Chapter 3: How VRML Works

Before we go any further, before we install and use VRML, it's a good idea to have some background on how VRML works, and, beyond that, how the Web operates.

The World Wide Web is built out of two essential components – browsers – sometimes called *clients* - and servers. They're complementary parts of a greater whole. The browsers make requests for information from Web servers, usually based upon your actions. For example, when you click on a link in a Web document, the browser generates a document request which is sent to the appropriate Web server.

The server, upon receiving the request, interprets it and attempts to fulfill the request by returning a document which matches the request made by the browser. When the reply is returned to the browser, some additional information is sent along with the document. This information, called *content type*, lets the browser know what kind of information it's receiving. Content type is very important, because without it, a Web browser would never know the difference between a text document, which is just alphabetic and numeric characters, or an image, which is a long run of data that would look like garbage if displayed as text. The browser must know the content type of a document in order to display it correctly.

For example, Netscape *Communicator* version 4.0 has the ability to play audio files of several formats, including AIFF, AU, WAV and MIDI formats. But it can't play MPEG audio - though it could play a soundtrack associated with a QuickTime movie. How does it tell one sound from another? Each of them are identified with a content type. These content types are in a format specified by Multimedia Internet Mail Extensions (MIME) – so they're often called MIME types. The MIME type for AIFF files is audio/aiff, while the MIME type for MPEG audio files is audio/mpeg. These MIME types are returned as *document content types* by Web servers when fulfilling a request.

Using highly sophisticated data communications technology, we've been able to eavesdrop on the communication between a Web browser and a Web server. The conversation goes like this:

> **Web Browser:** Excuse me, Mr. Server, do you have the file "important.wrl?"
>
> **Web Server:** Yep...it's about 10,000 characters, two weeks old, and it's VRML.
>
> **WB:** Mr. Server, please send me "important.wrl".
>
> **WS:** Here it is...<send send send send send send send>....done!
>
> **WB:** Thanks!

This example would be much the same, regardless of what type of document the browser requested of the server. VRML documents require no changes to the way Web servers operate; that's a good thing, because it means that it's very easy for people to add VRML to existing Web sites. The one change that might have to be made is insignificant - you might have to tell the Web server that VRML documents have an *extension* - a file ending - of ".wrl", and that the MIME type of VRML documents is **model/vmrl**. With this information, the Web server can detect VRML documents, and can inform a browser that a VRML document is being transmitted. These days, most Web servers come pre-configured with the VRML MIME type information. That's the *only* change that is required to make a Web server VRML-capable, and one reason why VRML has become so popular, especially with Web server administrators.

NOTE: In the earlier days of VRML, an experimental MIME type, ***x-world/x-vrml*** *was used. That's been abandoned in favor of the new international standard* ***model/vrml****. However, most VRML browsers remain compatible with both the newer and the older MIME types.*

Once the Web browser has been notified that that it's receiving a document with a content type that identifies it as VRML, the browser loads its associated VRML plug-in. If the browser doesn't have a plug-in to handle VRML, it will ask the user what to do. But – if the plug-in is available – the browser will hand the document to the plug-in for further processing and display.

The first step toward viewing a VRML document is to retrieve of the set of VRML documents which make up the VRML "world" or "scene". A Web server which receives the request for a VRML document attempts to answer the request with a reply. This reply has a content type of model/vrml, because it's a VRML document. The browser loads the VRML plug-in.

Once the document has been received and delivered to the VRML plug-in, it's *parsed*, that is, it's translated from the human-readable text of VRML into a set of objects that the computer can understand. The computer can't understand text; it must translate each text item into an equivalent digital representation before it can "understand" what visible objects and qualities are present in the VRML world.

Once the VRML plug-in has created a list of visible objects, the *renderer* draws them upon the display – generally the monitor attached to your computer. In the real world, you can't see in the dark, so – just like the real world - the virtual worlds created with VRML have lights that illuminate the environment within the computer to make objects visible. Now you've got a VRML world on your display, and ready to examined.

After parsing and rendering, the VRML plug-in has completed all of its setup. Now the *simulator* begins its operation. Virtual worlds have rules – really, tiny programs – which determine their behavior. For example, a ball could be bouncing up and down, or a doorbell might ring a bell if you click on it with your mouse. Or you might want to *navigate* through the environment, virtually walk through it as if you were on your own

two feet in cyberspace, so you could get a better look at where you are and what's around you there. The computer has to be vigilant – aware – of these events, and that's the job of the simulator. The simulator maintains a list of things that *could* happen; that list of things is defined by the VRML world itself. Many times a second the simulator checks to see if any of these possibilities actually have occurred. These actualities are known as ***events*** – a key concept in VRML. If any events have occurred, the simulator performs event processing on each event, then goes and waits for more events, processes them, and so on, in a loop that continues until you exit the Web browser, or go somewhere else and exit the VRML world.

