In the late summer of 1997, we sat in a pub in Toronto, Canada, and began planning an ambitious project: to create a new entertainment medium. The idea was to mount a live performance of Shakespeare’s *A Midsummer Night’s Dream* and “broadcast” it over the Internet. The sets, props, and all the characters were to be modeled using the Virtual Reality Modeling Language (VRML) 2.0. A group of actors would provide the voices, and puppeteers would control the characters. Both the voices and the motion data would be digitized, compressed, and sent out over the Internet in real time. Moreover, people would be able to access this broadcast with a 28.8-Kbps modem connection and a 150-MHz Pentium computer. Less than six months later, VRML Dream made Internet history.1

VRML Dream was the first live, streaming, VRML entertainment project with a running time of more than two or three minutes, and it proved that streaming both motion and voice data over standard Internet connections was possible. This article looks at the development of the project, its models and technology, and the main issues that influenced the final design. It also examines the technology needed to create live, streaming, real-time 3D animation for the Internet as well as some of the challenges that lie ahead.

In search of a new medium

The VRML Dream Project grew out of our interest in both new media and VRML. New media and the Information Superhighway have been the “visions for the future” that will, at least according to the pundits, radically change the way people learn, work, and entertain themselves. However, most of the technologies being touted for this revolution—streaming video and audio, push technology, and other network-based technologies—involve the delivery of traditional media forms through digital communication devices.

In contrast, VRML has been a new media standard in search of an application. A number of companies and individuals have searched for the “killer app” for the standard. Some of the most interesting applications are based in narrative content. A notable narrative project was IrishSpace (http://pluto.njcc.com/~paulsam/irish/Welcome.html), the story of the survivors of a meteor disaster and plague. The running time for this production was more than one hour. Actors provided the voices for the characters, and VRML was used to create the 3D models and animation. However, making the production available over the Internet has not been practical. The sound files for individual sections alone averaged 8 to 10 Mbytes. In contrast, short VRML-based animation such as “Floops” and the Dilbert 3D cartoons (both produced by Protozoa) were designed for Internet access. However, these examples were limited to short (2-minute) vignettes and still needed relatively long download times over a 28.8-Kbps modem.

We both recognized VRML’s potential as a storytelling medium. In *Special Edition Using VRML,* we noted,

All the world may indeed be a stage, but the virtual world is even more than that: it has no ‘fourth wall,’ and no backstage; it is an all-encompassing theater of the mind, in which we all play our parts, and write them as well.

We believed that the standard could become the basis for a truly new medium that could take full advantage of digital technology. Whereas traditional forms of media control every aspect of how viewers can watch the story, a VRML-based production would let audience members move around the scene themselves and watch the play from the viewpoint of any character in the scene. They could also watch the production using a “director’s” viewpoint that controlled what they watched. VRML could, therefore, provide a greater degree of flexibility for narrative structure.

After a number of discussions, we saw that an interesting test of this concept would be to broadcast a “live”...
virtual reality (VR) theater production over the Internet. We also wanted to make the production accessible by as wide an audience as possible. Moreover, the funding needed to pay for developing such an endeavor was not easily available. Therefore, we developed a set of criteria for our project, as follows:

- The project would be developed using volunteers.
- It would use a recognizable work for its story.
- We would use open standards wherever possible.
- The audio for the project had to use an existing solution.
- The project would use existing VRML browsers and Java.
- The production had to be accessible using a 28.8-Kbps modem connection and a 150-MHz Pentium home computer with no 3D acceleration hardware.

Roehl suggested that Shakespeare’s *A Midsummer Night’s Dream* would be appropriate for this medium. Since my background was in Shakespearean studies, I edited the play down to a half-hour running time. We created a mailing list for the project and released a call for volunteers to the vrml and shaksper mailing lists. Thus the VRML Dream Project was born.

**Models for a new medium**

The models for VRML Dream presented a challenging set of problems. The VRML Dream Project aimed primarily to be a viable theatrical production. Moreover, the selection of VRML as the broadcast medium was not simply a technical decision. According to the artistic statement, a key reason for using VRML was “to enhance particular themes inherent within the play itself” (see http://www.vrmldream.com/vrmldream/artistic.html). In other words, the medium also had to play a role in the artistic goals of the production.

At the same time, however, the models had to conform to the kinds of restrictions inherent in developing material for real-time 3D animation and the Internet. The characters and sets had to have low enough polygon counts to allow for smooth animation on the target computer platform. The low polygon requirement placed huge restrictions on the amount of detail that could be used in the models. And since the goal was to ensure that VRML Dream was accessible to people with a standard 28.8-Kbps modem connection, the model designers also had to consider the file size of each model.

