Distributed 3D Audio Rendering

Colin Low, Laurent Babarit
Internet Communications Services
HP Laboratories Bristol

HPL-98-11
January, 1998

internet, multimedia, audio, 3D, virtual environments

This paper discusses the use of audio spatialisation techniques to complement current developments in Voice-over-IP. The result is to create a shared synthetic audio environment in which Voice-over-IP streams from several participants appear to be spatially located. While the intent is the inter-work with multi-user VRML environments, our proposals build on current Internet multimedia proposals and are independent of the user interface used to navigate a virtual space.

Internal Accession Date Only
© Copyright Hewlett-Packard Company 1998

Distributed 3D Audio Rendering

Abstract

This paper discusses the use of audio spatialisation techniques to complement current developments in Voice-over-IP. The result is to create a shared synthetic audio environment in which Voice-over-IP streams from several participants appear to be spatially located. While the intent is to inter-work with multi-user VRML environments, our proposals build on current Internet multimedia proposals and are independent of the user interface used to navigate a virtual space.

1.0 Introduction

Telephony and teleconferencing create an illusion of proximity. In current implementations there is no *environment* (other than the imagination) in which users can be said to be "proximate". In a typical teleconference all participants share the same null space somewhere inside a telephone ear piece.

One of the most exciting developments in the Internet and the WWW is the perception of the network as a place. This approach has been described as "social virtual reality" [Curt94] [Wat96] and "social computing" [Rock97]. The combination of computing and networking provides the means to create rich synthetic environments where people can meet, communicate and socialise. There is a wealth of early examples of this concept: Usenet newsgroups, mailing lists, MUDs, Internet Relay Chat, MBONE conferencing, and WWW sites. Early examples concentrated on text, but more recent applications use audio and video conferencing. Applications such as Sony's Community Place and Online! Traveller integrate synthetic visual environments based on VRML [VRML97] with audio conferencing.

Our everyday experience of communicating with other people takes place in a spatial context. We use space as a way of initiating and breaking conversations. We use space to create privacy. We already possess a rich selection of social protocols for managing meetings in a spatial environment. We can contrast this with the telephone, which in its use is primarily one-to-one and intentional: that is, I call you when I have something I want to say. Telephony in its current form is too limited to support richer and more intuitive protocols for communication and socialisation. Nevertheless the telephone has become the model for voice communication.

A great deal of research in the Internet community has concentrated on the mechanics of transmitting digitised and compressed audio across an IP Wide Area Network (WAN). Voice over IP (VoIP) applications in general adopt the user interface assumptions of traditional telephony (but see [Hard1]) - the network is a technology for delivering a monaural stream of audio into an ear piece.

In this paper we intend to show how something richer than traditional telephony can be provided to a collection of communicating individuals *without impacting the delivery of audio* - that is, we are concerned with the presentation of audio to the user. Audio rendering is a term that is used to describe how sources of digitised sound are composed within a synthetic environment to create an illusion of physical space. Most people are familiar with 3D computer graphics, where a collection of geometrical objects with colours, textures and light sources are rendered to produce the illusion of real physical space. Something analogous is possible with sound. Distributed audio rendering is the presentation of sound to a collection of users in a way that creates the collective illusion that they occupy a shared physical space. It combines ambient sound, incidental ("Foley") sounds, and real-time voice communication, and uses sound spatialisation algorithms and environmental acoustic effects (such as reverberation) to provide a convincing sense of a place that is shared by users who may bc half a world apart.

The work presented in this paper diverges from the general trend in distributed virtual environments (DVE) by concentrating on audio. We had specific goals in doing this:

- to create a set of distributed audio rendering abstractions that could be used in conjunction with a number of distributed virtual environment technologies. Although we have multi-user VRML developments in mind (such as Living Worlds [Liv97]), there is no strong bias towards 3D visual environments.
- to leverage work on multimedia session protocols emerging in the Internet multimedia community - that is, we wanted to build upwards on a base of developing standards.
- to explore uses of spatial sound for multi-party communication as a medium in it own right.

In section 2.0 we discuss the technology behind sound spatialisation. In section 3.0 we go on to review the use of audio in social virtual reality/distributed virtual environments, and identify how communication sessions are established. In section 4.0 we discuss the concept of the multimedia session. In section 6.0 we go on to show how the familiar idea of a conference or session can be employed in the context of distributed audio rendering, and in section 7.0 we discuss what kind of audio rendering information needs to be communicated between session participants. Lastly, we outline the implementation of these ideas, and present our conclusions.

