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

Locales and Beacons:
Efficient and Precise Support For Large Multi-User Virtual Environments

John W. Barrus
Richard C. Waters
David B. Anderson

TR-95-16a August, 1996

Abstract

There is a natural desire to make multi-user virtual environments large in spatial extent, in numbers of objects, and in numbers of users interacting with the environment. However, doing this brings up several problems: efficiently managing the flow of large amounts of data between large numbers of users, representing precise position and velocity information about objects that are arrayed across a large volume of space, and allowing designers to create parts of a virtual environment separately and combine them together later. Locales are an efficient method for solving these problems by breaking up a virtual world into chunks that can be described and communicated independently.

While having many benefits, locales introduce a problem: finding something when you do not know what locale it is in. This is solved by the companion concept of beacons, which makes it possible to find something no matter where it is.

The initial version of this technical report appeared in November, 1995.
With the discussion of beacons removed, this revised version will appear as
"Locales: Supporting Large Multiuser Virtual Environments,"
IEEE Computer Graphics and Applications, 16(6):50–57, November 1996.

Contents

1 Introduction

There is a natural desire to make multi-user virtual environments large in spatial extent, large in numbers of objects, and large in numbers of users interacting with the environment. However, doing this brings up several problems: efficiently managing the flow of large amounts of data between large numbers of users, representing precise position and velocity information about objects that are arrayed across a large volume of space, and allowing designers to create parts of a virtual environment separately and combine them together later.

The concept of *locales* is based on the idea that while a virtual world may be very large, most of what can be observed by a single user at a given moment is nevertheless local in nature. That is to say, one would expect a large virtual world to be large primarily because, like a city, it combines a large number of relatively small, localized activities (e.g., a conversation involving a few people here and a simulation involving several objects there) rather than because it contains individual activities that are very large and complex.

Locales divide a virtual world into chunks that can be processed separately. It is important to realize that this division is purely an implementation issue; it is not apparent to the user. A user sees several locales at once, typically including the locale containing the user's point of view and the locales that are the neighbors of this locale. The user does not see any seams between the locales nor any abrupt changes as the point of view moves from one locale to another, thereby changing the neighborhood set.

This differs from VRML on the web and most virtual environments where it is possible to jump from one model/environment to another, but there is no support for simultaneously viewing models from different sources. The purpose of locales is to allow virtual worlds to be combined so that you can see and interact with several at once and can move smoothly from one to another.

The way locales divide up a virtual world is based on three key features, each of which provides important leverage on the problem of bigness.

Separate communication channels - Each locale is associated with a separate set of multicast addresses.
Local coordinate systems - Each locale has its own coordinate system.
Arbitrary geometry and relationships - The shape, size, and orientation of locales can be chosen arbitrarily. It is expected that they will be irregular in shape and size, conforming to the natural boundaries of the virtual world. Further, the relationships between neighboring locales are stated locally.

These features confer five key benefits:

Efficient communication - Using locale-specific communication channels allows a process to attend to what is happening in some locales while completely ignoring others.
Efficient culling - By designating only a small part of the virtual world as relevant, a locale neighborhood serves as a highly efficient culling mechanism.
Precise positions - Since each locale has its own coordinate system, one always has high positional precision in whatever locale is the current focus of attention, even if the virtual world as a whole is very large.

Easy combination of separately design pieces - Rather than being oriented toward cutting up an existing world, locales are intended to be a basis for combining separately designed subworlds into a larger world. The fact that the relationships between locales are local in nature and do not involve any global coordinate system facilitates the combination of pieces designed by different people into a large virtual world.

Interesting effects - The local nature of the relationships between locales also makes it possible to support a variety of interesting effects.

In addition to locales, Spline incorporates a companion concept called beacons, which support a content-addressable communication strategy. Beacons allow a user process to select what objects to be informed about based on tags rather than on locations. This is a useful complement to the content-addressabable communication provided by locales.

Locales were developed as part of a system called Spline (for Scalable Platform for Large Interactive Networked Environments). The next two sections briefly present Spline and the details of what a locale is. This is followed by an in-depth discussion of the five benefits of locales listed above. The paper concludes with a discussion of how locales are used in a virtual world called Diamond Park, which was implemented using Spline.

