NASA Technical Memorandum 110308

Development of a Flight Simulation Data Visualization Workstation

Joseph A. Kaplan and Ronnie Chen
Langley Research Center
Hampton, Virginia

Patrick S. Kenney, Christopher M. Koval, and Brian K. Hutchinson
UNISYS Corporation
Hampton, Virginia

December 1996

National Aeronautics and
Space Administration
Langley Research Center
Hampton, Virginia 23681-0001

Abstract

Today's modern flight simulation research produces vast amounts of time sensitive data. The meaning of this data can be difficult to assess while in its raw format . Therefore, a method of breaking the data down and presenting it to the user in a graphical format is necessary. Simulation Graphics (SimGraph) is intended as a data visualization software package that will incorporate simulation data into a variety of animated graphical displays for easy interpretation by the simulation researcher. 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 traces the design, development and implementation of the SimGraph program, and is intended to be a programmer's reference guide.

Table of Contents

Table of Figures

1.0 - Introduction

Simulation Graphics (SimGraph) was designed to facilitate comprehension of the tremendous amount of data produced from flight simulation tests in the Flight Simulation Facility at NASA Langley Research Center. Data produced from the flight simulation is presented by SimGraph in a variety of animated graphical displays; these include 3-Dimensional Views, Heads-Up Displays, Strip Charts, and Status Indicators. Since these baseline displays may not satisfy the individual needs of the research community, SimGraph can be easily modified to accommodate the addition of new graphical displays, thereby customizing the software to the users' particular environments. SimGraph utilizes the X Window System, which runs on a variety of computer platforms, to manage its graphical displays. It can exchange data with the simulation program in a number of ways, including Internet Domain Sockets, UNIX Domain Sockets, and the Advanced Real-Time Simulation System (ARTSS)[1].

This paper is written to provide background on the development of SimGraph and is intended to serve as an addendum to the internal documentation of the software itself. Users who wish to adapt the software to their own specific needs or use segments of the code can find information about the software's workings and primary principles of design.

2.0 - High Level Design

SimGraph was developed upon a C++ object-oriented framework. This object-oriented framework handles the communication and data transfer that is standard with all of the Flight Simulation Facility's data visualization software. By developing this framework and thoroughly debugging it, the software can be continually reused. The graphics, which are usually unique to each request, are then inserted into this framework.

Object-oriented design allows a system to be viewed as a collection of objects and the interaction between those objects. This interaction occurs between their interfaces (Figure 1). 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.

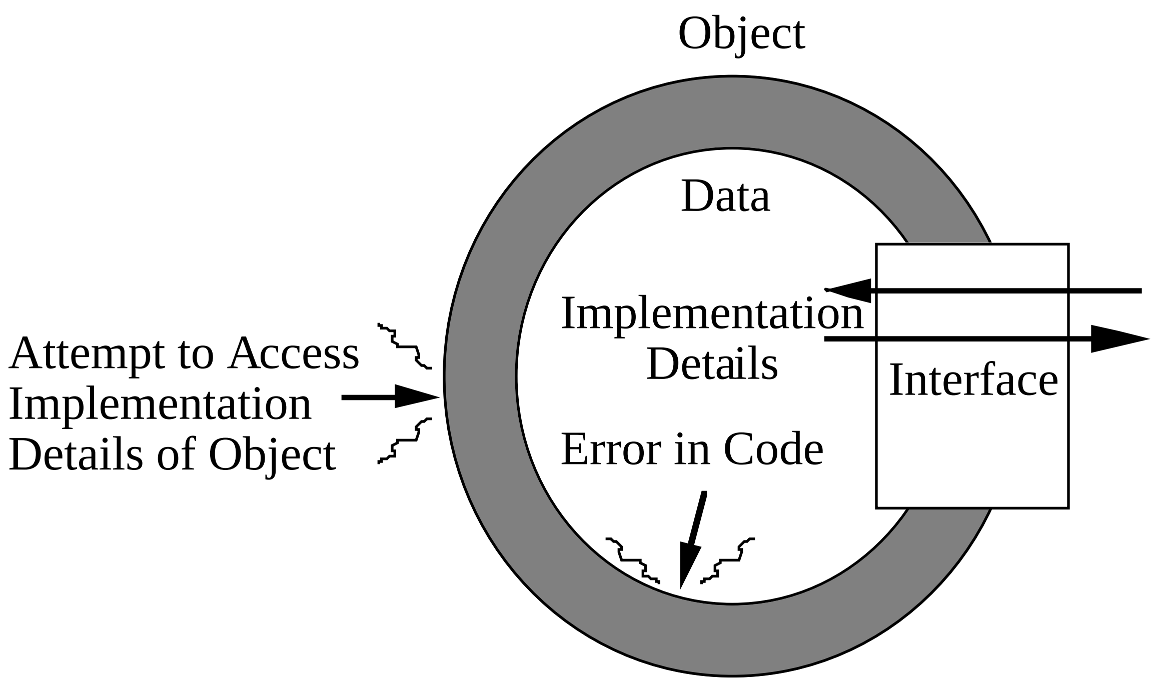


Figure 1 - Object-Oriented Design

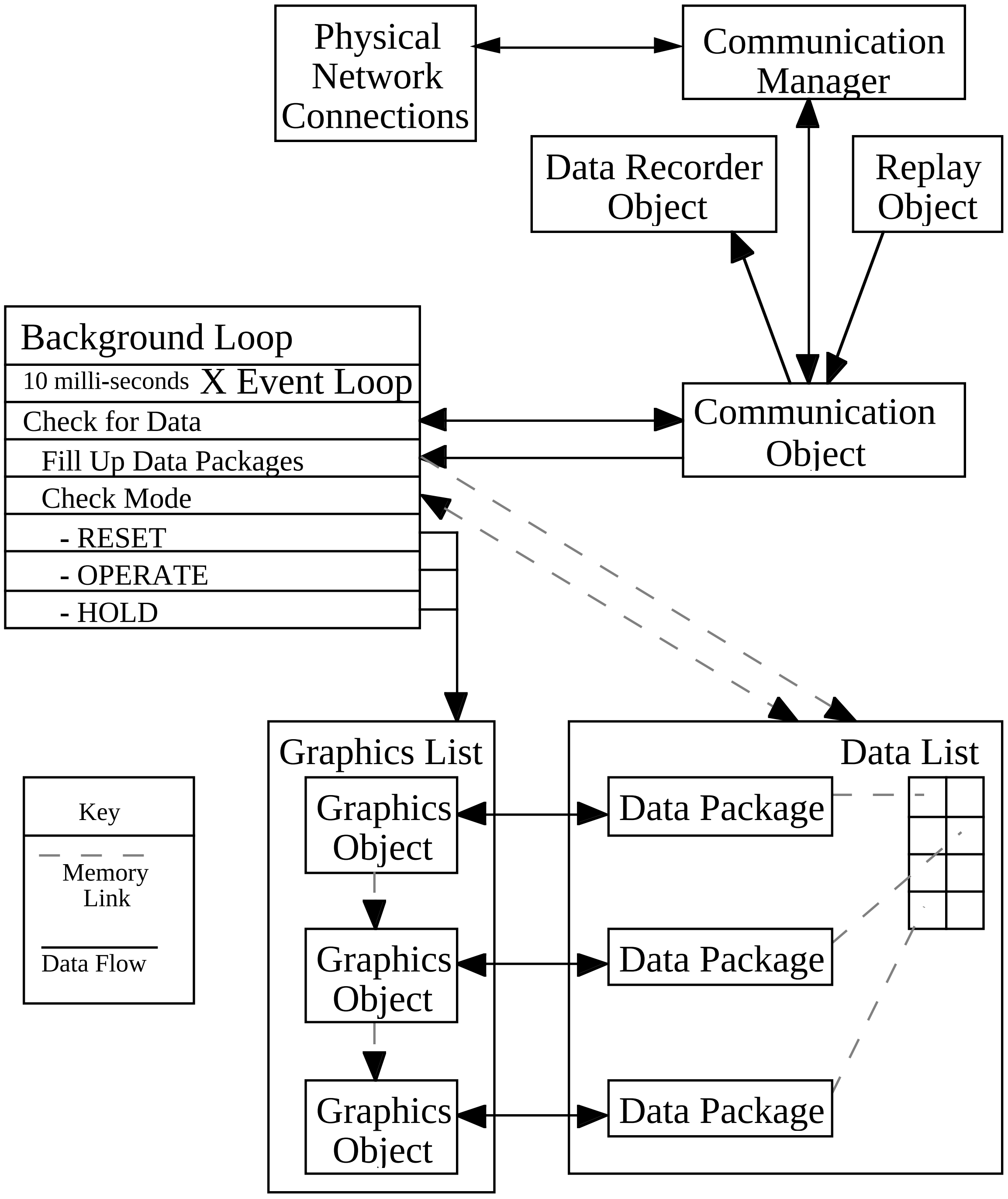


Figure 2 - Framework for SimGraph

Figure 2 shows a detailed schematic view of SimGraph. The function that drives the entire system is the background loop of the SimGraph program. At each iteration through the loop, the X Window System is given no more than 10 milliseconds to process any events that are pending[2]. When the 10 milliseconds are up or when there are no more events in the queue to process, the background loop checks with the Communication Object to determine if there is any data from the simulation program that needs to be processed. If SimGraph is connected to a simulation program, the Communication Object contacts a

Communication Network Manager Object. If data exists, the Communication Network Manager Object will inform the Communication Object, which in turn informs the background loop. If there is no data ready to be processed, the background loop returns to the top and allows the X Window System to process its pending events.

The third step of the background loop, if data exists, is to transfer the data into the Data List. The background loop once again contacts the Communication Object and asks for the current buffer of information. The Communication Object then contacts the Communication Network Manager Object and makes the same requests. This buffer of information is then transferred back to the background loop. If SimGraph is saving data from the simulation program, a Data Recorder Object will listen in on the data stream between the Communication Object and the Communication Network Manager Object and save this information to disk. If SimGraph is replaying a previously saved simulation run, any requests made to the Communication Network Manager Object are intercepted by the Replay Object and the data is transferred in from disk.

Once the data is transferred to the background loop, it is passed off to the Data List. The Data List will sort through the block of data and reform the initial data packages. Error checking also occurs at this stage to ensure that the data was correctly transmitted.

After filling up the data packages, the background loop checks for the current mode of the simulation. This is done by requesting the mode package from the Data List. The mode package is transferred from the Data List to the background loop where it is examined. The background loop will then perform certain operations depending upon the mode. If the simulation is in RESET, the Graphics List is told to reset all of the Graphic Objects. This is done by calling the reset method on those Objects. If the simulation is in HOLD, the graphics list queries each Graphic Object for its unique data package, retrieves that data package from the Data List, then passes that data package to the Graphic Object’s hold method. The same events will occur if the simulation is in OPERATE, with the exception that the Graphic Object’s operate method will be called.

3.0 - Detailed Design of SimGraph

3.1 - Communication Object

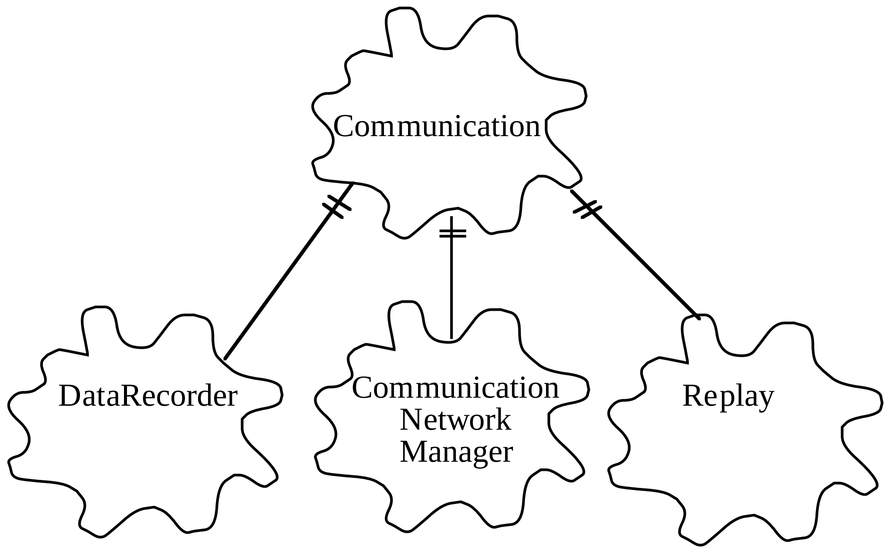


Figure 3 - Overview of Communication Object

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 researcher may choose to save the data in a file so that the data can be reviewed at a later date. When viewing this replay, the Communication Object obtains the simulation data from a file instead of a connection to the simulation program.

3.1.1 - Communication Network Manager Object

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, including: Internet Domain Sockets, UNIX Domain Sockets, and the ARTSS system. The ARTSS system is a unique 50 megabit fiber optic ring network that is installed in the Flight Simulation Facilities[3]. 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 internally. Adding new methods of communication is as simple as editing the Communication Object to handle those possibilities and recompiling the program. The rest of the SimGraph program will not be affected by those changes.

3.1.2 - Data Recorder Object

The Data Recorder Object archives the communication that occurs between the Communication Object and the Communication Network Manager Object. This includes the data from the simulation program and requests by the user to add or delete additional graphic displays. If the data for a Graphic is not being sent from the simulation program, it will be impossible for the user to view this data during replay. 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 due to the amount of memory that would be necessary to accommodate it.

3.1.3 - Replay Object

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. The user may select from a specific simulation run if multiple runs are present in the data file. All the information for the displays will be loaded from a data file instead of being communicated from the simulation program.

If the user selects the Replay Button when the SimGraph program executes, the Replay File Selection Window will be displayed (Figure 4). The user may now select a data file to open. By convention, SimGraph data files end with ".data" as the suffix. After the user selects a data file, the program will create a Replay Object. The Replay Object will open the data file for parsing. The file is parsed for two reasons. First, the data file usually needs to be cleaned up to remove multiple RESET and HOLD states. Second, the data file is broken down into multiple simulation run data files, one for each simulation run that is present in the original data file. After the file has been cleaned up and separated into different files, the Replay Object opens up the first simulation run data file and checks for the Graphic Objects that are present. After the simulation run file has been opened, the user is presented with the Replay Control Window (Figure 5).

By selecting the Next Run Button or the Previous Run Button, the user can move between the various simulation run data files. It is possible that only a single run will be present, so the user will only be allowed to view that single file. If the user can select a different run, the current simulation run data file will be closed and the new one opened. Any new simulation run data files are scanned to determine which Graphic Objects' data are present. Any currently opened Graphic Objects will be closed.

