**FLIGHT SIMULATOR COMPILED IN PYTHON VIA JIT AND OPENGL**

A Level Computer Science Project

Adrian Draber

Table of Contents

[1 Analysis 2](#_Toc140134427)

[1.1 Introduction 2](#_Toc140134428)

[1.2 Existing products 2](#_Toc140134429)

[YSFLIGHT 2](#_Toc140134430)

[1.3 Objectives 6](#_Toc140134431)

[1.4 Modules 7](#_Toc140134432)

[1.5 Implementation Research 8](#_Toc140134433)

[Transformations 8](#_Toc140134434)

[Culling 9](#_Toc140134435)

[2 Documented Design 9](#_Toc140134436)

[2.1 Prototyping 9](#_Toc140134437)

[3 References 9](#_Toc140134438)

# Analysis

## Introduction

Flight simulators are a genre of program where the user assumes control over an aircraft, such as a plane. Usually, they strive for realism, particularly because they are used to train pilots and other jobs involved in the air industry such as air traffic controllers. From a technical perspective, they present an interesting challenge due to the requirement to render three-dimensional environments and replicate the physics of real aircraft.

In this project, I aim to build a virtual flight simulator from the ground up, focusing on achieving accurate rendering of three-dimensional environments as well as replicating a very basic system of aircraft controls. The project should primarily aim to demonstrate the ability to use mathematics to project environments onto a two-dimensional screen and do so in an effective and optimised manner. Additionally, the program should also function as a learning tool for – primarily amateur or recreational – pilots, and thus possess systems such as measuring the G-force on the cockpit, or the ability to take off/land. These are important as making mistakes while flying a real plane could be potentially dangerous, so it could assist in helping amateurs recognise dangerous manoeuvres.

## Existing products

The following is an analysis of some existing flight simulators:

### YSFLIGHT

YS Flight[[1]](#endnote-1) is a Flight Simulator written in C++ using OpenGL by Soji Yamakawa, with its first release in 1999, and it has received regular updates ever since.

According to Yamakawa, the program was written for the purposes of *“(1) writing my own flight simulator, and (2) writing a software used by hundreds of thousands of people over the world. I am always so happy to receive encouraging emails about YSFLIGHT.”*. Initially created as a school project, it was turned into software to help people learn to understand aircraft physics and mechanics: *“Microsoft Flight Simulator is a great piece of work, but I also believe it is nice to have a flight simulator that everyone can casually play during the lunch break. That has been the concept of YSFLIGHT. But, I put many elements that I learned from my flight training in YSFLIGHT. I do use YSFLIGHT for practicing IFR approaches in a Cessna for myself (of course I'm not logging time for it though.) I hope YSFLIGHT serves you well for the future!”.[[2]](#endnote-2)*

Booting up YS Flight, we are treated to this menu where we choose a starting location and model of plane, among other factors:

**

*Figure 1: The screen before starting a flight.*

YS Flight is primarily designed for joystick control; however, the mouse and keyboard can be used to simulate joystick input. Upon entering the cockpit, several features of the control scheme stand out:



*Figure 2: After starting; plane is on the runway.*

* There is a virtual joystick on screen, seen on the left of the cockpit in the image. This allows the pilot to always see the status of the joystick, which is useful for mouse controls.
* There are physical flaps and spoiler indicators (bottom right of the UI). These are the controls used on real places and are responsible for turning motions. Having physical bars showing the orientation of each helps with visibility of the controls substantially.
* There is a virtual plane on the left side of the UI, showing its orientation relative to the ground - this is useful because YS Flight primarily employs a camera perspective from inside the cockpit.
* However, one drawback is that controls aren’t easily explained, and you can’t access the control menuy from inside the simulator.

