WAXweb: Toward Dynamic MOO-based VRML

Tom Meyer, David Blair, and D. Brookshire Conner
NSF/ARPA Science and Technology Center for
Computer Graphics and Scientific Visualization,
Brown University Site
twm@cs.brown.edu, artist1@interport.net, and dbc@cs.brown.edu

Abstract

We describe the structure and development of WAXweb, a dynamic MOO-based hypermedia database which is being used as a VRML server. We also discuss the future of 3D MUD-like systems, describing the particular problems of highly-interactive, distributed 3D scenes. So that we can begin to experiment with these areas using commonly-available VRML browsers, this paper describes several extensions for VRML 1.x which will allow for simple dynamic multiuser interactions.

1 Introduction

VRML [BPP95] was intended not only as a static 3D document format for the World-Wide Web, but also as a way to progress towards the development of distributed, collaborative work and play areas. Such ideas were inspired both by MUDs (Multi-User Domains), which are shared text-based virtual spaces on the Internet, and by the development of low-cost computers capable of interactive 3D graphics.

1.1 MUDs

MUDs, or Multi-User Domains, evolved out of multi-player Adventure-style games in the early 80s. These began as hack-and-slash style games, but some of the MUDs began to evolve into more social areas, somewhat reminiscent of chat lines. Although many of the earlier systems relied on hard-coded behaviors, MUD systems began to incorporate internal scripting languages. One particularly flexible server, MOO (MUD Object-Oriented), is now being widely used by the research community to support collaborative work, due to the ease of modifying the environment to support scholarship and sharing information.

The MOO server is distributed by Pavel Curtis as part of his research at Xerox PARC, studying collaborative computer systems [CN93]. Although there continue to be a large number of MOOs solely devoted to socializing, MOO systems have been established at the MIT Media Lab (collaborative environment for media researchers), the University of Virginia (postmodern theorists), CalTech (astronomers), and the Weizmann Institute of Science in Israel (biologists).

1.2 History of WAXweb

WAXweb, at `<http://bug.village.virginia.edu>`, is based on David Blair's film, *WAX or the Discovery of Television Among the Bees*, which was the first independent feature film edited using a nonlinear editing system, and was broadcast over the mbone (multimedia backbone) in 1992. The hypermedia version was initially constructed using Storyspace, and then imported into a MOO-based hypertext authoring system, as a large object-oriented database. We use the MOO to dynamically serve HTML documents from the underlying database, and deliver 3000 WWW pages, containing 7000 pictures, many interleaving layers of semantic indexing, 1 1/2 hours of MPEG video, and the entire audio of the film in English, French, German, and Japanese [MBH94].

2 VRML Implementation

The original film had roughly more than ten minutes of computer-generated imagery; we have taken 250 of the 3D models, originally developed on an Amiga, and imported them as VRML models, to serve as a spatial and contextual index to the multimedia data. All the VRML scenes are generated dynamically, from the underlying object-oriented database.

2.1 Object model

The underlying object model and programming language of the MOO server [Cur93] encourage rapid exploration. It uses a prototype-delegation object model with dynamic inheritance, which we believe to be almost a necessity for any highly-interactive, flexible environment. It incorporates a persistent object store, and new objects and methods can be added while the server is running. Although the language is not compiled, which causes some slowness under extreme server load, we feel that the benefits from rapid prototyping and a cleaner integration of content and behavior far outweigh the potential loss of speed. Additionally, any functions which are called particularly often can be recoded into C and compiled into the server.

Because of this, we were able to create a standard VRML method which transforms the SGML-based representation of the content of each page into a default VRML visualization. Although this is not a perfect transformation, the majority of the pages of the database now can be viewed as either plain text, HTML, or VRML.

2.2 Dynamic/Static optimization

Although it is useful for the database to be dynamically delivered as VRML, delivering all the VRML from inside the MOO led to extremely slow transfer rates. Since the majority of the VRML objects consist of linked pieces of fairly static geometry, we adopted a heterogeneous approach, where individual pieces of geometry (an indexed face set, and its points) are extracted out into sub-files. These files are placed on a separate HTTP server, for speed, while the MOO dynamically assembles a set of WWWInlines pointing to them. This allows the interpreted code to modify the linking behaviors of the scene, as well as the relative transformations of pieces of geometry, without sacrificing too much speed.

2.3 Integration of VRML, HTML, and MOOs

Because every page can be rendered as HTML or as VRML, users can navigate through the multimedia database in HTML or VRML. As an experimental feature, the server can also deliver multi-part documents to a Netscape 1.1 browser. In this situation, the server delivers a single document which encapsulates an HTML and a VRML document; Netscape extracts the subparts and renders the HTML, passing the VRML to a user-defined client.