2.0 Spatial Sound

Synthetic sound spatialisation is the processing of monaural sound in such a way that when it impinges on our ears, it reproduces the characteristics of a sound located in a 3D space external to the listener. A detailed explanation of the psychoacoustics of spatial hearing can be found in [Beg94]. Briefly, sound travelling to our ears will usually arrive at one ear sooner than the other. Sound is also diffracted by the head and shoulders so that there are frequency-dependent intensity differences at the two ears. Some sound is reflected by the structure of the ears, and this introduces spatially-dependent notches in the sound frequency spectrum. These timing, intensity and spectral effects are transformed by the brain into a perception of sound in space.

These effects are modelled by a Head-Related Transform Function (HRTF). A digitised monaural audio stream can be convolved with an artificial HRTF to create a binaural audio stream that reproduces the timing, frequency and spectral effects of a genuine spatial sound source. Environmental acoustic effects such as reverberation can also be added. For ideal results an HRTF needs to be tailored to the physical characteristics of person, but dummy-head HRTFs can still produce highly pleasing results.

Sound spatialisation is a computationally intensive task that is ideal for digital signal processing hardware. Low cost consumer hardware that can carry out a spatialised mix of several concurrent streaming and static sound sources (e.g. boards based on the A3D chipset from Aureal) has become available for the Wintel platform. Microsoft has defined the device-independent DirectSound interface for spatial sound as part of DirectX [DirX], and although this interface is Wintel specific, the PC games market alone will ensure the widespread acceptance and use of low-cost spatial sound hardware.

A spatial mix of several sources of sound is hardware intensive, and must be recomputed relative to each audio observer. This has important implications for a distributed audio rendering architecture which will be discussed later.

3.0 Related Work

Much of the agenda for social virtual reality has been spelled out as part of the Jupiter [Curt95] project. Jupiter is an extension of the MUD concept [Curt94], which in its original form consisted of a multi-user environment in which text was used both for the description of places and objects, and also as a means of communication between participants. Behaviours can be added to objects and places using a built-in programming language, and multi-user aspects such as ownership, modifiability, extensibility, and concurrency are handled with a relatively high level of sophistication.

Jupiter has a highly developed model for audio and video conferencing. Microphones and cameras are modelled as sources, and video and speaker panes as sinks. Sources and sinks communicate through a channel, which corresponds to a multicast address. The membership of a communication session is modelled through the dual abstractions of a *channel manager* and a *key manager*. The latter manages encryption key notifications to ensure that sessions are private. Although any object may own a channel, the normal convention is for a channel to be owned by a room or place so that all the people in the room can communicate. Jupiter is based on dumb client software; the object model shared by the users of the environment resides in a single server.

Open Community [Wat97][Yer97] is a distributed virtual environment (DVE) infrastructure based on SPLINE [Wat96]. SPLINE provides a distributed shared object model, and is optimised to reduced the overhead and latency associated with achieving a consistent view across multiple clients. As DVEs are communication intensive, the unit of scaling in SPLINE is the *locale*. Each locale corresponds to a multicast group, and an observer in one locale need not observe the communication traffic corresponding to another locale. Audio classes such as sources and observers are derived from the top-level class in the object architecture, and so are subsumed within the overall architecture. Audio can consist of ambient, Foley and real-time sources, and sounds are rendered within the spatial context of the enclosing locale, using a multicast group for the real-time audio streams, which are mixed and rendered with other sounds at the client. Open Community provides a very complete solution, with the disadvantage (if it can be called that) that the distributed audio rendering is embedded within the whole, and although the internal protocols have recently been published [Wat97a], the architecture is relatively monolithic.

Other research DVEs we have studied which incorporate audio communication are MASSIVE and DIVE. MASSIVE [Green95][Green96][Green96a] makes extensive use of a spatial extent called an *aura* to manage communication through multicast groups. DIVE [Hag96] assigns a multicast address to a spatial extent called a *world*, and resembles Open Community in this respect.

An example of a commercial solution incorporating audio conferencing within a DVE is the Onlive! 3D Community Server and Onlive! Traveller. From the information available it appears that multiple VoI audio streams are mixed using a central mixing server, an approach which is practical for monaural sound, but less practical for a spatial mix, where each client needs a computationally intensive individual mix. Sony's Community Place VRML browser and accompanying Community Place Bureau server permit monaural audio chat within a VRML context.

