Virtual Reality Modelling Language for Visualisation of Flight Simulations

Roger A. Stuckey, PhD
Air Operations Division
Defence Science and Technology Organisation
506 Lorimer St, Fishermans Bend, Melbourne, VIC 3207
Roger.Stuckey@dsto.defence.gov.au

Abstract. The Virtual Reality Modelling Language (VRML) is used to provide a graphical display for real-time simulations of helicopters carrying external loads. Following computation of the helicopter load response to a series of pre-defined control inputs, a script is utilised to generate output in VRML format, suitable for viewing in any one of the browsers available on the Internet. This file, linked with helicopter, load and terrain models then constitutes a recreation of the simulation in a virtual world. The language is compact, comprising only the positional and orientation keys for each body in the scene, as well as the geometric models themselves. Additional detail, such as background grids, can be created if required, while objects such as patches can be added to indicate load-deviation angles. The script was developed using quaternion transformations of the helicopter, load, cables and viewpoints in Cartesian space. Using the External Authoring Interface (EAI), further enhancements to the simulation have also been implemented. These include a timer function, written in Java, allowing synchronisation across multiple frames and consequently the facility to display more than one viewpoint at a time. Another Java routine that provides time-history trace plots along with the simulation has been developed. The tool has proven invaluable in the assessment of helicopter stability when operating with external loads.

1. INTRODUCTION

The Virtual Reality Modelling Language (VRML) [1] has been used as a standard for modelling 3D virtual worlds on the World Wide Web for several years now. Much of the attention has been in the creation of virtual worlds, with the aim of building shared communities on the web. Another area of application has been scientific visualisation in three dimensions. VRML provides an ideal environment in which data can be presented, explored and interacted with beyond many traditional forms of analysis. Under the Chinook Flight Operations Task at DSTO, one of the first requirements was to build a helicopter slung-load dynamic model for analysis of the dynamic behaviour. The simulation model was developed in MATLAB [2]; however, that was found to be inadequate for visualisation of the resulting response. As a consequence, it was decided to utilise the Virtual Reality Modelling Language for graphical presentation of the data.

1.1 Brief History of VRML

Following the First International Conference on the web in 1994, Mark Pesce and Tony Parisi set out to prescribe a standardised language for the description of 3D scenes on the web. The VRML standard was to be open, platform independent, and it was to support 3D multimedia and shared virtual worlds on the Internet. Soon after, members from the VRML mailing list, including Rikk Carey and Gavin Bell, formalised the first draft of the language specification. VRML 1.0 was based on the Open Inventor ASCII file format, developed by Silicon Graphics Inc. (SGI), and supported complete descriptions of scenes with geometry, lighting, materials and a user interface, as well as the necessary networking extensions. In 1995, SGI released WebSpace Navigator, the first VRML browser, and soon after, Intervista released WorldView for Windows. Also in that year, several of the mailing list's leading experts formed a subgroup, named the VRML Architecture Group (VAG), whose purpose was to develop a scalable, fully interactive standard for 3D shared worlds. By mid 1996, the official VRML 2.0 specification was published, and by the end of 1997 it had been standardised by the International Standards Organisation as VRML97 (ISO/IEC 14772-1:1997).

1.2 Issues with Simulation Model Analysis

In the field of simulation, system hardware can be costly, and the visualisation software is often commercial, using proprietary Application Programming Interfaces (APIs) and file formats. These aspects can make it difficult to view simulations on typical desktop computers, let alone share the information across a network on different platforms.

The Virtual Reality Modelling Language is an open, standardised file interchange format that supports scripting in other standard languages such as Java and Javascript. VRML browsers (both plug-ins and standalone versions) are freely available on the Internet for a wide range of platforms, and they typically include many features such as real-time playback, advanced modelling and rendering capabilities including spline and NURBS objects, lighting effects, varying level-of-detail and anti-aliasing. Moreover, they have the advantage of being able to be shared across the Internet easily, opening up possibilities for the dissemination of graphical information. VRML files can be hosted on any web server for transfer and presentation on a client

machine. They are also compact and modular, allowing for rapid, efficient download of the virtual world with only those constituents required at any one time.

