to it
            ColorChangePanel colorChangePanel = new ColorChangePanel(inputPanel,
objectsInScene[objectListBox.SelectedIndex], this);
            colorChangePanel.Show();
        }
    }

    //If the deleteObjectButton is clicked
    private void deleteObjectButton_Click(object sender, EventArgs e)
    {
        //If there's a valid item selected
        if (objectListBox.Items.Count > 0)
        {
            object objectToBeDeleted =
objectsInScene[objectListBox.SelectedIndex];

            //Finds the correct type of object
            if (objectToBeDeleted is Point)
            {
                Point p = (Point)objectToBeDeleted;
                //Passes the deleted point to the input panel
                inputPanel.pointToBeDeleted = p;
                //Stops the list updating before the object has been deleted from
the scene, by pausing for 10ms
                Thread.Sleep(10);
                clearListBox();
                updateListBox();
            }
            else if (objectToBeDeleted is Line)
            {
                Line l = (Line)objectToBeDeleted;
                inputPanel.lineToBeDeleted = l;
                Thread.Sleep(10);
                clearListBox();
                updateListBox();
            }
            else if (objectToBeDeleted is Angle)
            {
                Angle a = (Angle)objectToBeDeleted;
                inputPanel.angleToBeDeleted = a;
                Thread.Sleep(10);
                clearListBox();
                updateListBox();
            }
        }
    }

    private void refreshButton_Click(object sender, EventArgs e)
    {
        updateListBox();
    }
}

```

### 8.11 Point.cs

```
using System;
using System.Collections.Generic;
using Tao.FreeGlut;
using OpenGL;

namespace VectorSoftware
{
    public class Point : ColoredObject
    {
        public VBO<Vector3> point;
        public Vector3 rawPoint;

        public string firstPoint;
        public string colorString;

        public Point(float x, float y, float z, Vector3 c)
        {
            //Initialise elements for the point
            elements = new VBO<int>(new int[] { 0 }, BufferTarget.ElementArrayBuffer);

            point = new VBO<Vector3>(new Vector3[] { new Vector3(x, y, z) });
            color = new VBO<Vector3>(new Vector3[] { c });

            //Store raw data about the point
            rawColor = c;
            rawPoint = new Vector3(x, y, z);

            firstPoint = x.ToString() + "," + y.ToString() + "," + z.ToString();

            //Find the appropriate name of the points colour
            int index = colors.IndexOf(rawColor);
            if (index > -1)
            {
                colorString = colorStrings[index];
            }
        }
    }
}
```

### 8.12 Program.cs

```
using System;
using System.Windows.Forms;

namespace VectorSoftware
{
    class Program
    {
        [STAThread]
        static void Main(string[] args)
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new InputPanel());
        }
    }
}
```

### 8.13 Scene.cs

```

using System;
using System.Collections.Generic; //needed for lists
using System.Windows.Input;
using System.Windows.Forms;

using System.ComponentModel; //needed for binding list

using Tao.FreeGlut;
using OpenGL;

namespace VectorSoftware
{
    public class Scene
    {
        private InputPanel inputPanel;

        private int width = 1000, height = 500;
        private ShaderProgram program;

        public List<Point> points = new List<Point>();
        public List<Line> lines = new List<Line>();
        public List<Angle> angles = new List<Angle>();

        private Vector3[] colors = new Vector3[10];
        private List<Vector3> unusedColors = new List<Vector3>();

        private System.Diagnostics.Stopwatch watch;
        private float angleX = 0, angleY = 0, angleZ = 0;
        private float cameraX = 0, cameraY = 0, cameraZ = 0;

        private bool anaglyphToggled = false;
        private Vector3 anaglyphCyan = new Vector3(0, 0.5, 0.5);
        private Vector3 anaglyphRed = new Vector3(1, 0, 0);

        public Scene(InputPanel ip)
        {
            //Stops GLUT trying to re-initialise if it has already been initialised.
            int glutTime = Glut.glutGet(Glut.GLUT_ELAPSED_TIME);
            if (glutTime <= 0)
            {
                Glut.glutInit();
            }

            //Initialises GLUT
            Glut.glutInitDisplayMode(Glut.GLUT_DOUBLE | Glut.GLUT_DEPTH);
            Glut.glutInitWindowSize(width, height);
            Glut.glutCreateWindow("Scene");

            Glut.glutSetOption(Glut.GLUT_ACTION_ON_WINDOW_CLOSE, 1);
            Glut.glutCloseFunc(closeFunc);

