SIMGRAPH - A FLIGHT SIMULATION DATA VISUALIZATION WORKSTATION

Joseph A. Kaplan
NASA Langley Research Center
Hampton, Virginia

Patrick S. Kenney
UNISYS Corporation
Hampton, Virginia

Abstract

Today's modern flight simulation research produces vast amounts of time sensitive data, making a qualitative analysis of the data difficult while it remains in a numerical representation. Therefore, a method of merging related data together and presenting it to the user in a more comprehensible format is necessary. Simulation Graphics (SimGraph) is an object-oriented data visualization software package that presents simulation data in animated graphical displays for easy interpretation. Data produced from a flight simulation is presented by SimGraph in several different formats, including: 3-Dimensional Views, Cockpit Control Views, Heads-Up Displays, Strip Charts, and Status Indicators. SimGraph can accommodate the addition of new graphical displays to allow the software to be customized to each user's particular environment. A new display can be developed and added to SimGraph without having to design a new application, allowing the graphics programmer to focus on the development of the graphical display. The SimGraph framework can be reused for a wide variety of visualization tasks. Although it was created for the flight simulation facilities at NASA Langley Research Center, SimGraph can be reconfigured to almost any data visualization environment. This paper describes the capabilities and operations of SimGraph.

Introduction

SimGraph was designed to improve comprehension of the large amount of data produced from flight simulation tests in the Flight Simulation Facility at NASA Langley Research Center (LaRC). Data produced from the flight simulation is presented by SimGraph in several different formats; these include 3-Dimensional Views, Cockpit Control Views, Heads-Up Displays, Strip Charts, and Status Indicators. Since these displays may not satisfy all of the needs of the research community, SimGraph can be easily modified to accommodate the addition of new graphical displays into its existing framework with minimal changes, thereby customizing the software to the users' particular environment. These graphical displays can be taken from existing programs and easily integrated into SimGraph.

High Level Design

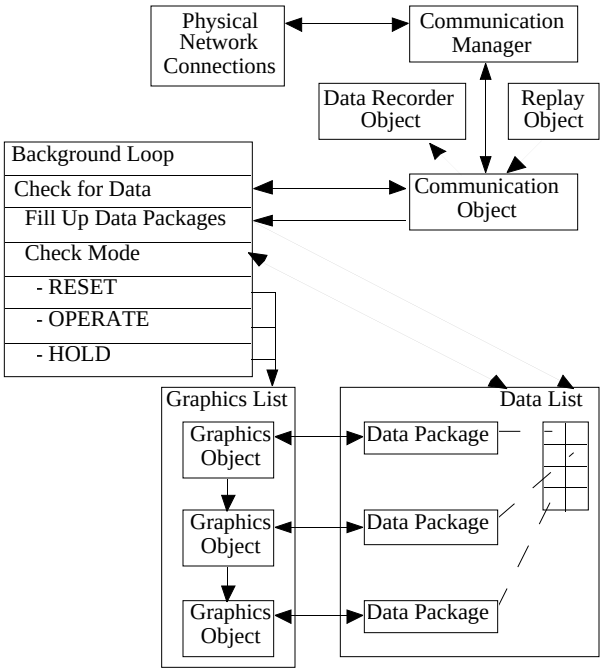

Figure 1 - High Level Design

The foundation of SimGraph is a C++ object-oriented framework. This object-oriented framework handles the communication and data transfer that is standard with all of the NASA LaRC's Flight Simulation Facility data visualization software. The framework has been thoroughly tested and is continually re-used, thus saving valuable programming resources. With each use, the framework is tested again, improving the quality of the software. The graphical displays, which sometimes are unique to each simulation, are then inserted into this

framework. As new displays are developed, they are added to an existing library. This library is an additional source of individual software components that can either be re-used or slightly modified. For example, a graphical display showing the cockpit controls of one fighter aircraft can be easily modified to show the cockpit controls of another fighter aircraft.

Object-oriented design allows a system to be viewed as a collection of objects and the interaction between those objects. This interaction occurs between the objects' interfaces (Figure 2). The implementation details of the objects are encapsulated behind the interface. Since the rest of the program cannot access the implementation details, all interaction with the object must take place via the interface. This forces clearly defined interfaces and aids in designing modular software. If all connections to an object are through the interface, then the implementation details of an object may be changed without affecting the rest of the system as long as the interface does not change. This simplifies maintenance of the software since errors within the object can be corrected without causing errors elsewhere in the program. The entire object may also be removed and replaced with another object that has the same interface.