2 Spline

The Spline platform provides a convenient architecture for implementing multi-user interactive environments that is based on a shared world model. The world model is an object base containing information about everything in a virtual world—where things are, what they look like, what sounds they are making, etc. Applications interact with each other by making changes in the world model and observing changes made by other applications.

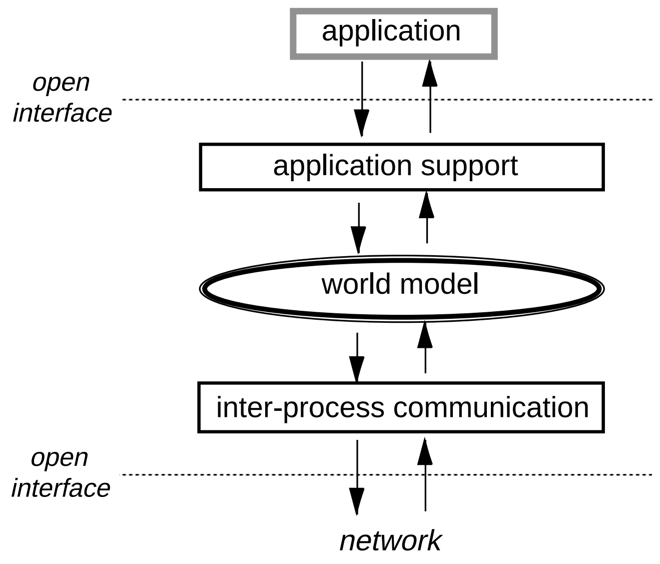

Figure 1: A Spline process.

Name	– Network-wide unique ID of thing X.
Parent	– Thing containing X, if any.
Transform	– Transformation matrix relating X to its parent.
Appearance	– 3D graphic model, if any, of X.

Figure 2: Key instance variables of a thing relevant to locales.

To allow rapid interaction between individual applications and the world model, Spline distributes the world model, maintaining a partial copy in each Spline process. Each copy contains the parts of the world model that are sufficiently near to the corresponding process's focus of attention. To achieve sufficiently low interaction latency, the copies are maintained approximately, but not exactly, consistent.

The structure of a Spline process is shown in Figure 1. The inter-process communication module sends out multicast messages describing objects in the world model and receives messages from other Spline processes about changes made remotely. As in SIMNET [1], each message describes the current state of a world model object and is used by the processes that receive it to update their world model copies. When hardware support for multicast communication is not available (e.g., over most wide area networks) tunnelling processes using unicast UDP communication can be used to simulate multicast communication.

Spline's Application Program Interface (API) consists primarily of operations for creating/deleting objects in the world model, reading/writing instance variables in objects, and detect changes made by other applications. The application support module contains various tools that facilitate interaction between an application and the local world model copy.

The most prevalent type of object in Spline's world model is called a *thing*. Figure 2 shows the most important instance variables of a thing.

Hierarchical relationships between things are represented by making one thing the parent of another. Following standard graphics practice, the position and orientation of a thing with respect to its parent are represented by 4x4 transformation matrices (transforms for short) specifying x-y-z positions, orientations, scaling, etc. using 32-bit floating point numbers. For example, one might make the torso of a robot be the parent of the robot's head, with the transform of the head specifying the position of the head in the coordinate system of the torso.

A Spline process joins into a group of communicating Spline processes by contacting a centralized Spline *session server* managing the group. Among other things, the session server assigns blocks of unique names to Spline processes, which the processes use to identify the objects they create.

3 Locales

Locales are the central organizing principle of the Spline world model. Every thing X is required to be contained in exactly one locale, either directly because X's parent P is a locale or indirectly because P is contained in a locale. (A locale that does not have a parent is deemed to contain itself.)

Figure 3 summarizes the data stored in a locale object. Like a thing, a locale has a parent. The boundary of a locale specifies the 3D volume occupied by the locale. There are many ways

Name – Network-wide unique ID of locale L.
Parent – Thing containing L, if any.
Boundary – Description of the 3D volume corresponding to L.
Addresses – Multicast addresses associated with L.
Neighbors – List of structures describing for each neighbor N:
    Neighbor name – Network-wide unique ID of N.
    Import transform – Transformation matrix relating N to L.
    Export transform – Transformation matrix relating L to N.

Figure 3: Key instance variables of a locale.

this could be represented, such as BSP trees. Spline uses a specialized representation [2] that is optimized for the very rapid determination of which parts of the boundary are immediately above and below a given reference point. This is used both for determining whether an object is within a a given locale and for terrain following by objects that are moving within a locale.