By using the two browsers side-by-side, we provide for the nearly seamless integration of HTML and VRML. By using a custom Emacs client, users can travel through the MOO-based space and see static HTML and VRML versions of each room that they are visiting, while taking advantage of all the interactivity of the MOO.

3 Research Areas for VRML and MUDs

Of course, *WAXweb* is not the typical VRML-based MUD that people envision when they start thinking about how one would build a 3D, multi-user interactive world. There are a number of reasons why a typical MUD architecture is not appropriate for highly-interactive 3D scenes:

- little or no support for 3D or spatialized operations
- designed for relatively low-speed text throughput
- inability to load interpreted code and compile it as needed
- centralized server model, with a dumb client

For these reasons, we have chosen to explore less-obvious ways of using a MUD, which still take advantage of their multi-participant nature and underlying flexibility. However, each of these problems represents a useful research area for future work on highly-interactive environments.

3.1 Spatialization of MUDs

Very few MUDs have a concept of a 3D coordinate system, and those that support one tend to be oriented toward interstellar combat. For example, the standard MOO server does not support floating-point math operations, and has no concept of vector and matrix math.

Another problem is that most interactive operations are solely oriented toward maintaining container relationships, e.g., "Bill, Sue and a green table are in the room. An open book is on the table." It is of course possible to create a feasible 3D geometry and coordinate transformation for each of the objects in a MUD, but it is nevertheless a difficult task to do well. A better approach would probably involve building a new database from scratch, and would require a subtle combination of semantic and geometric modeling.

3.2 Throughput issues

MUDs are designed for relatively slow text entry and display, at a rate of at most a few hundred characters per minute. 3D scenes not only require more bandwidth to display and update than text-only worlds, but users also generate much more rapid input, and expect near-instantaneous responses to their actions.

Some of this could be dealt with by properly tuning servers for rapid, high-bandwidth interaction, and by using multiple interaction threads. Clients which maintain simple predictive models of the world could also help a great deal in reducing lag times, and reducing the bandwidth demands on the server.

3.3 Dynamic compilation of interpreted code

There are a few modern languages, such as Common Lisp, or Self, for which it is common to prototype interpreted code. As this code is executed more often without being modified, it can either be automatically copiled into a faster form, or the user can choose to compile it.

Without dynamic compilation, the more flexible, interpreted interactive systems will always be substantially slower than the special-cased, optimized systems. However, Self has shown that it is capable of incrementally producing code which can run nearly as fast as compiled C.

3.4 Centralized server model

Most MUDs use a centralized server, with a single copy of the database keeping track of all modifications, and routing all messages between clients. As more users try to use these systems, the bandwidth demands tend to increase linearly if all the users are partitioned into separate rooms, or as $O(n^2)$ if all users are able to see what the others are doing, since all messages must go to all other participants.

Although several distributed database systems exist, they tend not to be optimized for high-bandwidth interactive applications. These systems all target high-reliability applications where updates happen relatively slowly, such as in airline reservation systems. Dealing with object migration and resource replication in an unreliable, high-update environment is a particularly difficult problem.

4 Proposed extensions for VRML 1.x

A system which solved the problems outlined above would probably include mechanisms for defining arbitrary new behaviors, and would also use new transaction protocols, rather than the fairly limited and stateless HTTP. However, such mechanisms will be very difficult to agree upon, and some simple extensions to VRML will allows us to begin to experiment with solving simple versions of those problems.

We would like to propose several extensions for VRML 1.x which would be useful to support preliminary experiments with multi-user interactive environments. Note that these are intended only as extremely preliminary ways of coping with some of the problems addressed above, and will not serve as a solution for large-scale, multi-participant shared virtual worlds.

4.1 Server push of subtrees

At the present time, VRML describes a completely static scene, expressed through a scene graph (a DAG). One of the simpler ways to add dynamic behavior to a scene would be through allowing the server to replace a subtree of the VRML description with a different piece of VRML. This could allow for dynamically changing geometries, interpolating camera positions, or entirely new scenes to be downloaded.

Netscape 1.1 introduced two preliminary extensions to the HTTP protocol which would allow for such behavior [Com95]. In "server push," the server sends a message of MIME type `multipart/x-mixed-replace`, which defines an incoming stream of messages, each of which will replace the previous one. The client holds the socket open and waits for new types of media, until the server or the user breaks the connection.

In "client pull", the server sends an additional field in the HTTP header, which notifies the client to reload this data at a specified time in the future. Although this does not allow a server to send data at a variable update rate, it has the benefit of not requiring that each user constantly take up a valuable socket.

If VRML 1.x browsers could be guaranteed to support either protocol, it would be possible to take a `WWWInline` node and return a dynamic document.

As an example of a very simple dynamic behavior, consider using server push to deliver a *Translate* node which immediately precedes a cube. As the server replaced the translation node with new ones, the cube could appear to animate. Several nodes could be added to the graph to replace that single translation, so new objects could appear and also move around. Since an inlined node can currently contain any valid VRML scene graph, this simple mechanism provides a great deal of the functionality that we will require for MOO-like behaviors.

