A Reusable, Real-Time Spacecraft Dynamics Simulator

Jeffrey J. Biesiadecki **David A. Henriquez** **Abhinandan Jain**
Jet Propulsion Laboratory/California Institute of Technology
4800 Oak Grove Drive M/S 198-235, Pasadena, CA 91109 USA

ABSTRACT

DARTS *Shell* (DSHELL) *is a multi-mission spacecraft simulator for development, test, and verification of flight software and hardware.* DSHELL *is portable from desktop workstations to real-time, hardware-in-the-loop simulation environments.*

DSHELL *combines the* DARTS *S/C flexible multibody dynamics computational engine with libraries of hardware models (for actuators, sensors, motors and encoders) into an integrated simulation environment that can be easily configured and interfaced with flight software and hardware for various real-time and non real-time S/C simulation needs.*

DSHELL *is in use by several of NASA's interplanetary deep space missions including Galileo, Cassini, Mars Pathfinder, and several projects in JPL's Flight System Testbed.*

1 INTRODUCTION

DSHELL is a high fidelity, multi-mission spacecraft dynamics simulation package. The five main goals of the DSHELL environment are: 1) significantly reduce the software development required to interface dynamics simulators, hardware models and hardware-in-the-loop devices; 2) eliminate the need for separate interface development efforts across the various testbeds (analysis, software and real-time) within a project; 3) allow easy migration of models between testbeds; 4) allow the easy support of a variety of S/C configurations and models and simulation environments for all the phases of a mission; and 5) allow the easy reuse and customization of hardware models across various missions.

There are specific classes of real-time hardware devices relevant to spacecraft dynamics. DSHELL groups these dynamics-dependent models into actuator, sensor, motor and encoder model classes. Actuators and motors are those devices that affect the dynamics, while sensors and encoders measure the spacecraft state. Hardware models (e.g. gyroscopes, thrusters, and star-scanners) are organized in libraries which can be created or augmented by the user. Each model has a standardized interface to the DARTS computation engine, the external simulation environment, and the user. The object-oriented model library includes extensive instrumentation for giving a user the high visibility into the simulation necessary for effective use as a design, development and test tool.

DSHELL hardware models are real-time because they complete their execution each tick of simulation time. This deterministic performance is required for the DSHELL models to be in "closed-loop" simulation with AACS flight software.

Figure 1 illustrates an example of a "closed-loop" spacecraft simulator implemented using DSHELL and LIBSIM. LIBSIM is a library which uses a data flow paradigm for connecting higher-level device and subsystem models, and provides special features for modeling faults. A LIBSIM model was created to encapsulate DSHELL, so that it would be embedded in the example simulator. This LIBSIM wrapper makes each DSHELL model's data and text interface available to the data flow of the simulation (figure 3).

The LIBSIM library is part of a collection of libraries and tools to enhance and extend the DSHELL simulation environment (See reference [1] and [2]), but are not required for a dynamics simulator. DSHELL is a fully capable environment for developing dynamics-dependent hardware models and for simulating spacecraft dynamics.

2 DSHELL DYNAMICS SIMULATOR

DSHELL is a library implemented in C++ and may be embedded in a higher level simulator as described in section 1. Or, a small `main()` routine can be written to send data between flight software and DSHELL models, and advance simulation time. For model development,