In each of these architectures similar questions and themes emerge:

- where is audio mixing carried out - distributed at the client or centralised in a server?
- the use of a scaling mechanism based on limitation of association - locales, places, rooms, auras.
- a mechanism to define implicit participation in an audio conference or session, generally identical to the overall scaling mechanism (locales, places, rooms, auras).
- the use of multicast groups as an efficient implementation of the scaling mechanism.
- the consistent group membership problem, or how (and whether) participants agree on the membership of an audio session in a distributed environment, and how state associated with the group or session is maintained.
- whether the implementation of audio session state is centralised or distributed.

Similar concerns can be found in discussions of conferencing or multimedia sessions. This is not surprising, as a multimedia session can be viewed as a form of distributed virtual environment (or vice-versa, depending on prejudice!). We now go on to discuss this area.

4.0 Conference/Session Concepts

The distinction between a conference and a multimedia session verges on the theological, and in what follows we use the following definition [Hand97a]: a multimedia session is a set of multimedia senders and receivers and the data streams flowing from senders to receivers. A review of Internet standardisation activities in this area can be found in [Hand97].

The set of issues dealt with by multimedia session/conference control can be decomposed as follows [Hand95][Jacob93]:

- meta-session management - how to initiate and finish conferences, how to advertise their availability, how to invite people to join.
- creation - selection of resources, deciding how participants rendezvous.
- participant selection - privacy and authentication.
- communication - audio, video, allocation of resources.
- coordination - floor control, ownership and permission.

There is a wide divergence in the solutions proposed and adopted. Two representative extremes are T.124 [T.124], the ITU-Ts recommendation on generic conference control, and tools such as **vat** in daily use on the MBONE [Mac94]. T.124 has a specified solution to each of the issues above, but it sacrifices scalability. MBONE tools are known to scale to several hundred participants, but pay the price of a relaxed view of who is participating. On the MBONE it is possible to participate in a conference simply by knowing the multicast addresses used for its transmissions. Attendance information is propagated lazily via multicast using Realtime Control Protocol (RTCP) messages as part of the Real-time Transport Protocol (RTP) [Schu96]. In T.124 membership can be restricted by requiring a password to join, but on the MBONE an additional mechanism for distributing encryption keys is needed.

At the root of many difficulties is the representation of session state. It can be held centrally in a server, an approach which is successful for dozens of participants (as Jupiter demonstrates), but which is less viable for thousands of participants. There are concerns about thousands of participants trying to maintain reliable communication with a server on a network like the Internet - the likelihood of drop-outs, partitions and false deaths would be high. In other words, although this approach promises a great deal, it is likely that part of the session state would be meaningless (and we don't know which part).

A second approach is to avoid dependence on a central server by using a fully decentralised algorithm where session state is distributed across participants. This requires us to implement solutions to some of the classic problems of distributed systems: group membership, election, coordination and consistency. Scaling remains a problem. The MBONE's relaxed approach to session state has the attraction of being scalable and robust.

In what follows we adopt an approach which attempts to insulate us from the details of the implementation of session state. Our goal is to discuss how widely distributed participants can share the illusion of a synthetic 3D space in which independently contributed sources of streaming sound are located. We have not solved the ancillary problem of session state in a large distributed system. It should be noted that DVE design is concerned with multiple participants modifying shared state, and is concerned with the same set of distributed algorithms as multimedia session control. Scaling is a similar concern (see for example [Lea96]).

5.0 Assumptions

We assume that users have workstations with sufficient power to spatially render 2-3 real time audio streams, and several ambient and Foley sounds. We also assume that users are connected to an IP network with sufficient bandwidth to receive 2-3 audio streams simultaneously. In normal conversation only one person talks, so multiple simultaneous audio streams will be the occasional and brief exception, not the rule (although some nationalities may disagree with this!). Lastly we assume the availability of IP multicast, with a multicast backbone network similar to the MBONE.

6.0 Session Control

6.1 Session Initiation

In a virtual environment one does not join a conference or a multimedia session: one joins a group of people, or one walks into a room where conversations are going on. We do not think of our daily coffee-machine discussions as "multimedia sessions". The session mechanism should be invisible to an inhabitant of a virtual environment. The session concept is a means of limiting the distribution of real-time information to a specific group of recipients.

