CARLETON UNIVERSITY
SCHOOL OF COMPUTER SCIENCE
Faculty Advisor: Professor Michel Barbeau (PhD)

Real-Time Flight Analysis

Extending Open Source Flight Simulators

Ryan Henry

April 9th, 2008

The work presented in this paper is new and was prepared only for COMP4905 Honours Project, Winter 2008, School of Computer Science, Carleton University.

Acknowledgements

Thank you to Professor Barbeau for taking the time to advise me on this project.

Thank you as well to my co-workers at Flightscape who provided the background knowledge I needed to understand the aircraft controls and mechanics.

Table of Contents

1. Introduction

1.1 Project Motivation

Traditionally, flight simulators have been used as a hobbyist past-time as the cost of actually flying is high. In some areas, flight simulators are used to train pilots and crew on normal and emergency procedures.

When flight simulators are used for training purposes, pilots tend to repeat the same simulation in order to learn from their mistakes and perfect procedures. While many existing flight simulators allow the user to replay a video of the flight, sometimes a more detailed analysis of an event is needed. An event can be anything from a warning light when banking too steeply to an actual crash. When reviewing these events, it can be helpful to see the conditions leading up to the event rather than only seeing what is currently happening.

1.2 Project Description

To extend the Flight Data Analysis capabilities of an Open Source flight simulator. This project will create an On-Screen Flight Data Analysis tool that allows the user to view graphs of specific data.

1.3 Summary of the Result

The project successfully added four on screen data analysis graphs with the following parameters: Airspeed, Pressure Altitude, Angle of Attack, and G-force.

1.4 Outline of the Report

Detailed background information on Flight Simulators is provided in Section 2. Development setup is reviewed in Section 3. Software Design is examined in detail in Section 4, including an explanation of the on-screen display of the project. Finally, results of the project are reviewed and examined in Sections 5 and 6, respectively.

2. Detailed Context/Background Information

In this section, we review the history of flight simulators, how they are used today and explore the motivation behind this project.

2.1 History of Flight Simulators

A flight simulator is a system that tries to recreate, or simulate, the experience of flying an aircraft. It is as realistic as possible. The different types of flight simulators range from video games up to full-size cockpit replicas mounted on hydraulic or electromechanical actuators, controlled by state of the art computer technology.

Flight simulators are extensively used in the aviation industry for design and development and for the training of pilots and other flight deck crew in both civil and military aircraft.

Figure 1: Microsoft Flight Simulator 1982

Figure 2: Microsoft Flight Simulator 2005

2.2 Flight Simulators as Training Devices

One category of flight simulators is the System Trainer simulators. These teach pilots how to operate various aircraft systems. Once a pilot understands the aircraft systems and controls, they can begin training on cockpit procedures.

Under normal operation, and especially in the event of an emergency, pilots are trained to follow a very detailed list of procedures. These procedures are designed to keep the aircraft operating correctly or, as safely and quickly as possible, bring the aircraft to a landing. In an ideal situation, these procedures are all that is necessary. In emergency situations, however, pilots may need to adapt the procedures. It is for this reason that users need to understand how the procedures were designed and what affect each has on the aircraft.

Current flight simulators try to create realism for the pilots. In practice, this trains pilot's instincts and, in emergency situations, allows the pilots to better perform. However, this has also led to a lack of data analysis software in the market. In existing flight simulators, in order to examine a pilot's mistakes during a procedure (or mistakes that led to an event), a user's only recourse is to watch a video playback of the event. Unless the user is an experienced pilot, they may miss certain details of the event, and hence not learn from it.

2.3 Data Analysis

When reviewing a procedure or event, if a user has more information, they will be better able to understand their mistakes. In actual flights, this data is stored in the aircraft's Flight Data Recorder (or blackbox). Blackboxes continuously capture the performance parameters of the aircraft. This information is typically used by an airline, aircraft manufacturer or investigator to determine what caused an event. While only 88 parameters (Wikipedia, Flight Data Recorders, 2008) are required as a minimum in the US, modern flight data recorders can capture hundreds of parameters (up to 1024 parameters in an Airbus A380).