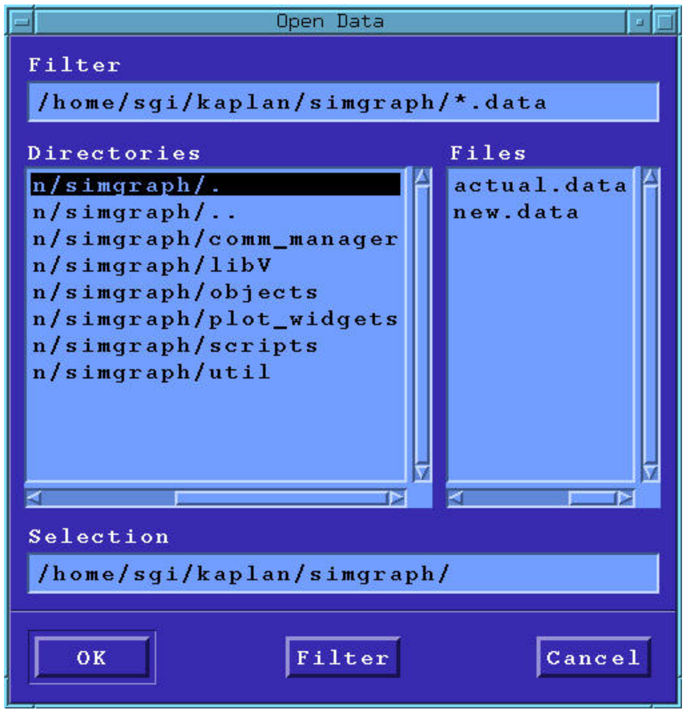


Figure 4 - Replay File Selection Window

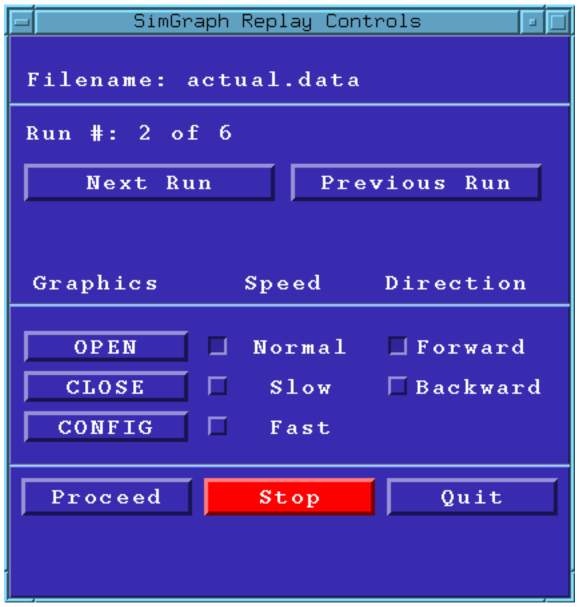
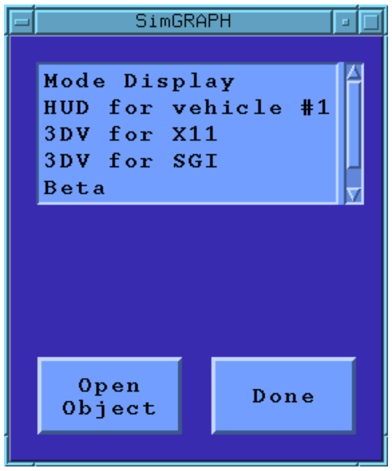


Figure 5 - Replay Control Window

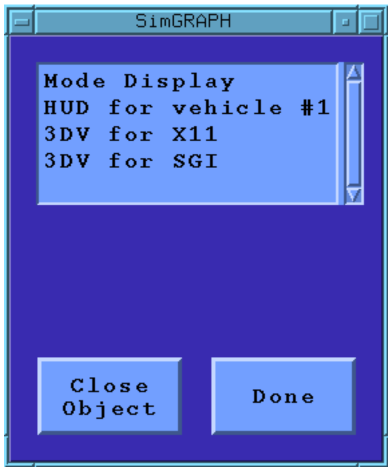
If the user would like to open Graphic Objects to be viewed during the replay, the Open Button on the Replay Control Window must be selected. Activating the Open Button causes the Open Graphic Object Box (Figure 6) to appear. The user may then select a Graphic Object and then press the Open Object Button to have that Graphic Object appear on the screen. Only those Graphic Objects being viewed during the original session will be available to the user during replay. If an Object is created and then destroyed, only the data that was sent from the simulation program for that period of time between creation and destruction will be available to the user during replay.



Mode Display
HUD for vehicle #1
3DV for X11
3DV for SGI
Beta

Figure 6 - Open Graphic Object Box

If the user would like to close a Graphic Object that is currently open during the replay session, the Close Button on the Replay Control Window must be selected. Activating the Close Button causes the Close Graphic Object Box (Figure 7) to appear. The Graphic Objects that are currently open will be shown in the scrollable list area. The user may select one of the Objects and press the Close Object Button. This will cause the Graphic Object to be removed from the screen.



Mode Display
HUD for vehicle #1
3DV for X11
3DV for SGI

Figure 7 - Close Graphic Object Box

Selection of the Config Button from the Replay Control Window (Figure 5) causes the User Configuration Window (Figure 21) to be presented to the user. The User Configuration Window is explained in detail in section 3.4. The remaining buttons on the Replay Control Window (Figure 5) dictate how the data is to be replayed. If Slow is selected, the data will update approximately twice a second. If Normal is selected, the data will update approximately 10 times a second. If Fast is selected, the data will update approximately 20 times a second. The Forward and Backward buttons allow the user to replay the data in the selected direction. 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 button will be highlighted in red.

3.2 - Data Storage

3.2.1 - Data Package

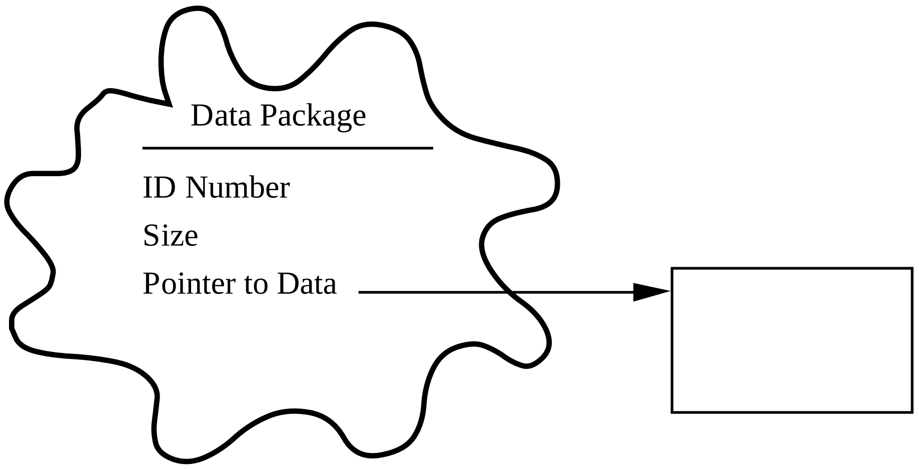


Figure 8 - Data Package Diagram

A Data Package (Figure 8) is a separate compartment that temporarily stores data for Graphic Objects. The ID number of the data details what kind of data the Data Package holds (i.e., information for a HUD, 3D View, Strip Chart, etc.). This is also the number by which the data is stored and retrieved. The size variable, which is equal to the amount of memory returned to the requesting procedure, is used when the Data Package is requested. The pointer to the actual data is initially set to NULL. When the Data Package is passed data to store, this pointer is checked to see if it is currently storing data. If the Data Package is empty, memory is allocated and then filled with the new data. If the Data Package is holding data, this area of memory is returned to the system heap. New memory is then allocated to the Data Package from the heap.

When a Data Package is requested, a pointer to constant data is returned. Since the pointer is to constant data, the Graphic Object must make a copy of the data if it is going to modify the data. This is done to ensure that the original data held in the Data Package is not corrupted by any one Graphic Object. This is important when multiple Graphic Objects are sharing one Data Package.

3.2.2 - Data List

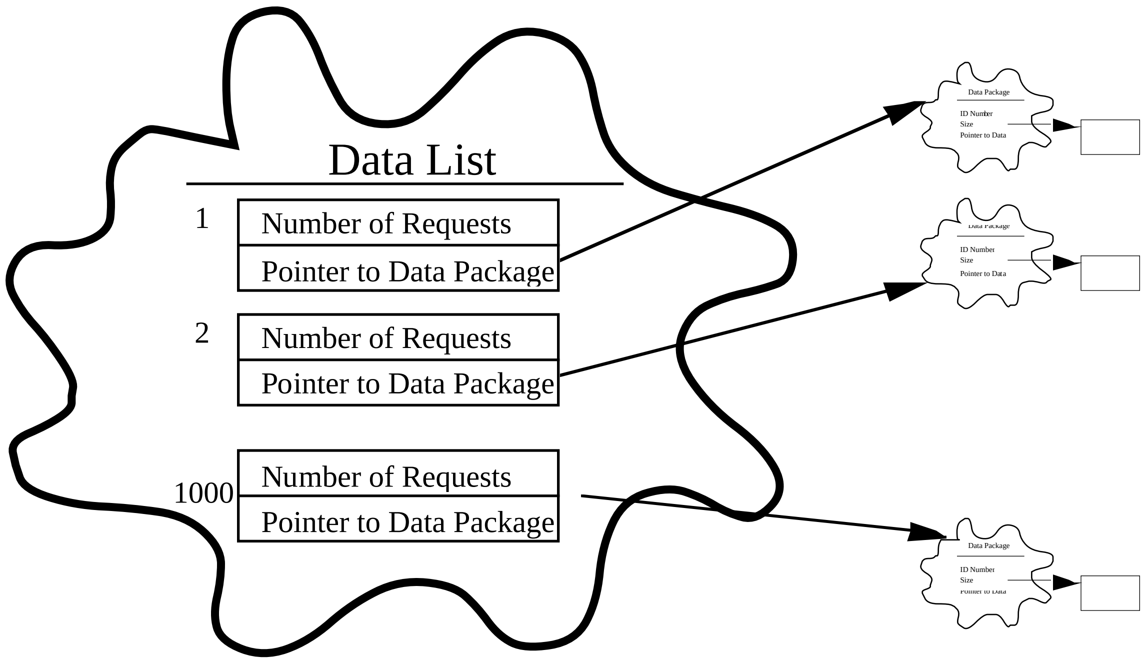


Figure 9 - Diagram of the Data List

The next portion of the SimGraph framework is the Data List. The Data List is actually a hash table that points to all of the Data Packages that are in use by SimGraph (Figure 9). The hash table not only keeps track of where the Data Packages are located in memory, but also the number of Graphic Objects that have requested the Data Package. This way, a Data Package will be transmitted only once, regardless of the number of Graphic Objects that need it. A Graphic Object can be deleted and the Data Package that corresponds to that Graphic Object will continue to be transmitted unless it was only in use by that particular Object.

When a new Graphic Object is requested, the Data List is checked for the existence of its Data Package. This is done by taking the ID number of the request and looking it up in the hash table. If the Data Package does not exist (i.e., the number of Graphic Objects requesting it is equal to zero), then the number of requests is set to one, a Data Package is created from the system heap, and the pointer is set to that Data Package. A flag is returned to the calling procedure to indicate that this Object is the first one to request the Data Package so appropriate action can be taken (i.e., send a message to the simulation program to begin sending the Data Package). If the Data Package does exist (i.e., the number of requests is one or greater), the number of requests is incremented by one. No message needs to be sent to the simulation program since the Data Package is already being sent.

When a Graphic Object is deleted, the Data List is checked to see if the Data Package is only being requested by one Graphic Object. This is done by checking the number of requests for that particular Data Package. If the number of requests is one, the memory used by the Data Package is returned to the system heap, the pointer in the hash table is set to NULL, and the number of requests is set to zero. A flag is returned that tells the calling procedure that this is the last Object requesting the Data Package so that the calling procedure may take appropriate action (i.e., send a message to the simulation program to stop sending the Data Package). If the number of requests is greater than one (i.e., the Data Package is being used by more than one Graphic Object), the number of requests is decremented by one.

The most important function of the Data List is data parsing. Data is sent from the simulation program in a large block (Figure 10). 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 fillData routine in the Data List. The fillData routine will parse through the data, ensure that it is correct and has not been corrupted during transit, and sort each of the individual Data Packages for later retrieval.

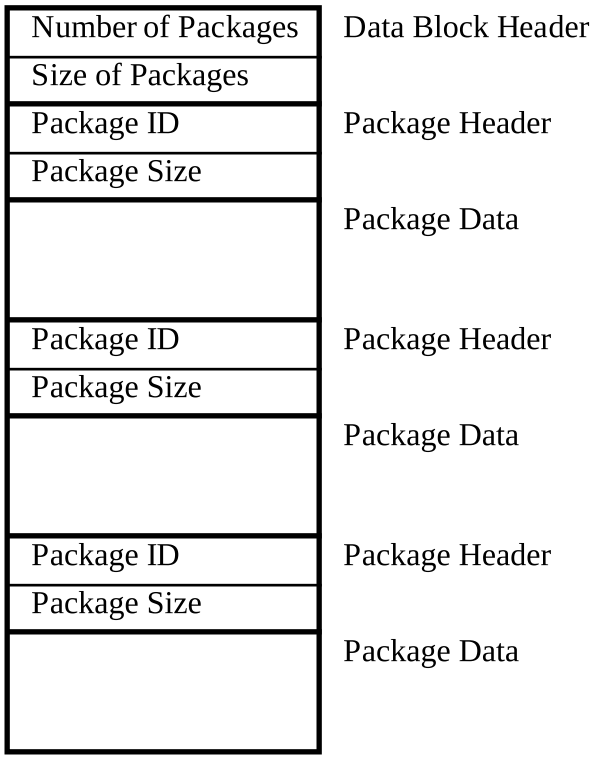


Figure 10 - Example of Data Block

The first item in the block is the Data Block Header. The Data Block Header consists of the number of packages and the size of all those packages. Based upon the number of packages, the fillData routine will begin to pull the Data Packages out of the Data Block (Figure 11). The first thing that is stripped out is the package header. The package header details the package ID number and the size of the information in the package. The Data List is checked to confirm that the package has been requested. If it has not, an error message is sent to the standard output. This error condition can occur if the Data Package has been deleted, but the simulation program has not processed the request yet. The information contained within the Data Block is then copied into the Data Package. The amount of information copied into the Data Package is dictated by the size variable given in the package header. The remaining packages are then extracted in the same manner.