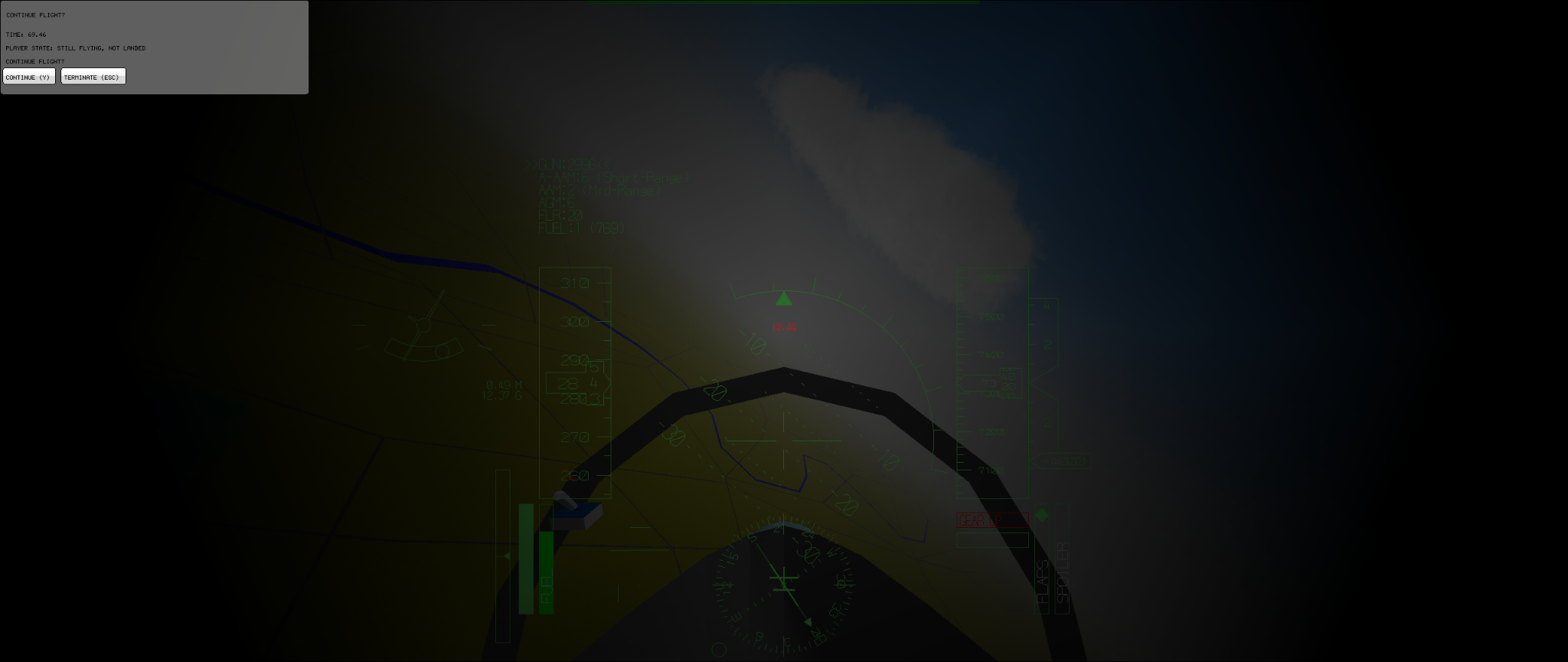
A video game screen shot of a plane

Description automatically generated

*Figure 3: The screen while in flight*

The simulator also contains several pre-built scenarios, such as trying to land a plane at a recreation of London Heathrow. After playing around with the flight controls for an hour, these are some interesting features I noticed:

* As noted in the Introduction, it is important for flight simulators to accurately convey dangerous manoeuvres to teach aspiring pilots in an environment where no risks are posed regarding their safety. YS Flight achieves this by colouring the entire screen red or black when a high G-force is reached, which functions as an easy-to-read indicator of pilot danger.
* Sideways motion of the joystick controls roll (sideways rotation of the plane) while up/down motion controls pitch (up-down rotation). Yaw, the rotation that corresponds to the direction on the map the plane is facing, is controlled with the ZXC keys (which physically control the “rudder” at the back of the plane).
* Pressing the “M” key reveals a second camera view, behind the cockpit. This is useful in the case of YS Flight since it can be used as a military dogfight simulator in addition to its main use as an amateur flight sim.
* In addition to visual indicators, various bleeps and other noises are used to communicate information to the pilot. These serve as easily recognisable warnings that recommend pilots to look at their gauges - for instance, they might indicate that fuel is not flowing to the engine during a turning manoeuvre due to the forces involved.



*Figure 4: The screen turns black as I hit 12.4G during a manoeuvre. In real life, this amount of force would kill the pilot.*

* Add more stuff here for other examples of products

## Objectives

The investigation asks for the following requirements to be satisfied:

1. Store procedurally generated 3d terrain as a mesh.
   1. Generate a Perlin noise map.
   2. From the Perlin noise map, get a set of vertices.
   3. Using the vertices, define triangles (tessellation)
2. Render the terrain on-screen according to the location of the player’s camera, and the direction in which it is pointing.
   1. Determine the vertices within the player’s current viewport.
   2. Use back-face culling technique to minimise the number of faces that must be drawn.
   3. Draw 2D triangles between the on-screen vertices.
   4. Update this process for every frame.
3. Render ‘props’ such as trees on the terrain.
4. Draw the plane model, as defined within the program.
5. Update the plane’s position using its linear velocity, angular velocity and current position.
6. Allow player input, such as the ability to shift the rudder and ailerons in order to perform turning manoeuvres.
7. Compile program to allow for execution.