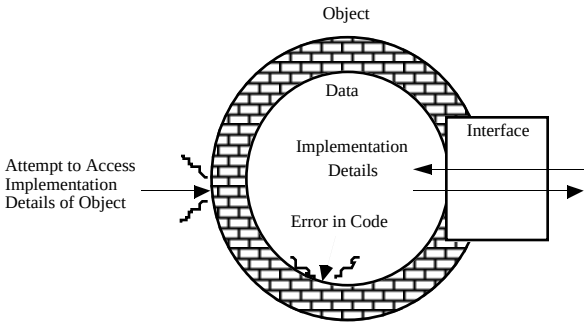

Figure 2 - Object-Oriented Design

SimGraph was designed to be computer platform independent in several aspects. It is written entirely in C++. SimGraph utilizes the X Window System to manage its graphical displays. The X Window System runs on a wide variety of computer platforms, ranging from supercomputers to personal computers. Anything that can be drawn inside an X Window can be used to build a graphical display. Several of the displays in SimGraph were written in the OpenGL graphics description language. The manner of communication between the simulation program and SimGraph is also platform independent. SimGraph can exchange data with the simulation program using Internet Domain Sockets, UNIX Domain Sockets, SCRAMNet Replicated Shared Memory from Systran Corporation, or the Advanced Real-Time Simulation System (ARTSS)[1], which is the NASA LaRC's real-time simulation input/output system. New methods of communication can be easily integrated into the program by extending the communication interface, which is encapsulated within a single object.

Detailed Design of SimGraph

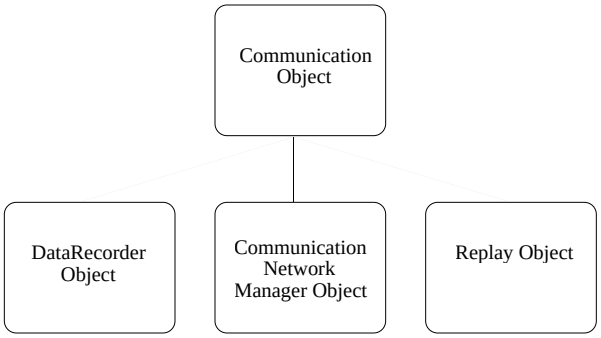

Figure 3 - Overview of the Communication Object

The Communication Object (Figure 3) is SimGraph's link to simulation data. All outside communication, either with a simulation program or a data file, is handled by the Communication Object. This includes both receiving and sending data. During a simulation run, the user may choose to save the data in a file so the data can be reviewed at a later date. When reviewing the data, the Communication Object obtains the simulation data from a file instead of a connection to the simulation program.

The Communication Network Manager Object is SimGraph's link to the simulation program that supplies it with data. The user is allowed to select from several methods of communication. The SimGraph program uses the same function calls to the Communication Object regardless of the method of communication selected by the user. This is possible because all of the implementation details of the communication have been encapsulated behind the interface. Adding new methods of communication is as simple as editing the Communication Object to handle those methods and recompiling the program. The rest of the SimGraph program will not be affected by those changes.

The Data Recorder Object archives the data transferred between the Communication Object and the Communication Network Manager Object. The Data Recorder Object is created when the user requests data to be saved, recording data from that time forward. SimGraph does not store any unnecessary or old data because of the

amount of memory that would be necessary to accommodate it.

The Replay Object of SimGraph allows the user to examine data from a previous SimGraph session. The data may be viewed at different speeds and directions. All the information for the displays will be loaded from a data file instead of being communicated from the simulation program.

Data Storage

The next portion of the SimGraph framework is the Data List (Figure 4). The Data List manages the data used by the Graphic Objects that are currently being viewed by the user. Because of the policy used to manage the data, a particular packet of data will only be transmitted once, regardless of the number of Graphic Objects that need it. A Graphic Object can be closed and the data that corresponds to it will continue to be transmitted unless it was only in use by that particular Object.

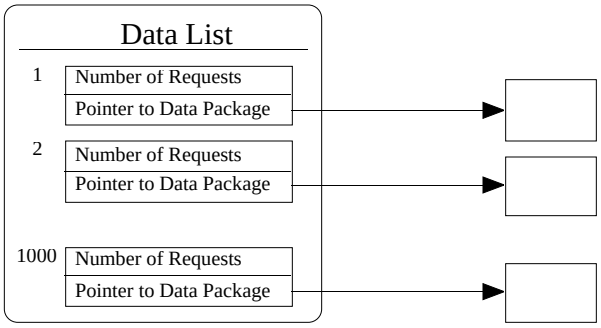

Figure 4 - Diagram of the Data List

The most important function of the Data List is data parsing (Figure 5). Data is sent from the simulation program in a large block. This block is the actual binary data which has been copied into a character array for transmission. When the Data Block is received by SimGraph, it is immediately passed to the Data List, which will parse the data, ensure that it is correct and has not been corrupted during transit, and break the data into the individual components for retrieval by the Graphic Objects.