A list of parameters used in this project is detailed in Section 4.2

2.3.1 Real-time Data Analysis

Real-time data analysis allows a user to examine specific data captured by a flight data recorder as they are reviewing an event. For instance, if an investigator is recreating a crash, they are probably looking for the exact set of circumstances that led to the crash. By viewing a graph of all relevant data up to and during the crash, the investigator can pin point exactly what caused the event; no matter if it was pilot error or a mechanical failure.

2.3.2 Companies using real-time analysis

As the airline industry grows, the needs for faster data processing has been increasing. Many companies in the Ottawa area currently specialize in Real-time Data Analysis for event and accident investigation. Below is a list of companies or organizations currently involved in data analysis:

- Flightscape (a division of CAE Inc.):

  Flightscape is a flight safety company providing expertise in flight recorder playback, analysis and flight sciences. Flightscape develops software tools that enable the effective study and understanding of recorded flight data to improve safety, maintenance and flight operations. Their Insight|Analysis product is widely used for accident investigation, and by aircraft manufacturers and airlines who wish to study complex events.

- DRS Technologies Inc.:

  DRS is a supplier of defense electronic products and systems to the US military. Their simulation products are used to train fighter pilots.

- NAV CANADA:

  NAV CANADA is Canada's civil air navigation services provider. They provide safe, effective and efficient air navigation services to aircraft operating in Canadian domestic airspace and in international airspace assigned to Canadian control. They also provide air traffic control, flight information, weather briefings, aeronautical information, airport advisory services and electronic aids to navigation.

- TSB:

  The Transportation Safety Board is an independent agency responsible for investigation of accidents involving aircraft (among others) in Canada (except aircraft of the armed forces and the intelligence agencies). It also investigates every civil aviation accident in Canada.

3. Development Setup

3.1 Software

An Open Source flight simulator was chosen in order to focus on implementing the Data Analysis controls rather than creating a simplistic simulator from scratch. CSP (or the Combat Simulator Project) was chosen since it allows for both native Windows and native GNU/Linux building. Originally, FlightGear was chosen as a flight simulator but this led to compatibility issues (see Section 6.2).

To run CSP a moderately fast computer (minimum 1GHz), 512MB of RAM, and a modern 3D graphics card (GeForce4 or better) is recommended. A system consisting of a Pentium Core2Duo 3GHz, 4GB of RAM, and a Geforce GTS 8800 640MB video card was used for all major development and testing.

CSP was built with the most up-to-date version of Windows Vista (SP1), Visual Studio 2005 (SP1) and Python 2.4. Under Windows, CSP requires:

- Microsoft Platform SDK
- Python 2.4
- SWIG 1.3.27
- SCONS 0.96.95
- CSP DevPack 0.6.1
- Balkan Terrain pack
- TortoiseSVN (for source checkout)

3.2 Setup

Building the latest version of CSP can be done by following these steps:

After installing the requirements above, the latest source must be checked out using TortoiseSVN. This can be found at: https://www.zerobar.net/svn/csp/trunk. For help using TortoiseSVN refer to: http://tortoisesvn.net/faq

The following environment variables are necessary to properly build CSP

```
PATH =C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin;
      C:\Program Files\Microsoft Visual Studio 8\Common7\IDE;
      C:\Program Files\Microsoft Visual Studio 8\VC\BIN;
      C:\Program Files\Microsoft Visual Studio 8\Common7\Tools;
      C:\Windows;
      C:\Windows\System32;
      C:\Program Files\Python24;
      C:\Program Files\swigwin-1.3.27

LIB =C:\Program Files\Microsoft Visual Studio 8\VC\LIB;
     C:\Program Files\Microsoft Platform SDK\Lib;
     C:\Program Files\Python24\libs
```

```
INCLUDE=C:\Program Files\Microsoft Visual Studio 8\VC\INCLUDE;
        C:\Program Files\Microsoft Platform SDK\INCLUDE

CSPDEVPACK=C:\Program Files\cspdevpack-0.6
```

The source contains Python scripts to build CSP. One of the goals of CSP is to be platform independent; these Python scripts allow the small developer team to focus on improving the simulator and not worrying about updating various project files across multiple platforms. While a Visual Studio project file can be generated using the Python scripts, it is not used to build the project, which made it difficult to debug the simulator (see section 6.2).