The addresses of a locale are the multicast addresses used when sending messages about the locale and the things contained in it. A set of addresses is used because different kinds of data (e.g., audio data as opposed to visual data) are communicated using different addresses. Multicast addresses are assigned by the Spline session server when a Spline process creates a new locale.

The world model is typically broken up into many locales. For instance, a virtual building might be broken up into locales corresponding to its rooms, corridors, and courtyards. If it is possible to see, hear, or go from a locale L_1 to another locale L_2, then L_2 is made a neighbor of L_1. For instance, consider the floor plan shown in Figure 4. The locale for room R needs to have as neighbors the locales for room R', hall H, and the outdoors O. Typically, which pairs of locales need to be made neighbors can be determined by statically computing visibility using methods like those of Teller [3].

A locale object L specifies which other locales are its neighbors and the exact geometric relationship between L and each neighbor N. The import transformation I specifies how to convert a transform T relative to the coordinate system of N into a transform T' relative to the coordinate system of L (i.e., $T' = IT$). This is used when one wants to know the position of something in N relative to L. The export transformation E is typically the inverse of the import transformation. It specifies how to convert a transform relative to the coordinate system

Figure 4: A simple example of locales.

of L into a transform relative to the coordinate system of N.

Consider again locale R in Figure 4. The transformations relating R and its neighbors R', H, and O might specify that the origin of R' is 3 meters north of R, the origin of H is 1 meters east and 3 meters south, and the origin of O is 50 meters east. The boundary of R might be a rectangular solid 3 by 3 by 2 meters high.

If a locale L has a parent, then this induces a neighbor relationship between L and the locale L' containing the parent of L. The transform T specifying the position of the parent of L with respect to L' is the export transform from L to L'. The import transform is T^{-1}.

Locales with parents are used when it is desirable to move one locale with respect to another. For example, you might put a car in its own locale and then move the car's locale from place to place through the locales corresponding to a virtual town.

3.1 Moving From One Locale To Another

As a thing X moves, it can eventually move outside the boundary of the locale L it is in. When this is the case, X needs to be transferred to the appropriate neighboring locale. This is done by changing the parent of X to the new locale N and computing a new transform T for X by multiplying T by the export transform from L to N.

So that there will not be any overhead for moving objects (such as the hands of a clock) that are known not to be leaving the locale they are in, Spline does not automatically check whether objects should be transferred from one locale to another. However, Spline provides a function that makes it easy for an application to check whether an object should be transferred.

3.2 Locale Servers

Locales are created dynamically by Spline processes just like any other Spline object. Once created, a locale is maintained and can be altered by the process that created it. Other processes can create objects and place them in the locale. These other processes are responsible for the maintenance of these other objects. Locales are not assigned by Spline, but rather are designed by the creators of parts of a virtual world.

Special processes called *locale servers* are used to maintain a concise record of the state of all the objects in a given locale. This is done so that a Spline process that becomes interested in a locale L can rapidly obtain complete, up-to-date information about the objects in L without having to individually contact all the processes that have objects in L. Whenever a new locale is created, it is assigned to a locale server.

4 Location-Addressable Communication

One of the greatest problems faced by the builders of large multi-user virtual environments is dealing with large numbers of changes in the environment. For example, consider a virtual world encompassing an entire town where hundreds of people are interacting. In this situation, the world model contains a huge number of objects and because many people are active, there are a lot of objects that are moving and changing. This implies that a great deal of information has to be communicated that specifies changes in the world model.

If an individual user attempts to maintain a complete picture of the evolving virtual world, he will have to receive and process a torrent of information about changes happening in the world. One would expect this processing load to grow more or less linearly in the number of users and, as the number of users grows, eventually overwhelm the resources of any one user.

From the earliest days of research on multi-user virtual worlds it has been realized that this problem can be avoided by recognizing that most of the information about changes is about parts of the virtual world a given user cannot see or hear or interact with in any way. A user can ignore all this extraneous information and maintain a picture of only the small part of the virtual world that is near.

