The Out Of Box Experience: Lessons Learned Creating Compelling VRML 2.0 Content

Sam Chen, Rob Myers, Rick Pasetto
Silicon Graphics, Inc.

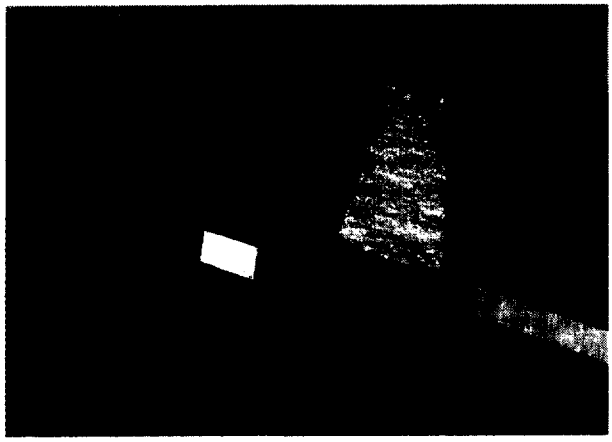

Figure 1. The Out Of Box Experience *Courtyard*.

Abstract

VRML 2.0 offers a new medium for producing compelling interactive content. The best results, however, require extensive attention to detail, and a thorough understanding of the constraints of the delivery mechanism. Creation of a large-scale, consumer-grade hypermedia environment served as our motivation to discover what this new medium can really deliver. In this paper, we present the challenges and pitfalls faced, decisions made, special techniques used, and the lessons learned during a professional VRML production pipeline. Both world builders and tool makers can benefit from the technical and creative issues identified here. We cover the entire production process: storyboarding, scene composition, modeling, illumination, navigation issues, embedded user interface, and effective use of sound and animation. Our conclusions show that performance and interactivity must be the number one consideration throughout this process.

CR Categories and Subject Descriptors: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Virtual Reality; I.3.6 [Computer Graphics]: Methodology and Techniques - Interaction techniques.

Additional Keywords: virtual worlds, virtual environments, navigation techniques, three-dimensional user interface, VRML

2011 N. Shoreline Blvd., Mountain View, CA 94043
sambo@sgi.com

VRML 97, Monterey CA USA

1 INTRODUCTION

Since its creation in October '94, VRML (Virtual Reality Modeling Language) [2] remained a niche technology with grass roots origins and little widespread appeal. Lacking experience and high-level authoring tools, world builders produced worlds that were static and simplistic at best. With adoption of the VRML 2.0 Moving Worlds Specification [1], a richer range of possibilities now awaits the exploration of world builders and tool makers. However, most of the attention so far has been paid to the technology itself; there has been little focus on the creative application of VRML.

In October '96, Silicon Graphics launched the O2, its new desktop multimedia workstation. An engaging Out Of Box Experience was incorporated into the system, to introduce each user to the capabilities of their new machine. To deliver a media-rich presentation central to O2's packaging, documentation, demonstration, and user interface, the Out Of Box Experience (or "OOBE") used HTML, MPEG, other standard digital media formats, and JavaScript with VRML 2.0, with the latter being the integration cornerstone (Fig. 1). This was an opportunity to demonstrate 3D as a viable user interface technology, and was the first application of VRML in such a large scale before a mainstream commercial audience.

The Out Of Box Experience was tuned specifically for performance using CosmoPlayer 1.0 on the O2 workstation. VRML 2.0 content was authored using CosmoWorlds 1.0 on Silicon Graphics workstations. While we believe all of the considerations presented below are of cross-platform interest, some of the issues detailed are specific to these configurations.

2 STORYBOARDING AND DESIGN

The OOBE storyboarding and design process started off like a typical multimedia production project. The design team, a collaboration between Construct Internet Design and Silicon Graphics, included an Artistic Director, a Producer, a Technical Producer, Writers, and VRML Architects. We launched a very informal process, where almost any idea was fair game. This quickly changed, as we realized what our constraints were and how difficult a complete implementation of our ambitions would be. The technical and practical implications of creating a large-scale VRML 2.0 experience had yet to be revealed. At this point, the VRML 2.0 Moving Worlds draft specification was still undergoing major changes every week, and VRML 2.0 authoring tools were practically nonexistent. After deciding to defer on advanced extensions for multi-user environments, chat rooms, avatars, and the like, we narrowed our scope down to exploiting the basic features of Moving Worlds and applying them effectively.

2.1 Designing for Responsiveness

Our biggest challenge was to design a large-scale world that both tells a meaningful story and maintains responsive user interactivity. Optimization should not be considered as a separate stage of the production; it must be a continuous process throughout the entirety of the project pipeline. Constant mindfulness about how to

achieve and sustain interactive graphics performance is a fundamental part of VRML world building.

In our quest for a highly interactive OOBE, we looked to the game industry for ideas and tricks. We learned to use simple geometry with texture maps in place of complex geometry. We studied the way CD-ROM games manage their asset complexity versus load time. We examined how to get the most richness with very little resources. These techniques have direct applications to VRML.

With these system constraints in mind, we storyboarded our worlds by defining an appropriate treatment for each functional domain. Design inspirations were drawn from theme parks, museums, and art galleries. These studies were refined into a core set of linked spaces: the *Entryspace*, *Courtyard*, *Innovations Gallery*, *System Tour*, and *Jungle Island*. To manage scene complexity and load time, each of these functional areas was encapsulated within its own separate VRML file. With this design, the user moves from world to world through hyperlinks that use the VRML Anchor mechanism.

2.2 Designing for Serendipity

Inspired by the free flow of ideas and relationships that characterized our early design process, we felt it was important to allow for serendipity in the user's browsing process as well. Whether it's giving players the freedom to create personalized rhythm combinations in the 3D drum machine *CyberGourds*, or the ability to tweak their own illumination effects in the lighting demo Boink, the user participates in the synthesis of his or her own experience to the degree that he or she invests in it. In other words, the user is rewarded for having an adventurous spirit.

Part of the overhead in creating for VRML is the necessity for "designing in the round." While traditional 2D media need only be rendered from one best view, the 3D designer must consider how his or her content looks from all angles. Thus, the user is rewarded for his or her exploration of the space.