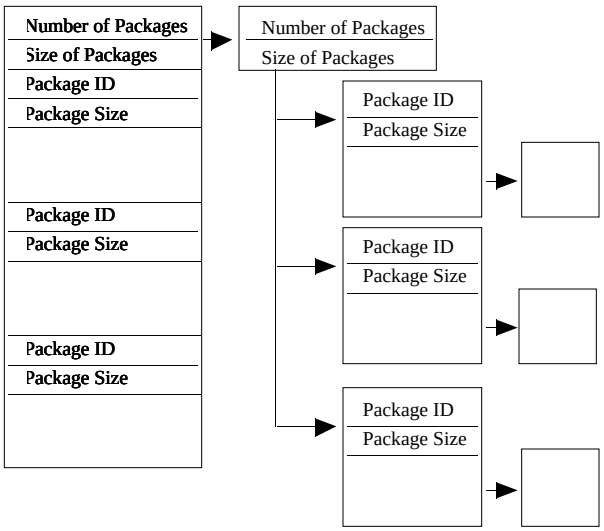

Figure 5 - Parsing of the Data Block

Graphic Objects

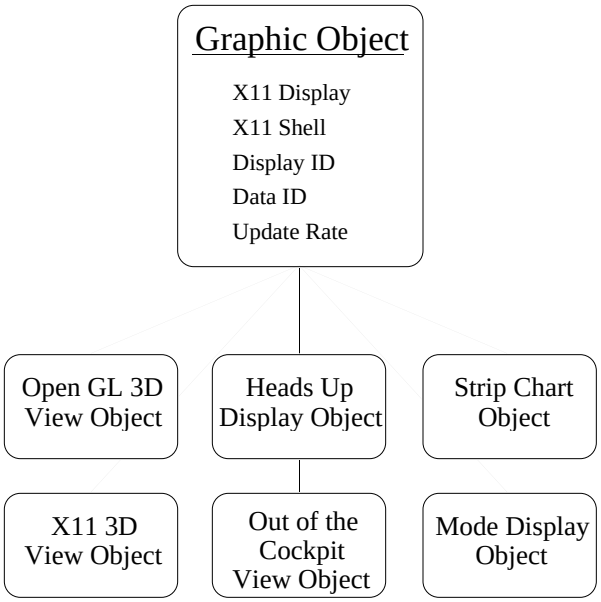

Figure 6 - Overview of Graphic Object

The Graphic Object is the cornerstone of SimGraph (Figure 6). Each display that is shown on the screen is represented by a Graphic Object. All of the Graphic Objects in SimGraph are derived from a generic parent object. This parent object is merely a skeleton of all the attributes that SimGraph Graphic Objects have in common.

The Graphic Objects have three methods in common which correspond to the three simulation states:

RESET, HOLD, and OPERATE. The initialize method is called when the Graphic Object is first created and during RESET. This routine will initialize all variables. The initialize method must be set up so that it can be called multiple times without causing any type of error (i.e., allocating without de-allocating memory, creating Widgets without destroying them, etc.). The second method is the update method, which is called when the simulation is in OPERATE. The Graphic Object's Data Package is sent to the Graphic Object, along with the size, when this method is called. The Graphic Object will unpack its data and make its graphic update appropriately. The third method is the hold method, which is called when the simulation is in HOLD.

Sample Graphic Objects

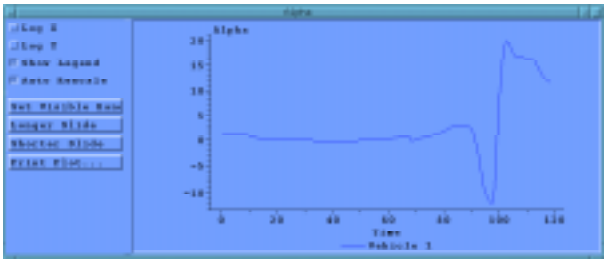

Figure 7 - Strip Chart Graphic Object

The Strip Chart Graphic Object (Figure 7) was designed to plot data sent from the simulation program. This Graphic Object will plot data on its drawing area, updating the plot whenever new data arrives. The Strip Chart Object is also capable of displaying multiple plots of data on the same drawing area. Each plot can be drawn with a distinctive line pattern to differentiate the separate plots. The main widget used for this Strip Chart Graphic Object was provided by Fermi National Laboratory[2].

