Department of Electrical and Electronic Engineering
Part IV Project
3D Interface for the Unmanned Aerial Vehicle

by Chris Mills, Brendan Cervin(Project Partner)
Supervisor: Burkhard Wuensche

September 13, 2004

UAV Project Abstract

An essential part of the future strategy for the New Zealand Defence Force is the commissioning of an Unmanned Aerial Vehicle (UAV) for unit support in the field. Phil Strong, a Defence Technology Agency scientist, has been developing a UAV that is nearing completion.

The aims of the project are to:

- To use a 3D environment to simulate the UAVs flight path.
- To integrate Joint Geospatial Facility (JGSF) terrain data into the 3D environment to simulate the terrain information. As part of this, the ability to output the generated maps into bitmap format for different applications would be required.
- To integrate the interface into the Virtual Maritime Systems Architecture in order to conduct Virtual Battleground scenarios.

After determining the requirements for the project, it was necessary to under take research into the 3D environment to use, which ultimately was a flight simulator. The choice of flight simulator was between FlightGear and Microsoft Flight Simulator 2002. FlightGear was the final choice as it has standard network communication and has a variety of tools for use for the 2nd aim of the project.

The design of the application which fulfilled the 1st aim consisted of two major classes, the UAV Poller and the FG (short for FlightGear) Interface. The Poller manages the gathering of flight information from the UAV and the FG Interface uses this data to send to FlightGear in the required format. This part of the project was implemented.

The implementation of the second part is nearing completion. TerraGear, tools and utilities used to construct scenery for FlightGear, has been compiled on Linux. Data in the correct format is now been requested from JGSF to test the utilities and create the required terrain and scenery.

Future work includes the completion of the 2nd aim of the project and then proceeding on to the 3rd aim of the project. The remaining work will be completed either by DTA or Chris Mills, as he is employed by the Navy.

Declaration of Originality

This report is my own unaided work and was not copied from nor written in collaboration with any other person.

Signed:_____

Acknowledgments

The author would like to thank Matt Hopkins for his help and endless enthusiasm, Sally Garret and the JGSF staff for the terrain data, Phil Strong for the UAV Simulator (HawkSim), Graham Macferson for his help with Microsoft Flight SImulator, Noah Coad for the .NET Serial COM (MSComm.ocx) Wrapper, Timothy J. Krel for his MFC wrapper classes, the developers associated with FlightGear and TerraGear projects, Wayne Ewington from Microsoft for organizing the provision of Microsoft Flight Simulator 2002 and finally our supervisor, Burkhard Wuensche, for his guidance and support.

Contents

List of Figures

1 Introduction

1.1 Background

An essential part of the future strategy of the New Zealand Defence Force (NZDF) is to commission an Unmanned Aerial Vehicle (UAV) for support of units in the field. The UAV, flown by an autopilot and remotely directed by an on-site operator, will assist with unit reconnaissance.

The Defence Technology Agency (DTA) provides research and development facilities to the NZDF and the Ministry of Defence [1]. They are located at the Naval Base Philomel, situated in Devonport, Auckland. The DTA has been developing the UAV and it is nearing completion. Phil Strong, a scientist at DTA, is heading the team developing the UAV.

At the moment the operator directs the UAV using a 2D map display. This 2D display uses a road map with contour lines and features for reference. The UAV is represented by a cross hair over this map. The operator simply selects a point on the map and the UAV autopilot steers the aircraft in that direction. This display is insufficient as it does not provide realistic terrain data that can be viewed easily. This limits the operators ability to judge the surrounding terrain and landmarks when directing the UAV.

A 3D environment is required to display a realistic representation of the UAV and its surrounding terrain. This would aid the operator in determining what direction to send the UAV and the correct altitude. He or she would be able to use it as reference when directing the UAV and avoid obstacles caused by terrain features and landmarks.

Accurate terrain data needs to be integrated in to the 3D environment in order to provide a true representation. The Defence Force uses its Joint Geospatial Support Facility (JGSF) for its terrain data requirements. Using this terrain data, which is of a very high resolution, to generate terrain and scenery in to the environment would provide accurate representation. Figure 1.1 demonstrates JGSF's terrain data and its capabilities.

The DTA is also actively involved with the development of the Virtual Maritime System Architecture (VMSA) [2]. There are five countries that are taking part in this development; New Zealand, Australia, Canada, the United Kingdom and the United States of America. The VMSA is an architecture that can be used to build virtual units from separate virtual systems called federates. A federate represents one system, which may be a weapons system, radar system, mechanical system or another system specified by VMSA. The federate allows for the simulation of that system. The units constructed are used in Virtual Battleground Scenarios, which are mock battles that take place in a networked computer environment.

The DTA is developing aircraft units for the VMSA. As part of this development they require a display that will be used by these aircraft units to view other units in a Virtual Battleground Scenario. Ideally the display would be viewed from the cockpit of these units. Graham Macferson, also a DTA scientist, is developing the aircraft units and is investigating the display to use in conjunction with those units.

The 3D environment required for the UAV could also be used for the display of VMSA aircraft units. The requirement is similar to the UAV's display, but

Figure 1: Elevation data overlaid with a road and contour map

instead of showing real-time data it will use virtual data from a Virtual Battleground Scenario. It would also be required to display other units from the scenario.

In both these cases the environment would provide a "Command & Control" overview. Commanders in charge of the UAV or a force within a scenario could utilise the environment to augment their other information systems.

1.2 Aims of the Project

Using the above requirements and information, three aims have been developed:

1. Develop a 2D / 3D environment to simulate the UAV and its flight path,

2. Use JGSF terrain data in the environment in order to create scenery with correct elevation and include features such as buildings, roads and waterways. This aim includes the requirement that a 2D bitmap of the created scenery can be generated, and

3. Integrate the Virtual Maritime System Architecture to display an aircraft unit and other units within a Virtual Battleground Scenario.

The 3D environment would need to use UAV flight data in order to show its correct position, orientation and location. It needs to be a 3D display so that the UAV operator and others interested in the display can easily recognise features within the environment. Freely movable cameras would be the best way to display both the UAV and its surrounding terrain.

The integration of JGSF data in to the environment will provide an accurate representation of the surrounding terrain for the UAV. This will aid the operator in making decisions about where he will direct the UAV next. The other requirement is that bitmaps can be generated using the created terrain. These 2D bitmaps will be used in 2D displays of the UAV (i.e. for instance in the original 2D map interface).

Integrating the VMSA in to this environment will fulfill the requirement to display an aircraft unit and other units within a scenario. Using the same 3D environment cuts down on time spent on developing two individual displays.

1.3 Project Method and Scope

The method of investigation will be to conduct primarily research activities in the first part of the project. This will allow time to investigate the building of or the adaptation of an existing 3D environment. Most of our research is Internet based as we will be looking at software development and applications. The scope of investigation will be, but not limited to, 3D graphics, simulations, terrain generation and investigation in to integration between architectures, one of which being VMSA.

1.4 Report Overview

This report covers the initial research of the project and the investigation in to applications, specifically flight simulators, to aid in the display. It then outlines a possible project implementation using flight simulators. Research of two flight simulators, Microsoft Flight Simulator 2002 and FlightGear, is discussed. Next is the application of the flight simulator for the first aim. The second aim covers terrain formats and the development of scenery generation tools. The use of them is covered next, followed by the 3rd aim of the project. The conclusion sums up the main findings of this project.

2 Initial Research

The three aims focus on developing or using an existing 3D environment for display. Initial research was dedicated toward investigating either developing a 3D environment or using an existing application that would satisfy our requirements.

2.1 3D Environments and Applications

The work required for building a 3D environment for our requirements is extensive. Complex utilities and libraries would need to be built to render a terrain and then manage a camera to move through it. Also the terrain would need to be built from terrain data from Joint Geospatial Support Facility (JGSF) and unit integration would be required for the VMSA.

After investigating these issues it was decided that it would not be viable to develop a 3D environment from 'scratch'. Instead applications that provide most of the functionality, which do exist, would be investigated.