2.3 Knowing Your Audience

It is crucial to know who your audience is and, if possible, on what platforms they'll be viewing your content. OOBE targets first-time and novice 3D users. Their experience needed to be friendly, easy to navigate, and not too overwhelming. The potential pitfall was letting the architectural complexity overrule the functionality of the space. The moment the user feels lost or out of control, the design has failed.

Knowing that our content is to be viewed on the O2 platform, at least initially, was a luxury. This granted us significant freedom to push our scene complexity to the limit within well-recognized boundaries. Without such a well-defined delivery goal up front, world builders must still decide on the minimum target platform required to meet their audience and keep the design strictly within those limitations.

3 MODELING AND GEOMETRY

A single incontestable lesson can be clearly engraved: at all stages, the design ambition must be driven by the basics of the performance budget. Balancing artistic license and high performance posed the greatest challenge to us, both technically and creatively.

3.1 The Complexity Budget

When planning the actual geometry, we first set a budget for how many polygons that we could have in the scene at a time. We wanted a frame rate of at least 10 frames per second on the O2, which worked out to around 4000 triangles (in any particular view, not total). As we progressed, we realized how this constraint really taxed our technical and creative skills; we knew that every triangle counted, and we made sure that no geometry was wasted.

3.2 Modeling and Conversion

Faced with the lack of a native VRML 2.0-based modeler at the start of our project, we had to rely on existing 3D modelers and file translators to start creating our worlds. The modelers used in the creation of OOBE included Nichiman N-Geometry, Radiance Ez3d, Alias 3Design, WebSpace Author 1.0 (VRML 1.0), and CosmoWorlds 1.0 (VRML 2.0). Using existing modelers turned out to be both a blessing and a curse. As is typically the case in 3D production environments, we found that each modeler had its own strong points. However, since some of these modelers were designed for realism, they would create very high polygon count models, which we would then have to reduce by hand to fit within our low polygon budget. Additionally, since each modeler had its own file format, we had to rely on the file converters to do their job right. Unfortunately, sometimes a converter would introduce unwanted redundancy, expand geometry, or just create illegal or bad VRML.

3.3 Structuring Models to Improve Performance

There are things to watch out for during the modeling phase other than pure polygon count. For OOBE, we had to consider the way the VRML 2.0 scene graph was constructed and its depth, the size of textures and the number of times they were used, the spatial relationship between objects, as well as other seemingly trivial but actually crucial factors.

In general, any work you can do up front in the organization of your scene graph can help rendering performance. For instance, with VRML 2.0's changes to the original VRML 1.0's attribute inheritance model, we noticed that scenes containing a lot of IndexedFaceSets with the same Appearance were getting slower performance on our target browser. We found that stuffing these geometries into a single IndexedFaceSet, with a single Appearance node, sped things up dramatically. We used CosmoWorlds' "PEP merging" tool in order to do this.

Another performance key when using IndexedFaceSets is to employ "backface culling" whenever possible. Appropriately applied to a model that forms a closed, solid hull, backface culling can effectively cut its rendering triangle count in half.

3.4 Managing Scene Complexity

The distribution of the complexity budget can be orchestrated directly, in response to the user's location, using Level Of Detail (LOD) nodes and ProximitySensors. We made sure that LOD switching was very gradual, to avoid disturbing "pops" as the user approached the object. This meant that there needed to be a lot of room between objects. For this reason, LODs were used only in the wide open spaces of the *Courtyard* and for the planets in *CyberAstronomy* and not in the spaces with more intimate settings.

ProximitySensors can also help in geometry management, where you can use them to "turn off" geometry when the viewer is not

near it. For instance, you can turn off all of the geometry inside a building when the viewer is outside it. An LOD would not work in this case because you may be close to the geometry but on the other side of a wall.

3.5 Quick Tips for Modeling High-Performance Geometry

- Determine your polygon budget and stick to it!
- Beware of spline-based modelers; they can produce high polygon count models.
- Don't rely on file conversion to give you well-formed VRML 2.0.
- Try using Textures to stand in for more complex geometry.
- Merge static geometry sharing common appearance attributes for faster rendering.
- Use backface culling whenever appropriate (IndexedFaceSet.solid = TRUE).
- Use LODs, ProximitySensors, and other VRML 2.0 constructs to manage scene complexity.

Other performance tuning tips:

- Remove Cylinder ends and Cone bottoms when they aren't visible.
- Use the Background node instead of constructing your own background geometry.
- Employ transparent objects sparingly.

4 LAYOUT

Layout is the process of scene composition, where objects are put into their proper place in the 3D scene. For OOBE, we used Cosmo Worlds 1.0 to compose our scenes.

4.1 Cull Volumes

The biggest performance pitfall during the layout stage is the improper spatial organization of the scene. Most browsers rely on render culling (performed at the Group and Transform nodes) to maximize rendering performance. Casually grouping two spatially separated objects under the same Group or Transform node reduces the browser's ability to cull needless rendering. Both objects will always be subject to rendering traversal whenever their connecting bounding volume falls within the viewing frustum, resulting in wasted overhead. For instance, it would be wasteful to group a chair in one room with other furniture scattered throughout other rooms, unless this structure is needed to produce some meaningful effect. Heeding this consideration, we made sure to keep spatial clusters such as the Innovation Gallery and *Jungle Island* each within their own containing Group nodes. Using graphical authoring tools to visually inspect cull volume bounding-boxes and the scene graph containership hierarchy, we ensured that our browser would perform the most effective render culling.

There is another side to this concept of grooming compact cull volumes. By spacing top-level objects far away from each other, a world can be structured to let the browser readily cull away objects that aren't in view. This, of course, allows much more complexity to be dedicated to each localized cluster, with the knowledge that this complexity doesn't have to be shared simultaneously with other clusters around the world. This is a well-known trick in the Visual Simulation industry.

In a related consideration, a sprawling model (such as a continuously snaking wall, roadway, or the ground terrain itself) may be a good candidate for spatial subdivision. If the user most frequently experiences only a small part of an object's whole geometry, performance may benefit from more efficient culling of its unseen portions. To do so, divide one monolithic, sprawling IndexedFaceSet into a few spatially compact chunks. Now, each of these independent IndexedFaceSets can be trivially culled out. This remedy is a balancing act, however, since care must be taken not to introduce excessive overhead with a gratuitous number of intermediate objects (see "Structuring Models to Improve Performance").

