Web-Based Mobile Robot Simulator

Dan Stormont

Utah State University
9590 Old Main Hill
Logan UT 84322-9590
stormont@hass.usu.edu

Many roboticists rely on simulation during the early phases of developing navigation algorithms for their autonomous mobile robots. While many commercial robots now come with robust development environments that include visual simulators, these tools aren't available to robotics researchers who are working with a custom-built robot or who have not yet determined which commercial robot will satisfy their requirements. Even researchers using one of the commercial development environments will have difficulty sharing their simulation results with colleagues or others who do not have access to the commercial development tools. Believing that the ability to share simulation results visually with the greatest number of people would be an important capability to have led to the development of the web-based simulation approach described in this paper.

First Generation

The "first generation" of this web-based simulation began with a text file full of waypoints. The text file was being generated by a probability grid-based navigation scheme being investigated for implementation on the mobile robot LOBOtomous at the University of New Mexico. Realizing that this cryptic output file would be incomprehensible to most viewers motivated the initial experimentation with a web-based simulation.

The most important consideration in this initial experiment was making the simulation results accessible to the largest possible audience. This motivated the decision to use the World Wide Web to distribute the visual results and the search for a format that would be platform (and web browser) independent. A number of different visualization tools were evaluated, including Chrome, Live 3D, QuickTime VR, Real-Time Authoring for Virtual Environments (RAVE), and the Virtual Reality Modeling Language (VRML). Of these, only VRML was an open standard that was supported on a large number of platforms with a number of freely distributed VRML browsers and plug-ins. Also, with the release of VRML 2.0, a number of the shortcomings in VRML that had prevented it from being used for anything more than static visualization of three dimensional objects were addressed. It was now possible to have an animated object move through a virtual environment, while providing the user with a nearly limitless number of viewpoints. Additional flexibility was provided by the ability to run active elements (like Java applets) from the VRML browser.

Having selected the tool to be used, the next step was to build a VRML model of the robot to be simulated, in this case, the robot LOBOtomous. LOBOtomous is a custom built robot with two driving wheels and two castors centered around a central pivot point. The wider base contains the motors and drive circuitry, while the narrower cylindrical body contains the PC-104 control computer, the ultrasonic sensors and their circuitry, in addition to any circuitry added to increase LOBOtomous' capabilities. LOBOtomous is shown in Figure 1.

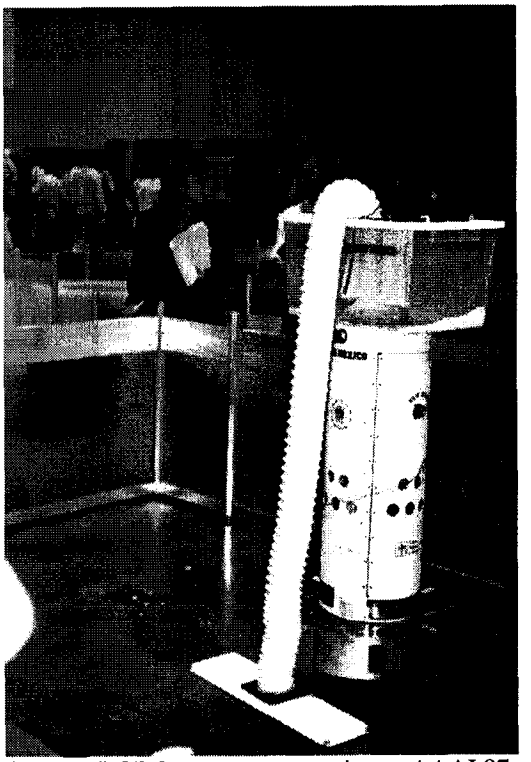

Figure 1. LOBOtomous vacuuming at AAAI 97.

The VRML model was built to have minimal detail while still being recognizable as LOBOtomous. A screen shot of the LOBOtomous model is shown in Figure 2.