            Glut.glutIdleFunc(OnRenderFrame);

            Gl.Enable(EnableCap.Blend);

            //Creates the rendering ShaderProgram
            program = new ShaderProgram(VertexShader, FragmentShader);

            program.Use();
        }
    }
}

```



```

program["projection_matrix"].SetValue(Matrix4.CreatePerspectiveFieldOfView(0.45f,
(float)width / height, 0.1f, 1000f));
    resetCamera();

    //Initialises colors
    colors[0] = new Vector3(1, 0.5f, 0);
    colors[1] = new Vector3(1, 1, 0);
    colors[2] = new Vector3(0.5f, 1, 0);
    colors[3] = new Vector3(0, 1, 0);
    colors[4] = new Vector3(0, 1, 0.5f);
    colors[5] = new Vector3(0, 0.5f, 1);
    colors[6] = new Vector3(0, 0, 1);
    colors[7] = new Vector3(0.5f, 0, 1);
    colors[8] = new Vector3(1, 0, 1);
    colors[9] = new Vector3(1, 0, 0.5f);

    //Starts the Stopwatch
    watch = System.Diagnostics.Stopwatch.StartNew();

    inputPanel = ip;

    //Enters the main loop
    Glut.glutMainLoop();
}

//The main loop
private void OnRenderFrame()
{
    //Gets the change in time between the previous frame and the current frame
    watch.Stop();
    float deltaTime = (float)watch.ElapsedTicks /
System.Diagnostics.Stopwatch.Frequency;
    watch.Restart();

    //Checks for rotation key presses
    if (Keyboard.IsKeyDown(Key.NumPad8))
    {
        angleX += deltaTime;
    }
    if (Keyboard.IsKeyDown(Key.NumPad2))
    {
        angleX -= deltaTime;
    }
    if (Keyboard.IsKeyDown(Key.NumPad4))
    {
        angleY += deltaTime;
    }
    if (Keyboard.IsKeyDown(Key.NumPad6))
    {
        angleY -= deltaTime;
    }
    if (Keyboard.IsKeyDown(Key.NumPad7))
    {
        angleZ += deltaTime;
    }
    if (Keyboard.IsKeyDown(Key.NumPad9))
    {
        angleZ -= deltaTime;
    }

    //Checks for translation key presses

```

```

    if (Keyboard.IsKeyDown(Key.Right))
    {
        cameraX += deltaTime;
    }
    if (Keyboard.IsKeyDown(Key.Left))
    {
        cameraX -= deltaTime;
    }
    if (Keyboard.IsKeyDown(Key.Up))
    {
        cameraY += deltaTime;
    }
    if (Keyboard.IsKeyDown(Key.Down))
    {
        cameraY -= deltaTime;
    }
    if (Keyboard.IsKeyDown(Key.PageUp))
    {
        cameraZ += deltaTime;
    }
    if (Keyboard.IsKeyDown(Key.PageDown))
    {
        cameraZ -= deltaTime;
    }

    //Checks for the reset key press
    if (Keyboard.IsKeyDown(Key.Home))
    {
        resetCamera();
    }

    //Assigns the field of view to the size of the window (required for
functionality with window resizing)
    Gl.Viewport(0, 0, Glut.glutGet(Glut.GLUT_WINDOW_WIDTH),
Glut.glutGet(Glut.GLUT_WINDOW_HEIGHT));

    //Clears the previous render frame's buffer
    Gl.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);

    //Blends overlapping lines to aid anaglyph functionality
    Gl.BlendFunc(BlendingFactorSrc.OneMinusDstColor,
BlendingFactorDest.DstColor);

    //Inputs raw objects from inputPanel
    List<Vector3> inputtedPointsRaw = inputPanel.getPoints();
    foreach (Vector3 v in inputtedPointsRaw)
    {
        Point p = new Point(v.x, v.y, v.z, getColor());
        points.Add(p);
    }

    List<Vector3[]> inputtedLinesRaw = inputPanel.getLines();
    foreach (Vector3[] v in inputtedLinesRaw)
    {
        Line l = new Line(v[0].x, v[0].y, v[0].z, v[1].x, v[1].y, v[1].z,
getColor());
        lines.Add(l);
    }

    List<Vector3[]> inputtedAnglesRaw = inputPanel.getAngles();
    foreach (Vector3[] v in inputtedAnglesRaw)
    {

```