In Open Community [Yer97] a session is implicitly associated with a spatial *locale*, in that each locale defines a multicast address to be used for audio streams. In Jupiter [Curt95] any object can act as the manager of a multimedia *channel*, and that owner is typically a room. In MASSIVE [Green95][Green96][Green96a] an *aura* defines a spatial extent; sources of sound contained within an audio aura will be heard.

As we are particularly interested in interworking with a VRML 2.0 DVE (e.g. a Living Worlds [Liv97] implementation) it is useful to discuss sessions in this context. In VRML 2.0 it is possible to associate a variety of sensors with VRML nodes. When a sensor is triggered, it can route an event to another VRML node. If that node is a SCRIPT node, then arbitrary logic can be invoked. For example, a room can be associated with a proximity sensor with the same spatial extent as the room, so that when the viewpoint of a user enters that space, an event is triggered. Likewise, each personal avatar in a shared world could have a three meter spherical proximity sensor attached, so that when a user's viewpoint moves within this three meter aura, an event is triggered. When the user moves away, another event is triggered. This ability to trigger events within VRML using a variety of criteria makes it possible to set up and tear down sessions without the user having to be involved. When I walk into a room, I trigger a script which establishes whether a session exists for that location; if there is no session, one can be created, and if there is, I can be issued an invitation to join.

Unlike current MBONE or T.124 conferences, details of these lightweight, spontaneous conferences are unlikely to be found in a registry - they could be, but announcing to the world using a session announcement protocol that I have bumped into Jim at the virtual coffee machine seems excessive. In the absence of an explicit session description, there has to be a way of associating a session with the context in which it takes place. When I walk into a virtual location titled "Coffee Area", then out of all the sessions that might be taking place concurrently, I would like to identify the one currently associated with "Coffee Area". Each session has to be named in a way that is both unique and meaningful within the external context, in this case a VRML 2.0 world. If we assume that each participant in a shared VRML 2.0 world sees the same graph of VRML nodes, then it is straightforward to extract contextual information which can be used to identify a session associated with a node.

Race conditions will be the norm, not the exception. When two avatars A and B approach, then both auras will intersect "virtually simultaneously". If I am avatar A, then I will trigger a proximity sensor associated with my local copy of avatar B, and I will create a session (named after A) and invite B to join. User B will experience the same situation in reverse; he or she will trigger a proximity sensor associated with avatar A, create a session (named after B), and invite me to join.

If we both had to interact through a centralised server then this kind of race condition could be avoided. If we do not assume any central logic, then this kind of race condition is inevitable. Sessions and invites need to be tagged in such a way that I can recognise that the invite I have just received is the complement of the invite I have just sent. One solution would be a card-cutting algorithm: A and B both insert a random number in their invites, and the one with the largest number accepts the invite. Our session abstractions (see 6.2 below) make heavy use of opaque identifiers which have no significance within the session layer other than to provide an external context in which to identify sessions and audio entities, and to resolve race conditions.

6.2 Session Abstractions

An audio session is a set of audio entities that exchange audio and rendering streams. Section 7.0 discusses the content of these streams. In this section, we summarize the abstractions and the programing interface used by an application running in a client workstation.

We define the following object classes :

- a **3DAudioSession** represents the local view of an audio session,
- a **3DAudioEntity** represents a participant in a session, that is, a source of audio and rendering streams. A **3DAudioEntity** is the abstract superclass for derived classes which include **3DAudioAvatar** and **3DAudioDevice**. A **3DAudioAvatar** stands for a human being, and can include additional personal information useful to other participants in a session. A **3DAudioDevice** represents various devices such as audio mixers, audio recorders etc.
- a **Local3DAudioEntity** is the local state equivalent of a **3DAudioEntity** and represents a *potential* session member. There can be several **Local3DAudioEntities** registered with the same **SessionManager,** each representing a distinct potential member of a session.
- a **SessionManager** manages session initiation and teardown, and interacts with other **SessionManagers** on other workstations to provide a global view of active sessions. It can also issue an invitation to a **Local3DAudioEntity** somewhere (using an opaque identifier) to participate in a **3DAudioSession**.

- **a 3DAudioObserver** represents the local audio point of view (POV) in a **3DAudioSession.**

To enable the correlation of objects which exist in an external context (e.g. a room in a shared virtual environment) with the audio environment, a **3DAudioEntity** is assigned a unique ID taken from an external context. A **3DAudioSession** is also identified in the same way.

