MERL – A MITSUBISHI ELECTRIC RESEARCH LABORATORY
http://www.merl.com

Diamond Park and Spline:

A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability

R. Waters, D. Anderson, J. Barrus,
D. Brogan, M. Casey, S. McKeown,
T. Nitta, I. Sterns, & W. Yerazunis

TR-96-02a November, 1996

Abstract

Diamond Park is a social virtual reality system in which multiple geographically separated users can speak to each other and participate in joint activities. The central theme of the park is cycling. Human visitors to the park are represented by 3D animated avatars and can explore a square mile of 3D terrain. In addition to human visitors, the park hosts a number of computer simulations including tour buses and autonomous animated figures.

Diamond Park is implemented using a software platform called Spline, which makes it easy to build virtual worlds where multiple people interact with each other and computer simulations in a 3D visual and audio environment. Spline performs all the processing necessary to maintain a distributed, modifiable, and extendable model of a virtual world that is shared between the participants.

The initial version of this technical report appeared in January, 1996.

Contents

1 Introduction

Our long term goal is to create social virtual reality systems where people can interact in real time for learning, work and play. In particular, we seek to create virtual worlds featuring: multiple, simultaneous, geographically separated users; multiple computer simulations interacting with the users; spoken interaction between the users; immersion in a 3D visual and audio environment; and comprehensive runtime modifiability and extendability.

As an initial experiment, we created a multi-user virtual world called Diamond Park (see Figure 1), which was first demonstrated at COMDEX in November of 1995. Diamond Park is notable for supporting the real-time interaction of multiple users in a large space with a relatively high level of visual, auditory, and kinesthetic detail. The most important part of a visit to the park is participating in activities with other visitors. We chose to focus primarily on social interaction in Diamond Park, because we believe that good support for social interaction will be the most important feature of social virtual reality applied to any purpose.

In the future, our goal at MERL is to construct a number of different social virtual reality systems in order to explore ways that such systems can support work and education as well as entertainment. To make this possible, we have focused much of our effort over the past three years on creating a flexible platform for building social virtual reality systems called Spline (for Scalable Platform for Large Interactive Network Environments).

Spline supports all the key features of the multi-user interactive environments we seek. As a result, it allows the application writer to focus on creating the content of a particular world without worrying about communication and synchronization between multiple users. We consider it a major validation of Spline that while Diamond Park required the creation of many large graphical models and large amounts of recorded sounds, Diamond Park required very little code over and above what is contained in Spline itself.

In comparison to other tools for supporting 3D virtual worlds, Spline is notable in that it supports both visual and audio interaction, allows runtime extensions of all kinds of data, and is designed to support large numbers of users.

Figure 1: Two visitors riding in Diamond Park.

Figure 2: Bicycle interface to Diamond Park.

Figure 3: Diamond Park's obstacle course.

Figure 4: Diamond Park Velodrome.

Figure 5: Diamond Park's transportation systems.

Figure 6: Interior of Outer Space Building.

Figure 7: The Poet's Grove in Diamond Park.

2 Diamond Park

Diamond Park combines visual, audio, and physical interaction. A square-mile of detailed and varied terrain was created to encourage visitors to spend time exploring. This is coupled with an equally rich audio environment including live sounds communicated between visitors, ambient background sounds and sound effects.

In addition, whole-body physical interaction is provided by means of modified recumbent exercise bicycles with variable pedal resistance (see Figure 2). Visitors on exercise bicycles are represented in the park as animated bicyclists like those in Figure 1. Spline's support for real-time conversation allows visitors to talk with one another as they ride.

The joint requirements of bicycling and social interaction shaped the park's character as an amalgam of a landscape park, a village and a World's Fair. As visitors roam around the park, they can participate in a variety of outdoor and indoor activities that create interesting social situations.

A visit to Diamond Park usually begins at the Orientation Center (seen in the distance in the center of Figure 1). This building was sited to provide a panoramic view of the park's landscape, which includes lakes, buildings and a complex network of paths. The Orientation Center contains a 3D table-top model of the park that shows the park's layout and features at a glance.

Birdsong and gentle wind can be heard in the background at the Orientation Center, helping to create a welcoming atmosphere. First-time visitors often head for the Central Plaza at the foot of the Orientation Center hill. The sounds of jazz and clinking dishes draw them towards the Plaza Cafe, a place to meet and chat with other visitors.

Since Diamond Park is usually experienced while pedaling an exercise bicycle, bicycle-related activities are an important aspect of a visit to the park. One of the activities available to visitors is an obstacle course (Figure 3) laid out on a flat grassy area near the lake. Visitors can also race or do time trials in the Velodrome (seen on the left in Figure 1). The 200-meter indoor track (Figure 4) was designed to Olympic bicycle racing specifications.

Cycling is not the only means of transportation within Diamond Park. Yellow hoverbuses (Figure 5) offer visitors an alternative way to explore the landscape. They provide a tour of the park's central area.

If visitors want to move quickly from one area of the park to another, they can use the obelisk teleportation system. An obelisk is located outside every major building and in various spots in the park's outer regions. (An obelisk appears to the left of the bus in Figure 5.) The 22 obelisks in the park share a small common interior with doors corresponding to each of the obelisks. By riding into an obelisk and then out the door connected to the obelisk nearest a desired destination, a visitor can travel long distances very quickly.

Visitors who enter the Outer Space Building are surprised when they discover that the building's relatively small exterior belies the presence of a vast interior filled with stars, orbiting asteroids and floating astronauts (Figure 6). Near the Outer Space Building is a site with an entirely different character: the Poet's Grove (Figure 7). This is a tranquil outdoor setting where recorded poetry mingles with the sounds of the wind and birds.

In addition to the avatars of human visitors, the park contains several different kinds of computer simulations. A simple example of this is the tour buses that travel around the park (see Figure 5). A more complex example is the asteroids orbiting in the Outer Space Building

(see Figure 6).

From time to time, Diamond Park has contained simulated bicyclists created at Georgia Tech. (Hodgins et al., 1995) following prerecorded paths. It has also contained simulated human figures with complex motions (e.g., pedestrians, an acrobat, and some baseball players) created by Boston Dynamics Inc. (Koechling, Payter & Raibert, 1996). The pedestrians can walk, jog, and run, switching smoothly from one gait to another under real-time computer control.

2.1 Design Considerations

Social virtual reality is just beginning to emerge as an art form. Interest is growing rapidly, but as yet, there are few examples of systems like the ones we seek. In the long run, we believe that the value and success of social virtual reality systems will depend more on the exact details of individual applications than on the basic capabilities of the underlying technology.

When designing Diamond Park, two issues were foremost in our minds. The first was our belief that artists should be involved as central designers from the earliest inception of Diamond Park. The second was the desire to create a balanced environment with whole-body physical interaction, visual detail, audio detail, and real-time performance.

2.1.1 The Bicycle Interface

The exercise bicycle as input device was the initial point of departure for the design of Diamond Park. One reason we made this choice is that we believe that adding whole-body interaction to a virtual world can greatly enhance its interest. Another reason is that we hoped that bicycling would be an appealing basis for non-violent interaction among people of all ages. We have been gratified to discover during various demonstrations that all kinds of people seem to enjoy riding through Diamond Park.

