Visualization Tools for SpecTRM

by

Orton T. Huang

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

August 8, 2003

Author_____

Department of Electrical Engineering and Computer Science
August 8, 2003

Certified by_____

Nancy Leveson
Thesis Supervisor

Accepted by_____

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Visualization Tools for SpecTRM

By
Orton T. Huang

Submitted to the
Department of Electrical Engineering and Computer Science

August 8, 2003

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

SpecTRM is a CAD system for digital automation. The intent of SpecTRM is to assist engineers in managing the requirements, design and evolution process aspects of developing software. The purpose of this thesis project is to extend SpecTRM even further through the development of visualization tools. This will consist of building tools in three general areas to increase the functionality of SpecTRM. These areas include post-simulation visualization tools, interactive visualizations and 3D visualizations. Post-simulation visualization involves taking the data produced by a SpecTRM model's simulation of a system and constructing tools that allow the user to see the data in a more meaningful way. Run-time interactive tools involve providing tools that help the user interact with and control the system during simulator operation. 3D visualizations provide a means for the user to see the output of the simulation transform 3D models. The goal of this thesis is to develop meaningful ways to understand and manage a complex software system in addition to all the tools that SpecTRM provides.

Thesis Supervisor: Nancy Leveson
Title: Professor, Department of Aeronautics and Astronautics
Software Engineering Research Lab

Acknowledgements

First of all, I'd like to thank those at MIT who made this thesis possible. Many thanks to my advisor, Nancy Leveson, for all her input, advice and encouragement. For her support for my thesis and for providing such a great environment to work in.

I also want to thank Jeffrey Howard who was always available and ready to answer questions over email and in person and Patrick Anderson, who provided lots of help with the Interactive Visualization Tool. To Jaime Lien, who contributed much of the Matlab stuff and Charita Williams, who helped work on the 3D Visualization Tool. Many thanks to all the SERL students, for their advice and insights and for making the lab an awesome place to work in.

I want to thank my family, for their love and support these past years. I want to thank my mother especially, I wouldn't be here without her hard work and love.

I am forever grateful to the Berkland family. To Pastor Paul and Becky JDSN, to whom I am forever indebted to their ministry and love. For starting Berkland Baptist Church through which I was saved. Thank you for giving me the privilege of living at Concord House and of being discipled there. To my shepherds, Pastor Chris and Sally SMN. For loving me as your own son, for taking care of my spiritual and physical needs for so many years, for your Bible studies, sacrifice, and the legacy you left with us. To Dave JDSN and Angela SMN, for your wisdom, love and encouragement. For guiding me this past year after I came back to MIT and for teaching me to stay the course and mature in my Christian walk. To James hyung, for all the love you have shown me the past 6 years and shepherding me ever since coming to MIT. To the MIT ABSK cell-staff, Hyun Soo noona, Susie noona, Donald hyung, Jung Yoon noona, Teresa noona, Julie noona and Soo for all your prayers, support and fellowship that sustained me the past year. To my class brothers, Austin and Eugene, for all your encouragement, for helping me keep things in perspective and for putting up with me. To the MIT ABSK undergraduate brothers and sisters, for all your prayers and encouragement. To the Concord Brothers, Danny hyung, Matthew hyung, Hero hyung, Jinu hyung, Eddie hyung, Walter, Jason, and Patrick (and Austin again) for your encouragement, love and

fellowship. And to all the other Berkland brothers and sisters not mentioned here, I am indebted to all your prayers and love.

Finally, and most importantly, to God who has provided me grace and mercy through His Son, Jesus Christ. Through Your salvation, You have revealed to me a greater purpose for my life and academics. May the rest of my days be pleasing to You.

Contents

Chapter 1

Introduction

SpecTRM is the Specification Toolkit and Requirements Methodology. It assists users in the development and maintenance of safety critical systems. As most of the decisions that affect the safety of a system are made early in a system's product life cycle, SpecTRM focuses on the development of the system requirements and specifications to produce better and higher quality software. This thesis work is about the development and implementation of various visualization tools for the SpecTRM toolkit.

The next couple of sections will give a brief overview of SpecTRM and its individual parts.

1.1 Brief Background

SpecTRM was developed by MIT Aeronautics and Astronautics Professor Nancy Leveson with the Software Engineering Research Lab (SERL) at MIT and the Safeware Enginering Corporation [14]. It is a CAD system created to assist engineers in managing the software requirements, design and evolution process. SpecTRM helps users understand the completeness and correctness of software through inspections, formal validation tools and simulations.

1.2 SpecTRM Components

SpecTRM consists of several components. The primary components are the Intent Specifications, SpecTRM-RL, the SpecTRM software and the SpecTRM Simulator [15].

1.2.1 Intent Specifications

Intent specifications were developed to help engineers through the requirements, design and evolution process of software development. Intent specification imposes a more structured form for engineers to follow to make the information relevant to the system more easily accessible and understandable to support the product life cycle.

Intent specifications help engineers in the following areas:

- Discover errors early in the development cycle. This allows errors to be fixed at the earliest possible stage with the least cost. The earlier errors are found, the less impact they incur on the overall system design.
- Trace requirements and design rationale in the system design.
- Build required system properties into the design from the very beginning.

An intent specification is a prescribed structure for keeping a written record of analysis, design, implementation, and operational history. It also integrates both formal and informal aspects of software development.

1.2.2 SpecTRM-RL

SpecTRM-RL is SpecTRM's formal modeling language designed to help assist users in ensuring that:

- Requirement system functionality has been incorporated
- The system design and architecture exhibits desirable properties, such as safety and fault tolerance
- Human-machine interactions have been appropriately designed
- Trade-offs between conflicting goals are resolved adequately

SpecTRM-RL uses a state machine as its underlying formal model. In building large complex systems, there are many interactions between components in the systems that cannot be thoroughly planned, understood, anticipated or guarded against. SpecTRM-RL has been designed to help manage the difficulty in designing and building complex systems. Through blackbox modeling, SpecTRM-RL more closely models an engineer's mental modes in evaluating a system's architecture. SpecTRM-RL is structured such that modelers are encouraged to write what a system does rather then how it does it. A specification consists of four parts:

- A specification of the supervisory modes of the controller being modeled
- A specification of the system's control modes
- A model of the controlled process that includes the inferred system state
- A specification of the inputs and outputs to the controller

1.2.3 SpecTRM Software

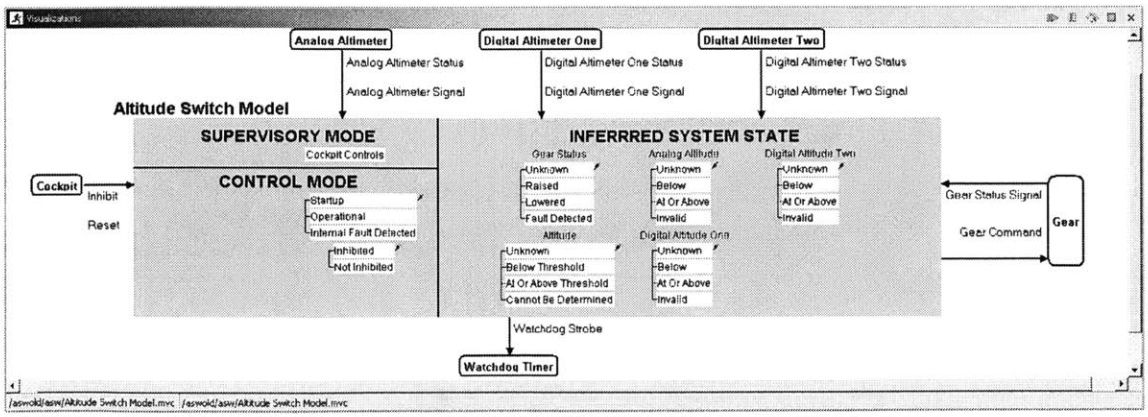
SpecTRM is a CAD system for digital automation. The intent of SpecTRM is to assist engineers in managing the requirements, design and evolution process aspects of developing software. Developing software for complex systems invariably includes

interaction between multiple disciplines. Heterogeneous systems require a great deal of knowledge across engineering disciplines. SpecTRM attempts to bridge the gap between the diverse groups of system designers and builders. It provides the following features to aid in designing and maintaining complex, heterogeneous systems:

- Support for Intent Specifications and SpecTRM-RL models
- Document Generation
- Platform Independence
- Project Browser
- Integrated Editor
- Data Dictionary
- Simulator
- Consistency and Completeness Analysis

1.2.4 SpecTRM Simulator

In a large complex system, each intent specification models the behavior of the system. SpecTRM uses SpecTRM-RL to describe the behavior of the system and SpecTRM-RL is based on automata theory which allows it to be formally analyzed. SpecTRM uses state machines to model the system behavior which allows it to execute and simulate the specifications. The simulator is an execution engine for these models. For the simulator to operate, the model must be well-formed and validated.



SpecTRM Simulator Visualization

This research work attempts to provide meaningful visualizations that can help the user understand how the SpecTRM model is operating. Presently, the inputs for the Simulator are described in a text file and outputs from the Simulator are displayed (see above image) or printed out to a file. The Simulation Visualizations and 3D Visualizations will attempt to provide another dimension in understanding the SpecTRM Simulator results.

1.3 Research Motivation

Many different types of modelers and simulators have been developed for all kinds of purposes. This section will look at some of the software that has been developed to aid users in modeling and simulating systems and describe some of the other research and work that has gone into this area.

1.3.1 Config

Config [10] is a hybrid discrete/continuous modeling and simulation tool developed at NASA Johnson. The purpose of Config is to design a system to evaluate the operability of life support systems. To do this, it provides an object oriented and graphical environment for building models and managing simulation tests. Config simulates operation scenarios in which flows and pressures change continuously while system reconfigurations occur as discrete events. One of the goals for the designers of Config was the early dynamic analysis of operational problems. In a system of flows and pressures, system accidents commonly result from a series of events that lead to a hazardous configuration. Through Config, the designers wanted to increase the range of system-level hazards that could be predicted. Config features object-oriented model types for devices and activity models that support the modeling of controllers, human operator procedures, actions and schedules. Config provides some very insightful ideas to improving design evaluation. Its strength is in simulating cascading effects on operations and failures and the interaction between the software system and other input sources, including human procedures and user-interfaces.

1.3.2 Modelica

Built up from several engineering domains, Modelica [4, 7, 8] was designed to analyze and synthesize complex systems through linking previously unrelated domains and the tighter coupling of subsystems. Modelica also uses an object-oriented modeling methodology coupled with equation-based modeling. Modelica itself is a formal language that allows sharing and reuse and is useful for long-term storage. In Modelica, models are mathematically described by synchronous differential, algebraic and discrete equations which lead to deterministic behavior and automatic synchronization of the continuous time and discrete event parts of the model. It supports both high level modeling by composition and detailed library component modeling by equations. However, Modelica is only the modeling language.

1.3.3 Dymola

Dymola [1] is a Modelica language translator that takes Modelica models and converts them into a form that can be simulated in an appropriate simulation environment. Dymola handles large, complex multi-engineering models. It has an open interface to other programs, includes 3D animation and has real-time simulation.

1.3.4 Simulink

Simulink [11] is an interactive tool for modeling, simulating and analyzing dynamic, multi-domain systems. It allows building block diagrams, simulation of system behavior and evaluation of performance. It has an extensive library of predefined blocks and hierarchical grouping of models and is useful for linear, nonlinear, continuous-time, discrete-time, mixed-signal and hybrid systems.

1.4 Research Purpose

Each of the systems mentioned above provides valuable insight into features that can be incorporated into SpecTRM. Config provides a basis for evaluating interaction between

the human and controller whereas Modelica and Simulink provide very well designed GUI's and simulators.

SpecTRM presently provides a simulator that aids developers in understanding the design and operation of a system, however some of its limitations are that the SpecTRM Simulator's inputs and outputs are primarily text-based and that it does not allow interaction with the simulator while it runs. Visualizations can provide many benefits in helping a software designer and developer analyze a system. In large complex systems, many events and interactions occur simultaneously. Various elements in the system affect the operation of other elements and in large systems with many components; it can be difficult to keep track of all of them. Often, a designer may want to focus on particular elements and their activity in the system. For instance, it could be whether the landing gear is operating correctly given all the inputs from the different integrated systems. As the simulation runs, the user may want to keep track of certain variable inputs and outputs in a clean and efficient way. The goal of this thesis is to provide the user with ways to customize and configure how the information is displayed, both in a 2D and 3D manner, to help facilitate the understanding of the system. Also, as SpecTRM does not provide a way to simultaneously interact with the simulator as it is running, another objective will be to provide access to that interaction so that the user can enter inputs into the system as it operates.

1.5 Thesis Outline

The next chapter will cover post-SpecTRM Simulator Visualizations. This involves taking the outputs of the Simulator and providing the user with methods to configure and customize that information and re-run the simulation with the customized view.

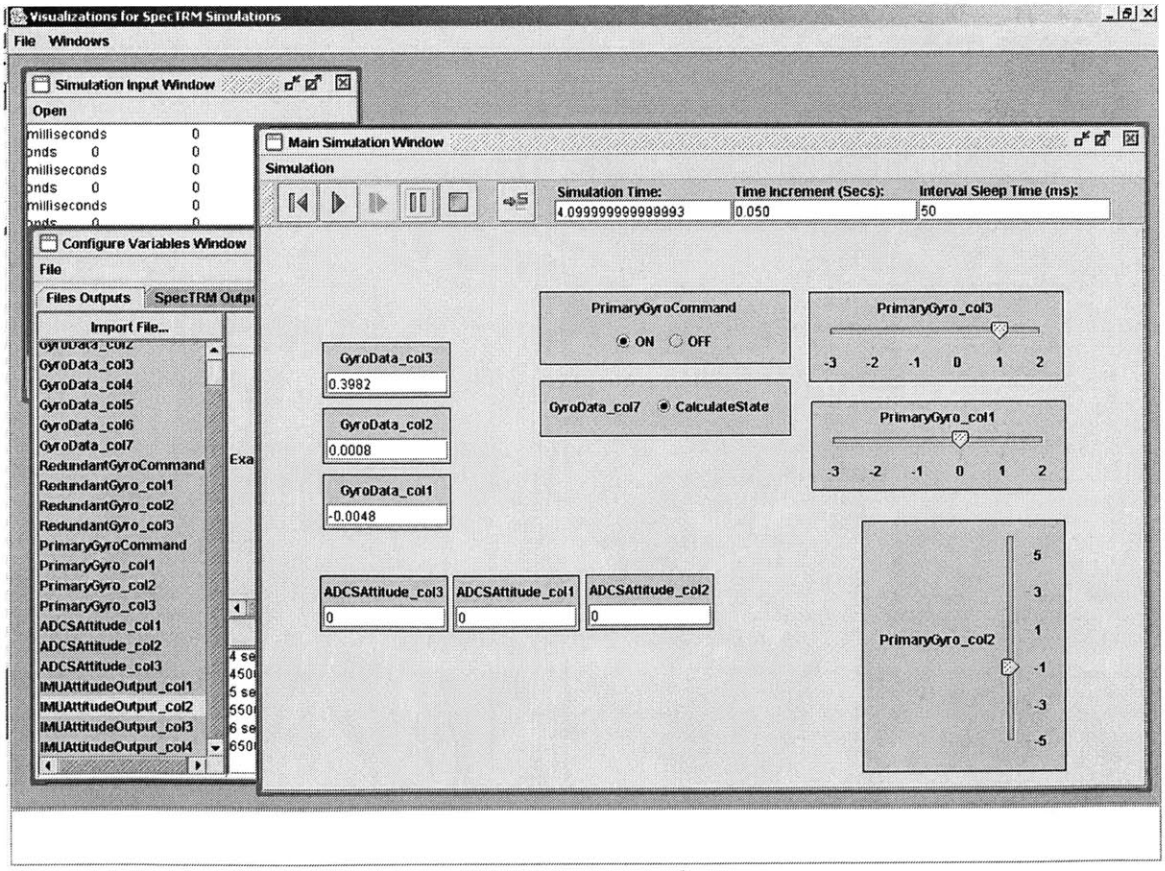
Chapter 3 covers the 3D Visualizations of the simulator output. The 3D Visualizations involve linking the SpecTRM Simulator with a 3D rendering engine and associating the model's elements with transformations to the 3D model.