4.2 User-space objects

We would like to be able to define simple interfaces which remain consistently positioned and oriented relative to the user. The HTML 3.0 specification [Rag95] calls such interfaces “Banners” in 2D, so it might make sense to adopt this terminology, though it is not as appropriate for 3D scenes.

```
Banner {
  Cube {}
}
```

The coordinates in a *Banner* are defined such that the unit square with corners at (-1,-1,0) and (1,1,0) will always fit within the viewing window.

4.3 Multimedia types

Although VRML has support for incorporating text and 2D images into the 3D scene, it does not have methods for including audio and video streams. Being able to handle such input will prove important, not only for environmental effects (bird songs, soundtracks, sunsets, television screens), but also for the eventual incorporation of live audio and video streams, to allow for virtual teleconferencing.

Fast dynamic texture maps are currently possible on either high-end graphics workstations or PCs, and the ubiquity of software video decoders and inexpensive teleconferencing equipment will make this one of the more interesting VRML applications. For video streams which contain synchronized audio data, we will also need to specify the characteristics of the associated audio. Since these media have common attributes, we propose new attribute nodes to control these attributes.

4.3.1 Media attribute nodes

A small set of useful attributes include playback rate, playback volume, whether playback is proceeding or not, and whether playback should be looped or not. We suggest the names `PlaybackRate`, `PlaybackVolume`, `PlaybackRunning`, `PlaybackLooping`, and `PlaybackStartTime`. Each has an optional name, and a field describing appropriate parameters. These attributes are specified as different nodes so that they can be set independently (e.g., turning on all media independent of whether any media are looped or not). A sixth node collects all of these attributes: `PlaybackAttributes`.

The `PlaybackRate` node has one field, `rate`, containing a floating point value used to multiply the native playback rate of media. The default `rate` is thus 1, indicating that media should proceed at their normal rate. A rate of 0.5 halves the playback rate, so that it takes twice as long to display the media in its entirety. A rate of 2 would double the playback rate. Note that a rate of -1 would indicate playing back in the reverse direction.

```
PlaybackRate {
  name      ""      # SFString
  rate      1       # SFFloat
}
```

The `PlaybackVolume` node has one field, `volume`, containing a floating point value used to multiply the volume of any audio media. As with rate, the default `volume` is thus 1, playing the media at normal volume.

```
PlaybackVolume {
  name      ""      # SFString
  volume    1       # SFFloat
}
```

The `PlaybackRunning` node has one field, `on`, containing a `SFBool` used to determine whether audio or video media should play back. A value of `TRUE` (the default) indicates that any sound will be playing and any video will be playing. A value of `FALSE` indicates that any audio or video will not play back.

```
PlaybackRunning {
  name      ""      # SFString
  on        1       # SFBool
}
```

The `PlaybackLooping` node has one field, `looping`, containing a `SFEnum` value which can be one of several values. `ONCE` indicates that a sound or video should play once and then stop. `END_TO_END` indicates that media should play continuously, restarting at the beginning when finished. Finally, `BACK_AND_FORTH` indicates that media should play continuously by reversing playback direction when the end is reached.

```
PlaybackLooping {
  name      ""      # SFString
  looping   ONCE    # SFEnum
}
```

The `PlaybackStartTime` node specifies where in the media playback should begin. It contains two fields, `absolute` and `time`. The field `absolute` is a `SFBool` field that describes whether the `time` field represents absolute or relative media time. `TRUE` (the default) indicates that `time` specifies an absolute time in seconds. `FALSE` indicates that `time` specifies a start time as a fraction of the duration of the media (with 0 being the beginning and 1 being the end). The start time is used when media are first displayed and whenever `PlaybackRunning` becomes `TRUE`.

```
PlaybackStartTime {
  name      ""      # SFString
  absolute  TRUE    # SFBool
  time      0       # SFFloat
}
```

Finally, the `PlaybackAttributes` node collects all of these attributes in one node as a convenience.

```
PlaybackAttributes {
  name      ""      # SFString
  rate      1       # SFFloat
  volume    1       # SFFloat
  on        1       # SFBool
  looping   ONCE    # SFEnum
  absolute  TRUE    # SFBool
  time      0       # SFFloat
}
```

4.3.2 Using WWWInline with other media

It might seem desireable to specify new “geometry” nodes corresponding to new media types. However, a `WWWInline` node is sufficient, except that the specification for VRML 1.0 indicates that the effect of a `WWWInline` that does not refer to VRML is undefined. We would thus like to propose clarifications for the behavior of a `WWWInline`, based on MIME major media types.

