

VRML BASED BEHAVIOR DATABASE EDITOR

John F. Richardson

SPAWAR SYSTEMS CENTER San Diego, Code D44202
53560 Hull Street, San Diego California 92152, USA
richards@nosc.mil

Abstract. This paper describes a set of models and related data elements that represent physical properties needed by the simulation algorithms in a virtual environment. This paper also describes a VRML based editor called the SimVRML editor. The data elements represent physical properties used by common models that perform general simulation within virtual environments. The data element class definitions described in this paper can be used by model algorithms and model classes related to the following simulation categories: Vehicle operations, ship and submarine operations, aircraft operations, sensor operations (electronic and visual), communications, role playing, combat operations, industrial, chemical and biological. The SimVRML editor is a virtual environment for editing simulation properties used within general virtual environments that support physical behaviors.

1 Introduction

This paper describes research into data element definitions and model related objects useful for Virtual Reality simulations using the VRML (Virtual Reality Modeling Language) specification. This paper also describes a VRML based user interface for editing static and dynamic state information for VRML based simulation via the Internet or any network in general.

The concept of VRML simulation is commonly referred to as VRML behaviors. The VRML 2 event model, routes and script nodes, in combination with high level programming languages, allows for the use of arbitrary simulation algorithms and models inside of a VRML virtual environment. The high level languages predominantly used for VRML simulation are Java, Javascript, C++ and C.

The VR system for editing and updating simulation static and dynamic state information is the SimVRML editor. The SimVRML editor provides a virtual environment for assigning values to physical properties that are used in VR simulation models and algorithms.

The definitions of the data elements operated on by the SimVRML editor are designed from an object oriented perspective and based upon Java as the VRML scripting language. The data element definitions and model objects can be directly mapped to Javascript or C++ data structures.

Previous research on physical properties for distributed network simulation required by general model algorithms has included the DIS (Distributed Interactive Simulation) protocol [1]. VRML research related to DIS is being carried out by the DIS-VRML-JAVA working group associated with the VRML Architecture Group. An alternative to DIS based VRML is the virtual reality transfer protocol [2]. The

research presented in this paper is about VRML simulations that are completely generic. It provides an extensible framework for the editing of physically based properties used by multifunctional sets of model algorithms.

2 Data Definitions for VR Simulation

This research is based upon the following metaphor. Physically based simulations require models and algorithms to implement the simulation. Large numbers of models exist whose algorithms involve physical properties that are well understood, standardized and can be approximated by a Level Of Detail (LOD) paradigm. Such a “simulation LOD” paradigm allows VR simulation models to use approximate algorithms to manage the computational complexity of simulations within a virtual environment. Managing simulation complexity is critical when immersed inside a VRML base virtual environment that includes multi-user behaviors.

Consider the following example of the simulation LOD paradigm applied to a VR/VE or VRML sensor simulation. In the case of a Radar sensor simulation, the coarsest simulation LOD involves the solution of the generic radar equation. The next finer LOD for radar simulation involves a more physically based model of ground clutter, weather and multipath effects. Finer radar model LOD's can be generated by considering more realistic models of system losses, electromagnetic interference, aspect dependant cross sections, antenna directivity, statistical techniques and even hostile actions among role playing fanatics. Descriptions of the equations for various radar model LOD's and their associated model properties can be found in various sources [3 , 4, 5].

These various radar model LOD's require well defined properties that need to be initialized to realistic values and updated as the VE simulation progresses. The SimVRML editor provides a VRML based interface that addresses the need to initialize and update VR/VE physical properties. These properties involve sensor models along with a set of models that provide for the simulation of common behaviors in a VR/VE/VRML universe.

The models that have properties defined for the SimVRML editor described in this paper are models for vehicles, ships, subs and aircraft motion, along with models for communications, sensors, role playing, combat operations, industrial, chemical and biological. The data element definitions for the properties simulated in the above models are defined as a series of Java classes.

The base class for the SimVRML editor deals with the variables and methods needed to manage data extraction, insertion, descriptive names for the JDBC objects and pointers to default geometries and model algorithms for a set of user defined LOD's. The SimVRML editor has a class that deals with the variables and methods needed to manage data extraction and insertion of dynamic data such as simulation time, pointers to the methods and models used in the simulation, along with status, position, motion parameters and other common simulation housekeeping data.