There are several types of application available for use. The first type of application that was looked at were flight simulators. Flight simulators were thought of initially as they display aircraft in 3D. Most have scenery generated from available terrain and image data, providing a realistic flight experience. They include flight models, scenery, and use positional information based on latitude and longitude. Also most simulators have a range of tools supporting the customisation of their features. This allows us to modify the different features of the flight simulator. There is also a wide range of flight simulators to use.

The decision was to use flight simulators as they fit the requirements of the project and are specialised for our needs.

2.2 Initial Design

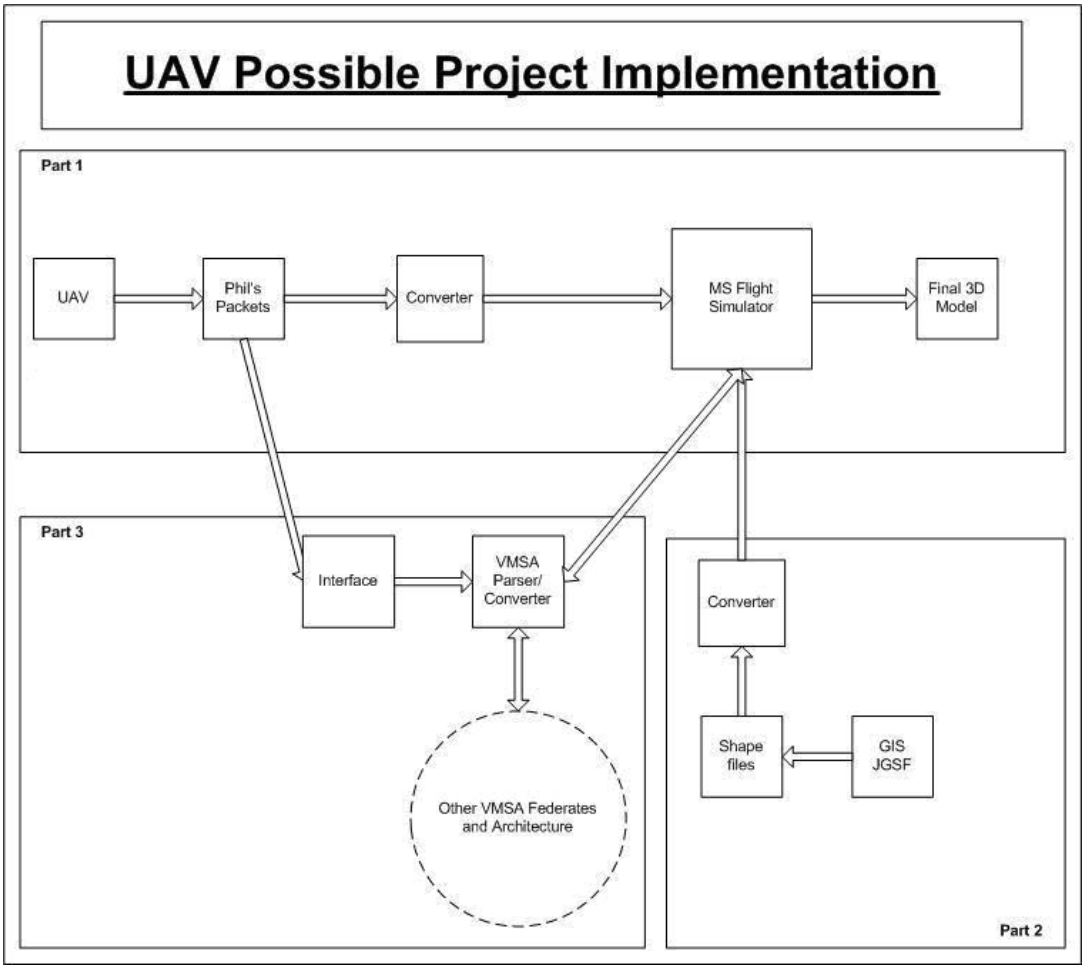
Now that a decision was made on the type of application that would be used, an initial project implementation was designed. This is displayed (Figure 2) and then is discussed below.

The first part of the project is to implement the UAV in the 3D environment, the flight simulator. This will require the delivery of packet information from the UAV to the flight simulator which will then update flight information. This requires a converter that will translate the packets in to a compatible format for the flight simulator.

The second part is to use terrain data to make suitable scenery for the flight simulator. Another converter is required to change terrain data from JGSF in to that format.

The third part is to provide a plug-in into the VMSA data which can pass on virtual unit information to the flight simulator for display. The plug-in would allow unit details to be converted to a format understood by the flight simulator and then display them within the simulator's 3D environment.

Figure 2: Possible Project Implementation



3 Flight Simulator Research

3.1 Flight Simulator Requirements

The first part of the implementation was to choose a suitable flight simulator to use. Taking the above implementation into account we developed three requirements for the flight simulator:

1. Must be able to use remote data to update an aircraft within the simulator display.

2. Provide tools that can generate terrain and scenery for the simulator from JGSF data.

3. Have a suitable utility to allow other units to be displayed within the display.

The first requirement is that the flight simulator can use data outside of the application to update an aircraft within the simulator. The aim is to use UAV data to update this type of aircraft information. This will change the position, orientation and location of the aircraft and provide a realistic representation of the UAV.

The second requirement is that tools are available to convert JGSF data in to the simulators scenery. Because JGSF uses standard format terrain data, the tool needs to support this type of data. The tools should also allow the placement of important features like roads and waterways also using publicly available data.

The third requirement is that unit position and information can be shown within the simulator. This equates to providing multi player support and modelling tools which provides an interface for unit insertion and distinct models for those units respectively.

There are two popular flight simulators that do provide for the requirements above which were investigated further. The first is Microsoft Flight Simulator 2002 (MSFS), a proprietary flight simulator with a large user base and developer support. The second is FlightGear, an open-source alternative to MSFS, which has a medium user base and developer support.

3.2 Flight Simulators Overview

3.2.1 Microsoft Flight Simulator 2002

Previous Work Previous work that had been conducted with Microsoft Flight Simulator 2002 (MSFS) prior to this project was Graham Macferson's plug-in to the VMSA. He has already developed a plug-in to the VMSA that displayed units from the architecture in to the display. Mr. Macferson was able to pilot an aircraft over these units, which were displayed as 3D blocks.

Project Investigation MSFS has a number of features [3] available:

Aircraft Models there are many different aircraft models available. Also it provides a tool, gMax, which allows the building of aircraft in conjunction with the aircraft editor provided by Microsoft.

AutoGen Scenery Microsoft technology that allows smooth blending in the scenery.

Flight Analysis maps and graphs that detail flight information.

Moving map and GPS positioning system the GPS provides point to point navigation with constant position updates on a moving map, plus ground speed, course to next waypoint, and other information.

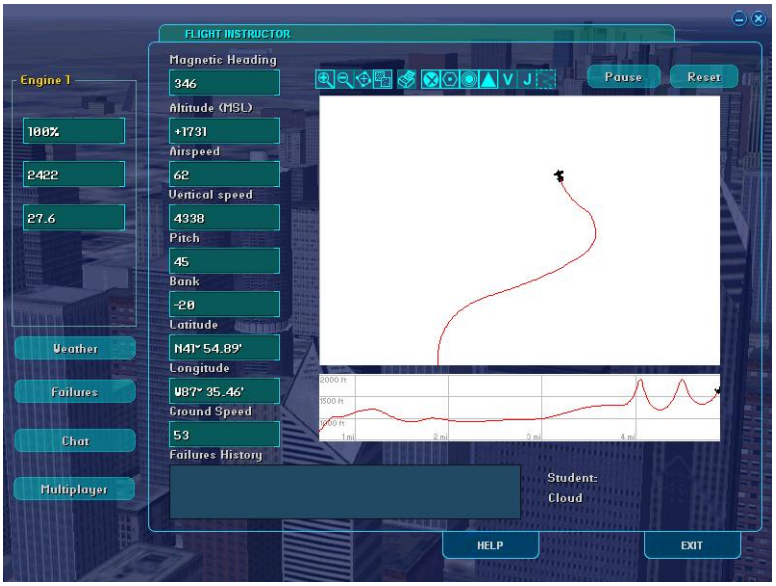
Multiplayer Capability allows flight with other players running MSFS. Up to 16 players allowed.