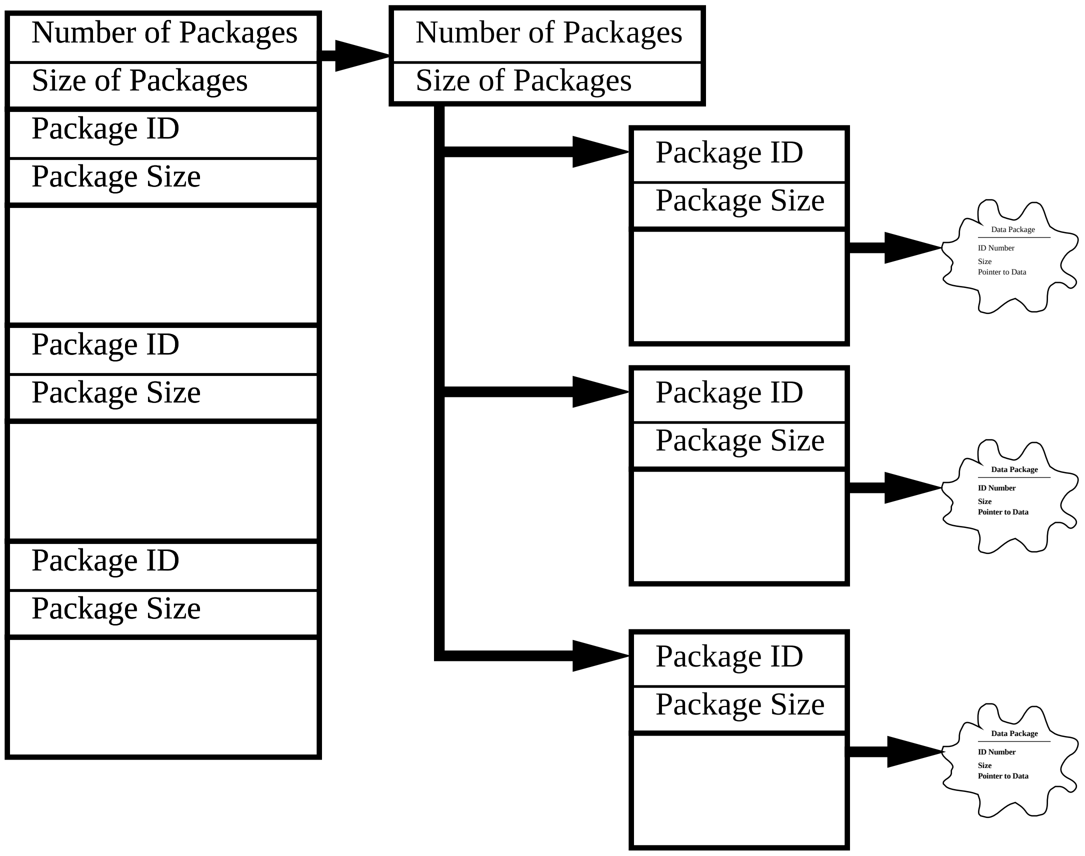


Figure 11 - Data Parsing Action

3.3 - Graphic Displays

3.3.1 - Graphic Objects

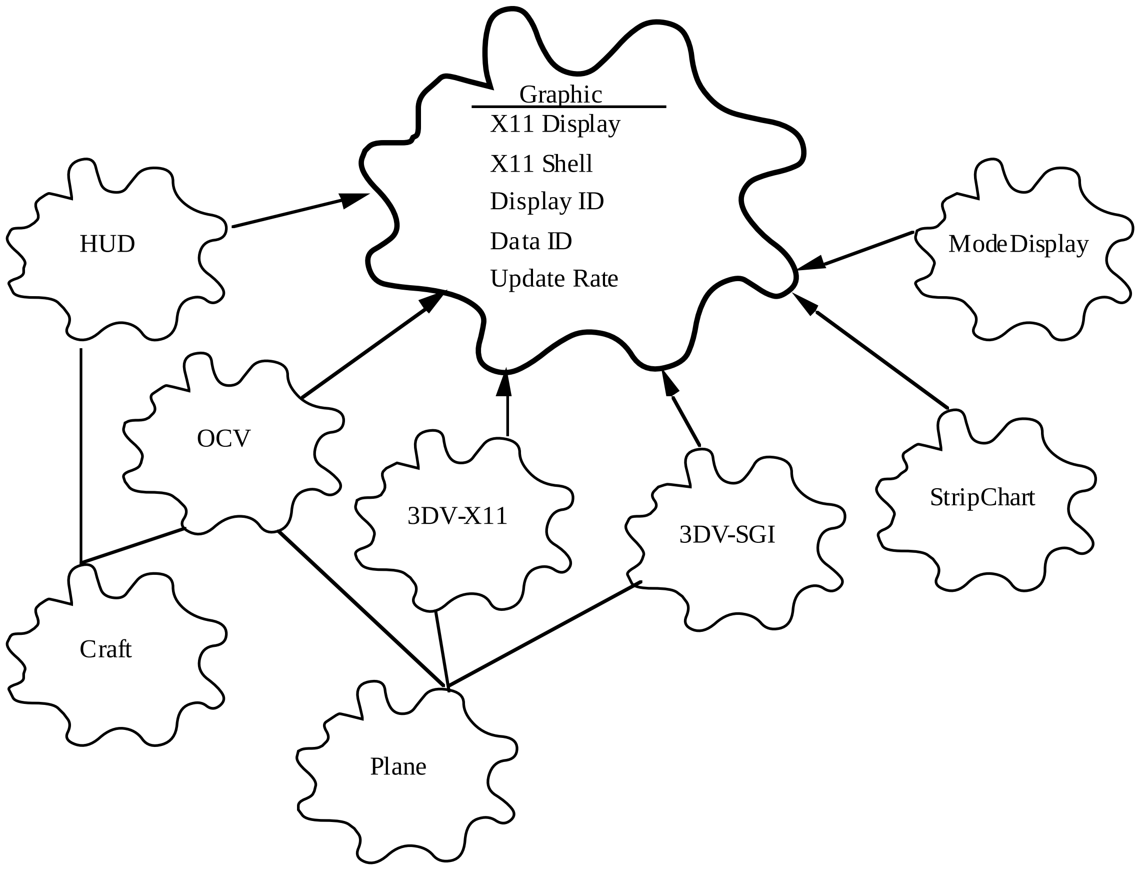


Figure 12 - Overview of Graphic Object

The Graphic Object is one of the cornerstones of SimGraph (Figure 12). 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. These attributes include an X Window System Display, an X Window System Shell, a unique Display ID, a Data Package ID, and an update rate. The X Window System Display variable is used to indicate which computer the graphic is being drawn on. An X Window System Shell is used as the container for the Graphic Object. All drawing is done inside of this shell, regardless of the graphics language used. It is possible to draw X, GL, and OpenGL code inside of an X Window System Shell. The Display ID is a unique ID number given to each Graphic Object. This is necessary when attempting to delete specific Graphic Objects. The Data Package ID represents the particular Data Package that this Graphic Object needs to update its displays. The update rate variable is currently unused.

All of the Graphic Objects have three methods in common; initialize, update, and hold. The initialize method is called when the Graphic Object is first created and during RESET. This routine will initialize all variables to their beginning states. The initialize method must be set up so that it can be called multiple times without causing any type of errors (i.e., allocating without de-allocating memory, creating Widgets without destroying them, etc.). The initialize method has no parameters associated with it. The second method that Graphic Objects have, and perhaps the most important, is the update method. The update method 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 that all Graphic Objects have is a hold method. This method is called when the simulation is in HOLD. Some Graphic Objects do not perform any type of action while in this loop, while others merely pass their arguments (data and size of data) on to the update method.

3.3.2 - Graphics List

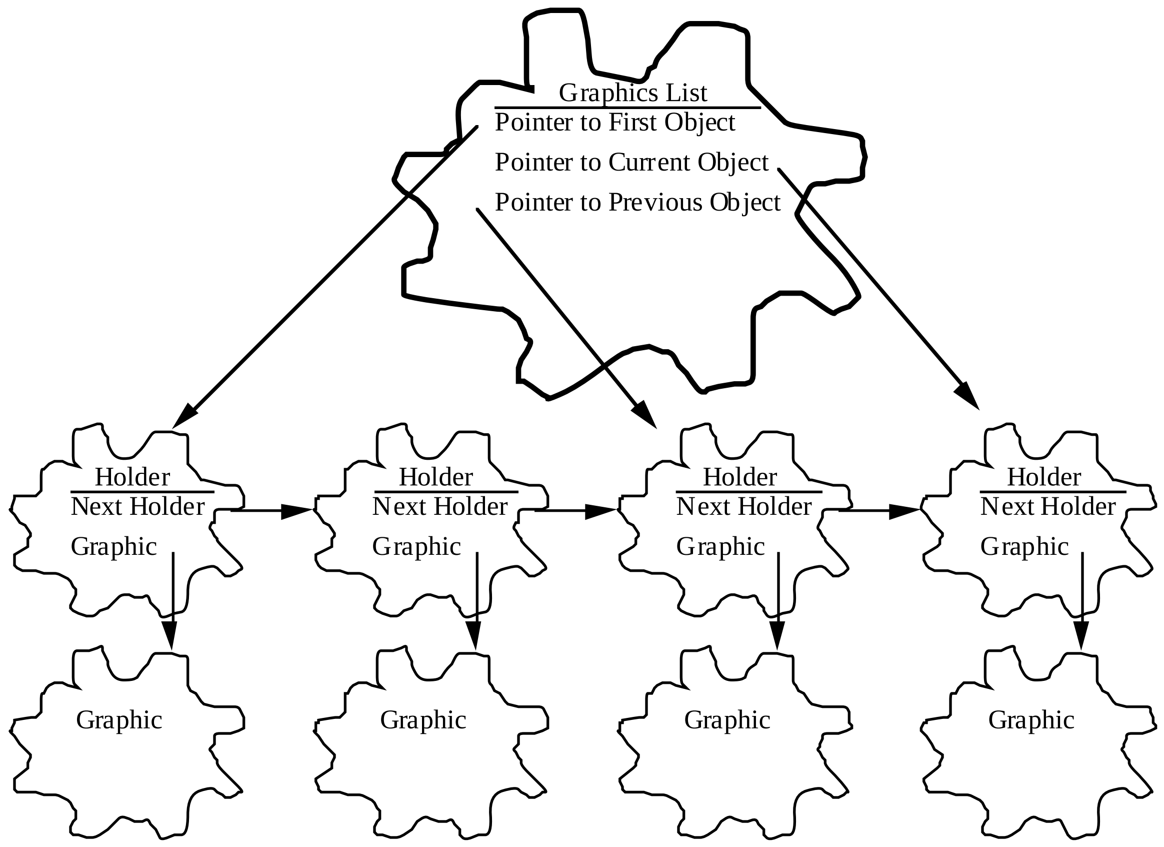


Figure 13 - Overview of Graphics List

The last portion of the SimGraph framework is the Graphics List (Figure 13). 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. Each Graphic Object is assigned a unique Display ID. The Graphic Object, once created, is inserted into the Graphics List. Removing a Graphic Object is done by using the Remove Graphic Object Button. The Graphic Object is referenced by its unique Display ID during this process. Each Display ID in the Graphic Objects will be compared to the selected one until a match is found. When a match is found, the Data Package associated with the Graphic Object will be removed, the Graphic Object will be removed from the Graphics List, and the Object will then be destroyed. The Graphic List maintains a pointer to a base Graphic Object, not the derived objects (i.e., Strip Chart, HUD, etc.). This allows new Graphics to be developed and integrated into SimGraph with minimal effort. For more information about adding new Graphic Objects, see Appendix C.

In order for the Graphics List to update all of the graphics, the Data List must completely parse the Data Block. Once that is done, the MODE Data Package is retrieved from the Data List and examined. If the simulation is in RESET, the resetAll method is called in the Graphics List Object. This method goes through all of the Graphic Objects and calls their initialize method. If the simulation is in HOLD, the Graphics List method holdALL is called. This method requests each Object’s Data Package ID. This is the ID of the Data Package that is needed to update the graphics. This Data Package is retrieved from the Data

List and passed to the Graphic Object's hold method. This is done for all of the Graphic Objects in the Graphics List. If the simulation is in OPERATE, the updateAll method is called. This method requests from each Object its Data Package ID. This is the ID of the Data Package that the Graphic Objects need to update their Graphics. This Data Package is retrieved from the Data List and passed to the Graphic Object's update method.

3.4 - User Interface

The Graphical User Interface for SimGraph is easy to use. The user is presented with a series of windows that offer different options. The first window is the Communication Selection Window (Figure 14). This window allows the user to select the method of communication between SimGraph and the simulation program. When the user selects a method, the window disappears. The details for the Replay Button have already been given under the Replay Object.

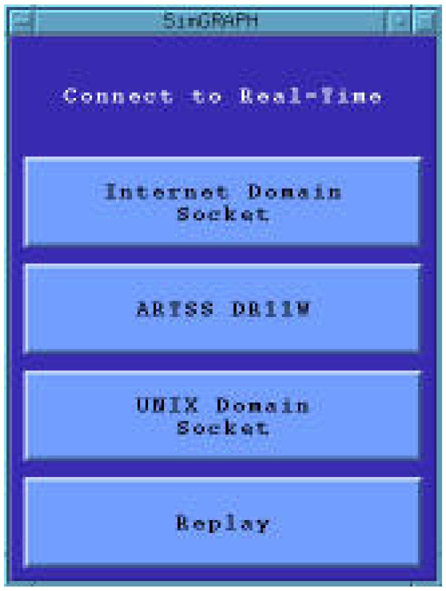


Figure 14 - Communication Selection Window

If the user selects Internet Domain Socket, UNIX Domain Socket, or ARTSS from the Communication Selection Window, the Main Option Window (Figure 15) will be shown. This window allows the user to save data, open Graphic Objects, save Graphic Objects and quit the program. This window stays on the screen for the duration of the SimGraph program.

The Save Data Button allows the user to access the Save Data Window (Figure 16), enabling the user to begin saving data to a file. A Data Recorder Object is created underneath the Communication Manager which listens to all of the communication and saves it to the disk. If the user is already saving data, then the user may cancel the saving of data by selecting the button labeled “Don’t Save Data.”

The Open Graphic Objects Button, if selected, presents the user with the Open Graphic Object Window (Figure 17). A single Object, the Display Menu Object, is created to manage this window and the data structure that accompanies it. This window is constructed based upon the contents of the “.simgraph_config” file. More details regarding the Display Menu Object can be found in Appendix D.

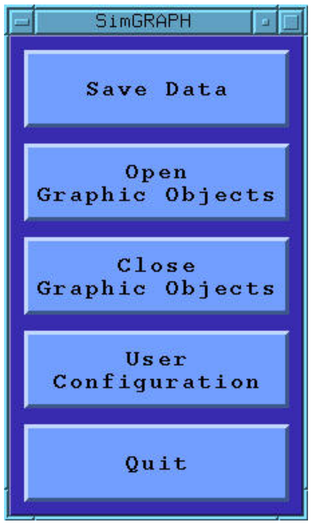


Figure 15 - Main Option Window

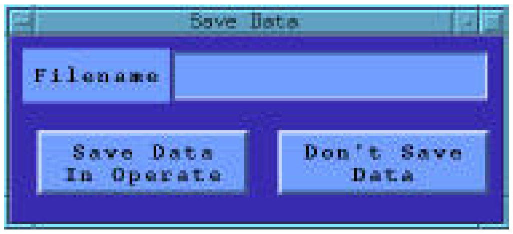


Figure 16 - Save Data Window

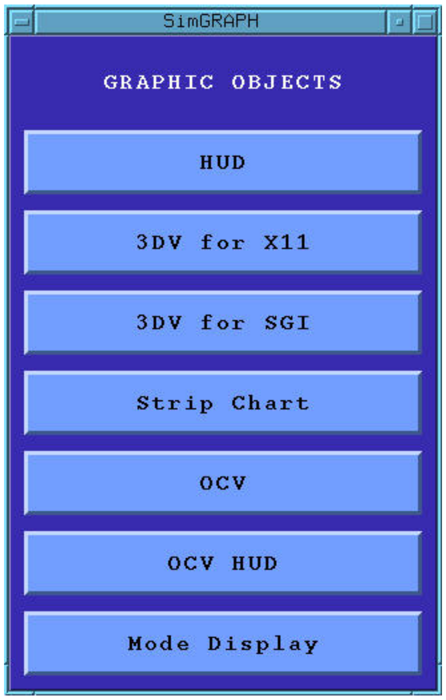


Figure 17 - Open Graphic Objects Window

Depending upon which Graphic Object is selected from the Open Graphic Objects Window, different actions occur. If a Graphic Object is selected that depends upon the number of vehicles in the simulation, a window similar to the HUD Selection Window (Figure 18) is presented to the user. In the case of Figure 18, 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. The only Graphic Object at the present time that does not follow these two examples is the Strip Chart Graphic Object. The names and number of the Strip Charts are sent over from the simulation program. Selecting the Strip Chart Button causes SimGraph to present the user with the Strip Chart Selection Window (Figure 19). 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.