2. SIMULATION IN VRML

The facility to generate VRML scenes from simulation data was developed in order to provide a graphical display of the aircraft response. It must be understood, therefore, that the simulation actually takes place prior to display in the VRML browser. For this project, the simulation is run within the MATLAB numerical computing environment, using a package named Helicopter Slung-Load Simulation (HSLSIM) [3]. Figure 2.1 outlines the basic scheme employed. The MATLAB simulation is able to utilise a high-fidelity helicopter model, ROTORGEN, which is written in Fortran. A supplementary script, HSLVRML, is then executed to output the simulation data to a file in VRML format, which can be hosted on the web and viewed with an appropriate browser. It is also possible to link the VRML simulation to Java applets via the External Authoring Interface (EAI) in order to provide additional information and/or functionality. One such applet, VRMLTRACE, generates a set of time-histories for arbitrary variables of the simulation in a neighbouring frame of the web browser. These components will be discussed in further detail below.

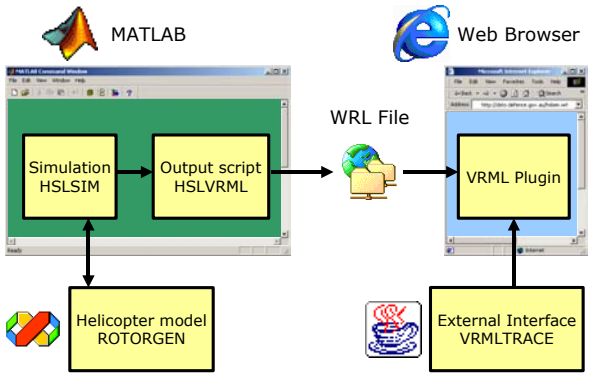

Figure 2.1: VRML generation scheme

2.1 Calculation of Simulation Data

The Helicopter Slung-Load Simulation package HSLSIM was developed at DSTO as a fundamental part of the Chinook Flight Operations Task sponsored by the Australian Army. In the past, the operations of helicopters carrying externally slung loads has often been limited and, in some cases, seriously hindered by stability and control problems. A program was consequently initiated within the Defence Science and Technology Organisation (DSTO) to use computer modelling and simulation to assist in defining the operational limits of the Australian Army Chinook CH-47D helicopter when carrying slung loads. The first phase in this program entailed the development of a helicopter slung-load model for simulation and analysis in order to provide a better understanding of the system dynamics and various effects involved.

HSLSIM was written in MATLAB, a high-performance numerical computing environment which is built around matrix mathematics, and therefore amenable to dynamic modelling and simulation work. MATLAB also provides an ideal workplace in which various analyses can be conducted following the simulation itself. The simulation code is a complex multi-body solution based on the formulation developed by Cicolani *et al.* [4]. In this formulation, the general system equations of motion are obtained from the Newton-Euler equations in terms of generalised coordinates and velocities. Following the explicit constraint method, which utilises d'Alembert's principle, the system is partitioned into coordinates such that the motion due to cable stretching is separated from that due to rigid-body, coupled dynamics. As a consequence, the constraint forces appear explicitly and a solution to the resultant generalised accelerations is determined by assuming a simple spring model for each cable.

A higher fidelity model than the one used initially was also integrated in the simulation via a MATLAB External (MEX) interface. ROTORGEN was developed by Heffley [5] for the US Army Aeroflightdynamics Directorate under NASA contract to Hoh Aeronautics Inc. It is described as a "minimal-complexity generic rotorcraft model" intended for manned simulation of large military helicopters and, in particular, the CH-47D Chinook tandem rotor helicopter.

The simulation model is first initialised with a set of flight conditions and given a pre-determined sequence of control inputs to be used in the computation of a response. The model is then trimmed according to the specified flight conditions until an equilibrium state is reached, and then integrated over the time range to produce a set of simulation data. The data comprises the helicopter and load positions, orientations, the control inputs and geometric information regarding the helicopter load configuration. As for most aeronautical systems [6], the positions are specified in right-handed Cartesian axes (with positive-z downward) and the orientations in Euler angles.