For example, in SIMNET [1] and in systems based on the IEEE Distributed Interactive Simulation standard (DIS) that grew out of SIMNET, 'filtering' is used to discard information about parts of the virtual world that need not concern a given user. This greatly reduces the storage and processing requirements at each node, but has the problem that even though much (or even most) of the information arriving at a node is discarded, information about every object in the virtual world arrives and has to be dealt with to at least a small degree. As a virtual world grows big enough, even a small degree of attention to each piece of information can swamp the resources available to any one user.

Locales take the logical next step of breaking a virtual world up into chunks that are associated with different communication addresses so that a user who is not interested in a given portion of the virtual world need not receive any messages about it. This multi-address approach has been independently proposed for inclusion in NPSNET [4] and is used to some extent in the Jupiter project [5] at Xerox PARC, which uses separate multicast addresses to support audio and video teleconferences in separate 'rooms'.

In Spline, the messages about changes to objects in a given locale are sent only to the appropriate address associated with that locale. By opening connections to some multicast addresses and not others, a user process can very efficiently obtain information about changes to objects in the locales surrounding the focus of attention, without receiving messages about changes to objects in other locales. This pushes the main burden of filtering onto the routers and interface cards in the network connecting the users, which are specifically designed to perform this kind of processing efficiently. The total processing required for a given user is only proportional to the number of other users that are near, not to the total number of other users. This means that a multi-user virtual environment can become extremely large as long as the users are sufficiently spread out.

A similar result is obtained by systems like WAVES [6] and RING [7] through the mediation of intermediate server processes rather than multicast messaging. This approach achieves increased flexibility at the cost of increased latency. For example, the intermediate servers can dynamically determine which user is near which other user and dynamically route messages accordingly. However, the fact that messages must go from users, to the servers, and then back to other users, rather than travelling directly from one user to another inevitably increases transit times.

5 Culling

The filtering of information based on locales is useful beyond merely reducing the number of incoming messages a Spline process has to handle. In particular, it dramatically reduces the size of the world model maintained by any one Spline process. This in turn reduces all aspects of the processing that has to be done by Spline and an application.

For instance, consider the way visual rendering is done in Spline. Visual rendering operates on everything in the locale L that contains the point of view and on everything in the neighbors of L. To support this, Spline maintains a scene graph that is isomorphic to the hierarchical structure of parent relationships between objects in the Spline world model. The coordinate system of L is used as the coordinate system of the scene graph as a whole. The relative positions of the neighbors of L are specified by the appropriate inter-locale import transforms.

Note that the scene graph created does not correspond to the entire virtual world, but only to the small part of the world that is in the locales surrounding the point of view. As a result, filtering based on locales serves as a fast and effective first order culling of the scene. The key advantage of this culling is that it is based on predetermined visibility constraints. This is faster than culling that is computed at run time, and it can rely on information that is too costly to compute at run time. For instance, locales can be used to omit the rendering of the interiors of buildings when they are not visible from the outside.

This use of locales follows along the path pioneered by work on the static determination of visibility regions [3, 7]. However, it is important to note that locales go beyond this work by supporting a number of additional features.

6 Precise Positions and Velocities

A second problem faced by the builders of large virtual environments is how to maintain precision over long distances. It is attractive to use 32-bit floating point numbers to represent positions, because they are compact and interface well with current graphics hardware. However, this causes a problem with precision. While 32-bit floating point numbers span a wide range of values, their precision drops approximately linearly with increasing magnitude as illustrated in the following table.

Distance From Origin	Positional Accuracy	Velocity Accuracy
1	$1.2\ 10^{-7}$	$3.6\ 10^{-6}$
10^3	$6.1\ 10^{-5}$	$1.8\ 10^{-3}$
10^6	.06	1.9

One meter from the origin of the coordinate system, 32-bit floating point positions (in meters) can be specified to an accuracy of $1.2\ 10^{-7}$ meters. Assuming that visual output is generated at 30 frames per second, smooth velocities (in meters/second) can be specified to an accuracy of approximately 30 times $1.2\ 10^{-7}$ meters per second or $3.6\ 10^{-6}$ meters/second. This precision should be more than adequate for just about any virtual world.

However, farther from the origin of the coordinate system, the precision of 32-bit floating point numbers drops dramatically. As shown in the table, precision is still adequate at 1 kilometer, but is so low at a distance of 1000 kilometers that it would be impossible to represent smooth motions; rather, objects would jump around in .06 meter (2.5 inch) increments.