The choice of bicycling as a theme drove the primary choice of activities in Diamond Park, which include riding in a large environment on rugged terrain and a path network, as well as racing in the Velodrome. In addition, it required significant accommodation, e.g., by making every building accessible via ramps.

To remain faithful to bicycling, we chose to have the interface consist solely of things that can be done on a bicycle, i.e., there is no requirement for a keyboard or mouse. Rather, everything happens by riding to one place or another. For example, the obelisk teleportation system is triggered by riding in one obelisk door and then out another.

An interesting design issue was what to use for a visitor's point of view. We chose an over-the-shoulder viewpoint because this allows a visitor to get a comprehensive view even though the field of view is relatively narrow. In particular, it allows visitors to see anyone who may be talking to them, no matter what direction they are looking. In addition, we have been interested to note that many visitors to Diamond Park seem to prefer having an overall view rather than a through-the-eyes view. It would seem best to provide for both options.

To increase the number of people who can use Diamond Park, we included a keyboard-based interface as well as the bicycle interface. In keeping with the bicycling theme, people using the keyboard interface are represented in the park as unicyclists.

2.1.2 The Landscape

When designing the setting of Diamond Park, we tried to create an environment that: (1) would be familiar and inviting for people of all ages and backgrounds, (2) could encompass a

wide range of activities, and (3) would evoke a sense of place and a spirit of community. To do this, we relied on ideas from many sources including landscape design and town planning, see Greenbie (1981) and Wylson & Wylson (1994).

The layout of Diamond Park was inspired by the landscape parks created in the 18th and 19th centuries by Humphrey Repton and Frederick Law Olmsted (Fabos, Milde & Weinmayr, 1968). In these parks, the deliberate planning of views, the mixture of wooded and open space, and a combination of man-made and natural features help draw the visitor through the physical setting. As in formal gardens, a number of meeting places such as the Central Plaza were created in Diamond Park.

The potentially alienating nature of a virtual experience led to the decision to design Diamond Park's buildings as iconic representations of familiar architecture (Venturi, Scott Brown & Izenour, 1977). A range of building styles appear throughout the park, each serving a specific purpose. Major public buildings such as the Orientation Center and Museum are based on neoclassical models. A vernacular Little House, hidden within its grove of trees, provides a contrast to the futuristic "expo style" of the Outer Space Building. A romantic 19th century Folly set on an inaccessible rocky island (shown in the background of Figure 7) hints at the wilder nature of the park's outlying reaches.

The park's textures were hand-painted in a conscious effort to create a slightly nostalgic, familiar place where first-time visitors would feel comfortable. By hand-painting all textures, we had much greater control over their level of detail. In addition, each one could be designed to be reused in a number of contexts, thereby minimizing our use of texture memory.

2.1.3 The Soundscape

A key feature of Diamond Park is that the audio environment is just as carefully designed as the visual environment. For example, the different parts of the park are associated with their own characteristic sounds. The Poet's Grove (see Figure 7) was created to showcase the importance of ambient sound. The grove is quite simple graphically—a grove of trees and a small plaza with benches—but once within this space, visitors enter a contemplative audio environment where poetry emanates mysteriously from the trees.

2.1.4 Tools

The experience of creating the graphic models for Diamond Park highlighted a number of weaknesses in currently available modeling tools. To start with, it was quite difficult to find a tool that would do an adequate job of helping us minimize the number of polygons in the models we were creating.

Another problem stemmed from the sheer size of the models. It was often essential to have multiple people working on the park at the same time. We sometimes had the models divided into as many as 25 separate sections being modified in parallel. It would have been valuable to have much better support than our tools offered for version control and for stitching pieces back together after modification.

A number of problems stemmed from frustrations involved in creating 3D images through an inherently 2D interface. For example, using a 2D mouse to draw objects in 3D is a slow and painful process. In addition, when looking at a shadowless image with no depth cues, it is very difficult to position objects in 3D. The simple modeling task of placing trees throughout the park took more than a week, because it was difficult to tell when the base of a tree was on the

ground, let alone exactly where the tree was in the overall landscape.

A key part of our effort was creating the underlying terrain for Diamond Park. Unfortunately, terrain creation tools for virtual environments are almost non-existent. Most commercially available tools either work primarily with measured data available from the U.S. government or provide limited and unrealistic tools for creating terrain features. The creation of realistic looking hills and valleys for Diamond Park was done by first drawing by hand a high resolution gray-scale image where brighter areas represented higher hills. The data in this image was then used as a height map and converted into terrain polygons by a commercial tool.

To create the park's Velodrome, we wrote a velodrome generator program, so that we could easily experiment by changing such variables as maximum bicycling speed, track length, and infield dimensions. In general, it would be helpful if there were more tools whose basic mode of operation was assembling 3D models from parameterized parts, rather than merely creating polygons one at a time.

2.2 Related Environments

There are only two kinds of multi-user virtual environments that have a long track record. The first of these is the kind of military simulation system typified by SIMNET (Pope, 1989; Calvin, et al., 1993) and the Distributed Interactive Simulation protocol (DIS) (Standard, 1993) that grew out of SIMNET. These systems feature large complex 3D landscapes and large numbers of users (supported by expensive special-purpose hardware). However, they focus on people shooting at each other from a distance rather than talking in groups. In addition, they are intended for staging preplanned exercises, rather than supporting an evolving on-line community.

The second line of multi-user virtual environments began with the multi-user dungeon games (MUDs) that became popular at universities in the 70's and 80's and have matured through systems like LucasFilm's Habitat (Morningstar & Farmer, 1991) and become common in the form of chat rooms on various on-line services. In general, MUDs are primarily text based using at most simple 2D graphics. However, they support large numbers of interacting users. At the forefront of research on MUDs, the Jupiter system (Curtis & Nichols, 1994) adds support for live audio and video, but not 3D graphics. In the course of his work on Jupiter, it was Pavel Curtis who coined the term 'social virtual reality'.

Recently, a number of systems have appeared that extend MUDs into the realm of 3D graphics. These include NTT Software's CyberCampus (`http://www.is.ntts.com`), Sony's Community Place (Honda, Matsuda, Rekimoto & Lea, 1995) (`http://sonypic.com/vs/`), and ImagiNation Network's CyberPark (`http://www.inngames.com/`). In CyberCampus, users interact in a simple 3D world featuring live audio and 2 Hz video. The Community Place browser and CyberPark support multiple people in a shared 3D graphical environment with background sounds but no live audio.

OnLive! Technologies' Traveler browser software (`http://www.onlive.com/`) supports live audio-based conversation and ambient sound in a 3D virtual environment. However, Traveler environments are designed specifically for 3D chat rather than for a variety of social interaction. OnLive's avatars consist of large, floating heads without bodies or limbs, which are adequate for conversation but not for physically-based activities.

Using bicycling as a motif, Diamond Park goes beyond chat by introducing complex interaction with a rich 3D environment. Unlike the systems above, Diamond Park has not been

fielded to a wide audience. Rather, utilizing hardware that can support both a detailed visual and a detailed audio environment, Diamond Park shows what the growth of PC power and network speeds will soon allow to be fielded.