4.2 Multiple Instantiation

Another task that is important during the layout stage is instancing. The VRML specification provides the DEF/USE mechanism for this purpose. In production, we found the benefit of instancing to be twofold. Because an instance reuses defined geometry, it is convenient at authoring time; changes made to the master node automatically update all of its clones. More importantly, instancing decreases file size, resulting in faster downloads and less memory usage at runtime. For nodes that require file system or net access (specifically nodes with the url field), instancing reduces download time significantly. In *Jungle Island* with its many instances of the same palm tree texture and straw hut inlines, instancing effectively alleviates the browser from having to fetch the same palm tree and hut multiple times.

4.3 Quick Tips for High-Performance Layout

- Avoid grouping spatially dispersed objects together, to achieve more effective culling.
- When possible, position top level objects far apart, to let render culling do its job.
- Spatially subdivide monolithic, sprawling objects, if typically viewed a little at a time.
- Use instancing, not copying, to reproduce common geometry throughout the scene.

5 NAVIGATION AND INTERFACE

In keeping with its role as packaging, demonstration, and documentation of its host machine, the Out Of Box Experience called for a media-rich, multifunctional user interface.

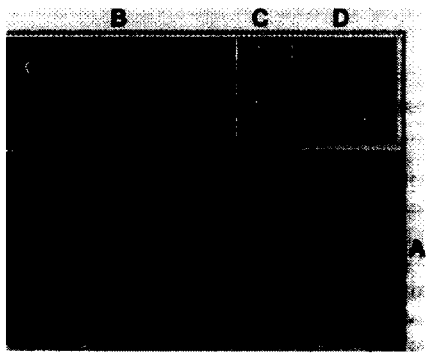

Figure 2. The general user-interface of OOBE: Window A contains primary VRML content that is the central focus of user's experience. Window B contains secondary HTML, MPEG, and VRML content that can be accessed through windows C and A. Window D is the 2D map used for navigation and orientation.

5.1 Full-Screen User Interface

Using Netscape Navigator and its frame technology, we were able to design an efficient and intuitive full-screen user interface for OOBE (Fig. 2). We divided the screen into 4 primary areas, with each frame hosting an appropriate plugin for the specific MIME types employed. The main window embeds the CosmoPlayer plug-in, presenting an ongoing, interactive, VRML-based experience on the center stage. The upper left frame alternates between HTML, JPEG images, MPEG movies, and secondary VRML insets, depending upon the supporting material requested. A strip of three small frames presents a context-sensitive table of contents, used to launch related assets. The upper-right frame remains dedicated to an overview "you are here" map, where users can click at any time to jump to other parts of the experience. We found that this scheme offered the flexibility and richness required to unify our wildly heterogeneous content, while allowing the user to remain focused on the immersive experience running center-screen.

It can be rather hard to get around in Cyberspace. We have witnessed many instances where users fumble with 3D trackballs and joysticks, quickly losing patience with not being able to get where they wish to go. It was apparent that OOBE needed an embedded navigational strategy transparent enough for 3D initiates, but which didn't limit more adventurous explorers from heading out on their own. We hosted a parallel suite of navigational paradigms to support a range of wayfinding styles.

5.2 Signpost Navigation

Most novices seem to prefer a closely-knit guided tour throughout the 3D experience, relying on signposts, maps, and navigational icons to keep them informed about where they are now, and what their next step should be. We devised a pair of navigational icons

```
PROTO WalkingMan [ ... ] {
  ...
    Anchor {
        ... geometry for back arrow ...
        url IS backURL
    }
    Anchor {
        ... geometry for to arrow ...
        url IS toURL
    }
  ...
}
DEF FOO WalkingMan {
    backURL "#ENTRANCE"
    toURL   "#EXIT"
}
```

Figure 3. (Left) "WalkingMan" navigation icon. (Right) "NavLite" navigation icon. Both are implemented using the PROTO and Anchor nodes.

to address this need: the distinctive "WalkingMan" and its abbreviated cousin, "NavLite" (Fig. 3). The navigational icons are quite conspicuous and easy to click on. Automated viewpoint animation is triggered when the user clicks on one of the arrows in the icon, bringing the user to the next (or previous) viewpoint in the tour sequence, as specified by the Anchors in the VRML file. (NOTE: OOBE relies on CosmoPlayer's built-in animate-to-viewpoint feature to deliver the user smoothly to their next destination. This feature may not be available in some browsers, resulting in a disorienting jump-cut. Rather than losing their users in this way, we heartily recommend that authors construct their own continuous viewpoint animations, if needed to conduct their guided tours.)

In the OOBE *Entryspace*, we specify the NONE viewer type, removing discretionary driving controls altogether, to rely solely on the tour icons for all navigation.

5.3 Named Viewpoint Navigation

Unlike the signpost navigation paradigm, which progresses step by step to each "must-see" locale, named viewpoints allow the user to see more of the world. This method was implemented by locating a named Viewpoint node at each of the selected "scenic vista" points throughout the world, which in most worlds in OOBE is a superset of the signpost locations. A user can visit any named viewpoint, at any time, as desired. For instance, the CosmoPlayer interface offers its users handy access to named viewpoints via its popup menu, the dashboard's viewpoint selection fixture, or PgUp/PgDn on the keyboard. The process of incorporating these viewpoints forced us to test our worlds for usability, encouraged us to seek out the prime "sweet spots," and ensured our users easy access to these photogenic locations.

5.4 Freeform Navigation

For the adventurous, OOBE offers the CosmoPlayer dashboard, with complete freedom to roam and explore through our worlds. With the addition of collision detection in Moving Worlds, roaming is now fun and most of all, intuitive. The dashboard supports WALK, EXAMINER, and FLY viewers, allowing the inquisitive user to toggle between these "vehicles" and enjoy the infinite range of perspectives and navigational possibilities that 3D offers. Plenty of "Easter Egg" surprises were authored throughout OOBE, to encourage and reward this type of adventurous spirit.