Despite these restrictions, the VRML Dream Project fulfilled its artistic goals while achieving its technical ones by establishing design parameters, leveraging the creativity and skills of its modelers, and reworking the models as they were submitted.

**Establishing the design parameters and finding the modelers**

After writing the script, I began to prepare an artistic statement that would establish a coherent vision for the production’s design. I started by going through the script to identify any places where the text made specific references to the look of the characters and the sets. For example, Hermia makes reference to the differences in height between herself and Helena:

> Now I perceive that she hath made a compare Between our statures; she hath urged her height; And with her personage, her tall personage, Her height, forsooth, she hath prevail’d with him. And are you grown so high in his esteem; Because I am so dwarfish and so low? How low am I, thou painted maypole?
These references made it clear that Heremia is much shorter than Helena. Indeed, these lines imply that Helena was unusually tall.

Once we identified the script elements, I prepared an interpretation of the play as well as a textual "sketch" of the different characters and sets. For example, the artistic statement notes that VRML Dream has three main settings: Athens, Quince's house, and the forest. The elements associated with Athens (the lovers, Theseus and Hypopolita, the supremacy of law) are associated with order and artifice. The forest and its elements (Puck, Oberon and Titania, magic and fairies) are associated with chaos and mutability.

We incorporated these themes in the design parameters for the sets. We gave Athens a sense of artificiality by using bright lighting and having the materials "constructed" entirely from stone with no green in sight. In contrast, the forest set featured organic shapes and soft, directional lighting.

The designs for the characters were also sketched out in the artistic vision statement. The Mechanicals, for example, are actors in a play and move in and out of character. To emphasize this theme of artificiality, the avatars of the Mechanicals were designed as robots (see Figure 1). The three main Mechanicals reflected their occupations: Bottom is a weaver; Quince, a carpenter; and Flute, a bellers maker. I provided sketches of Quince and Flute to act as a starting point for the volunteers (see Figure 2).

Once the artistic statement and the initial character designs were made available, we sent out a message asking for volunteers to create the models. Three people—Dennis McKenzie, John Nikkel, and Andrew Reitemeyer—would become the primary modelers, with J. Eric Mason and Stasia McGehee working on specific characters.

**H-Anim**

One of the key design decisions we made was to use the Humanoid Animation Working Group's specification (H-Anim) for the layout of its characters. H-Anim specifies a standard method for representing humanoids in VRML 2.0. Bodies are divided into segments connected by joints (visit the Humanoid Working Group's Web site at http://ece.uwaterloo.ca/~h-anim/). Changing its joint rotations controls a model's body movements. This standard provided the VRML Dream interface designers with a template for sending movement data. It also gave the animators a way of reusing already existing movement data with other characters.

While H-Anim allows for the use of every joint in the human body, you do not have to create characters with that amount of detail. In VRML Dream, the Athenian characters only required opposable thumbs and index fingers to be done individually. The other three fingers could be created in a "mitten" style, thereby reducing the number of polygons needed to render a hand.

**The genesis of the models**

Since the VRML Dream Project was a volunteer effort, we gave the modelers a certain degree of freedom to design and implement the characters and sets. However, translating designs into actual VRML models still required some guidelines. All the models were to use the VRML convention of 1 unit equals 1 meter as a way of standardizing the size. The method of creating the VRML code was left to the individual modelers.

**Polygon count versus file size.** As the first models were being developed, discussions about their design and implementation became the main topics on the VRML Dream mailing list. The modelers announced that a character, set, or prop was completed and made it available for viewing over the Web. The members of the VRML Dream list would then provide assessments of the models. We would also reply directly to individual modelers to offer advice and recommend design changes. As the modeling process continued, the VRML Dream Project established a number of key principles as follows:

1. All files had to be small. The files could not exceed 60 Kbytes when uncompressed.
2. The models had to have a rendering rate of least 5 frames per second (fps) when viewed in a VRML browser.
3. The polygon counts and number of texture maps used for the characters, sets, and props had to be minimized.

One of the first issues raised on the mailing list was the frame rate. A number of people on the list noted that the VRML browsers were having problems rendering some of the early versions of the models at acceptable frame rates. This issue was a major concern for the project. To make VRML Dream a success, the animation of the characters and the camera movements had to be fairly smooth even when several characters appeared on the set simultaneously.