Once the data have been collected, it is then passed to the output script HSLVRML for processing and generation of the VRML file. There are a number of transformations necessary to convert the position and orientation data for helicopter and load into a form suitable for inclusion in the VMRL model. The primary transformation is that from Euler orientation to quaternion orientation, the latter being represented by a unit vector axis $x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$, plus a rotation angle α about that axis. To convert Euler angles to quaternions, the following expressions [7] are used:

$$q_1 = \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\varphi}{2}) + \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\varphi}{2})$$

$$q_2 = -\cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\varphi}{2}) + \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\varphi}{2})$$

$$q_3 = \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\varphi}{2}) + \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\varphi}{2})$$

$$q_4 = \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\varphi}{2}) - \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\varphi}{2})$$

Then, to convert from quaternions to a rotation axis and angle, the following are used:

$$\alpha = 2\cos^{-1}(q_1)$$

$$x = q_2 / \sin(\frac{\alpha}{2})$$

$$y = q_3 / \sin(\frac{\alpha}{2})$$

$$z = q_4 / \sin(\frac{\alpha}{2})$$

Similar transformations are also needed for the rotors and the sling cables, both of which are transformed about the helicopter body-axes before transformation to inertial axes using the equations above. For example, the rotors are rotated about their own axes according to the current time-step, then translated to their positions up on the forward and aft helicopter hubs, and then they are rotated and translated along with the helicopter. Calculation of the position and orientation for each of the sling cables is somewhat more involved, however. It requires that the positions of the attachment points first be determined from the helicopter and load, and then that the equivalent quaternion rotations be calculated for cylinder nodes starting from their default orientations. Another set of coordinates that are required for the VRML model are the vertices of the load-displacement patch, a semi-transparent arc illustrating the deviation of the load from its trim location relative to the helicopter. As with all of the above data, these transformations must be performed for each time step, since they all depend on the position and orientation of the helicopter and load, which changes throughout the simulation.

In addition to the transformations of the geometric nodes, several additional sets of data are required for the moving viewpoints in the VRML world. The viewpoints include typical orthographic perspectives of User defined, Side, Front, Top and Behind, as well as Fixed-Inertial, a Zoom/Pan view, Fixed-Aircraft (user defined) and Fixed-Aircraft (downward looking). The orthographic viewpoints remain static in orientation and positionally fixed relative to the helicopter throughout the simulation. Orientation of the dynamic viewpoints, such as the Fixed-Inertial view, must be recalculated for each time step, as with the helicopter and load. Both the Fixed-Inertial and Zoom/Pan views are aligned along the vector drawn from a fixed position specified in inertial axes to the helicopter's origin. While the Fixed-Inertial view remains fixed at the location specified, the Zoom/Pan view maintains a constant absolute distance, typically much closer, from the helicopter. The Fixed-Aircraft views undergo the same transformations as the helicopter, with the addition of a rotation and translation applied to each beforehand.

The last set of nodes that are created are the gridlines and ground surface, the dimensions of which are calculated to suit each simulation. The grid is created purely for reference information and covers each wall of a rectangular box shape, where the size of the box is equal to the domain of the helicopter and load movement, plus some arbitrary buffer.

After all of these nodes have been output to file, HSLVRML adds the background information, some lights, a set of default navigation information and the routes necessary for animation. As a final step, the file can be compressed into a compact binary format using GZIP [8], which is still able to be read by the VRML browser, but small enough for fast transmission over the Internet.

2.2 Structure of VRML Model

As stated in the Virtual Reality Modelling Language Functional Specification [9], "A VRML file consists of the following major functional components: the header, the scene graph, the prototypes, and event routing. The contents of this file are processed for presentation and interaction by a program known as a browser."