The effects of imprecision are illustrated in Figure 5. The track at the bottom of the figure shows a one inch cone moving along a slanting line 1 kilometer from the origin of a virtual world. It shows the exact line the point of the cone is intended to follow and superimposes 15 snapshots of the cone at equal time intervals. The track at the top shows the same cone trying to move along a parallel path 1000 kilometers from the origin. The imprecision of 32-bit floating

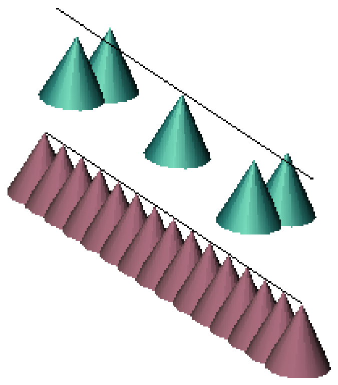

Figure 5: Spacial aliasing due to floating point representation.

point numbers at this distance causes the point of the cone to jump around erratically from grid point to grid point rather than following the desired line. Another effect of the imprecision is that while 15 images of the cone are shown, there are only 5 distinct cone images because consecutive images often appear in exactly the same place.

There are several ways to avoid problems with precision. Most simply, you can keep the virtual world small. This is the case with most of the virtual worlds that currently exist. Alternatively, you can use numbers with more bits of precision. This is the approach taken by the DIS protocol, which uses 64-bit numbers. Locales adopt a third approach—using multiple, connected coordinate systems.

Each locale has its own local coordinate system. The position of an object is specified solely with respect to the locale it is in. Locales are kept small enough—e.g., less than a kilometer on a side—that high precision is always available. To avoid errors in the positioning of locales with respect to each other, there is no overall global coordinate system. Rather the relationship between a locale and its neighbors is specified locally. This makes it possible to specify positions and velocities precisely in an arbitrarily large virtual world using 32-bit numbers.

The basic effect of locales is that wherever you are, you are near the origin of the coordinate system in use. Many operations take place within a single locale. When operations involve multiple locales, the positions of the objects in other locales are converted into positions in the locale you are in. Inevitably, if one of these objects is far from you, its converted position will not be very precise; however, this does not matter because since the object is far away, you cannot interact with it precisely in any case.

7 Combining Separately Designed Pieces

A third problem facing the builders of large virtual environments is how to successfully combine the work of many designers into a coherent whole. Locales address this problem by providing a nice mechanism for encapsulating pieces designed by different people so that they can be easily combined.

Because locales are related only locally on a neighbor-to-neighbor basis with no global coordinate system, locales only require local rather than global consistency. If a designer is creating a locale that is adjacent to another locale, then he must make sure that the locales are consistent with each other—i.e., match up well at their contact points. However, designers of locales that are distant from each other need not be constrained by each other's activities in any way. In addition, the transforms used to relate adjacent locales can be used to adjust for many kinds of differences. For example, simple transforms can be used to connect a room designed with Z-up to a room designed with Y-up or rooms built on different scales.

8 Interesting Effects

When using locales, a designer has total freedom in the choice of transforms relating a locale to its neighbors. Most of the time, one is likely to create transforms so that they correspond to simple Euclidean space. For example, when designing a building full of rooms each in its own locale, one might ensure that:

- Whenever a locale L has a neighbor N: the export transform E is the inverse of the import transform I; E and I specify nothing other than translation and rotation; and L is a neighbor of N with export transform E^{-1} and the import transform I^{-1}.
- Every chain of neighbor relations from L to itself corresponds to the identity transform. (A corollary of this is that if there are two chains of neighbor relations from L to another locale N, then they correspond to the same net transform.)
- The locales for the rooms partition the space corresponding to the building as a whole without overlapping or leaving gaps.

However, none of the constraints above are necessary for efficient processing and none are demanded by Spline. By selectively breaking them, a number of interesting and useful effects can be obtained.

For example, you can implement a mirror by making a locale be a neighbor of itself with an import transform specifying reflection. You can create a toroidal region by using a transform that puts a locale beside itself. You can create one-way windows from one locale to another by making one-way connections. You can alleviate the space crunch in a virtual building by allowing a room to be much bigger than the building that contains it.

9 Beacons

Sending all the messages about objects in a virtual world via the multicast addresses of the containing locales allows very efficient filtering, but introduces a chicken and egg problem. How is a process to find out anything about an object (e.g. what locale it is in) unless it already knows what locale the object is in? In Spline, this problem is addressed by introducing a special class of objects called *beacons* that can be located without knowing what locale they are in.