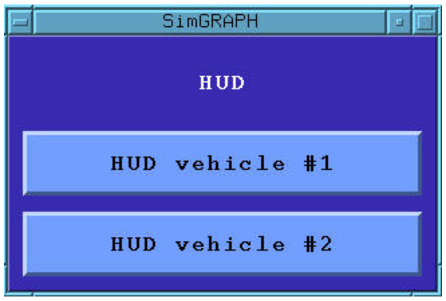


Figure 18 - HUD Selection Window

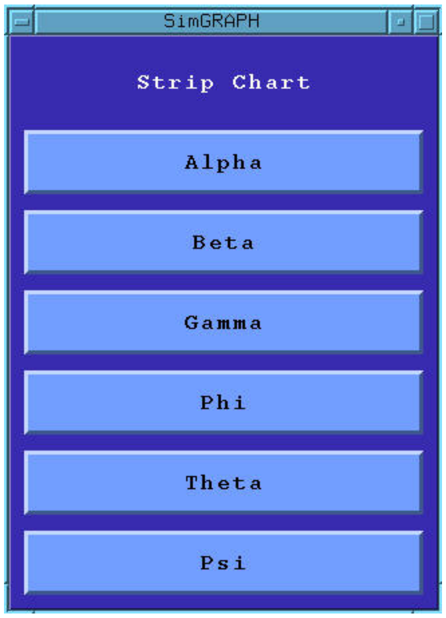


Figure 19 - Strip Chart Selection Window

The Close Graphic Objects Button on the Main Option Window (Figure 15) allows the user to access the Close Graphic Objects Window (Figure 20). This window is managed by a Graphic Selection Object which keeps track of all the currently opened graphics. When an item from the list is selected and the Close Object Button is pressed, the Graphic Selection Object will delete that item and free up its associated memory. When the user has finished closing Graphic Objects, selecting the Done Button will close the Close Graphic Objects Window.

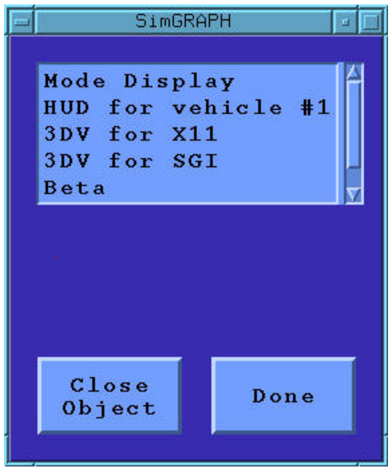


Figure 20 - Close Graphic Objects Window

Selection of the User Configuration Button from the Main Option Window (Figure 15) causes the User Configuration Window (Figure 21) to be presented to the user. This window allows the user to save and load configuration files. These configuration files consist of the open Graphic Objects and their positions on the screen. When a configuration file is opened, all of the currently opened Graphic Objects are closed. The Graphic Objects in the configuration file are then opened and positioned on the screen.

The name of the configuration file ends with ".config" by convention. There is the possibility that a configuration file will refuse to be opened if the current number of vehicles or Strip Charts does not equal the number in the configuration file. This is done to prevent SimGraph from requesting data from vehicles that do not exist.

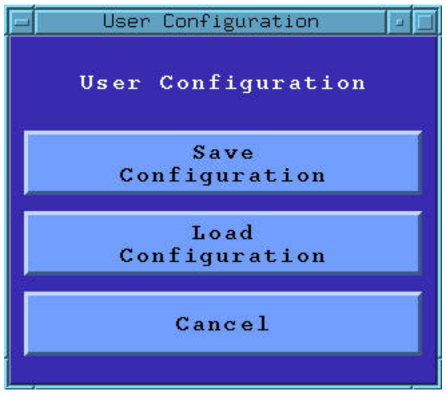


Figure 21 - User Configuration Window

4.0 - Graphic Objects

4.1 - Strip Chart

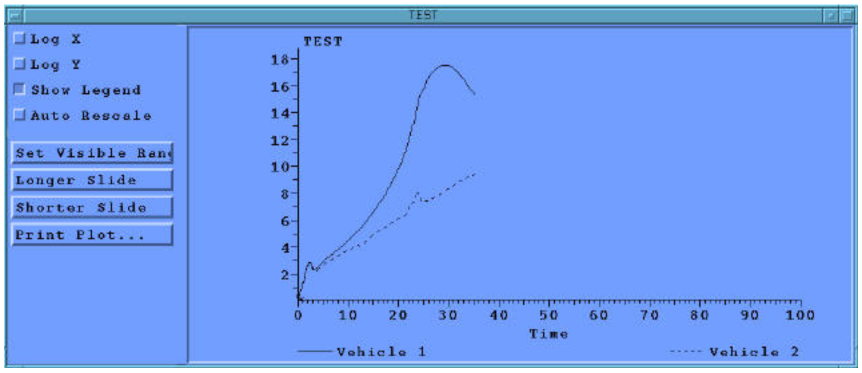


Figure 22 - Strip Chart Graphic Object

The Strip Chart Graphic Object was designed to plot data sent from the simulation program. This Graphic Object will plot data on its drawing area, updating that plot whenever new data arrives. The Strip Chart Graphic 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 files for the Strip Chart Graphic Object are: StripChart.cpp, StripChart.ipp, and StripChart.hpp. Several library files are compiled into the Strip Chart Graphic Object, including libNUtil.a and libPlotW.a. These two library files are taken from the Histo-Scope widgets, a set of six high-performance Motif Widgets for graphing and plotting[7].

When the Strip Chart Graphic Object is constructed, a large array is allocated to hold the data. An XY Plot Widget is then created inside of a form and assigned to the shell widget from the parent Graphic Object. Having created these necessary widgets, any necessary buttons are created and assigned. These buttons perform various functions, such as dumping postscript versions of the plots, changing the visible ranges, etc. The initialization function is then called, which clears all the indexing variables and zeroes out the large data storage array.

When the simulation is in HOLD, no action occurs with the Strip Chart Graphic Object. The hold method immediately returns. This can be changed if the user would like to see the data as it occurs. The results, however, of plotting non-changing time values versus changing data values are unpredictable and may not reflect the users intentions.

When the simulation is in OPERATE, the Strip Chart Graphic Object takes data passed to it and copies it into a large array. This array is then given to the XY Plot Widget and the Widget updates itself. The data is also checked to see if the XY Plot Widget needs to be rescaled. The XY Plot Widget can rescale itself by drawing all of the data. If a simulation has been running for a length of time, small fluctuations in the data may be missed because the XY Plot Widget will attempt to scale it correctly. A sliding scale has been implemented so that only the last 100 elements will be plotted on the screen. This sliding scale can be lengthened or shortened by selecting the button on the Strip Chart Graphic Object. This scaling can also be completely turned off in favor of the XY Plot Widget's built-in scaling.

When all the scaling has been taken care of, the XY Plot Widget is requested to update itself.

The first item in the Strip Chart Data Package is the Strip Chart package header, which details the Data Package ID number and the package size. Both of these are unsigned integers. The next item is the Strip Chart package. The first element of the Strip Chart package is the X-axis data value, which is a float. The X-axis value is most commonly the time value in the simulation program unless the user requests an alternate variable. The next item is the number of Y-axis elements, an unsigned integer. The Y-axis values follow the Strip Chart package. The package size in the Strip Chart package header must include the size of the Strip Chart package, plus the size of the Y-axis values that follow the package.

4.2 - Mode Indicator

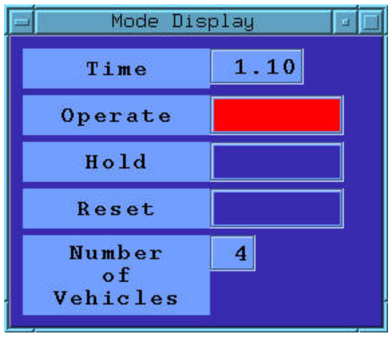


Figure 23 - Mode Indicator Graphic Object

The Mode Indicator Graphic Object's purpose is to give the user an indication of what is taking place in the simulation. It presents the time of the current simulation run, the mode of the simulation, and the number of vehicles in the current simulation. In the past, researchers would sit at the console, which allowed mode control of the simulation. If the researchers wanted to know what the time or mode of the simulation was, they only had to look at the real-time console. Now, researchers may sit in the cockpit with the pilots, so they will no longer have easy access to this information. This display was developed to give researchers access to this important information.

The creation of the Mode Indicator Graphic Object is completed by allocating several Widgets and positioning them correctly. Since this Graphic Object is completely written in Motif, it is perhaps the least complex of all the Graphic Objects. After creating the Graphic Object, the internal mode flag is initialized and the Object is put into RESET. When any of the methods are called, the mode information is determined by the method that was called, not by the information stored in the data package. This allows the Graphic Object to be correct in RESET, when normally it would receive no information from its Data Package.

The data package sent to the Mode Indicator Graphic Object is fairly simple. The first item transmitted is the Mode Indicator package header, which details the Data Package ID number and the package size. Both of these are unsigned integers. The next item is the Mode Indicator package. The first element of the Mode Indicator package is simulation time, which is a float. The next item is the current simulation mode, which is an enumerated type. The last element is an unsigned integer which gives the number of vehicles in the simulation. The package size in the Mode Indicator package header includes only the size of the Mode Indicator Package.

The Mode Data Package is used in other places within SimGraph. The Mode Data Package is used to determine the mode of the simulation, and hence, which method to call for all of the Graphic Objects. If the simulation is in HOLD, then all of the hold methods for the Graphic Objects are called. If the simulation is in RESET, then all of the reset methods for the Graphic Objects are called. The Mode Data Package is always sent from the simulation program.

4.3 - 3D View

4.3.1 - GL

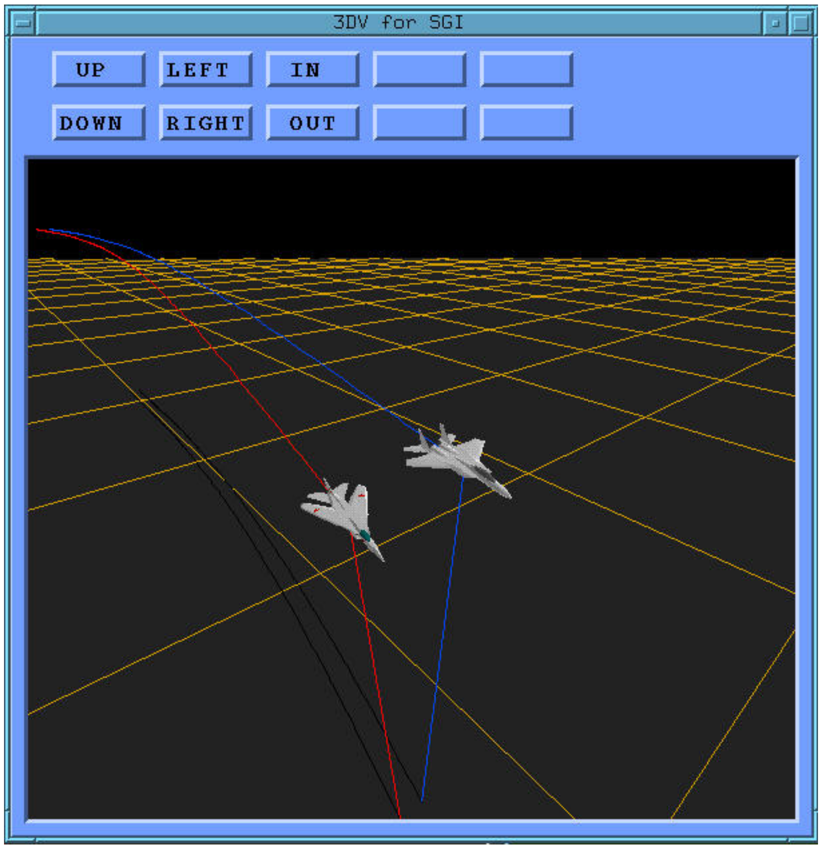


Figure 24 - SGI GL 3D View Graphic Object

The SGI GL 3D View Graphic Object plots the position of the simulated aircraft in the correct orientation and then allows the user to move the viewpoint that looks at these aircraft in space. GL is a Graphics Language that works only on Silicon Graphics, Incorporated computers. By moving the viewpoint, the user can gain new insight into what is actually occurring inside the simulation.

The main files for the SGI GL 3D View Graphic Object are: TDVsgi.cpp, TDVsgi.ipp, TDVsgi.hpp, and TDVsgi.dpp. All of the GL code was removed from the Combat Monitor[4] and the drawing was redirected to a GLX Widget[5]. A GLX Widget allows GL to be drawn into an X Widget. By drawing into the X Widget, it was possible to place the movement buttons inside the same shell as the GLX .

Creating the SGI GL 3D View Graphic Object is trivial. The GLX widget is created and placed on a form Widget. The various maneuvering buttons are then placed above the GLX widget. These buttons are hooked into their callback functions. All GL drawing is then redirected to the GLX Widget, the perspective is set, and the grid is initialized.

When the simulation is in HOLD, no action occurs with the SGI GL 3D View Graphic Object. The hold method immediately returns. This can be changed if the user would like to see the data as it occurs. The hold method returns immediately to ensure that the contrails and shadows are drawn correctly.

When the simulation is in OPERATE, the update method for the SGI GL 3D View Graphic Object is called. First, the display is set to the GLX Widget that is stored for this Graphic Object. The Data Package is then unpacked and the number of aircraft is checked to make sure that it is in the required range. Once the data is unpacked, it is transferred to the graphic method.

The data package sent to the SGI GL 3D View Graphic Object is the same Data Package that is sent to the X 3D View Graphic Object. The first item that is sent across is the SGI GL 3D View package header, which gives the Data Package ID number and the size of the data that follows. The only data that is sent in the SGI GL 3D View message is the number of aircraft that will be shown. This is sent as an unsigned integer. Immediately after this message is sent, an array of plane structures is also sent. The size of this array is included in the size of the data that is sent in the package header. The plane structure gives the current aircraft's x position, y position, z position, roll, pitch, yaw, a flag indicating whether the plane is still active, and a range variable. The roll, pitch, and yaw angles should be sent in degrees. The range variable is currently inactive, so a default value of 1000.0 is sent for that variable. All of the variables, with the exception of the alive flag which is an unsigned integer, are sent as a floating point value. This data structure is detailed in the SimGraphMessages.hpp file.