Chapter 4 deals with making SpecTRM interactive. This involves extending the SpecTRM development environment with an Eclipse plug-in that allows the Simulator to receive external inputs during simulation operation.

Finally, Chapter 5 covers conclusions, lessons learned and suggestions for future work.

Chapter 2

Post-SpecTRM Simulation Visualization Tool



Visualization Tool

2.1 Purpose

The purpose of the Post-SpecTRM Simulation Visualization Tool is to provide the user with another means to understand the SpecTRM Simulator's output. The SpecTRM Simulator has its own visualizations, which were already depicted in Section 1.2.4. The Post-SpecTRM Simulation Visualization Tool attempts to build on those visualizations by introducing two mechanisms: the ability to configure how the output is displayed through a visual widget and the ability to arrange the placement of the widgets.

The visualizations developed here help the user to further understand and visualize the simulation's operation. With SpecTRM, the two primary ways a user can evaluate the outputs of the system are either on screen as the Simulator is running or by looking at the output files generated by the SpecTRM Simulator. The first method is somewhat limited with larger systems as it becomes difficult for users to focus on just a few outputs or elements in the model. The SpecTRM visualization provides a means of viewing the whole system at once, which, while having its benefits, doesn't allow the user to concentrate his or her focus very well. In the second method, the Simulator generates output files where a file represents an element in the model and each line of the file has a timestamp along with the value that is outputted by the model's element. The limitation in this case is understanding how several elements are interacting with one another during the simulation. To do this, one would have to open all the files of interest and investigate the values and changes together.

When the simulations get larger and larger as more elements are added, it becomes harder to understand and follow the simulation results. These visualizations are meant to facilitate understanding of the simulation outputs. The purpose of the Post-SpecTRM Simulation Visualization Tool is to provide users with a way to select particular elements of interest, allow the elements to be configured in various ways for viewing, and then plug these configured elements into an interface for post-viewing.

2.2 Overview

The post-SpecTRM Simulation Visualizations is written in Java. It takes the outputs of the SpecTRM Simulator and allows the user to configure how he wants that data to be displayed. The user decides from the selection of widgets available what kind of widget he or she wants to use to represent the data. Then the user "plugs" these widgets into the simulation window and arranges them as he or she sees fit. Once the user is done arranging the widgets, he or she reruns the simulation and the data is then displayed using the widgets in the simulation window.

2.3 Visual Components

There are three main components to the Post-SpecTRM Simulation Visualizations that are seen by the user and that the user uses to interact with the system. They are:

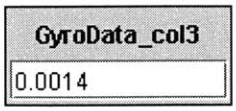
- The Main Simulation Window
- The Variable Configuration Window
- The Simulation Input Window

Behind the scenes, the other main components are:

- Variable Data class
- Simulation Engine class
- Time Line class

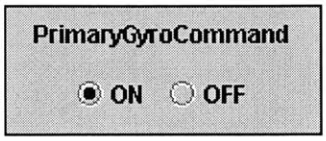
The Main Simulation Window is where the widgets get plugged into and it is also where the simulation is run (or observed). The Variable Configuration Window is where the user configures the variables that get inserted into the Main Simulation Window. And the Simulation Input Window is where the SpecTRM output data is imported into the Visualization Tool.

The main types of GUI Widgets that can currently be used to display information in the system are the Text Widget, Radio Button Widget, Numerical Slider Widget and the Graph Widget.



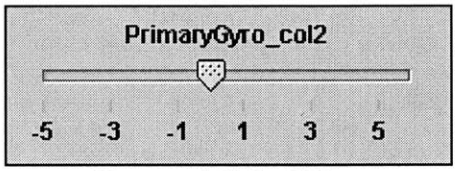
Text Widget

The Text Widget simply displays the value of the element in text.



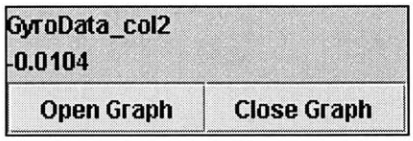
Radio Button Widget

The Radio Button Widget uses pre-defined values determined from the input file as the radio button options.



Numerical Slider Widget

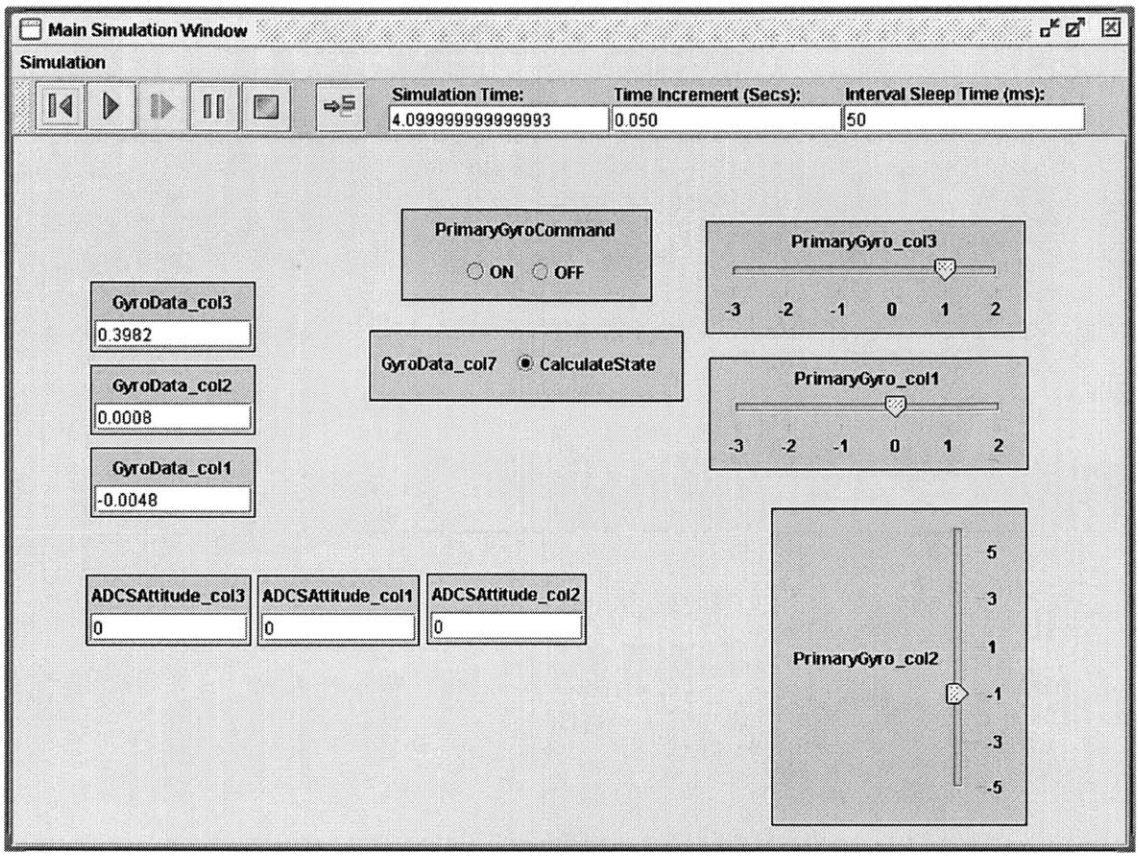
The Numerical Slider Widget uses a slider between two numerical values to display the information.



Graph Widget

The Graph Widget, displays the data for the variable versus the simulation time. The Graph widget opens up another display using Matlab. This Variable will be discussed more in detail later.

2.3.1 Main Simulation Window



Main Simulation Window

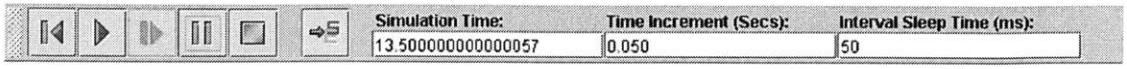
The Main Simulation Window is where all the action occurs. Variables, which encapsulate a model element's information, are represented by GUI widgets and added to this frame. When the simulation is run, the widgets are updated according to their events in the timeline. It is split up into two main parts, the Simulator Controls Frame (on top) and the Visualization Frame (on bottom).

2.3.1.1 Simulator Controls Frame

The Simulator Controls allow the user to control how the simulator operates. It gives the user four direct controls over the simulation:

- Play
- Pause
- Stop
- Return to the Beginning

In addition to these controls over the operation of the simulator, there are also options to go directly to a specific time in the simulation, to set the time step interval (Time Increment) and finally to set the simulation wait time between intervals (Interval Sleep Time).



Simulator Controls

The two time settings (Time Increment and Interval Sleep Time) may be a little confusing. It is important to make the distinction between simulator time and real time. Simulator time is where the simulator is in its timeline.

To clarify, the Time Increment setting determines the time increment between steps in the simulator time. For instance, if the Time Increment is set to 50 milliseconds, then the simulator time increases in intervals of 50 milliseconds. Interval Sleep Time is how long the simulator waits between successive operations (in real time). So, after the simulator increments the simulator time by the Time Increment, it performs all the necessary operations such as updating all the widgets for events that happened within the last simulator time interval. Then, it waits (or sleeps) the Interval Sleep Time before incrementing the simulator time by the time increment. The purpose of waiting this time increment is so the user can adjust how fast the simulator operates. This is important in allowing the user to control the speed in which the simulator updates. Sometimes, the user may want to see the events occurring in real time, but other times, the user may want to investigate closely the values of elements during a short span of time and thus slow down the simulator.

For instance, the user may want the simulator to stretch out one second of simulation time to ten seconds of real time. If the Time Increment is 50 milliseconds, then the Interval Sleep Time should be 500 milliseconds. So, the simulator sleeps for 500 milliseconds after every update (this is assuming that the operations to update the GUI are negligible, which they are not).

2.3.1.2 Main Simulation Visualization Frame

The bottom half of the Main Simulation Window is the Main Simulation Visualization Frame (Visualization Frame). This is where all the widgets are "plugged" into the Simulator for viewing. After variables are imported from their files and configured, the user can place them into this frame. When a user right clicks in this area, a popup menu shows up that gives the user two options. The first option is to show the Configure Variables Frame. Below this option is the list of Variable Widgets that have been configured and that the user can insert into the Visualization Frame. These widgets are divided into widgets that simply display information, and widgets that can be used to input information into SpecTRM. This chapter deals only with the former ones. When the user left clicks on the name of the variable to be added, the widget for that variable appears in the Visualization Frame where the user originally right-clicked.

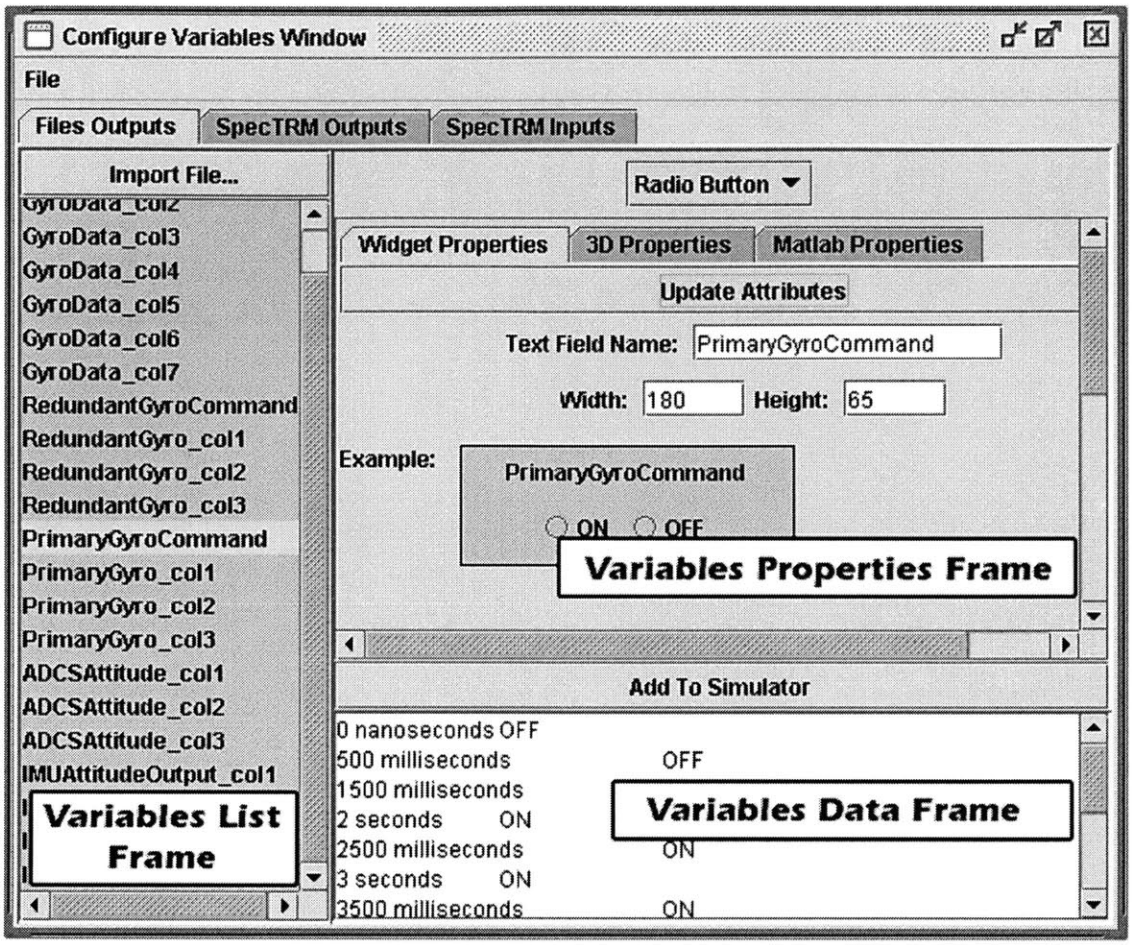
After a widget has been inserted into the Visualization Frame, the user can still configure or adjust its properties in the Configure Variables Window. The user can also move the widget around to position it in a different location. This is accomplished by left-clicking on the widget and dragging it to another location.