A final line of development relevant to Diamond Park is other virtual environments designed specifically for bicycling. In 1989, the Cyberia team at AutoDesk created a PC-based virtual world for bicycling (see pages 188–189 in Rheingold, 1991). Seated on a stationary bicycle called the HiCycle, a user could explore a varied 3D terrain. Currently, Tectrix (`http://www.tectrix.com/products/VRBike/VR_Bike.html`) and RacerMate (`http://www.wsmith.com/velolinq/racemate/race3.htm`) produce commercial systems of a similar flavor, using PC's and Nintendo game consoles, respectively. These systems provide some support for multi-user interaction by allowing a small number of machines to be connected together as long as they are located in the same room.

3 Spline

Diamond Park was implemented using a software platform called Spline (for Scalable Platform for Large Interactive Network Environments) that provides comprehensive support for the key features of the environments we seek. In particular, Diamond Park was implemented using Spline version 1.5 which was completed in the fall of 1995. We are currently nearing completion of Spline version 3.0, which improves on Spline 1.5 in many ways. However, the basic architecture and approach of Spline 3.0 are the same as Spline 1.5. The following presents the core features that are shared by all versions of Spline.

3.1 The World Model

Above all else, Spline provides a convenient architecture for implementing multi-user interactive environments. This architecture is centered on a *world model* that mediates all interaction. Figure 8 illustrates the application programming model presented by Spline. It shows five applications interacting through the world model.

Spline applications do not communicate directly with each other, but rather only with the world model. This allows applications to be written without thinking about how communication is achieved. An application does exactly the same things when it is interacting with an application running in shared memory on the same machine as it does when interacting with

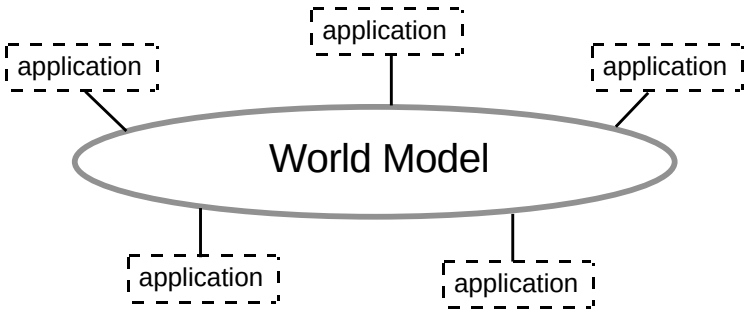

Figure 8: Spline's programming model.

an application connected via the Internet.

Spline's world model is not a scene graph, but rather an object-oriented database that does not consider one kind of content to be any more important than another. In particular, we believe that audio information and autonomous behavior are at least as important as visual information and should not be limited by constraints inherited from visual rendering.

The world model specifies what objects exist in the virtual world, where they are, what they look like, and what sounds they are making. The world model does not contain historical information, but rather just a snapshot of what the virtual world is like at the current moment. As the virtual world changes second by second, the world model changes.

The emphasis in the design of the world model is on the term 'database', not 'object oriented'. The objects have methods associated with them, but by far the dominant operation consists of reading and writing data stored in the instance variables of world model objects. Applications observe the virtual world by retrieving data from the world model.

Applications affect the virtual world by adding, removing, and modifying objects in the world model. To avoid readers/writers conflicts, each object in the world model has one process as its owner and only the owning process can modify it. However, the ownership of an object can be transferred from one process to another.

By itself, Spline does not cause objects to persist over time. An object exists only so long as the application that owns it runs. To have persistent objects, an application must provide persistent processes that accept ownership of these objects. These processes could make use of a persistent file format for Spline objects to provide efficient long term support for infrequently visited parts of a virtual world.

3.2 Scalability

Application programmers using Spline are encouraged to think in terms of Figure 8. However, it would not work well to use a centralized architecture when actually implementing Spline. Rather, Spline operates as shown in Figure 9. To provide low latency interaction with the world model, the world model is replicated so that a copy resides in each application process. Messages sent over the computer network linking the processes are used to propagate changes from one world model copy to another.

A central feature of Spline is that it is designed to be scalable to a large number of users (e.g., thousands) interacting in real time. Two key aspects of Spline support this: providing only approximate equality of local world model copies and dividing the world model into chunks each of which is communicated only to the small group of users that are actually interested in it, rather than to all the users of the world model.

Distributed databases typically require that all local copies of the database must agree exactly on the information in the database. However, this requires object locking and handshaking that is incompatible with real-time interaction if there are more than a very small number of users. In contrast, Spline focuses at all times on speed, providing only approximate equality between world model copies.

The primary way world model copies are only approximately equal is that different users observe things as occurring at slightly different times. We call this a *relativity model* of communication and is not unlike the real world. When you hear sounds from distant sources, you do not hear a sound that is being made now, but rather a sound that was made seconds ago. Therefore, people in different locations do not hear the same things at the same time. How

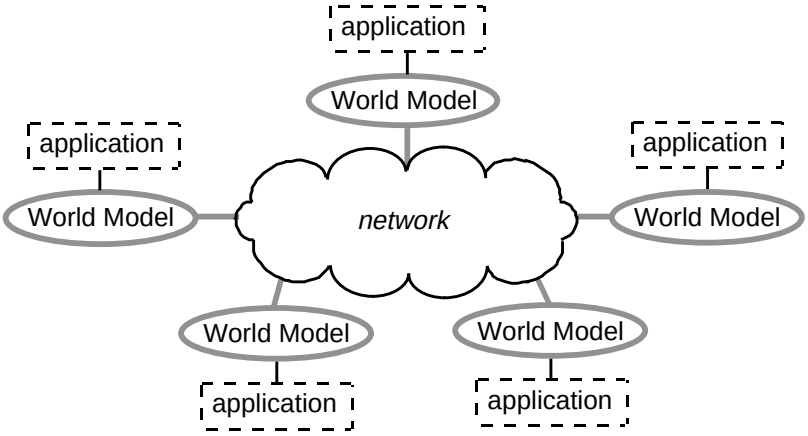

Figure 9: Spline's communication model.

great the differences are depends on how far apart the sound sources and people are.

Similarly, when a Spline process finds out about a world model change, it is not finding out about a change that is happening now, but rather one that happened some time ago. How long ago this is depends on the network 'distance' between the two processes. In general, this distance is not more than a couple hundred milliseconds and does not lead to world model differences that are unduly large.

Having only approximate equality of world model copies allows real-time interaction, but does not of itself prevent the computation required to maintain each local world model copy from growing in proportion to the total number of simultaneous users of a virtual world. To prevent this, Spline breaks the world model up into many small chunks called *locales* (Barrus, Waters & Anderson, 1996) and communicates information about a given locale only to the small number of users that are near enough to that locale to be interested in it. Each locale is associated with a separate multicast communication channel so that processes that are not interested in a locale do not have to expend any processing ignoring it. This allows Spline to scale based solely on the maximum number of users that are gathered in any one locale, rather than on the total number of users in the virtual world.

3.3 Spline Servers

A significant feature of Figure 9 is that it does not contain a central process. To minimize latency, and prevent bottlenecks, the primary communication in Spline is peer-to-peer rather than passing through centralized processes.

In Spline 1.5, 100% of the communication was peer-to-peer. However, in Spline 3.0 centralized processes are used for four key services. The way these processes interact with Spline applications is illustrated in Figure 10.