To execute the Python scripts, open a Command Prompt and change the current directory to the "csp" folder in your working copy root folder. Once in the "csp" folder, execute the following command:

python tools/setup.py

This step only needs to be done once. It will configure the Python installation to know about CSP. Run the following command:

scons config

The "config" build argument runs various tests to check that the environment is configured correctly. Finally run the following command:

scons all

The "all" build argument builds all of CSP. A few warnings are expected — particularly when compiling source files generated by SWIG — but the build should run to completion without error. This step can take anywhere from a few minutes to tens of minutes depending on the computer.

To more easily edit the source code, we generate a Visual Studio project file suing the following command:

scons vcproj

Once the source code has built, the simulation can be run with:

cd bin
python sim.py

4. Software Design

4.1 Design Patterns

Visitor

The visitor design pattern is a way of separating an algorithm from an object structure. A practical result of this separation is the ability to add new operations to existing object structures without modifying those structures.

In CSP actions on system's models are typically implemented using the visitor pattern, whereby an instance of a visitor subclass is passed to the system's model, and then recursively to all children until each node of the tree has been visited. At each node, the visitor selectively performs its intended operation.

Composite

The Composite pattern allows a group of objects to be treated in the same way as a single instance of an object.

In CSP, system trees contain only two types of nodes: System and SystemsModel. The composite implementation treats both of these as generic nodes, but specialized visitor subclasses can act independently on the different types.

4.2 Description of Extension

The project's goal was to implement some real-time data analysis for the simulator. As adding hundreds of parameters was not within the scope of the project, some major flight parameters were chosen. Each of the parameter graphs will show the data as time progresses.

- Pressure Altitude

  The indicated altitude when an altimeter is set to an agreed pressure setting. This setting, 101.325 kPa, is a baseline pressure setting equivalent to the International Standard Atmosphere (ISA) at sea level.

- Airspeed

  The speed of the aircraft relative to the air. While different airspeed measures exist, the project will display the Indicated Airspeed; which is the airspeed reading uncorrected for instruments, position or other errors.

- Angle of Attack

  The angle of attack is related to the lift coefficient of the aircraft. While there is no mathematical relationship between lift and the angle of attack, increasing the lift coefficient generally increases the angle of attack. At a maximum, this value indicates a lift coefficient that will cause a stall in the flow of air

- G-forces

  The number of gravitational units (or g's) of acceleration subjected on the aircraft. This value is important as both the aircraft and the human pilot can only withstand so many G-forces.

4.3 Interpreting the Display

The project displays multiple parameter graphs. While not visible by default, the user can enable each of the graphs separately using the F5 - F8 keys.

F5: Airspeed
F6: Pressure Altitude
F7: Angle of Attack
F8: G-force

As seen in Figure 3, each graph consists of a standard XY graph. The X-axis is always Time, measured in seconds and will display 90s worth of data. This is always the last 90 seconds of data. The Y-axis is the data axis, measured here in knots. The scale of the data axis changes as the data values change. Each graph updates every second and data will be saved even if the graph is not visible. See Figures 4-6 in the Appendix for screenshots of each parameter graph, and Figures 7-8 for the simulator screen with and without all graphs enabled.