One area where frame rates could be improved was by reducing the number of polygons to render a model. Although it was important to provide a certain amount of detail, the number of polygons to create a model had an inverse effect on the frame rates. A large number of polygons resulted in a lower frame rate. Conversely, a model using fewer polygons had a higher frame rate. Therefore, it was in the project's best interest for the modelers to use as few polygons as possible.

The VRML Dream mailing list also expressed concerns about the size of the VRML and texture files. Since the goal was to make the broadcast available over the Inter-
net, the file sizes had to be minimized. One area where this goal could be achieved was by reducing the number of texture maps as well as optimizing the ones that were needed. All texture maps were reduced to 256 colors or less, and we kept their size to 128 by 128 pixels. In the end, the original VRML Dream production used only 12 texture maps totalling approximately 36 Kbytes.

A number of discussions in the mailing list focused on using primitives (such as spheres, cylinders, and cones) versus using IndexedFaceSets (IFS). Primitives require very little file space to define a model. However, the number of polygons used to render those shapes can be quite high. For example, Hermia’s hair was originally modeled using a black sphere, which required almost 300 polygons to render. In contrast, Dennis McKenzie remodeled her hair using an IFS and achieved the same result using less than 100 polygons.

Balancing the size of the file with the polygon count, however, quickly became a nonissue for the mailing list. As more models were created, members of the list determined that the increase in file size was fairly negligible once the files were compressed. In contrast, using primitives could significantly affect the frame rates. Therefore, the bulk of the models used an IFS as their base.

Further optimizations. After the modelers finished their work, we began combining the files to see if the rendering speeds and download times for the production were acceptable. In most cases, they were too slow. Therefore, elements had to be redesigned to improve the frame rate.

One example was the set for Theseus’s temple. In Andrew Reitemeyer’s original design (see Figure 3), the set contained a number of props including a throne, two
large urns, a table, and several scrolls. Moreover, the back of the set had walls, and the various models used texture mapping heavily. While this set conformed to the target frame rate when viewed alone, any characters added to it reduced the speed to unacceptable levels.

To improve the frame rate and reduce the size of the files, J. Eric Mason and I began redesigning the set. Mason began by reducing the details in the columns and increasing the `creaseAngle` values to create the curves. I removed most of the props and the back of the throne. I also reduced the number of columns in the scene and eliminated most of the texture maps. The result was a doubling of the frame rate and a greatly reduced file size. Moreover, these changes were accomplished without compromising the main design elements (see Figure 4).

Once the models were finished, we looked for ways to further reduce file sizes. A key factor in this reduction lies in the deletion of default values from the VRML files. Authoring packages such as Ligos Corporation’s V-Realm Builder and Cosmo Software’s Cosmo Worlds prove extremely useful in creating complex models and environments. However, they also have a tendency to include default values for various fields. These elements are not required by the VRML browser and can be omitted from a VRML file.

Using Vorlon, a free VRML checker from Trapezium Software, I identified where default values appeared in the different VRML files. I then used a text editor to manually remove them. This process alone reduced the size of the VRML files by 10 to 15 percent. Another method for reducing file size was removing unnecessary white space. In the process of creating VRML models, authoring packages would use indentation to make it easier for users to edit their files using a text editor. By writing a short Unix script to remove these indentations, Roehl reduced the size of the files by an additional 20 to 30 percent.

**The technology of VRML Dream**

The audience could view the performance using a VRML browser (either Cosmo Player or Worldview) and listen to it using a freeware audio application called Speak Freely developed by John Walker (http://www.fourmilab.ch). The only hardware required was an ordinary Pentium with a sound card and a 28-Kbps modem. Using these technologies, VRML Dream let the audience become more intimately involved in the performance by placing them in the midst of the action. Audience members could switch between a number of preset viewpoints, follow the default “director’s view,” watch the show through the eyes of one of the characters, or navigate through the virtual set and choose their own angle.

At the production studio, a group of actors gathered around a set of microphones, which were fed into an analog mixing console. We fed the output from that mixer to a PC running Speak Freely and transmitted the audio data to a Unix system that retransmitted it over the Internet. At the same time, a group of puppeteers controlled the virtual characters’ movements. That data was sent to the Unix system for redistribution.

Much of the software needed to produce VRML Dream had to be developed by the project team. To understand how it works, it’s necessary to examine the flow of 3D motion data through the system from the puppeteering station to the client machine.