In the lower right of the figure is the *session manager*. It handles the connection of new users to an on-going Spline session and their eventual disconnection. Its workload is proportional only to the number of users that enter or leave the session in a given minute, not the total number of connected users.

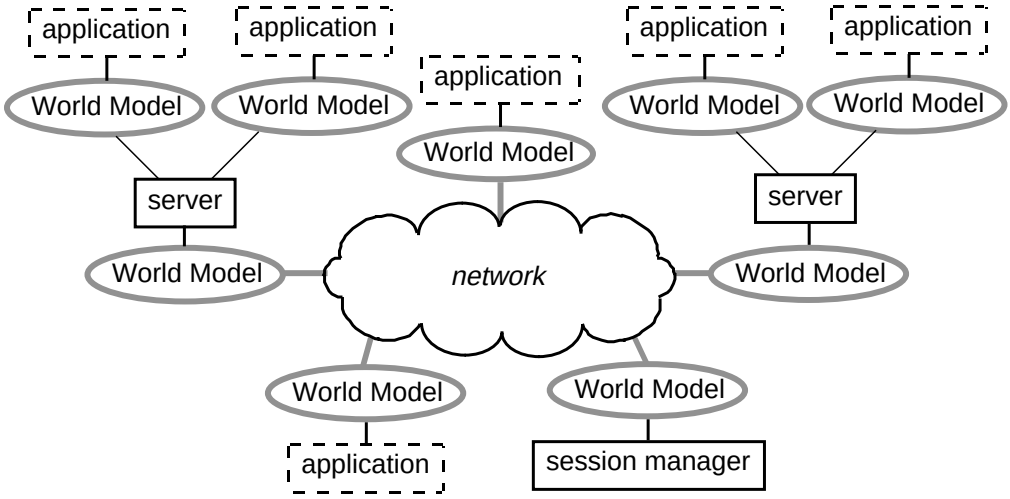

Figure 10: Spline servers.

One or more *Spline servers* are included in a Spline session to provide three important services. One of these services supports users with slow network connections (e.g., modems) that do not allow them to operate as first class Spline peers and two of the services support peer processes.

To support a user with a low speed link, a Spline server intercepts all communication to and from the user. The message traffic to the user is compressed to take maximum advantage of the bandwidth available. As part of this, audio streams are combined and localized before sending them to the user. Spline servers are replicated as needed so that no one server has to support more users than it can handle. Two are shown in Figure 10.

When the locus of attention of a Spline process enters a new locale, it needs to be informed of the objects that exist in the locale. To allow the process to obtain this information very quickly, Spline servers are used to cache the current state of each locale. This state can be rapidly downloaded to processes that become interested. Responsibility for locales is parceled out among a set of Spline servers, which is made large enough that no one server is responsible for a larger piece of the virtual world than it can easily handle.

Because the world model copy in a Spline process only contains information about the objects in the locales the process is attending to, there has to be an explicit mechanism for locating far away objects in the virtual world. This is done by having the Spline servers act as name servers associating names with specialized objects called *beacons*. A process can use this name service to rapidly locate any beacon. As with locales, responsibility for the name space is parceled out among a set of Spline servers, which is made large enough that no one server is responsible for more beacons than it can easily handle.

As shown in Figure 10, Spline 3.0 will utilize a hybrid communication model in which client/server communication is primarily point-to-point but server/server communication is peer-to-peer using multicast. In addition, users with sufficiently fast network connections (e.g., users on corporate intranets) can interact directly with each other and the servers using peer-to-peer multicast.

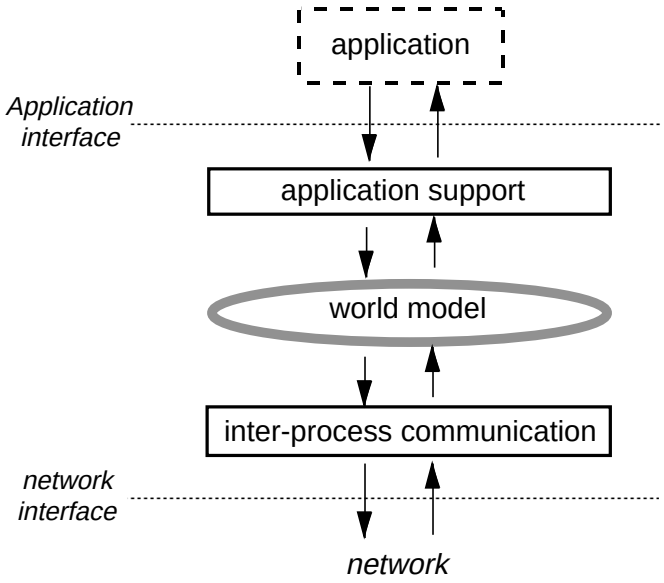

Figure 11: A Spline process.

3.4 A Spline Process

The structure of a single Spline process is shown in Figure 11. The dashed box at the top of the figure shows how an application written using Spline fits into the picture.

The foundation of Spline is the inter-process communication module. It provides all the processing necessary to maintain approximate consistency between the world model copies associated with a group of communicating Splines, sending messages describing changes in the world model caused by the local application and receiving messages from other Spline processes about changes made remotely. The network interface of Spline specifies the format of these messages. Any process that obeys this interface can interoperate with Spline.

The messages sent by Spline are of three kinds, corresponding to three kinds of data in the world model: small rapidly changing objects, large slowly changing objects, and continuous streams of data. An important feature of Spline is that it includes an efficient scheme for synchronizing these different kinds of data.

The most prevalent kind of object in the world model is small things that can change rapidly. For example, an object representing something in the virtual world (e.g., a chair) requires only a small description—i.e., to specify its position and orientation, whether it is contained in some other object, and which appearance should be used when displaying it. Spline allows the features of small objects to be changed very rapidly.

Messages describing changes in small objects are sent using single User Datagram Protocol (UDP) messages. This allows them to be communicated very rapidly. The objects must be small enough so that a message describing one will fit in one UDP message.

Graphic models, recorded sounds, and behaviors are represented using large objects. These objects are identified by Universal Resource Locators (URLs) and communicated using standard World Wide Web protocols. Standard formats are used so that standard tools can be used to

create particular models, sounds, and behaviors. In Spline 3.0 the primary formats will be VRML, WAVE, and Java respectively. There is no limitation on the size of the large objects in Spline, but it must be realized that several seconds can be required to communicate them. Fortunately, since these objects change infrequently, this latency can generally be masked by preloading the objects before they need to be used.

The final kind of object in the world model corresponds to continuous streams of data such as sound captured by a microphone. These streams are communicated in small chunks using UDP messages. At the moment, Spline does not support video streams. When it does, they will be communicated in a similar fashion, but of necessity using larger messages.

A central feature of Spline is that using the various messaging approaches above, every kind of data in the world model can be communicated between Spline processes. Therefore, applications can modify and extend every aspect of a virtual world. Further, while it is often advantageous to prestore data for an application to use (i.e., by delivering it on a CD-ROM) this is not necessary. Once started, an application will fetch everything it needs that has not been prestored.

