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

A Level Computer Science Project

Adrian draber

# Table of Contents

[Analysis 3](#_Toc137110323)

[Introduction 3](#_Toc137110324)

UPDATE ONCE FINISHED

# 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:

### Microsoft AirSim

Microsoft AirSim is an open source flight simulator framework

## 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.

# References