The user can arrange all the widget icons in a manner according to the user's preference or need. If the user wants to group some elements together, he or she can arrange them side-by-side. The user can also arrange the elements to resemble how the actual interface may be implemented in the future. For instance, if the user wants to emulate the control panel for a pilot, this gives the user the flexibility to arrange or rearrange all the widget icons in the appropriate manner. This can help the user understand the simulation results in a more intuitive manner (depending on the arrangement) or by association (depending on proximity).

2.3.2 Configure Variables Window

The Configure Variables Window has three sections:

- Variables List Frame
- Variables Properties Frame



Configure Variables Window

- Variables Data Frame

The Configure Variables Window is where users configure the properties of the Variable's widget icon, that is, how the variable is going to be displayed in the Visualization Frame. The Variable is how a SpecTRM model's element is represented in this system. It contains all the information relevant to the model including its name and output data. The user can adjust various properties of the variable's icon, such as its title, size dimensions, the type of widget icon to display the variable, along with other properties specific to the type of widget icon.

2.3.2.1 Variables List Frame

This Frame simply shows the list of Variables that have been imported into the system. The user can select any of the variables in this list for configuration and make them available for insertion into the Visualization Frame.

Note: When a variable has multiple columns of data, each data column shows up as a new variable with the name:

```
[Variable Name] + "_" + "col" + [Column Number]
```

For example, the third row of the variable named "VariableOne" will show up as "VariableOne_col1". This design decision gives the user control over how he or she wants each column displayed. Because each column can be displayed with a different kind of widget, this gives the user flexibility in deciding which widget icon best fits the data to be displayed.

When a user left-clicks on any variable name in the list, its Variables Properties Frame appears in the right top frame and its data appears in the Variables Data Frame.

2.3.2.2 Variables Properties Frame

The Variables Properties Frame is where the user configures the properties of the variable's widget icon. The Variable's widget icon is the graphical icon that represents the variable and displays its information in the Visualization Frame.

First, the user must select the type of widget icon to represent the variable's information. The types of widget icons that are built in right now are Numerical Slider, Text, Radio Button and Graph. These types can be extended or new types can be built (More explanation in the "Variables" section)

The combo box at the top of this frame allows the user to select the variable's widget type. Once one of the types is selected, the property panel for that widget type appears in the center of this frame.

Some options are always available for every widget type. For instance, the "Text Field Name" is the title of the widget icon. The "Width" and "Height" of the icon are also always available. In addition to these properties, each widget icon type also has its own properties that can be adjusted. For instance, for the Numerical Slider, the user can choose the slider's lowest and highest value. The user can also decide if it should be a horizontal or vertical slider. Under all the property options, an example of the icon is displayed when it is first created.

Finally, the "Update Attributes" button updates the widget icon to have the new properties that have been entered.

2.3.2.3 Variables Data Frame

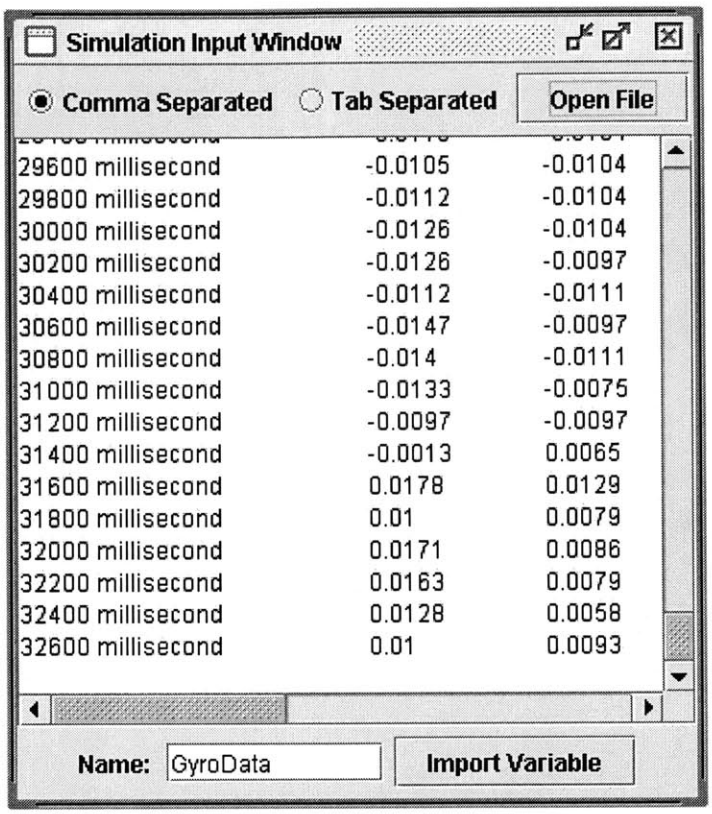
This Frame simply shows the variable's data. The first column is the simulation time in which the data for the variable changes and the second column is the new value of the variable.

2.3.3 Simulation Input Window

The Simulation Input Window is where the user imports information from the files that the SpecTRM Simulator produces. The user can get to this window by clicking on `File->Import File` from the Configure Variables Window or `Windows->Import File` from the main Swing application window.

A typical output file from the SpecTRM Simulator looks like this:

```
3500 milliseconds    0.0  0.0  0.0
4 seconds 0.0  0.0  0.0
5 seconds 1.0  0.0  0.0
6 seconds 0.0  0.0  0.0
7 seconds -1.0 0.0  0.0
8 seconds 0.0  0.0  0.0
9 seconds 0.0  0.0  0.0
```



Simulation Input Window

The first column is the simulator time stamp. The rest of the columns are the data.

The user can open a file for import by selecting `Open->File` from the menu bar in the Simulation Input Window. SpecTRM Simulator output files have the suffix ".csv" for comma separated value files. The Simulation Input Frame also gives the user the option to input files that are tabbed separated files. After a file is opened, the user can change its name. When he or she clicks on "Import Variable", the variable is created, imported into the system and its name is added to the Variables List Frame. If it has multiple columns, it is represented by multiple variable names in the Variables List Frame as described above in [2.3.2.1].

2.4 System Components

The next couple of sections deal with the actual system implementation. These chapters will help the user understand the system for future extensions, improvements and

changes. The primary components that will be discussed are the Variables Data class which holds all the data and can restore the entire state of the simulator, the Simulation Engine class which handles events in the simulator and the Time Line class.

2.4.1 Variables Data

This is the class that handles the state of the Variables in the system. When Variables are imported, added and configured into the system, their state is placed here. This class is implemented by a simple Java Hashtable which holds objects of type Variable. The actual data for each variable is in the Variable class.

This class offers a means of retrieving those Variables for use in the system and a simple means of maintaining all the information.

2.4.1.1 Variables

As mentioned previously, the Variable class represents a SpecTRM model's element. It holds the element's properties, including its output data. This class also includes the element's widget icon representation and its icon configuration panel.

Just a reminder, output elements of SpecTRM come in two kinds, display outputs and output commands [15, p.37]. Outputs describe data leaving the system. A Variable is a representation that encapsulates the data that is outputted from the System. The Variable's name is typically the element's identifier in the SpecTRM model. Its data is the messages that are produced by the SpecTRM Simulation.

Each Variable also has its own properties for configuring its widget icon. When the Variable is instantiated or created, its configuration frame is also created, and a default widget icon for the Variable is created. After the icon is created, the user can then adjust its properties.

2.4.2 Simulation Engine

The Simulation Engine is, as the name infers, the engine for the simulation. It takes care of placing events into the TimeLine and calling the widget icons to update themselves

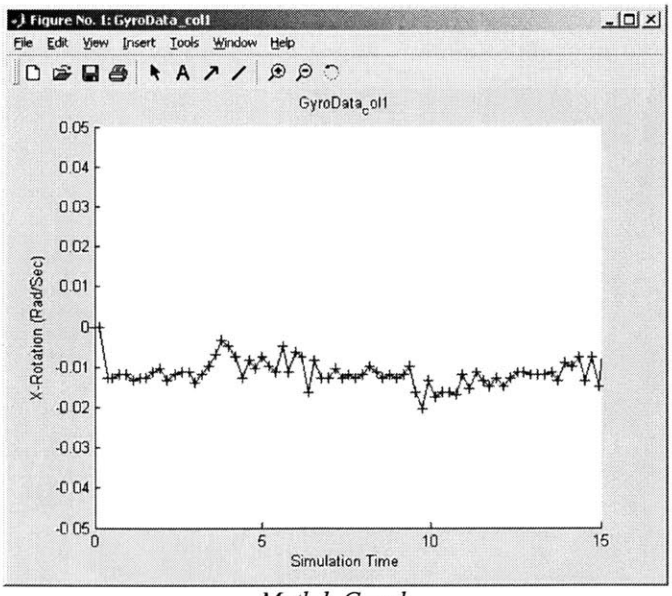
while the simulation is operating. When variables are added to the system, all of their events are added to the TimeLine.

When the user presses "Play" or any of the other simulator control buttons, the Simulation Engine determines which icons are to be updated at the appropriate times and calls these icons to update themselves in the Visualization Frame.

2.4.2.1 Time Line

This is where the events are stored for recall. It simply keeps track of what variables have events at what times. When that simulator reaches a point in time, the Time Line class determines which variables have events and need to be updated and it notifies the Simulation Engine which then calls the widget icon to update itself.

2.5 Matlab and Graphs



Matlab Graph

One of the Variables that can be created to represent an element from a SpecTRM model is the Graph Variable. This variable allows a user to plot an element, which has a single output, on the y-axis, against the time on the x-axis. To create and plot the graphs, Matlab is used. Using Matlab may seem a little excessive for this task alone, but Matlab was also

incorporated using other methods into the visualizations. JMatLink [12], a third-party product was used to establish the connectivity between the visualizations and Matlab.

Matlab is also used to process outputs before they are displayed for the Text Variable and the Graph Variable. If the outputs are numerical, the user can set up the system such that the output is run through a Matlab script before it is displayed on the screen.

Jaime Lien, an undergraduate student, developed most of the Matlab related tools. The next section will give an overview of JMatLink and cover installation of the Matlab tool.

2.5.1 JMatLink

JMatLink connects Java and Matlab through JNI (Java Native Interface) allowing users to take advantage of Matlab's powerful computational engine inside Java applications. JMatLink was developed by Stefan Muller between 1999 to 2001. It does not appear to be in development anymore.

MathWorks, the developers of Matlab also appear to be developing a Java interface for Matlab but as of Release 13 (Version 6.5), the interface has not been made public. The current version of Matlab allows users to write Java applications to be incorporated into Matlab, but does not allow Matlab to be accessed from Java applications which is what the graphing visualization tool requires.

JMatLink communicates with Matlab through JNI. Matlab has a well known C API with various methods like engOpen and engEvalString. JMatLink provides the method signatures and various other methods to allow a user to call the Matlab functions from Java.

2.5.2 JMatLink Compilation and Installation

Because JMatLink has not been in development for a while (the last version of JMatLink was developed using Matlab 5.2), and the Matlab API has been modified since its last release, the last version of JMatLink uses some deprecated C API methods. In particular, JMatLink uses two deprecated methods engPutArray and engGetArray which have been

replaced by engPutVariable and engGetVariable respectively. In addition, the rest of the visualization suite is packaged into `edu.mit.spectrminteractive` while JMatLink was compiled using the default package. This prevents Eclipse from properly importing the JMatLink package. These two issues required JMatLink to be modified and recompiled and also the creation of a new dll file.

Refer to Appendix B for installation instructions and notes on compiling JMatLink.

2.6 User's Guide

This section will describe how to get the system up and running.

Note: Please see Appendix A to make sure that the SpecTRM Visualization Tool package is installed properly before continuing.

2.6.1 Starting Up the Simulator

The Visualization Tools package was designed as a plug-in for SpecTRM. To start the simulator, SpecTRM must first be installed and opened. After opening the appropriate model, from the top menu bar, click `Simulate->Start Simulator`. This will start up the SpecTRM Simulator (which is not actually necessary for visualizing models from the output files, but is necessary to start up the Visualization Tool as it is a plug-in into the Simulator). This action will open up the Visualization Tools for SpecTRM Window. By default, the Main Simulation Window will already be opened as a window inside the Visualization Tools for SpecTRM Window.

Note: This part of the Simulator can more or less be run totally separate from the rest of the plug-in. However, this would involve making sure the right image files and the 3D-related files are in the appropriate directories, then the Simulator could be started by simply calling Java on the Simulator class. But to make things simpler and uniform, we'll always start up the Visualization through the SpecTRM Simulator.

2.6.2 Configuring and Adding Variables

1. To Open the Configure Variables Window

Right-click on the Visualization Frame in the Main Simulation Window (the main area below the controls). This will open up a popup menu. Then, left-click on "Show Config Variables Window", and that will open up the Configure Variables Window. You can also open the Configure Variables Window by choosing `Windows->Configure Variables` from the main menu bar.

2. To Open the Simulation Input Window

When the Configure Variables Window has been opened, the "Files Outputs" tab will by default be selected. Click on "Import File..." below that tab or on `File-> Import` File to open the Simulation Input Window.

3. Importing a File

In the Simulation Input Window, first select if the file to be imported into the system has comma-separated values or tab-separated values. Then click on the "Open File" button to open a file chooser dialog. This will allow the user to choose a file from the file system to be imported into the Visualization Tool. The files that are chosen must be the output (or input) files from the SpecTRM Simulator. When a file is chosen, its data will appear in the Simulation Input Window. The user can rename the element at the bottom of the Simulation Input Window. After choosing a file, click the "Import Variable" button and this will import the variable into the system for use. This causes the Variable to appear in the Variables List Frame of the Configure Variables Window.

4. Configuring a Variable

Select the variable's name in the Variables List Frame. Then, click on "Select type..." to choose a type for the variable. Select a type from the choices that appear in the combo box. After a type has been selected, its Variables Properties Frame will appear in the middle right frame, allowing the user to edit that variable type's properties. Its

example widget icon will also appear at the bottom of this frame. After editing its properties, update the variable's properties and its icon by clicking the "Update Attributes" button.

