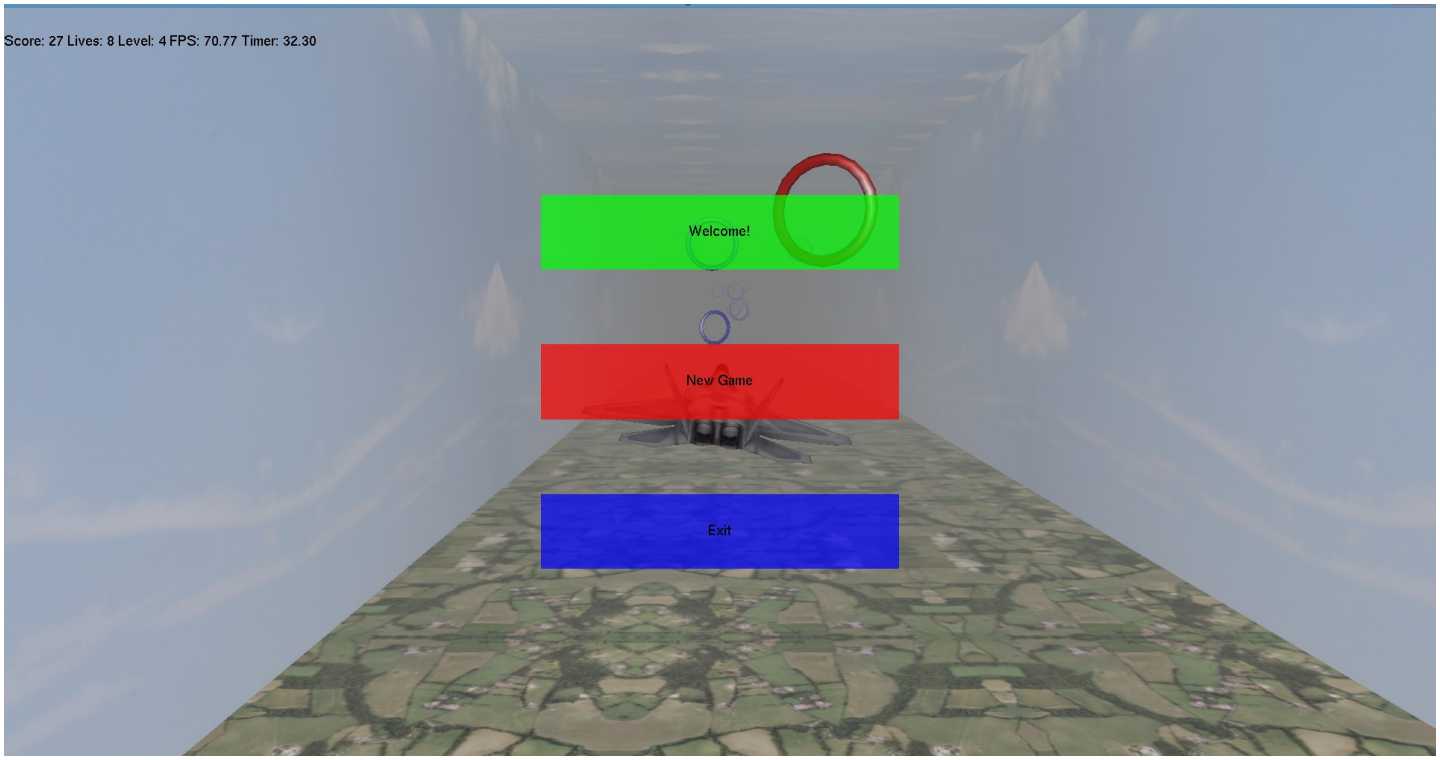
Computer Vision and Graphics Assignment

3D Flight Simulator Game Report

Joshua Tyler



1 Introduction

The outcome of this project was to implement a 3D 3rd person flight simulator, consisting of a plane moving through a series of rings using mouse and keyboard. This simulator had to be implemented using OpenGL and GLUT.

The objectives of this project break down as follows:

1. The game has to display a course layout on screen, and feature a method of gameplay (i.e. keeping track of the score/time taken etc.)
2. The project has to load an aeroplane model from a file and display it.
3. The plane movement has to be controllable with the keyboard.
4. The camera has to follow the plane.