The Mode Indicator Graphic Object (Figure 8) presents the user with the current time of the simulation run, the mode of the simulation, and the number of vehicles currently in the simulation. The mode the simulation is in is indicated by a red box. In Figure 8, the simulation is in Operate.

The OpenGL 3D View Graphic Object (Figure 9) draws the simulation aircraft in their correct orientation and allows the user to move the viewpoint around these aircraft. By moving the viewpoint, the user can gain new insight into what is actually occurring with the simulation aircraft. The user can also easily understand the physical orientation of the vehicles and relative positions to one another. The OpenGL graphics system is a software interface to graphics hardware that runs on multiple platforms[3]. The software for this display was originally taken from the VISION software package[4]. User configurable ground traces, altitude traces, aircraft snapshots, and multiple viewpoints are just some of the new features that have been added.

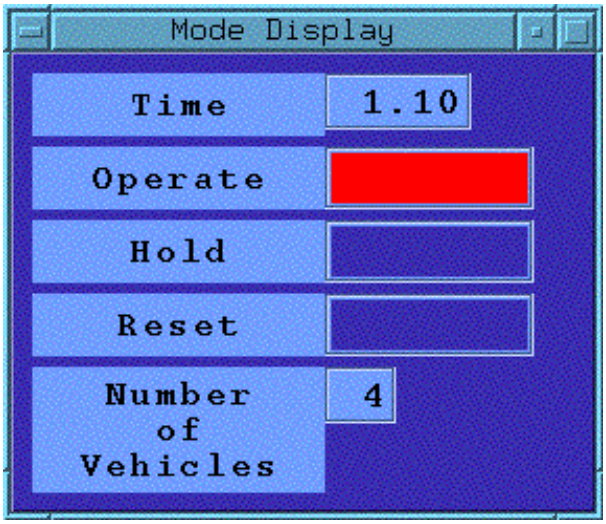

Figure 8 - Mode Indicator Graphic Object

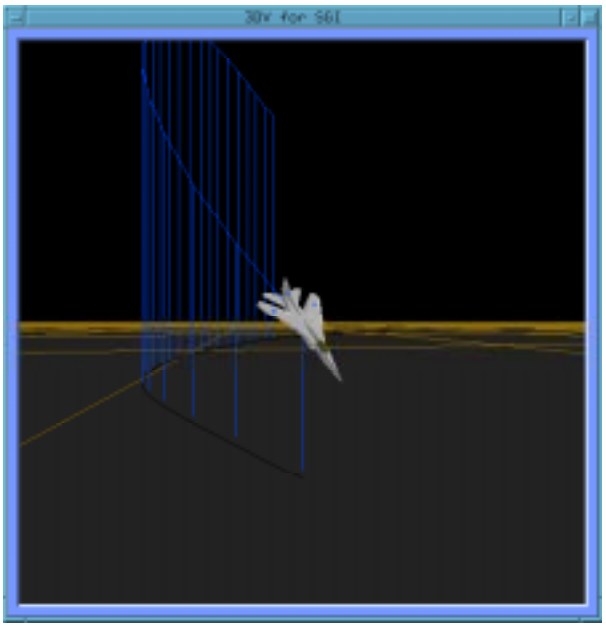

Figure 9 - OpenGL 3D View Graphic Object

The X Window System 3D View Graphic Object (Figure 10) was designed as an alternative for the OpenGL 3D View Graphic Object. The main difference between the two Graphic Objects is the 3D View

Graphic Object utilizes the X Window System for its drawing routines as opposed to the OpenGL Graphics Language. While the X Window System does not have the speed or the complexity of OpenGL, it does run on a greater number of platforms[5]. The LibV Graphics Library used to draw this object was taken from ACM, a multi-player flight application developed by Riley Rainey[6].