The SimVRML editor operates on many common Objects that are hierarchical in structure. Such hierarchical objects consist of data elements describing physical properties and also data elements that describe equipment that in turn are described by data elements within other subclasses. Thus, a Vehicle object may have a radio object and an engine object as components of its hierarchical structure.

The SimVRML base class and its subclasses have a public variable that is an enumerated type. The enumeration's are simply ASCII descriptions (less than 33 characters) of the various properties that are shared by the common objects that are simulated via the above models. There are also methods that are used to extract and insert valuations for the properties in the SimVRML base class and its subclasses. Using derived classes and virtual functions and providing an API for the JDBC fields, the SimVRML editor can be made extensible. Thus, a bigger and better model algorithm that needs more sophisticated properties can be "plugged" into the SimVRML editor. Other designers could then just publish their enumerations and "get and put" virtual functions.

Subclasses of the SimVRML base class consist of classes related to physical properties for the following object categories. Hierarchical high level objects like aircraft, ships and submarines, vehicles and people. Low level object categories of the SimVRML editor are sensors, weapons, facilities (Architectural objects), machinery, structural elements of objects, computer hardware and software, chemical and biological. There is also an interaction class that is used by the engagement model and to also keep track of static properties that can be applied to current behavior interactions between objects. The SimVRML base class has common physical properties that are used by all the model algorithms.

3 Model Related Data / Implementation for VR Simulations

In the SimVRML editor system, the number and sophistication of the properties in the enumeration public variable in the various classes is driven by the sophistication of the model algorithms. The simulation LOD for the SimVRML system is different for the various models.

The sensor models and the SimVRML sensor classes representing the various sensor properties have a highly realistic LOD. The sensor model requires properties related to the advanced radar, active and passive sonar, jamming, electrooptical and lidar equations. The radar model requires properties for the basic radar equation, plus properties related to advance features including electromagnetic interference, ground clutter, antenna configuration, variable cross section, countermeasures and multipath effects. The radar model produces a signal to noise ratio that is generated from a discretization of probability of detection curves versus false alarm rate and number of pulses illuminating the detected object. Physical properties are needed for all the input parameters of the various LOD's of the radar equation. Some radar equation parameters for low level radar calculations lump multiple physical functions into one parameter and approximate the parameters physical value. Realistic radar calculations for high level LOD radar models deaggregate the lumped physical properties for advanced radar model features. These high level radar models require appropriate properties for input into the radar algorithms.

The sonar model requires properties for input into the active and passive sonar equations, sonobuoys, towed arrays and variable depth sonar's [6]. Sonar calculations depend upon the marine environment as much as they depend upon the sonar equipment physical properties. For sonar calculations, the environment is divided into 3 sound propagation zones. Environment effects due to transmission / reverberation loss properties are arrays of values for various ocean conditions and

target position relative to the 3 sound propagation zones. The simplest sonar model LOD has table lookup properties for good, fair and bad ocean environments. Sonar classification models require individual target noise properties for subcomponents of an object. Likely subcomponents are machinery (primarily engines and propellers), underwater sensors (especially active and towed) and architectural structures (hull). Target noise properties are required for various speed values. Probability of detection is calculated for individual noise components by dividing the calculated signal to noise ratio by a standard deviation of a normal distribution related to the noise component and then comparing to a normal distribution with mean 1. The primary problem in sonar modeling is generating reasonable physical values for sonar environment and target noise.

The basic Visibility model operates on object detectability properties. Sufficient weather effects are built into the VRML specification for very simple visual calculations. An advanced visibility model based upon electrooptical algorithms [7] requires calculation of the number of sensor pixels encompassing the detected object. Detection probability is generated via an exponential probability distribution that can be discretized into a table look up. The table look up function of the model is based upon optical signal to noise thresholds for the various categories of objects in the virtual environment. If an Infrared or FLIR model is needed then the thresholds require temperature dependence factors.