4.3.2 - X

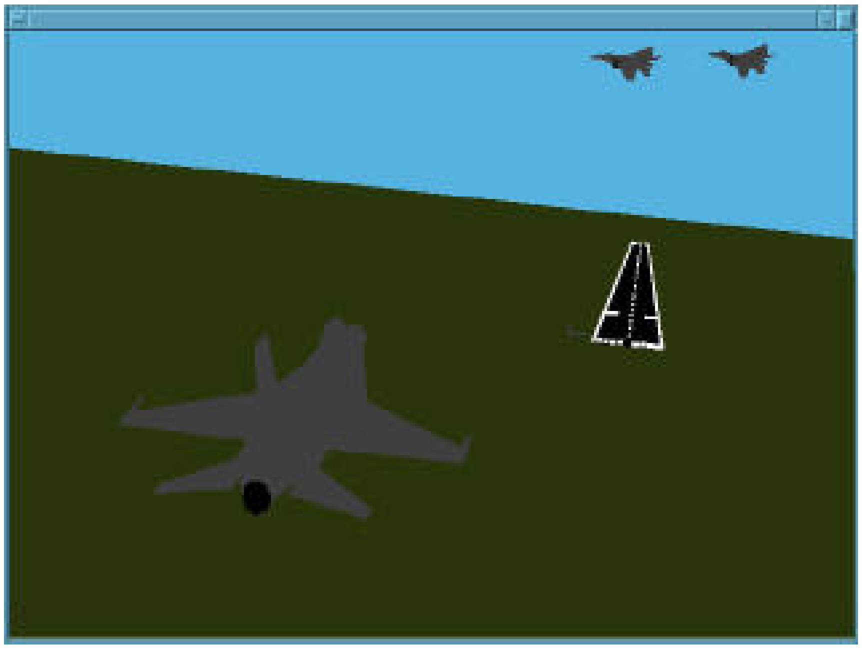


Figure 25 - X Window System 3D View Graphic Object

The X Window System 3D View Graphic Object was designed as an alternative for the SGI GL 3D View Graphic Object. The 3D View Graphic Object plots the position of the simulated aircraft in the correct orientation and then allows the user to move the viewpoint that looks at these aircraft in space. By moving the viewpoint, the user can gain new insight into what is actually occurring inside the simulation.

The main files for the X 3D View Graphic Object are: TDVx11.cpp, TDVx11.ipp, TDVx11.hpp, and TDVx11.dpp. There is one library file, libV.a, that is used by the X 3D View. Originally, the V library was intended to be included into ACM, an Aerial Combat Simulation for the X Window System[6].

Creating the X 3D View Graphic Object actually involves opening an additional shell. This Graphic Object is the only Object that has this requirement. This is necessary because the V library that is used to draw the graphics demands a simple X Window System Window for its drawing. One requirement of SimGraph is that all drawing be done inside of an X Window System Shell Widget. To make this work, it was necessary to make the V library draw inside the X Window System Shell's Window. Since the window is being directly accessed, another shell was created to hold the push buttons used by this Graphic Object. These buttons allow the user to change the viewing perspective of the eyepoint. The eyepoint is moved around the simulated vehicles while being focused on the mid-point. Once the buttons have been created, the V library is initialized to draw into the window. By initializing the V library, colors are allocated, the window is set up, and various structures (airplanes, runways, towers, etc.) are loaded into memory. These structures are then drawn on the screen to show the simulated vehicles.

The X 3D View Graphic Object performs the same functionality in its hold and update methods. This is done by having the hold method pass its parameters to the update method. In the update method, the screen is checked to determine if it needs to be refreshed. The V library only redraws the components of the display that have changed. If the display has been covered and then re-exposed, it will need to be completely redrawn. Since it is difficult to determine if this has occurred, the display is completely refreshed every ten iterations. Following this check, the data is unpacked from the data package. The mid-point between the vehicles is determined and the eyepoint is set up. The view is then drawn by positioning the vehicles and exposing the drawing buffer.

The data package sent to the X 3D View Graphic Object is fairly complex. The first item that is sent is the X 3D View package header, which gives the Data Package ID number and the size of the data that follows. The only data that is sent in the X 3D View message is the number of aircraft that will be shown. This is sent as an unsigned integer. Immediately after this message is sent, an array of plane structures is also sent. The size of this array is included in the size of the data that is sent in the package header. The plane structure gives the current aircraft's x position, y position, z position, roll, pitch, yaw, a flag indicating whether the plane is still active, and a range variable. The roll, pitch, and yaw angles are sent over in degrees. The range variable is currently inactive, so a default value of 1000.0 is sent for that variable. All of the variables, with the exception of the alive flag which is an unsigned integer, are sent as a floating point value. This data structure is detailed in the SimGraphMessages.hpp file.

4.4 - Heads-Up Display

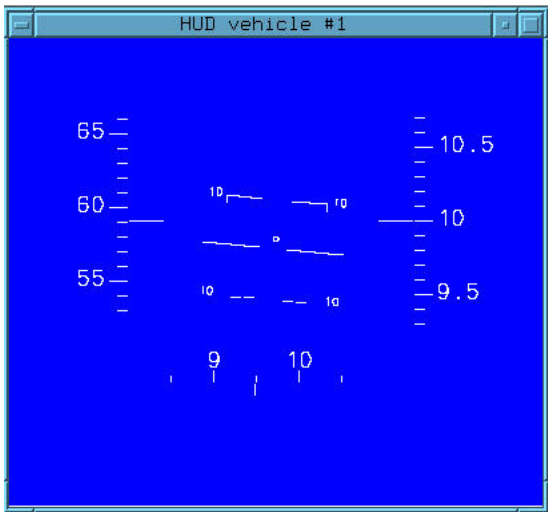


Figure 26 - Heads-Up Display Graphic Object

The Heads-Up Display (HUD) Graphic Object 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, can easily be added to the HUD Graphic Object.

The main files for the HUD Graphic Object are: Hud.cpp, Hud.ipp, Hud.hpp, Hud.dpp, Hudimath.cpp, Hudimath.hpp, and Hudscale.hpp. There is one library file, libV.a, that is used by the HUD. This is the same library file used with X Window System 3D View Graphic Object.

Creating the HUD Graphic Object is similar to the creation of the X 3D View Graphic Object, however the HUD does not require an additional shell. The Graphic Object's shell window is drawn on by the V library. By initializing the V library, colors are allocated, the window is set up, and the various scales (altitude, heading, and velocity) are loaded and initialized.

The HUD Graphic Object performs the same functionality both in it's hold or update method. This is done by having the hold method pass it's parameters to the update method. In the update method the screen is checked to determine if it needs to be refreshed. Following this check, the data is unpacked from the data package. The various pieces of the HUD (altitude and velocity scale, pitch ladder, and heading) are drawn and the drawing buffer is exposed.

The first item that is transmitted to the HUD Graphic Object is the HUD package header, which gives the Data Package ID number and the size of the data that follows. This is sent as an unsigned integer. This package header is followed by the vehicle's x velocity, y velocity, z velocity, heading, pitch, roll, altitude, and velocity which are sent in a floating point format.

4.5 - Out of the Cockpit View

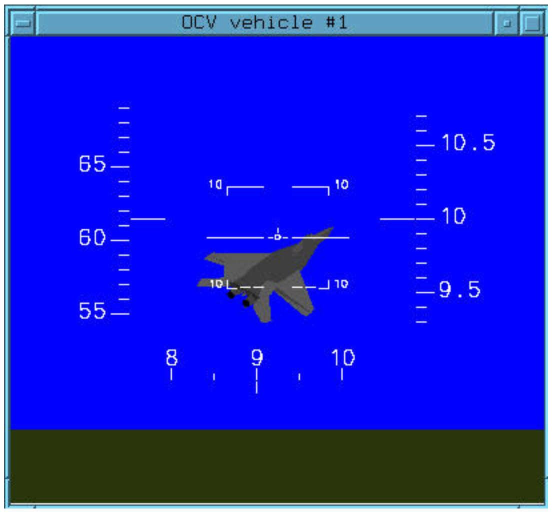


Figure 27 - Out of the Cockpit View Graphic Object

The Out of the Cockpit View (OCV) Graphic Object gives the user a representative view of what the pilot is seeing. The OCV Graphic Object plots the location 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 27 is the OCV-HUD Graphic Object. This Object has this name because a HUD is superimposed upon the Out of the Cockpit View. An additional OCV Graphic Object is available without the HUD and it is referred to as the OCV Graphic Object.

The main files for the OCV-HUD Graphic Object are: OcvHud.cpp, OcvHud.ipp, OcvHud.hpp, and OcvHud.dpp. The main files for the OCV Graphic Object are: Ocv.cpp, Ocv.ipp, Ocv.hpp, and Ocv.dpp. There is one library file, libV.a, that is used in both of the OCV Graphic Objects. This is the same library file used with X Window System 3D View Graphic Object.

The OCV Graphic Object performs the same functionality in both its hold or update method. This is accomplished by having the hold method pass its parameters to the update method. In the update method, the screen is checked to determine if it needs to be refreshed. Following this check, the data is unpacked from the data package. The mid-point between the vehicles is determined and the eyepoint is set up. The view is then drawn by positioning the vehicles and exposing the drawing buffer.

The first item that is transmitted is the OCV package header, which gives the Data Package ID number and the size of the data that follows. This is sent as an unsigned integer. This package header is followed by the number of the vehicle being used to show the out of the cockpit view, which is an unsigned integer. Next, the ownship's x velocity, y velocity, z velocity, heading, pitch, roll, altitude, and velocity are sent as in a floating point format. The number of vehicles in the simulation follows as an unsigned integer. Immediately after this message is sent, an array of Plane structures is also sent. The size of this array is included in the size of the data that is sent in the package header. The Plane structure gives the current aircraft's x position, y position, z position, roll, pitch, yaw, a flag indicating whether the plane is still active, and a range variable. The roll, pitch, and yaw angles should be sent over in degrees. The range variable is currently inactive, so a default value of 1000.0 is sent for that variable. The size of the data that is sent in the package header must take into account all of these variables.

5.0 - Conclusions

The increasing complexity of modern flight simulation has created the need for tools to aid in the visualization of the tremendous amount of data that is produced. SimGraph was created not only to simplify data interpretation but also to allow the users to adapt it to suit the requirements of their research. 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 Graphics 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. Any system capable of sending data in the specified format can drive the Graphic Objects that were discussed. It is also possible to use the framework that was developed and create new Graphic Objects to adapt the software to individual environments. The author welcomes inquiries for additional uses of the system.

Appendix A - Files

List of all files and their purpose

Graphics

Graphic Parent Object

Graphic.cpp	Function definitions for main parent object.
Graphic.hpp	Declarations for main parent object.
Graphic.ipp	Inline function definitions for main parent object.

3D View - GL

TDVsgi.cpp	Function definitions for 3D View Object written in GL.
TDVsgi.dpp	Function definitions for data files related to 3D View GL Object.
TDVsgi.hpp	Declarations for 3D View GL Object.
TDVsgicolor.hpp	Declarations for color table for 3D View GL Object .

3D View - X

TDVx11.cpp	Function definitions for 3D View Object written in X. Uses Vlib Graphics library[6].
TDVx11.dpp	Function definitions for data tables related to 3D View X Object.
TDVx11.hpp	Declarations for 3D View X Object.
TDVx11.ipp	Inline function definitions for 3D View X Object.

Heads-Up Display

Hud.cpp	Function definitions for Heads-Up Display. Uses Vlib Graphics library[6].
Hud.dpp	Function definitions for data tables related to Heads Up Display.
Hud.hpp	Function declarations for Heads-Up Display.
Hud.ipp	Inline function definitions for Heads-Up Display.
Hudimath.cpp	Function definitions used by Heads-Up Display Object.
Hudimath.hpp	Function declarations used by Heads-Up Display Object.
Hudscale.hpp	Function declarations used by Heads-Up Display Object.

Out of the Cockpit View

Ocv.cpp	Function definitions for Out of the Cockpit View Graphic Object. Uses Vlib Graphics library[6].
Ocv.dpp	Function definitions for data tables related to Out of the Cockpit View Graphic Object.
Ocv.hpp	Function declarations used by Out of the Cockpit View Graphic Object.
Ocv.ipp	Inline function definitions for Out of the Cockpit View Graphic Object.
OcvHud.cpp	Function definitions for the Out of the Cockpit View/Heads-Up Display Graphic Object. Uses Vlib Graphics library[6].
OcvHud.dpp	Function definitions for data tables related to Out of the Cockpit View/Heads-Up Display Graphic Object.
OcvHud.hpp	Function declarations used by Out of the Cockpit View/Heads-Up Display Graphic Object.

OcvHud.ipp	Inline function definitions for Out of the Cockpit View/Heads-Up Display Graphic Object.
Craft.hpp	Function declarations used in Vlib graphic routines. Modified from original file by Joseph Kaplan. Used by all Graphic Objects that inherit Vlib Graphics library[6].

Mode Display

ModeDisplay.cpp	Function definitions for Mode Display Graphic Object.
ModeDisplay.hpp	Function declarations for Mode Display Graphic Object.
ModeDisplay.ipp	Inline function definitions for Mode Display Graphic Object.

Vlib Graphics library.

Alib.h	One of the three header files for Vlib Graphics library[6].
VFont.h	One of the three header files for Vlib Graphics library[6].
Vlib.h	One of the three header files for Vlib Graphics library[6].
libV.a	Library file which contains Vlib Graphics library[6].

Strip Chart

StripChart.cpp	Function definitions for Strip Chart Graphic Object.
StripChart.hpp	Function declarations for Strip Chart Graphic Object.
StripChart.ipp	Inline function definitions for Strip Chart Graphic Object.

XY Plot Widget

libNUtil.a	Utility Library for XY Plot Widgets written by Mark Edel.
libPlotW.a	XY Plot Widget Library written by Mark Edel.

Overhead For Graphics

Opening and Closing Graphics

openGraphic.cpp	Definition of functions which open Graphic Objects.
openGraphic.hpp	Declaration of functions which open Graphic Objects.
GraphicSelection.cpp	Definition of functions for Graphic Selection Object which creates a screen for users to open or close Graphic Objects.
GraphicSelection.hpp	Declaration of functions for Graphic Selection Object which creates a screen for users to open or close Graphic Objects.

Linked List

LinkList.cpp	Definition of functions for Linked List Object which holds open Graphic Objects.
LinkList.hpp	Declaration of functions for Linked List Object which holds open Graphic Objects.

Graphics Stack

Stack.cpp	Definition of functions for Stack Object used to hold information about open Graphic Objects.
Stack.hpp	Declaration of functions for Stack Object used to hold information about open Graphic Objects.
Stack.ipp	Inline function definitions for Stack Object used to hold information about open Graphic Objects.