The file is typically formatted in UTF-8 clear text encoding, also known as Unicode, which is defined in the header. The scene graph contains nodes used in the model. Nodes are abstractions of various real-world objects and concepts, such as boxes, interpolators, lights and material descriptions, organised into a transformation hierarchy. Prototypes define new node types in terms of already defined (built-in or prototyped) node types. They can be included in the file in which they are used or defined externally. The scene graph also includes routes, which define how and when events are propagated through the model to affect other nodes. When generated, these events are sent to their routed destinations in time order and processed. It is the execution engine that processes events, reads and edits the route graph and makes changes to the transform hierarchy.

Figure 2.2 illustrates the conceptual execution model utilised in the simulations.

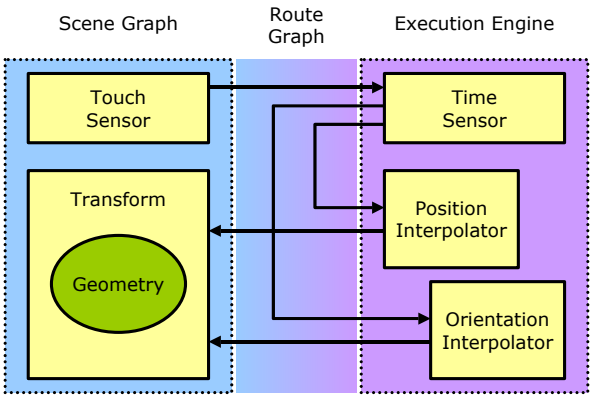

Figure 2.2: The conceptual VRML execution model

The general order in which execution takes place is as follows:

1. User clicks on a Touch Sensor, sending a *touchTime* event to the Time Sensor *startTime*.
2. The Time Sensor becomes active and starts generating continuous events, including *fraction_changed* - a variable in the closed interval [0,1] indicating the completed fraction of the current cycle - which is sent to the *set_fraction* events of various Position and Orientation Interpolators.
3. The *value_changed* events of those Position and Orientation Interpolators are sent to the *translation* and *rotation* fields, respectively, of the corresponding Transforms.
4. Any geometries within the Transforms undergo translation and rotation according to the values set.
5. Execution continues with steps 2 through 4, repeated until the Time Sensor time reaches its *cycleInterval*, upon which it becomes inactive and the event clock is stopped.

The browser takes care of the interpretation, execution and presentation of the VRML file, and facilitates navigation through the virtual world as well as interaction from the user. Most importantly, perhaps, the browser controls the passage of time in a world through Time Sensors and typically the times generated will approximate "real" time. This necessitates the use of Interpolators to calculate the state of the Transform nodes at any one time, since no assumptions can be made about how often a Time Sensor will generate events. Interpolators are designed for linear key-framed animation, which means that, given a scalar position within the list of keys, they will interpolate within the list of *keyValues* to return a corresponding value. For the helicopter slung load simulation, these include various Position and Orientation Interpolators for all moving objects in the scene.

The physical format of the VRML simulation model essentially comprises two parts: the main file that details the virtual world, and a group of external prototype files containing the geometric models. These prototypes include the helicopter and loads, which are chosen from a set of models depending on the configuration and do not change during the simulation. Both helicopter and load models are comprised mainly of Transforms, simple shape nodes, such as Boxes, and more complex ones, such as IndexedFaceSets. An ImageTexture node within the Shape node of the Helicopter and Load may also contain a URL link to an image that is mapped on to the parent Shape. If this is omitted, the Appearance of the Shape is set by the Material node instead. Contained within the main VRML file are references to those external nodes, the Time and Touch Sensors, Interpolator nodes, additional geometric model nodes, Viewpoints, the simulation grid, background, lighting and navigation information, as well as the route graph. Figure 2.3 outlines the basic structure of the VRML model.