```

//Make the colour any color in array of colors as they won't actually
be rendered
Line line1 = new Line(v[0].x, v[0].y, v[0].z, v[1].x, v[1].y, v[1].z,
colors[0]);
Line line2 = new Line(v[2].x, v[2].y, v[2].z, v[3].x, v[3].y, v[3].z,
colors[0]);

Angle a = new Angle(line1, line2, getColor());
angles.Add(a);
}

//Inputs coloured objects from inputPanel
List<Vector3[]> inputtedPoints = inputPanel.getPointsWithColor();
foreach (Vector3[] v in inputtedPoints)
{
    Point p = new Point(v[0].x, v[0].y, v[0].z, v[1]);
    points.Add(p);
}

List<Vector3[]> inputtedLines = inputPanel.getLinesWithColor();
foreach (Vector3[] v in inputtedLines)
{
    Line l = new Line(v[0].x, v[0].y, v[0].z, v[1].x, v[1].y, v[1].z,
v[2]);
    lines.Add(l);
}

List<Vector3[]> inputtedAngles = inputPanel.getAnglesWithColor();
foreach (Vector3[] v in inputtedAngles)
{
    //Make the colour any color in array of colors as they won't actually
be rendered
    Line line1 = new Line(v[0].x, v[0].y, v[0].z, v[1].x, v[1].y, v[1].z,
colors[0]);
    Line line2 = new Line(v[2].x, v[2].y, v[2].z, v[3].x, v[3].y, v[3].z,
colors[0]);

    Angle a = new Angle(line1, line2, v[4]);
    angles.Add(a);
}

//Checks for objects to be deleted
if (inputPanel.pointToBeDeleted != null)
{
    Point pointToBeDeleted = inputPanel.pointToBeDeleted;
    if(points.Contains(pointToBeDeleted))
    {
        //Delete point
        points.Remove(pointToBeDeleted);
    }
    inputPanel.pointToBeDeleted = null;
}

if (inputPanel.lineToBeDeleted != null)
{
    Line lineToBeDeleted = inputPanel.lineToBeDeleted;
    if (lines.Contains(lineToBeDeleted))
    {
        //Delete line
        lines.Remove(lineToBeDeleted);
    }
    inputPanel.lineToBeDeleted = null;
}

```

```

    }

    if (inputPanel.angleToBeDeleted != null)
    {
        Angle angleToBeDeleted = inputPanel.angleToBeDeleted;
        if (angles.Contains(angleToBeDeleted))
        {
            //Delete angle
            angles.Remove(angleToBeDeleted);
        }
        inputPanel.angleToBeDeleted = null;
    }

    //Checks for object with a colour change request from the user
    object colorChangeObject = inputPanel.getColorChangeObject();
    if (colorChangeObject != null)
    {
        //Gets the new colour
        Vector3 newColor = inputPanel.getNewColor();

        if (colorChangeObject is Point)
        {
            Point p = (Point)colorChangeObject;
            if (points.Contains(p))
            {
                points.Remove(p);
                p = new Point(p.rawPoint.x, p.rawPoint.y, p.rawPoint.z,
newColor);
                points.Add(p);
            }
        }
        else if (colorChangeObject is Line)
        {
            Line l = (Line)colorChangeObject;
            if (lines.Contains(l))
            {
                lines.Remove(l);
                l = new Line(l.rawFirstPoint.x, l.rawFirstPoint.y,
l.rawFirstPoint.z, l.rawSecondPoint.x, l.rawSecondPoint.y, l.rawSecondPoint.z,
newColor);
                lines.Add(l);
            }
        }
        else if (colorChangeObject is Angle)
        {
            Angle a = (Angle)colorChangeObject;
            if (angles.Contains(a))
            {
                angles.Remove(a);
                a = new Angle(a.Line1, a.Line2, newColor);
                angles.Add(a);
            }
        }
    }

    //Pass current objects in scene to the inputPanel
    inputPanel.setPointsInScene(points);
    inputPanel.setLinesInScene(lines);
    inputPanel.setAnglesInScene(angles);

    //Gets whether the anaglyph should be rendered or not
    anaglyphToggled = inputPanel.getAnaglyphToggled();

```

```

    if (anaglyphToggled)
    {
        //Renders anaglyph

        //Calculates camera coordinates based upon position and angle
        Vector3 cameraPos = new Vector3(cameraX - (Math.Sin(angleY) * (10 -
cameraZ )), cameraY + (Math.Sin(angleX) * (10 - cameraZ)), (10 - cameraZ));