The six major types are `text`, `message`, `multipart`, `application`, `image`, `audio`, and `video`. We will now outline ways to present these media in a VRML scene in a manner that strives to integrate the media with the 3D environment. Note that we describe methods of dealing with all the types for the sake of completeness; we feel that the most important major types to implement are image, audio, and video.

- `text` and `message` major types represent text-based content. We suggest that these be handled by displaying them in a window centered at the origin as transformed by the current transformation. As a browser option, this window may be either a screen-aligned window or a 3D window spanning 1×1 square in the xy plane, transformed appropriately.

- `multipart` media represent messages in several parts. We have already discussed Netscape's `x-mixed-replace` minor type. Others should be handled suitably. Note that `alternate` can be implemented using a `Switch`.
- `application` media must, by their nature, be handled on a case-by-case basis. We recommend that, as a suggested implementation, the applications be embedded in the VRML scene (e.g., by texturing output on to a polygon).
- `image` media can of course be used as textures. When used as a `WWWInline`, we suggest that they be textured automatically onto a 1×1 square in the xy plane. They should fit within this square but their aspect ratio should be preserved. This plane should be transformed by the current aspect ratio.
- `audio` media should be played according to the playback attributes described above. In addition, they should be spatially located at the origin, and transformed accordingly. A browser may not be able to perform a true sound spatialization. If this is the case, it is recommended that the volume be modified to account for the distance from the sound's position to the camera (i.e., distant sounds are quieter).
- `video` media should also be played according to the playback attributes. The image component should be displayed in a textured manner as if it were `image` media. The audio component should be treated in a spatialized manner as if it were `audio` media.

4.3.3 Implementation notes

We have implemented a prototype texture-mapped MPEG decoder on a SGI Onyx with Reality Engine 2, and find that it renders in real-time, even with a fairly naive implementation. Since several PC graphics libraries already provide dynamic texture mapping and have robust support for multimedia, it may not be difficult to bring such capabilities to PC- and Mac-based browsers.

In addition, the playback attributes can be implemented in a straightforward fashion using Open Inventor (a common implementation of VRML browsers). Each attribute can be defined as a new Element in the Inventor traversal state. New media can be defined as new Nodes in Inventor, using Inventor utilities and nodes (such as texturing and accumulating transformations) to perform the spatializations and image texturing described above. The `WWWInline` node can then instantiate a suitable node based on the MIME type of a downloaded document.

5 Conclusions

It is currently possible to generate simple, static renderings of MUD-like scenes in VRML, but it will not be possible to have dynamic updates of these scenes without extending VRML in minor ways. We have presented what we feel is a simple way to add a large amount of dynamic, multi-participant behavior to these worlds without excessive modifications to the VRML 1.0 specification.

6 Acknowledgements

Thanks to the Brown University Graphics Group, especially Andries van Dam, John F. Hughes, David Klaphaak, Suzanne Hader, Emma Johnson, Laura Mullen, and Brook Conner. Henry Fuchs and Greg Turk from the University of North Carolina provided impetus for the integration of video in the VRML spec. John Unsworth of the Institute for Advanced Technology in the Humanities at the University of Virginia provided a machine and disk space. Thanks to Mark Pesce, Jan Hardenburgh, and the Open Inventor Group at SGI for their discussions and early bug-testing of WAXweb.

This work was supported in part by grants from NSF, ARPA, NASA, IBM, Taco, Sun, SGI, HP, Microsoft, Xing, OKI, ONR grant N00014-91-J-4052, ARPA order 8225, and the New York State Council for the Arts.

References

[BPP95] Gavin Bell, Anthony Parisi, and Mark Pesce. The Virtual Reality Modeling Language Version 1.0 Specification. `http://vrml.wired.com /vrml.tech/vrml10-3.html`, 1995.

[CN93] Pavel Curtis and David A. Nichols. MUDs Grow Up: Social Virtual Reality in the Real World. In *Proceedings of the Third International Conference on Cyberspace*, 1993. `ftp://parcftp.xerox.com /pub/MOO/papers/MUDsGrowUp.ps`.

[Com95] Netscape Communications. An Exploration of Dynamic Documents. `http://home.netscape.com /assist/net_sites/pushpull.html`, 1995.

[Cur93] Pavel Curtis. LambdaMOO Programmers Manual. `ftp://parcftp.xerox.com /pub/MOO/ProgrammersManual.texinfo_toc.html`, 1993.

[MBH94] Tom Meyer, David Blair, and Suzanne Hader. WAXweb: A MOO-based Hypermedia System for WWW. In *Proceedings of the Second International WWW Conference*, 1994. `http://www.ncsa.uiuc.edu /SDG/IT94/Proceedings/VR/meyer.waxweb /meyer.html`.

[Rag95] Dave Raggett. HyperText Markup Language, Version 3.0. *Internet Draft*, 1995. `http://www.w3.org /hypertext/WWW/MarkUp/html3/CoverPage.html`.