5. Adding GUI Widget to the Visualization Frame

To add a variable's widget to the Visualization Frame, select its name in the Variables List Frame and click on the "Add to Simulator" button. This will add the variable's name to the popup list that appears when the user right-clicks on the Visualization Frame in the Main Simulation Window. To add the widget to the Visualization Frame, right-click anywhere on the Visualization Frame, then in the popup list that appears, left-click on the name of the variable. The widget icon can be moved to another location by left-clicking it and dragging it to that location.

2.6.3 Running the Simulator

After placing the GUI widgets into the Visualization Frame, the Simulator can be started by clicking the "Play" or "Start" button in the Simulator Controls Frame or by clicking `Simulation->Start Simulation` from the menu bar. The other three buttons, "Pause", "Stop", and "Go to Beginning" can also be used to control the Simulator. The Simulator stops when the last event in the TimeLine has been displayed.

On the right side of the Simulator Controls Frame, the user can adjust the Simulation Time, Time Increment or Interval Sleep Time by simply typing in a new value into the respective text box. The user can type in a particular value for the Simulation Time and the Simulator will start from that new time value. These values can be adjusted anytime during the Simulator's operation.

Chapter 3

3D Visualization Tool

3.1 Purpose

The purpose of the 3D Visualization Tool is to associate a 3D model with a SpecTRM model's outputs and provide continuous three-dimensional rendering during simulation. This tool provides the user with another means to understand the events and results of the SpecTRM Simulation. Through the 3D Visualization tool, users can build a 3D model of the physical object and connect transformations of the model to outputs of the SpecTRM model. This tool provides users with a practical visualization of whether the SpecTRM model is operating as expected.

Incorporated in the tool presently are methods for rotational and translational movement. For instance, a user can build a satellite model and use the outputs of its thruster elements to affect rotational movement on the x, y or z axis. The 3D

Visualization Tool will continuously render the object (satellite in this case) as it rotates and moves.

Currently, the SpecTRM model provides information in a text-based format for the output values from the SpecTRM Simulator. By looking at the output of the gyros alone, it can be difficult to determine if the satellite is operating and responding properly to the inputs. The 3D Visualization helps SpecTRM users ensure that the outputs are producing the desired result by depicting how the 3D model is moving during the simulation.

3.2 Overview

Rendering for the 3D Visualization Tool is based on an open source game-rendering engine: OGRE, Object-Oriented Graphics Rendering Engine. OGRE handles all of the image rendering in the system.

Models are created in Studio Max or any other OGRE modeling compatible program and imported into the OGRE system at runtime (along with the model's textures).

The methods to manipulate the object(s) are written in C++ and implemented in classes that are compiled into Sim3D.dll. The simulator, written in Java, communicates with these methods through Java JNI (Java Native Interface). Using the 3D Visualization Tool interface, SpecTRM outputs are then associated with the appropriate method that modifies the movement of the 3D model.

3.3 OGRE

OGRE [16] is a flexible 3D engine that is written in C++. Its design goal is to make the development of games and demos utilizing 3D hardware easier. It is a scene-oriented engine that provides an interface based on intuitive classes such as world objects while abstracting away all the underlying details and libraries like Direct 3D and OpenGL.

3.3.1 About OGRE

As mentioned above, OGRE stands for "Object-Oriented Graphics Rendering Engine" and is a scene-oriented, flexible 3D engine. It is written in C++ and one of its main design goals was to make it easier and more intuitive for developers to produce 3D games and demos utilizing available 3D hardware. OGRE's class library abstracts away all the details of using Direct3D and OpenGL. It also provides an interface based on intuitive classes such as world objects.

OGRE does not assume a type of game or demo in its design. It uses a flexible class hierarchy to allow developers to design specialized plug-ins for any kind of scene organization. Specialized scene managers can be plugged in to take on the task of rendering efficiently different types of scenery while the rest of the engine functions as before.

OGRE is available under the GNU Lesser General Public License [5] meaning that it can be used in any manner by the developer as long as changes to its core engine are released with the product. The source of the application that is developed from their software does not have to be released.

OGRE also comes with a very well documented API and several tutorials to help novice users get acquainted with the system.

3.3.2 Design Rational

There are several other game engines out there that can be used, but OGRE was chosen for several reasons.

Other game engines that were investigated were Genesis3D [2] and the Quake game engines [6] among others.

OGRE was chosen because of the following reasons:

- Well-Documented API
    - Some of the other game engines simply did not have enough documentation. Some had unofficial API's written by other users and were difficult enough to simply setup and install, much less learn and use.

OGRE has a very well documented API and many tutorials written by other users.

- Open Source
    - Its source code is available, allowing greater flexibility for SpecTRM users for future modifications. Also, because of its involvement in the Open Source movement, other users regularly contribute source code and help others in application programming.
- Licensing
    - By using the GNU Lesser Public License, there are no restrictions on the use of the engine, as long as modifications to it are made public. This does not affect how the 3D Visualization Tool will be using it, as the 3D Visualization Tool is an application of the engine, not actual modifications to the engine.
- Constant Development
    - Some of the more popular engines that are deployed are no longer being developed or no longer have the resources or community to support their development. OGRE has a committed core group of developers.
- Wide User-Base
    - OGRE also has a thriving community where users can ask questions and find help. They also have an IRC channel (Server: irc.freenode.net; Channel: #Ogre3D) where users can pose questions to more experienced users.
- Rendering Rate
    - A game engine-like rendering engine was used because of the particular task at hand. The 3D Visualization Tool needs a continuous rendering engine that can rapidly and efficiently render a scene. Though other kinds of graphics engines may render better-looking images, OGRE typically renders somewhere between 40-60 frames per second allowing real-time simulation rendering.
- Not Made for a Particular Purpose, Easily Extensible

- Some of the engines investigated were simply too narrow in their purpose and focus. OGRE is easily extensible and users are always building different kinds of plug-ins for its use. In the future, when the needs for the 3D Visualization Tool grow, OGRE should more easily adapt to the changing needs of SpecTRM users.

The main difficulties with OGRE is that it is quite complex (and somewhat bulky) and its installation procedure (for future extension and development) is quite time-consuming and complex. Simply using it is a matter of having all the right libraries and media files in the right place. However, adding new functionality to the 3D Visualization Tool requires an understanding of OGRE and the 3D Visualization Tool development cycle.

Genesis3D

Genesis3D is also a real-time 3D rendering environment, similar to OGRE. However, Genesis3D had several concerns. Genesis3D simply did not have a well enough documented API. In addition, the tutorials and samples were not as good as OGRE. Often, they were difficult to understand and follow. Performance-wise, OGRE also outperformed Genesis3D in rendering time (frames per second) and CPU usage, which was an important consideration, considering SpecTRM and the Visualization Tool's intensive CPU and memory usage. Also, it was not clear whether Genesis3D was still being developed as much as some of the other game engines. The updates on their website and news comments were not as recent as other Game engine's websites.

Quake game engines

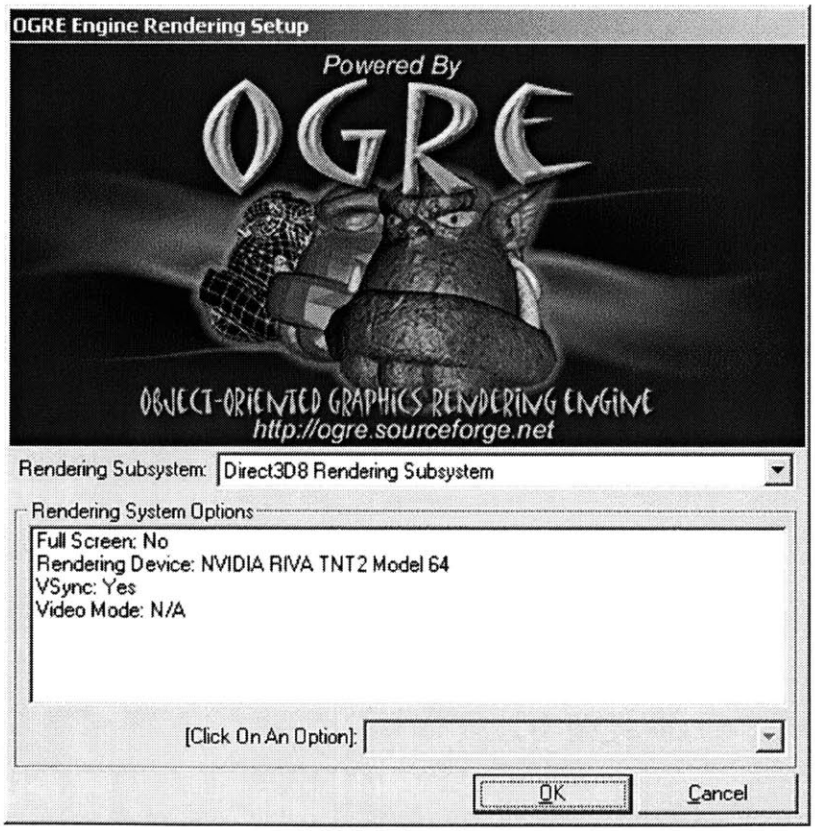
Though the quake engines have excellent performance, wide user-base and many other good merits, the greatest problem was the licensing issues with the most recent versions of its engine.

3.4 Using the 3D Visualization Tool

The 3D Visualization Tool provides various methods that the user can associate with a SpecTRM model's outputs. They consist of adding an object to a scene and controlling its rotational and translational movement. However, if the user wants to define new types of movement or extend its current capabilities, the user needs to install the development environment and modify the 3D Visualization Tool code.

This section will cover how to use the pre-packaged methods developed in the 3D Visualization Tool.

3.4.1 Loading the OGRE Engine



OGRE Splash Screen

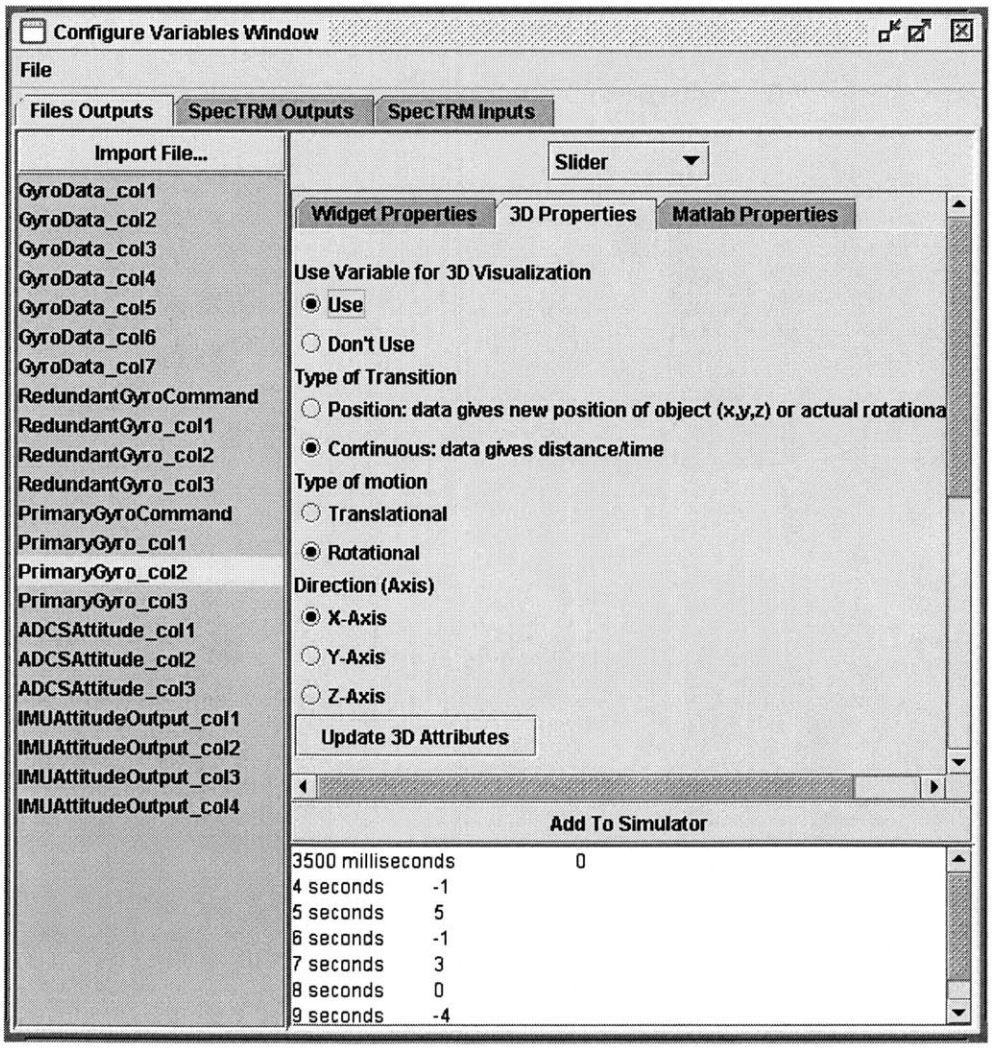
To load the OGRE engine, click on `Windows->3D Simulation Window` from the main application menu bar. This will open up a window with some test functions for the

3D engine. Click on `Simulation->Start` from the 3D Simulation Window to load OGRE.

Here, the user can choose which rendering subsystem to use, either Direct3D (versions 7 or 8) or OpenGL. The user can also select some other options such as whether or not to render full screen. After the choices have been selected, click the "OK" button and OGRE will start rendering.

The engine will continue in rendering mode until it is closed.

3.4.2 Associating SpecTRM Outputs to the 3D Model



3D Configuration Window

At the time this document was written, the 3D Visualization Tool only supports the rendering and transformation of a single object. There are two types of transformations that are permitted, rotational and translation. For rotational transformations, the input can be of two types. The first input type is the actual rotational speed (radians/second) of the object. The second input type is an incremental rotational speed (for instance, add or subtract one radian/second to the current rotation on the x-axis). For translations, the only input currently accepted is the actual coordinates of the object. Without an underlying physics system, it was too difficult to implement an accurate system for translations that takes into account actual distances, physics and other factors.

To associate the output values from an element to the 3D model, click on the 3D Properties tab in the Configure Variables Window before the variable is added to the simulator. Click on "Use" under the first option and select the other options for that variable. After the options have been configured, click "Update 3D Attributes".

After the 3D Attributes have been updated, then the variable can be added to the simulator. Before the simulator is started, make sure the 3D window is opened for viewing.

3.5 Developing the 3D Visualization Tool

This section covers how to extend the 3D Visualization Tool. The methods that come packaged with the 3D Visualization Tool are only some methods that touch the surface of OGRE's capabilities. Future SpecTRM users will have different 3D visualization needs and will no doubt find the need to extend what the tool is capable of presently. This section covers how to install the development environment for OGRE and how to write new functions that are more specific for the user's needs.