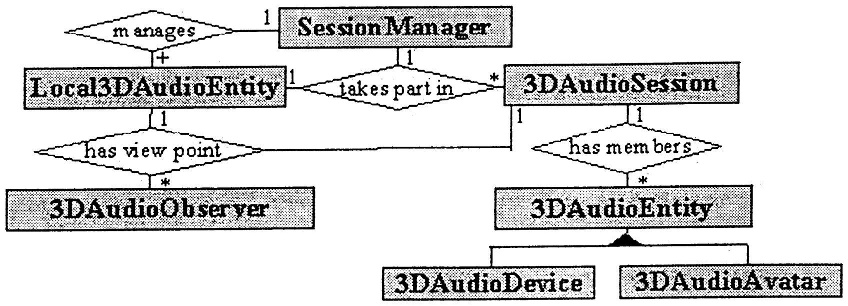

Figure 1: Object relationship diagram

The relationship of these classes is shown above in Figure 1. In the following, we outline the interfaces of each object. What follows does not cover interfaces for permission, privacy etc.

SessionManager

Each participant requires a local **SessionManager** component. The **SessionManager** maintains a small pool of current sessions associated with a participant. The participant is represented by a **Local3DAudioEntity**. Each participant registers one or more **Local3DAudioEntity** objects with the **SessionManager**. The **Local3DAudioEntity** contains the state needed to represent a participant as a potential member of a session, and can represent a person, a robot, or devices such as an audio mixer or sound recorder.

A new session can be created using **SessionManager.create3DAudioSession.** The session is named with an opaque uniqueID, and is passed a set of audio environment parameters as discussed in section 7.1. It is possible to query whether a session with a given name already exists using **SessionManager.isSession().** It is possible to enumerate all the active sessions which a **SessionManager** knows about (typically one, but there could be several). It is possible to register **Local3DAudioEntity** objects with a SessionManager so that it can accept invitations (using the uniqueID (opaque handle) of the **Local3DAudioEntity)** and pass them on as a **Local3DAudioEntity. invite_notification**.

To invite a remote participant into an existing **3DAudioSession**, the embedding application (for example, a VRML 2.0 script node) calls **SessionManager.invite**(), supplying a uniqueID. This invitation is disseminated to all **SessionManagers** (we use a multicast-like notification service [Bra97] with filters over notification contents to implement this), and the Session Manager with a **Local3DAudioEntity** registered using that uniqueID will pass the invitation as an **invite_notification**. When an invitation notification (which includes all the information needed to join the session) is received, the **Local3DAudioEntity** can join the session by calling **3DAudioSession::join**().

The underlying protocol for issuing this invitation could be the proposed Session Initiation Protocol (SIP) [Hand97b], which includes a full session description as specified in the proposed Session

Description Protocol (SDP) [Hand97a].

SessionManager Interface
SessionManager() // constructor
registerLocal3DAudioEntity(Local3DAudioEntity anEntity)
getCurrent3DAudioSessions() -> sequence of 3DAudioSession
create3DAudioSession(uniqueID handle, sessionParameters params) -> 3DAudioSession
invite(3DAudioSession aSession, uniqueID invitee)
isSession(uniqueID aSession) -> true,false

Table 1: SessionManager Interface

Local3DAudioEntity, Local3DAudioObserver and 3DAudioEntity

A **Local3DAudioEntity** represents a potential session participant. It contains state needed to participate in a session - textual identification, preferred communication modes etc. The **Local3DAudioEntity** contains **audioSourceParameters** for describing the spatial characteristics of the audio source. These are outlined in section 7.0. It is also the rendezvous used to receive invite notifications to join a session.

Local3DAudioEntity Interface
Local3DAudioEntity(uniqueID handle) // constructor
setUserInformation(userInformation u)
setSourceInformation(audioSourceParameters params)
invite_notification(remoteId, 3DAudioSession)

Table 2: Local3DAudioEntity Interface

A 3DAudioObserver defines a point-of-view (POV) for an observer. Each participant in a session requires an audio viewpoint (analogous to the visual viewpoint in VMRL) from which to listen to a session. This is done by creating a **3DAudioObserver** and registering with the corresponding **3DAudioSession**. The POV can then be updated as the participant moves around in the virtual audio space. In practice the audio POV might be attached to the same spatial position as the visual POV, so that sound and vision coincide.