These three steps - parsing, rendering, and simulation – are common to all VRML plug-ins, in all Web browsers. However, these aren't one-shot operations; they can all happen simultaneously. How? A VRML world can be *distributed* - that is, it can be spread across the Web in many different places. In the same way that an HTML page can be composed of text from one place and images from another, a VRML world can specify that some of its objects comes from *this* place, while other objects come from *that* place.

For example, I might be a VRML shop-builder; I design retail stores in cyberspace. If I were a real shop-builder I'd have a crew of my own, and I'd likely subcontract the sheet rock installation to one contractor, the painting to another, and the cabinet work to yet another. I've distributed the work; even though the shop is my creation, I've borrowed talent from other places to make the job a success. Distributed VRML is a bit like using subcontractors - you reference other people's work, out on the Web, but it's included in your world.

This means that VRML worlds often load in stages; first the basic world description is loaded, and then, if there are *inline* worlds – literally, worlds within the world - the browser loads these worlds *after* the basic scene has been loaded. Computer speeds aren't ever quite as fast as we would like, and neither are modems quite as capable as the demands we make upon them. For this reason, there is almost always some delay involved in loading a VRML world - it rarely appears immediately, or all at once. However, in order to make this a lot less confusing, the VRML plug-in will show you an objects will appear before it's been downloaded, parsed and rendered. Before the object appears, it's shown as an empty box of approximately the correct dimensions (called a *bounding box*), which is replaced by the actual object when it's been read in. Called *lazy loading*, it allows the VRML plug-in to take its time - when it has no choice, that is - loading the world from several different places; but it does its best to give you an accurate indication of what the world will look like when it's loaded in its entirety.

This means that you don't need to wait for an entire VRML world to load onto your computer before you begin to navigate through it - the world will continue to load while you move through or interact with the portion you've already received. In a rich VRML world with hundreds of objects, this can be a very important feature.

Perhaps the closest analogy to the information within a VRML document is set design for the theatre. For instance, Samuel Beckett's play *Waiting for Godot* specifies a sparse

stage; the only set decoration is a tree in center rear stage. That's a simple layout, and would be a very simple VRML document, defining only the tree's position, shape and visible characteristics – that is, the color and texture of its bark. A more complicated set, for Eugene O'Neill's *Long Day's Journey into Night* might contain the drawing room of the Tyrone household, crowded with furniture and memorabilia. A VRML document describing this stage set would be much larger.

But, beyond describing the content and layout of a world, the VRML document also defines the interactive qualities of these objects, that is, it defines how action upon these objects can trigger events, and what happens once an event has been triggered. A lamp can have a visible "On" switch which – when you click on it - sends an "on" event to a light source associated with the lamp.

All of the linking that you find in the Web today is present in VRML worlds; it's just another type of the interactivity offered in VRML. It's easy to describe an *anchor* – that is, a link into the Web – which is connected to an object defined in a VRML document. When you click on the object the VRML plug-in immediately loads the link, just as it would in an HTML document.

So, taking our examples a bit further, the tree on the set of *Godot* could be linked to the script of the play. Or, the items in the Tyrone residence could be linked into the lines for the various characters. This linking capability makes VRML very powerful; an object, even any part of an object in a VRML world can be linked to any object available in the Web. Even more, it's possible to link VRML worlds together. Just as it's possible to travel from page to page in the Web by clicking, you can travel - it's called *teleporting* - from world to world.

Each VRML scene has a "point of view", which is called a *camera*. (The stage metaphor gets a lot of work in VRML.) You see the scene through the eye of the camera. It's also possible to predefine "viewpoints", which are VRML's equivalent of "scenic areas" - where the world's creator has created several points of view. In most VRML plug-ins it is possible to go directly to any viewpoint by making a menu selection (these viewpoints have names associated with them like "Entrance" or "Ticket Booth"). This can be a great convenience, especially for novice VRML users, who sometimes "get lost" – disoriented - as they're learning to navigate through cyberspace. It keeps them on a "flight path" to all points of interest within the virtual world. For example, if you create a walkthrough of a famous museum like the Louvre in Paris, you might create viewpoints in front of some of their famous works like *Mona Lisa*, so visitors can travel directly to those points of interest.

Those are the basics features of VRML. Now that you understand how Web browsers talk to Web servers, and how VRML plug-ins parse, render and simulate VRML worlds, let's move onto a basic introduction in 3D graphics.