The cost of MSFS is in the range of NZ$50-100, depending on what version is purchased. It is available from most stores selling computer games and products. It has a large support base, as it is the most popular flight simulator on the market.

For the first requirement, MSFS should be able to display a remote aircraft in its display. Microsoft can view other planes by using the Instructor Session screen (Figure 3). This screen is used by a flight instructor within a game to view his or her "students" flight information. It displays altitude, heading, location using latitude and longitude, air and vertical speed, and orientation of the aircraft. It also displays two 2D moving maps. One is a top-down arrangement which shows

a history of the aircrafts change in heading. The other displays a line graph showing the aircrafts change in altitude over time.

Figure 3: MSFS Instructor Session Interface



This display can be used to convey information about the UAV, if UAV data is used to update it. Its most lacking feature however is that there is no 3D display of the aircraft. This is also the only way to view a remote aircraft in MSFS.

The second requirement is that tools are provided to generate scenery. Microsoft already has scenery that is of a high quality. It provides higher resolution data for those areas that are popular with its users; the Grand Canyon and Chicago areas are good examples. It uses the BGL format, specific to MSFS, to store its scenery files [4].

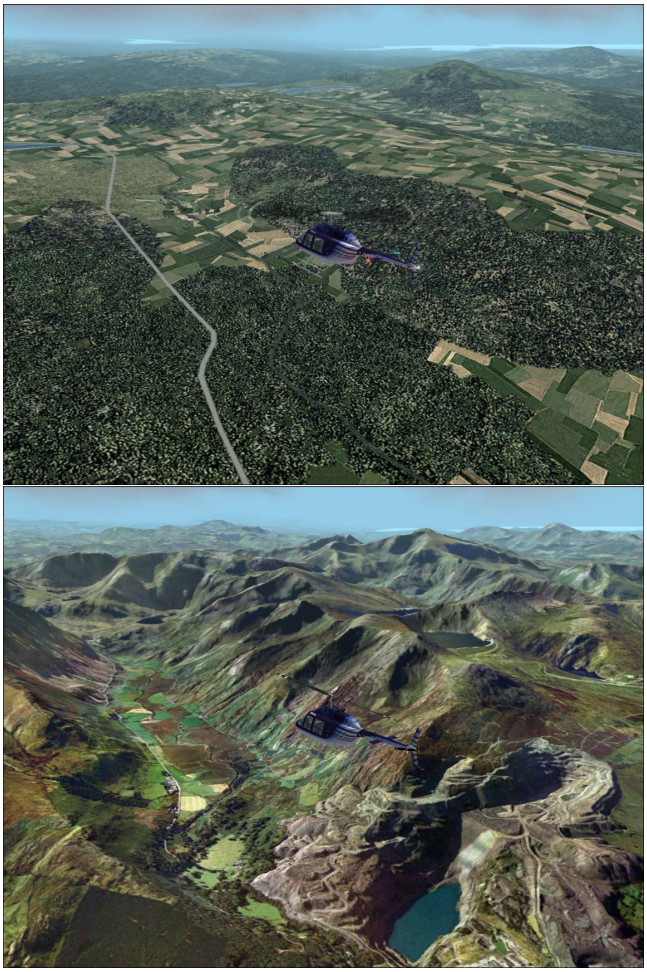
There are a number of tools, from Microsoft and third parties that can generate scenery and the terrain mesh in BGL format. Microsoft provides tools with its Terrain Software Development Kit (SDK) [5]. These tools generate the terrain mesh using terrain data from United States Geological Survey (USGS).

Other applications and companies that produce terrain mesh and scenery are FSGenesis [6] and VFR Terrain and Photographic Scenery [7]. FSGenesis sells scenery for MSFS. Their terrain is in a variety of resolutions, ranging from 100m to 10m (refer to section 5.1.1 for discussion on terrain resolution).

VFR Terrain and Photographic Scenery are applications that can be purchased to build BGL files as well. The first can use high resolution data to build accurate terrain meshes and the second is used to overlay overhead pictures on to that terrain. Below are sample pictures obtained from the VFR website (Figure 4) that shows terrain before and after using these tools. These two tools are limited to use in England and Wales however.

The third requirement of the simulator is that it has multiplayer functionality. MSFS provides this over either TCP/IP or IPX. There is a limit to the number of players which is set at 16.

Figure 4: Comparison between default MSFS scenery and VFR scenery



3.2.2 FlightGear

FlightGear has similar features [8] to MSFS, including:

Flight Dynamic Model (FDM) the model that FlightGear uses to store and work with aircraft properties. There are currently three types of FDM able to be used within FlightGear.

Extensive and Accurate World Scenery Database uses high resolution data, specifically SRTM data and USGS DEM data (refer to Glossary (pg. 31).

Accurate Sky Model provides correct positioning of sun, moon and stars depending on where the simulator is being used.

Aircraft modelling ability to create and alter new and existing aircraft.

Open-Source able to investigate internal properties of the simulator and alter it to individual purposes.

Networking Properties there is a number of networking options allowing FlightGear to communicate with external software, including other instances of FlightGear. A multi player protocol is available for using FlightGear on a local network.

A number of Open-Source utilities FlightGear is supported by a number of open-source tools.

FlightGear costs nothing to use. It is freely available from the Internet and consists of two downloads, the binaries and the terrain data, which together are approximately 200Mb in size. There is developer support for FlightGear and mailing-lists for any questions.

For the first requirement, FlightGear must be able to display a remote aircraft in its environment. FlightGear can use an external Flight Dynamic Model (FDM) to display other aircraft within its simulation. This can be used to view other aircraft being piloted on other instances of FlightGear over a network.

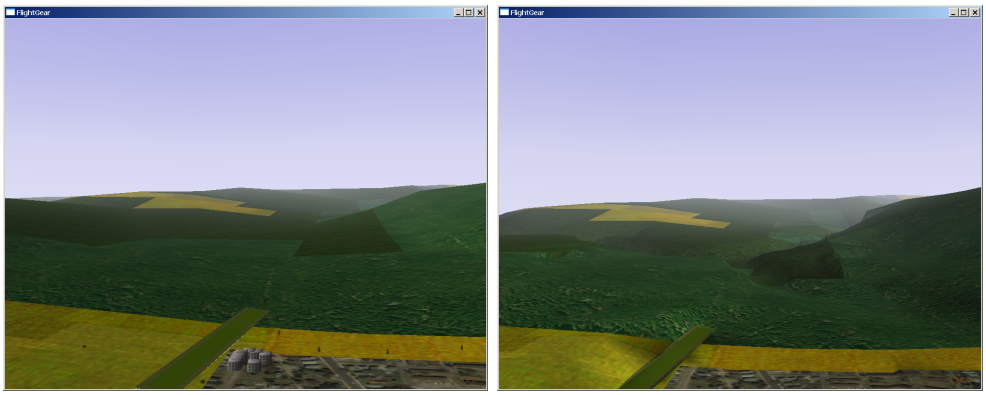
The viewer of the aircraft can use the 3D camera built in to FlightGear to view an external aircraft from within the cockpit or from outside the aircraft. It allows for full movement of the camera, so it is possible to view directly behind the aircraft or off to one side of the aircraft. This is supported in internal (i.e. cockpit) and external (i.e. outside the aircraft) views.

The second requirement is to generate FlightGear scenery using provided data from JGSF. FlightGear scenery generation is supported by three open-source projects; TerraGear [9], FlightGear Scenery Designer (FGSD) [10] and Atlas [11].

TerraGear is a collection of libraries and tools that provide support for FlightGear scenery generation. It uses a number of formats available in the Geographic Information System (refer to section 5.1.1).

FGSD is a tool that is used to enhance FlightGear scenery by providing image overlays and altering elevation data using contour maps. It can also provide 3D object placement as well. Below are two images comparing the terrain before and after using FGSD (Figure 5). Below those images is another image of terrain textured with aerial photographs that create a photo-realistic scenery of an airport (Figure 6).