3.5.1 Overview

There are two parts to the 3D Visualization Tool. On one side, we have the 3D renderer that includes OGRE and all its sub-parts. On the other side, we have the interface to the rest of the Visualization Tool.

The OGRE side is developed using Visual C++. This includes writing the methods that implement the actual creation and movement of objects in the scene. These methods, like "rotateX" for rotation on the X-axis of an object, are packaged in "Sim3D.dll". On the other side is an interface with the rest of the Visualization Tool, written in Java, which allows a user to take outputs from the SpecTRM Simulator and use those output values to perform an action on the object in the scene.

These two pieces are connected through Java's JNI (Java Native Interface) which allows developers to integrate native code with Java.

3.5.2 Necessary Software

Listed below are the necessary software packages needed to develop and extend the 3D Visualization Tool's capabilities.

1. Microsoft Visual C++ 6
2. Visual C++ Service Pack 4 (or higher)
3. OGRE source code
    a. (download from ogre.sourceforge.net)
4. STLPort 4.5.3
    a. (download from http://www.stlport.org)
5. DirectX 8.1 SDK
    a. (download from http://msdn.microsoft.com)
6. 3rd Party Plug-ins for OGRE
    a. (download from http://ogre.sourceforge.net/downloads/)
7. JNI for JDK
    a. (download from http://java.sun.com)
8. Sim3D.dll code
9. Java Visualization code

OGRE is written in C++. Visual C++ (with Service Pack 4 or higher) is required to compile it and to write new methods utilizing its API (which results in the Sim3D.dll). STLPort 4.5.3, the Direct X 8.1 SDK and the 3rd Party Plug-ins are also required for

OGRE compilation. The latest JDK (Java Development Kit) usually comes with JNI installed and it is required to create the Java header files (more on this later) and to compile the Sim3D.dll. The Java Visualization Tool code has the method names which call the methods in the Sim3D.dll.

To build 3D models for incorporation into the scene, the following are required:

1. 3D Studio Max or some other OGRE-compatible modeling program.
2. 3D Studio Max Exporter
    a. (download from http://ogre.sourceforge.net/modules.php?op=modload&name=Downloads&file=index&req=viewdownload&cid=3)

3.5.3 OGRE Compilation

This section will cover how to get OGRE compiled properly. First, the necessary software packages need to be installed. Then the Visual C++ environment has to be set properly after which OGRE can be compiled.

3.5.3.1 Installing Packages

OGRE, because of its sophisticated nature and reliance on other third-party products, requires the installation of a various tools and software packages before it can be compiled properly. Below is a list of directions for compiling OGRE.

Note: This version of the 3D Visualization Tool was built using the specific software below. Though I believe it will work with new versions (such as .NET and later Direct X SDK's), I only worked with the available resources.

1. Install Microsoft Visual C++ 6 (OGRE and everything should work with .NET also).
2. Install VC++ Service Pack 4.

a. Available at: http://msdn.microsoft.com/vstudio/downloads/updates/sp/vs6/sp4/default.aspx

3. Install STLPort 4.5.3
    a. Available at: http://www.stlport.org/download.html
    b. Open the command prompt in windows.
    c. Run `VCVARS32.BAT` from the VC++ folder. Usually, this file is located in: `{Visual Studio Root}\VC98\Bin`
    d. Go to the STLPort install directory.
    e. Type `'nmake -f vc6.mak clean all'` (in the same command prompt window)
    f. Then type `'nmake -f vc6.mak install'`
4. Install DirectX 8.1 SDK
    a. Available at: http://www.microsoft.com/downloads/details.aspx?FamilyId=DA65CA82-0AE9-412A-B278-7C0AFC0D5974&displaylang=en

3.5.3.2 Setting up the Environment for OGRE

After all the software packages and tools have been installed, the Visual C++ environment needs to be set before building OGRE. This involves ensuring that OGRE can find all the necessary library files from the packages.

After starting up Visual C++, open the ogre project file titled: Ogre.dsw. This file is in `{Ogre}\` where {Ogre} refers to the Ogre installation directory. By default, it is named "ogrenew".

In the Visual C++, go to:
`Tools->Options->Show Directories For->Include Files`
The following directories should be included.
Make sure the first directory is:

- `{STLPORT-4.5.3}\STLPORT`

also, make sure the DirectX SDK \include directory is here.

- {DirectX SDK}\include

Make sure that it can also find Ogre.h:

- {Ogre}\ogrenew\ogremain\include

Next, go to:

Tools->Options->Show Directories For->Library Files

Make sure to include:

- {STLPORT-4.5.3}\LIB

and, the \lib director for DirectX

- {DirectX SDK}\lib

After all these are set, make sure to close VC++ and then reopen it.

3.5.3.3 Building OGRE

After all the necessary include directories are set, building OGRE is simply a matter of opening {ogre}\Ogre.dsw again and performing a Batch Build. This will compile all the OGRE source files and several examples that come pre-packaged with the OGRE source code.

3.5.4 Sim3D.dll Compilation

Sim3D.dll is the C++ project that has all the methods for creating objects and controlling their movement. These methods invoke OGRE API calls for creating nodes and scenes and performing transformations on those nodes. It also has JNI wrappers for these methods which invoke OGRE API calls

There are three files related to the Sim3D.dll which are:

- Sim3D.cpp
- Sim3DApplication.h
- Simulation3D.h

Sim3D.cpp has all the JNI wrappers for the method calls. When the main 3D Visualization Tool calls a method to be performed (from the Java side), like translate object A on its x-axis, that Java method is translated through JNI to a C++ method. Those method names are defined in Sim3D.cpp.

Sim3DApplication.h is where the native methods are actually implemented. The scene and its nodes are created here, as well as node transformations are implemented here. These methods are for the most part API calls to OGRE to perform the appropriate transformations.

Simulation3D.h is the Java header file that is generated from the Java class which declares the native method signatures. In the main Visualization Tool system, a class titled "Simulation3D.java" declares these native methods. These declarations simply contain the signature for the native method. Through these declarations or signatures, other Java classes can call the native methods. From Simulation3D.java, a header file is generated for the native method using "`javah -jni`". That header file then contains the formal signature for the native method which is then included in Sim3D.cpp.

The implementation of the native method and the header files are compiled into the shared library Sim3D.dll. Simulation3D then loads this library and makes it functions available for the other classes in the 3D Visualization Tool to use.

3.5.4.1 Setting up the Environment for the Sim3D.dll Project and Compiling

Sim3D was created in the `{ogre}\Samples\Sim3D` directory where most other applications and samples for OGRE reside. This section will cover setting up the environment to compile Sim3D in that directory.

1. The first step is to copy the Sim3D.cpp, Sim3DApplication.h and Simulation3D.h files into the `{ogre}\Samples\Sim3D` directory.
2. After starting up Visual C++, create a new "Win32 Dynamic-Link Library" project.
3. Title the name of the project "Sim3D" and change the location of the project to point to the folder where the files were copied – `{ogre}\Samples\Sim3D`
4. Select "An empty project" when prompted for the type of project to be created.

5. After the new project has been created, switch to the "File View" tab and expand the tree.
6. Right-click on "Source Files" and click "Add Files to Folder"
7. Select Sim3D.cpp from the directory it has been copied to and click "Ok".
8. Right-click on "Header Files" and click "Add Files to Folder"
9. Select Sim3DApplication.h from the directory it has been copied to and click "Ok".
10. Save the project.

Next, the compiler settings have to be changed.

The following settings have to be changed to compile correctly. They are all found under `Project->Settings`. When the user clicks on `Project->Settings` in Visual C++, a "Project Settings" dialog will pop-up to configure the settings.

In this dialog, the following settings have to be entered:

- Debug : General : Executable = `..\Common\Bin\Debug\Sim3D.dll`
- Debug : General : Working Directory = `..\Common\Bin\Debug`
- C++ : Preprocessor : Additional Include Directories = `..\Common\Include`
- C++ : Code Generation : Use runtime library = Debug Multithreaded DLL
- Link : General : Output file name = `..\Common\Bin\Debug\Sim3D.dll`
- Link : Input : Additional Library Path = `..\..\OgreMain\Lib\Debug`
- Link : Input : Object/library modules += OgreMain.lib
    - Add "OgreMain.lib" to the beginning of the .lib files.

Also make sure that Visual C++ can find the Ogre libraries and header files.

- Under `Tools->Options`, Go to the tab titled "directories". Under "Show directories for:" select "Include files" and make sure the following directories are listed there:
    - `{Ogre root}\OgreMain\Include`
    - `{Ogre root}\Samples\Common\Include`

- Under `Tools->Options`, Go to the tab titled "directories". Under "Show directories for:" select "Library Files" and make sure the following directories are listed there:
    - `{Ogre root}\OgreMain\Lib`

Also, for JNI to work, make sure that the JNI version for the Java JDK is currently installed. Most of the recent JDK's come with JNI already installed. Also, the VC++ compiler needs to be able to find the JNI libraries.

- Install JNI for version of JDK being used or install latest JDK
- Under `Tools->Options`, a dialog box will appear. Go to the tab titled "directories". Under "Show directories for:" select "Include files" and make sure the following directories are in there:
    - `{Java JDK root}\include`
    - `{Java JDK root}\include\win32`

Compiling and building Sim3D.dll after all the setup is complete should just be a matter of clicking `Build->Rebuild` All from the Visual C++ menu. The compilation should produce the Sim3D.dll in the `{Ogre root}\Samples\Common\bin\Debug`. Any compiled files will be placed in this directory.

3.5.5 The Visualization and OGRE, How it Works Together

This section discusses how the 3D Visualization Tool interacts with the OGRE renderer.

3.5.5.1 On the Java Side

On the Java side, the 3D Visualization Tool communicates with the OGRE software through JNI (Java Native Interface). The tutorial for JNI is at http://java.sun.com/docs/books/tutorial/native1.1/index.html.

The basic steps are (directly from the website):

1. Begin by writing the Java program. Create a Java class that declares the native method; this class contains the declaration or signature for the native method. It also includes a `main` method which calls the native method.
2. Compile the Java class that declares the native method and the `main` method.
3. Generate a header file for the native method using `javah` with the native interface flag `-jni`. Once you've generated the header file you have the formal signature for your native method.
4. Write the implementation of the native method in the programming language of your choice, such as C or C++.
5. Compile the header and implementation files into a shared library file.
6. Run the Java program.

We've covered steps four and five. This section will cover the first three steps for the 3D Visualization Tool.

One thing to note is that the java classes are all in the package "edu.mit.spectrminteractive". This affects how we compile the header classes.

In Simulation3D.java are the declarations for the native methods. The declarations look like:

- `public native void translateX( float val );`

This is the method that is used in the other classes to translate the object along the X-axis.

In addition to the method declarations, the Sim3D.dll file is also loaded in the Simulation3D.java class. It is loaded with the following code:

```
static {
      try {
            System.loadLibrary( "Sim3D" );
      } catch (UnsatisfiedLinkError ule ) { }
}
```

After the method declarations are written, the class is compiled using the regular syntax.

Since the plug-in was written in Eclipse, Eclipse will compile the class and place it in the appropriate directory. Eclipse puts the Simulation3D.class into

```
{root}\workspace\edu.mit.spectrminteractive\bin\edu\mit\spectrmin
teractive
```

Next, we want to create the header class from the Simulation3D.class file by calling the following function:

- `javah -jni Simulation3D`

in the directory where the Simulation3D class has been compiled to.

This produces a header class with functions that look like:

```
JNIEXPORT void JNICALL Java_Simulation3D_testfunc (JNIEnv *,
jobject);
/*
 * Class:     Simulation3D
 * Method:    testfunc
 * Signature: ()V
 */
```

The only problem with this method signature is that it assumes a flat tree structure. To work properly, the packaging information needs to be included. The signature for this particular method should look like:

```
JNIEXPORT void JNICALL
Java_edu_mit_spectrminteractive_Simulation3D_testfunc (JNIEnv *,
jobject);
```

With the extra keywords "edu_mit_spectrminteractive" between "Java_" and "_Simulation3D" that describe the packaging information.

So, the next step is to change all the signatures in Simulation3D.h to reflect this. Every method has a signature, so each one has to be modified to reflect the packaging information.

3.5.5.2 On the Sim3D side

Earlier, the three files (Sim3D.cpp, Sim3DApplication.h, Simulation3D.h) were simply copied into the source directory; this section covers what those files contain.

The actual function for the above signatures are implemented on the C side in the file Sim3DApplication.h. The function signature should be added in the Sim3D.cpp file. The method who's signature is described above would look like:

```
JNIEXPORT void JNICALL
Java_edu_mit_spectrminteractive_Simulation3D_testfunc(JNIEnv
*env, jobject obj, Real xVal)
{
    try {
        app.translateX( xVal );
    } catch( Ogre::Exception& e ) {
        MessageBox( NULL, e.getFullDescription().c_str(), "An
exception has occured!", MB_OK | MB_ICONERROR | MB_TASKMODAL );
    }
    return;
}
```

Where `app.translateX()` is the function that will translate the object in the X direction by `xVal`. But this is where the code would otherwise be customized.

The way it is set up to communicate with OGRE, the actual function implementation is in the Sim3DApplication.h file. That is where the real communication between this Sim3D program and the OGRE engine occurs. So, `translateX` is a function that is declared in the Sim3DApplication.h and it is essentially an OGRE API call to translate the node in the X direction.

This is what it looks like in Sim3DApplication.h:

```
void translateX( Real distance ) {
          mShipNode->translate( distance, 0.0, 0.0 );
     }
```

Now, we rebuild all the files in VC++

- `Build->Rebuild All`

This covers all the basics for how OGRE is installed and how it operates with the 3D Visualization Tool. Though it installation can be somewhat tedious, its flexibility and power are very applicable and useful.

3.6 Modifying or Adding Functionality

This section will briefly cover the steps to modify or add methods.

1. The first step is to add the new method name's signature in Simulation3D.java.
2. Then, after this class has been recompiled, create the header files again by using "`java -jni`".
3. Take that header class (Simulation3D.h) and make sure the signatures accommodate the packaging information (add edu_mit_spectrminteractive to the header signature).
4. Take that Simulation3D.h and place it in the `{Ogre}\Samples\Sim3D` directory to make sure that VC++ uses it for compiling.
5. In Sim3DApplication.h, create and implement the new method.
6. In Sim3D.cpp, write the signature and call the method implemented in Sim3DApplication.h from there.
7. Recompile Sim3D.dll
8. Take Sim3D.dll from `{Ogre}\Samples\Common\bin\Debug` and place it where Eclipse can find it `{SpecTRM root}\eclipse\jre\bin`.

These instructions should allow users to extend the 3D Visualization Tool's basic capabilities to more easily accommodate their needs.