## Modules

We will make use of the following modules:

* OpenGL[[3]](#endnote-3) – OpenGL is a graphics API for rendering vector graphics. This will be the primary method we will use to draw what is going onscreen. Additionally, to extend OpenGL’s functionality, we will utilise the OpenGL Utility Library (GLU) and OpenGL Utility Toolkit (GLUT)[[4]](#endnote-4). In Python, this will require installation of the pyOpenGL and pyOpenGL-accelerate modules, as well as of a GLUT library such as freeGLUT to the PC/Virtual Environment. The justification for choosing this library is its ease of use and support with most PC systems, which is what led to it being considered the “standard” library for amateur graphics projects.
* NumPy[[5]](#endnote-5) – This is a library that extends Python’s mathematical features. For instance, it allows us to use arrays, whereas base python only allows for linked lists – this is useful because arrays are significantly more efficient than linked lists. Given the nature of graphics needing to be updated many times per second (30fps is considered the “minimum standard” by many), it is crucial that we write an efficient program.
* PyGame[[6]](#endnote-6) – We will use PyGame in order to handle both our display and input of controls. While PyGame provides many other uses, such as drawing content directly onto the screen, this will be handled by OpenGL in our application.

## Implementation Research

### Transformations

OpenGL functions primarily on the principle of vertices, which have lines, triangles and squares (“quads”) drawn between them. We can move these vertices via the use of matrix transformations:

A black background with white text

Description automatically generated

*Figure 5: The transformation matrix in OpenGL.*

Here, we have a 3x3 matrix representing the points themselves, as well as a fourth row and column that allows us to perform translations. Some examples of specific uses to achieve certain matrix transformations:

A black screen with white text

Description automatically generated

A black background with white text

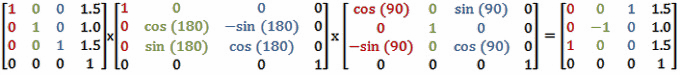
Description automatically generated

A black background with white text

Description automatically generated

*Figure 6: Example of a translation; scaling; and rotation around the x axis[[7]](#endnote-7)*

One thing to note is that matrices are non-commutative. This means that the order of operations does matter. In particular, matrices are always modified by operations behind them. The order of operations of a matrix is hence often “reversed”. We can also merge matrices into a single transformation:



*Figure 7: If performed separately, we’d consider the rotation around y to be performed first, followed by the rotation around x and finally the translation. Here we can also simplify this to a single transformation.*

While OpenGL does not require us to manually input matrices, it does still follow the rules of matrix mathematics – for instance, in OpenGL, transformations will appear to be performed in the “reversed” order compared to their definition in the code. Additionally, since matrix transforms are applied to all vertices, we need to make use of a stack so that only specified sets of vertices are available at any time to be transformed – else, every time we perform any sort of transform, everything in the world is transformed.

### Culling

*A colorful cube with black background

Description automatically generated*

*Figure 8: An image from one of my prototypes. No culling algorithm is being used; we can see that the magenta face clips through the cube, since it is drawn after the cyan face (but before the grey and yellow faces).*

The culling method used by OpenGL involves discarding faces that are “facing away” from the viewport, in a process called Face Culling. This is achieved via winding order. When triangles are drawn, the vertices are defined in an order. The order of the vertices is used to define whether a triangle’s vertices are defined as counterclockwise or clockwise. By default, any triangle with counterclockwise vertices is considered to be seen from the front, while triangles with clockwise vertices are seen from the back.

A diagram of a triangle

Description automatically generated

*Figure 9: Two triangles, drawn with clockwise and counterclockwise vertices. OpenGL considers the clockwise triangle to be seen from the back and it would be culled.[[8]](#endnote-8)*

OpenGL also allows us to change whether front or back faces are culled. For instance, we can set it so faces considered to be front facing are culled instead.

*A colorful cube with a black background

Description automatically generated*

*Figure 10: A cube with all front-facing faces culled.*

# Documented Design

## Prototyping

# References

TODO: Put this in Harvard reference format

1. ANALYSIS :

   <https://en.wikipedia.org/wiki/YSFlight>, accessed 13 July 2023 [↑](#endnote-ref-1)
2. <https://github.com/captainys/YSFLIGHT>, accessed 13 July 2023 [↑](#endnote-ref-2)
3. <https://www.opengl.org/>, accessed 13 July 2023 [↑](#endnote-ref-3)
4. <https://www.opengl.org/resources/libraries/glut/glut_downloads.php>, accessed 13 July 2023 [↑](#endnote-ref-4)
5. <https://numpy.org/>, accessed 13 July 2023 [↑](#endnote-ref-5)
6. <https://www.pygame.org/>, accessed 13 July 2023 [↑](#endnote-ref-6)
7. Images of matrices courtesy of <http://www.codinglabs.net/article_world_view_projection_matrix.aspx>, written by Marco Alami, accessed 13 July 2023 [↑](#endnote-ref-7)
8. Image courtesy of <https://www.khronos.org/opengl/wiki/Face_Culling>, accessed 13 July 2023 [↑](#endnote-ref-8)