Consider the following situations. Suppose that you are in the process of logging into a virtual world. Before you can see or hear anything, you have to put yourself somewhere, but your process cannot find out anything about the virtual world before you find out what locales various things in the world are in. Similarly, suppose that you are interacting in a virtual world

Tag – An identifier specified by the creator of the beacon.
Address – The multicast address of the locale containing the beacon.

Figure 6: The key fields of a beacon object.

and wish to locate a particular portion of the world (or service provided in the world) that someone has told you about. Unless you fortuitously happen to be in the same locale as what you are looking for (or in a neighbor of this locale), your local copy of the world model will not contain any information about what you are looking for. Finally, suppose you want to locate a friend of yours who you think may also be interacting in the virtual world. Since your friend may be moving around from locale to locale, even an organized search may fail to find him.

In Spline, these conundrums are resolved by using beacon objects. As shown in Figure 6, a beacon has two key fields: a tag and the multicast address of the locale that contains the beacon. As a group, the beacons in a virtual world act as a content-addressable index from tags to the multicast addresses of locales. They make it possible to decide what locales to attend to based on what the locales contain.

The key feature of a beacon is that in addition to broadcasting messages about itself via the multicast address of the locale it is in, it sends messages about itself to a beacon server process. To ensure that this mechanism is scalable to large virtual environments, an unbounded number of potential beacon server processes are allowed. Which server to contact is determined by hashing on a beacon's tag.

A function is provided whereby an application can request that Spline obtain information about every beacon with a particular tag. Spline does this by connecting to the beacon multicast address corresponding to the tag in question. After sufficient time has elapsed for a new steady state of the local world model to be achieved (at most a few seconds) a process can be assured that it has complete information about every beacon with the indicated tag. If the process decides that it is particularly interested in any of these beacons, it can then request that Spline connect to the locale containing the selected beacon(s). The local world model will then be filled out with complete information about all the objects in the associated locale(s).

Designers use beacons to mark things they want people to be able to find. The only real complexity involved with beacons revolves around the ways tags are communicated between users. There are three ways this is done.

First, a designer can create a beacon and then publish the tag, e.g., on the World Wide Web. For example, if someone creates a part of a virtual world and wants other people to visit, he can mark the area with a beacon and publish the tag. Similarly, if someone wants to make it easy for people to find them in a virtual world he should include a beacon as part of the avatar representing him in the virtual world. He can then communicate the tag on this beacon to those he wants to be able to find him (e.g. in Email messages).

Second, one can create a beacon and not publish the tag. In this case, one is not inviting people in general to locate the beacon, however, people that run into the beacon can subsequently keep track of it no matter where it moves by remembering the beacon's tag. As an example of the utility of this use of beacons, consider the following problem.

Suppose that you have met another person in a virtual world and he asks you to follow him so that he can show you around. An easy way to do this is to attach the visual POV controlling what you see to his representation in the virtual world as a subobject. Your POV will then go wherever he goes and see whatever he sees.

However, what happens if he transfers to another locale? If your Spline process is already connected to the address of the new locale then there is no problem. You get messages specifying that his avatar is in the new locale. Your POV is a subobject of his avatar and therefore implicitly moves to the new locale as well. However, if he moves to a locale your Spline process is not connected to, you stop receiving any information about his avatar. He disappears from view just the same as he would if he logged out of the virtual world, or if his machine crashed. Your POV is left dangling as the child of a non-existent object. To prevent this from happening, a person who wants to make it possible for people to follow him should include a beacon in his avatar. By requesting that your Spline process attend to the tag in this beacon, you can guarantee that your process will maintain contact with his avatar no matter where it moves.

The third way that tags can be communicated is as follows. Someone who wants to provide a service can publish a tag, without creating a beacon. A person who wants to use the service can then create a beacon using the published tag. The service provider can determine where services should be rendered by looking for the appearance of beacons with the tag.

An interesting example of this is used to communicate between an application and a visual renderer. The Spline rendering application that creates the images seen by a user can run in a completely separate process (or even on a separate machine) from the main application interacting with a user. This brings up an interesting question, how can a given visual renderer determine which user it should generate images for? The connection is made by having the user's application create a beacon that specifies which visual renderer it wants to use. This is done by using a tag corresponding to the Internet address of the machine where the visual renderer is running. By looking at beacon tags, a given visual renderer can link up with the appropriate user process.