3DAudioObserver
Local3DAudioObserver() // constructor setPosition(3DVector aVector) setVelocity(3DVector aVector) setOrientation(3DVector aVector) getPositon -> 3DVector getVelocity() -> 3DVector getOrientation -> 3DVector etc.

Table 3: 3DAudioObserver Interface

A **3DAudioEntity** represents a member of a **3DAudioSession**. When a **Local3DAudioEntity** joins a session it supplies information about the participant when is made available to other participants as a **3DAudioEntity.**

3DAudioEntity Interface
getID() getInformation() -> userInformation getPosition() -> 3DVector getOrientation() -> 3DVector getVelocity() -> 3DVector getAudioSourceParameters() -> audioSourceParameters

Table 3: 3DAudioEntity Interface

3DAudioSession

3DAudioSession contains **sessionParameters** that describe the audio environment and information about the underlying transport used to carry audio and rendering streams. These parameters are discussed on the next session on Audio Rendering Control.

It also contains operations for :

- joining and leaving a session,
- receiving join and leave notifications,
- enumerating the members of a session.

The most contentious operation in the **3DAudioSession** interface is **enumerateMembers()**, which returns a sequence of **3DAudioEntity**. The result of this operation depends on the underlying implementation - if we adopt a relaxed approach to membership, then the result means "these are the members I currently know about".

3DAudioSession Interface

join(3DAudioEntity anEntity)
 	
leave(3DAudioEntity anEntity)
 	
enumerateMembers() -> sequence of 3DAudioEntity
 	
join_notification(3DAudioEntity anEntity)
 	
leave_notification(3DAudioEntity anEntity)
 	
registerPOV(3DAudioObserver)
 	
getSessionParameters() -> sessionParameters

Table 4 : 3DAudioSession Interface

In summary, 3D audio session management shares similar concerns with any multimedia session management system, with the following additions:

- a session must declare a rendering environment as described below in section 7.0.
- sessions are created and joined implicitly, and we need to have ways of relating audio sessions and audio entities to an external context which helps us to relate sessions to, for example, (virtual) spatial locations, or (virtual) groups of people. In addition, we need to resolve race conditions.
- participants continually update their position, orientation and other audio parameters in virtual 3D space. The mechanism for propagating this information is discussed below.

7.0 3D Audio Rendering Control

We have not so far discussed how multiple audio streams from several sources are rendered in a spatial context. The abstractions defined in the previous section were a deliberate attempt to avoid confronting this detail - so long as a **3DAudioEntity** updates its position, orientation and other parameters, we assume that sound can be correctly rendered so that each participant hears the correct spatial mix with respect to his or her audio POV.

We now discuss our "preferred implementation", with the caveat that there are several other possibilities. We assume that a session corresponds to a dynamically-obtained IP multicast address, and that each participant in a session binds to that multicast address and receives RTP packets [Schu96] containing an audio payload in the same way as several MBONE tools. The audio streams from different participants are then mixed in the participant's workstation. Audio conferencing tools such as **rat** [Hard96] and **vat** currently perform a simple additive mix. What we need is for our 3D audio conferencing tool to carry out a spatial mix, rendering each participant/**3DAudioEntity** at the correct spatial position relative to the local POV. To do this the mixer needs to be continuously updated with the spatial rendering parameters corresponding to each audio stream.

What we suggest is that spatial rendering parameters should be an RTP payload type, and that each source of spatial audio should originate a corresponding RTP stream of rendering parameters. Each recipient can then synchronise an RTP audio payload with the corresponding RTP 3D audio rendering payload and carry out a spatial mix. This is similar to techniques used to synchronise RTP audio and video payloads.

In order to communicate 3D audio rendering information, a source and recipient must agree

- an audio environment model
- an audio emission model

7.1 Environment Model

An environment model defines a coordinate system, an origin, and a unit of measure. It also describes audio propagation characteristics such as the speed of sound, the dispersiveness of the medium, and Doppler effects. An environment will also introduce complex diffraction and reflection effects which cumulatively result in reverberation. Each order of reverberation is characterized by its decay time and the reflection coefficient. The decay time is dependent on the distance to the reverberating surface. The detailed computation of environmental effects on propagation is directly analogous to ray-tracing in 3D graphics and is often referred to as *auralisation.*

In models such as in DirectSound [DirX], decay time is fixed and the order is 1. This works well for giving the illusion of being in a middle of a large space but not for typical small rooms. The Java3D model [Java3D] provides a spectrum of possibilities, from single reverberation with fixed decay time to multiple order reverberations where the decay time is calculated given a geometrical representation of the environment with associated reflection coefficients. It is important to note that while most people would not know the difference between a crude reverberation filter and sophisticated auralisation, environmental acoustic effects are important in disambiguating location in the vertical/front-back plane, and also for determining the distance of the source from the listener.