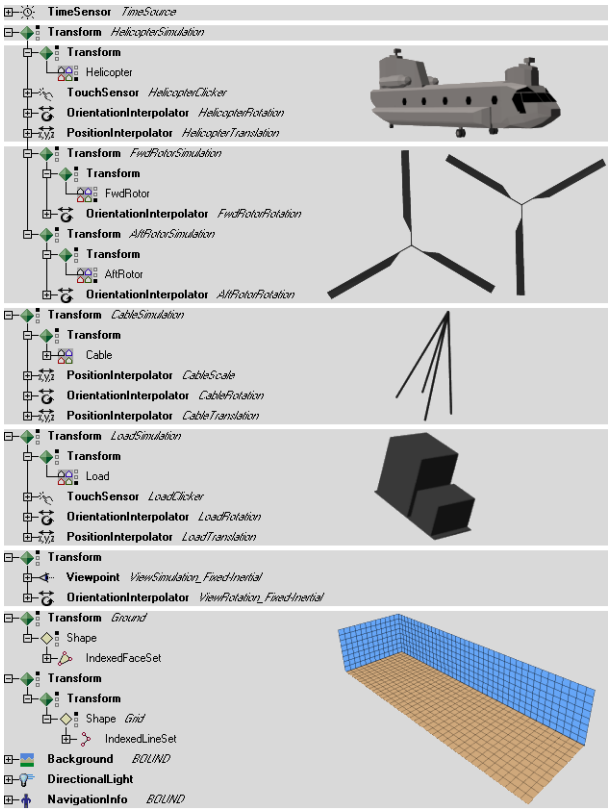

Figure 2.3: Transform hierarchy of VRML model

The TimeSensor is listed first, and is instantiated with a *cycleInterval* equal to the total simulation time in seconds. The remaining parameters, such as the *startTime* and loop, are left at their default values.

Next are the Transform nodes that parent the helicopter, rotors, cables and loads, as well as their respective Translation and Orientation Interpolators. Each Transform node has a set of children, as well as a *rotation*, *scale* and *orientation* defined by vectors. The HelicopterSimulation Transform contains several children, including the helicopter and rotor transformation hierarchy. For the helicopter, a node is created by reference to an external prototype, declared at the start of the file in an EXTERNPROTO field. A

TouchSensor node renders all geometric nodes within the same Transform sensitive to activation from user input, such as a mouse button. When the mouse button is pressed and released over either the helicopter or rotors, the TouchSensor generates two *isActive* events and a *touchTime* event. In addition to the geometric helicopter node, an OrientationInterpolator node and PositionInterpolator node are grouped within the Transform. It is not necessary that they be grouped in this way, since the Route Graph independently defines all interaction in the virtual world, but is done so for clarity. Both Interpolators define an array of scalar keys and an array of vector *keyValues* - the *orientation* and *position* at each key.

The FwdRotorSimulation and AftRotorSimulation constitute two more Transforms within the HelicopterSimulation Transform, and each holds a further Transform of its own, as well as an OrientationInterpolator. There are two successive Transforms in this case, so as to effect a constant rotation rate about one axis - the rotor hub - and then another *rotation* and *translation* to move the rotor to its position on the helicopter. Within the second Transform lie the geometric rotor nodes, comprised, once again, of simple shape objects such as Cylinders and IndexedFaceSets.

The CableSimulation and LoadSimulation nodes are defined outside the main HelicopterSimulation Transform, but could just as easily have been defined within. They are separate here because their *positions* and *orientations* are calculated in inertial axes, as opposed to helicopter axes, and so their motion is absolute rather than relative. There are actually several Cable nodes in the model shown, although only one has been included in the hierarchy, since their structure is identical. Much like the helicopter, they contain further Transform nodes, Interpolator nodes and geometric nodes. The CableSimulation includes the addition of a PositionInterpolator for the changing scale, or stretching of the cable. The LoadSimulation also has a TouchSensor node, the same as that in the HelicopterSimulation Transform. Both TouchSensors are routed to the TimeSensor, so that either can start the simulation.

A series of Transforms containing the various Viewpoints follow, though only one is listed in the figure. These consist of a Viewpoint node, with *fieldOfView*, *orientation*, *position* and *description* fields, and an OrientationInterpolator. Some viewpoints, such as the Fixed-Aircraft, also include a PositionInterpolator.

Two transforms for the Ground and Grid nodes are next. Only one of the six similar Grid nodes is listed. The Ground is constructed from an IndexedFaceSet, while each of the Grid faces is built from an IndexedLineSet.