Communications

Communication.cpp	Definition of functions for Communication Object which handles data acquisition and storage for all graphics.
Communication.hpp	Declaration of functions for Communication Object which handles data acquisition and storage for all graphics.
libcomm.a	Library archive containing all necessary code to use CommLink Objects. This archive is maintained by UNISYS.

<u>Definition of unique data structures</u>

SimControl.hpp	Definition of enumerated Mode type.
SimGraphMessages.hpp	Declaration of data structures used throughout program.
SimGraphString.hpp	Definition of enumerated type SimGraphString.

Interface

<u>Main interface</u>

interface.cpp	Main interface file.
interface.hpp	Declaration of data structures used throughout program.

<u>Graphics Selection Menus</u>

DisplayMenu.cpp	Definition of functions for Display Menu Object.
DisplayMenu.hpp	Declaration of functions for Display Menu Object.
.simgraph_config	Configuration file for Display Menus.

<u>User Configuration</u>

userConfig.hpp	Declaration of functions for User Configuration Window.
userConfig.cpp	Definition of functions for User Configuration Window.

Data Handling

<u>Data List</u>

DataList.cpp	Definition of functions for the Data List Object.
DataList.hpp	Declaration of functions for the Data List Object.
DataList.ipp	Inline function definitions for the Data List Object.

<u>Data Packages</u>

DataPackage.cpp	Definition of functions for the Data Package Object.
DataPackage.hpp	Declaration of functions for the Data Package Object.
DataPackage.ipp	Inline function definitions for the Data Package Object.

<u>Data Storage</u>

DataRecorder.cpp	Definition of functions for the Data Recorder Object.
DataRecorder.hpp	Declaration of functions for the Data Recorder Object.

<u>Data Retrieval</u>

Replay.cpp	Definition of functions for the Replay Object.
Replay.hpp	Declaration of functions for the Replay Object.
saveData.cpp	Definition of functions for dialog boxes related to Data Recorder Object.
saveData.hpp	Declaration of functions for dialog boxes related to Data Recorder Object.
openData.cpp	Definition of functions for dialog boxes related to Replay Object.

openData.hpp	Declaration of functions for dialog boxes related to Replay Object.
Makefile	File that contains build procedures. Used with gmake.

Appendix B - Low Level Communication Protocol

1.0 - Connecting

1.1 There are two possible ways for SimGraph to connect to the Host Program:

1.1.1 Internet Domain Sockets - the two programs will exchange data over an ethernet network

1.1.2 UNIX Domain Sockets - the two programs will exchange data through a pipe

1.2 The Host Program will send down the first data block after the connection is established. This first data block will consist of an INTRO package followed by a MODE package.

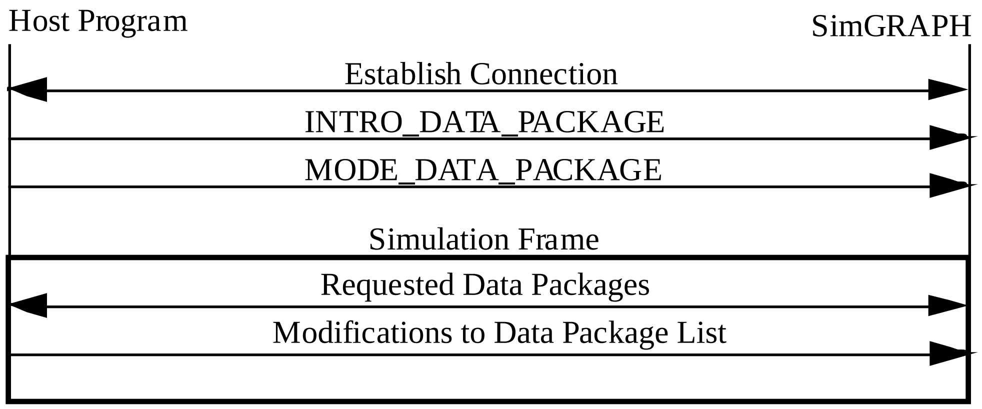


Figure 28 - Diagram of Connection

2.0 - Communicating

2.1 When data is sent from the Host Program to SimGraph, all requested data packages will be sent in a single data block.

2.2 The Host Program, once told to send a data package, will continue to send a data package every frame until told to stop.

2.3 The Host Program must always send down a MODE package. In addition, if the Host Program is in RESET, an INTRO package must be sent.

2.4 The Host Program will continue to send data to SimGraph unless an EOT message is received.

2.5 Modifications of requested data packages

2.5.1 After sending the requested data packages, the Host Program will look for any modifications coming from SimGraph. These modifications will be sent in the standard response structure.

2.5.2 The type of modification will be detailed in the response structure, either an ADD or DELETE. The data package ID number will follow.

3.0 - Data Block

The data format for a block of data from the Host Program to SimGraph is shown in Figure 29.

3.1 The first item in the data block is an InterMachine Header. This InterMachine Header is used by the SimGraph's Communication Object.

3.1.1 The first item is the block number. Initially, this number should be set to zero. Every time a new block is sent to SimGraph, it should be incremented by one.

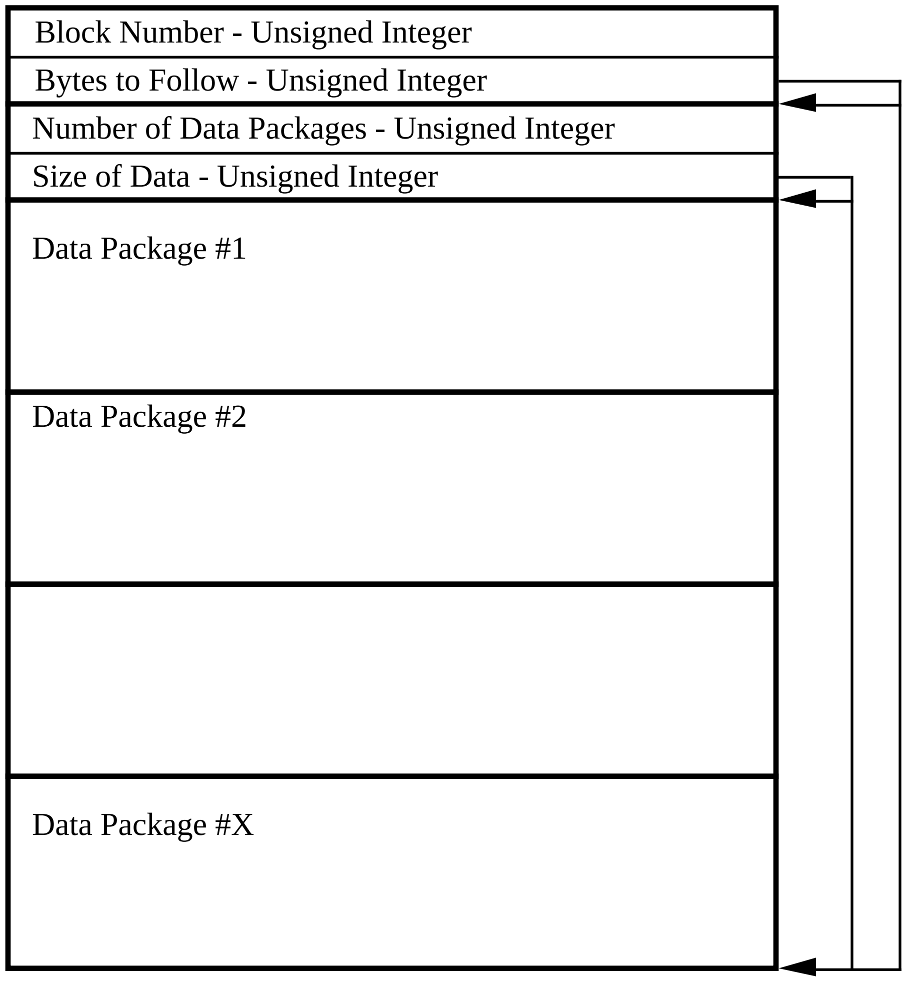
3.1.2 The second item is the remaining size of the data block, not counting the size of the InterMachine Header.

3.2 The second item in the data block is the Data Block Header. The Data Block Header is used by SimGraph.

3.2.1 The first item in the Data Block Header is the number of data packages in the current data block.

3.2.2 The second item is the total size of the data packages, including the data package headers. This size variable does not include the size of the first two items in the data block (i.e., The InterMachine Header and the Data Block Header).

3.3 The first two items in the data block are immediately followed by the individual data packages.



Block Number - Unsigned Integer
Bytes to Follow - Unsigned Integer
Number of Data Packages - Unsigned Integer
Size of Data - Unsigned Integer
Data Package #1
Data Package #2
Data Package #X

Figure 29 - Format for data block

4.0 - Data Packages

The format for the individual data packages is shown in Figure 30.

4.1 The first item in the data package is the ID number of that particular data package.

4.2 Each data package is given a unique ID.

4.3 The size of the data which follows does not include the data package ID or itself in the calculation.

4.4 The data for each particular data package will follow the size of the data.

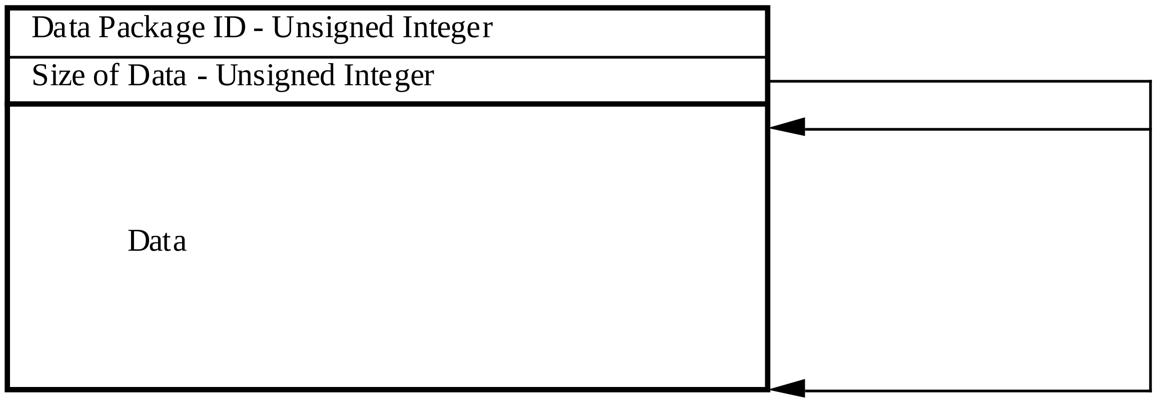

Data Package ID - Unsigned Integer
Size of Data - Unsigned Integer
Data

Figure 30 - Format for data package

5.0 - Mode Data Package

The description for the data package that goes with the Mode display is shown in Figure 31. This data package should be sent every iteration that a connection is established.

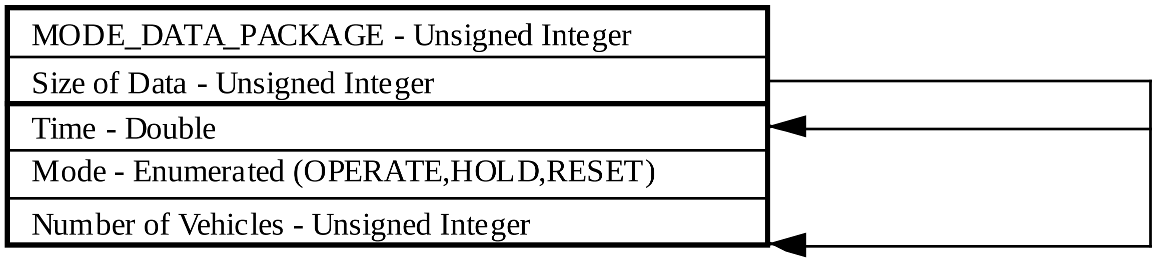

MODE_DATA_PACKAGE - Unsigned Integer
Size of Data - Unsigned Integer
Time - Double
Mode - Enumerated (OPERATE,HOLD,RESET)
Number of Vehicles - Unsigned Integer

Figure 31 - Format for Mode Data Package

6.0 - Intro Data Package

The description for the Intro data package is shown in Figure 32.

6.1 This data package should always be sent in RESET.

6.2 The Strip Chart names must be terminated by a \0.

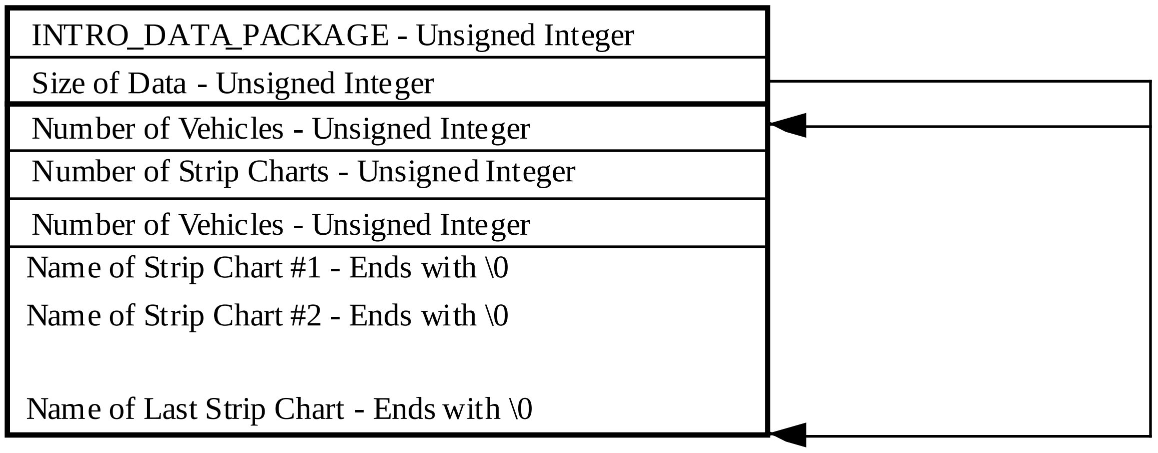

INTRO_DATA_PACKAGE - Unsigned Integer
Size of Data - Unsigned Integer
Number of Vehicles - Unsigned Integer
Number of Strip Charts - Unsigned Integer
Number of Vehicles - Unsigned Integer
Name of Strip Chart #1 - Ends with \0
Name of Strip Chart #2 - Ends with \0
Name of Last Strip Chart - Ends with \0

Figure 32 - Format for Intro Data Package

7.0 - HUD Data Package

The description for the data package that goes with the Heads-Up Display is shown in Figure 33.