**Puppeteering**

Each puppeteering station had a number of standard input devices such as a keyboard, a mouse, and one or more joysticks. The software also allowed the use of six degree-of-freedom (DOF) input devices such as spaceballs or trackers. In addition, the software was designed in such a way that integrating motion capture systems was straightforward. Each axis of each input device and each input button was mapped to a virtual “data channel.” In turn, the data channels could be mapped to any arbitrary event in a VRML scene graph. For example, the shift key on the keyboard can be treated as a binary input that can control whether an eyelid is open or closed on one of the characters’ faces. Similarly, one axis of a joystick device can be used to control the bending of an elbow, and one axis of a mouse can control the tilting of a character’s head.

The project’s programming team wrote puppeteering software in pure Java, with the exception of some native methods in the classes that dealt with the joysticks. Since Java has no joystick support, the use of native methods could not be avoided. The 6-DOF interface was adapted from earlier work involving the use of DirectInput with Java. The native methods were essentially an interface to the DirectInput system, which was how the team provided support for a range of 6-DOF input devices. The software also maintains an internal representation of the state of each character and periodically retransmits that state as well as any state changes. The data is encoded into a simple packet format and sent via a user datagram protocol (UDP) to a “relay” server. The reason for periodically retransmitting the state is twofold. First, it ensures that even if packets are lost the state will quickly be updated on the client side. Second, it allows audience members who join in late to obtain the complete and up-to-date state of the entire world in just a few seconds.

The team used several techniques to reduce the amount of data. Since most of the data consisted of joint rotations of the various characters, they needed an efficient encoding of that rotation information. Instead of using VRML’s axis-angle format, which required four floating-point numbers per joint, the software transmitted the joint data using Euler angles. Since most joints in the body cannot rotate more than ±127 degrees, and one degree of accuracy more than suffices for this application, 1 byte per angle was transmitted. With a 1-byte channel identifier, only 4 bytes were sent per joint per frame. This figure was a vast improvement over the 16 bytes required to send a standard VRML rotation. For some joints, such as the elbows and knees, only a single byte needed to be sent. This dramatically reduced the amount of data transmitted.

The programming team digitized and compressed the audio data using the Global System for Mobile telecommunications (GSM) algorithm. The resulting compressed data packets were encapsulated using the...
Real-Time Protocol (RTP) and again sent via UDP. (For more information about RTP see http://www.cs.columbia.edu/~hgs/rtp/.) The GSM encoding required only 13 Kbits per second, which meant that even on a 28-Kbps line, half the bandwidth remained available for the motion data. The audio encoding and playback was handled entirely by Speak Freely. The team chose Speak Freely because it was free and available with full source code in C for both Unix and Windows.

Relays

Another component of the system is the relay server. When the audio and 3D animation data arrive at the relay server, they’re sent to a number of other hosts, including both client systems and other instances of the relay server running elsewhere. In effect, the relay servers form an inverted tree of hosts that relay data from the root server to the leaf node clients (see Figure 5). Relays and clients that wish to receive data from a particular relay periodically send keep-alive packets to the relay to let it know that it should continue sending data.

A multithreaded Java application, the relay program can handle both the audio and motion data streams. It also remains independent of the content or format of the data itself. In addition, the relay program is highly configurable and even supports remote reconfiguration via telnet. That is, an administrator can telnet to a relay, enter a password, and then change almost any aspect of the relay’s configuration.

Initially the project team was concerned over whether a Java application would be able to keep up with the steady stream of data, but those fears proved unfounded. Provided the fan-out (the number of listeners per relay) was kept to a reasonable number, performance was quite acceptable. The relay program could also log the amount of data transferred. In theory, future versions could even track the number of audience members who view the performance.

Obviously, the relay-tree approach has some disadvantages. The main one is that the latency increases with the depth of the tree. However, the latency in a one-way data flow is irrelevant. Even if the audience sees the performance happening several seconds after it “actually” took place, from their subjective standpoint it’s still instantaneous. This is one way in which a streaming performance differs substantially from a multiuser environment.

The other significant disadvantage of the relay-tree system is that it needs a large number of relay servers, which requires considerable organization and coordination. Fortunately, the VRML Dream was a relatively small-scale test. The relay-tree approach resembles the way network television programs are distributed. From a central head end, the programming is sent to local affiliated stations, which in turn broadcast the signal in their area so that viewers can receive it. This is analogous to the relay tree for VRML Dream. Eventually, it may be possible to transmit additional streams from a relay host to the client in order to supply secondary content such as advertising. This approach again resembles the network television model, in which individual stations derive part of their revenue from local advertising.

In the longer term, multicasting will likely be preferable to the relay-tree approach. However, the lack of widespread support for multicasting made it necessary to use a relay-tree for the original VRML Dream.