3.7 Installing the 3D Visualization Tool for Eclipse

The Java classes come packaged with the other visualization tools. However, the dll's, config files and media files need to be placed in the appropriate directories. If the user follows the directions for installation in Appendix A, the necessary files will be placed in the appropriate directories. The following directions are for users who develop the 3D Visualization Tool.

OGRE is dependent on several dll's and other types of files that have to be in the right place for it to run. Because of the way Eclipse works, the following dll's were placed in `{SpecTRM root}\eclipse\jre\bin` directory.

- Sim3D.dll
- OgreMain.dll
- OgrePlatform.dll
- devil.dll
- ilu.dll
- SDL.dll
- Plugin_BSPSceneManager.dll
- Plugin_FileSystem.dll
- Plugin_ParticleFX.dll
- RenderSystem_Direct3D7.dll
- RenderSystem_SDL.dll
- Plugin_OctreeSceneManager.dll
- Plugin_GuiElements.dll
- RenderSystem_Direct3D8.dll

These files can be copied from the `{Ogre root}\Samples\common\bin\debug` directory.

Next, OGRE needs its configuration files to be in the working directory. For SpecTRM, the working directory is its root: `{SpecTRM Root}\` (where the SpecTRM.bat file is). Finally, the Media directory (the directory with the 3D models, textures and other related files) needs to be also in the root directory.

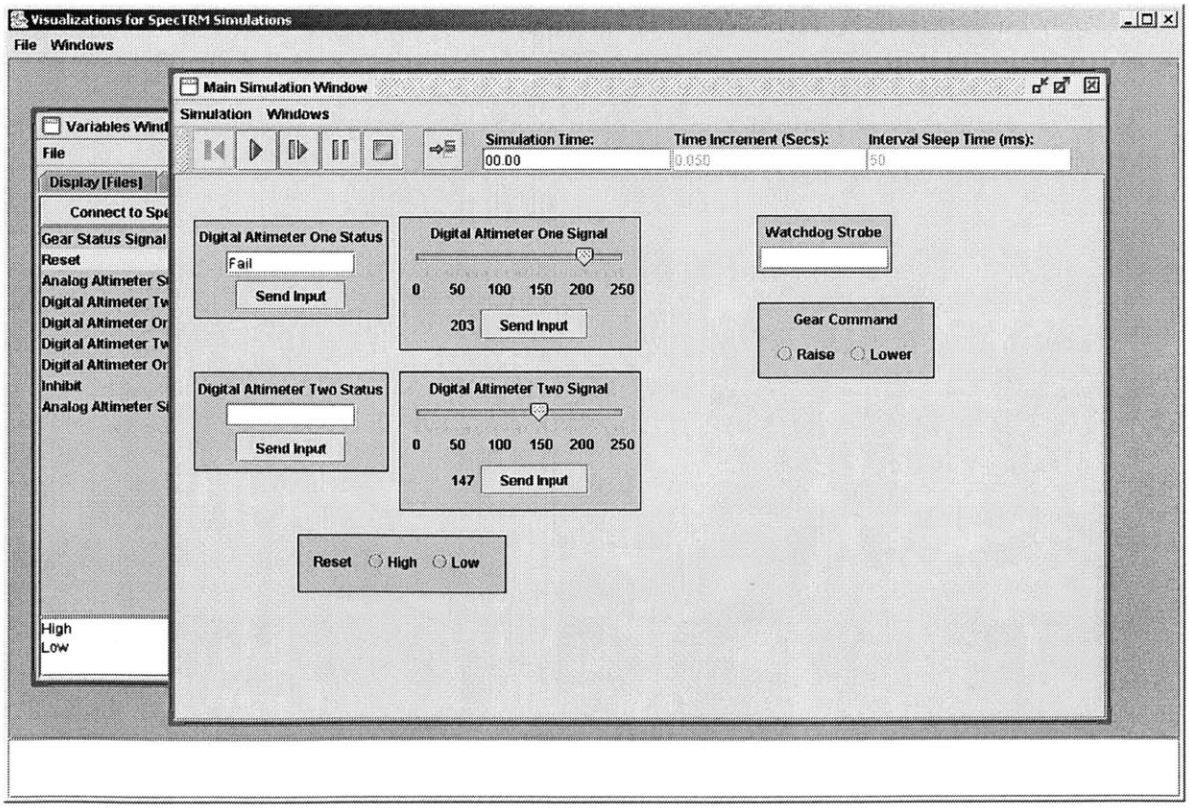
3.8 Difficulties with 3D Visualization

The greatest difficulty with developing a 3D visualization tool for objects in physical space is developing a physics system for the visualization. Neither SpecTRM nor the 3D Visualization Tool provides the physics system that is needed for a full system. Rotational and translation information can be provided to the 3D Visualization Tool in terms of degrees or radians per second for rotational movement or some measurement of distance per second for translation movement, but most output from SpecTRM models do not provide this information. More likely, they provide something similar to: "thruster A: on". That in turn has to be turned into rotational or translation movement that the 3D

system can understand. This will play an important role in future development of the 3D Visualization system.

Chapter 4

Interactive Visualization Tool



Interactive Visualization Interface

4.1 Purpose

The purpose of the Interactive Visualization Tool is to build a system that allows the user to interact with the SpecTRM Simulator while it operates. Presently, inputs for elements in the model are placed into files that the SpecTRM Simulator imports. These files are read at runtime and at the appropriate simulation time, the input values from the file are received into the simulation and assigned to their respective elements.

An example input file looks like:

```
1 second,    300,  Norm
2 second,    290,  Norm
3 second,    280,  Norm
4 second,    270,  Norm
5 second,    260,  Norm
6 second,    250,  Norm
7 second,    240,  Norm
8 second,    230,  Norm
9 second,    220,  Norm
10 second,   210,  Norm
11 second,   200,  Norm
12 second,   190,  Norm
13 second,   180,  Norm
14 second,   170,  Norm
15 second,   160,  Norm
```

In this example, the first column indicates the simulation time at which the element receives its new value. The second and third columns indicate the value that the element should receive. In this example of a Digital Altimeter, the second column is the signal value and the third column is the status value. When the Simulator first starts up, these values are read from the file and whenever the simulator time reaches any of the time values indicated in the file, the value of the Digital Altimeter is set to the corresponding values.

The primary purpose of the Interactive Visualization Tool is to give the user the ability to dynamically alter inputs into the system while the simulator is running. There are various reasons and examples for making the SpecTRM Simulator interactive.

The first reason is to allow the user to interact with the Simulator instead of simply setting up the inputs into the system with static files. As the SpecTRM Simulator

generates outputs, it is useful for the SpecTRM user to be able to respond to these outputs. To test how certain inputs would affect the system in certain states is another reason to make SpecTRM interactive.

Another reason for developing the Interactive Visualization Tool is so that SpecTRM can better model the real system. SpecTRM users can use the simulator to test the model's human interactivity and if the model takes into account human operation and error properly. It can also be used to train operators who will eventually use the real system. As actual system operators use SpecTRM to test the system, data for how people will actually use and react to the system, and even more importantly, how actual people will incorrectly use the system becomes available and can be used to build a better system. This allows another method for fault and error detection, primarily from human operation.

The Interactive Visualization Tool allows SpecTRM to be extended by more than its traditional usage of determining correctness and completeness.

4.2 Overview

The current release of SpecTRM (1.0.0) was developed using the Eclipse Platform [3] by Safeware Engineering Corporation [14]. By using Eclipse, SpecTRM allows third party software developers to write plug-ins that extend its capabilities.

The Interactive Visualization Tool is written as an Eclipse plug-in for SpecTRM. It extends SpecTRM Simulator's API by delivering inputs to the Simulator and receiving outputs from it. Similar to the Post-SpecTRM Simulation Visualization, the user places widget icons into the Visualization Frame to represent elements in the SpecTRM model. However, the difference is that these widgets allow users to input values into the SpecTRM Simulator while it is running. At the same time, the user can use these widgets to display the outputs of the system while it operates.

4.3 Background: Eclipse and SpecTRM

The current version of SpecTRM (version 1.0.0) was built using Eclipse. Eclipse is an integrated development environment that can be used to create a plethora of diverse applications.

4.3.1 Eclipse Overview

The Eclipse platform was designed with several requirements in mind. The following goals for Eclipse are taken directly from their white paper titled "Eclipse Platform Technical Overview." [13]

- Support the construction of a variety of tools for application development
- Support an unrestricted set of tool providers, including independent software vendors
- Support tools to manipulate arbitrary content types
- Facilitate seamless integration of tools within and across different content types and tool providers
- Support both GUI and non-GUI based application development environments
- Run on a wide variety of operating systems.
- Capitalize on the popularity of the Java Programming language for writing tools

Eclipse was designed for building integrated web and application development tooling. It encourages the development of integrated features via its plug-in model. At its core is a discovery mechanism to find plug-ins. The Eclipse platform is built around "extension points" which are well-defined places in the system where other tools (plug-ins) can add functionality.

To find out more about Eclipse, please visit their website [3].

4.3.2 Eclipse and the SpecTRM API

Eclipse provides users with the ability to more easily build plug-ins that extend SpecTRM's capabilities. The Eclipse platform exposes the interfaces for SpecTRM,

making them more clearly defined and easily understood. Through using Eclipse's model of extension points, other users can add functionality to the system more easily.

4.4 Extending the SpecTRM API

This section will cover the basics of extending the SpecTRM API and how the Interactive Visualization Tool utilizes it.

Presently, the Visualization Tool is written as a separate Java Swing application that is incorporated into an Eclipse plug-in for SpecTRM. This section was done with extensive help and advice from Patrick Anderson. Some of the directions for extending the SpecTRM API were provided by him and are simply reworded for this particular situation.

First of all, it is necessary to install the SpecTRM SDK (Software Development Kit). This version of the Interactive Visualization Tool was built using SpecTRM Version 1.0.0 downloaded from www.safeware-eng.com.

4.4.1 Create a New Plug-in Project

As described previously, SpecTRM and all Eclipse projects are extended through plug-ins. The next couple of sections will describe how the Interactive Visualization Tool plug-in is created and what is necessary to build a plug-in for SpecTRM.

Open up SpecTRM by clicking on the `SpecTRM.bat` file in the root directory of the SpecTRM installation.

After SpecTRM has loaded, the new plug-in's environment was set up in the following manner.

1. In the Navigator view, right click and select `New->Project`
2. A dialog box will appear with two frames. In the left frame, select "Plug-in Development". In the right frame, select "Plug-in Project". Then click the "Next" button.
3. Under the "Project Name" field, enter a name for the project. This project was titled "edu.mit.spectrminteractive". Then click the "Next" button.

4. The default settings were adequate for this page. Click the "Next" button.
5. Select "Default Plug-in Structure". Make sure that the "Create a plug-in project using a code generation wizard" button is also selected. Click the "Next" button.
6. It is optional to change the "Plug-in Name" and "Provider" fields. The other default settings were adequate. Click the "Finish" button.

This sets up the initial project environment in SpecTRM. It also creates a directory `{SpecTRM ROOT}\workspace\{Project Name}` where all the source and compiled files will be located. For the Interactive Visualization Tool, all the files were placed in `{SpecTRM ROOT}\workspace\edu.mit.spectrminteractive`.

4.4.2 Adding the Extension Point Definitions

Extension points in Eclipse are defined by XML code. In the plug-in's `project.xsd` file, the XML code defines a set of extension points that the plug-in satisfies. As mentioned above, extension points define how new functionality is added to SpecTRM.

After a new project has been created, the `project.xsd` file is normally automatically opened in the editor view. If not, click on "plugin.xml" in the Package Explorer view.

1. Click the "Source" tab on the bottom of the editor view. Generic XML source generated by the wizard should appear.
2. The code for the following XML snippet was added right before the final tag (which should be `</plugin>`).

```
<extension
      id="edu.mit.spectrminteractive.viewProvider"
      name="SpectrmInteractive"
      point="com.spectrm.simulator.simulatorViewProvider">
   <simulatorViewProvider
         name="SpectrmInteractive"
         class="edu.mit.spectrminteractive.ViewProvider"
         id="edu.mit.spectrminteractive.viewProvider"
         extension="mvc">
   </simulatorViewProvider>
</extension>
```

3. Save the file

Notes on the "extension" tag:

- "id" is a unique value which normally follows the convention of using the base package name and adding a suffix which indicates this plug-in's extension.
- "name" is a user readable string
- "point" is the name of the extension that this plug-in fills.
  This plug-in extends the SpecTRM Simulator, and in the SpecTRM Simulator API, the extension that is used here is the simulatorViewProvider.
- "class" is the name of the class that provides the extension. Here, the class ViewProvider extends the spectrm.simulator.simulatorViewProvider extension.

After the file has been saved, under the "Overview" tab in the editor view of the `project.xsd` file, "com.spectrm.simulator.simulatorViewProvider" should appear under the "Extensions" that are defined for this plug-in.

4.4.3 Adding Required Plug-ins

In the "Overview" tab of the editor view of the `project.xsd` file, the following plug-ins should be listed:

- `com.spectrm.simulator`
- `com.spectrm.core`

1. Click on the "Dependencies" tab in the editor view
2. Click the "Add" button and add the two plug-ins listed above
3. Save

4.4.4 Creating a Class that Implements the Interface

This section covers how to create the class that implements the interface described by the extension. In this case, it will cover how to implement the ViewProvider class.
In the Package Explorer view, open the "src" directory for edu.mit.spectrminteractive.

1. Right-click on the package "edu.mit.spectrminteractive". Under the "src" directory and select `New->Class` and a new dialog should appear.
2. In the new dialog, enter the name of the class that was defined in the "class" tag in the xml document. In this case, "ViewProvider" is the class name.
3. Click "Finish". The new class should open in the editor.
4. Import the appropriate packages by adding the following line under the word "package" near the top of the file.

```
import
com.spectrm.simulator.api.ISimulatorViewProvider;
```

5. Then, add the following line to the class definition:

```
implements ISimulatorViewProvider
```

So, the new class definition looks like:

```
public class ViewProvider implements
ISimulatorViewProvider
```

6. Right-click on anywhere in the body of the class. A popup menu appears. Choose "`Source->Override/Implement Methods...`" and a dialog should appear.
7. Choose the "OK" button since the default settings are adequate. This fills out the signatures of the methods that need to be implemented from the ISimulatorViewProvider.

4.4.5 Implementing the Methods for the ViewProvider Class

By implementing the ISimulatorViewProvider interface, two primary methods need to be written:

- `getRequiredFileExtension()`
- `createView()`

The `getRequiredFileExtension` method simply returns the string "mvc". By returning this string, the extension can piggyback on the normal visualization files that come packaged with SpecTRM. Any simulation that uses the standard SpecTRM visualization will also call the Interactive Visualization Tool code.