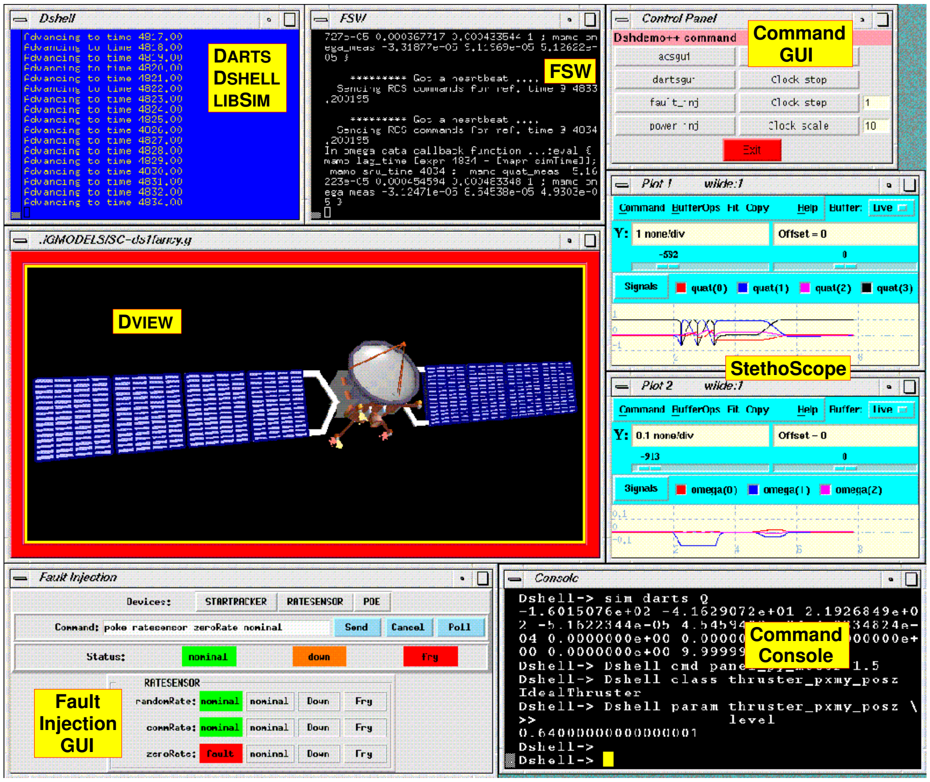

Figure 1: Example spacecraft simulation

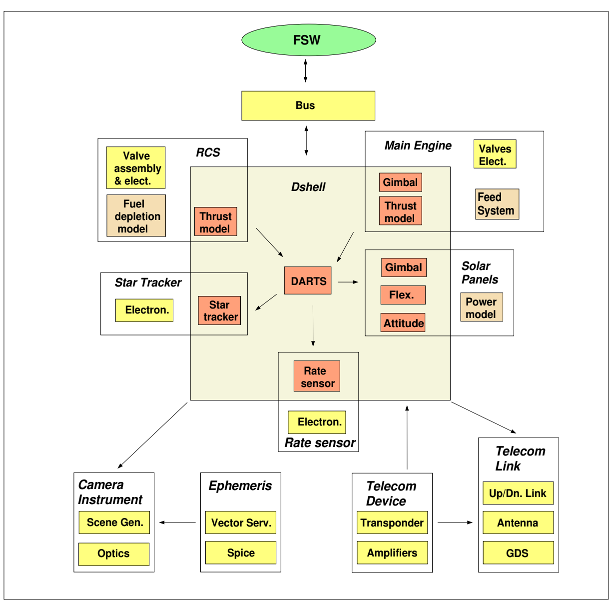

Figure 2: Types of models in a spacecraft simulation

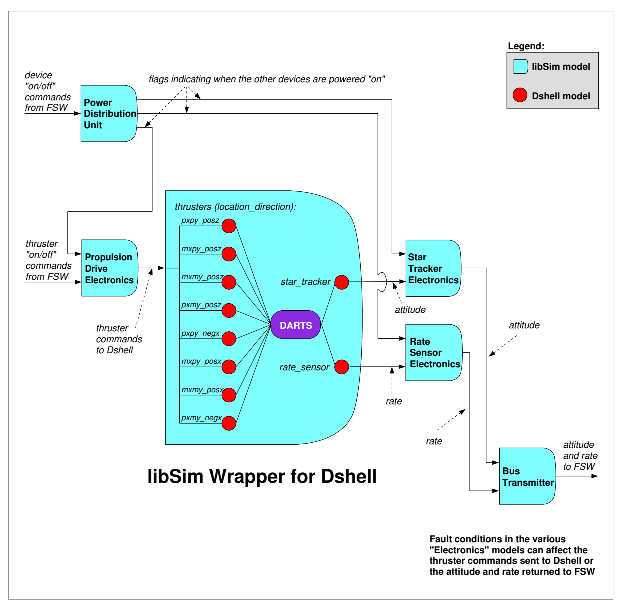

Figure 3: DSHELL wrapped by LIBSIM

a generic "open-loop" version of `main()` is available in which the user controls time and data to and from models. This is useful for writing batch scripts to do regression testing.