5.5 The Avatar's Perspective

Even after handing the dashboard's free-form controls over to their user, the author maintains a lot of responsibility for the quality of the trip. The NavigationInfo's avatarSize and speed fields exert significant influence on the user's navigational experience. We found that varying the avatarSize field by a few units produces a vastly different look for the same world, significantly affecting the apparent ease of navigation. In the OOBE *Courtyard*, varying the avatarSize value determined whether the river, the shoreline, and other important landmarks were visible or not. We believe that the ability to see more landmark features helps users orient themselves contextually and navigate cognitively. At the same time, the speed field must be selected carefully to work well with a given avatarSize. For instance, we found that as the camera moved closer to a texture-mapped terrain, its apparent speed increased. This worked to our advantage in *Jungle Island*, where we felt the amount of time it took to travel from one end of the island to the other was too short. We were able to prolong the experience by lowering the speed while specifying a relatively small avatarSize.

5.6 Architecture and Lighting as Navigational Aids

The spatial architecture of the scene and its lighting serve a traditional role as content by conveying story, place, and mood. However, in the brave new VRML world, architecture and lighting can also be applied as potent navigational aids (Fig. 4). OOBE's *Entryspace* establishes a golden spiral form that descends and expands to lead the traveler downward. Animating lights dim and brighten to refocus the user's attention on the next station along the

navigational path. These elements work together to produce a comfortable environment that pulls the user along its intended course. The lights are triggered by ProximitySensors, and animated using ScalarInterpolators attached to the light's intensity field. Animating lights are also used in the Industry Gallery for similar effect.

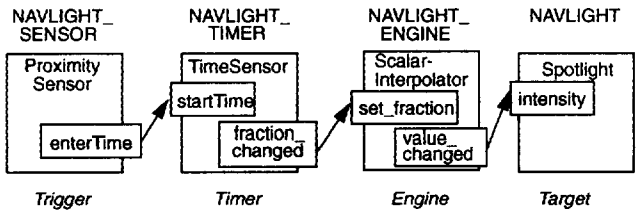

Figure 4. Using lighting as a navigational tool: routing diagram showing how this lighting is implemented.

5.7 3D Widgets

With the addition of interactivity and sensors in Moving Worlds, entirely self-contained 3D widgets can be implemented in VRML. The construction of an in-scene widget typically involves an originating trigger, often routed first to a conditional logic script, then to an animation engine of some sort, and finally to its targeted appearance or transform node field. In the Boink space, the lighting widgets are constructed in this way (Fig. 5).

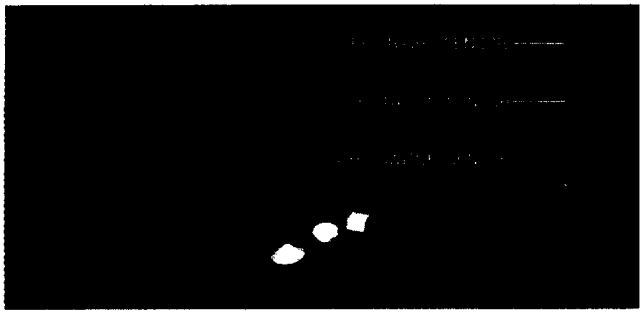

Figure 5. 3D tracklight widgets in Boink world. Each lighting fixture contains three embedded VRML sensors for its translation, rotation, and color selection.

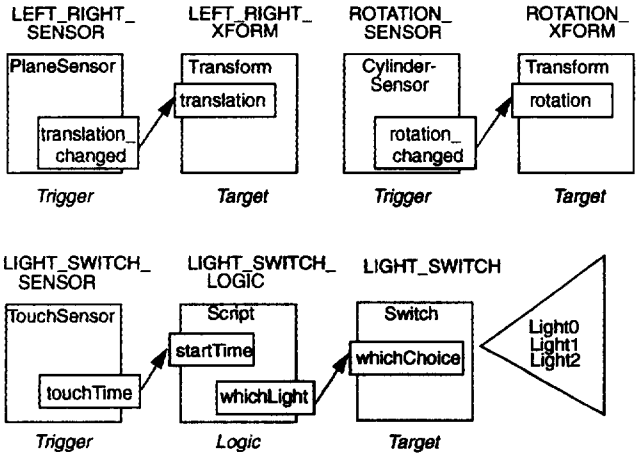

Figure 6:.Routing diagram showing event flow for the three types of lighting widgets: (Top Left) Translation; (Top Right) Rotation; (Bottom) Color-Selection.

We found that embedding the experimental lighting controls directly within the 3D playground was critical to the success of Boink. By allowing the user to remain focused within the performance setting, the user interaction stays in context without breaking metaphor. The result is much more engaging play, and a more memorable learning experience.

Similarly, the 3D texture mapping widget in *RaptorBuilder* was built to support in-scene manipulation. Using only TouchSensors and Scripts, the raptor texture map is wired to respond directly to mouse actions, the same way it would in a 2D texture map editor.

Figure 7. 3D texture map editor in *RaptorBuilder* world. These widgets perform scaling, translation, and selection of texture maps.

Breaking through the widget mentality, it can be exciting to establish the object itself as its own intrinsic user interface. In *CyberGourds*, the user controls what is heard by interacting with the gourds themselves. The head and stalk are clicked to stop and start a rhythm loop, while the baby buds are used to change the rhythm samples. Likewise, in *RaptorBuilder*, clicking on the individual bones and dragging on the various body parts triggers a specific behavior. We feel this approach is key to maintaining an immersive experience throughout the space, with the added benefit of extending the user interface metaphor into the third dimension.

```
PROTO ClickMe [ ... ] {
  ...
    Anchor {
          ... clickable geometry ...
          url IS url
          parameter IS parameter
    }
  ...
}

DEF FOO ClickMe {
    url         "http://foo.bar.com"
    parameter   "target=topFrame"
}
```

Figure 8. The ClickMe widget is implemented using a PROTO with an embedded Anchor node.