The `createView` method is called during the setup of the simulator to contribute elements to the Interactive Visualization Tool views. In this case, the `createView` doesn't contribute an SWT composite, but opens a separate Swing window. `createView` also instantiates three other classes.

- `Simulator.java`
- `InteractiveValueChangeListener.java`
- `InteractiveDataSource.java`

The Simulator class is the primary application that controls everything (same as the Post-SpecTRM Simulation Visualization). The InteractiveValueChangeListener listens for outputs from the SpecTRM Simulator while the InteractiveDataSource takes inputs from the Interactive Visualization Tool and sends them to the SpecTRM Simulator.

4.5 Connecting the Interactive Visualization with SpecTRM

The previous sections covered how the SpecTRM API is extended so that it can properly connect with the Interactive Visualization Tool. The ViewProvider class provides the extension to the SpecTRM Simulator, but the actual work is done in the two classes mentioned above, the InteractiveValueChangeListener and the InteractiveDataSource.

These two classes are the core of the interaction between the Interactive Visualization Tool and the SpecTRM Simulator. The InteractiveValueChangeListener listens to events from the SpecTRM Simulator and sends them to the appropriate elements in the Interactive Visualization Tool for display. When the user interacts and changes the values of elements in the Interactive Visualization Tool, events are created and sent to the Simulator through the InteractiveDataSource.

4.5.1 InteractiveValueChangeListener

Four types of listeners can be added to the SpecTRM Simulator: A DataSink, a DataSource, a SimulatorStateChangeListener and a ValueChangeListener. The InteractiveValueChangeListener class is of the last type (it implements

IValueChangeListener, which listens to changing values in the SpecTRM Simulator). An element in the SpecTRM Simulator changes its value each time it is evaluated, at which point the Simulator will notify all its ValueChangeListeners.

There are three types of events the ValueChangeListener listens for: when input values are received, when output values are sent and when the value of an element changes. Input values are received from input files, other simulations or any other plug-ins that provide DataSources. Output values are sent to files, other simulations or any other plug-ins that provide DataSinks.

In terms of outputs, the Interactive Visualization Tool is only concerned with the outputs from the Simulator, therefore, only the outputValueSent() method is implemented in this class. When outputs are received from the system, the InteractiveValueChangeListener discovers which elements sent them. If that element has a GUI widget in the Visualization Frame, then it calls the widget to update itself with the new value.

4.5.2 InteractiveDataSource

The InteractiveDataSource is the class that provides inputs into the SpecTRM Simulator. When a user enters an input through an element's GUI widget, that input gets passed onto the SpecTRM Simulator through this class.

This class implements IDataSource which has two methods: `setupInputs(ISimulationReferences[] references)` and `getEarliestEventsOccuringBeforeOrAt(long time)`. `setupInputs()` takes the ISimulationReferences from the SpecTRM Simulator (these are references to particular elements in a particular simulation in the SpecTRM Simulator) and attaches it to the corresponding input fields. `getEarliestEventsOccuringBeforeOrAt()` is a method that returns all events that occur before or at the given parameter time. The SpecTRM Simulator periodically calls this function for all of its DataSources. Events for the DataSource are queued up as they come in and when the SpecTRM Simulator calls the DataSource's `getEarliestEventsOccuringBeforeOrAt` method, all the events

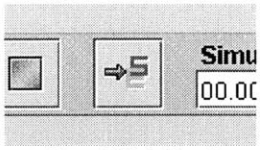
in the queue that happen at or before the given time are returned to the SpecTRM Simulator.

The InteractiveDataSource is a subclass of the AbstractDataSource. AbstractDataSource is a class in the SpecTRM API that implements some of the basic functionality of a DataSource that the SpecTRM Simulator expects (such as the event queue). The two important methods that are over-ridden are `readInputs(ISimulationReference[] isr)` and `noMoreEvents()`. The Simulator calls `getEarliestEventsOccuringBeforeOrAt()` to get the events from the queue, but the class that actually puts events on the queue is `readInputs()`. When there are events available for insertion into the queue, `noMoreEvents()` returns false (The AbstractDataSource thread is periodically calling `noMoreEvents()` to see if there are any new events to be placed on the queue). When `noMoreEvents()` returns false, then the AbstractDataSource thread calls `readInputs()` to read the inputs from its source into the queue. For example, when the DataSource is an input file, then all of the inputs are read when the Simulator starts up after which `noMoreEvents()` will always return true. For the InteractiveDataSource, the user enters events dynamically into the element's GUI widgets. When a new value for an element is entered, `noMoreEvents` returns false until the input is read off of the widget icon and placed on the queue.

4.6 Using the Interactive Visualization Tool

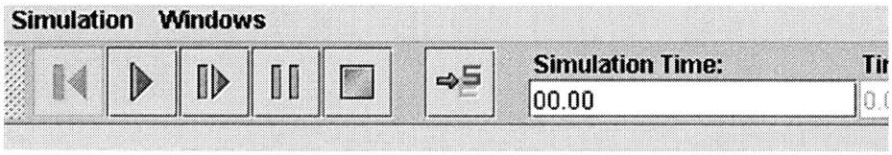
The Interactive Visualization Tool is automatically started when the SpecTRM Simulator is started (in the SpecTRM menu bar: `Simulate->Start Simulator`). Most of the examples and images in this section are from the example Altitude Switch model that comes pre-packaged with SpecTRM.

After the Visualization Tool has loaded, the Main Simulation Window appears. The Visualization Tool is loaded into its default state, which is the Post-SpecTRM Simulation Visualization state. To change to the Interactive Visualization Tool state, click on the "Connect to SpecTRM Simulator" button between the simulation controls and time controls in the Main Simulation Window.



"Connect to SpecTRM Simulator" button

When the user clicks this button, the Interactive Visualization Tool is activated. The Interactive Visualization Tool provides a different set of simulation controller buttons than the Post-SpecTRM Simulation Visualization Tool. Now, the controllers mirror the SpecTRM Simulator controllers.



Interactive Visualization Tool Controllers

These buttons now control the SpecTRM Simulator itself, so you can start, step through the simulator, pause and stop it using these buttons. Under the label "Simulation Time", the SpecTRM Simulator's time is now shown here.

4.6.1 Configure Variables Window

There are two ways to open the Configure Variables Window to configure inputs and outputs into the SpecTRM Simulator. The first option is to right click in the main area of the Main Simulation Window and click "Show Configure Variables Window". The other option is to choose `Windows->Configure Variables` from the main menu bar of the Visualization Tool.

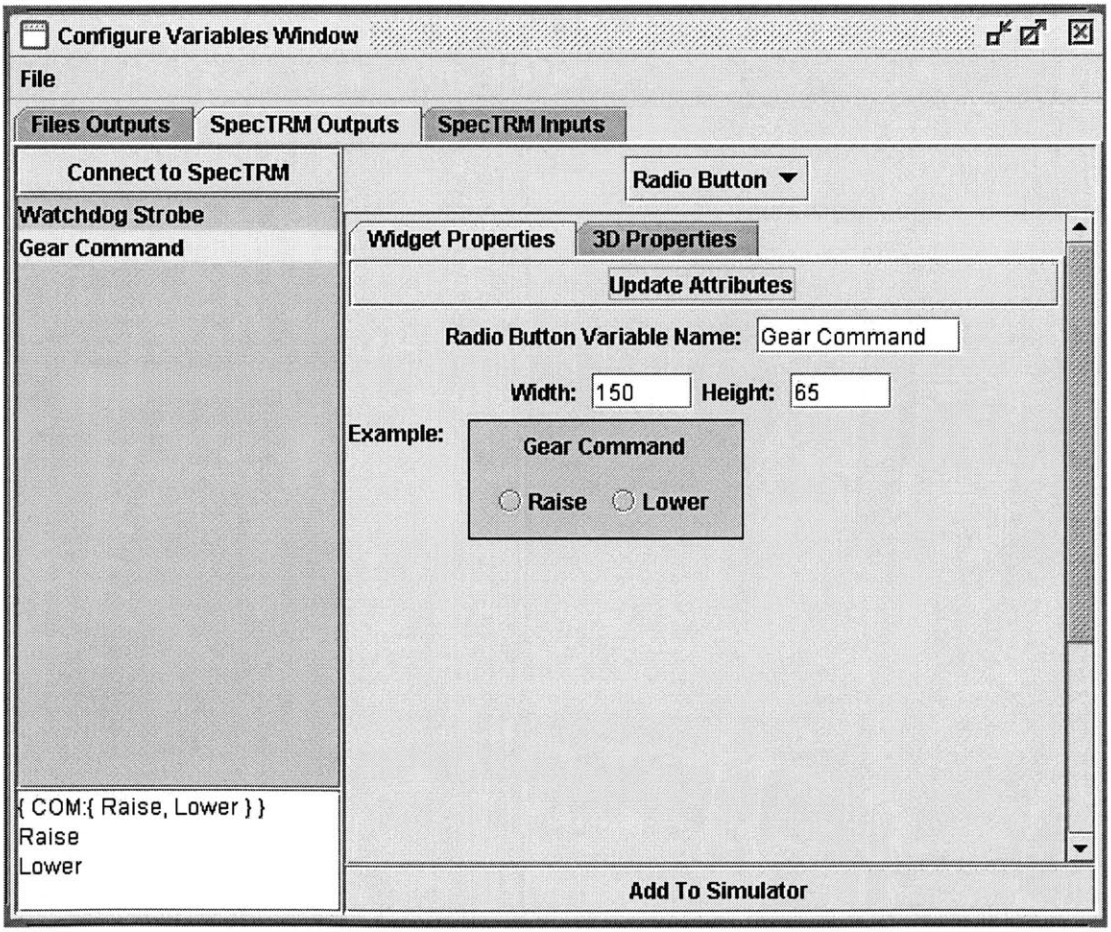
Similar to the Post-SpecTRM Simulation Visualization Tool, the Configure Variables Window is used for configuring the GUI Widget types for elements in the SpecTRM model. It is also used to establish the connection between the SpecTRM Simulator and the Interactive Visualization Tool to make available elements that can be used for inputs and outputs.

When the Configure Variables Window is opened, it automatically opens the tab for file input configuration.

The other two tabs in the window are for configuring inputs and outputs from SpecTRM. The next two sections will discuss how inputs into the SpecTRM Simulator and outputs from it are configured.

4.6.1.1 SpecTRM Outputs

The "SpecTRM Outputs" tabbed panel is used to configure output elements from the SpecTRM models for display in the Interactive Visualization Tool.



Configure SpecTRM Outputs Frame

When the "Connect To SpecTRM" button is clicked on the top-left corner of the tabbed pane, any output elements from the SpecTRM models are displayed on the left hand side.

Clicking on any of the names allows the user to select a type of GUI widget for that element. After a Variable has been chosen, the "Select type..." Combo box on the upper right corner becomes active. Also, when an element is selected from the list on the left side, the possible values that it can take on appear in the bottom right text box.

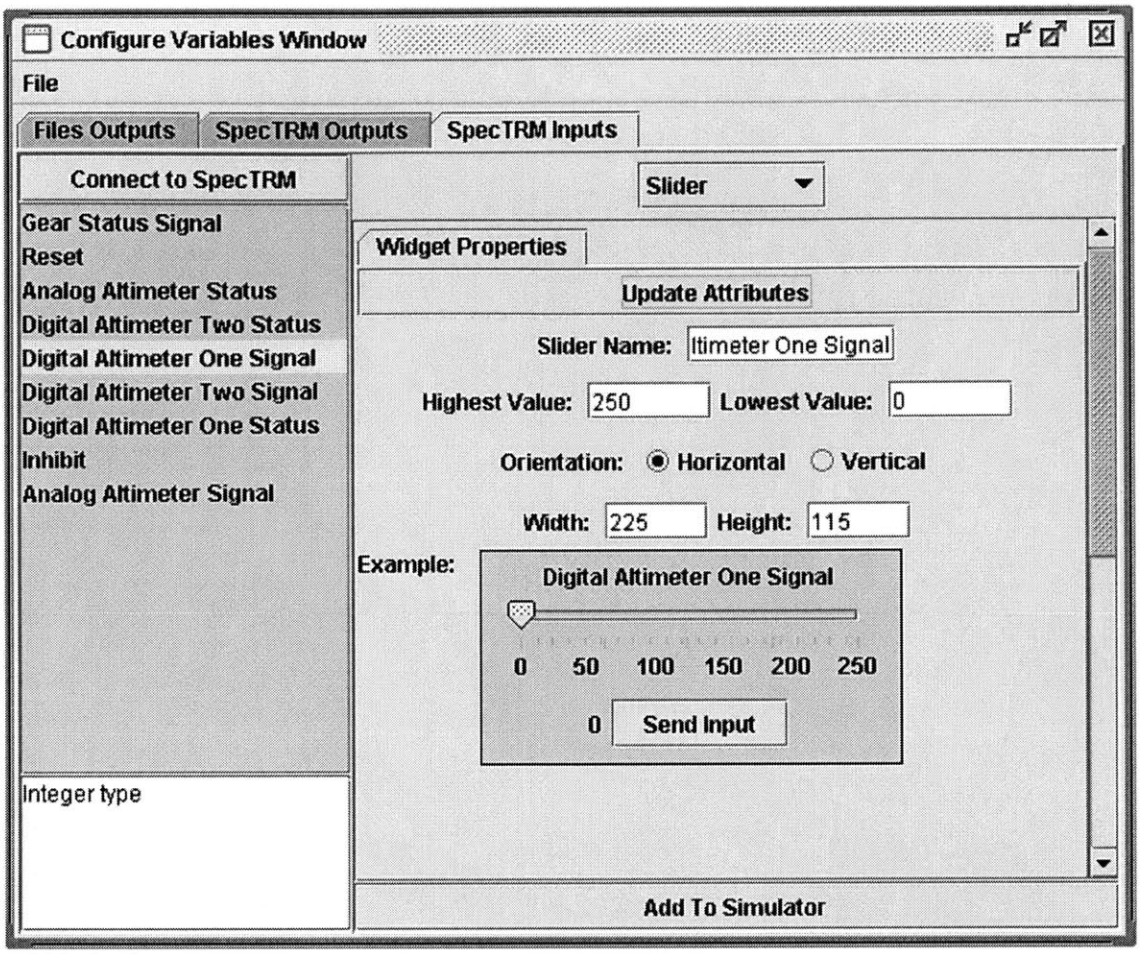
Clicking on the "Select type..." combo box gives the user the available types for the element. When a type has been selected, its configuration details show up in the Variables Properties Frame in the right-center. Each GUI widget type has some basic configuration details that can be used to modify its characteristics. After the GUI widget has been modified to meet the user's needs, clicking on the "Add to Simulator" button makes the widget available for placement into the Main Simulation Window.