Each tick of simulation time is an **I/O step** for DSHELL. Inputs and outputs to and from DSHELL models are expected to occur within that tick. In "closed-loop" simulation, the size of an **I/O step** is set to the length of one flight software RTI. However in "open-loop", the **I/O step** can be set to any interval of time which satisfies the data rate requirements of the simulator. Each **I/O step** consists of an integer number of **integration steps**. And for each integration step, DARTS computes the multi-body dynamics. The size of an **integration step** is set so that numerical stability is ensured.

2.1 DARTS – Dynamics Algorithms for Real-Time Simulation

The DARTS dynamics compute engine [3] implements a fast and efficient spatial algebra recursive algorithm [4, 5] for solving the dynamics of flexible, multi-body, tree-topology systems. It is also used for non-spacecraft applications, such as molecular dynamics [6]. DARTS is a library implemented in ANSI C and is available for Unix and VxWorks platforms.

An analyst provides an input file that is read at run time. If DARTS is used in conjunction with DSHELL, all the DARTS information is written in or sourced in the DSHELL input file. The analyst uses DARTS input to specify the **bodies** that make up the spacecraft, their masses, inertial and flexibility properties, as well as the types of **hinges** that bind them together. There are many types of hinges available to connect two bodies (i.e. pin, U-joint, gimbal, translational, and others). Bodies must be connected in a tree topology, with each body having a single parent body, and the root of the tree being referred to as the **base body**. The **nodes** on each body are also specified in the DARTS input file. Nodes are named locations on a body where forces are applied or dynamics properties are computed. Because the preceding information is not hard-coded, dynamics models can be easily constructed or edited for different missions, without necessitating the recompilation of source code.

2.2 DSHELL Model Classes

DSHELL provides C++ base classes for hardware device models. **Actuators** can impart a force on a node of a body, such as a thruster. **Sensors** are attached to a node of a body and make use of dynamics calculations produced by DARTS for that node. Examples of sensor models include star trackers and gyroscopes. **Motors** are attached to hinges and are used to articulate the bodies that the hinge connects. **Encoders** are also attached to hinges, but are used to determine the position of the hinge. DSHELL device models are massless, and other than applying a force or articulating a body, do not affect the dynamics of the spacecraft. All four of these classes are derived from a common base class (**Model**), which defines data and methods associated with each model (figure 7).

Data for DSHELL models consists of parameters, discrete states, continuous states, commands, and outputs. **Parameters** are values that are set while reading the DSHELL input file upon startup, but are not changeable by the model itself. **Discrete states** are initialized at startup, and may be modified by both the model and the user during run time. **Continuous states** are updated by the numerical integrator in DARTS, and require the model builder to provide a method for computing the derivatives of these states. **Commands** are time tagged data structures sent by flight software, and **outputs** are time tagged data structures sent to flight software. Parameters, discrete states, commands and outputs may be of any basic C data type (such as *int* or *double*), C enumeration, structure, or fixed-size array. Structures may be nested arrays, arrays of structures, and so on. Continuous states are either *double* or arrays of *double*.

There are various methods available for a DSHELL model to define its behavior. `startIoStep()` and `endIoStep()` methods are called at the beginning and end of an I/O step, and are typically used by models to retrieve commands from and send data to flight software, respectively. `startIntegrationStep()` and `endIntegrationStep()` methods are called at the beginning and end of an integration step, and are typically used to compute discrete states. Each integration step, an integrator calls a function to compute the time derivative of the DARTS state vector. This function also calls `preDeriv()` and `postDeriv()` methods for each DSHELL model immediately before and after computation of DARTS derivatives. The `preDeriv()` method is typically used for actuators to apply forces to the nodes that they are attached to. The `postDeriv()` method is typically used to compute the time derivative of any continuous states the model may have. The number of times that these derivative methods are actually called per integration step depends on the numerical integration algorithm selected.

DSHELL models do not interact with each other directly, so the relative order in which their methods are executed is of no consequence. (figure 4).

The base classes provide several methods useful to a model, including methods to get the simulation time, step sizes, and DARTS information. These would be called from the model's `startIoStep()` and `endIoStep()` and other methods described in the previous paragraph.

2.3 DSHELL Model Libraries

Classes for actual device models are derived from any of the four base classes described in section 2.2. The code for model classes may be grouped into reusable libraries, organized by mission, by vendor, or by the type of device. There are several models available for thrusters, gyroscopes, star scanners, accelerometers, and other devices used on JPL spacecraft. They can be used as-is for quick prototype simulations, or as a starting point for developing similar models on a new spacecraft.