Figure 5: Comparison between default FlightGear scenery and FGSD scenery



Atlas is a utility to convert FlightGear terrain into bitmaps. It also has the useful feature of displaying a cross-hair over these images indicating the location of an aircraft in a running FlightGear instance.

The third requirement is that it has multiplayer ability. FlightGear has some multiplayer ability but the development is still in the alpha stage. The current multiplayer has been described as:

Figure 6: Photo-realistic Airport Scenery

the jittery, unstable presence of the 'other plane' [12]

under certain conditions.

3.2.3 Flight Simulator Comparison

A comparison between FlightGear and MSFS in terms of the three requirements:

Display an aircraft using remote data FlightGear uses a 3D display with a directional 3D camera which allows complete view around and from inside the aircraft. MSFS uses a 2D interface that displays the aircrafts flight information.

Generate terrain using JGSF data MSFS is supported by the Terrain SDK and third party tools which can generate realistic terrain. FlightGear is supported by TerraGear, FGSD and Atlas which also produce realistic scenery and provide other useful features.

Must have multiplayer support MSFS 2002 has multiplayer support for 16 players. FlightGear multiplayer support is still under development, but it does provide limited multiplayer that doesnt work correctly under certain conditions.

The decision was made to use FlightGear, based on the reason that it could display remote aircraft in 3D, thereby fulfilling our first requirement. It also provides scenery and while it doesn't have a fully working multiplayer capability, it is believed it will provide a means to do so in the near future.

4 1st Aim: Develop a 3D Environment to Simulate the UAV and its Flight Path

The flight simulator that will be used to implement this aim is FlightGear. Now there is a need to provide a converter or an interface between the UAV and FlightGear. This will provide data from the UAV for FlightGear in order to display it in the simulation. From now on this package will be called the UAV Interface.

4.1 Research

The communication from the UAV and to FlightGear needs to be researched to fully understand the requirements of the UAV Interface.

The UAV communicates wirelessly using packets for both sending and receiving. The uplink packets request flight information from or direct the UAV, and the downlink packets return flight information about the UAV. They are specified in two documents produced by the DTA [13][14]. Since these documents are awaiting classification, these details can not be disclosed.

FlightGear, as mentioned before, uses an external Flight Dynamic Model (FDM). Using this and sending it across the network, it is possible to display a remote aircraft in another instance of FlightGear. Using this property it is possible to update the external FDM with UAV data and send it to the other instance of FlightGear.

4.2 Design decisions

For the implementation, the decision was made to use C# where possible in order to speed development and because it is similar in syntax to Java, a language that the project team is familiar with. There may be some requirement to code in C++ as well, as FlightGear is developed primarily in that language. However, every opportunity will be taken to avoid this, as the projects teams strength is not in C++.

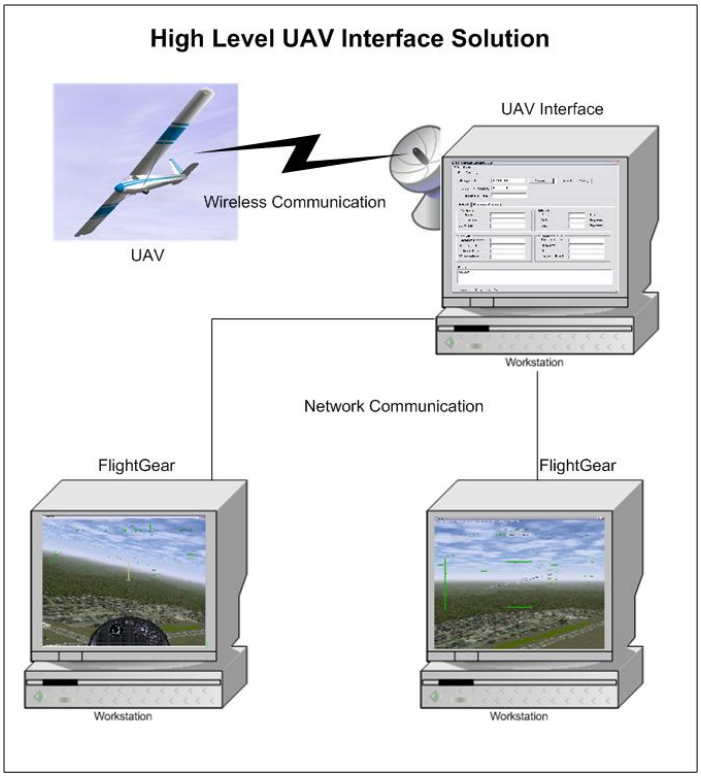
HawkSim is a simulator for the UAV developed at the DTA. It can be controlled via uplink packets (refer to section 4.1) and can respond to requests for flight information. It automatically starts with the UAV over Hobsonville Airbase on the North Shore. Whilst in development, the HawkSim will be used as a substitute for real UAV data.

4.3 Solution and Implementation

Below is the high-level designed solution (Figure 7). The UAV is displayed communicating to the workstation. This workstation is in control of the communication of this system. It is responsible for polling the UAV for its data and then storing it on its return. Once enough data is collated it is sent out to the other workstations. The format in which it is sent is compatible with FlightGear. The other workstations are running instances of FlightGear that receive the data and produce an image of the UAV flying through simulated terrain.

The advantage of sending it over the network to individual workstations is that different displays can be set up. In the diagram there are two different

Figure 7: High Level Solution



views for the UAV, using the same orientation and position information. This is useful when different views are required on the network. For example, the UAV operator may require a different display than the Commander in charge of the UAV.

This solution is converted in to a high level implementation diagram below (Figure 8). It shows the various components and the interaction between them.

4.3.1 Serial Port

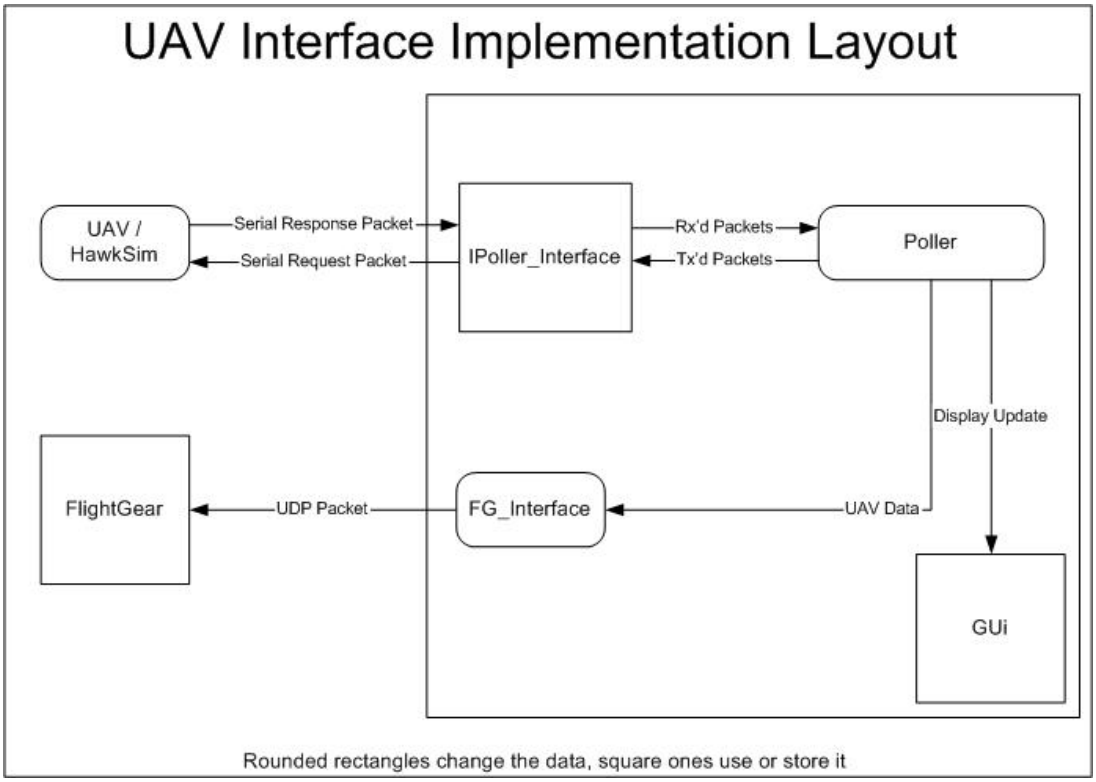
The UAV or HawkSim receives and sends UAV specific packets (refer to section 4.1). The received, or uplink, packets are generated by the Poller through the IPoller_Interface. The IPoller_Interface uses serial ports to communicate to the UAV / HawkSim. This is to simulate sending this information wirelessly over an aerial connected to the serial port.