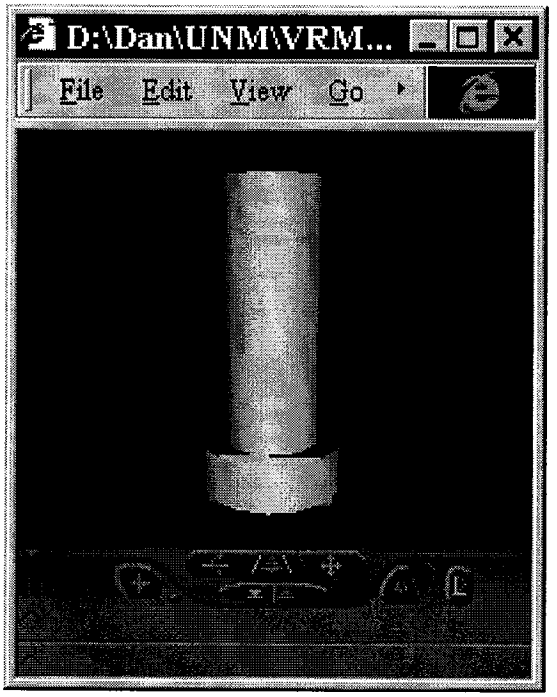

Figure 2. VRML model of the robot LOBOtomous

Figure 3 shows LOBOtomous in a simulated lab environment. The lab environment is a very simplistic version of the robotics lab at UNM, since the furniture is not detailed (the lab benches and other obstacles are nothing more than solid blocks in the model), but the model does allow the visualization of the robot's motion through this environment from starting point to goal.

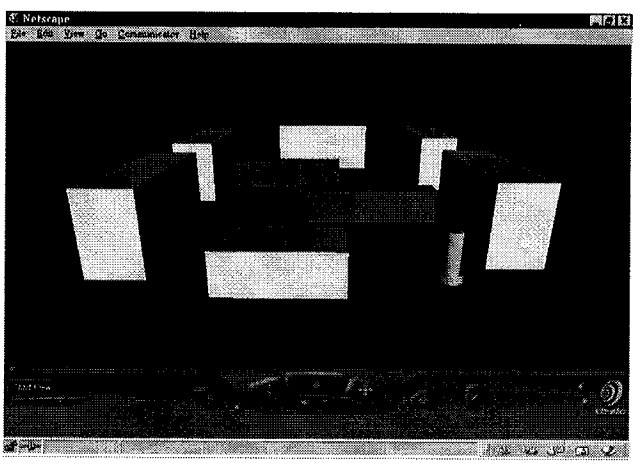

Figure 3. VRML model of lab.

The drawback to the first implementation is that it is not very flexible. The animation of the robot's path through the lab was hand coded in VRML using the text output of a simulated run of the robot through the lab. Changes in the algorithm, environment, or even start and goal positions requires recoding the animation. This lack of flexibility motivated the next generation of the VRML simulation.

Next Generation

The lack of flexibility in the first version of the VRML simulation influenced the design of the several iterations in the next generation of the simulator. I wanted the next iteration to be more detailed and more flexible than the first. This section will describe the steps in designing this next generation of the VRML Mobile Robot Simulator.

The first step in creating a simulation is to build (or, once a sufficiently large library has been built up, select) a model of the robot whose motion will be simulated. For this example, I will illustrate coding a small robot called Little Blue (a take-off on the name of the Utah State mascot, Big Blue). Little Blue is a four wheeled robot built from a remote-control Red Fox toy car. Little Blue has two motors for differential steering and has enough room for a BASIC Stamp or PIC controller and some small sensors for obstacle avoidance and goal location. Figure 4 shows a Red Fox car prior to being modified.

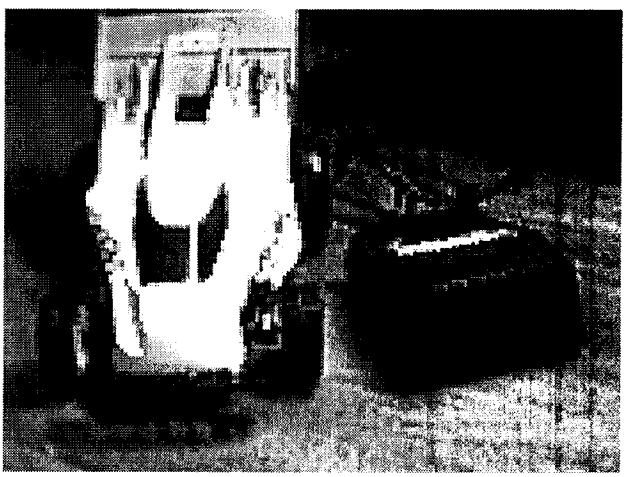

Figure 4. Red Fox remote-control toy car.