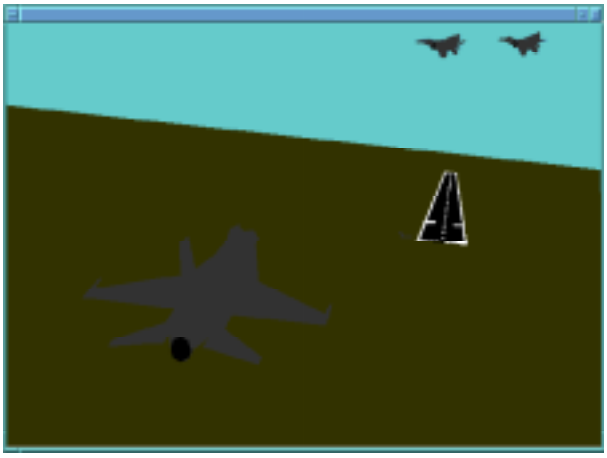

Figure 10 - X Window System 3D View Graphic Object

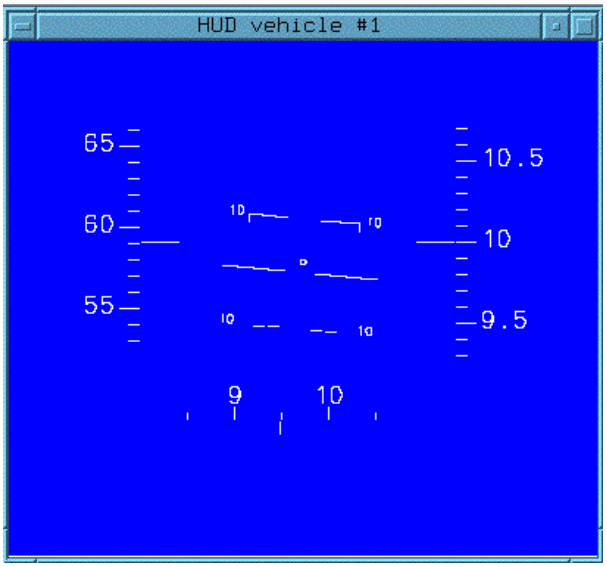

Figure 11 - Heads-Up Display Graphic Object

Figure 12 - OpenGL Heads-up Display Graphic Object

The Heads-Up Display (HUD) Graphic Object (Figures 11 and 12) displays a representative view of the pilot's HUD. HUDs typically give information about the current state of the vehicle, such as heading, pitch, roll, altitude, and velocity. Additional information, such as weapon status, fuel state, and rates of closure to other aircraft, is sometimes included and can easily be added to the HUD Graphic Object. The object shown in Figure 11 utilizes the LibV Graphics Library to draw its information. The object in Figure 12 uses the OpenGL graphics system.

The Out-of-the-Cockpit View (OCV) Graphic Object (Figure 13) gives the user a representative view of what the pilot is seeing. The OCV Graphic Object plots the locations of the simulated vehicles and positions the eyepoint at one of these locations. The resulting view is then shown. Two separate OCV Graphic Objects are available. The Graphic Object shown in Figure 13 is the OCV-HUD Graphic Object. This object has this name because a Heads-Up Display is superimposed upon the OCV View. An additional OCV Graphic Object is available without the HUD and is referred to as the OCV Graphic Object. This object also utilizes the LibV Graphics Library.

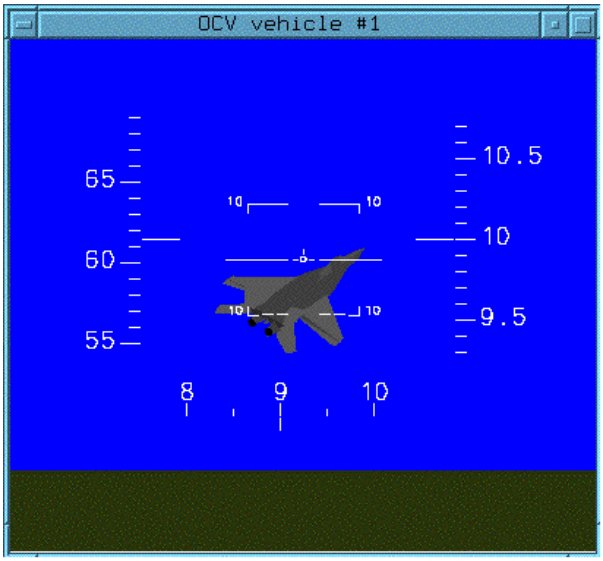

Figure 13 - Out-of-the-Cockpit View Graphic Object

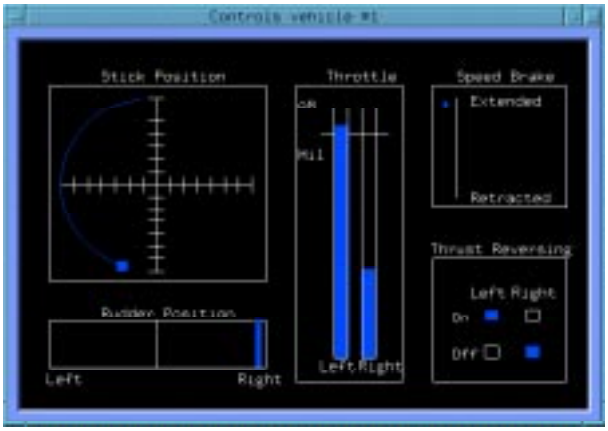

Figure 14 - OpenGL Cockpit Controls Graphic Object

The Cockpit Controls Graphic Object (Figure 14) displays a current view of the cockpit controls. The stick position display has a ten second trace. Some of the information displayed includes stick, rudder, throttle, and speed brake positions.