The serial port code was developed by Timothy J. Krel [15]. This implementation of socket classes uses C# wrapper classes around Microsoft Foundation Classes (MFC). This was implemented to support legacy ports for the .NET framework. These wrapper classes provide .NET framework code (compatible with C#) to use the MFC classes.

Originally, alternative code was being used that was developed by Noah Coad [16]. This implementation used an ActiveX control embedded in a hidden Windows Form in order to access the methods within the control. This was removed from the solution in favour of Mr. Krels code for the following reasons:

- The installation of the Krel's code was simpler than Coads.

Figure 8: High Level Implementation



- Licensing requirements for Coad's code including owning the Visual Studio C++ Version 6 Compiler license which we didn't have.
- Krel's code was more modular in its approach which was a design preference.
- Krel's code included complete configuration panels for the serial port, making it easy to configure the serial port.

4.3.2 IPoller_Interface

The IPoller_Interface allows communication to a variety of devices. It describes simple methods that other communication packages implement (for instance, Krels package now implements the interface). This means that a simple interface has been provided that enables the Poller to change communication mediums (or devices) to ones which expose this interface.

4.3.3 Poller

The Poller is responsible for polling the UAV / HawkSim for data and storing the data that is returned. Once enough data has been collated it is then responsible for passing that data to the other components within the UAV interface.

The packets that are sent use the cyclic redundancy check (CRC). Each packet that is received has its checksum checked. The checksum is used to check the integrity of the packet after transmission. If the packet has an incorrect checksum the Poller removes this packet from use in the update of flight data.

The Poller utilises the IPoller_Interface for its communication purposes with the UAV / HawkSim. Each new packet generates an event and updates the information in the FG_Interface. The Graphical User Interface (GUI) listens for the events and on each occurrence updates its display. The Poller has a reference to an instance of the FlightGear and uses its setter methods to update its information. The Poller cycles through polling the UAV and sending its data on to the other components every 20 seconds (50 Hz).

4.3.4 Graphical User Interface (GUI)

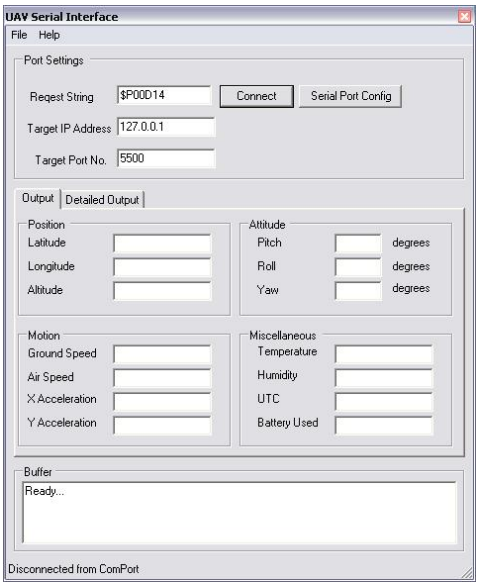


Figure 9: UAV Graphical User Interface

The GUI (Figure 9) is the front end for the Poller. It aids the user in configuring and starting the Poller. It has text fields and buttons that set up and start the the Poller:

IP Address this configures the Poller to send FlightGear-formatted data on to the correct workstation (through the FG_Interface)

Request String the string that is passed to the UAV to request all relevant information to update the FlightGear instance. This string may change in the future so it has been included in an editable text field.

Target Port No. the port number that FlightGear information is sent on. Can change depending on the configuration of FlightGear.

Connect connects the Poller to the serial port. It secures the serial port for its use.

Serial Port Config. opens the dialog provided by Mr. Krels code. Allows advanced configuration of the serial port.

Continuous Send starts sending requests for information to the UAV. This only appears once connected to the serial port.

Output tab displays the flight information received via packet.

Buffer outputs other messages generated by the Poller. This includes checksum error messages.

4.3.5 FG_Interface

The FG_Interface is responsible for updating an external FDM and then sending it to an instance of FlightGear. It has been compiled in to a Managed C++ DLL under the .NET framework so that it can be used in our applications that are primarily compiled in C#, a .NET language. C++ is used because the FDM provided by FlightGear is coded in C++ and any code that interacts with it is also required to be in C++.

A Managed C++ DLL is the .NET frameworks method of making C++, which is inherently unmanaged, a managed language by adding pre-compiler switches to the methods. For example, "__gc was added to the methods in this class to make them fall under the garbage collection routines run by the .NET framework. This is so that it can be used by C# to use in the UAV Interface.

The FG_Interface exposes numerous setter methods for setting properties in the FDM. It also exposes the method `send` which sends the FDM over the network. It does this using the User Datagram Protocol (UDP), a connectionless protocol for networks.

The network information (e.g. IP address, port number) is set on initialisation of the interface. The Poller uses the information from the GUI text fields described above to pass on this information.

Below is a basic UML diagram showing the relationships of each component (Figure 10). You will notice that the basic relationship between all the components is one to one. In Appendix A there are more detailed diagrams of each UML component.

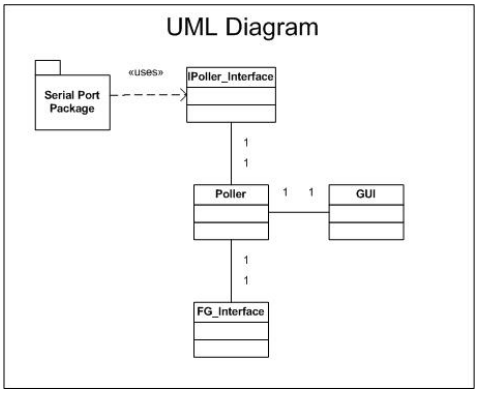


Figure 10: Basic UML Diagram

4.3.6 Results

The implementation was tested using the HawkSim simulator and an instance of FlightGear running on the same computer. FlightGear displayed the UAV (we chose an existing aircraft, the built-in glider, to represent the UAV because of its visual similarities) in flight successfully.

We were able to pan within the cockpit and outside the aircraft using the 3D camera within FlightGear. The screenshots below (Figure 11) show the UAV flying near Whenuapai Airbase and looking back towards the Waitemata Harbour.