In this example, the VRML model is of a Little Blue robot configured for the Trinity College Fire Fighting Home Robot Contest. For those not familiar with this contest, it is an annual contest held at Trinity College in Hartford, Connecticut. The purpose of the contest is to have a robot navigate a maze that is meant to simulate a house, locate, and then put out a fire in the house. The fire is represented by a candle approximately six inches in height. The contest rewards robots that use sensors instead of just dead-reckoning by awarding bonus points for various active navigation schemes, like map building. The emphasis is on speed in locating the fire and the ability to put it out. (More information about this annual contest can

be found at http://www.trincoll.edu/events/robot/.) Thus, a robot that can succeed in this contest needs sensors for determining its position in the maze, a sensor for locating the fire, and some mechanism for extinguishing the flame. This is why the Little Blue robot depicted in the VRML model has infrared emitter/detector pairs on the front for obstacle avoidance (not shown) and a phototransistor for detecting a flame. A tube on top of the robot directs a spray of compressed air at the flame to put it out. The VRML model of this version of Little Blue is shown in Figure 5.

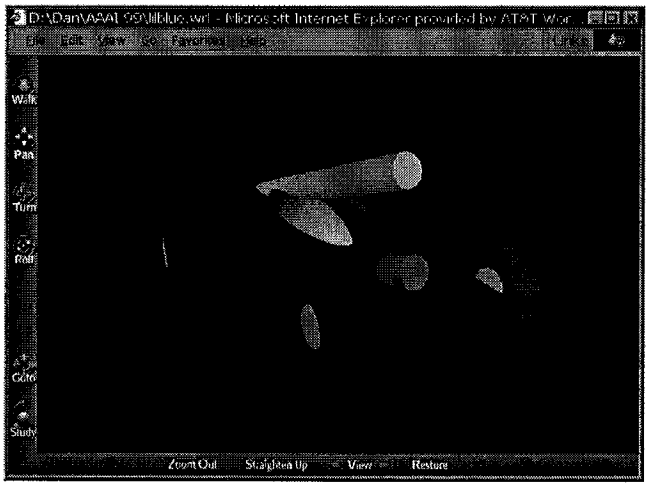

Figure 5. VRML model of Little Blue fire fighting robot.

With the VRML model of Little Blue created, a VRML world for it to navigate through needs to be created. (VRML environments are usually referred to as worlds, even the standard file extension for VRML is .wrl for world.) Obviously, since the Little Blue model is designed for the Trinity College Fire Fighting Home Robot Contest, the world created for it should be a version of a maze used for the contest. Figure 6 shows an overhead view of the maze created for Little Blue to operate in. Note that the maze is not designed exactly to the dimensions of any particular contest layout, rather it is representative of a typical maze. While a properly proportioned maze could easily be created in VRML, I felt it better for this example to stick with a representative rather than an accurate maze.

There are a couple of important features of the maze that should be noted. The most obvious thing is that the maze doesn't have a roof, since it is possible to see inside the maze. This is how the real maze is to allow access to the robots, but a real house would have a roof. VRML allows the creation of a roof and can limit access to the interior to only those entrances that would allow entry in the real world, such as open doors or open windows. Thus, VRML could be used to simulate a typical interior environment such as a home or an office environment, including any reference points or features that would normally be found on a ceiling, such as light fixtures and air conditioning vents. A realistic simulation of an office environment for a robot that uses visual clues on the ceiling (e.g., counting the number of light fixtures passed) can be created in VRML.

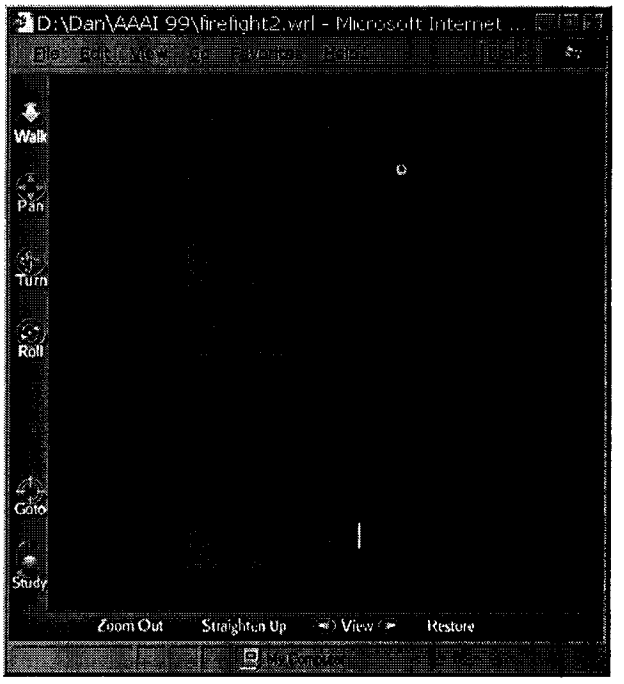

Figure 6. Overhead view of a Trinity contest maze.