The Communications model has a moderately realistic LOD. The communications model requires properties related to the basic communications and jamming equations, which are similar to the radar equations without a cross section factor. Properties related to advanced communications features that depend upon clutter, antenna configuration, interference, multipath and terrain are similar to those properties needed by the radar equations to calculate similar radar effects. The Communications network model has a low LOD to calculate path delays. Path delays are calculated by assuming that delay points behaves according to M/M/1 single server queue with exponentially distributed service rate and interarrival time [8]. Properties needed by the communications network models are mostly dynamic properties related to compatibility of network nodes and communications equipment operational status. Such properties reside in the classes used for simulation housekeeping.

The motion model is a low to medium LOD model that requires properties related to basic ballistics, navigation, navigation error, along with maximum and minimum properties (weight, speed, altitude...). The Flight, ship and submarine operations models are similar to the queuing and probabilistic elements of the communications network model but require interaction with the motion model, fuel consumption properties and specific motion constraints. Fuel consumption algorithms require speed and fuel factors for minimum fuel use and also fuel factors to interpolate fuel use at speeds other than the minimum fuel usage velocity. Ship and submarine operations are similar to aircraft flight operations but with different constraints. Motion constraints can be thought of as a set of rules depending on the type of simulation object and the model applied to the object. Navigation errors are drawn from a normal distribution with mean zero and standard deviation specific to the navigational sensor object. The motion model and all the other models require system failure calculations. For failure calculations, we have equation 1 where A is mean time

before failure and B is probability of failure. If B = 0 then the failure time is the current time plus some appropriately large constant.

$$\text{Failure time} = A [(\text{Log}(1-\text{rand}(x)))/(\text{Log}(1-B))] + \text{current time} \quad (1)$$

The engagement model is a combat operations model. The engagement model requires properties related to engagement probabilities (failure, detection, acquisition, hit) and equipment characteristics. Air related engagements are based upon simple probabilistic equations. Other properties related to air engagements and air to ground engagements are sensor activation time, sensor scanning arc, lists of target sensors, lists of targets and weapon lifetime. Ground based direct fire engagements are related to a discretization of the Lancaster equations [9] based upon lookup tables. The lookup tables are indexed based upon the characteristics of the objects engaging. Ground engagements between hierarchical objects, such as groups of vehicles are indexed based upon group characteristics. The results of the lookup table searches are passed to a simple probabilistic model. Ground based indirect fire, which is engagement between objects separated farther than a user specified threshold, is modeled via a probabilistic model. Indirect fire assets such as artillery are also included in the direct fire model.

Engagement damage is related to a set of probability factors for failure, deception and repair along with defensive and offensive characteristics. Damage factors are calculated for the following categories of an object: structural, fuel, equipment, and operational status. Damage factors are simple multipliers that are compared to thresholds that enable/disable/ or repair/destroy components of the object.

It is very hard to generate standard properties for people. This is due to the complexity of people and diversity of opinion on how to approximate human behavior. In the SimVRML system, basic person physical properties would be compared to environmental constraints that would be managed by a basic simulation housekeeping process. Beyond constraint checking, such social models would be relegated in the SimVRML system to managing verbal communication within the VR scene and input / output of commands within the VR simulation.

In the SimVRML system, models for structural, chemical, biological processing and any other remaining behavior functions are queuing models augmented by very low level algorithms. These low level algorithms use the current set of properties as thresholds that either disable / enable the object or switch from one geometry LOD to another. As an example, if the melting point is exceeded then the object consisting of a solid geometry would be replaced by an indexed face set of the melted object. Another example would be the failure of an object that would be used by a damage model to affect the state of other objects in the virtual environment.

4 Model Interaction and Object Interrelationships

In order for the extendable features of the SimVRML editor to function optimally, the interaction of the model algorithms and the object interrelationships need to be defined. The level of complexity of the interrelationships of the object physical properties in an extendable and hierarchical VE generated by SimVRML class schemas is just the local complexity of the data / model interaction. If the virtual

environment is distributed or if control is shared in a local virtual environment then several factors influence the complexity of the object / model interactions. The primary factors are time, transfer / acquisition of control, transfer / acquisition of editing rights to the properties and control / acquisition of model algorithms and model LOD's. Time synchronization depends upon agreement on how to broadcast time, when to declare a time stamp valid and how to interpolate time stamps. Much of the preliminary agreement on time is present in the VRML 97 specification [10] as it applies to nodes and simple behaviors.