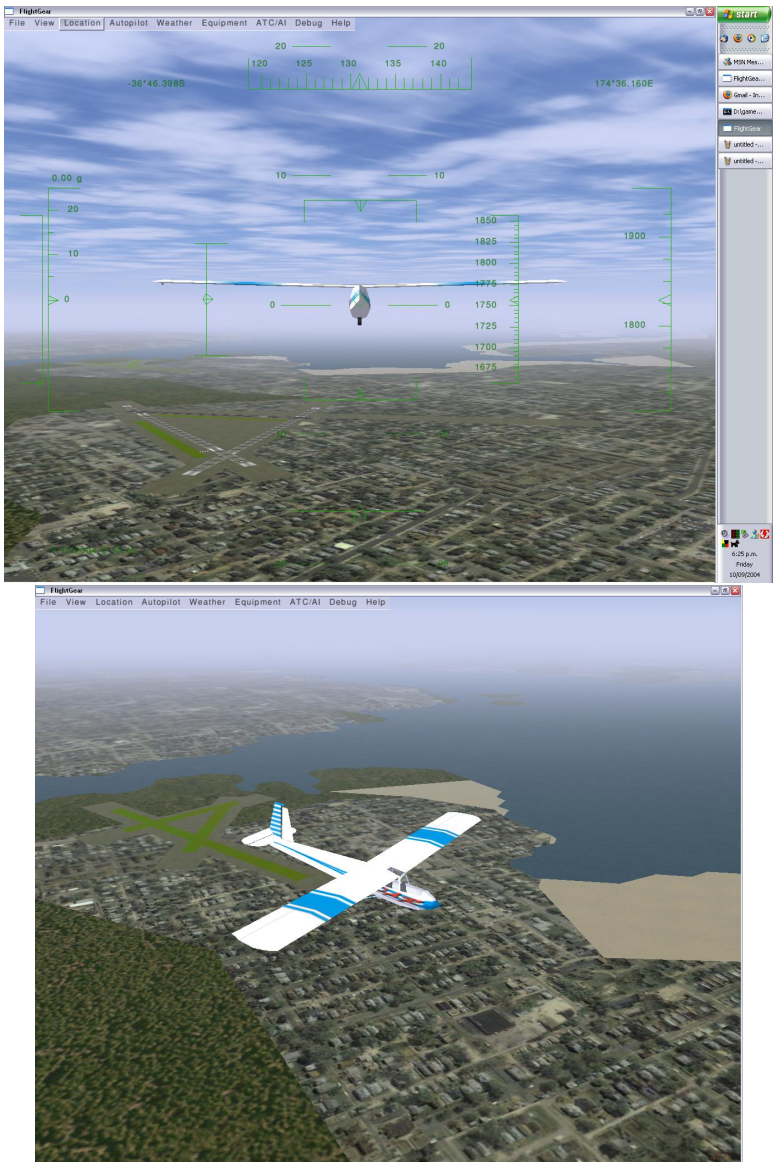


Figure 11: Screenshots of the UAV over the North Shore

The UAV Interface and its included components work well, apart from a few minor bugs within the GUI. The 1st Aim of the project has been fulfilled. The next part of the project continues with the 2nd aim; generating FlightGear terrain using JGSF data.

5 2nd Aim: Use JGSF Terrain Data in to the Environment in Order to Create Scenery with Correct Elevation and Features

5.1 Research

In order to understand the requirements for this aim, research was undertaken to investigate the JGSF data and the terrain tools that TerraGear provides. Also research in to other useful utilities were also investigated, specifically the FlightGear Scenery Designer and Atlas.

5.1.1 Terrain Data Format Overview

Terrain data comes in two varieties; raster and vector based. Raster based data is based on a grid or a matrix, where the average values in a cell is used to represent the data in that cell [17]. The data can be any sort of data but here it is primarily elevation data. The size of the cell may vary, the smaller the size the better the resolution.

For instance, terrain data that has a resolution of 1km is not very realistic as it averages values in a reasonably large cell and may miss distinct features in the terrain such as hills and valleys. If the resolution is increased (i.e. the cell coverage is reduced) a better picture is produced as it will now include the missing features. Obviously the higher the resolution the more information and the bigger the size of the terrain files.

Vector based terrain data stores its information as a collection of points, lines and polygons, where a line is a collection of related points and a polygon is a collection of related lines [18]. Vector data has the benefit of displaying at the maximum resolution of the device using the data, as vectors can hold the exact location of a piece of terrain.

Vector data can describe exact points and approximate its lines and polygons, providing an accurate means of display. There are different sets of data which vary with the accuracy of the lines and polygons and the amount of point data available.

Examples of raster data include Digital Elevation Maps (DEM) and Digital Terrain Elevation Data (DTED) data. Vector data examples include Vector Product Format (VPF) and ShapeFiles. These formats are described further in the Glossary (pg. 31).

5.1.2 Joint Geospatial Support Facility Terrain (JGSF) Data Formats

The primary role of JGSF is described on the New Zealand Defence Forces (NZDF) website:

> The primary military role of JGSF is to acquire, collate and maintain high integrity databases for production and dissemination of Geospatial products and services to static and deployed NZDF HQs and Force Elements. JGSF is also the NZDFs central provider of maps and charts, digital databases and feature foundation data and undertakes commercial hydrographic services on behalf of the RNZN. [19]

For this reason, JGSF operates a large database of a variety of data for the Defence Force. It includes but is not limited to; DTED, VPF and ShapeFiles.

Specifically, for this project, JGSF provided the following data:

1. NZ Topographic Data, WGS 84 (ShapeFile)
2. NZ DTED Level 2, WGS84 (DTED)
3. NZ VMAP Level1 Data Lib (VPF)

VMAP and WGS 84 are also described in the Glossary (pg. 31).

5.1.3 TerraGear and associated Data Formats

TerraGear is a collection of libraries and tools that are used to build FlightGear scenery. They have been developed on-line as an open-source project. They use freely available terrain data (supplied by the Geographic Information System) with the tools.

The aims of the TerraGear project are to use freely available GIS data to build 3D maps, including terrain roads, buildings, rivers and bridges. Also it is to be used in real-time rendering applications; FlightGear being the primary example.

The data that TerraGear tools can use include; DEM, Shuttle Radar Topography Mission (SRTM) data, VPF (in Vector Map Level 0 or 1), and ShapeFiles. These formats are also described in the Glossary (pg. 31).

The tools provided with TerraGear include:

raw2ascii converts raw DEM files to ASCII format.

demchop chops up ASCII DEM files into smaller files for TerraGear to handle.

hgtchop chops up SRTM files, similar to demchop.

tgvpf uses VPF files, specifically, Vector Map Level 0 or 1 to create scenery objects and draw boundaries for scenery textures.

fgfs-tools-client and server uses the data compiled from the chop utilities and the tgvpf command to construct the FlightGear scenery. Uses a client and server architecture so that they can be used in a nodal network or on a microprocessor machine.

5.1.4 FlightGear Scenery Designer

FGSD is a project that helps a user to design custom sceneries for FlightGear. Included in the application is the ability to place 3D objects and alter elevation by using imported bitmaps as guides. Planned developments include providing photo-realistic sceneries and airport design.

FGSD could be used in this project if alterations to the produced TerraGear are required.

5.1.5 Atlas

Atlas is a utility to be used in conjunction with FlightGear. It is used to create bitmap images from FlightGear scenery and then browse those maps. It can also display a crosshair representing an aircraft in a running instance of FlightGear.

As one of the aims is to produce 2D bitmaps of scenery; this tool will be incorporated in to the project.

5.2 Compilation of the TerraGear libraries

For the compilation of TerraGear various platforms were used. As TerraGear is developed in Linux, a compilation on this platform was used as a basis to compare other compilations.

There is a preference within the Defence force for the Windows platform. Therefore compilation on this platform using CygWin, a Linux like environment for Windows, and Windows compilers was also conducted.

5.2.1 Linux

For this platform, different distributions of the Linux operating system were used. They included; RedHat 8.0, RedHat Enterprise Edition (RedHat ES), Mandrake 10 and Fedora Core 2. On each different version, compilation was different and on a couple it did not compile at all. RedHat ES and Mandrake 10 compilation failed due to missing libraries. These libraries were in the correct directories and were present, but the configure scripts failed to find them.

On the other Linux distributions the TerraGear tools and libraries did compile. They were used to successfully generate scenery which was then imported in to FlightGear. This scenery displayed correctly in a running instance of FlightGear.

5.2.2 CygWin

The main reason for compilation on CygWin was because of its Linux like environment. It provides Linux utilties and system calls based on top of the Windows operating system. This enables easy compilation of Linux source code in to Windows binaries. Also, the development lists for TerraGear suggested compiling using CygWin. The produced Windows binaries could then be used in a front end application.

The compilation of TerraGear on CygWin failed. The errors that were experienced were similar to those in the failed Linux distributions. Libraries that were installed were not found in the compilation. There was difficulty in the setup of CygWin as well which hindered progress. The setup utility provided for CygWin was awkward to use and did not install requested libraries.

After installation and initial attempts, CygWin was discarded as a viable option to use in compilation.

5.2.3 Windows

The main reason for using Windows is to fit in with the NZDF preferred platform. For development, Visual Studio .NET was used to compile the source code

for TerraGear and its associated libraries. There were many difficulties in the compilation for Windows which hindered development.

Difficulties in compiling included:

- Compilation configurations in the header files needed to be changed in order to compile. The original developers of TerraGear were using Microsoft Visual C++ 6 compiler. This caused minor errors when compilation was tried with the .NET compiler.