An automatic code generator is available to simplify model development and to ensure a consistent interface to all the models. The model developer writes a text file that describes the model, listing the types, names, and descriptions of the parameters, states, commands, and outputs associated with the model. The DMODEL graphical user interface is available for generating this description file (figure 5).

2.4 DSHELL Model Libraries

The code generator takes the model data description file as input, and generates a C++ header file and stub source file for the model class (figure 6). The developer then fills in methods (`startIoStep()`, `endIoStep()` and the rest) as needed to define the model's behavior. Very little knowledge of C++ is needed, but it is useful to be familiar with C.

The automatic code generator also makes an **interface class**, specific to the model class that the developer is defining (figure 7). It is never necessary for the developer to change this code. This class provides model-specific functions to issue commands and retrieve outputs from a model. It also generates code commonly needed to define a text interface to the model's data, and other methods needed by DSHELL. The command and output functions would typically be called from the simulator or `main()` routine that

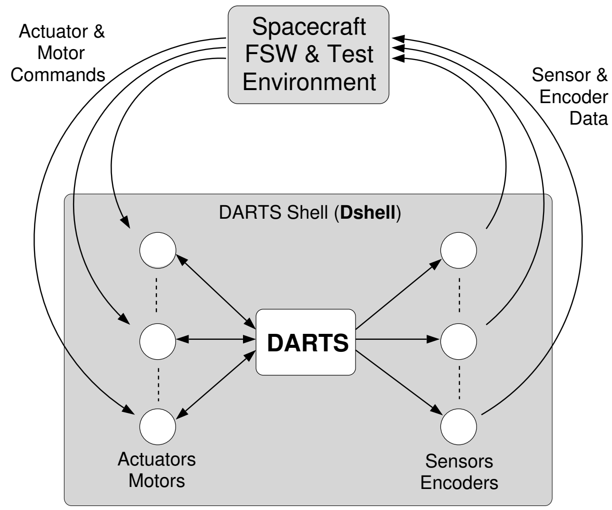

Figure 4: Typical data flow for a DSHELL simulation

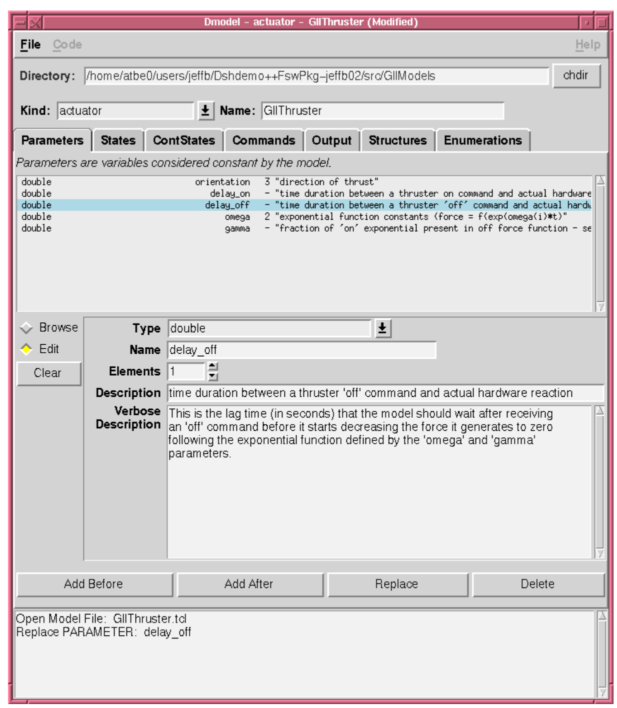

Figure 5: GUI for Dshell model building

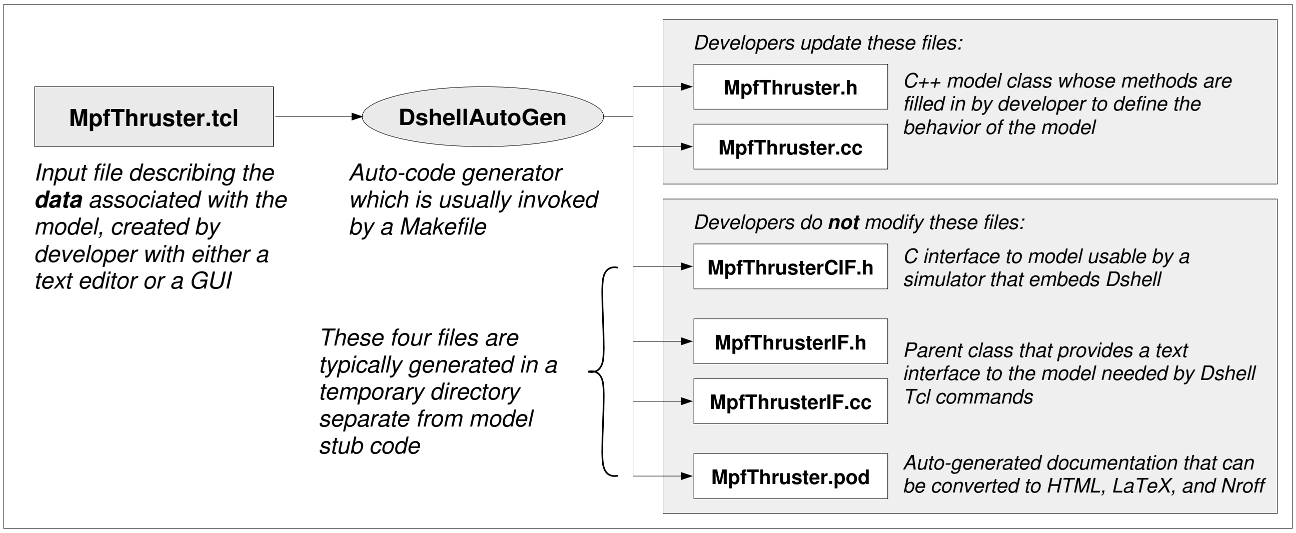

Figure 6: Input and Output Files for the DSHELL Automatic Code Generator

calls other DSHELL routines. They are model-specific to keep them type-safe (avoiding the use of **void *** pointers reduces the occurrence of some programming errors). This also allows a simpler interface for commands and outputs of basic types, and is faster than performing any kind of marshalling or conversion of structures. The code