5.8 "ClickMe"s

In OOBE, it is not always obvious which objects are triggers for behaviors and animations. This was a conscious design decision, to encourage users to explore and click on anything they come across, to maintain the element of surprise and delight. However, certain elements were important enough to encourage even the most recalcitrant users to click them. We tagged these areas with

an overt "ClickMe" icon, purposely designed to be conspicuous and inviting to click on (Fig. 8). We found this blunt instrument to be an effective technique in places where the success of the experience hinged upon establishing unambiguous communication with the user.

6 ANIMATION

One of the most significant improvements over VRML 1.0 is the support for animation and behavior in Moving Worlds. In a nutshell, animations in VRML usually begin with a trigger that sets off a timer. This timer, in turn, drives an engine that modifies field values defining certain characteristics of the scene. Exceptions and embellishments abound, but they are all characterized by this basic pipeline.

6.1 Animation Triggers

In OOBE, our animations are usually triggered in one of three ways:

- Proximity: When the user crosses into a region of space, specified with a ProximitySensor, a trigger event is sent to start a TimeSensor that drives the animation sequence (Fig. 9). The self-opening gates in *Jungle Island* are an example of this.

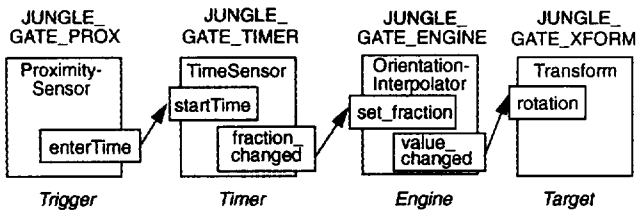

Figure 9. Routing diagram for an animation triggered by a ProximitySensor.

- User Input: When the user clicks an object, an associated TouchSensor detects this interaction and sends a triggering touchTime event to start a TimeSensor that drives the animation sequence (Fig. 10). The *Jungle Island* gates can also be opened this way.

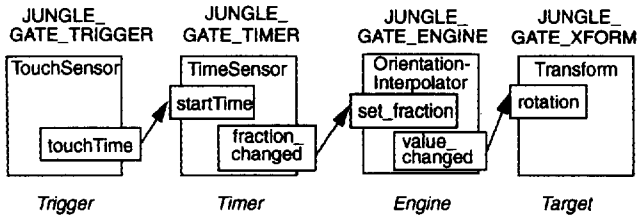

Figure 10. Routing diagram for an animation triggered by a TouchSensor.

- Auto Start: Animations can also be started up automatically when a scene is first loaded. This usually requires setting the TimeSensor fields with special-case values (Fig. 11). The animating waterfall in *Jungle Island* is triggered this way.

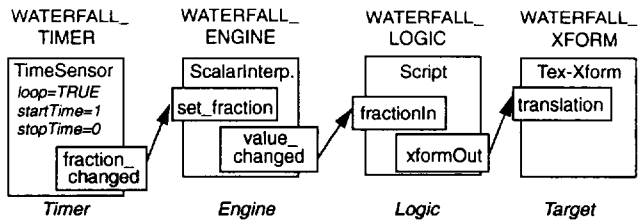

Figure 11. Routing diagram for an auto-starting animation.

6.2 Organic Animations

Animation is achieved by interpolating through a series of keyframe values, carefully posed by the world builder. In OOBE, we often employed organic animations, to give our subjects more life and personality. We asked our objects to squash and stretch, ease in and out, and follow through with secondary actions [5]. The Moving Worlds interpolators provide only basic linear interpolation between their keyframes, so we often resorted to higher-level means to generate keyframes that achieve the subtlety we desired. While the bouncing shapes in Boink were keyframed manually using traditional animator techniques (eyeballing), the *RaptorBuilder* animation sequences were performed using a curve-based keyframe animator in Cosmo Worlds. Because acceleration and deceleration are accommodated, curve-based animations tend to be less jerky and flow more naturally. We found the use of a curve-based animation system to be critical in achieving organic animations. Note that animation curves will require conversion to linear keyframes to be compliant with the VRML 2.0 specification, which can increase file size.

6.3 3D Morphing

In *CyberAnatomy101*, we pushed the envelope of traditional animation by creating something a bit out of the ordinary. *CyberAnatomy101* features 3D morphing of our subject from a checkered floor into a humanoid (Fig. 12). We implemented this effect using the CoordinateInterpolator. The result was organic and effective. As pleased as we were with the outcome, we found the process quite painstaking. In creating the morphing sequence, intermediate shapes were composed with the exact same number of vertices and in the same sequence, with each shape representing a keyframe. This requirement is extremely important in the proper outcome of the morph. Each set of coordinate point values was then cut and pasted sequentially into the keyValue field of the CoordinateInterpolator until we ended up with a long linear list of SFVec3f's. The vertex count in this list should be an exact multiple of the number of keyframes in the animation. At this point, the CoordinateInterpolator can use the keys to index into the list of coordinate vectors for the 3D morphing sequence (Fig. 13).

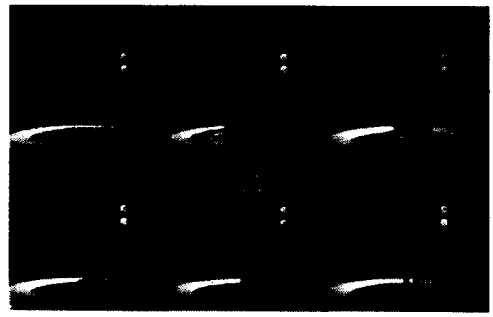

Figure 12. Time-lapse photographs of the 3D morph in *CyberAnatomy101* space.

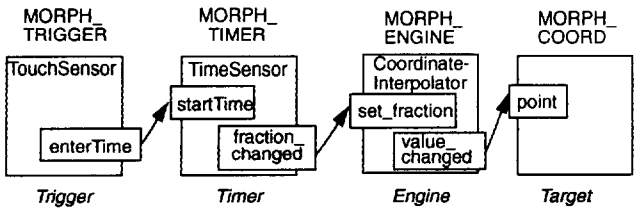

Figure 13. Routing diagram for 3D morphing. The morphing engine (implemented with a CoordinateInterpolator) interpolates between 3D objects specified as key "poses."