- The file system separators all needed to be changed. The file system separators in Linux are different than in Windows; '/' and '\' respectively. This syntactic difference caused a lot of errors until a thorough find and replace of the entire source within TerraGear was conducted.

- A subtle error in the demchop utility caused a number of errors when running. The way that Linux and Windows stores doubles are different. Linux stores the exponential part of the double using 2 characters but Windows uses 3. When read back in, the different format caused errors as it read in one character short of what it was meant to. The code was changed successfully to rectify this problem.

- In the fgfs-tools-client and server utilities there was significant use of Linux operating systems command, such as `fork`. This required a major rewrite of the source to use Windows commands and libraries. One example was replacing the `fork` command with threads to provide the same functionality. Another is the `mkdir` command and its Linux switch '`-p`'. In Windows the switch is not supported and so needed to be removed for it to work correctly.

After these and several other minor errors, the Windows compilation succeeded. When the utilities were used the scenery was produced in the correct folders. When importing the scenery in to FlightGear, however, it did not display correctly. The scenery did not display like the Linux compiled scenery tried previously.

On comparison, it was noticed that the file sizes between the different built sceneries (Linux and Windows) had different sizes. The project team believed that this was similar to the error experienced with demchop. Other small, hidden differences were causing the utility to fail.

The Windows compilation was discarded, after a considerable amount of time was spent on it. The due date for the project was approaching, and the project team decided to focus on results with the Linux tools instead.

5.3 Using JGSF data with the Compiled Tools

The data provided by JGSF, as listed above (refer to section 5.1.2), was not in the correct format. Conversion tools were investigated to see if it was possible to convert the files to a suitable format. A tool that was investigated was the Geographic Resources Analysis Support System (GRASS). This is a widely used tool for the manipulation and conversion of terrain data. Unfortunately, the conversion that we tried, DTED to USGS DEM format, was not supported by GRASS.

Because of this and the lateness with which the terrain data arrived, we were not able to build any scenery using JGSF data. After contacting them about the differing formats, it was discovered that they can produce data in the format that we require. Once again due to time constraints the project team decided to leave this issue until after other project work was completed.

5.4 Results

The Linux compilation of TerraGear succeeded. The tools can be used to make scenery for FlightGear once the required data is available. Unfortunately there was no successful compilation of the tools in Windows and data supplied to us by JGSF was incompatible with the tools. Once JGSF has provided new compatible data, research shows that the tools compiled under Linux should be able to produce the scenery.

6 3rd Aim: Integrate the Virtual Maritime System Architecture in to the display

The next aim was to integrate Virtual Maritime System Architecture (VMSA) in to the display. This included virtual units within a Virtual Battleground Scenario. These units would be displayed from an aircrafts point of view. Unfortunately, due to time constraints, only the research was completed for this aim.

6.1 Research

6.1.1 Multiplayer

The VMSA units were to be integrated in to FlightGears display using multiplayer. At this time the multiplayer engine is still under development. No development was conducted in order to complete this aim.

The future multiplayer engine is believed to be based on the flight dynamic model. If this is the case, the FG_Interface can be used to update individual FDMs for use in the multiplayer; each unit would have an instance of the FG_Interface. The units would then update their individual FDMs using the provided interface. The FDMs are then used by the simulator to update the positions of the units.

6.1.2 Debrief

Another application that has been investigated just recently is the Debrief application [20]. This tool is used by the Royal Navy in England to replay maritime operations using 2D and 3D displays. The files that store the replays are simple text files and are human readable. The 3D display uses models that represent different types of units but it does not produce terrain. Terrain generation, in this respect, is not required because the exercises happen at sea in open water.

This is not an alternative to FlightGear; the display from aircraft units is still required. It can be used to display maritime operations however, if an interface to real-time data is developed.

This information, including emails between the project team and the head developer of Debrief, has been passed on to Graham Macferson at DTA.

7 Conclusions

The aims of the project were:

- Develop a 2D / 3D environment to simulate the Unmanned Aerial Vehicle (UAV) and its flight path,
- Use Joint Geospatial Support Facility (JGSF) terrain data in the environment in order to create scenery with correct elevation and include features such as buildings, roads and waterways. This aim includes the requirement that a 2D bitmap of the created scenery can be generated, and
- Integrate the Virtual Maritime System Architecture (VMSA) to display an aircraft unit and other units within a Virtual Battleground Scenario.

After initial investigation, a decision to use a flight simulator was made. The flight simulator was required to fulfil three requirements:

- Use remote data to update an aircraft within the simulator display.
- Provide tools that would generate scenery for the display from JGSF data or its equivalent.
- Have a suitable utility to allow other units to be displayed within the display.

Two flight simulators were looked at; Microsoft Flight Simulator and FlightGear. After some research FlightGear was chosen based on its 3D display of remote aircraft, its scenery generation tools, and its predicted multiplayer engine.

The first aim of the project was to display the UAV in a 3D environment. Using FlightGears external Flight Dynamic Model (FDM) and the UAVs packets, an initial solution was designed. The design used an UAV interface which converts the UAV flight information in to FlightGear FDMs, which are then sent via a network to running instances of FlightGear.

The implementation uses a serial port package, a Poller, a Graphical User Interface (GUI) and FG_Interface. The Poller polls the UAV through the serial port to retrieve flight data. This is sent to the GUI and the FG_Interface. The FG_Interface updates a FDM and sends this to FlightGear. This FDM updates the position and orientation of the displayed aircraft. This implementation fulfills the first aim of the project.

The second aim of the project was to use TerraGear tools, used for creating FlightGear scenery, with JGSF data. The tools were successfully compiled on Linux and used on provided data. The compilation of these tools failed on CygWin and Windows. The JGSF data was in the incorrect format for the tools and investigation in to conversion tools was conducted. These issues were not resolved due to time constraints.

Due to time constraints the 3rd aim of the project was only researched. It was determined that some components from the first aim of the project, namely the FG_Interface could be reused for unit integration into FlightGear. Also, Debrief, a maritime operation analysis tool could be used for displaying units from the VMSA.

7.1 Future Work

The future work involves the completion of the second aim of the project. Once compatible data for use with TerraGear becomes available, terrain and scenery can be generated. The third aim of the project will be explored further by the DTA or by Chris Mills, as he is employed with the Navy. This may involve more development with FlightGear and its multiplayer abilities or focus may shift on to applying Debrief to the VMSA instead.

8 Glossary

DEM Digital Elevation Model : A model of terrain relief in the form of a MATRIX. Easch element of the DEM is regared as a node of an imaginary grid. The gird is defined by identifying one of its corner (lower left usually), the distance between nodes in both the X and Y directions, the number of nodes in both the X and Y directions and the gird orientation. [21]

DTED DTED2 is the basic high resolution elevation data source for all military activities and systems that require landform, slope, elevation, and/or terrain roughness in a digital format. DTED 2 is a uniform gridded matrix of terrain elevation values with post spacing of one arc second (approximately 30 meters). [22]

ShapeFile stores nontopological geometry and attribute information for the spatial features in a data set. The geometry for a feature is stored as a shape comprising a set of vector coordinates. Shapefiles handle single features that overlap or that are noncontiguous. Shapefiles can support point, line and area features. [23]

SRTM The Shuttle Radar Topography Mission (SRTM) obtained elevation data on a near-global scale to generate the most complete high-resolution digital topographic database of Earth. SRTM consisted of a specially modified radar system that flew onboard the Space Shuttle Endeavour during an 11-day mission in February of 2000. [24]

VMAP Vector Map (VMap) Level 0 is an updated and improved version of the National Imagery and Mapping Agency's (NIMA) Digital Chart of the World (DCW). The VMap Level 0 database provides worldwide coverage of vector-based geospatial data which can be viewed at 1:1,000,000 scale. It consists of geographic, attribute, and textual data stored on compact disc read-only memory (CD-ROM). [25]

VPF The Vector Product Format (VPF) is a standard format, structure, and organization for large geographic databases that are based on a georelational data model and are intended for direct use. [26]

WGS84 World Geodetic System 1984. WGS 84 is an earth fixed global reference frame, including an earth model. [27]

References

[1] "Defence Technology Agency," September 2004. http://www.nzdf.mil.nz/corporate/hqnzdf.html#Defence

[2] S. Canney, J. Best, and A. Cramp, "Virtual Maritime System Architecture Decription Document Issue 2.00," tech. rep., Defence Science and Technology Organisation, July 2002.

