Software Requirements Specification

For

OS: Flight Simulator

Version 1.0

Keshav Bharat (1RV16CS070)

Kshitij Tiwari (1RV16CS075)

RVCE

20th August, 2018

Table of Contents

Table of Contents ii

Revision History ii

1. Introduction 3

1.1 Purpose 3

1.2 Document Conventions 3

1.3 Intended Audience and Reading Suggestions 3

1.4 Product Scope 3

1.5 References 3

2. Overall Description 3

2.1 Product Perspective 3

2.2 Product Functions 4

2.3 User Classes and Characteristics 4

2.4 Design and Implementation Constraints 4

2.5 User Documentation 4

2.6 Assumptions and Dependencies 4

3. External Interface Requirements 5

3.1 User Interface 5

3.2 Hardware Interface 5

3.3 Software Interfaces 5

4. Functional Requirements 5

4.1 Description and Priority 5

4.2 Stimulus/ Response Sequences 5

4.3 Functional Requirements 6

5. Non-Functional Design 7

5.1 Performance Requirements 7

5.2 Security Requirements 7

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Purpose

The goal of the **FlightGear** project is to create a sophisticated and open flight simulator framework for use in research or academic environments, pilot training, as an industry engineering tool, for DIY-ers to pursue their favorite interesting flight simulation idea, and last but certainly not least as a fun, realistic, and challenging desktop flight simulator.

## Document Conventions

Open Source model of software has been assumed here. The contributions of the project inherit from the existing code available.

## Intended Audience and Reading Suggestions

Target Audience is developers familiar with Open Source flow of programming.

## Product Scope

A big problem with commercial PC flight simulators is their closed source nature and lack of extensibility. There are so many people across the world with great ideas for enhancing the currently available simulators who have the ability to write code, and who have a desire to learn and contribute. Many people involved in education and research could use a flight simulator frame work on which to build their own projects.

## References

1. http://wiki.flightgear.org/Howto:Start\_core\_development#Hacking\_the\_C.2B.2B\_code
2. http://wiki.flightgear.org/Howto:Add\_new\_fgcommands\_to\_FlightGear
3. http://wiki.flightgear.org/Howto:Extending\_Canvas\_to\_support\_rendering\_3D\_models

# Overall Description

This section will give an overview of the whole system. The system will be explained in its context to show how the system interacts with other systems and introduce its basic functionality. It will also describe what type of stakeholders will use the system and what functionality is available for each type. Finally, the constraints and assumptions for the system will be presented.

## Product Perspective

There is considerable lag in frame rates due to garbage collection occurring during runtime. Further, it is desirable to improve the 3d rendering capabilities of the existing software.

The HUD in the aircraft is currently displayed by means of raw OpenGL calls. We aim to replace these with commands issued from a custom API.

This will make it much easier for other developers to add features and work on improving the efficiency of the rendering.

## Product Functions

The user will be able to query the HUD either within/ exterior to the cockpit. For validation purposes, the API may query OpenGL to verify the correctness of the spatial orientation, altitude, and heading.

## User Classes and Characteristics

There are two types of users interacting with the system: casual users and developers. Each of these two types of users has different use of the system so each of them has their own requirements.  
Casual Users are primarily players who use the simulator for various purposes. The system should provide them with an easy way of accessing the HUD, and performing aesthetic adjustments such as thickness, colour, etc.

Developers are primarily involved in extending the functionality of existing software. The API code should be well documented, easy to understand, and extensible by some means described in the documentation.

## Design and Implementation Constraints

Compatibility with C++ is a constraint for the application. Since both OpenGL and Canvas are heavily reliant on C++, their functionality will be unavailable without C++.

Time taken to render is a constraint for making the system usable to end users. There should be minimum delay in rendering the panel.

Both developer and end user components will be constrained by available RAM space, RAM speed, and disk fetch times. A pixel will have to first be fetched from the disk into the RAM, and then have to be transformed in the RAM before being rendered.

## User Documentation

All official documents will be applicable for the system as the core modules are not being disturbed.

## Assumptions and Dependencies

Main Dependencies are:

1. CMake
2. Git
3. MSVC
4. Windows SDK

# External Interface Requirements

## User Interface

The user interface is designed in C++ for Windows platforms. Development of a Qt based graphical interface is being actively developed for a multi-platform GUI interface (MacOS, Windows, Linux).

## Hardware Interface

The device types supported by the simulator are:

* Joystick
* Mouse
* Standard Flight Pedals

Data will be transferred serially to the hardware using protocols such as USB.

## Software Interfaces

FlightGear depends on SimGear, while SimGear depends on some 3rd party libraries such as OpenSceneGraph (for rendering), plib (utility functions, joystick support, GUI etc), OpenAL (sound) and others like boost.

OpenSceneGraph is based on the OpenGL APIs for cross platform functionality.

# Functional Requirements

This section includes the requirements that specify all the fundamental actions of the software system for the user classes.

## Description and Priority

The system must maintain a database of airplanes and their individual properties. This has a high priority as the realism is very important for any flight simulator.

## Stimulus/ Response Sequences

* End user chooses FGRUN, Qt, or command line based launcher.
* Launch Wizard opens, and configuration options are loaded.
* User chooses configuration options, and launches simulation.
* Simulation environment loads.

End User only :

* Press the key for HUD
* Validate the HUD matches the internal cockpit instruments.
* Use the HUD as a convenient replacement for conventional cockpit controls.

Developers only :

* Determine the component of HUD desired to be modified.
* Call API functions to modify the HUD properties.
* Test the changes.
* Commit the changes.

## Functional Requirements

The various use cases and the corresponding requirements are detailed in this section.

### End Users

#### Git based versioning

Given that the user has cloned official repository using git, the system should git based versioning to ensure that the user can easily update to the latest versions.

#### Screen Tearing Control

Given that the application has been installed, and the displays support synchronization such as freesync or gsync, support for these techniques should be available to the user.

#### Joystick Support

The system should provide user with facility to calibrate a joystick to provide inputs to the aircraft. Post calibration, the user will be ab;e to view changes in the HUD corresponding to joystick position.

### Developers

#### Abstract OpenGL

The system should provide the developer with an abstraction of OpenGL. The API calls alone must be required for rendering the HUD.

#### Git Based Versioning

Given that the developer has made a change to the rendering code, and given that it is approved, the system must allow for changes to be pushed via git versioning system.

# Non-Functional Design

This section describes the requirements which are not part of the core functionality of the system, but have to be met nevertheless for QoS.

## Performance Requirements

In the Nasal scripting language, memory is not manually allocated and freed, rather it is handled by the VM. The VM automatically allocates memory from “memory pools”, which are periodically checked for their validity, and invalid references are removed. This process is called Garbage Collection. Garbage Collection has an adverse effect on framerates, causing tearing. We require that the framerate is affected minimally by the Garbage Collector under heavy loads, such as complex scripts.

## Security Requirements

The system has a capability for online DNS based multiplayer mode. This consists of several clients connecting to a single server. It is desirable to ensure end to end security in each communication channel from the channel to the server. This can be implemented by introducing an encryption scheme at each end.