Some other features of the maze that should be apparent are the start position (the red S in a circle that Little Blue is sitting on) and the goal, which in this case is the lighted candle in the room in the upper right hand corner. Figures 7 and 8 show different views of the start and goal positions.

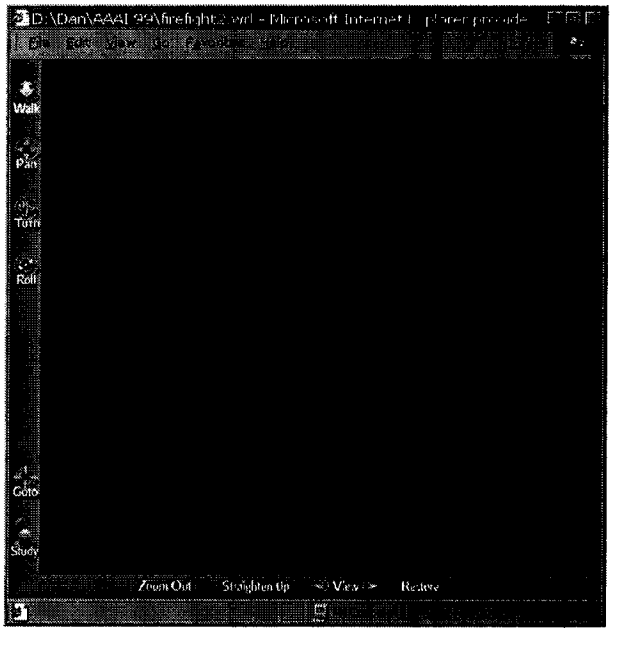

Figure 7. At the start position, looking down the hall.

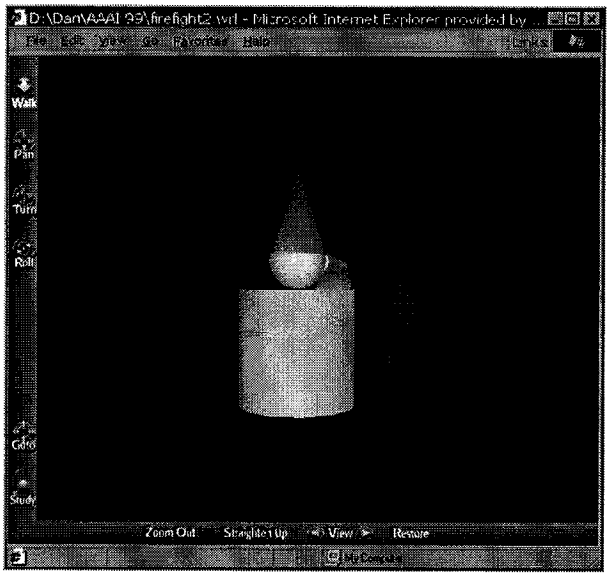

Figure 8. Little Blue at the goal position.

Figures 7 and 8 illustrate one of the nicer capabilities of VRML: the ability to define camera positions in the VRML world. The overhead, start, and goal views are all established by creating viewpoints. There are also viewpoints (not shown here) looking in to each of the rooms in the maze. The viewpoints can be selected by clicking on the desired viewpoint in the VRML browser, making it possible to quickly change viewpoints while the animation is running. The strength of this ability is that it allows the creator of the VRML world to create viewpoints that emphasize locations or areas of interest for the viewer.