In a complex VRML simulation one method of agreement about the overall virtual environment within a distributed simulation is to adopt the rules inherent in the DIS specification. The SimVRML editor adopts a hybrid approach consisting of the DIS rules and methods to override the rules or disallow overriding based upon model / object "rights". The SimVRML base class has methods to extract data sets from multiple object schemas and to set the structure of the data requirements based upon the model algorithms and model interactions.

The results of the model and data interactions requires that model execution, data storage and data extraction be synchronized. One way to affect global control of various model and physical property interactions is to treat the system as though it was a parallel simulation environment. The housekeeping management scheme chosen for the Virtual Environment generates a set of simulation semaphores. These simulation semaphores are used to make sure that data is extracted or stored at the correct times and that models for the physical behaviors execute in a realistic sequence that produces common sense results.

The SimVRML system is designed to be an extendable system with user defined models and physical properties added to a "base system" of models and physical properties. In this case there can be multiple simulation semaphore sets that can be activated or deactivated based upon virtual environment rights possessed by controlling avatars or simulation umpires. Suppose you have a VR simulation that has user defined models appended to a base collection of models and their associated physical properties along with their defined interrelationships. If you start with a realistic simulation semaphore set that produces realistic simulation results, how do you produce a realistic simulation semaphore set for the extended VE containing the user defined models. One way to satisfy the requirement that the extended system produce realistic results is to require a sanity check on the results. The simplest form of sanity check would require the simulation state to be outside of obvious error states. Further sanity checks would compare the simulation state for conformation to test parameter limits. Such sanity checks require well defined test simulations within test virtual environments.

Consider the model and physical property interactions for simple linear motion. First, consider scale. Aircraft and ship positions are best stored in geodetic coordinates while ground motion is best served by Cartesian or Universal Transverse Mercator (UTM) coordinates. Motion inside a building is most likely to be Cartesian. Motion over several kilometers is probably best represented by UTM coordinates. Next, consider weather and environmental conditions. Object states are needed for daylight calculations. Navigation properties have to be combined with object motion characteristics. Motion requires awareness of the surrounding objects that exceeds simple collision detection. Motion requires collection of data regarding the visibility

status of objects and the properties of the collecting sensors. Thus, data must be extracted from multiple object classes. Motion requires execution synchronization between the motion, navigation and detection algorithms at the minimum. If objects are interacting in a manner that alters the properties and geometry of the interacting objects then interaction algorithms must execute. Results of physical object interactions need to be propagated to the other motion algorithms under the control of motion semaphores.

One solution for the simplification of synchronization of model algorithms is to separate behaviors that must be event driven and simulation cycle based. Cycle based algorithms can be applied to all objects in a simulation subject to pre defined conditions. Processing VR behaviors via a Simulation cycle and then broadcasting the results to the VR objects affected by the behaviors solves some of the problems introduced by object and model interactions. Processing via simulation cycles introduces other problems. One problem is related to a requirement that user actions should be able to trigger model execution asynchronously. Event based algorithm execution requires awareness of all other active and possibly planned events and a set of rules and rights to umpire simulation logjams. Very simple model LOD's can be implemented so that events can be reliably precomputed and queued. Complex model LOD's with significant numbers of property and model interrelationships require constant updating of the event queue.

Consider the detection phase of the motion calculation in the event based portion of the VR simulation. The object in motion would have an instantiation in the VR world along with an instantiation of the detector such as eyes or a night vision goggle. There would also be an instantiation of an object that represents the model that is being executed to process the detection. The methods of the "detection model" object would extract the relevant physical properties from various object classes and spawn events to manage and localize the effects of the interactions between the models and physical properties. Users can make their own constructors / destructors and methods for model objects. Thus the user can attempt to manage the problems posed by model and property interrelationships by incorporating management of the interrelationships in the objects representing the models.

One paradigm for managing the relationships between physical properties and also the simulation model algorithms is to use standard object oriented design (OOD) methodologies such as OMT [11] and Shlaer-Mellor [12]. OMT and other OOD methodologies attempt to generate correct relationships between physical properties, events and simulation model algorithms. By applying OMT or other such OOD methodologies to the SimVRML base classes and simulation model algorithms, a reasonable management scheme for the property / model interrelationships of a baseline SimVRML system can be produced.