        //Calculates the interocularDistance
        double convergenceDepth = 10;
        double interocularDistance = ((convergenceDepth / 2) +
(double)inputPanel.getAnaglyphAdjust()) / 30;

        //Calculates positions of each eye
        Vector3 leftEye = new Vector3(cameraPos.x - interocularDistance,
cameraPos.y, cameraPos.z);
        Vector3 rightEye = new Vector3(cameraPos.x + interocularDistance,
cameraPos.y, cameraPos.z);

        Vector3 leftPoint;
        Vector3 rightPoint;

        //Calculates and renders every object in the anaglyph
        foreach (Point p in points)
        {
            leftPoint = calculateAnaglyphPoint(p.rawPoint, leftEye);
            rightPoint = calculateAnaglyphPoint(p.rawPoint, rightEye);

            drawPoint(new Point(leftPoint.x, leftPoint.y, leftPoint.z,
anaglyphCyan));
            drawPoint(new Point(rightPoint.x, rightPoint.y, rightPoint.z,
anaglyphRed));
        }

        foreach (Line l in lines)
        {
            leftPoint = calculateAnaglyphPoint(l.rawFirstPoint, leftEye);
            rightPoint = calculateAnaglyphPoint(l.rawSecondPoint, leftEye);

            Line leftLine = new Line(leftPoint.x, leftPoint.y, leftPoint.z,
rightPoint.x, rightPoint.y, rightPoint.z, anaglyphCyan);

            leftPoint = calculateAnaglyphPoint(l.rawFirstPoint, rightEye);
            rightPoint = calculateAnaglyphPoint(l.rawSecondPoint, rightEye);

            Line rightLine = new Line(leftPoint.x, leftPoint.y, leftPoint.z,
rightPoint.x, rightPoint.y, rightPoint.z, anaglyphRed);

            drawLine(leftLine);
            drawLine(rightLine);
        }
        foreach (Angle a in angles)
        {
            leftPoint = calculateAnaglyphPoint(a.point1, leftEye);
            rightPoint = calculateAnaglyphPoint(a.point2, leftEye);

            Line leftLine = new Line(leftPoint.x, leftPoint.y, leftPoint.z,
rightPoint.x, rightPoint.y, rightPoint.z, anaglyphCyan);

            leftPoint = calculateAnaglyphPoint(a.point1, rightEye);
            rightPoint = calculateAnaglyphPoint(a.point2, rightEye);

```

```

        Line rightLine = new Line(leftPoint.x, leftPoint.y, leftPoint.z,
rightPoint.x, rightPoint.y, rightPoint.z, anaglyphRed);

        drawLine(leftLine);
        drawLine(rightLine);
    }
}

//Makes sure every colour is rendered
Gl.ColorMask(true, true, true, true);

//Draws every object is the anaglyph is not being rendered
if (!anaglyphToggled)
{
    foreach (Point p in points)
    {
        drawPoint(p);
    }

    foreach (Line l in lines)
    {
        drawLine(l);
    }

    foreach (Angle a in angles)
    {
        drawAngle(a);
    }
}

//Adjusts camera position based on inputs this frame
program["view_matrix"].SetValue(Matrix4.CreateTranslation(new Vector3(0,
0, cameraZ)) * Matrix4.CreateRotationX(angleX) * Matrix4.CreateRotationY(angleY) *
Matrix4.CreateRotationZ(angleZ) * Matrix4.LookAt(new Vector3(0, 0, 10), new
Vector3(cameraX, cameraY, 0), Vector3.Up));

//Swaps the render buffer to the other, pre-prepared one.
Glut.glutSwapBuffers();
}

private void closeFunc()
{
    //Tells inputPanel that GLUT and the scene are closing
    inputPanel.closeScene();
    Glut.glutLeaveMainLoop();
}

private string VertexShader = @"
in vec3 vertexPosition;
in vec3 vertexColor;

out vec3 color;

uniform mat4 projection_matrix;
uniform mat4 view_matrix;
uniform mat4 model_matrix;

void main(void)
{
    color = vertexColor;
    gl_Position = projection_matrix * view_matrix * model_matrix *
vec4(vertexPosition, 1);

```

```

}
";

    private string FragmentShader = @"
in vec3 color;

out vec4 fragment;

void main(void)
{
    fragment = vec4(color, 1);
}
";

    private void resetCamera()
    {
        //Resets the camera to its original position
        program["view_matrix"].SetValue(Matrix4.LookAt(new Vector3(0, 0, 10),
Vector3.Zero, Vector3.Up));
        angleX = 0;
        angleY = 0;
        angleZ = 0;
        cameraX = 0;
        cameraY = 0;
        cameraZ = 0;
    }

    private Vector3 getColor()
    {
        setUpUnusedColors();