Having discussed the capabilities of VRML, let's look at some VRML script. Figure 9 shows the VRML script for the candle in the maze. Starting at the top, the comment #VRML 2.0 utf8 is a mandatory line that tells the VRML browser which version of VRML and which character set to use (utf8 is the Universal Character Set Transform Format-8, of which the ASCII character set is a subset). After that you will see the transform for drawing the white cylinder that makes up the base of the candle, the transform for drawing the yellow sphere that is the base of the flame, followed by the transform for the orange cone that is the top of the flame. The reason these are all transforms is that the default in VRML is to draw an object at the center (0.0 0.0 0.0) of the coordinate system. In order to stack the geometric figures to make a candle and to ensure the candle is sitting on the floor, it is necessary to translate the shapes in the coordinate system to get them in the right location in the VRML world. Determining the correct translation to use is one of the most difficult aspects of working with VRML. I could also have grouped the elements of the candle together, which would allow moving the candle as an entity instead of translating each element separately. If multiple candles were needed in a VRML world, this would be the approach to take, as duplicate candles could be created and placed in the world.

```
#VRML V2.0 utf8
# Draw the goal (a candle)

Transform {
  translation 12.5 -5.0 -30.0
  children [
    Shape {
      appearance Appearance {
        material Material {}
      }
      geometry Cylinder {
        radius 1.0
        height 2.0
      }
    }
  ]
}

Transform {
  translation 12.5 -3.5 -30.0
  children [
    Shape {
      appearance DEF Yellow
      Appearance {
        material Material {
          diffuseColor 1.0 1.0 0.0
        }
      }
      geometry Sphere {
        radius 0.5
      }
    }
  ]
}

Transform {
  translation 12.5 -2.65 -30.0
  children [
    Shape {
      appearance DEF Orange
      Appearance {
        material Material {
          diffuseColor 1.0 0.5 0.0
        }
      }
      geometry Cone {
        bottomRadius 0.5
        height 1.5
      }
    }
  ]
}
```

Figure 9. VRML script for the candle.

Finally, having built a VRML model of a robot and a VRML world for it to operate in, the last step is to animate the motion of the robot in the maze. IN VRML, an animation is started by clicking on the object that is being animated. The object will continue to move (in the case of an infinite animation, like a rotating sign) or it will reach a stopping point. If an animation has a stopping point, clicking on the object again will send it back to the start and execute the animation again. While the ideal solution would be to run the navigation in real time, for this iteration of the robot simulator I had not solved the problem of running a Java applet from within the VRML environment. However, the animation was more sophisticated than the first generation simulation. In the UNM lab simulation, LOBOtomous never rotated when making a turn – it just changed positions. This caused the somewhat humorous behavior of the LOBOtomous model sliding sideways for some segments of its path. For the Trinity maze, rotations have been included in the motion of Little Blue so it will turn to enter a room. The rotation is still not perfect – Little Blue pivots around its geometric center even though this would be impossible for a robot utilizing Ackerman steering, like Little Blue. This inaccuracy in the displayed motion is one of the areas that needs improvement in future iterations of the robot simulator.

VRML Strengths and Weaknesses

VRML's flexibility can be a tremendous asset in creating a truly flexible robot simulation environment. One example is in creating the walls of the VRML world. One way to do this is to create each wall and each obstacle as a cube. This was the approach taken for the UNM lab environment described in the first generation section. The problem with this approach is that it is time consuming to create and not easy to modify. A better alternative from a flexibility standpoint is using the VRML IndexedFaceSet, which will draw faces from one coordinate to another. This allows a world to be described by a list of coordinates, which could be imported from another source. This capability of VRML lends itself to creating a menu of floor layouts or generation of layouts by another program (like a Java applet).

Another VRML capability that adds to its flexibility is animation, although there are some difficulties working with VRML animations that must be overcome. Animation in VRML uses the concept of a clock that determines when an interpolator will fire. There are three types of interpolators: position, orientation, and scale. The position and orientation interpolators will be the ones most frequently used for robotics applications. Figure 10 illustrates the use of the position interpolator as Little Blue moves from one point to another in the hallway. Figure 11 illustrates the use of the orientation interpolator as Little Blue turns to enter a room.