Additional marks are also available for implementing extra features. My project implements the following additional features:

5. Multiple levels
6. Horizontal and vertical ring movement, as well as spinning rings.
7. Textured plane, side walls, floor and back walls.
8. Scene lighting and material properties of the rings.
9. Three axis plane control (pitch, yaw and roll) and realistic physics (e.g. air resistance).
10. Mouse camera and movement control.
11. Xbox 360 controller support including analogue movement, analogue engine force and vibration feedback of various intensity.
12. Various preset camera positions including cockpit view and side views.
13. Fog option to obscure distant rings.
14. Tiered menu system, including a pause menu and 'game over' menu.
15. Three difficulty settings, which adjust features such as engine force, ring size and ring spacing.
16. Autopilot demonstration mode.
17. Timer, score tracking, and fps display.
18. "Turbo" functionality, to allow the user to trigger an increase in engine speed for a limited time.
19. The option to invert the joystick controls when to Xbox controller is used.

These features are implemented through a variety of methods, some use in built OpenGL functionality and some use bespoke functions. Many of the features rely heavily upon vector maths, which is implemented through custom data types and functions.

The details of the techniques used are discussed in the following section of this report.

2 Method

2.1 Overview

When the program is first opened the main() function is called. main() performs one time actions which are necessary for setting up the program. This procedure is:

1. Levels are read from text files to arrays,.
2. Detection to determine if an Xbox controller is connected is performed.

3. GLUT is initialised to create a window.
4. Registering all of the GLUT callback functions (such as the display function, mouse function, keyboard function, reshape function and timer functions).
5. Loading the plan mesh.
6. Calculating the maximum and minimum co-ordinates of the plane mesh (for collision detection).
7. newGame() is called, this function sets the global parameters necessary for a new game.
8. initGl() is called.This initialises the OpenGL API to the parameters which we want. This consists of: setting up the camera, setting the lighting, setting up the material properties for the rings, setting up the fog and loading the textures.
9. Finally glutMainLoop() is called in order to allow GLUT to take over and callback the previously registered callback functions when necessary.

newGame() sets all the parameters necessary for starting a new game. This procedure is contained in a separate function because we may need to start a new game from different points in the code. This is because the menu system allows the user to start a new game arbitrarily. In addition newGame() can reset only the parameters necessary to move to the next level. The procedure newGame uses is:

1. The direction and velocity vectures are zeroed, along with the force and yaw angle.
2. If we want a new game (rather than moving to the next level), the number of lives, score and timer are reset.
3. The map arrays are transferred to a linked list. This is done here because the arrays contain integer co-ordinates for the rings, rather than an absolute position. Transferring the rings to a linked list means that these co-ordinates can be translated to absolute co-ordinates in the game space, and their position can be influenced by the current difficulty (so that there is less distance between rings on a harder difficulty for example.
4. The game over, pause and mouse view latch variables are reset. (mouse view latch allows the user to lock into a custom viewpoint).
5. The menu mode (whether or not the menu is displayed) and autopilot are set as necessary.
6. The wall co-ordinates are set. These depend on the ring positions.
7. The initial position of the plane is set.

The main callback functions are the timer() function and the idle() function.

The timer() function is called 10ms after it is registered in main(). Once it has run it registers another callback to itself, this means that it runs every 10ms. The timer function performs actions which need to be fun at regular intervals. The actions it perfroms are:

1. It checks to see if the plane has collided with any of the rings or any of the walls.
2. It checks if the user has died, and if so sets the variables to pause the game and display the menu on screen.
3. Processes user input. This means both toggling global states such as fog and camera angle, as well as adjusting the direction vector and engine force scaler.
4. moveRings() is then called to change the position and angle of the rings which are meant to move/rotate.
5. The new position of the plane is calculated using Newtonian kinematics.

The idle function is a lot simpler, all that is does is call the calcFps() function, which keeps track of the time elapsed since the game started, uses that to display a timer on screen, and calculates the

number of times the display is redrawn a second. Secondly, the idle function calls glutPostRedisplay() to request that the display is redrawn.