Last in the list are the Background, DirectionalLight and NavigationInfo nodes for the virtual world. The Background node has a number of fields including the *groundAngle*, *groundColor*, *skyAngle* and *skyColor* values. The DirectionalLight node has an *ambientIntensity*, *color*, *direction* and *intensity* among others. The NavigationInfo node includes a *headlight*, *speed* and *type* field. Each of the different Viewpoints is listed in the type field for easy navigation.

Apart from the external prototype declarations at the start of the file, the only other set of semantics not included in the above diagram are the routes. Routes are not nodes, but are construct statements establishing event paths between nodes. They may be established only from *eventOut* fields to *eventIn* fields, and the types of the *eventIn* and the *eventOut* must match exactly. As an example, the route path for the helicopter's rotation and translation is constructed with the following code:

```
ROUTE HClick.touchTime        TO TimeS.startTime
ROUTE TimeS.fraction_changed  TO HRot.set_fraction
ROUTE TimeS.fraction_changed  TO HTrans.set_fraction
ROUTE HRot.value_changed      TO HSim.rotation
ROUTE HTrans.value_changed    TO HSim.translation
```

Route statements can either appear at the top hierarchical level of the VRML file, in a prototype definition, or inside a node wherever fields may appear.

3. EXTENDING THE SIMULATION

Some time after the VRML script was written, a few enhancements were desired to improve the functionality and provide more information during the simulation replay. Primarily, these included the ability to view several frames at once, and the display of other system variables. The first enhancement would allow for either different simulations to be run concurrently, or different views of the same simulation to be displayed at the same time. The second would provide indication of other important variables, not shown in the virtual world, such as the control inputs driving the helicopter response.

Both of these enhancements were made possible through using the External Authoring Interface (EAI) and the Java Virtual Machine [10]. The EAI is a proposed Informative Annex to the VRML97 specification and currently the most effective way to incorporate additional media components into the display.

3.1 Synchronising across Frames

The problem of synchronising multiple frames - each containing VRML modes - within the web browser was reasonably straight forward, although it did require the creation of a supplementary VRML file containing another TimeSensor node. This timer is placed in a frame of zero size, since there's nothing to display, and it runs continuously. The Java applet accesses the TimeSensor nodes from the timer, and the VRML

simulations, as well as references to their *eventOut* and *eventIn* fields, respectively. It essentially routes them together, and sets the *startTime* of both dormant TimeSensor nodes in the VRML simulations to the *time* of the already running TimeSensor node in the timer, upon activation of a button widget. This starts the simulations. However, since they are still both internally routed to their own TimeSensors, they will only remain synchronised if the browser keeps both streams running at the same speed. For most applications, particularly when the same or similar models are being run, this has shown to be sufficient.

Figure 3.1 shows a screen-captured image of the web browser with two simulations running synchronously. The simulations demonstrated the effect of sling configuration on the helicopter slung-load stability [11].