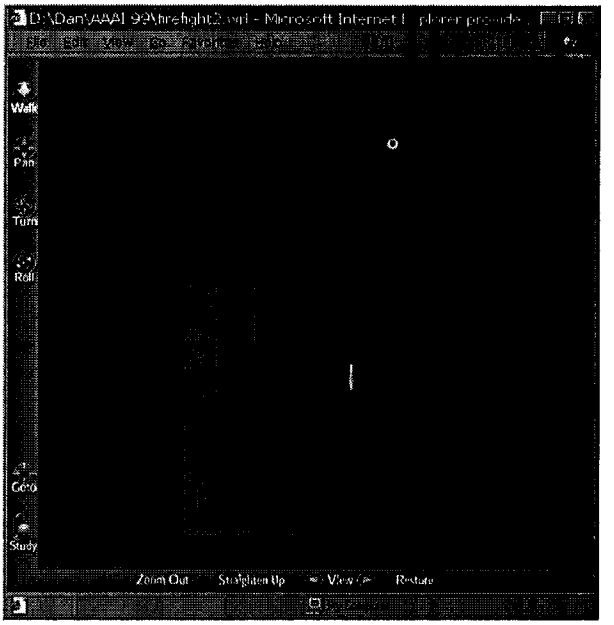

Figure 10. Little Blue moving down the hall.

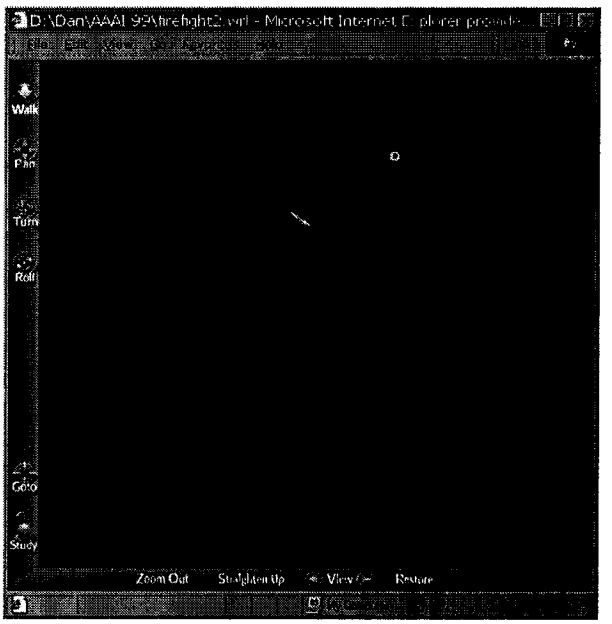

Figure 11. Little Blue turns to enter a room.

The biggest problem is that the position and orientation interpolators are separate in VRML, so the changes in the interpolators must be synchronized in order for the desired motions to occur in the right order. Essentially, this means each entry in one interpolator must have a corresponding entry in the other interpolator, even if there is no change in the value of the other interpolator. This isn't really as big a problem as it sounds, since many robotics libraries use separate position and orientation commands, so the VRML mechanism is actually very similar. The animation interpolators appear to offer promise for on-line execution

of navigation algorithms using Java applets and the VRML Script node, which allows for execution of external programs from within the VRML world.

The script node in VRML is the capability with the greatest promise for creating a general purpose web-based mobile robot simulator, but it is also the capability that has proven to be the most difficult to get working. The biggest problem is trying to get the applet output into the format required by VRML. Another problem is the issue of control – should the Java applet be embedded in the VRML file as a script node or should the Java applet be run outside of the VRML world and display the VRML world when appropriate? I haven't determined the answer to this question yet, but the answer is key to creating the type of robot simulator I envision.

What Next?

There are a number of capabilities of VRML that I have not yet exploited in building web-based mobile robot simulations. I intend to continue extending the capabilities of these simulation tools as a part of my research into robot swarms for planetary exploration at Utah State University. This mobile robot simulator will play a key role in the simulation phase of my current research

Specifically, some of the potential enhancements being worked on now will provide the user of the simulation with the following capabilities:

- Use the mouse to select the start and goal locations;
- Navigate through a more realistic environment, with obstacles that are recognizable (not just geometric shapes) and possibly varying sensor effects based on the obstacle shape and material;
- Select the navigation algorithm to be used from a list of Java applets;
- Select the type of robot simulated, with the appropriate motion model for the drive configuration; and
- Select or create the environment to be navigated through while on-line.

As enhancements become available, they will be posted on the Utah State University web server at http:// www.usu.edu/~afrotc/cadre/HTML/stormont/vrml.html. Any of the simulations may be downloaded, used, and modified freely. I would also strongly recommend the VRML 2.0 Sourcebook by Ames, Nadeau, and Moreland as a good reference book when working with VRML. This book was published by John Wiley and Sons in 1997.