7 APPEARANCE AND MATERIAL PROPERTIES

Understanding the interplay of material, color, texture, and lighting is the key to achieving great looking VRML worlds. But this is often more complicated than it seems. In reality, most world builders will notice drastic disparities in rendering results from browser to browser. It was clear up front that we would be relying heavily on texture mapping to achieve the organic look we were after. However, we found that even the use of full-color textures on their own somehow possessed a sterility that left more to be desired.

7.1 Color-Per-Vertex and One-Component Textures

We found the look we desired by combining polygonal color-per-vertex shading with the application of one-component textures, letting them blend together to achieve greater richness. The result was a more true-to-life rendering of an otherwise sterile and ordinary subject. (See color plates)

According to the Moving Worlds specification [1], only one- and two-component textures can blend with color-per-vertex information on the surface they're combined with. This cuts out the ability to experiment with the much richer look which could be achieved by blending three- and four-component textures with color-per-vertex surfaces. For this production, we converted all three- and four-component textures down to grayscale and adjusted their levels in Adobe Photoshop™ in order to attain the richer blending we sought.

Our experience has shown that the ability to author color-per-vertex information is highly beneficial in a VRML authoring tool. World builders and tool makers should learn to exploit this feature more, and browser writers should pay attention to supporting this capability well. The difference can be quite rewarding, as seen in figure 15c.

7.2 Specularity and Shininess

The addition of specularity and shininess values can further enhance the richness of an object rendered with one-component texture and color-per-vertex information. For example, the island heads on *Jungle Island* benefit from this technique. The result is a warm sheen that is only noticeable at certain viewing angles. This subtlety actually enhances the dynamics of the overall look without overpowering the other appearance elements.

7.3 Texture Mapping

One of the best techniques for adding apparent complexity and details to a scene efficiently is to model them in image space, using texture maps. Because the O2 workstation excels in this area, it made sense to explore the use of texture maps in places where geometry was traditionally the only alternative.

When using texture maps in VRML, factors to be considered include:

- file size
- texture size
- use of textures with alpha transparency
- cross-platform compatibility

In the majority of OOBE, we used JPEG textures to leverage their great image quality to file size ratio. However, for grayscale textures, we used SGI's BW file format; and for color transparency textures such as the palm trees on *Jungle Island*, we used SGI's RGBA file format. However, neither of these formats are VRML standard. Authors should instead use PNG for color transparency textures, and perhaps GIF for grayscale. In general, we limited all textures to be no larger than 128x128 pixels, allowing for a few exceptions where more detail was critical.

When texture maps are used as billboard signs or display screens in a world, we find that increasing the emissiveColor value to 1.0 is helpful to bring out the color fidelity of the texture. Within the customer spaces in *Innovations Gallery*, we used this technique to simulate backlit wall displays.

It is well understood that texture maps can add apparent geometry to a scene, perhaps making it appear twice as complex. But we also discovered that texture mapping has a valuable impact in the way it aids navigation. In the OOBE *Courtyard*, where sweeping pathways are constructed with large triangles for efficiency, textures introduced vital spatial cues to the navigator, drastically improving interactive feedback regarding the speed of travel and even minor changes in direction (Fig 14). This technique is used in *Jungle Island* extensively as well.

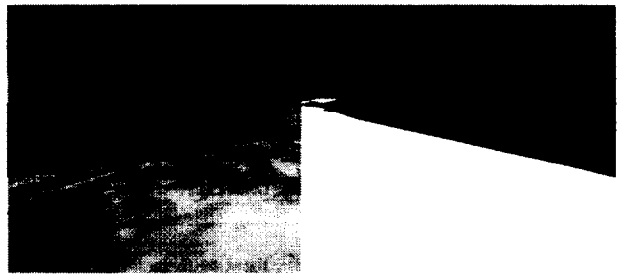

Figure 14. Texture mapping the ground can enhance the navigational experience. On the right is "Mainstreet" without a texture map. Notice there is very little depth cue due to lack of surface features. The application of a one-component texture, shown on the left, provides speed and directional cues when navigating.

Because it is often impossible for an authoring system to automatically know how a texture should to be mapped onto a complex surface, it can make certain assumptions. Unfortunately, these usually result in texture smearing and clumping. Given this, world builders will find that a tool that allows them to control how a texture should be mapped onto geometry, with absolute control down to the texture coordinate level, is indispensable. Our production team used Cosmo Worlds' Texture Applicator extensively, to fine-tune elaborate mappings such as the *Jungle Island* rainforest texture. We would be eager to see tool builders support higher level texturing strategies, such as direct 3D texture painting[3] and procedural texturing, in the future.

8 LIGHTING

Today, most worlds use nothing but the default browser headlight; lighting practice in VRML is at best an afterthought. Yet, some of the most memorable motion pictures are remembered for their creative use of lighting. Imagine if Blade Runner were lit with only one headlight and nothing else. Great lighting can evoke moods and contribute significantly to the storytelling process [7]. However, lighting does not come for free. An understanding of how various types of lights contribute to a scene creatively, as well as how they affect scene performance at runtime, is crucial to the successful use of lighting in VRML.

8.1 Lighting for Mood

A seldom explored aspect of VRML lighting is its ability to create mood. In the Alcoa customer gallery, the space is dominated by heavy machinery using extreme heat and pressure to manufacture automobile wheels. We applied animating orange and red lights here, to engender the feeling of heat in an hostile industrial environment. Similarly, in the *RaptorBuilder* space, a sense of activity and undercurrent is achieved by modulating the intensity field of the DirectionalLight source, simulating the illumination of molten lava (Fig 15).

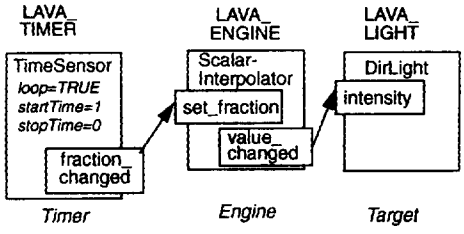

Figure 15. Routing diagram for a pulsating DirectionalLight source simulating lava flow.