10 Diamond Park

Diamond Park [8] is the first large-scale virtual world built using Spline. The park consists of a square mile of virtual landscape with about a dozen buildings, two lakes, and two ornamental pools scattered around a central valley surrounded by hills (see Figure 7). Multiple users interact in the park while riding steerable computer-controlled exercise bicycles with variable pedal resistance. When near each other, visitors can converse verbally.

Currently, the objects in Diamond Park are divided among 62 locales. As discussed below, some of these locales represent major activity areas of the park; however, most of the locales are used to create appropriate transitions between the major locales.

10.1 Desert House

The Desert House shown in Figure 8 is a recent addition to Diamond Park. An interesting feature of the Desert House is that the interior is much larger than the exterior. To illustrate this, Figure 8 shows the interior and exterior of the Desert House superimposed in the same image—a view never seen by users of Diamond Park.

Figure 9 shows the locales used to implement the Desert House. The locale P contains the landscape surrounding the Desert House while locale D contains the interior of the building. The Desert House can be entered through two vestibules. Consider the vestibule on the left in

Figure 7: A panoramic view in Diamond Park.

Figure 8: The inside and outside of the Desert House.

Figure 9: The Desert House locales.

Figure 10: One of the Desert House vestibules.

Figure 11: Moving through an obelisk.

Figure 9. It is divided into two locales V_1 and V_2. V_1 is a neighbor of P and can be seen from certain vantage points in the park. V_2 is a neighbor of V_1 & D, and can be seen from certain places inside the Desert House.

Critically, D is not a neighbor of P and there is no locale that D and P both neighbor. This means that the objects in D and P are never simultaneously rendered. As a result, visitors never experience the slow frame rates that would be caused by having the graphics hardware simultaneously draw the complex scenes corresponding to D and P.

The appearance of one of the Diamond Park vestibules is shown in Figure 10. In this figure, transparent polygons have been added to highlight the boundaries of the locales. Notice that the vestibule has been carefully designed so that when you are in P you can see V_1, but not V_2 or D, and when you are in V_1 you can see P and V_2, but not D. This is essential, because when you are in P or V_1, the objects in D are not rendered.

Consider what happens when a visitor goes into the left vestibule in Figure 9, across the interior of the Desert House, and out the right vestibule. Because the interior of the Desert House is big, the visitor has traveled a long way inside the Desert House. However, because the outside of the Desert House is much smaller, the visitor has traveled a much shorter distance across Diamond Park. This paradox does not obtrude on visitors to the park because at each moment the scene they see is locally consistent. There are no mathematical contradictions because the locales in the Desert House are only related pairwise, with no global reference frame.

10.2 Transportation Obelisks

Diamond Park is equipped with 22 small obelisks dotted around the landscape that act as a rapid transit system. A visitor can enter any obelisk and then exit any other after traveling only a short distance. This effect is supported by locales. As with the Desert House, two perspectives are simultaneously maintained that are geometrically inconsistent, but in a very useful way. There are 22 obelisk exteriors, which share one relatively small interior space. From the perspective of the interior, the 22 obelisk doors are close together, arrayed around the walls of the interior space. From the perspective of the park, the obelisks are far apart. Figure 11 shows a sequence of snapshots illustrating what it looks like to go through an obelisk from one part of the park to another.

Whenever locales are used to maintain inconsistent perspectives, the problem arises of controlling sight lines so that the inconsistencies are not visually apparent. In the Desert House, sight lines through the vestibules are controlled by introducing corners you cannot see around. In the obelisks, sight lines are controlled by filling each entryway with a picture of what is beyond the entry. When entering and leaving obelisks, visitors walk right through these pictures but cannot see beyond them until after they have passed through.

10.3 Evaluation

Because Diamond Park consists of only one square mile of terrain, precise positioning and movement would be possible even if locales were not used. However, the other virtues of locales are clearly demonstrated by Diamond Park.

Breaking the park into locales allows complex graphic models to be used in many different places without overloading the graphic rendering power available, because two complex models never have to be rendered at once. Specifically, Diamond Park is divided into four major zones:

the outside landscape and three complex building interiors. At any one moment, the scene graph being rendered never contains more than about 35% of the total model (the outside landscape is more complex than any one building interior).