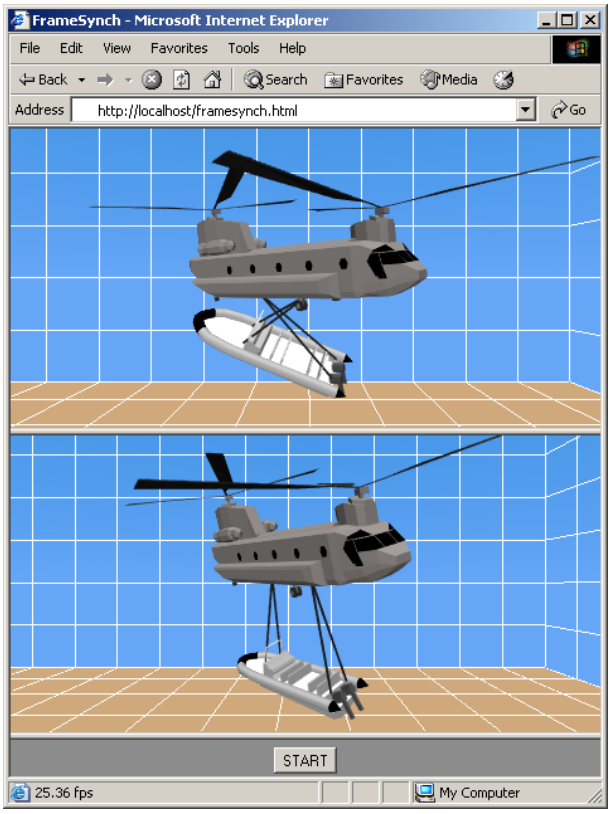

Figure 3.1: Synchronised simulation models

3.2 Displaying Additional Variables

The Java implementation of an additional display was somewhat more detailed, but relied on a similar methodology to that applied in the frame synchronisation case. Two frames are employed in this enhancement, with the VRML simulation in one and a set of moving trace plots in the other. Once again, a separate, individual TimeSensor node is utilised to keep time in both frames. However, the Java applet also accesses those nodes whose values are to be displayed on each of the trace plots. For variables not used in the simulation display, such as the control positions, this necessitates the inclusion of additional *dummy* nodes in the model. In this case, four ScalarInterpolator nodes have been added - one for each of the four helicopter controls. Also, four routes from the local TimeSensor to each of the ScalarInterpolators are necessary in order for them to keep time with the rest of the VRML model. Now the local TimeSensor is activated via the independent TimeSensor upon button press, and the Java applet receives *eventOut* values from the ScalarInterpolator nodes as the time proceeds. In a similar manner to the case above, each trace and the VRML simulation are not truly synchronised, since they all run on different threads. This can be noticeable for high frame rates or when there is a large amount of data to plot, but is generally not a problem.

Figure 3.2 illustrates the web browser with a simulation and Java display running.

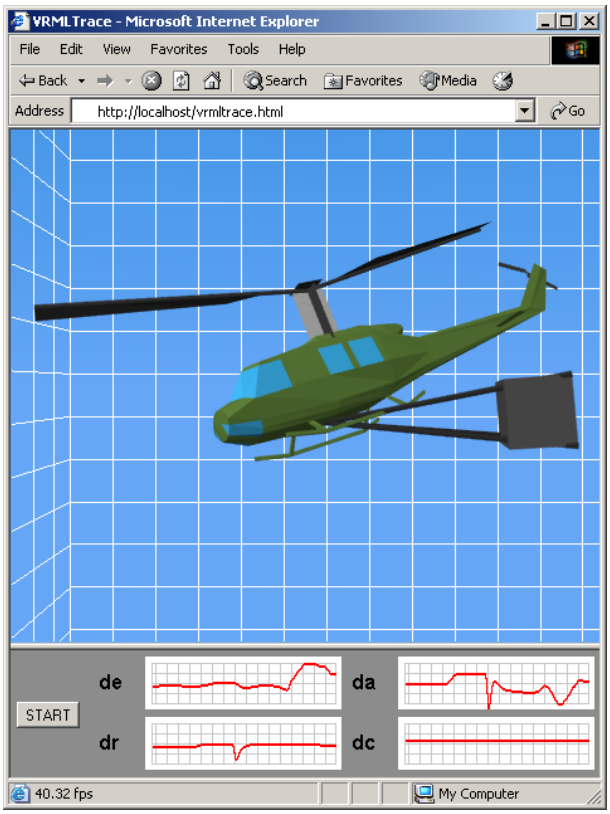

Figure 3.2: Simulation and variable trace plots

4. SCOPE FOR FURTHER DEVELOPMENT

There is considerable scope for further development in the visualisation of simulations using VRML. One idea is to make the output script software independent, since there is no real need to have it run from within MATLAB. This requires rewriting the code in an open standard language, for example Java or Perl/Tk, that can be run on a wide range of platforms. Furthermore, if the format of the input data was standardised, or even generalised, then output from other simulation programs could be processed to return a VRML file.

The thought of running the actual simulation inside a VRML model using scripting is feasible but not practicable in a real-world sense. Simulation models are typically very complex and incorporate a significant proportion of legacy code written in multiple languages, and the VRML paradigm is too restrictive to manage

such programs. VRML as a standard is also quite immature, and for all of its strengths, it still lacks in functionality and extensibility. Fortunately a new standard currently in development, named X3D [12], is addressing those very issues, although such a solution would never be able to compete with dedicated simulation programs that use low-level APIs for display [13]. If there is a future in using VRML or X3D in simulation, it will be as a lightweight visualisation tool, with scope for interaction and collaboration across networks.