Taking our cues from successful cinematography, we usually avoid use of the default VRML headlight. As in snapshots taken with common point-and-shoot cameras, front-lit subjects tend to look flat and uninteresting. Studio photographs, on the other hand, possess a dimensional quality to them, due to their studied use of multiple light sources placed to the side and behind the subject.

8.2 Simulated Lighting using Color-Per-Vertex

Unfortunately, a completely literal lighting plot, as used to illuminate a stage or movie set, is out of the question, given today's limitations in lighting performance. An alternative way to achieve these dramatic lighting effects, however, is through the very practical use of color-per-vertex shading throughout the scene.

In essence, lighting effects can be applied directly onto the geometry itself, to simulate shadows, shading, and highlights. One benefit of this technique is the ability to selectively control how a scene and its objects are shaded in ways that real-world lighting cannot achieve. This is analogous to a painter selectively painting shadows and nuances for maximum effect throughout the scene, in contrast to the cinematographer's lighting, which must follow from the laws of physics. Many commercial systems utilize this technique to achieve high-performance, photorealistic rendering [7].

Color-per-vertex shading can be used in conjunction with normal VRML lights, or it can be used entirely in place of "real" lighting. The latter case can speed up rendering dramatically, because no runtime lighting calculations need be performed. For the tunnels in the *Courtyard*, we use a combination of DirectionalLight and color-per-vertex lighting to transition from a sunlit environment to a dark, ominous tunnel, and back out into the sunshine again (Fig. 16). This interplay between the color-per-vertex shading, the sunlight, and the orange specular component of the tunnel results in an evocative blend of warmth, shade, and dimension.

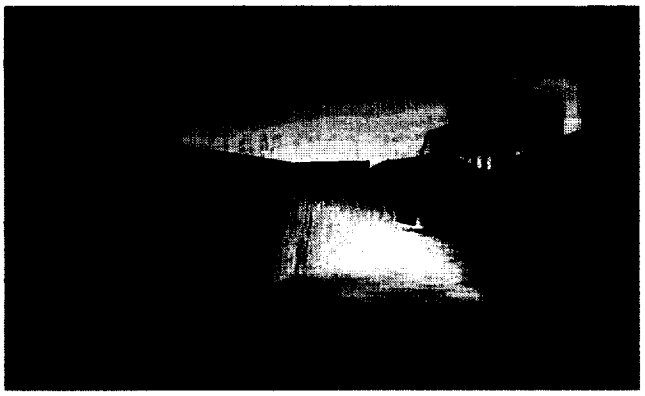

Figure 16. Using color-per-vertex to paint lighting onto geometry. This technique is more flexible and performs faster than real lighting.

These techniques can be extremely difficult to control without a tool that enables the interactive control of light placement and direction in the scene, and the ability to edit color-per-vertex information under the influence of actual lighting conditions.

8.3 Performance Considerations

Lighting performance overhead is a direct function of the number and type of lights active within the scene. In general, a DirectionalLight is quite a bit less expensive than a PointLight or a SpotLight [4]. Given this fact of life, we defaulted to the DirectionalLight for lighting in most cases. We splurged on PointLights or SpotLights when we could afford to take advantage of their dramatic attenuation characteristics, at the cost of economizing elsewhere in the same scene. The beamWidth and cutOffAngle fields allowed us great selective control in defining how much light interacts with our featured geometry.

Our lighting setup usually consists of one or two DirectionalLights as rim lights, counter-balanced by a SpotLight as the primary fill. For spaces containing up to ten light sources, such as the *Entryspace*, we use Switch and Script nodes to turn lights on and off depending on proximity. This way, there are never more than three lights on at any one time. Generally, we find that we are able to attain our sustained frame rate goal of 10 frames per second while using this lighting strategy on the O2 workstation. Note that PC platforms may have greater lighting limitations.

9 SOUND

Like an engaging soundtrack that contributes to a motion picture, sound plays a crucial role in the virtual reality medium. Effective use of sound can establish setting, convey mood, evoke emotions, and even foreshadow the unseen. It is an integral part of the VRML experience.

9.1 Resource Requirements

Because sound files tend to be large, understanding their resource requirements is an important part of the load-balancing strategy. Proper choice of sampling rate, sample size, file format, and num-

ber of channels and samples will ensure short load time and efficient CPU usage. Again, the target platform and delivery mechanism must be considered. Playing a 44.1 kHz sample through built-in factory speakers can be wasteful, while playing an 8 kHz sample through power speakers can magnify this mismatch.

Initially, OOBE was designed to deliver CD-quality (44.1 kHz) audio samples. This was later downgraded to 22.05 kHz to reduce file size. Because our sounds are generally spatialized, it was unnecessary to use two-channel stereo samples. We used mono sources in the WAV format either created from scratch or sampled off sound library CDs.

9.2 Designing for Ambience

The purpose of ambient sounds is to support the visuals without competing with them. Ambient sounds can be either musical or not, ranging from an instrumental composition to rain forest noises. The challenge lies in authoring ambient sounds that appear to loop infinitely without sounding repetitious. This requires selecting the right sample and proper loop point (the transition between the end and the beginning of a loop). Loop points must not be noticeable.

In OOBE, we found the optimum ambient sample length to be about 15 seconds, based on load time and nonrepetition. We use ambient sounds that are effective in conveying the appropriate mood in a given space. Selections are also picked based on how easy they are to loop. An improper choice can immediately become annoying after a few iterations. One useful technique for reducing repetition in a sample is to combine it with another sample of different length during playback. When these separate samples are playing simultaneously, the phase difference creates the illusion of randomness which effectively masks out repetition.

For fluidity, multiple ambient sounds should flow seamlessly from one to another. This is achieved through specifying overlapping radii of influence in the Sound nodes and letting the samples cross fade.

9.3 Designing for Effect

The role of sound effects is to support and reinforce animation. Sound effects need to synchronize accurately with what is being animated. Due to system latency, or samples that inherently do not match animation keyframes, synchronization remains the primary challenge and the focus of authoring.

The majority of custom sound effects in OOBE were a combination of many individual samples merged and synched to motion. Samples are inserted at time slices roughly corresponding to the animation keyframes of interest. Further timing refinements were usually necessary to match the precise attack times with the height of actions. This step involves trial and error for optimum playback result. Once samples are merged, it is difficult to make timing changes. We found that minor adjustment to animation keyframes is often a more feasible solution than tweaking the sound sample itself.