3.5 The Spline API

Spline's Application Program Interface (API) consists primarily of operations for creating/deleting objects in the world model and reading/writing instance variables in these objects. The application support module contains various facilities that make application writing easier.

The core of Spline is written in ANSI C. However, for the convenience of application writers, high level language interfaces are provided as well. In Spline 1.5 the primary high level API was Scheme. An interactive interface was created by interfacing GNU Scheme to Spline. Having an interactive interface greatly speeds the implementation of applications by supporting more rapid evolution. In Spline 3.0, Java will be the primary high level interface.

Another interesting application support tool consists of special support for the smooth motion of objects. The simplest way for an application to move an object along a trajectory from one position and orientation to another is to repetitively set the object's position and orientation to one spot after another along the trajectory. However, to get smooth appearing motion by this method, the position and orientation must be specified many times per second (i.e., 30 times or more) which leads to high computation and communication costs. Spline provides interpolation-based facilities that allow smooth motion to be achieved while communicating at most a few positions and orientations per second.

Another common problem experienced in virtual worlds is the need to move an avatar or other object along the surface of the ground, while avoiding fixed obstacles. To support this, Spline provides a general terrain following facility (Barrus & Waters, 1996). Given any 3D point, this facility can determine the height of the 'ground' below a query point and whether a collision with a fixed obstacle is occurring in only a few microseconds. This is done by organizing the polygons describing the terrain in a virtual world in a non-uniform depth, axis-aligned quadtree structure where the leaves contain at most a small number of polygons. Inside each leaf, polygon data is stored using both bounding boxes and a 2D binary partitioning tree to limit the number of floating point operations during the final polygon intersection check.

It is expected that world builders will build additional facilities like smooth motion and terrain following. This is easy to do by subclassing Spline's built-in objects.

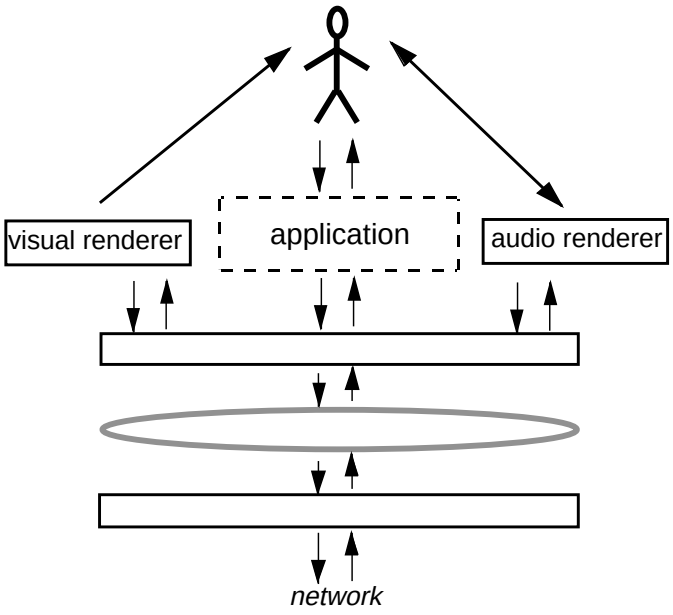

Figure 12: Typical configuration supporting a user.

3.6 Rendering

Figure 12 shows Spline being used to support an application that interacts with a human user. The primary feature of the figure is that three Spline applications are used in this situation. The main application (in the dashed box) presents an interface to the user and interacts with the Spline process in the middle of the figure.

Visual and audio rendering modules are provided as part of Spline; however rather than being tightly integrated with Spline, they are separate applications running on top of Spline. They use the same API as other applications and do not have to be tightly coupled with the main application.

One advantage of the loose coupling of renderers is that the renderers interacting with a person can run in separate Spline processes from the main application. This allows them to operate on separate machines in situations where maximum performance is required. In Spline 1.5 visual and audio rendering was required to run in separate processes. In Spline 3.0 the primary mode of operation is for them to run in the same process with the main application, sharing a single copy of the world model as shown in Figure 12.

The greatest advantage of the loose coupling between rendering and Spline is that Spline is not tied to any one renderer. Rather, Spline is designed to be easily interfaced to almost any renderer. For example, the Spline interface for the visual render used by Diamond Park was implemented in only two weeks. Default renderers are supplied with Spline, but it is expected that demanding applications will switch to renderers that are tuned to the task at hand.

Consider visual rendering as an example. In the world model, objects can have positions, orientations, and appearances. Visual rendering is controlled by visual Point of View (POV) objects that act as cameras. Using an explicit object for this allows Spline to support a wide

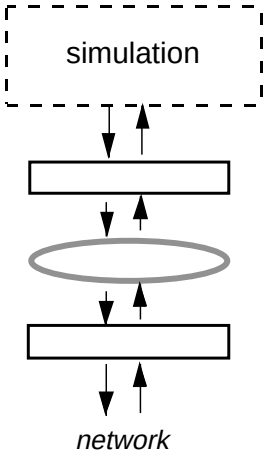

Figure 13: Typical configuration supporting a simulation.

variety of interaction models including through-the-eyes views and more distant views. By moving a visual POV around, a user process controls what is rendered. The visual renderer creates a scene graph by combining the appearances associated with the objects that are near enough to be seen and renders the scene graph from the vantage point specified by the visual POV object. Spline itself does nothing with the graphic models that describe the appearances of objects. The only thing that matters is that the visual renderer being used can load them.

Audio rendering is supported in a similar fashion. The world model contains objects representing sources of sounds. These objects specify the places where the sounds are located. They can either be point sources or diffuse ambient sources. Sound to be played through these objects can be captured live from a microphone or prestored in recorded sound objects. The point of view from which a given user perceives sound is represented by an audio POV object. The audio renderer creates an *audio scene graph* by combining the sounds associated with sound source objects that are near enough to be heard and renderers this scene graph from the vantage point specified by the audio POV object. Spline itself does nothing with sound encodings. The only thing that matters is that the audio renderer being used can decode them.

3.7 Supporting Simulations

From the earliest days of work on Spline, we paid close attention to supporting interaction with computer simulations as well as people. The way this is done is shown in Figure 13.

A simulation connected to Spline operates just like any other application using the same API to interact with the world model. However, no visual or audio rendering is needed, because there is no person to see or hear it. Large powerful computers without support for graphics or sound can be used to manage shared content. Complex simulations, intelligent agents, and large persistent databases can all be directly connected to a virtual world using Spline.

It is important to note that it is a great deal easier to say that a simulation interacts with the world model just like any other application than it is to make it possible for a simulation to effectively interact with the world model. The reason for this is that applications supporting a user have a human being in the loop and simulations do not.

For example, it is typically easy for a person to look at an avatar and determine which way the avatar is facing, based on a rendered image. However, it verges on impossible for a program to tell where the face of an avatar is by looking at a list of polygons. To deal with this kind of problem, the object hierarchy used in Spline's world model has been designed to make information such as which way an avatar is facing easily accessible to programs. Specifically, Spline's world model contains an object class that is used to identify avatars as opposed to other kinds of objects and is designed so that an application can easily determine which direction is up for a given object and which way the object faces.