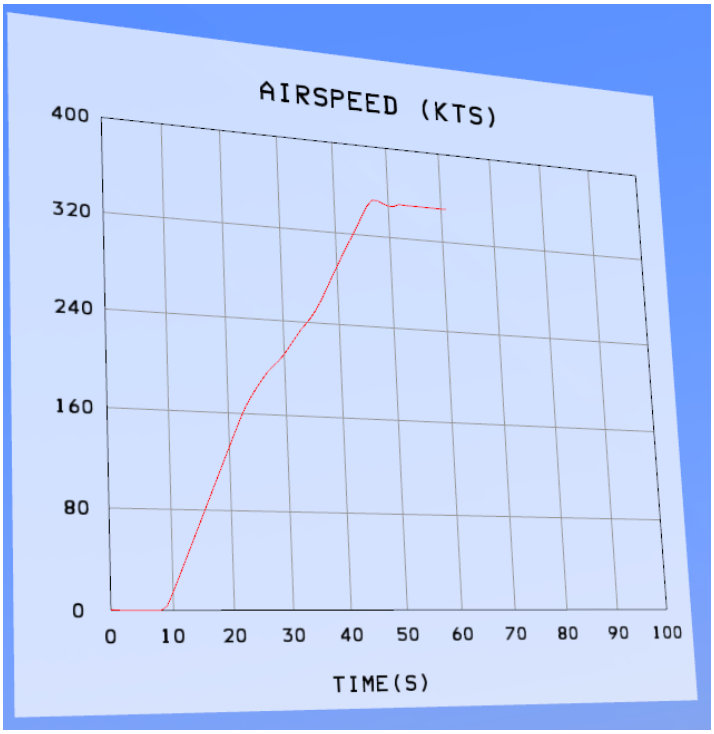

Figure 3: Airspeed Graph

5. Description of the Result

5.1 Data Calculations

While an internal clock was already available in the simulator, the desired data parameters were not accurately represented. Even though the default F-16 model shows airspeed for the aircraft, this is only represented in graphical terms (ie. using an XYZ screen coordinate system).

The initial development focused on implementing the required data parameters. The previously calculated airspeed was a good starting point. The XYZ coordinate-based airspeed was easily converted into true airspeed through the internal scale of the terrain. To determine the true airspeed (in knots) we used the following formula:

1knot = 1.852 Km/H

The Pressure Altitude is calculated in a similar way. Based on the Y-axis location of the aircraft, the altitude of the aircraft was converted to world coordinates. Using the following formula, and the values calculated for the pressure at certain heights (using the 1976 Standard Atmospheric model), the Pressure Altitude value can be calculated using:

$$A_{pressure} = A_{ASL} + (29.92\ inHg - P_{at\ Altitude}) * 1000\ ft/inHg.$$

Having already calculated the airspeed, and using the aircraft heading given by CSP, angle of attack (AOA) was calculated. AOA is used to describe the angle between the chord line of the wing of a fixed-wing aircraft and the vector representing the relative motion between the aircraft and the atmosphere.

In military aircraft, the G-force exerted on a pilot is the limiting human factor of the aircraft. Military aircraft and pilots with pressure suits can experience up to 9 g's before blacking-out. While the true g-force requires a calculation of centripetal motion and varies based on global position, this value was calculated using a more simplistic approach based on altitude.

5.2 Graphs and Model Generation

Initially, the graphs used a simple overlay on the screen. However, with the amount of data that needed to be shown, this became a problem (further discussed in section 6.2). It was decided that using dynamically generated models would allow the graphs to be a part of the aircraft HUD (heads-up display) and improve their integration into the simulator.

In conventional model-based graphics programming, predefined static models are generally used. These models cannot adapt to new situations, and they have to be very specific and cannot be generated from a single generic model even though they are very similar.

CSP uses an XML based model management system that allows certain parts of the aircraft to change with the simulator. For example, the raising of landing gear, and the HUD elements that show heading and airspeed. To start, new graph models were added into the XML system. This created a blank area on which to draw the graphs. Unlike the existing elements, the graph data had to be calculated rather than animated. Using some methods within the Open Scene Graph (OSG) library, the graphs were constructed. This allowed the necessary data to be sent into the existing model system so that it could be updated as the simulator progressed.

6. Evaluation of the Result

While the main goal of the project was met by adding real-time data analysis to CSP, this would not have been possible without the ability to dynamically generate the graph models. By creating the graphs as an extension of the aircraft HUD, valuable screen real estate was saved without comprising the quality of the data or the functionality of the simulator.

