A framework for interactive three-dimensional sound and spatial audio processing in a virtual environment

Michael Wozniewski

Master of Engineering

Department of Electrical & Computer Engineering

McGill University

Montreal, Quebec

August, 2006

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfilment of the requirements of the degree of Master of Engineering

ACKNOWLEDGEMENTS

I would like to start by acknowledging McGill University (particularly the Centre For Intelligent Machines) and La Société Des Arts Technologiques, which have both provided incredibly motivational workplaces to carry out this research. I would like to thank my supervisor, Jeremy R. Cooperstock, who provided me with interesting topics to investigate, valuable guidance, and the freedom to explore my interests. Marcelo Wanderley has been an incredible source of knowledge and inspiration. Pierre-Olivier Charlebois, who worked and contemplated over this project from conception, was essential to form the basis of the work. Yet this work could not have succeeded without Zack Settel, who offered years of experience related to music technology, and whose vision has guided the project from the start.

I would like to also acknowledge the generous support of NSERC and Canada Council for the Arts, which have funded the research through their New Media Initiative. I am also grateful to the developers and maintainers of the PureData and OpenSceneGraph projects. It is the freedom and communication of the open source software community that makes projects like this possible, and will allow this project to be freely distributed to artists and musicians who do not have the funds to develop such systems by themselves.

Lastly, incredible praise must be paid to my family and friends who have supported me throughout this venture. They have listened to countless sessions of seemingly nonsensical excitement, supported me during installations and performances, and put up with my disappearances during times of intense productivity. Hopefully you know who you are; thank you.

ABSTRACT

Immersive virtual environments offer the possibility of natural interaction within a virtual world that is familiar to users because it is based on everyday activity. The use of such environments for the representation and control of 3-D sound remains largely unexplored. We propose a novel paradigm for interacting with sound and using virtual space as the medium for spatial audio processing. A supporting software framework has been developed that provides functionality not available in other 3-D audio systems, including powerful control over the directivity of sound and the ability to bend the rules of physics for musical purposes. These features provide the necessary tools to create virtual scenes for audio engineering, musical creation, listening, and performance. Tracking technology allows the use of gesture-based interaction techniques to control the environment, resulting in many possibilities for novel applications.

ABRÉGÉ

Les environnements virtuels immersifs peuvent procurer une interaction naturelle au sein d'un monde virtuel qui soit familière aux usagers car ils sont basés sur des actions de tous les jours. L'utilisation de tels environnements pour la représentation et le contrôle de sons 3-D reste largement inexplorée. On propose un nouveau paradigme pour l'interaction avec le son et l'utilisation d'un espace virtuel comme instrument pour le traitement spatial du son. Une architecture logicielle procurant des fonctionnalité qui ne sont pas disponibles dans d'autres systèmes pour sons 3-D a été développée, offrant un contrôle puissant sur la direction du son et la capacité de déformer les règles de la physique dans des buts musicaux. Ces fonctionnalités procurent les outils nécessaires pour créer des scènes virtuelles pour l'ingénierie du son, la création, l'écoute et l'interprétation musicale. L'utilisation de technologies de suivis de mouvements permet l'utilisation de techniques naturelles basées sur les gestes pour le contrôle de l'environnement, ce qui ouvre la voie à plusieurs nouvelles applications.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

"If there's any object in human experience that's a precedent for what a computer should be, it's a musical instrument: a device where you can explore a huge range of possibilities through an interface that connects your mind and your body, allowing you to be emotionally authentic and expressive."

- Jaron Lanier (virtual reality pioneer, musician) [37]

The creative and artistic community has always been at the forefront of new technology, utilizing the newest tools available, and inspiring novel trends and practises. Artists today work far less with the paintbrush, canvas, and easel than with technologies like the mouse, keyboard, and perhaps the latest Macintosh computer. A vast majority of graphics, cinema, and music are digital in form, sometimes originating entirely in the processors of computers. The reasons for this are plentiful. Digital production saves costs, modifications are easier, and extremely powerful tools are being developed to facilitate the artistic process. For example, an animator no longer needs to draw a character frame-by-frame. He or she can simply specify a start and end position, and underlying models/constraints of body motion can render the appropriate animation. In audio, an engineer once needed to splice tape together in order to combine sounds, and patch cords were used to mix signals together. Today, digital audio workstations can manage these operations with a few clicks.