3.8 Hardware/Software Requirements

The primary operating environment for Spline 1.5 was IRIX on SGI machines. However, the core modules shown in Figure 11 are easily portable and were tested under HPUX on HP machines. In contrast, the renderers for Spline 1.5 were strongly tied to SGI machines.

The visual renderer for Spline 1.5 was implemented using the Performer toolkit from SGI. Specifically, it was implemented by modifying the sample Performer application called Perfly so that the scene graph to be displayed was obtained from Spline's world model. The audio renderer for Spline 1.5 used standard algorithms to create stereo sound for the user to hear and to capture what the user says into a microphone. Volume attenuation was used to indicate distance and stereo panning was used to indicate direction. Although much of the audio renderer for Spline 1.5 is basically machine independent, the lower levels of the renderer are closely tied to the particular sound board API used on SGI machines.

The primary operating environment for Spline 3.0 will be Windows 95 on Pentium-based PCs. The final choice of default renderers for Spline 3.0 has not yet been made. Unfortunately, it does not appear that any choice today will support applications with graphic models as complex as Diamond Park on PCs. However, given the rapid pace of development of graphics accelerator boards, this should be possible soon.

3.9 Related Platforms

To achieve scalability to large numbers of users, Spline adopts the basic approach pioneered by SIMNET (Calvin, et al., 1993) and DIS (Standard, 1993) namely the realization that achieving real-time performance in a scalable distributed system requires giving up on having a completely consistent view of the world model throughout the entire system. In Spline 1.5 we also borrowed a number of architectural ideas from SIMNET and DIS, particularly the use of messages with complete object state (rather than incremental updates) as a means of coping with the unreliability of UDP, the use of keep-alives as a way to reduce the impact of lost messages and for informing late comers of the current state of the simulation, peer-to-peer messaging with no central services for scalability and fault tolerance, and dead-reckoning to reduce the number of messages about object motion.

However, Spline goes beyond DIS in five important ways: (1) Spline is a complete platform providing visual and audio rendering modules not just a communication standard, (2) much greater emphasis is placed on communicating audio information, (3) every kind of data in the world model can be transmitted between Spline processes at run time, (4) the ownership of objects can be transferred from one process to another, and (5) the world model is broken up into locales that are communicated separately. Spline shares this last feature with NPSNET (Macedonia et al., 1994; 1995) (`http://www-npsnet.cs.nps.navy.mil/npsnet/`) but as dis-

cussed in (Barrus, Waters & Anderson, 1996), Spline's 'locales' are more flexible than the 'areas of interest' in NPSNET.

In Spline 3.0 we have diverged further from DIS. We have adopted a different strategy for object state messages that is built on the Scalable Reliable Multicast protocol (Floyd, Jacobson & Weinmayr). Being able to depend on the eventual delivery of certain kinds of messages has permitted us to eliminate keep-alives, and has simplified robust deletion of objects from the world model and transfer of ownership. By arranging for in-order arrival of messages on a per-object basis we are also able to use differential messages that encode changes to objects rather than sending full object descriptions in each message. We have also introduced a scalable set of servers to manage shared resources (see Figure 10).

Much recent work on shared 3D spaces has focused on the World Wide Web (WWW) using the Virtual Reality Modeling Language (VRML). For the most part, these systems support only fixed background scenes, with limited sharing of avatars. Further, most currently available commercial multi-user platforms rely on a single, central server, which puts a limit on the scalability of the system and increases latency.

The systems that are most related to Spline are ones that have experimented with one or more of the same basic techniques for achieving greater scalability—namely, relaxing consistency, avoiding central servers, partitioning the virtual world, and communicating behaviors rather than state changes.

Like Spline, DIVE (Carlsson & Hagsand, 1993; Hagsand, 1996) (`http://www.sics.se/dce/dive/dive.html`) avoids central servers, and relies instead on peer-to-peer messaging. DIVE's simulations are partitioned into separate virtual worlds, each with its own world database and multicast communications. A process may only participate in one world at a time, but may leave and enter new groups dynamically. If it is the first to enter a world, a process initializes the world state from a file. As the simulation progresses, each participating process maintains a complete copy of the world model using reliable multicast protocols and distributed locks to keep the world up-to-date and consistent. Processes joining later are informed of the current world state by one of the processes that is already participating in that world.

Community Place (Lea, Honda & Matsuda, in press) has adopted a hybrid client-server/peer-to-peer approach where servers partition the world spatially and establish multicast groups to support those partitions. They have been studying how to apply different models of distributed consistency to distributed virtual environments.

In BrickNet (Singh, et al., 1994) servers manage the communication between clients while also acting as object request brokers. Clients "own" their objects, but may "lease" some of their capabilities to other clients. Depending on the circumstances, leasing may be read-only, or may allow modification. These object sharing operations are mediated by the BrickNet server. Both BrickNet (Singh et al, 1995) and the Environment Manager built on the MR Toolkit (Wang, Green & Shaw 1995) (`http://web.cs.ualberta.ca/~graphics/MRToolkit.html`) share program code to build structure and execute object behaviors in virtual worlds. Like BrickNet, WAVES (Kazman, 1996) and RING (Funkhouser, 1995) overcome some of the limitations of a central server by incorporating multiple communicating servers.

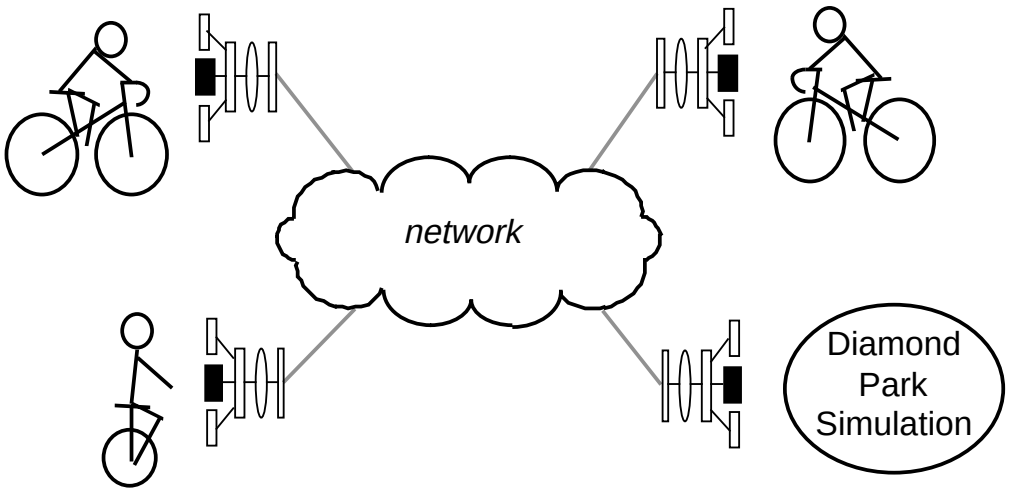

Figure 14: Three visitors in Diamond Park.

4 Diamond Park & Spline

In addition to being an experiment in the creation of a social virtual reality system, Diamond Park was an experiment in the use of Spline. In particular, it tests the claim that Spline makes it easy to create such systems.

4.1 Diamond Park Implementation

Figure 14 illustrates a setup similar to the one that was used to demonstrate Diamond Park at COMDEX in November, 1995. The setup features Spline supporting three visitors: two on bicycles (see Figure 2) and one on a unicycle. The Spline process in the upper right supports the Diamond Park setting itself—i.e., the landscape, the buildings, the background sounds, and the automated tour buses.