7.1 The Data Package ID will vary depending on which vehicle's information is being sent to the SimGraph computer.

7.2 Multiple HUD data packages may be sent.

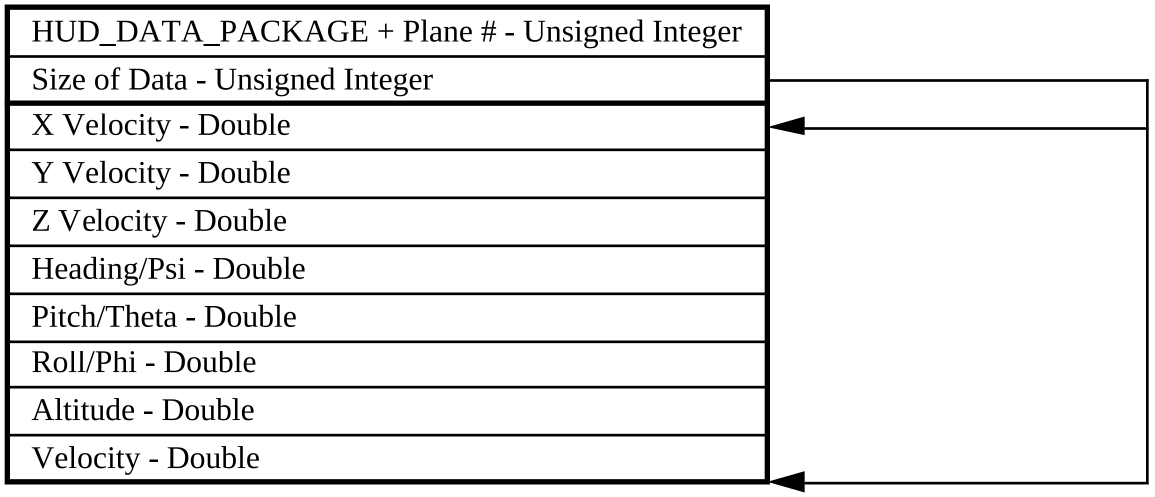


Figure 33 - Format for the HUD Data Package

8.0 - Strip Chart Data Package

The description for the data package for the Strip Chart Display is shown in Figure 34.

8.1 The Data Package ID will vary depending on which strip was requested by the user.

8.2 Multiple Strip Chart data packages may be sent.

8.3 The X axis variable should always correspond to the time variable.

8.4 The Y variables should always be sent in the same order.

8.5 Variables should only be added or deleted while in RESET.

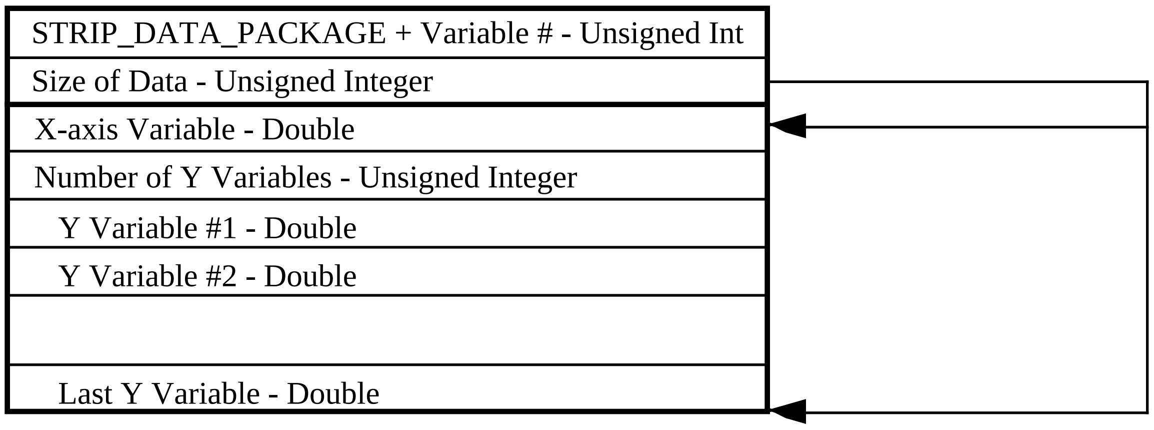

STRIP_DATA_PACKAGE + Variable # - Unsigned Int
Size of Data - Unsigned Integer
X-axis Variable - Double
Number of Y Variables - Unsigned Integer
Y Variable #1 - Double
Y Variable #2 - Double
Last Y Variable - Double

Figure 34 - Format for the Strip Chart Data Package

9.0 - 3D View Data Package

This data package represents two different 3D View graphical displays. This data package is described in Figure 35.

9.1 The TDVSGI is written in GL. The value of TDVSGI is sent for TDV_DATA_PACKAGE.

9.2 The TDVX is written in X. The value of TDVX is sent for TDV_DATA_PACKAGE.

9.3 The number of vehicles is sent, followed by the data for the individual planes.

9.4 The type of the vehicle is sent according to the following table:

Type Vehicle Value	TDVX11	TDVSGI
0	F18	F15
1	MIG23	MIG23
2	AIM9C	MIG23
3	C172	MIG23
4	F16	MIG23
5	F18	MIG23
6	MIG23 w/Modified colors	MIG23
7	KC135	MIG23
8	MIG25	MIG23
9	MIG29	MIG23
10	X29	MIG23

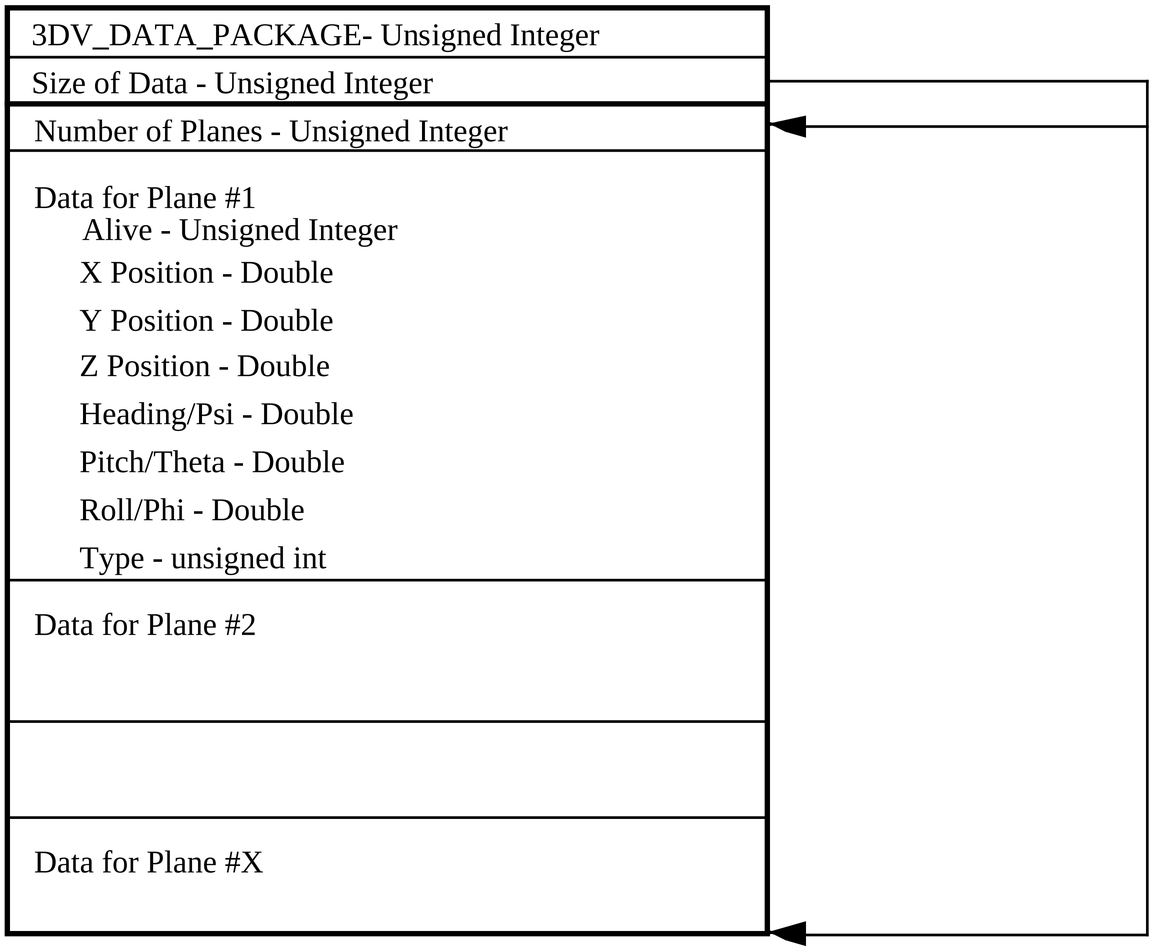


Figure 35 - Format for 3D View Package

10.0 - OCV Data Package

This data package represents two different Out of the Cockpit View (OCV) graphical displays. This data package is described in Figure 36.

10.1 The Data Package ID will vary depending upon which OCV was selected by the user.

10.2 The value of OCV should be sent for OCV_DATA_PACKAGE.

10.3 The OCVHUD is an Out of the Cockpit View with an overlaid Heads-Up Display. The value of OCVHUD should be sent for OCV_DATA_PACKAGE.

10.4 The number of the current vehicle, followed by the vehicle's information, the number of planes, and an array of plane structure is to be transmitted to SimGraph.

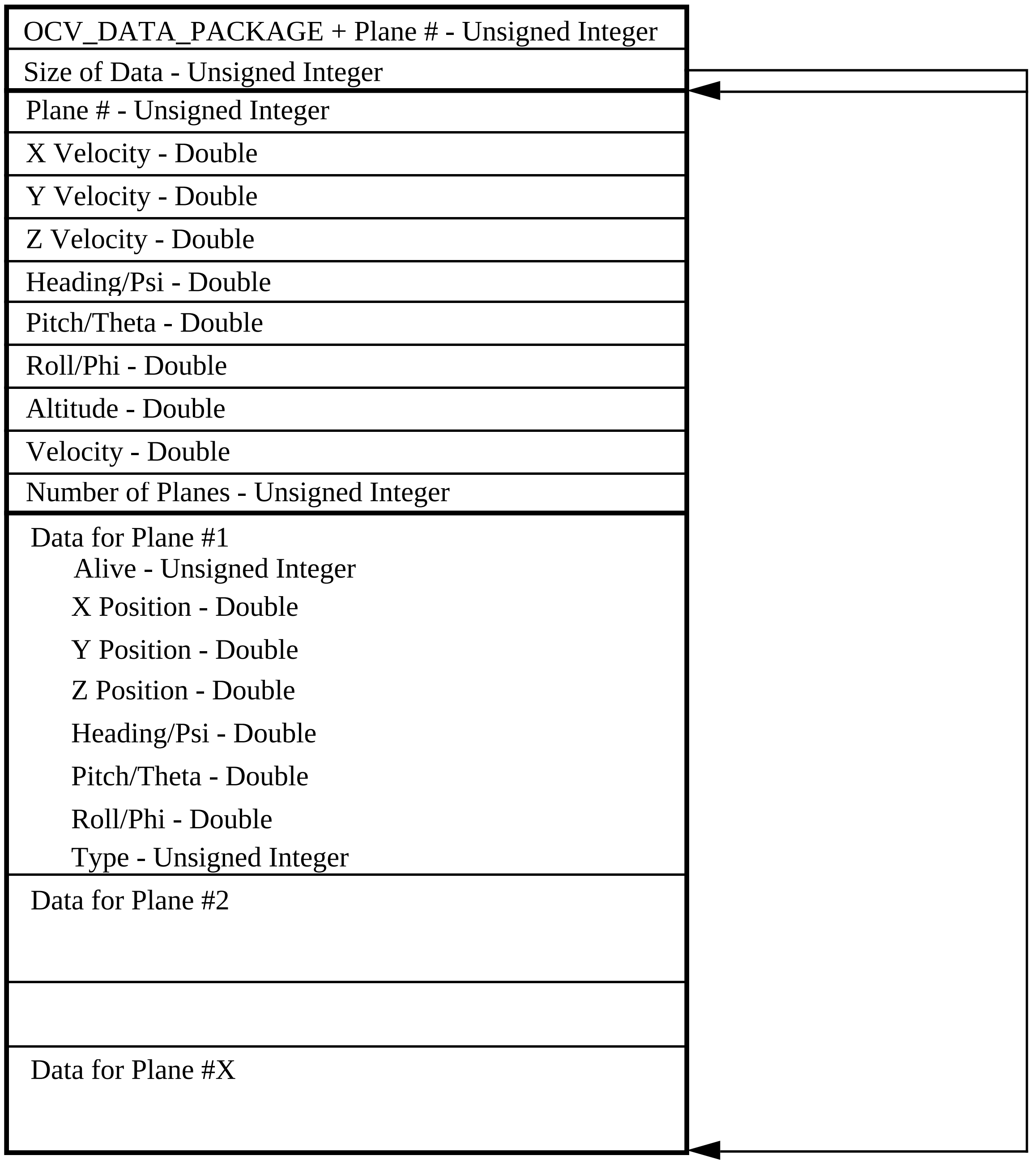


Figure 36 - Format for the OCV Data Package

Appendix C - How to Add a New Graphics

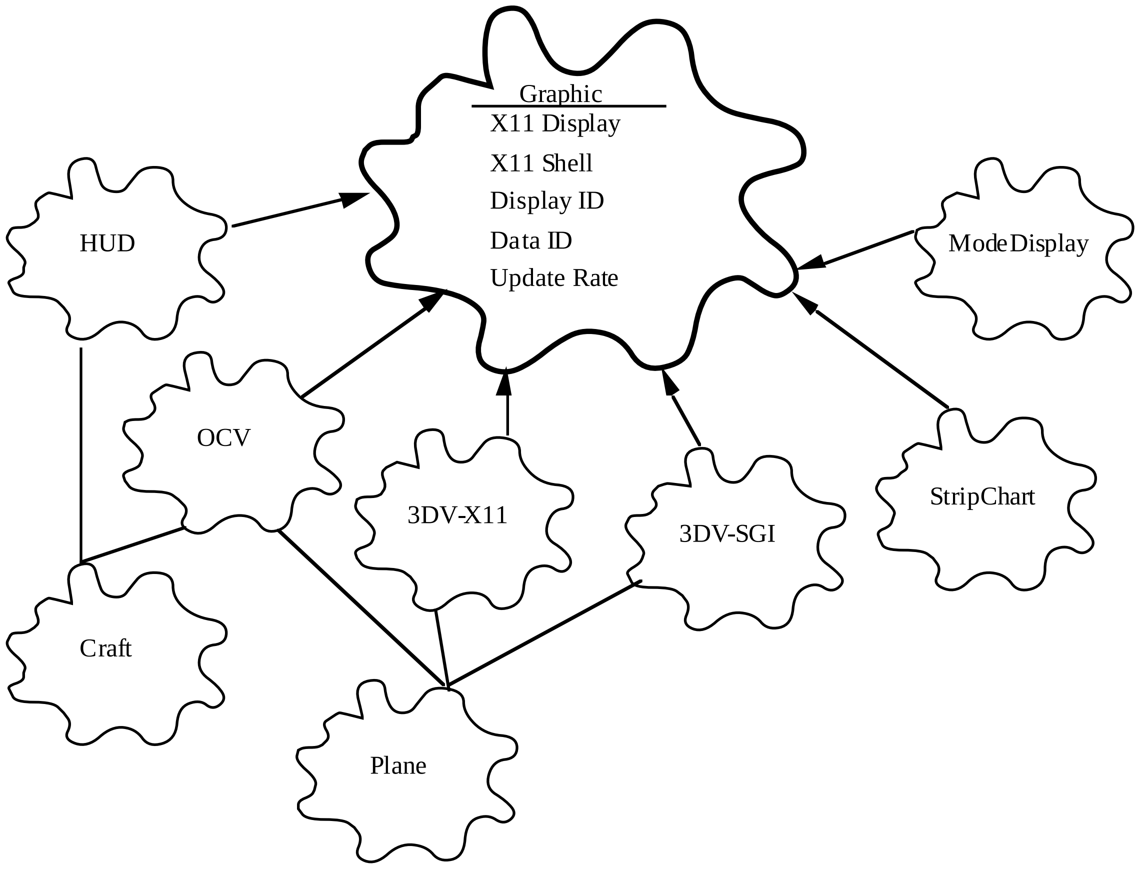


Figure 37 - Overview of Graphic Object

The Graphic Object is one of the cornerstones of SimGraph (Figure 37). 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. These attributes include: an X Window System Display, an X Window System Shell, a unique Display ID, a Data Package ID, and an update rate. The X Window System Display variable is used to indicate which computer the graphic is being drawn on[2]. An X Window System Shell is used as the container for the Graphic Object. All drawing is done inside of this shell, regardless of the graphics language used. It is possible to draw X, GL, and OpenGL code inside of an X Window System Shell[2]. The Display ID is a unique ID number given to each Graphic Object. This is necessary when attempting to delete specific Graphic Objects. The Data Package ID represents the particular Data Package that this Graphic Object needs to update its displays. The Update Rate variable, while unused at the present time, might eventually dictate the rate at which to redraw the graphic.