One area however, where multimedia technology struggles to help artists, is with real-time performance or interactive digital work of any kind. This is particularly evident when that interaction involves several modalities

and many participants. The control interfaces between artists, computers, and other digital devices are still quite inadequate. The typical artist or performer might want to vary several parameters at one time, at a very fast rate, with high precision. To achieve this, artists must have access to the state-of-the-art in sensing technology and computing power, but there are many other important and difficult considerations. Interfaces need to be as expressive as possible, yet with simple design that is easily understandable and usable. These requirements are often contradictory, since expressiveness is achieved with interactions that require substantial training to fully master. Consider, for example, that traditional musical instruments such as the violin can take a decade of study before one can become virtuosic in their use.

To provide rich and complex behaviours is however possible if interfaces take advantage of innate human abilities, and harness the skills that humans learn from their regular daily interactions. A person will know for instance, that objects fall according to gravity, that nearer objects will sound louder, that pointing a finger at an object can be interpreted as a selection mechanism, and so on. By designing control systems based on rules that exist in the physical world, users can perform complicated tasks without needing to memorize a new skill set. The expressiveness of their actions is thus already highly developed, and the problem then becomes an issue of how we can map those actions to interesting effects.

The research presented here is an investigation into a control system based on real-world physics, and can thus be classed together with other work in virtual reality technology. We allow users to create a virtual three-dimensional (3-D) environment, where their bodies are modelled in space, as are the positions of various artistic elements. This is an interactive and

perceptually engulfing experience that causes the users to feel like they are *inside* of the artwork or instrument. Rich 3-D graphics are available for display, and also provide a visual interface to help control the artwork. However, the primary focus of this research is the development of a paradigm for interaction with 3-D sound, and a method of accomplishing complicated audio processing using spatial relations.

With that in mind, this document begins by providing the reader with an overview of immersive virtual environments (Chapter 2). Specific attention is paid to the role of audio and also to the various technologies that are used to control these systems. We explain how a user's body and gestures can be tracked, allowing us to design the *natural* iteraction techniques mentioned above. Chapter 3 then reviews the physics of sound, including effects such as wave propagation, reverberation, and Doppler. Knowledge of these phenomena help us to design a physically modelled sound engine that is capable of simulating realistic audio for the environment. We motivate the design based on related work in the field, which is described in Chapter 4. Though the visual display is not reviewed in detail, readers should note that graphics play an essential role. We allow users to *see* the audio elements that they are controlling, providing the necessary feedback for fully natural interaction.

The paradigm itself requires five important features to be in place (elaborated upon in Chapter 5):

1. Use an *immersive environment* with merged physical/virtual geometry.
2. Rely heavily on *physical interaction* with sound in space.
3. Employ a *3-D control interface* based on familiar spatial activity.
4. Provide *generic 3-D audio nodes* that perform audio processing.
5. Allow for *rule-bending* of the acoustic model of sound propagation.

The first three have already been mentioned in some way, yet the fourth and fifth may appear confusing. These last two concepts differentiate our work from many others. We strive to go beyond the traditional 3-D audio APIs (DirectX, MPEG-4, OpenAL, X3D, etc.), and offer greater control to the user. With our framework, we provide generic audio nodes that can be used as either sound sources, listeners, or audio processing entities that arbitrarily transform sound at some location in space. It is also possible to distort the (perceptually accurate) model of sound propagation, providing results that may be more interesting to artists and musicians. For example, Doppler shift can be diminished or disabled to prevent detuning of pitched sound material in a tonal music context, and sound can be directed with a fine focus so that users can choose exactly where and how sound travels within their scenes.

In order to provide these features, a software framework has been developed that supports the rule-bending of physical models, and provides a node-based structure that can be used to construct virtual audio scenes. The framework provides acoustic rendering of the virtual scene based on the physical modelling of audio propagation, and visual rendering is accomplished using a powerful 3-D graphics engine. Chapter 6 will describe the implementation in detail. It should be noted that this framework has been introduced in previous publications [83, 84, 85] with collaborating authors Zack Settel and Jeremy R. Cooperstock. Settel in particular should be credited with the conception of the paradigm, yet much of the development, including the software architecture and user tracking research, has been the contribution of this thesis author.

Ultimately, we provide the tools for artists to create interactive environments that take the form of realistic scenes, with similar geometry

and behaviour to that of the real world. These scenes can act as *virtual instruments*, which transform sound based on spatial interactions. Manipulating nodes in space (moving, turning, etc.) becomes the replacement for operations such as mixing, patching, bussing, and panning, which are used in traditional sound studios. This ability to control signal processing with natural body motion and distorted physical modelling results in new metaphors for DSP (digital signal processing) design and the potential for novel sonic applications. Several such applications have been explored and are described in Chapter 8.