The outline boxes in Figure 14 correspond to standard parts of Spline. The solid boxes correspond to code specifically written for Diamond Park. Most of the features of the park (e.g., visitors seeing and talking to each other) are directly supported by Spline, with only a small amount of code being particular to the park. For instance, the specific features of the park setting (e.g., the motion of the tour buses) are implemented using 600 lines of Scheme code, which took only a couple of weeks to write.

The only other code that had to be written specifically for Diamond Park was support for interaction with the specific interface devices used with the park. For example, 3,000 lines of C code written in 3 weeks were required to implement a device driver to communicate with the mechanical bicycling device (see Figure 2) and a mathematical simulation that computes how the bicycle should move and what pedal resistance visitors should feel as they pedal around the park. The unicycle interface is basically similar, but was somewhat easier to implement, since it merely takes mouse and keyboard input rather than interacting with special hardware.

All in all, the code for Diamond Park consists of about 6,000 lines created in 3 man-months. The version of Spline used to support Diamond Park (Spline 1.5) consists of about 25,000 lines

of C code and required 4 man-years of design and implementation.

Spline greatly reduced the code needed to support Diamond Park; however, there were other areas where large amounts of effort were still required. In particular, designing what should happen in Diamond Park took approximately a man-year of effort and constructing the 3D visual models consumed another 8 months. Designing the soundscape and creating the appropriate prerecorded sounds required a few months of effort.

The concept of locales in Spline (Barrus, Waters & Anderson, 1996) had a fundamental impact on the design of Diamond Park in two ways. First, by partitioning the park into sections that are handled separately, locales allow the park to be much more complex than it otherwise could be. In particular, the insides of complex buildings such as the Velodrome and Outer Space Building are placed in separate locales from each other and the outside landscape. This allows the use of more complex 3D models, because no two of these models ever have to be rendered at the same time. Second, locales support a number of interesting effects in the park such as allowing the interior of the Outer Space Building to be much bigger than its exterior and supporting rapid travel from one place to another in the park through the obelisks.

A final point is that the basic architectural approach of Spline, which uses the world model database to mediate all the interaction between various parts of an application, proved very useful when implementing Diamond Park. Ten people worked on the park. For the most part, they worked in isolation, debugging their code in single-user mode by testing that it correctly interacted with the Spline world model. In a final system integration phase that lasted only a week, these pieces were put together to form the full Diamond Park experience. Except for some minor teething problems stemming from the fact that the integration caused both the world model and the message traffic between Spline processes to grow to ten times the size we had ever experimented with before, integration went without incident.

4.2 Performance

Returning again to Figure 14, it is interesting to consider the amount of computation and communication required when supporting Diamond Park with Spline. When demonstrating the park we use SGI machines connected by an Ethernet local area network (LAN).

In the situation in Figure 14, the world model of Diamond Park as a whole contains approximately 600 objects with only 100-500 objects represented in any one process at a given moment. (The entire landscape of the park is only one object.) The graphic models associated with these objects contain some 55,000 polygons, but typically only about 6,000–12,000 polygons have to be rendered for a user at any given moment. Depending on where they are, the users in Figure 14 hear 3–6 sound sources localized and then mixed together.

The most costly part of supporting Diamond Park is the visual rendering of the 3D graphic models. However, because Spline's audio rendering takes place purely in software, it is also costly. Further, to minimize latency while preventing sound drop outs, sound rendering must be given much higher priority than visual rendering. The Diamond Park application itself and the overhead introduced by Spline runs a distant third in computational demands.

The performance obtained with Diamond Park depends on exactly what machines are used to support the individual users. We often support a single user by running all three Spline activities (visual renderer, audio renderer, and application) on a single-processor SGI Reality Station. In this situation, we achieve frame rates of 6–12 Hz depending on how much of the park is in view. Offloading sound rendering to a second machine improves visual rendering

performance to 7.5–15 Hz. When creating a video tape of Diamond Park, we used a four-processor Onyx for visual rendering and offloaded all other processing to other machines. In that situation, we achieved frame rates of 20–60 Hz.

A key question for Spline in general is exactly how much network bandwidth is required per user. However, because of our use of an Ethernet LAN, network usage was of little concern to the Diamond Park demonstration. As a result, we paid very little attention to minimizing bandwidth when implementing Spline 1.5, and therefore data about the network usage of Spline 1.5 is of little value. For example, sound streams were communicated using 16-bit linear samples at 16,000 samples per second. This amounts to 256 kilobits per second per user. Using readily available encodings the sound bandwidth could easily be reduced by more than an order of magnitude. We have paid a great deal more attention to bandwidth while implementing Spline 3.0. We will run experiments testing the bandwidth requirements and scalability of Spline 3.0 as soon as the implementation of Spline 3.0 has been completed.

5 Future Directions

This paper presents Diamond Park and Spline 1.5, the software platform Diamond Park is built on. This work was completed in November 1995. Through 1996, the focus of research on social virtual reality at MERL has been on the creation of an improved software platform called Spline 3.0. We expect that Spline 3.0 will be completed in early 1997.

There are three key differences between Spline 3.0 and Spline 1.5. First, Spline 3.0 has been redesigned in a host of detailed ways to take advantage of the lessons learned by building Diamond Park. The basic skeletal structure of the system is the same, but essentially every part of the API has been improved in one way or another.

Second, Spline 3.0 has been reimplemented from the ground up with Windows 95 on PCs as the primary platform. We intend that like Spline 1.5, Spline 3.0 will run on SGI machines; however, we have focused on PCs to create a system that can be very widely used.

Third, Spline 3.0 will be much better integrated with other systems relevant to the development of distributed virtual environments. Spline 3.0 will use VRML as its principal 3D modeling file format. In contrast, the models for Diamond Park were created using tools with open but non-standard formats. Spline 3.0 will use Java as its principal API language. In contrast, Spline 1.5 used Scheme as its principal API language. Spline 3.0 will support operation over the Internet as it exists today. Spline 1.5 has been used almost exclusively on local area networks.

In 1997, the focus at MERL will shift away from implementing Spline to using it. We plan to do this for two reasons. On one hand, it would be unwise to continue modifying Spline without evaluating the benefits of the modifications made to date. On the other hand, we at MERL are interested in returning to an investigation of novel applications.

In this regard, it is worthy of note that Spline 1.5 has been used to support two other significant applications in addition to Diamond Park. As described in (Walker & Torres, 1995), Spline 1.5 forms the basis for a multi-user drama system called VIVA (for Virtual Interaction with Virtual Actors). In VIVA human users perform some of the roles in a play and a computer simulation performs the rest. The human-controlled and computer-controlled characters interact on a simple 2.5D stage. Because the script the human users are following is known in advance, it is easy for the computer simulations to track what the human users are saying

using speech recognition and say their own lines using speech generation or prerecorded sound.