In order to add a graphic to SimGraph, the following files must be modified as follows:

1. .simgraph_config

First row:
Number of total Graphics

Remaining Rows - One for each graphic type:

Columns	Markings	Description
1-3	Integer	Data ID
4		Multiple Graphic Indicator
	X	Indicates that this graphic will have a separate option for each vehicle in the simulation
	Y	Indicates that this graphic will have a separate option for each Strip Chart Variable
	Space	Indicates that this graphic will only have a single option
5-80	String	Name of Display

Example

```
6
100XHUD
200 TDV for X
300 TDV for SGI
400YStrip Chart
500XOCV
550XOCVHUD
600 Mode Display
```

2. openGraphic.cpp

Edit the routine labeled openGraphic. When the section detailed below is found, the change is easy to make. Simply go to the section of package numbers needed, and insert the necessary code to create a new object of whatever new display and assign that object to new_graphic. It is not necessary to change anything else. For example, if a graphic of type Dial is to be created, the following lines, in bold typeface, would have to be added:

```
if ((data_package_number >= HUD) && (data_package_number <= HUD +
MAX_VEHICLES))
{
    new_graphic = new Hud(display, current->name, data_package_number);
}
else if ((data_package_number>=TDVX) &&
        (data_package_number<=TDVX+MAX_VEHICLES))
{
    new_graphic = new TDVx11(display, current->name, data_package_number);
}
else if ((data_package_number>=TDVSGI) &&
        (data_package_number<=TDVSGI+MAX_VEHICLES))
```

```
{
    new_graphic = new TDVsgi(display, current->name, data_package_number);
}
elseif
((data_package_number>=STRIP)&&(data_package_number<=STRIP+MAX_STRIPS))
{
    new_graphic = new StripChart(display, current->name, data_package_number);
}
elseif
((data_package_number>=OCV)&&(data_package_number<=OCV+MAX_VEHICLES))
{
    new_graphic = new Ocv(display, current->name, data_package_number);
}
elseif ((data_package_number >= OCVHUD) &&
        (data_package_number <= OCVHUD + MAX_VEHICLES))
{
    new_graphic = new OcvHud(display, current->name, data_package_number);
}
elseif (data_package_number == MODE)
{
    new_graphic = new ModeDisplay(display, current->name, data_package_number);
}
elseif  ((data_package_number>=DIAL)  &&
            (data_package_number<=DIAL+ MAX_VEHICLES))
{
    new_graphic  =  new  Dial(display,  current->name,  data_package_number);
}
```

3. The new graphic object must be written. The new graphic object must inherit the Graphic class specified in Graphic.hpp. The five methods described below must be included in the new Graphic Object.

Method	Arguments		Description
Constructor	1	X Window Display	describes the machine and screen this graphic will be displayed on
	2	Character String	a character string that will be put on top of the graphic's window
	3	Data Package ID	the Data Package ID that this Graphic Object will need to request from the Data List in order to update it's graphics
Destructor		NONE	
initialize		NONE	
hold	1	Size of Data	Size of Data in byte of the next parameter
	2	Character Pointer	Pointer to data stored in character array that is needed by Graphic Object to update displays
	3	Direction of Movement	Details the direction of data movement for graphics that store time related information
update	1	Size of Data	Size of Data in byte of the next parameter
	2	Character Pointer	Pointer to data stored in character array that is needed by Graphic Object to update displays
	3	Direction of Movement	Details the direction of data movement for graphics that store time related information

Constructor - The Constructor method must create the Graphic Object and have it displayed on the screen. This Graphic Object must have the graphics drawn inside of an X Window System Shell.

Destructor - The Destructor method will be called when the Graphic Object is destroyed, so it must free all memory that has been allocated from the heap by the Graphic Object.

initialize - This void method is called when the simulation is in RESET. It can also be called from the Constructor Method, but it is not required. This method should reset the Graphic Object to whatever initial conditions are important. hold - This void method is called when the simulation is in HOLD. Some Graphic Objects will continue to update their graphics while others will simply just return without doing anything. Some Graphic Objects, in order to update their displays, will simply pass their arguments on to their update methods.

update - This void method is called when the simulation is in OPERATE. This method should unpack its data from the character pointer, check for its integrity, and then update its displays.

4. The openGraphic.cpp file discussed in #2 above must be edited so that it includes the new header file that describes the new Graphic Object.

5. Edit the SimGraphMessage.hpp file to include the type of message that you will be passing back and forth. Specifically, the unique Data Package ID number for your graphic must be added to the following section:

```
const unsigned int INTRO  =   1;
const unsigned int HUD    = 100;
const unsigned int TDVX11 = 200;
const unsigned int TDVSGI = 300;
const unsigned int STRIP  = 400;
const unsigned int OCV    = 500;
const unsigned int OCVHUD = 550;
const unsigned int MODE   = 600;
const unsigned int DIAL = 700;
```

Also, the specific message must be added.

```
// Dial package
struct DialPackage
{
   float value;
} ;
```

This is the message that is copied out of the data buffer in the hold and update methods of the new Graphic Object.

```
void Dial::update(unsigned int size_of_data, char* data,
                  MoveDir direction)
{

   char color[] = "SlateBlue";
   char temp_string[80];
   struct DialPackage dial_mess;

   bcopy(data, &dial_mess, sizeof(struct DialPackage));
```

The data is copied from the data pointer into the new DialPackage message structure. All access to the data is now conducted through the DialPackage message structure.

Appendix D - Description of the Display Menu Object

The Display Menu Object is used to create the Open Graphic Objects Window (Figure 17) that is shown to the user after the Open Graphic Button on the Main Option Window (Figure 15) has been pressed. It would be time consuming for the programmer to have to edit the file that was responsible for these menus, if the programmer was familiar with the X Window System. By changing a configuration file that is read when SimGraph is initialized, the menus will be configured by the Display Menu Object automatically.

When this object is initially created, a data file is read from the hard disk that details the local configuration of SimGraph. This file is called ".simgraph_config" and it lists the number of Graphic Objects being used, the Data Package ID associated with each Graphic Object, and the names of the Graphic Objects. During creation of the Object, the file is parsed and the data found is stored for later use. The basic building block of the internal data structure for the Display Menu Object is shown in Figure 37. Since the types of graphics available to the user will remain constant throughout the run, the first level of the Display Menu Object's data structure is built. Based upon the sample configuration file shown in Figure 39, the data structure shown in Figure 40 is built.

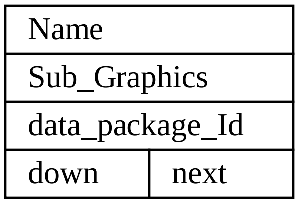


Figure 38 - Menu Structure

```
4
100XHUD
300 TDV for SGI
400YStrip Chart
600 Mode Display
```

Figure 39 - Sample configuration file

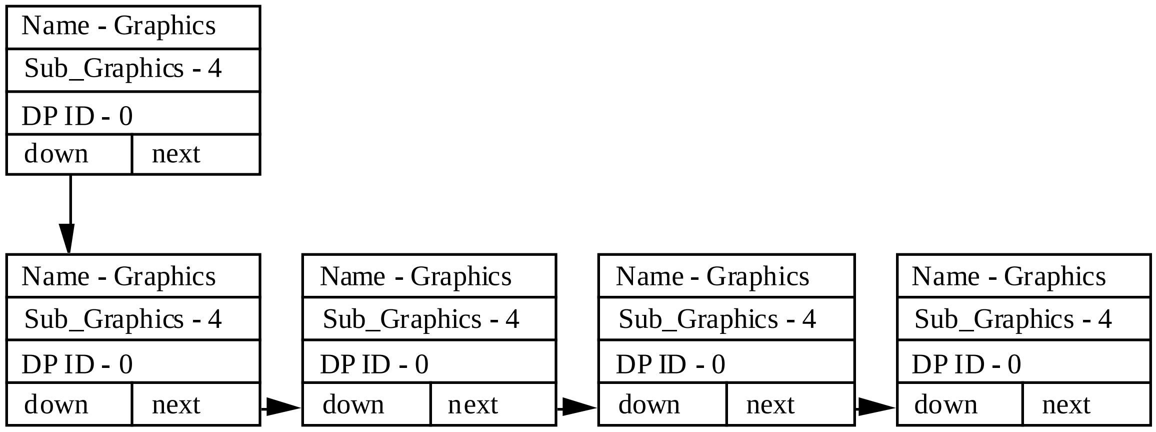


Figure 40 - Initial Data Structure

When SimGraph connects to the simulation program, an INTRO Package is exchanged. This INTRO package contains the number of vehicles in the simulation, the number of Strip Charts, and the names of the Strip Charts. This information is very important when building the Display Menus. When the simulation is in RESET, the number of vehicles is checked to see if a change has occurred. If a change has occurred, then the createMenus method is called with the new information collected from the INTRO Data Package. The createMenus method cleans up any previously existing data structures and proceeds to build a completed data structure based upon the INTRO Data Package. The resulting data structure that is built by createMenus is shown in Figure 41.

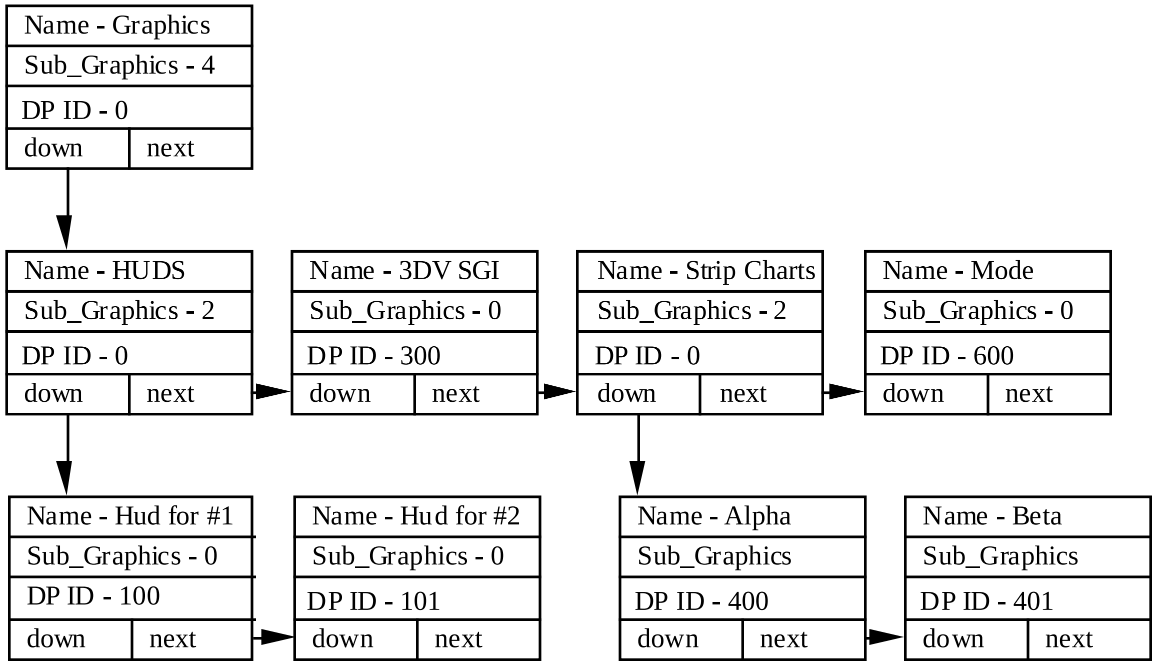


Figure 41 - Completed Data Structure

Once the data structure shown in Figure 41 is built, it must be parsed to build the Open Graphic Objects Window (Figure 17). The openMenus method of the Display Menu Object is setup as an X Window System Callback. The client data that is passed to this callback is a structure called a “MenuPack,” which is shown in Figure 42. The shell variable would point to the shell for the current Open Display Window and allow it to be destroyed. The node variable points to current Menu structure (i.e., one of the nodes in Figure 40). The temp_DisplayMenu variable is a pointer to the current Display Menu Object. The display variable is an X Window System variable.

```
typedef struct
{
  Widget shell;
  Menu* node;
  DisplayMenu*
  temp_DisplayMenu;
  Display* display;
} MenuPack;
```

Figure 42 - MenuPack Structure

When openMenus is first called, the node variable is checked. If the node variable is empty (i.e., set to NULL), then it is assumed that the node currently selected is the top node. If a shell has been passed to openMenus, then that shell is destroyed. Various components of the Open Display Window are built, followed by the number of push buttons corresponding to the number of the node's sub-graphics. Each of the sub-graphics are then searched for their names so the buttons can be labeled correctly. The window is then shown to the user.

The callbacks for the push buttons are determined in one of two ways. The first method of doing the callbacks is implemented if the graphic has sub-graphics under it. If it does, then the callback for that button is a call to the openMenus routine again. The parameters are set up so that the selected graphic will be at the top of the hierarchy, thus showing the user the sub-graphics. An example of this would be if the user selected the HUD graphic. A new menu would appear that showed the user a choice between the HUD for vehicle one or the HUD for vehicle two. The second method of assigning a callback would be done for graphics that do not have sub-graphics. This would be the case for any button that leads directly to a Graphic Object. The callback in this case is to the openGraphic routine, which will actually create the Graphic Object. The only parameter sent to this routine is a "GraphicPack" structure. This structure gives the shell of the Open Displays Window (so it can be destroyed), the Data Package ID number, and the new graphics name so that it can be properly labeled.

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

3. Crawford, Daniel J.; Cleveland II, Jeff I. “The New Langley Research Center Advanced Real-Time Simulation (ARTS) System” AIAA 86-2680, October 1986.

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

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

6. Riley Rainey, ACM Version 4.3 The Aerial Combat Simulation for X11, February, 1994.

7. Mark Edel, Histo-Scope Plotting Widget Set Release 1.0, February, 1995.