Unfortunately there was not enough time to add automatic data analysis. This improvement would have led to better feedback for the pilot using known maximum and minimum data values. Future development would be focused on this and on improving the number of supported parameters.

6.1 Problems Encountered

Open Source simulators

There are very few open source flight simulators. The most advanced one is FlightGear. While FlightGear is technically multi-platform, its roots are in Linux. A windows compatible development version does exist; but, the required third-party libraries are not always compatible with one another. After spending a few weeks trying to start the project in FlightGear, that path was abandoned. CSP was chosen for its simplicity and platform independence.

Debugging

CSP's platform independence led to another problem. In order to build the system on multiple platforms, the simulator uses various python scripts to compile the code. As this is done from the command line and is a simple script, in order to debug the code, developers have to rely on a python generated Visual Studio project file and attach the debugger to the running application. This frequently leads to synchronization errors and much frustration.

Drawing the graphs

With so many different graphs being displayed on the screen at once, it became difficult to actually see the simulated world. The graphs were then dynamically generated as models in the aircraft HUD. CSP allows the user to look around using the mouse and zoom in using shift-scrollwheel. By taking advantage of this, the size of each graph was compressed to a more reasonable size; if the user wishes to see a graph more closely, they can simply zoom in on the desired area.

References

(2006, April 26). Retrieved from CSP: http://csp.sourceforge.net/wiki/Main_Page

FAQ. (n.d.). Retrieved March 2008, from FlightGear: http://www.flightgear.org/Docs/FAQ.shtml

Insight|*Analysis*. (n.d.). Retrieved Aprl 8, 2008, from Flightscape: http://flightscape.com/products/analysis.php

Support/Reference Guides. (n.d.). Retrieved March 2008, from OpenSceneGraph: http://www.openscenegraph.org/projects/osg/wiki/Support/ReferenceGuides

Wikipedia. (2008, March 26). *Flight Data Recorders*. Retrieved from Wikipedia: http://en.wikipedia.org/wiki/Flight_data_recorder#Design

Wikipedia. (2008, January 28). *Lift Coefficient*. Retrieved from Wikipedia: http://en.wikipedia.org/wiki/Lift_coefficient