The OpenGL Aspect Circle Graphic Object (Figure 15) shows the three dimensional perspective view of the other aircraft involved in the simulation in relation to the current aircraft. One circle shows the location of the other aircrafts in vertical relation to the current aircraft (i.e., are they above or below the current aircraft). The other circle shows the location of the other aircraft in horizontal relation to the current aircraft (i.e., are they to the left or the right of the current aircraft). Both circles give information regarding whether the other aircraft are behind or in front of the current aircraft. In Figure 15, there are two aircraft at the same altitude as the present aircraft. One is on the right side of the present aircraft and the other is on the left side.

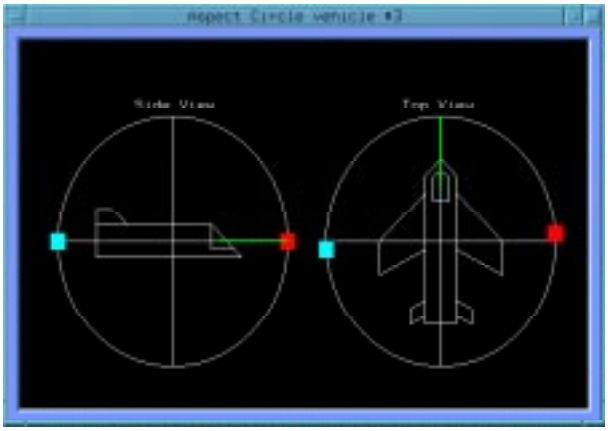

Figure 15 - OpenGL Aspect Circle Graphic Object

Graphic List

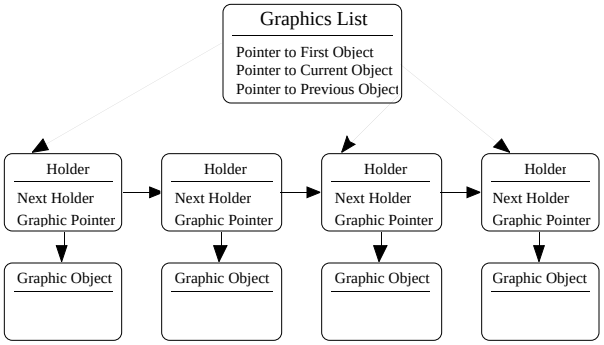

Figure 16 - Overview of Graphics List

The last portion of the SimGraph framework is the Graphics List (Figure 16). The Graphics List is a linked list that is composed of Graphic Objects. Graphic Objects are added to this list when the user requests a new graphic to be displayed on the screen.

Data Flow

Figure 17 shows the high level data flow inside of SimGraph. When the user requests for a new Graphic Object to be displayed on the screen in step 1, the interface passes the message to the Communication Object. The Communication Object transmits the request to the real time program in step 2. The real-time simulation program packs the data for the new Graphic Object along with the data for all of the previously requested data. In step 4, a new Data Block

is sent to the SimGraph Computer. The Communication Object reads in this Data Block in step 5. The Data Block is then passed to the Data List to be parsed and split into individual Data Packages in step 6. Step 7 finishes the data flow when the Graphic List updates each of the graphics by passing the requested data.