We assume that all the members of a session share the same environment model, and this is communicated as part of joining a session.

7.2 Emission Model

The emission model for a sound source describes the directional properties of the source, and how the sound is attenuated with distance. Sound sources, like light sources, occur in the literature in three forms:

- ambient
- unidirectional point
- directional

Examples of directional emission models can be found in Intel RSX [Int96] and VRML 2.0 [VRML97], which use an ellipsoid model, and Microsoft DirectSound [DirX] and SUN's Java3D [Java3D] which use a cone model.

The authors are currently investigating audio rendering models as a basis for an RTP audio rendering payload.

8.0 Implementation

Our ongoing implementation consists of session layer abstractions written in Java, with native code interfaces driving a heavily modified RAT [Hard1] (an existing MBONE conferencing tool). RAT is modified to carry out spatial mixing and communicates with Microsoft's DirectSound [DirX] interface. We are currently defining a strawman RTP audio rendering payload which will be incorporated into RAT. Our current generation of hardware is Diamond's Monster 3D Sound card for the Wintel platform. Support for DirectX means we currently have to use MS Windows 95.

The implementation of the session layer abstractions uses the Keryx notification service [Bra97]. The notification service provides the advantages of a shared multicast channel for group communications, but does not use multicast. Notifications are forwarded using a mesh of distributors, and distributors forward notifications that satisfy logical filters over the contents of a notification. It was designed to support very large numbers of event types in a scalable way. Communication is many-to-many, but anonymous, so that senders do not need to know the identity of receivers, and this makes it straightforward to implement protocols such as invitating somone to a session using an opaque handle, as described in 6.1.

9.0 Conclusions

Several distributed virtual environments (DVE) we have investigated incorporate streaming audio between participants. A DVE requires internal protocols to maintain consistency between replicated copies of the object database used in the DVE, and the audio session model tends to be subsumed within the overall DVE structure.

Our approach is closer to the Internet multimedia community, in that we try to avoid making assumptions about an enclosing DVE, and extend the familiar multimedia session concept to include spatial audio rendering for multiple participants in a distributed environment. Integration between distributed spatial sound sessions, and multi-user visual environments based on Internet standards such as HTTP and VRML, results in sessions that are lightweight and triggered implicitly by criteria which could include spatial proximity and location.

References

[Beg94] Durand Begault, *3D Sound for Virtual Reality and Multimedia*, Academic Press 1994

[Bra97] Søren Brandt, Anders Kristensen, *Web Push as an Internet Notification Service*, W3C Push Workshop 1997, available at http://keryxsoft.hpl.hp.com/

[Curt94] Pavel Curtis, David A. Nichols, *MUDs Grow Up: Social Virtual Reality in the Real World*, COMPCON '94, available at ftp://ftp.lambda.moo.mud.org/pub/MOO/papers/

[Curt95] Pavel Curtis, Michael Dixon, Ron Frederick, David A. Nichols, *The Jupiter Audio/Video Architecture: Secure Multimedia in Network Places*, Proceedings of the ACM Multimedia '95, available at ftp://ftp.lambda.moo.mud.org/pub/MOO/papers/

[DirX] *Microsoft DirectX 5.0 Software Development Kit*, currently at http://www.microsoft.com/directx/resources/devdl.htm

[Green95] Chris Greenhalgh, Steve Benford, *MASSIVE: a Distributed Virtual Reality System Incorporating Spatial Trading*, IEEE 1995

[Green96] Chris Greenhalgh, *Spatial Scope and Multicast in Large Virtual Environments*, Technical Report NOTTCS-TR-96-7, The University of Nottingham

[Green96a] Chris Greenhalgh, **Dynamic, embodied multicast groups in MASSIVE-2**, Technical Report NOTTCS-TR-96-8, The University of Nottingham