        //If there are unused colours, chose a random one and return it
        if (unusedColors.Count > 0)
        {
            Random r = new Random();
            int randInt = r.Next(0, unusedColors.Count - 1);
            return unusedColors[randInt];
        }
        //If there are no unused colours, chose a random colour from the original
list and return it
        else
        {
            Random r = new Random();
            int randInt = r.Next(0, colors.Length - 1);
            return colors[randInt];
        }
    }

    private void setUpUnusedColors()
    {
        //Add every colour to unusedColors
        foreach (Vector3 c in colors)
        {
            if (!unusedColors.Contains(c))
            {
                unusedColors.Add(c);
            }
        }

        //For each object, remove the object's colour from unusedColors
        foreach (Point p in points)
        {

```

```

        if (unusedColors.Contains(p.rawColor))
        {
            unusedColors.Remove(p.rawColor);
        }
    }

    foreach (Line l in lines)
    {
        if (unusedColors.Contains(l.rawColor))
        {
            unusedColors.Remove(l.rawColor);
        }
    }

    foreach (Angle a in angles)
    {
        if (unusedColors.Contains(a.rawColor))
        {
            unusedColors.Remove(a.rawColor);
        }
    }
}

//Uses similar triangles to calculate the point's anaglyph position
private Vector3 calculateAnaglyphPoint(Vector3 point, Vector3 eye)
{
    //Find the distance between the point and the eye
    double vertexDepth = Math.Sqrt(
        Math.Pow(eye.x - point.x, 2) +
        Math.Pow(eye.y - point.y, 2) +
        Math.Pow(eye.z - point.z, 2));

    //Calculates the ratio between the point's distance to the origin, and the
    distance between the point and the eye
    double ratio = Math.Sqrt(
        Math.Pow(point.x, 2) +
        Math.Pow(point.y, 2) +
        Math.Pow(point.z, 2)) / vertexDepth;

    //Find the x coordinate of the anaglyph point
    double offsetX = point.x - eye.x;
    double parallaxX = ratio * offsetX;
    double actualX = eye.x + offsetX - parallaxX;

    //Find the y coordinate of the anaglyph point
    double offsetY = point.y - eye.y;
    double parallaxY = ratio * offsetY;
    double actualY = eye.y + offsetY - parallaxY;

    //Find the z coordinate of the anaglyph point
    double offsetZ = point.z - eye.z;
    double parallaxZ = ratio * offsetZ;
    double actualZ = eye.z + offsetZ - parallaxZ;

    return new Vector3(actualX, actualY, actualZ);
}

//Resets the model matrix
private void resetModelMatrix()
{
    program["model_matrix"].SetValue(Matrix4.CreateTranslation(Vector3.Zero));
}

```



```
private void drawPoint(Point point)
{
    resetModelMatrix();

    //Assign the necessary buffers to information about the point
    Gl.BindBufferToShaderAttribute(point.point, program, "vertexPosition");
    Gl.BindBufferToShaderAttribute(point.color, program, "vertexColor");
    Gl.BindBuffer(point.elements);

    //Draw the point
    Gl.DrawElements(BeginMode.Points, point.elements.Count,
DrawElementsType.UnsignedInt, IntPtr.Zero);
}

private void drawLine(Line line)
{
    resetModelMatrix();

    //Assign the necessary buffers to information about the line
    Gl.BindBufferToShaderAttribute(line.line, program, "vertexPosition");
    Gl.BindBufferToShaderAttribute(line.color, program, "vertexColor");
    Gl.BindBuffer(line.elements);

    //Draw the line
    Gl.DrawElements(BeginMode.Lines, line.elements.Count,
DrawElementsType.UnsignedInt, IntPtr.Zero);
}

private void drawAngle(Angle angle)
{
    resetModelMatrix();

    //Assign the necessary buffers to information about the angle
    Gl.BindBufferToShaderAttribute(angle.line, program, "vertexPosition");
    Gl.BindBufferToShaderAttribute(angle.color, program, "vertexColor");
    Gl.BindBuffer(angle.elements);

    //Draw the line
    Gl.DrawElements(BeginMode.Lines, angle.elements.Count,
DrawElementsType.UnsignedInt, IntPtr.Zero);
}
}
```