5. CONCLUSION

The Virtual Reality Modelling Language has proven to be invaluable in the visualisation and analysis of helicopter slung-load simulations. Under the associated task at DSTO, a methodology has been developed and the software written to generate VRML models from flight data, allowing the simulation to be viewed on the Internet by anyone with a freely available browser installed.

The VRML model has also been extended to display simultaneous simulations or additional data sets during replay. Even with the added capability, the requirements are minimal for viewing the VRML simulation, and can be done using many of the computer platforms in common use today.

REFERENCES

1. Carey, R. & Bell, G (1997) "The Annotated VRML 2.0 Reference Manual", Addison-Wesley, Reading Massachusetts. http://www.web3d.org/resources/vrml_ref_manual/Book.html
2. The MathWorks, Inc. (2000) "MATLAB The Language of Technical Computing. Using MATLAB Version 6", The Mathworks, Natick, MA. http://www.mathworks.com/
3. Stuckey, R.A. (2001) "Mathematical Modelling of Helicopter Slung-Load Systems", *DSTO Technical Report*, TR-1257, Dec. http://www.dsto.defence.gov.au/corporate/reports/DSTO-TR-1257.pdf
4. Cicolani, L. and Kanning, G. (1992) "Equations of Motion of Slung-load Systems, Including Multilift Systems", *NASA Technical Paper* TP-3280, Ames Research Center, Nov.
5. Heffley, R. (1997) "ROTORGEN Minimal-Complexity Simulator Math Model with Application to Cargo Helicopters", *NASA Contractor Report* CR-196705 Hoh Aeronautics, Inc., Lomita, CA., Mar.
6. ANSI/AIAA (1992) "Recommended Practice for Atmospheric and Space Flight Vehicle Coordinate Systems", ANSI/AIAA R-004-1992, American Institute of Aeronautics and Astronautics; American National Standards Institute. http://www.aiaa.org/
7. Shoemake, K. (1985) "Animating Rotation with Quaternion Curves" *Association for Computing Machinery (ACM) SIGGRAPH Computer Graphics, Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, Vol. 19 Issue 3, Jul., pp. 245-254. http://www.acm.org/
8. Deutsch, P. (1996) "DEFLATE Compressed Data Format Specification Version 1.3" and "GZIP File Format Specification Version 4.3", Internet Corporation for Assigned Names and Numbers (*ICANN) InterNIC Request For Comments* RFC-1951 & 1952. http://www.gzip.org/
9. ISO/IEC (1996) "The Virtual Reality Modeling Language (VRML) - Parts 1-2: Functional Specification and UTF-8 Encoding and; External Authoring Interface. International Standard", ISO/IEC 14772-1:1997. http://www.web3d.org/technicalinfo/specifications/vrml97
10. Sun Microsystems, Inc. "Java™ 2 Platform, Standard Edition, V1.4.0 API Specification" Sun Microsystems, Inc., Palo Alto, CA. http://java.sun.com/j2se/1.4/docs/api/index.html
11. Stuckey, R.A. (2000) "Dynamic Simulation of the CH-47D Helicopter and Externally Slung Boat" *American Helicopter Society (AHS), Proceedings of the 3rd Australia Pacific Vertiflite Conference on Helicopter Technology*, Jul.
12. Web3D X3D Task Group (2002) "Extensible 3D (X3D) Parts 1-3: Architecture and Base Components; Application Programmer Interfaces and; Data Encodings. International Standard" *Draft Specification*, Web3D Consortium, Feb. http://www.web3d.org/TaskGroups/x3d/specification/
13. Kelty, L., Beckett, P. & Zalcman, L. (1999) "Desktop Simulation" *SimTecT99 Conference Proceedings*, Mar. http://www.cse.rmit.edu.au/simtect/1999/papers/073.doc