Equally important, visitors congregated in one area of the park can interact without taxing the computational resources of users in other areas of the park, because a visitor in a given locale does not have to process any information about visitors in distant locales. If the visitors are equally distributed among the major zones, then an individual visitor receives messages from only 25% of the other visitors.

It is important to realize that the numbers above are a function of the complexity of the specific Diamond Park model. If the model contained 100 buildings with highly complex interiors, then the scene graph at any given moment would only contain about 1% of the model and a visitor would receive messages from only 1% of the other visitors.

Locales are used to achieve a number of interesting effects in Diamond Park. This includes buildings with large interiors such as the Desert House and the kind of non-Euclidean connectivity illustrated by the obelisks.

Perhaps the most important single virtue of locales is their support for combining separately designed pieces. For example, the Desert House interior was designed long after the rest of Diamond Park. It was placed inside an existing building in the park. However, when doing this the designers of the Desert House interior were not constrained by the coordinate system of the park as a whole, or its orientation, or scale, or which axis was up, or in fact even by the size of the existing building. Beyond this, they were not required to use the same graphic modeling tool. The only modification of the preexisting Diamond Park model that was needed was the addition of doorways into the building that became the Desert House.

11 Future Directions

To date, Diamond Park has been the only major use of Spline and locales. This experience led to a number of improvements in locales. These include allowing locales to have other locales as their parents and a greatly improved representation for the boundary of a locale. It is important that a range of further experiments be performed, and we expect that this will lead to further improvements.

An interesting area where improvement could be made is in the way locales support audio rendering. Currently, one set of transformations controls both the visual and audio relationship between locales. It might be better to include a separate specification of the way sound propagates from a locale to its neighbors. This would allow improved audio rendering without complex reasoning about what polygons constitute floors and walls.

Another important direction of research is the creation of tools to support the design of locales. In some situations, e.g., breaking the model for a building with many rooms into locales, it should be possible to design locales completely automatically, using visibility analysis [3]. In other situations, it is appropriate for a designer to take more direct control. However, even then, semi-automated tools for visualizing and modifying locales are essential.

Acknowledgments

R. Waters and D. Anderson are the principal architects and implementors of Spline, with major contributions by J. Barrus, J. Marks, D. Marmor, M. Casey, and W. Yerazunis. J. Barrus, S. McKeown, and I. Sterns are the principal architects and implementors of Diamond Park, with major contributions by D. Anderson, D. Brogan, M. Casey, J. Hodgens, R. Waters, and W. Yerazunis.

References

[1] J. Calvin, et al, "The SIMNET Virtual World Architecture", *Proc. IEEE Virtual Reality Annual International Symposium*, 450–455, (Seattle WA, Sept. 1993), IEEE Computer Society Press, Los Alamitos CA, 1993.

[2] J.W. Barrus and R.C. Waters, *QOTA: A Fast, Multi-Purpose Algorithm For Terrain Following in Virtual Environments*, technical report 96-17, MERL Cambridge MA, July 1996.

[3] T.A. Funkhouser, C.H. Sequin, and S.J. Teller, "Management of Large Amounts of Data in Interactive Building Walkthroughs", *ACM SIGGRAPH* Special Issue on 1992 Symposium on Interactive 3D Graphics, (Cambridge, MA), pp. 11-20, 1992.

[4] M.R. Macedonia, et al "Exploiting Reality with Multicast Groups", *IEEE Computer Graphics and Applications*, 15(5):38–45, September 1995.

[5] P. Curtis and D.A. Nichols, "MUDs Grow Up: Social Virtual Reality in the Real World", *Proc. 1994 IEEE Computer Conference*, 193–200, January 1994.

[6] R. Kazman, "Load Balancing, Latency Management and Separation of Concerns in a Distributed Virtual World", in *Parallel Computations - Paradigms and Applications*, A. Zomaya (ed.), van Nostrand, 1996.

[7] T.A. Funkhouser, "RING: A Client-Server System for Multi-User Virtual Environments", *ACM SIGGRAPH* Special Issue on 1995 Symposium on Interactive 3D Graphics, (Monterey, CA), pp. 85-92, 1995.

[8] D.B. Anderson, et al, *Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability*, technical report 96-02, MERL Cambridge MA, January 1996.