By using OMT and other equivalent OOD methodologies, users planning to extend the baseline SimVRML system can minimize effects of their user defined physical properties and models upon the SimVRML baseline system. Similarly, baseline rules for simulation cycle based execution within the SimVRML system can be produced by standard methods of semaphore design. The Shlaer-Mellor Methodology has been analyzed mathematically to examine the question of correctness of OOD for event driven simulations [13].

The SimVRML editor can be used to initialize the state of a Virtual Environment containing realistic general behaviors. This initialization procedure is performed by editing the physical properties that are dynamic. The default VRML description of a virtual environment contains a wealth of geometry, position and VR sensor information. Beyond geometry and standard VRML sensors and interpolators, what do you put in the housekeeping classes to attempt to produce a simulation semaphore set that provides accepted results under test conditions? The housekeeping properties should include enough simulation state data to keep track of object states used by the models, simulation rights and the defined semaphore sets. In addition to the geometry inherent in VRML nodes, these housekeeping properties keep track of simulation commands and data that can be used to map the model algorithms to the physical properties.

In addition to managing data and model relationships, housekeeping properties are used to store descriptions of multi-user interactions and pointers to static physical properties for interactions. These interactions are different from the model and physical property interactions described above. Multi-user interactions describe behavior effects between objects that are actually engaging in simulated virtual behavior. One of the fundamental multi-user interactions between objects in a virtual environment is information communication. Housekeeping properties are used to track communications text and dynamic data communications within the virtual environment. One function of the Housekeeping properties is to track the simulation state of speech simulations and to track spoken simulation command input. By keeping track of simulation LOD's in the housekeeping properties, the correct set of static physical properties can be extracted by a generic virtual getModel method. By combining low level state information with high level interaction and model LOD state information a virtual environment simulation can truly become more than the sum of its parts.

5 SimVRML Editor Interface

The SimVRML editor was created using 3D Website Builder [14] and Cosmo Worlds [15]. Although there is still no integrated development environment for VRML behaviors that is affordable, acceptable components are now becoming available. 3D Website Builder is used to create the geometry of the SimVRML editor interface. Cosmo Worlds is used for polygon reduction, VR/VE/VRML sensor creation, VRML routing and scripting within the script node. VRML scripting was done in Javascript [16, 17] and Java [18, 19].

The SimVRML editor interface is a compromise between the goals of a 3D anthropomorphic VR/VE/VRML interface and the cardinal goals of webcentric virtual reality: speed, speed and more speed. A fully anthropomorphic virtual reality behavior editor interface could be realistically composed of 3D subcomponents constructed using thousands of polygons. For example, the average power property used by the sensor model algorithms could have a realistic model of a generator unit with clickable parts in an anthropomorphic version of the editor interface.

The design paradigm applied to the SimVRML editor interface reduces the anthropomorphic aspect of the interface to texture maps. Photo images or rendered images of 3D models representing the various simulation model properties are applied

to 3D palettes constructed using simple solid primitives. For properties representing high level hierarchical subcomponents of an object, such as equipment, this texture map strategy works very well and is illustrated in Figure 1.



Fig. 1. Equipment Palette SimVRML Interface for Hierarchical Object

For more esoteric and abstract properties with no isomorphism's between anthropomorphism and algorithmic functionality, a different reduction strategy was developed. In the case of abstract properties, an anthropomorphic texture map or maps, representing an aggregate of related properties was generated. Hot spots representing the individual properties of the aggregate texture map are used by the editor to edit property values. Consider the basic radar equation. Sets of texture maps of generic radar properties are placed near to a texture map of a text version of the generic radar equation. The equation texture map is subdivided and overlain upon 3D primitives. Subdivision is necessary to avoid browser environments that do not support the VRML image texture or material nodes transparency features. Since the Virtual Reality Modeling Language is extensible, extending the SimVRML editor interface by users who have added extra model properties is a simple task. Extending the SimVRML editor interface is primarily a case of image processing and 3D geometry building

User input is through a series of knobs and sliders. This input strategy is made necessary due to the lack of a keyboard sensor in the VRML specification. The 3D Input interface for both sliders and knobs is similar. The interface consists of a VRML model of a slider or knob constructed from primitives and indexed face sets, along

with a 3D indicator and a 3D control. 3D indicators are used to display the current value of the property. The user modifies the property values using the movable portions of the slider or knob by clicking or dragging. The user can also use the 3D control to edit property values by clicking on subcomponents of the control, such as arrows. The radar equation texture map palette is illustrated in Figure 2.