Appendix

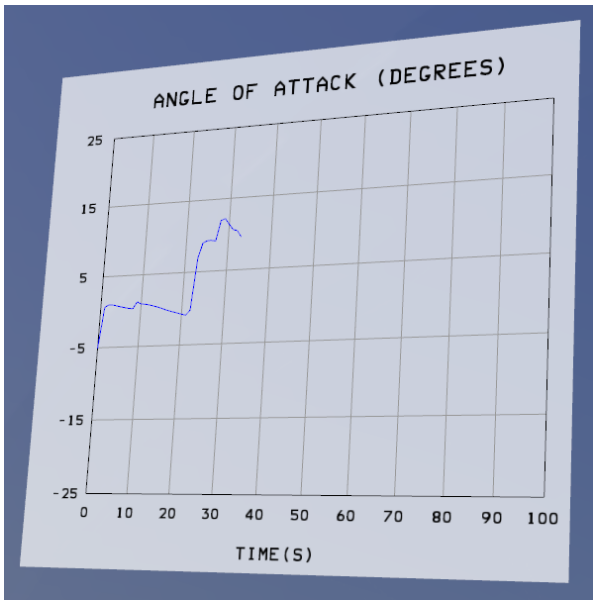

Figure 4: Angle of Attack Graph

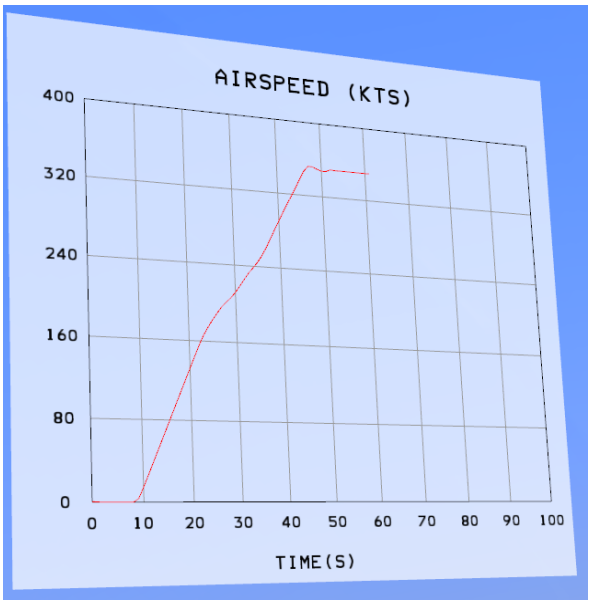

Figure 5: Airspeed Graph

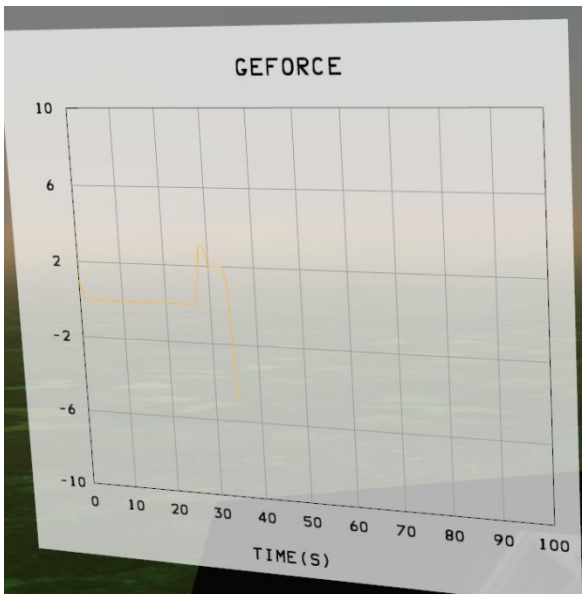

Figure 6: G-force Graph