The display function performs everything necessary to draw the world on screen. The actions necessary for this are:

1. The colour and depth buffers are cleared.
2. The modelview matrix is loaded and initialised to identity. This means that we are able to start from scratch.
3. Depending on the current camera angle, one of five gluLookAt() functions is run. This sets the camera to look at the plane as if it was located at the origin.
4. The viewpoint is rotated around the y axis depending on the current yaw angle of the plane.
5. If we are currently using the mouse (or xbox joystick) to control the camera, two further rotations are performed depending on the mouse/joystick position.
6. The viewpoint is translated by the plane position.
7. The plane mesh is drawn, for this the modelview matrix is pushed onto (and later popped off) the stack. This is done to preserve it's state and draw the plane in the correct place. The drawing of the plane itself consists of translating to the necessary position. Rotating the view so that model is correctly orientated and rotated by the current yaw angle. The view is then further rotated so that the plane tilts up/down and left/right depending on the current direction vector. Textures are enabled, the correct texture is bound, and the plane is drawn.
8. The rings are then drawn. This happens using a while loop which traverses the ring linked list and draws each ring in turn. Drawing an individual ring follows this procedure:
    - The matrix is pushed onto the stack, to preserve it's state and allow us to draw each ring individually.
    - The viewpoint is translated to the ring's co-ordinates.
    - The viewpoint is rotated about the y axis by the current angle of the ring, this is what allows rings to rotate on screen (as the angle is incremented in the timer function).
    - The ring is then drawn.
    - The matrix is popped
    - Finally the colour is set to green. This is done here because the first ring is drawn in red while the rest are green.
    - The pointer to the next ring to draw is set to the next item in the list and the loop rolls around.
9. The walls are then drawn. This is done using the glDrawArrays() function, in order to draw the walls with the minimum number of function calls. The vertex and texture co-ordinates had previously been calculated in newGame(). Drawing each wall consists binding the necessary texture, telling OpenGL where the texture and vertex co-ordinate arrays are using glVertexPointer() and glTexCoordPointer() and drawing the wall using glDrawArrays().
10. The HUD text is drawn on screen, this consists of pushing both the projection and modelview matrices and loading the identity matrix into them. The string to print on screen is then set using sprint() and the string is printed onto the screen using renderText(). The matrices are then popped again.
11. The menu is then drawn on screen using drawMenu(). The exact text which is displayed differs depending on which menu is to be drawn (because the game uses a tiered menu system).
12. Fnally glFlush() is called to execute all the openGL commands we have issued.

2.2 More detail for specific techniques

Unfortunately there is not scope in this report to scratch far below the surface of how the flight simulator operates. The amount of functionality means that many functions and subroutines are necessary, in total the program features 55 custom functions. In this section I therefore intend to give further detail about just a couple of the procedures the flight simulator uses work.

2.2.1 Calculation of plane position

The flight simulator is intended to mimic real world physics as closely as possible, as such it calculates the position using vector mathematics, taking factors such as air resistance into account. All vector functions are implemented using custom C functions which I have written myself. The procedure for calculating the position is as follows:

1. The magnitude of the velocity vector is found.
2. The acceleration scaler is found by taking the current engine force and subtracting the square of the velocity magnitude multiplied by an air resistance co-efficient. This mimics real world air resistance.
3. The velocity magnitude is then updated by adding the acceleration multiplied by the delay in seconds since this function was last called.
4. The direction vector (which is set by user input) is then normalised and rotated about the y axis by the yaw angle.
5. The velocity vector is then set by multiplying the direction vector by the velocity magnitude.
6. Finally the position vector is updated by adding to it the product of the velocity and the delay.

2.2.2 Collision Detection

The collision detection with the rings is calculated using the ringCollDetect() function which is called in the timer function. When ringCollDetect() is passed the co-ordinates of a ring, and the current angle of a ring, it is able to detect if the plane is outside the ring, inside the ring, or in collision with the ring. For a ring with an angle of zero (i.e. perpendicular to the planes direction of motion), the procedure first checks if the aeroplane's furthest most point is past the ring centre minus the inner radius, it also checks if the aeroplane's closest point is closer than the planes centre point plus the inner radius. If this is true, the it means the aeroplane intersects with the ring on the distance axis.

If this first test is true, it then checks if the planes left most point is less than the centre of the ring plus total radius of the ring, and also if the right most point is greater than the centre of the ring minus the total radius of the ring. If this is true, it intersects on the left/right axis.

Finally it performs the same test for the up/down axis. The aeroplane's top point is checked for the centre plus the total radius and the lowest point is checked for the centre minus the total radius.