[3] "Microsoft Flight Simulator 2002 Home Page," September 2004. http://www.microsoft.com/games/pc/fs2002.aspx.

[4] "BGL Tools," September 2004. http://www.scenery.org/design_utilities_a.htm.

[5] Microsoft Corporation, *Creating Terrain and Land Classification*, September 2002.

[6] "FSGenesis Home Page," September 2004. http://www.fsgenesis.net.

[7] "VFR Scenery of England and Wales," September 2004. http://www.horizonsimulation.com/index.html.

[8] "FlightGear Project," September 2004. http://www.flightgear.org/.

[9] "TerraGear Project," September 2004. http://www.terragear.org/.

[10] "FlightGear Scenery Designer," September 2004. http://fgsd.sourceforge.net.

[11] "Atlas Home Page," September 2004. http://atlas.sourceforge.net/.

[12] M. Matkovic, "FlightGear Development List," September 2004. http://baron.flightgear.org/pipermail/flightgear-devel/2004-June/028717.html.

[13] P. Strong, "UAV Uplink Specification." This report is awaiting security vetting and as such is restricted.

[14] P. Strong, "UAV Downlink Specification." This report is awaiting security vetting and as such is restricted.

[15] T. J. Krell, "Serial Communications : The .NET Way," September 2004. http://www.codeproject.com/dotnet/DotNetComPorts.asp.

[16] ".NET CoadTools," April 2004. http://www.coad.net/noah.

[17] "SIS Glossary," 2004 September.

[18] "Glossary of Terminology:L through Z," 2004 September.

[19] "Joint Geospatial Support Facility," September 2004. http://www.nzdf.mil.nz/jgsf/index.html.

[20] "Debrief," September 2004. http://www.debrief.info/index.shtml.

[21] "Glossary Terms for Geographic Information Systems," September 2004. http://www.geocities.com/capecanaveral/9727/gist.html.

[22] "Global Change Master Directory," September 2004. http://gcmd.nasa.gov/records/GCMDCanadaDNDDTED2.html.

[23] "ShapeFile Glossary," September 2004. http://pcsa.hrsa.gov/datamaps/glossaries/gisglos

[24] "Shuttle Radar Topography Mission Home Page," September 2004. http://www2.jpl.nasa.gov/srtm/.

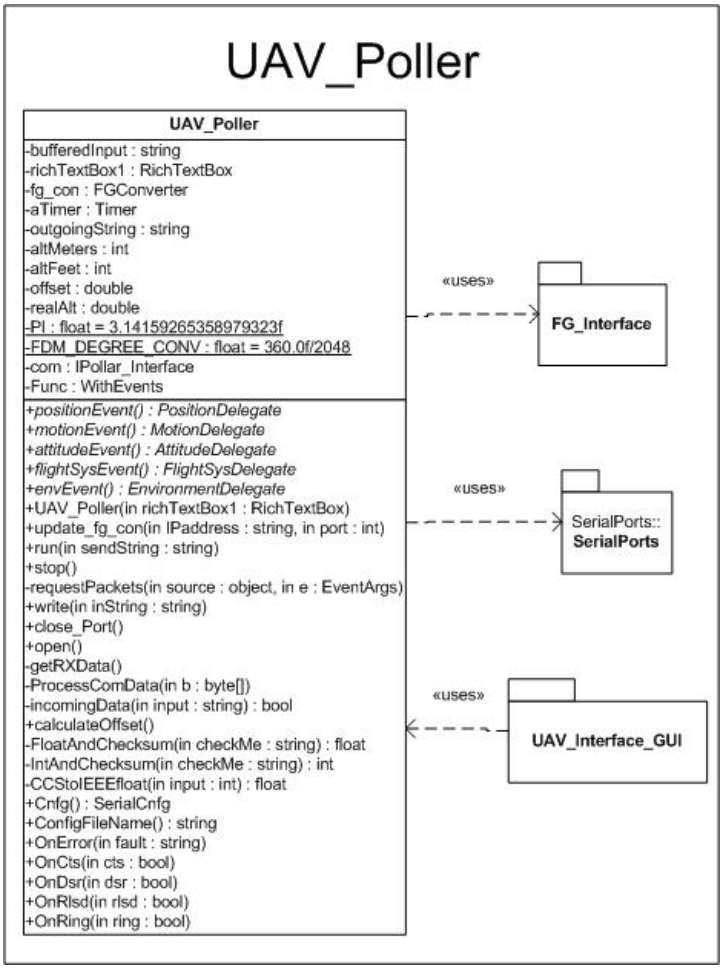
[25] "Vector Map Level 0 (VMap)," September 2004. http://earth-info.nga.mil/publications/vmap0.htm.

[26] "National Geospatial Intelligence Agency Home Page," September 2004. http://WWW.NIMA.MIL.

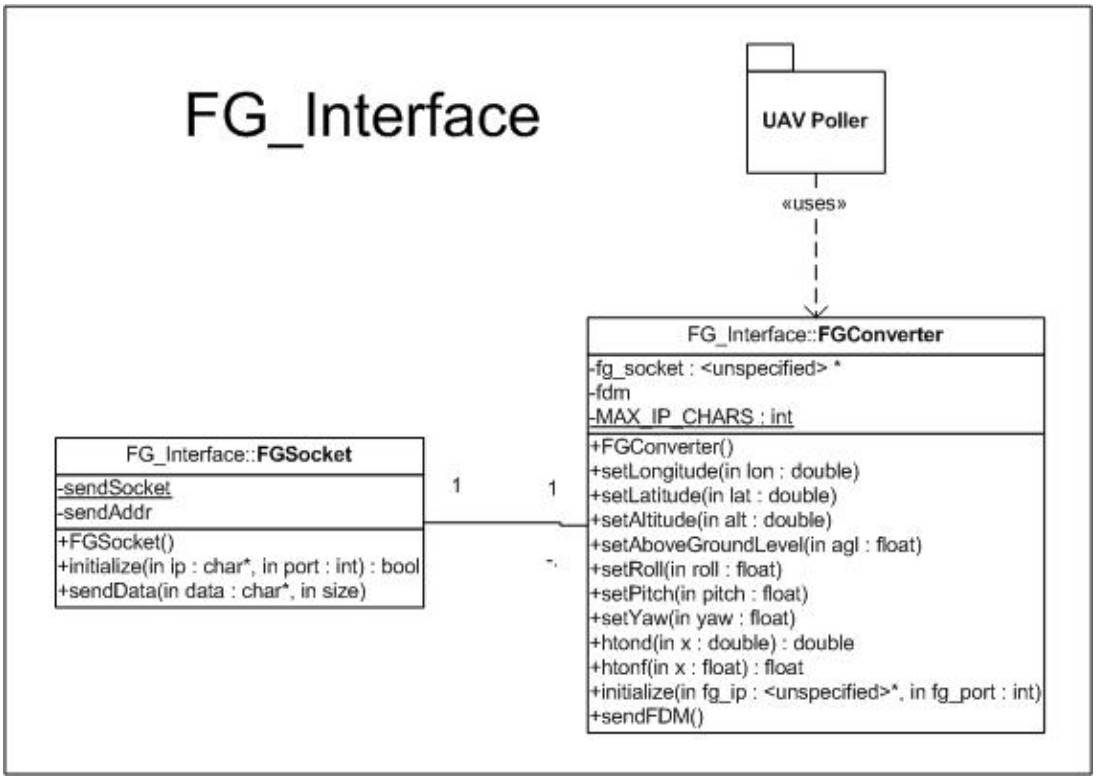
[27] "Abbreviations and Glossary," September 2004. http://www.lech.priv.at/publications/rwbook/node1027.html.

A Appendices

A.1 UAV Poller



A.2 FG Interface



A.3 Graphical User Interface