generated for the interface class is meant to eliminate tedious coding by a developer that is typically needed for a model. It is generated in a class separate from the actual model class to clearly delineate code that the developer should modify. This helps keep the code for the stub model class small.

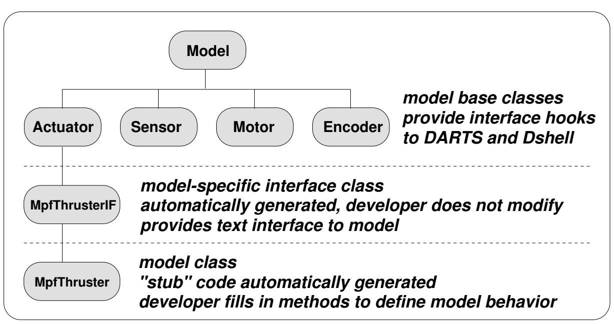

Figure 7: DSHELL class hierarchy

2.5 DSHELL Run Time Environment

The input file containing DARTS information may also contain statements to instantiate models, specifying the model class and instance name. States and parameters for the model may be initialized here as well. Again, not hard-coding this information makes it easier to change configurations without recompiling code.

The free software package *Tool Command Language (Tcl)* [7] is used for the command line and script interface. DSHELL has an extensive set of *Tcl* commands which can be used to get information about the simulation and models therein. In particular, the values of model states and parameters can be peeked and poked from the command line, commands to models can be issued as if they came from flight software, and outputs from models can be examined. There are commands available to query which models are instantiated, the data types and descriptions of model states. A graphical user interface could use the results from the query commands to dynamically display DSHELL state data. A GUI programmer using the query commands would not need to change any code if the simulation configuration changes or new models are added. A prototype of such a GUI has been implemented using *Tk*.

DARTS and DSHELL model state variables can be **checkpointed** to a text file containing "poke" commands. This file can be edited by the user if necessary without needing to know any syntax other than the already familiar *Tcl* commands. On a subsequent run, this file can be used to initialize states and **resume** a previous run.