Sound effects can be bound to animating geometry for maximized spatial effect. By adding a Sound node to the parent Transform that is animating, sound can follow geometry wherever it goes. Sound level rises or attenuates depending on proximity. This spatial illusion is achieved by specifying a relatively small radius of influence in the Sound node. The effect can be quite convincing, as demonstrated by the animating racing cars in the Team Sauber customer gallery.

9.4 Quick Tips for Effective Sounds

- Make interactive frame rate a higher priority than sound.
- Choose sound parameters based on target platform limitations and type of delivery mechanism.
- Use short mono samples.
- Keep the sampling rate below 32 kHz unless quality is most important and download time is not an issue.
- Select short ambient samples that do not sound repetitive and are easy to loop.
- Use multiple loops of different lengths to mask repetition.
- Instance sound whenever possible to reduce load time.
- Make an effort to synchronize attack times to animation keyframes of interest.
- Use a small radius of influence for sounds bound to animating geometry.

10 SUMMARY AND CONCLUSION

In this paper we have presented a real-world example of a major VRML 2.0 production project. Table 1 illustrates the scope of this project:

Table 1: . OOBE Statistics

Approximate number of polygons per scene	1500-5000
Size of scene, in pixels	1255x630
Number of textures, total	1,061
Number of WRL files, total	73
Number of WAV sound files, total	165
Number of MPEG movies, total	69
Number of HTML files, total	829
Total size, in MB	318.7

We have highlighted the challenges and pitfalls we faced, the design decisions we made and the lessons we learned during each stage of the production. Here are some of those lessons:

- Make performance a first-class citizen throughout the project.
- Determine your polygon budget and stick to it!
- Let user interaction and serendipity rule over high polygonal detail.
- Don't rely on file conversion to give you well-formed VRML 2.0.
- Watch out when using spline-based or curve-based modelers which are designed for realism, because they can produce high polygon count models.
- Use LODs, ProximitySensors, VisibilitySensor and other VRML 2.0 constructs to help interactivity.
- Use instancing.
- Use lighting both to indicate mood and to aid navigation .
- Use different navigational "styles" for users with different familiarities with 3D navigation.
- Provide 3D "widgets" that are consistent and familiar throughout the experience.
- Use many types of animations to give a compelling experience, with different ways to trigger them.

- Use color-per-vertex and texture mapping to provide detail and enrich the visual presentation without adding polygons.
- Use color-per-vertex to simulate lighting, without incurring lighting's performance cost.
- Use one-component textures plus material color for a richer effect than three-color textures provide.

In conclusion, we hope that both future content creators and VRML 2.0 world building tool makers learn from the experiences we had in developing OOBE. The production experience taught us many invaluable lessons about how to use this technology effectively in the creation of compelling content. The most resonant lesson remains the need to fully understand and respect the limitations of the underlying graphics platform, and how to squeeze the most VRML out of them. If Content is King, then Performance is the Kingdom. Without an acceptable interactive frame rate, the content simply cannot rule. Our experience with OOBE has shown that a compelling VRML experience cannot have one without the other -- they must have both.

There are many applications of VRML 2.0 which we would like to investigate in the future for OOBE. Areas in behavioral animations using mathematical oscillators and flocking behaviors [8], avatars, and multi-participant worlds are candidates.

11 ACKNOWLEDGMENTS

The authors would like to thank all the reviewers of this paper for their time and feedback: Dave Story, Jackie Neider, Howard Look, Rich Gossweiler, Ed Allard, and Eric Debolle. A special thanks goes to Josie Wernecke, who helped out tremendously with the formatting, diagrams, and general readability. Also, many thanks to the OOBE and Cosmo engineering team for their invaluable contributions, and to Construct Internet Design (www.construct.net) for their initial models and behaviors. Finally, thanks to Silicon Graphics, Inc., for making this type of creative and technical work possible.

References

[1] Bell, G., Carey, R., and Marrin, C. The Virtual Reality Modeling Language (VRML) Version 2.0 Moving Worlds Specification. 1996. In *VRML Architecture Group Web Site:* http://vag.vrml.org

[2] Bell, G., Parisi, T., Pesce, M. The Virtual Reality Modeling Language (VRML) Version 1.0 Specification, 26 May 1995. In *VRML Repository:* http://www.sdsc.edu/vrml

[3] Hanrahan, P. and Haeberli, P. Direct WYSIWYG Painting and Texturing On 3D Shapes. Proceedings of SIGGRAPH '94 (Orlando, FL, July 24-29). In Computer Graphics Proceedings, Annual Conf. Series, ACM, New York. 1994. 215-223.

[4] Hartman, J. and Wernecke, J. *The VRML 2.0 Handbook: Building Moving Worlds on the Web.* Addison-Wesley Publishing, Menlo Park, CA. 1996. 101.

[5] Lasseter, J. Principles of Traditional Animation Applied to 3D Computer Animation. Proceedings of SIGGRAPH '87 (Anaheim, CA, July 27-31). In Computer Graphics Proceedings, Annual Conf. Series, ACM, New York. 1987. 35-44.

[6] Neider, J., Davis, T., and Woo, M. *OpenGL Programming Guide: The Official Guide To Learning OpenGL.* Release 1. Addison-Wesley Publishing, Menlo Park, CA. 1993.

[7] Pausch, R., Snoddy, J., Taylor, R., Watson, S., and Haseltine, E. Disney's Aladdin: First Steps Toward Storytelling In Virtual Reality. Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4-9). In Computer Graphics Proceedings, Annual Conf. Series, ACM, New York. 1996. 193-203.

[8] Reynolds, C.W. Flocks, Herds, And Schools: A Distributed Behavioral Model. Proceedings of SIGGRAPH '87 (Anaheim, CA, July 27-31). In Computer Graphics Proceedings, Annual Conf. Series, ACM, New York. 1987. 25-34.

[9] Wernecke, J. *The Inventor Mentor: Programming Object-Oriented 3D Graphics With Open Inventor.* Release 2. Addison-Wesley Publishing, Menlo Park, CA. 1994.