The client side

The final part of the system is the client-side software. It receives the data stream and uses it to update the world. For the first performance, the client software consisted of a Java applet that ran on the same Hypertext Markup Language (HTML) page as the VRML world. The applet received the streaming VRML events and fed them across the External Authoring Interface (EAI), a mechanism for Java applets to access the world running inside a VRML browser. Once inside the VRML world, the events were sent to a Script node in the scene graph for subsequent distribution to the various characters, lights, sets, and so on.

This approach, however, had its share of problems. An important concern was the overhead imposed by going through the EAI. Each EAI call would force a redraw of the screen, to make sure that the current view of the world was correct. It was therefore necessary to cluster the updates into an MFFloat object (essentially an array of floating-point values), hand it across the EAI to a Script node, and have the Script node differentiate the data and send it to the various nodes in the scene. The client applet also cached the state of each entity in the world so that only values that actually changed were sent through the EAI.

This approach certainly was awkward and inefficient. We’re currently working on finding better techniques.
The road to a new medium

The VRML Dream performance was multicast live over the Internet on 26 April 1998. The production had problems with the transmission of the live audio due mostly to Speak Freely’s inability to handle large multicasts. In addition to the audio problems, some viewers didn’t receive the streaming motion data. Adding additional monitoring capabilities to the Relay server would help track these problems. For example, if the viewers who did not receive the data were connected to the same relay, we would know that that particular instance of Relay was not working correctly. If the data was transmitted from the Relay to the client, the client-side debugging information would help narrow down the problem.

Despite these problems, the VRML Dream project was a success. The motion data system worked for the majority of users, and subsequent tests have shown that the motion and audio streaming technology does work. Moreover, interest in the technology continues to be high. Indeed, the response to the project has been overwhelming. Even though the project limited its press releases largely to specific mailing lists, the Web site received an average of six hundred hits per day in the weeks leading up to the performance.

The VRML Dream Project proved that it’s possible to use VRML as a medium for streaming 3D real-time animation broadcasts over the Internet. It taught us about the process involved in creating this kind of content. However, the experiment has not ended—it has just begun. The VRML Streaming Working Group (http://www.vrml.org/WorkingGroups/vrml-streams/) is developing standards for the streaming of audio and event data into a VRML world. Once such a standard is in place, much of the VRML Dream software will no longer be necessary, since streaming support will be built into the browsers. In the longer term, we expect the MPEG-4 initiative to provide good, efficient solutions to streaming 3D content over the Internet.

Commercial interests are also beginning to see the potential of this type of medium. Companies like Oz (http://www.oz.com) and Cicada Interactive (http://www.cicadaweb.com) are developing their own software for producing real-time 3D animation over the Internet. Along with Jonathan Kean, we have formed The VRML Dream Company in order to continue developing the technology and create new projects. The company is working on ways to improve the audio quality and has developed interfaces for controlling avatars with motion capture equipment. More information about the company and its projects is available at http://www.vrmldream.com.

The VRML Dream Project was the first step in establishing a truly new medium for the digital age, but it’s not the last. As streaming audio and motion data improve, we’ll see more and more content being developed for it. At the same time, we must be careful not to blindly apply the methods of more traditional forms such as video. We must continue to take risks and experiment with these new technologies to determine what will and will not work.

References


Stephen Naoyuki Matsuba is a founding partner of The VRML Dream Company and is co-chair of the VRML Streaming Working Group. His VRML-based work includes designing and developing commercial sites for Onyx Computers and the HS Project’s prototype retail Web site. He also created Construct(s) and Même-ing: An Essay in Three Dimensions, a series of five VRML environments exploring issues concerning the new media. He co-wrote Special Edition Using VRML (Que Publications, 1996) and was a contributing author for Hacking Java: The Java Professional’s Resource Kit (Que Publications, 1997).

Bernie Roehl is a founding partner of The VRML Dream Company and a software developer based at the University of Waterloo in Ontario, Canada. He developed the freeware VR package called Rend386, which won the 1995 Meckler award for outstanding software achievement. He has authored a number of books on VR including Playing God: Creating Virtual Worlds (Waite Group, 1994) and Late Night VRML 2.0 with Java (Ziff-Davis Press, 1997). He is a co-chair of the VRML Streaming Working Group and the Humanoid Animation Working Group.

Readers may contact the authors at The VRML Dream Company, 620 Jarvis St., Ste. 2712, Toronto, Ontario, Canada M4Y 2R8, e-mail matsuba@vrmldream.com or broehl@vrmldream.com.