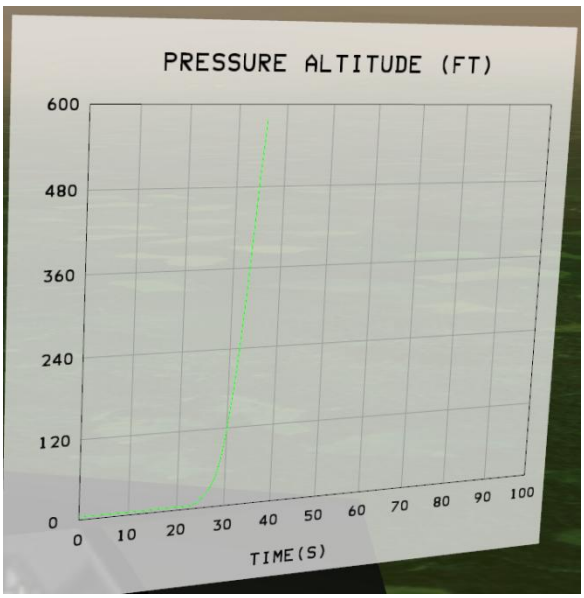

Figure 7: Pressure Altitude Graph

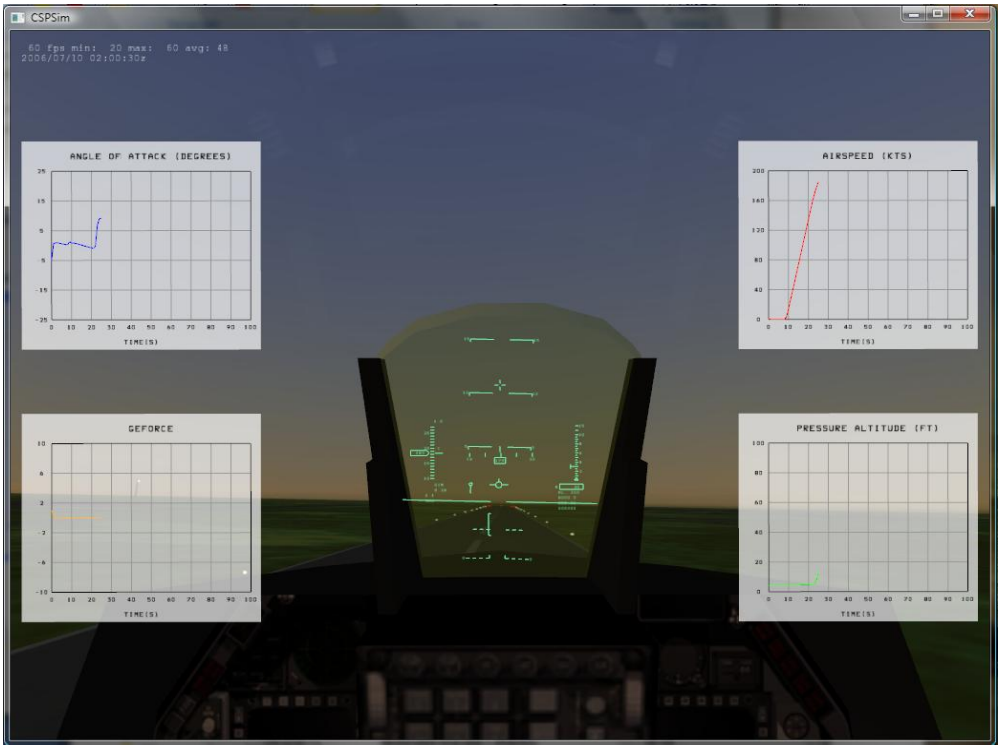

Figure 8: CSP with all graphs visible

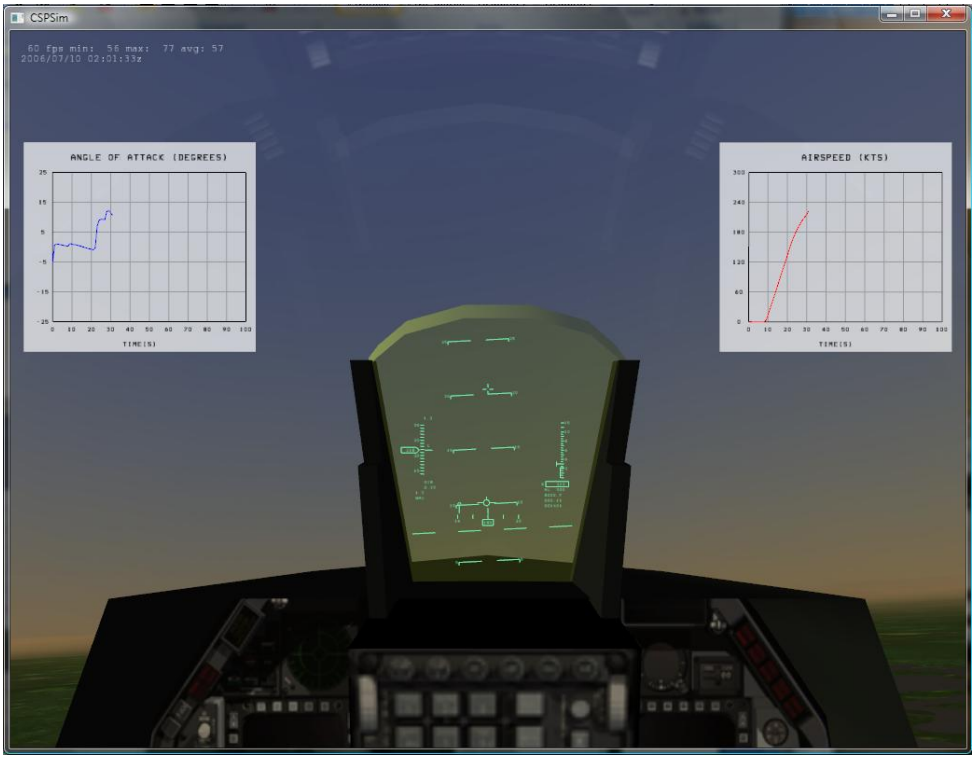

Figure 9: CSP with selected graphs visible