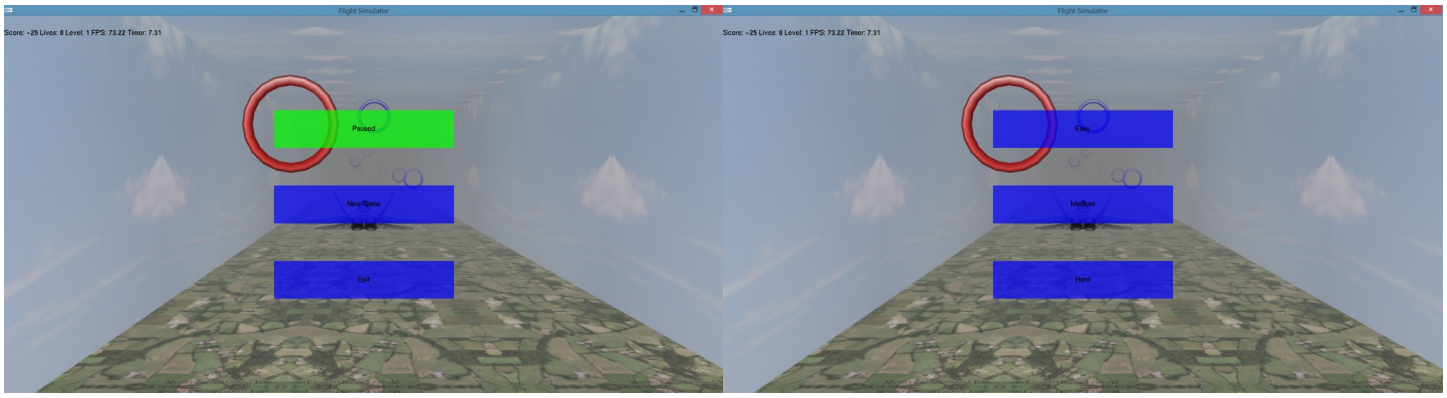
If all of these tests are true then the plane is either inside the ring or in collision with it. If this is the case then the tests are performed again, but with only the inside radius of the ring, to check whether it is inside or in collision.

In order to modify the algorithm for rings with a non-zero rotation angle, sin and cosine functions are used to modify the allowable aperture for the left/right and distance axes. For example, the left/right aperture for being inside the ring, as well as the aperture for colliding, is multiplied by the magnitude of the cosine of the angle, so that when the ring is 90° to the aeroplane's direction of motion, the allowable aperture for being inside the ring is zero.

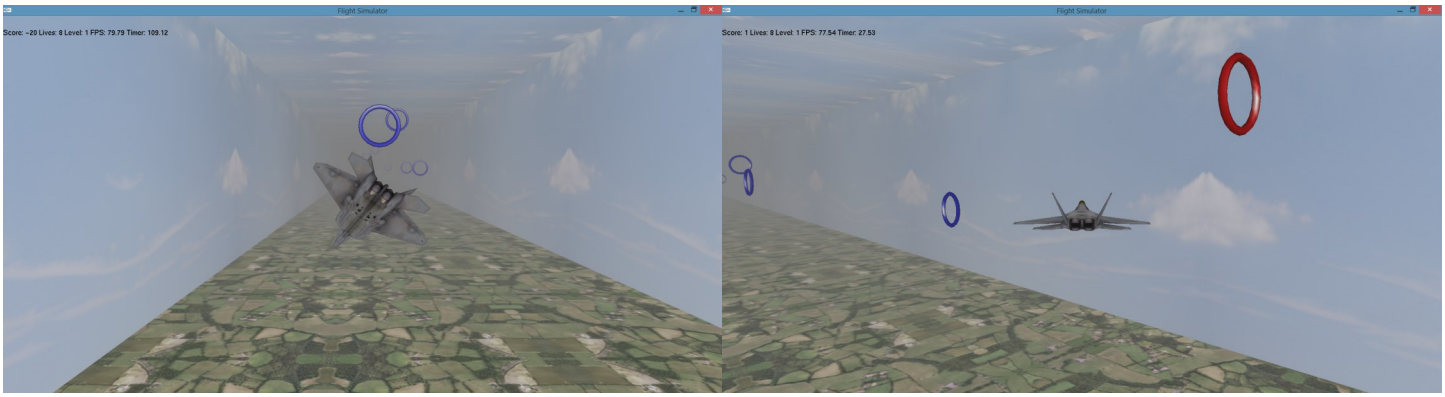
3 Results

In this section I will demonstrate some of the features of the flight simulator by means of screenshots.

The pictures below show the menu system of the game. The menu is displayed when the game is first started (overlaid upon an autopilot navigating the course) as well as when the game is paused and the player dies. The text displayed on the top menu item depends on the reason that the menu is displayed (i.e. it displays game over when the game is over, pause if it is paused etc.). If the user selects “New game”, the difficulty sub-menu is displayed. The two screenshots below show the pause menu and difficulty submenu. The menu items change colour upon mouse over.