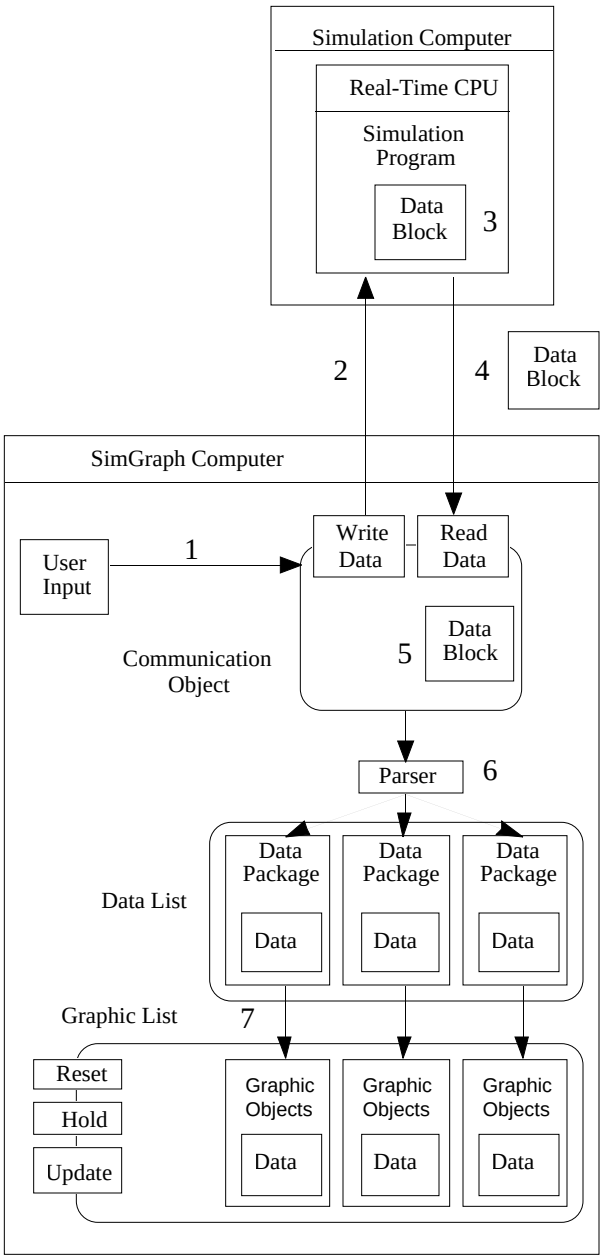

Figure 17 - High Level Data Flow

User Interface

The first window that is presented to the user is the Communication Selection Window (Figure 18). This window allows the method of communication between SimGraph and the simulation program to be selected.

Figure 18 - Communication Selection Window

If any option other than Replay is selected from the Communication Selection Window, the Main Option Window in Figure 19 will be shown. This window allows the user to save data, open Graphic Objects, close Graphic Objects, load (or save) User Configurations, and quit the program. This window stays on the screen for the duration of the SimGraph program.

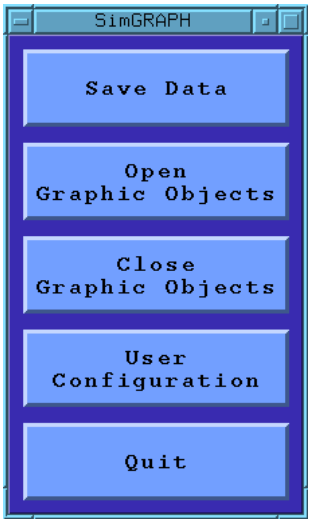

Figure 19 - Main Option Window

The Save Data Button in Figure 19 allows access to the Save Data Window (Figure 20). This window enables the user to start or terminate saving data to a file.

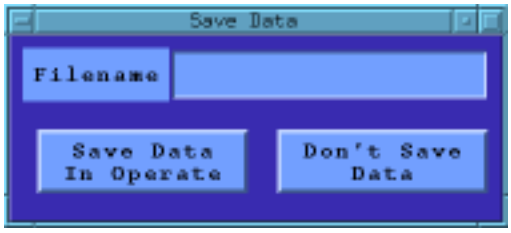

Figure 20 - Save Data Window

The Open Graphic Objects Button in Figure 19, when selected, presents the user with the Open Graphic Objects Window (Figure 21). The Open Graphic Objects Window is the only method for the user to select Graphic Objects to be presented. This window will change depending upon which Graphic Objects are available.

Figure 21 - Open Graphic Objects Window

Different actions can occur based upon the Graphic Object which is selected from the Open Graphic Objects Window. If the Graphic Object selected depends upon the number of vehicles in the simulation, a window similar to the HUD Selection Window (Figure 22) is presented to the user. In the case of Figure 22, the simulation has two vehicles in it. If there were four vehicles in the simulation, the HUD Selection Window would present the user with four buttons, HUDs for vehicles one through four. If the Graphic Object does not show an individual vehicle status but instead presents information about the entire simulation, such as the Mode Display, then pressing the button on the Open Graphic Objects Window would cause that Graphic Object to be created. Selecting the Strip Chart Button causes SimGraph to present the user with the Strip Chart Selection Window (Figure 23). The window that the user selects from (i.e., the HUD Selection Window or the Open Graphic Objects Window) disappears after a button is pressed.

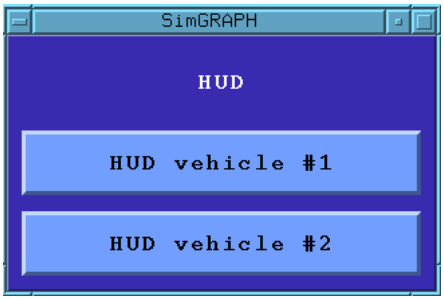

Figure 22 - HUD Selection Window

Figure 23 - Strip Chart Selection Window

The Close Graphic Objects Button on the Main Option Window (Figure 19) allows the user to access the Close Graphic Objects Window (Figure 24). The window presents the user with a list of the Graphic Objects that

are currently open. The user can select one of the Graphic Objects and then select the Close Object Button. The Graphic Object will be removed from the screen.