DSHELL can also keep track of multiple DARTS dynamics models. Alternate dynamics models of the same spacecraft can be selected from (such as in-cruise versus in-orbit models with different fuel slosh behavior, or pre- versus post- probe release). Only one such alternate dynamics model may be active at any given time, and DSHELL device models implicitly interface only to the active model. Or, multiple spacecraft can be bookkept, as in the New Millennium Program's Deep Space Flight 3 formation flying mission. Any combination of alternate models for multiple spacecraft is allowed.

DSHELL models can be activated or deactivated from the *Tcl* command line or startup file. This is useful for debugging, or if there are alternate models for the same device (perhaps one which interfaces to actual hardware-in-the-loop).

It is also possible to schedule C functions and *Tcl* scripts at run time for either one-time or repeated execution. This is very handy for debugging and monitoring variables. It is also useful for interfacing DSHELL to other tools. Such interfaces have been created to Real-Time Innovation, Inc.'s data monitoring tool *StethoScope* and to JPL's 3D viewer *Dview*. Interfaces to other tools can be created in a similar manner, without having to change DSHELL code. Aside from keeping DSHELL code smaller and cleaner, it makes it easy to mix and match interfaces among testbeds which use different monitoring tools.

3 CONCLUSION

A reusable, real-time spacecraft simulator is essential for the design, development, testing and integration of flight software and hardware. DSHELL was made for just such a need. Its real-time performance and design allows real hardware to be swapped with any DSHELL hardware model, and vice versa. This characteristic removes domain boundaries for the types of testbeds in which DSHELL can be used. DSHELL can migrate from a poor fidelity, pure software simulation, to a high fidelity, hybrid hardware and software simulation.

Currently, DSHELL and the ATBE toolkit are being used in the development of Cassini **High Speed Simulator** (**HSS**). **HSS** will be used during Cassini mission operations to test command sequences prior to uplink. It is an adaptation of Cassini's **Flight Software Development System** (**FSDS**), which is the testbed for Cassini's AACS flight software. DSHELL provides **HSS** with a seamless *Tcl* interface to its DARTS state variables and parameters, and its four alternate spacecraft dynamics models. The DSHELL interface also provides visibility into the DSHELL and DARTS data for the external monitoring tools *StethoScope* and DVIEW. The real-time graphical displays of the dynamic state of the spacecraft allow the **HSS** analyst to analyze and debug during the simulation, and post-simulation.

DSHELL has been used by several other NASA's inter-planetary missions. Mars Pathfinder used in its test and verification of its flight software. Galileo is using it in mission operations. And JPL's Flight System Testbed uses DSHELL in several of its current and on-going projects, such as Stardust and Neptune Orbiter.

For more information on DSHELL, visit the DSHELL web site: `http://dshell.jpl.nasa.gov`

4 ACKNOWLEDGEMENTS

The authors would like to express their thanks for the work performed by the other DSHELL team members: Sally Chou, James Fu, Gani Ganapathi, Chester Joe, Patti Koenig and Ling Su.

The research described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

REFERENCES

[1] J. Biesiadecki, A. Jain, and M. James, "A Reconfigurable Testbed Environment for Spacecraft Autonomy," in *i-SAIRAS'97*, (Tokyo, Japan), July 1997.

[2] J. Biesiadecki and A. Jain, "A Reconfigurable Testbed Environment for Spacecraft Autonomy," in *Simulators for European Space Programmes, 4th Workshop*, (Noordwijk, The Netherlands), ESTEC, Oct. 1996.

[3] A. Jain and G. Man, "Real–Time Simulation of the Cassini Spacecraft Using DARTS: Functional Capabilities and the Spatial Algebra Algorithm," in *5th Annual Conference on Aerospace Computational Control*, Aug. 1992.

[4] G. Rodriguez, K. Kreutz-Delgado, and A. Jain, "A Spatial Operator Algebra for Manipulator Modeling and Control," *The International Journal of Robotics Research*, vol. 10, pp. 371–381, Aug. 1991.

[5] A. Jain, "Unified Formulation of Dynamics for Serial Rigid Multibody Systems," *Journal of Guidance, Control and Dynamics*, vol. 14, pp. 531–542, May–June 1991.

[6] A. Jain, N. Vaidehi, and G. Rodriguez, “A Fast Recursive Algorithm for Molecular Dynamics Simulations,” *Journal of Computational Physics*, vol. 106, pp. 258–268, June 1993.

[7] J. K. Ousterhout, *Tcl and the Tk Toolkit.* Addison Wesley, 1994.