[Hag96] Olof Hagsand, *Interactive Multiuser VEs in the DIVE System*, IEEE Multimedia, Spring 1996

[Hand95] Mark Handley, Ian Wakeman, *CCCP: Conference Control Channel Protocol - A scalable base for building conference control applications*, Proc. SIGCOMM '95, available at ftp://cs.ucl.ac.uk/mice/publications/cccp.ps.gz

[Hand97] M. Handley, J. Crowcroft, C. Bormann, J. Ott, *The Internet Multimedia Conferencing Architecture*, IETF DRAFT draft-ietf-mmusic-confarch-00.txt, available at http://www.ietf.org/ids.by.wg/mmusic.html

[Hand97a] Mark Handley, Van Jacobson, *SDP: Session Description Protocol*, Internet Draft 1997 draft-ietf-mmusic-sdp-04.ps available at http://www.ietf.org/ids.by.wg/mmusic.html

[Hand97b] Mark Handley, Henning Schulzrinne, Eve Schooler, *SIP: Session Initiation Protocol*, Internet Draft 1997 draft-ietf-mmusic-sip-04.ps available at http://www.ietf.org/ids.by.wg/mmusic.html

[Hard96] Vicky Hardman, Marcus Iken, *Enhanced Reality Audio in Interactive Networked Environments*, Proceedings of the Framework for Interactive Virtual Environments(FIVE) conference, December 1996, Pisa Italy. Also available at http://www-mice.cs.ucl.ac.uk/mice/rat/pub.html

[Int96] *Realistic 3D Sound Experience*, Intel RSX Software Development Kit, Intel Inc. 1996 available at http://developer.intel.com/ial/rsx/papers.htm

[Jacob93] Van Jacobson, Steve McCanne, Sally Floyd, *A Conferencing Architecture for Lightweight Sessions*, MICE seminar series 1993, available at ftp://cs.ucl.ac.uk/mice/seminars/931115_jacobson/

[Java3D] *Java 3D API Specification*, Version 1.0 1997, Sun Inc, available at http://java.sun.com/products/java-media/3D/forDevelopers/3Dguide/

[Lea96] Rodger Lea, Yasuaki Honda, Kouichi Matsuda, Olof Hagsand, Martin Stenius, *Issues in the design of a scalable shared virtual environment for the Internet*, 1996, available from http://www.csl.sony.co.jp/person/rodger/HICSS/hicss97.html

[Liv97] Living Worlds, *Making VRML 2.0 Worlds Interpersonal and Interoperable*, specifications available at http://www.livingworlds.com

[Mac94] Michael R. Macedonia, Donald P. Brutzman, *Mbone Provides Audio and Video Across the Internet*, IEEE Computer April 1994

[Rock97] Robert Rockwell, *An infrastructure for social software*, IEEE Spectrum March 1997

[Schu96] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, **RTP: A Transport Protocol for Real-Time Applications**, RFC 1889, 1996

[T.124] *Generic Conference Control*, ITU-T T.124, 1995

[Tel94] Gerard Tel, *Introduction to Distributed Algorithms*, Cambridge University Press, 1994

[VRML97] *The Virtual Reality Modeling Language*, ISO/IEC DIS 14772-1, April 97, available at http://www.vrml.org/Specifications/VRML97/

[Wat96] R. Waters, D. Anderson, J. Barrus, D. Brogan, M. Casey, S. McKeown, T. Nitta, I. Sterns, W. Yerazunis, *Diamond Park and Spline - A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability*, TR-96-02a, Mitsubishi Electric Research Lab 1996, availablc at http://www.merl.com/projects/dp/index.html

[Wat97] Richard C. Waters, David B. Anderson, *The Java Open Community Version 0.9 Application Program Interface*, Mitsubishi Electric Research Lab 1997, available at http://www.merl.com/opencom/opencom-java-api.html

[Wat97a] Richard C. Waters, David B. Anderson, Derek L. Schwenke, *The Interactive Sharing Transfer Protocol 1.0*, TR-97-10, Mitsubishi Electric Information Technology Center America 1997, available at http://www.meitca.com/opencom/

[Yer97] Bill Yerazunis, Barry Perlman, *High Level Overview of Open Community*, Mitsubishi Electric Information Technology Center America 1997

URLs

Aureal: www.aureal.com

Community Place: http://vs.spiw.com/vs/

Onlive!: www.onlive.com