Another application of Spline 1.5 was the implementation of a networked music environment called Network Rave created by Michael Casey. In this application, a 3D environment contains several parts of a piece of commercially produced dance music. Users fly through objects in the environment to cause changes in the piece. Since the environment is networked, users hear the changes that other users make as well as their own contributions. Network Rave utilizes the full audio capability of Spline, including live speech streams, ten simultaneous high-quality (32kHz) sample-synchronized audio loops, and control over the speaker position and volume for each loop in response to user actions. By switching on different pre-composed audio loops, users create and mix the music themselves. However, because they are combining components in carefully designed ways, the result always sounds coherent and synchronized.

The first experiment we intend to perform with Spline 3.0 is detailed scalability testing. To date, our use of Spline has involved at most about a dozen simultaneously interacting processes. We plan to create an artificial application that is parameterized so that a variety of different communication loads can be simulated. Using this, we will experiment with hundreds of communicating Spline processes and if we can obtain the use of a sufficiently large number of computers, thousands of processes.

At MERL, we are particularly interested in education as an application for social virtual reality. For instance, we would like to create an environment for foreign language practice. Such an environment would feature human users talking to each other and to intelligent simulations capable of simple conversations in the foreign language. An interesting first step in this direction might be to use a VIVA-like system in conjunction with foreign language plays.

In addition to using Spline within MERL, we intend to make Spline 3.0 available free of charge for research purposes soon after the system is completed. For rapid progress to be possible, many experiments in social virtual reality must be pursued in parallel.

Acknowledgments

Many people contributed to Diamond Park and Spline. Barrus, McKeown, and Sterns are the principal architects and implementors of Diamond Park. Additional major contributions were made by Anderson, Brogan, Casey, Jessica Hodgins, Nitta, Waters, and Yerazunis. Altitude Inc. constructed the bicycle interfaces used with Diamond Park by modifying equipment supplied by CyberGear inc. The motions of some of the animated figures used in Diamond Park were created by Boston Dynamics inc.

Waters and Anderson are the principal architects and implementors of Spline. Additional major contributions were made by Barrus, Joe Marks, David Marmor, Casey, and Yerazunis.

References

Barrus, J.W., Waters, R.C., & Anderson, D.B. (1996). Locales: Supporting large multiuser virtual environments. *IEEE Computer Graphics and Applications*, 16(6):50–57.

Barrus, J.W., & Waters, R.C. (1996). *QOTA: A fast, multi-purpose algorithm for terrain following in virtual environments*, (Report No. TR 96-17). Cambridge MA: MERL - A Mitsubishi Electric Research Laboratory.

Calvin, J., et al. (1993). The SIMNET virtual world architecture. In *Proceedings of the IEEE Virtual Reality Annual International Symposium* (pp 450–455). Los Alamitos CA: IEEE Computer Society Press.

Carlsson, C., & Hagsand, O. (1993). DIVE—A multi-user virtual reality system. In *Proceedings of the IEEE Virtual Reality Annual International Symposium* (pp 394–400). Los Alamitos CA: IEEE Computer Society Press.

Curtis, P., & Nichols, D.A. (1994). MUDs grow up: Social virtual reality in the real world. In *Proceedings of the 1994 IEEE Computer Conference* (pp. 193–200). Los Alamitos CA: IEEE Computer Society Press.

Fabos, J., Milde, G., & Weinmayr, V.M. (1968). *Frederick Law Olmsted, Sr: Founder of landscape architecture in America*. Amherst MA: Univ. of MA Press.

Floyd, A., Jacobson, V., & McCanne, S. (1995). A reliable multicast framework for light-weight sessions and application level framing. *ACM SIGCOMM*.

Funkhouser, T.A. (1995). RING: A client-server system for multi-user virtual environments. *ACM SIGGRAPH* (Special Issue on 1995 Symposium on Interactive 3D Graphics). pp. 85-92.

Greenbie, B.B. (1981). *Spaces: Dimensions of the human landscape*. New Haven CN: Yale Univ. Press.

Hagsand, O. (1996). Interactive multi-user virtual environments in the DIVE system. *IEEE MultiMedia*, 3(1):30–39.

Hodgins, J.K., Wooten, W.L., Brogan, D.C., & O'Brien, J.F. (1995). Animated human athletics. In *Proceedings of SIGGRAPH 95* (pp 71–78). NY NY: ACM Press.

Honda Y., Matsuda, K., Rekimoto, J., & Lea, R. (1995). Virtual society: Extending the WWW to support a multi-user interactive shared 3D environment. In *Proceedings of the first annual Symposium on the Virtual Reality Modeling Language* (pp. 109–116). New York NY: ACM Press.

Kazman, R. (1996). Load balancing, latency management and separation of concerns in a distributed virtual world. In Zomaya, A. (Ed.). *Parallel Computations - Paradigms and Applications*. Van Nostrand.

Koechling J., Playter R., & Raibert M., (1996). *Interactive humans for virtual environments*, (Report No. BDI-1996-1). Cambridge MA: Boston Dynamics Inc.

Lea, R., Honda, Y., & Matsuda, K. (In press). Virtual society: Collaboration in 3D spaces on the internet. (`http://www.csl.sony.co.jp/person/rodger/CSCW/cscw.html`). To appear in *CSCW Journal* (special issue on CSCW and the WWW).

Macedonia, M., Pratt, D., & Zyda, M. (1994). NPSNET: A network software architecture for large scale virtual environments. *Presence*, 3(4):265-287. Cambridge MA: MIT Press.

Macedonia, M., Zyda, M., Pratt, D., Brutzman, D., & Barham, P. (1995). Exploiting reality with multicast groups. *IEEE Computer Graphics and Applications*, 15(5):38-45. Los Alamitos CA: IEEE Computer Society Press.

Morningstar, C., & Farmer, F.R. (1991). The lessons of LucasFilms' Habitat. In M. Benedikt (Ed.). *Cyberspace: First steps.* (pp. 273-301). Cambridge MA: MIT Press.

Pope, A. (1989). *The SIMNET network and protocols*, (Report No. 7102). Cambridge MA: BBN systems and technologies.

Rheingold, H. (1991) *Virtual Reality*. New York NY: Simon and Schuster.

Singh, G., Serra, L., Png, W., & Ng, H. (1994) Bricknet: A software toolkit for networked virtual worlds. *Presence*, 3(1):19–34. Cambridge MA: MIT Press.

Singh, G., et al. (1995). BrickNet: Sharing object behaviors on the net. In *Proceedings of the IEEE Virtual Reality Annual International Symposium* (pp 19–25). Los Alamitos CA: IEEE Computer Society Press.

Standard for information technology, protocols for distributed interactive simulation. (DIS ANSI/IEEE standard 1278-1993). (1993). American National Standards Institute.

Venturi R., Scott Brown, D., & Izenour, S. (1977). *Learning from Las Vegas: The forgotten symbolism of architectural form.* Cambridge MA: MIT Press.

Walker, M.A., & Torres, O.E. (1995). Linguistic variation and character in the VIVA virtual theater. In G. Ball (Chair) *Lifelike Computer Characters Conference.*

Wang, Q., Green, M., & Shaw, C. (1995). EM - An environment manager for building networked virtual environments. In *Proceedings of the IEEE Virtual Reality Annual International Symposium* (pp 11–18). Los Alamitos CA: IEEE Computer Society Press.

Wylson, A., & Wylson, P. (1994). *Theme parks, leisure centres, zoos and aquaria.* Longman.