The plane has three axes of movement and the plane model rotates to reflect this. The two pictures below show these three axes of movement. The left hand picture shows the plane pointing down on the pitch axis and right on the roll axis. The right hand picture shows the plane level on the pitch and roll axis, but pointing right on the yaw axis. Both the plane model and the camera have been rotated to reflect this. Because all three axis of motion are possible in different magnitudes simultaneously, both pictures could have been combined into one, . However it has been demonstrated here in two separate pictures for clarity.



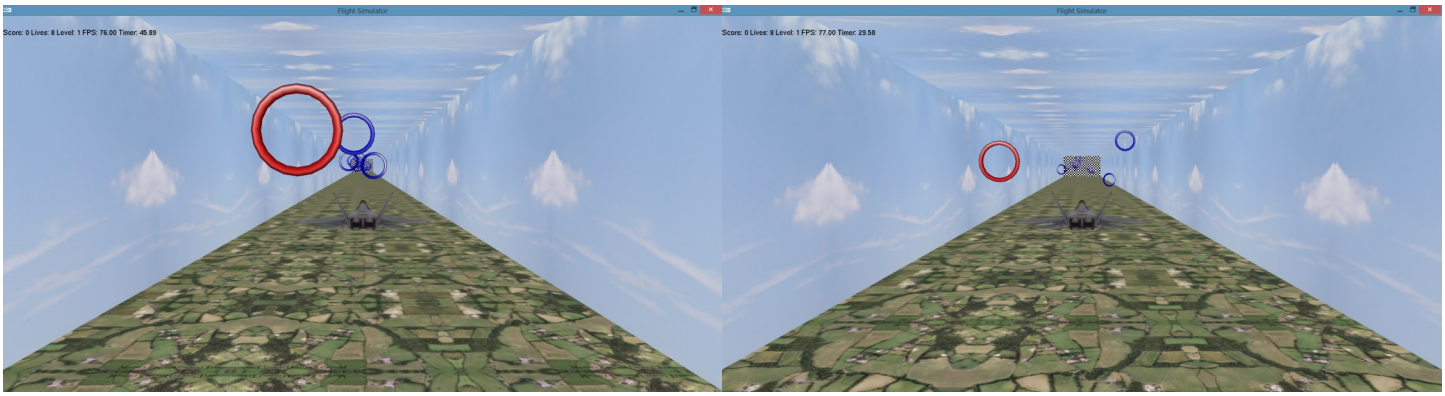
The plane course has controllable fog. The fog model I chose uses an exponential decay and it greatly improves the visual appeal of the scene. Distant rings are obscured and near rings are visible. It also helps mask the repetitions in the floor and wall textures. Of the two pictures below, the left hand picture shows fog turned on and the right hand picture shows it turned off.

The final two pictures show the differences between the two difficulty modes.

The left hand picture shows the start of level 1 on easy difficulty, in this mode the rings are spaced close together horizontally and vertically, but far apart in depth. This makes it easier to navigate the course. The rings are also very large on this difficulty level, and the plane engine is modelled as producing a low amount of force (meaning only slow speeds are possible).

In contrast to this the right hand picture shows hard mode. On this difficulty setting the rings are further apart horizontally and vertically, but closer together in depth. In addition the rings are very small and the engine produces a large amount of force (meaning the plane travels very quickly).

Medium difficulty offers a compromise between the two. Note, fog has been disabled in these screenshots for clarity.



4 Conclusion

In conclusion, I think that the flight simulator which I have produced is a very good fulfilment of the project. All the criteria for the assignment laid out in the introduction have been met, and an extensive list of extra features have been added.

I have learned a lot from this project, especially about slightly more advanced features of OpenGL such as textures and lighting. In addition I have learned a lot about capturing and processing user input from a variety of sources.

However, I think the area which I have learned most about during the course of this project is the general structure and procedures involved with computer graphics. Most of the previous programming which I have done has been almost entirely procedural, i.e. the program receives and input which it processes and then gives an output and terminates. However computer graphics projects generally run indefinitely (until terminated by the user). The processes run are procedural in themselves, but are stochastic in the sense that they are triggered by an event (a timer, user input, etc.), run, and then wait until they are called again. This style of programming takes some getting used to, and requires thought in order to structure program in a logical order.