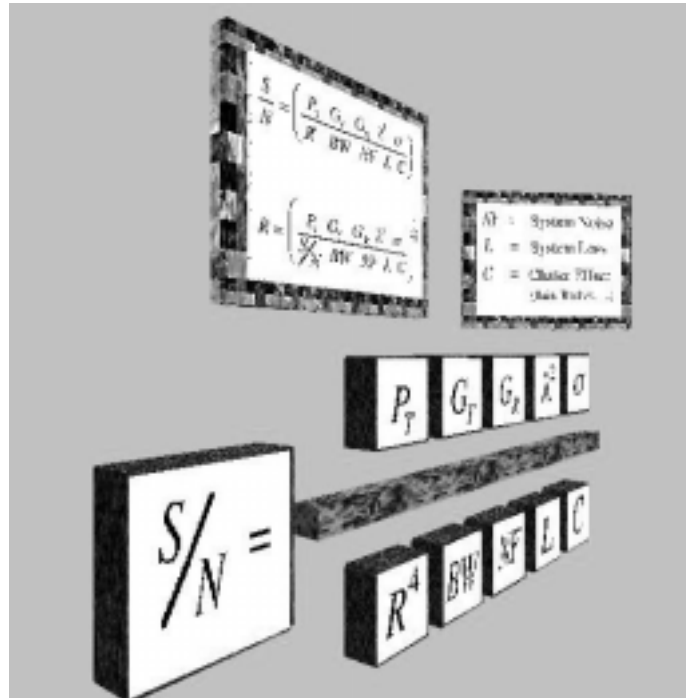


Fig. 2. Aggregate Texture Maps for Radar Parameters in SimVRML Editor

Sliders, knobs, controls and indicators have their own standard set of “SimVRML editor properties”. These interface properties are properties for minimum / maximum input values, data type of values, current value, type / size / color of 3D text for display of the current values, scaling factors and position / orientation relative to the slider / knob interface components. Just as the simulation models require a set of standard properties, there is a need for some sort of standard set of input properties for virtual environments. The geometry of the input widgets is up to the input interface designer. The SimVRML editor interface implements what I consider a minimal set of input properties. Figure 3 illustrates scalar data types. Arrays and other structures could be displayed by duplicating and offsetting VRML group nodes representing the various types of scalar data types displayed via geometry primitives and VRML text.

The SimVRML editor has an option for 2D operation. This option is enabled from within the 3D virtual environment and is based upon Java applets. The 2D option has a SimVRML applet for each of the various object classes. The 3D user input virtual environment is also illustrated in Figure 3