One thing of note for outputs is that these variables can also be used for the 3D Visualization Tool by clicking on the "3D Properties" tab and configuring them as described in Chapter 3.

4.6.1.2 SpecTRM Inputs

The "SpecTRM Inputs" tabbed panel is used to configure input elements from the SpecTRM models for display in the Interactive Visualization. As we'll see, inputs and outputs do not differ that much in their configuration, but their use differs greatly. Outputs are used only to display information, whereas inputs are used to interact with the system.

Configuring a SpecTRM Input is mostly the same as configuring an output, with a few differences. Some widgets (Slider and Text icons) now have a "Send Input" button. When the user clicks this button, the event is registered and sent to the SpecTRM Simulator. Also, the Graph Variable is not available for inputs. The possible values of a Variable are also indicated in the bottom left text area. If the possible values of the element are of type enumeration (meaning that there are a closed set of possible values), then that list of values will appear. If the element is of type integer or an open set, then that will also be mentioned. Not every widget type can be used for every element type. For instance, a Radio Button Variable cannot be used to represent an element whose type is integer.



Configure SpecTRM Inputs

4.6.2 Using the Main Simulation Window

The Main Simulation Window is where all the action occurs. The GUI Widgets for elements are added and controlled here, as is the SpecTRM Simulator.

4.6.2.1 Controlling the SpecTRM Simulator

As mentioned earlier, the buttons on the top-left side of the button menu bar in the Main Simulation Window are used for controlling the SpecTRM Simulator. The "Play" button runs the Simulator using the time increment and interval sleep times described in the SpecTRM Simulator model. The "Step" button increments the time one step at a time. "Pause" and "Stop" are self-explanatory.

4.6.2.2 Adding Widgets to the Main Simulation Window

The names of the widgets that have been "added" to the Simulator (see `Configure Variables Window->SpecTRM Outputs` and `SpecTRM Inputs`) appear in a list in the popup menu when the user right clicks on the Visualization Frame (the area below the button menu bar).

When an element's name is selected from that list, its GUI widget appears in the Visualization Frame where the mouse was originally right-clicked. Even after an element has been added to the Main Simulation Window, its properties can still be reconfigured (size and other characteristics). Widgets can also be moved around in the window by right clicking anywhere in the widget and dragging it to another place.

4.6.3 Sending Inputs

Before any inputs can be sent, the Simulator should first be started by clicking the "Play" button in either the SpecTRM Simulator window or the Main Simulation Window. At the appropriate time, the user can enter a value using an element's GUI widget and send its input. For Numerical Slider widgets and Text widgets, the user has to explicitly click the "Send Input" button. For Radio Buttons, when the user selects a new button, the input will automatically be sent to the Simulator. Each input will have the Simulation Time at which it was inputted associated with it. If the user is "stepping" through the simulation, the SpecTRM Simulator will receive the input the next time the user presses the "Step" button. If the user is "playing" through the simulation, the input will be received as the next event.

4.7 Post-SpecTRM Simulation and Interactive Simulation

The two simulations can run simultaneously if the user finds it necessary to have both running at the same time. The user can import output files and configure them for 2D or 3D display. After configuring the output files, the user can also connect the visualization

tool to operate interactively. The simulation time will be based on the SpecTRM Simulator's time

Chapter 5

Conclusions, Lessons Learned and Future Work

5.1 Conclusions

The previous three chapters covered a series of visualization tools that were developed to aid SpecTRM users in understanding and visualizing SpecTRM models. Through the Post-SpecTRM Simulation Visualization Tool, the 3D Visualization Tool and the Interactive Visualization Tool, SpecTRM users now have more options for visualizing the results of a model's simulation.

The Post-SpecTRM Simulation Visualization Tool allows users to take the output of the SpecTRM Simulator and associate widgets icons for the model's elements to view the simulation. Through the 3D Visualization Tool, users can associate the movement of 3D models with an element's output and visualize more practically how that output will

affect the 3D model. Finally, in the Interactive Visualization Tool, users can interact with the SpecTRM Simulator while it operates.

As models get more complex, it becomes increasingly difficult to understand if the outputs produced from the model are correct and sensible. The development of these tools gives users more flexibility to focus on a particular aspect of the model through the visual aids provided through them.

The Visualization Tools developed in this thesis involved a mixture of writing new tools and leveraging existing tools from other areas and then bringing them all together to extend SpecTRM's capabilities. There are many powerful products such as OGRE and Matlab that can increase SpecTRM's utility and the Visualization Tools provided various ways by which they could be leveraged and incorporated into the SpecTRM system.

The main contributions the Visualization Tools for SpecTRM introduced are: the incorporation of 3D visualizations, adding interactivity to SpecTRM's Simulator, attaching Matlab to SpecTRM, and giving the user another means to visualize the outputs through arranging configurable widget icons for SpecTRM elements. Each of these tools provides a different dimension and aspect to understanding and using SpecTRM that was previously unavailable.

5.2 Lessons Learned

Through the work of this thesis, several lessons were learned and these lessons will serve to help future SpecTRM developers with their work.

Through developing the 3D Visualization Tool, a powerful, extensible and flexible graphics engine was found and incorporated into the simulation. Though it was a difficult process in determining a graphics engine to fit the requirements and needs of SpecTRM users, OGRE's capabilities made it very well suited for this application. Although OGRE is somewhat time-consuming in its setup and installation process, its flexibility and future extensibility will definitely be beneficial in the long run. Future users will have the benefit of studying how the 3D Visualization Tool was built and use that knowledge to extend SpecTRM for their own needs.

Through the work on the 3D Visualization Tool, it was also made evident that with any kind of 3D rendering package, it is necessary to have a subsystem that deals with the physics outside of the black box system. That was definitely beyond the scope of this project and is something worth looking into in the future.

The Interactive Visualization Tool both exposed and made it clearer what is involved in writing a plug-in for SpecTRM. It involved learning both about Eclipse and the SpecTRM software's design, which was somewhat difficult as SpecTRM kept evolving over the course of this project and the documentation in some areas was limited. However, through the development of the Interactive Visualization Tool, future developers can take what was learned in the process to help write their own plug-ins for the Simulator.

Because Matlab's Java API is not yet completed and its documentation is not available, the visualizations had to make use of JMatLink which in turn uses JNI and the Matlab C API. Through the work on the Graph Variable and post-processing of SpecTRM outputs, the power of Matlab's computational engine was attached to the visualizations.

Through the 3D Visualization Tool, the Interactive Visualization Tool, the Post-SpecTRM Simulator Visualization Tool, and the Matlab extensions, new functionality was added to the SpecTRM Simulator to extend its capabilities. The goal of this thesis was to give the user more tools to understand and utilize the outputs of a SpecTRM model. To various degrees, that was achieved with the tools developed here.

5.3 Future Work

There are several areas of improvement that can be made to each of the tools.

One of the main goals in the future would be to implement all the tools in Eclipse. Presently, the Post-SpecTRM Simulation Visualization Tool and the Interactive Visualization Tool run in a separate Swing window from the SpecTRM Simulator. However, this task may be somewhat difficult until all the interfaces are stabilized and well-documented. Eventually, all of the tools' look-and-feel should be consistent across the entire SpecTRM toolset.

Also, adding more configurable widgets to the available set would be a valuable addition to the toolset. Presently, there are only the Text, Radio Button, Graph and Numerical Slider icons available. These classes extend the basic AbstractInstantiatedVariable and AbstractIcon classes which will make future development of widgets easier and more accessible.

Cleaning up all the OGRE stuff so that it can more easily be packaged with SpecTRM would be another future goal. Presently, because of OGRE's setup, various files need to be in various locations for the system to find them. This coupled with Eclipse's use and determination of workspace and run-time space makes it difficult to keep all the files nicely packaged in one directory. In the future, it would help to figure out a way to make the installation of OGRE's files less difficult.

Recently, as of July 19, 2003, a package titled Ogre4J which is linked from the Ogre website has been made public. This package, a Java interface to OGRE, may be worth investigating to expand the 3D Visualization capabilities. Ogre4J is also released under the GNU Lesser Public License. Presently, adding new functionality to Simulation3D to allow the Simulator to communicate with OGRE involves various steps including recompiling Simulation3D and the Sim3D.dll. This process may be circumvented by using this new package.

In the future, when the Java API (and/or its documentation) becomes available for Matlab, it will also be more efficient to utilize it instead of combining JNI and the C API.

Appendix A: Installing the SpecTRM Visualization Package

All of the files necessary for the Visualizations developed in this thesis are packaged into the zip file `spectrminteractive.zip`. This can be located at:

- http://www.mit.edu/~kenshin/ra/spectrminteractive1.0.0.zip

Unzipping these files into the SpecTRM root installation directory will install the visualizations.

The following directories and files are included:

```
{SpecTRM}\
      ogre.cfg
      Plugins.cfg
      resources.cfg
      (These are configuration files related to OGRE)

{SpecTRM}\images\
      Play24.gif
      Pause24.gif
      .... Other images related to visualization interface

{SpecTRM}\Media\
      Example.material
      satellite.mesh
      ... Other 3D media files, including meshes, textures and images

{SpecTRM}\eclipse\jre\bin

      JMatLink.dll

      Devil.dll
      Ilu.dll
      OgreMain.dll
      OgrePlatform.dll
      Plugin_BSPSceneManager.dll
      Plugin_FileSystem.dll
      Plugin_GuiElements.dll
      Plugin_OctreeSceneManager.dll
      Plugin_ParticleFX.dll
      RenderSystem_Direct3D7.dll
      RenderSystem_Direct3D8.dll
      RenderSystem_SDL.dll
      SDL.dll
      Sim3D.dll
```

(These are OGRE-related libraries including the JNI interfaces for OGRE and Matlab)

```
{SpecTRM}\eclipse\plugins\edu.mit.spectrminteractive_1.0.0
      Plugin.xml
      Spectrminteractive.jar
      Spectrminteractivesrc.zip
```

(These are the visualization files and interfaces. The source files are in the zip file)

Note: These files have to be placed in these directories relative to the root SpecTRM installation directory otherwise the visualizations will not function properly. By unzipping the file in the root directory, they will be placed in the appropriate places. The OGRE configuration files, the image directory and media directory need to be in the SpecTRM working directory (where `SpecTRM.bat` is). All dynamic-linked libraries have to be in `SpecTRM\eclipse\jre\bin` because of how the Windows operating system searches for dll's. And the plug-in itself needs to be in the `eclipse\plugins` directory for SpecTRM to find and use it.

Appendix B: (Re)Compiling JMatLink

The following steps recompile JMatLink to allow it to work with the rest of the visualizations.

1. Retrieve source code from http://www.held-mueller.de/JMatLink/
2. In the source code, the two classes that are needed are JMatLink.c and JMatLink.java. JMatLink.c provides the method implementations in C and communicate with Matlab through the Matlab's C API. JMatLink.java provides the method signatures through which JNI can use to establish a connection between the rest of the Java code and the C API. JNI is discussed more in depth in Chapter 3 with the 3D Visualizations. Please refer to that Chapter for more details or clarification as the instructions are very similar.
3. In both the JMatLink.c and JMatLink.java files, replace all references to engPutArray and engGetArray with engPutVariable and engGetVariable respectively.
4. Import JMatLink.java into the Eclipse plug-in environment where the edu.mit.spectrminteractive plug-in is being developed. Compile the class in Eclipse.
5. Create the Java header file from JMatLink.class.
   From the command prompt, in the `{SpecTRM root}\workspace\edu.mit.spectrminteractive\bin` directory, call "`javah -jni edu.mit.spectrminteractive.JMatLink`". This will create a file titled `edu_mit_spectrminteractive_JMatLink.h` which has the method signatures.
6. Rename `edu_mit_spectrminteractive_JMatLink.h` to "JMatLink.h"
7. Setup the Visual C++ environment.
   The following directories should be included (`Tools->Options->Directories->Show Directories` for)
   a. Include files
      i. `{Matlab root}\extern\include`
   b. Library files
      i. `{Matlab root}\extern\lib`
      ii. `{Matlab root}\extern\lib\win32`
      iii. `{Matlab root}\bin\win32`
      iv. `{Matlab root}\extern\lib\win32\Microsoft\msvc70`
8. The following libraries have to be linked (`Project->Settings->Link->Category->General->Object\library modules`)
   a. `libeng.lib`
   b. `libmat.lib`
   c. `libmatlb.lib`
   d. `libmex.lib`
   e. `libmx.lib`
9. In Visual C++, create a new Windows Dynamic-Link Library project titled "JMatLink".

10. Add the JMatLink.c file to the source files for the project and the JMatLink.h file to the header files for the project. Also, make sure that JMatLink.h is in the working directory for the project.
11. Build the dll, this will produce JMatLink.dll which should be placed in `{SpecTRM root}\eclipse\jre\bin`

Bibliography

[1] Dynasim Company, *Dymola for Your Complex Simulations*, Available from World Wide Web: http://www.dynasim.se

[2] Eclipse Entertainment, *Genesis3D*, Available from World Wide Web: http://www.genesis3d.com/

[3] Eclipse Project, *Eclipse Platform*, Available from World Wide Web: http://www.eclipse.org/

[4] Elmqvist, H., Mattsson, S. E., Otter, M. Object-Oriented and Hybrid Modeling in Modelica. In *Journal Européen des systèmes automatisés* (January 2001).

[5] The GNU Project, *GNU Lesser General Public License*, Available from World Wide Web: http://www.gnu.org/licenses/lgpl.html

[6] ID Software, *Quake Game Engine*, Available from World Wide Web: http://www.idsoftware.com/business/home/technology/

[7] Otter, M., Elmqvist, H., Mattsson, S. E. Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle. (1999)

[8] Otter M., Elmqvist, H. Modelica – Language, Libraries, Tools, Workshop and EU-Project RealSim. In *Simulation News Europe* (December 2000) pp. 3-8

[9] Malin, J. T., L. D. Fleming, D. R. Throop, Predicting System Accidents with Model Analysis during Hybrid Simulation.

[10] Malin, J.T., L. Flores, L. Fleming, D. Throop, 2002. Using CONFIG for Simulation of Operation of Water Recovery Subsystems for Advanced Control Software Evaluation.

[11] The Mathworks Company, *Simulink 5.0.2*, Availabe from World Wide Web: http://www.mathworks.com/products/simulink/technicalliterature.jsp

[12] Muller, Stefan, *JMatLink, MATLAB Java Classes*, (August 16, 2001)

[13] Object Technology International, Inc., *Eclipse Platform Technical Overview*, (February 2003)

[14] Safeware Engineering Corporation, *SpecTRM*, Available from World Wide Web: http://www.safeware-eng.com

[15] SpecTRM User Manual, Safeware Engineering Corporation

[16] Streeting, Steve, *OGRE: Object-Oriented Graphics Rendering Engine*, Available from World Wide Web: http://ogre.sourceforge.net