Figure 24 - Close Graphic Objects Window

Selection of the User Configuration option from the Main Option Window (Figure 19) causes the User Configuration Window (Figure 25) to be presented to the user. This window allows the user to save and load configuration files. These configuration files consist of the Graphic Objects that are open and their positions on the screen.

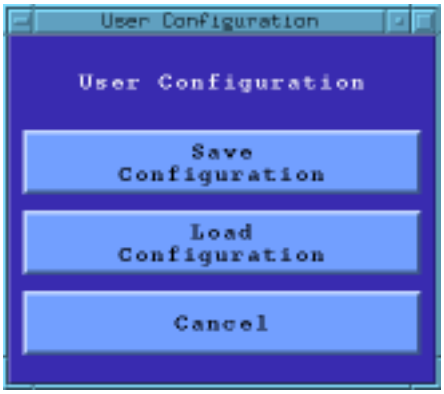

Figure 25 - User Configuration Window

The Replay Button on the Communication Selection Window (Figure 18) allows the user to examine data from a previous SimGraph session. The data may be viewed either forward or backward at various speeds. All the information for the displays will be loaded from a data file instead of being transmitted from the simulation program. After the user selects a data file, the program replaces the normal menu system with the Replay Control Window (Figure 26).

Figure 26 - Replay Control Window

If the user wishes to open Graphic Objects to be viewed during the replay, the Open Button on the Replay Control Window must be selected. Activating the Open push-button causes the Open Graphic Object Box (Figure 27) to appear. The user may then select which Graphic Objects will be displayed on the screen.

Figure 27 - Open Graphic Object Box

To close an open Graphic Object, the Close Button on the Replay Control Window must be selected. Activating the Close Button causes the Close Graphic Object Box (Figure 28) to appear. The Graphic Objects

that are currently open will be shown in the scrollable list area.

Selecting the Config Button from the Replay Control Window (Figure 26) will cause the User Configuration Window (Figure 25) to be displayed. The remaining buttons on the Replay Control Window dictate how the data is to be replayed. If Normal is selected, the data will be replayed in real-time, if possible. If Slow is selected, the data be shown at one fifth of real-time speed (i.e., the time between frames will be multiplied by five). If Fast is selected, the data will be shown as fast as possible. The next panel of buttons is used to indicate replay direction. Forward and Backward allow the user to move through the data in the selected directions. The bottom panel of buttons allow the user to begin playing data (Proceed), halt the playing of data (Stop), and exit SimGraph (Quit). The currently selected option will be highlighted in red.

Figure 28 - Close Graphic Object Box

Conclusions

The increasing complexity of modern flight simulation has created the need for tools to aid in the visualization of the vast amount of data that is produced. SimGraph was created to aid users in data interpretation and allow for rapid modification to handle future requirements. The modular design of the program allows the user to pick and choose from a library of graphical displays that will meet the needs of the research being conducted.

The Graphic Objects discussed were used to evaluate a prototype SimGraph. The first production model of SimGraph is to be utilized by the Transport Research Facilities at NASA Langley Research Center. Several new Graphic Objects are currently under production for this simulation.

While this system was specifically designed to work with the flight simulation software at NASA Langley Research Center, it is not exclusive to that research environment. It is possible to use the framework that was developed and create new Graphic Objects to adapt the software to individual environments. The authors welcome inquiries for additional uses of the system.

References

1. Crawford, D. J.; Cleveland II, Jeff I. "Langley Advanced Real-Time Simulation (ARTS) System", Journal of Aircraft, Volume 25, Number 2, February 1988, Pages 170-177.

2. Edel, Mark. "Histo-Scope Plotting Widget Release 4" ftp://ftp.fnal.gov/pub/histoscope/v4_0/histosgi.tar (4 Feb. 1995)

3. OpenGL Architecture Review Board, OpenGL Programming Guide The Official Guide to Learning OpenGL, Release 1. Reading, Massachusetts, 1993.

4. Dare, A.; Burley, J. "Pilot/Vehicle Display Development From Simulation To Flight", AIAA 92-4174, August 1992.

5. O'Reilly, Tim; Nye, Adrian X Toolkit Intrinsics Programming Manual, Motif Edition, Volume 4, November 1992.

6. Rainey, Riley. "ACM Version 4.8. The Aerial Combat Simulation for X11" ftp://ftp.netcom.com/pub/ra/rainey/acm/acm-4.8.tar.gz (10 Apr. 1997)