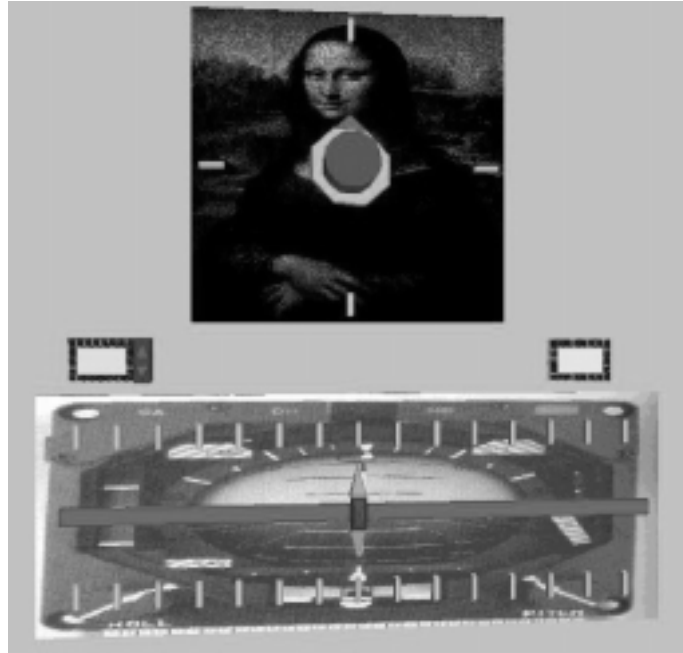


Fig. 3. Slider and Knob Input Interface With Control (arrows) and Indicator

One of the most important considerations related to editing properties for VR simulations is the realism of the data values. The majority of properties in the subclasses of the SimVRML base class have well known published reference sources. Commercial instances of the various abstract objects are particularly well documented. Military and adventure instances of simulation objects are not well documented. Potential sources for such adventure data are user guides for various game simulators, documents published by Jane's Information Group and manufacturers advertising brochures, repair manuals and user's manuals. Many physical properties can be obtained through standard scientific reference works such as those published by the Chemical Rubber Company. Fascinating and generally correct sources of physical properties are newsletters and publications of hobby organizations such as the American Radio Relay League.

6 Future Work

The SimVRML editor is used to edit static characteristics for physical properties used by VRML behavior models. The next obvious extension is to create a SimVRML layout editor to operate on the variables and methods of the dynamic properties. As an example, consider a Yacht with no engine, antennas, communications or navigation equipment. Using such a layout editor, the user could drag and drop engines, radio and other equipment onto the VRML yacht. The equipment would be from palettes containing thumbnails created by querying the VRML simulation behavior database.

The Model algorithms and the methods for extracting data need for the algorithms have to be implemented through a SimVRML behavior editor. There is also a need for a SimVRML command editor to dynamically input parameters into the VR behaviors. One obvious enhancement to the various model algorithms is to use the fact that a VR/VE/VRML simulation is intrinsically in 3 dimensions. The current damage model simply decides on the fate of a particular VR object. An enhanced damage model could also change the geometry of the object using the createVRMLFromURL method or by inserting and deleting nodes. Physically relevant 3-D dials and indicators would replace two-dimensional textual status boards. In addition a set of OMT diagrams for the baseline SimVRML editor and related models have to be generated.

References

1. Wicks, J., Burgess, P.: Standard for Distributed Interactive Simulation – Application Protocols. IST-CR-95-06, Institute for Simulation and Training (1995)
2. Dodsworth, C.: Digital Illusions. Addison-Wesley (1997)
3. Skolnik, M.: Radar Handbook (2nd Ed.). McGraw-Hill Publishing Company (1990)
4. Interference Notebook. RADC-TR-66-1, Rome Air Development Center (1966)
5. Wehner, D.: High-Resolution Radar (2nd Ed.). Artech House (1995)
6. Horton, J.W.: Fundamentals of Sonar (2nd Ed.). United States Naval Institute (1959)
7. Holst, G.: Electro-Optical Imaging System Performance. JCD Publishing (1995)
8. Kleinrock, L.: Queuing Systems (Volume 1). John Wiley & Sons (1975)
9. Przemieniecki, J.S.: Introduction to Mathematical Methods in Defense Analyses. American Institute of Aeronautics and Astronautics, Inc. (1990)
10. VRML 97 ISO/IEC 14772-1:1997
11. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-Oriented Modeling and Design. Prentice-Hall (1991)
12. Bowman, V.: Paradigm Plus Methods Manual. Platinum Technology (1996)
13. Shlaer, S., Lang, N.: Shlaer-Mellor Method: The OOA96 Report. Project Technology, Inc. (1996)
14. Scott, A.: Virtus 3-D Website Builder Users Guide. Virtus Corporation (1996)
15. Hartman, J., Vogt, W.: Cosmo Worlds 2.0 Users Guide, SGI Inc. (1998)
16. Bell, G., Carey, R.: The Annotated VRML 2.0 Reference Manual. Addison-Wesley (1997)
17. Ames, A., Nadeau, D., Moreland, J.: VRML 2.0 Sourcebook (2nd Ed.). John Wiley & Sons, Inc. (1997)
18. Lea, R., Matsuda, K., Miyashita, K.: Java for 3D and VRML Worlds. New Riders Publishing (1996)
19. Brown, G., Couch, J., Reed-Ballreich, C., Roehl, B., Rohaly, T.: Late Night VRML 2.0 with Java. Ziff-Davis Press (1997)