CHAPTER 2
Overview Of Immersive Virtual Environments

Virtual reality (VR) research has been a growing field of interest since at least 1965, when Sutherland proposed "The Ultimate Display" [77] for looking into a computer-generated world. This display would provide a user-centered perspective into the world through a variety of senses including vision, sound, kinesthetic feedback, and possibly even smell or taste. Ultimately, Sutherland even imagined a display that could control the existence of matter so that "a chair displayed in a room would be good enough to sit in". Though our technology has not yet realized all of his imagination, we have made significant progress in the modelling of *virtual environments* (VEs), where objects, their properties, and their behaviours can be described in 3-D. Additionally, many advances have been made to improve the display of these environments and the methods used to control them. Users can be perceptually immersed and use natural human body motion to interact with VR hardware. These *immersive virtual environments* (IVEs) lead to a suspension of disbelief, where users begin to ignore the medium in which the worlds are presented.

2.1 Graphical Display

VEs have traditionally received more attention in the graphical domain than any other. Researchers have created highly realistic 3D worlds, with compelling physical models that try to convince the viewer that what they are seeing could be real. Immersive display technologies such as the CAVE [26] have seen proliferation into VR research labs across the globe. Various

CAVE-like systems exist (X-Rooms [43], BNAVE [45], Chromium [41], and Blue-C [35] to name a few), which use rear-projection displays to surround the user, projecting on the floor and ceiling in addition to several walls. User position is tracked so that perspective correction can be computed for each screen. That is, since a user is not constrained to stand in a specific location within these spaces, his or her viewpoint does not necessarily lie on the normal axis of the projection for all screens. Position tracking must be used to compute the proper viewing frustums based on user position. This correction is known as *off-axis projection* [26].

2.2 Input Devices and Control Mechanisms

When users occupy immersive environments, they should feel as though they are actually part of the virtual world being projected. This means that interactions with the world must be as natural as possible, and traditional computer input devices like the mouse and keyboard have to be abandoned. For our framework, there are two main interactions that need to be supported. First, in order to render the sound field accurately, we need to know the location of the user's ears in space, or at least their head position and orientation. Second, we would like to offer the possibility of interacting with virtual objects using hand gestures. For example, a user should be able to point in the direction of an object, grab it, and apply some sort of modification. We will discuss how such interactions can be recognized in Chapter 7, yet first it is important to know what kind of input devices and sensing technologies exist. The low-level features that these devices report can be combined to detect gestures or other motions of significance.

2.2.1 Overview of Sensing Technologies

When considering sensors for human-machine interaction, one must take into account several factors. Each device will have different precision, resolution, latency, stability, sensitivity to ambient conditions, and ergonomic appeal. Furthermore, there are different types of characteristics to measure. Buxton [19] and Card [21] both provide taxonomies of input devices, identifying the characteristic being sensed (e.g., position, rotation, pressure) and the number of degrees of freedom (DoFs).[1] It is however difficult to discuss all sensor possibilities, so we will focus on the two challenges mentioned above: acquiring head position and orientation, and recognizing gestures such as pointing and grabbing.

One common approach for tracking body motion in IVEs is to use electromagnetic devices such as the *Liberty* from Polhemus [10] or the *Flock of Birds* from Ascension [1]. For these systems, sensors are worn on the body that report the magnetic field at a particular location in space. This information is sent to a reference station that has the ability to localize the sensors with six degrees of freedom (DoFs), namely, 3-D position (x,y,z) and 3-D orientation (ϕ, θ, ψ). These devices typically have very fast update rates (several hundred updates per second) with high accuracy (usually, within 2 or 3 millimetres in position, and 1 or 2 degrees in orientation). Additionally, they are not susceptible to occlusions, allowing an object to pass between the sensor and reference station without affecting the resulting measurement.

[1] A DoF represents one independent variable or dimension that the device can sense. For example, a typical mouse has two degrees of freedom (ignoring the buttons) because it reports both horizontal and vertical displacement.

These systems do however have a few drawbacks. Metallic objects cause interference with the electromagnetic field, which prohibits mounting these sensors on many musical instruments. The fact that they have to be worn by a user can diminish the sense of immersion since the user might feel uncomfortable. Furthermore, the sensors are too big and clumsy for adequate placement on numerous small objects such as fingertips.

Some more specialized sensors exist that can be used in various combinations. Accelerometers, gyroscopes, and compasses can all be used to infer orientation cues. For example, the natural acceleration due to gravity can be sensed by an accelerometer, and used to describe the tilt of the sensor with respect to the earth's surface. However, when worn by a person, the tilt information can become innacurate since human motions can exhibit greater velocity change than the natural acceleration of gravity. This is when a compass that measures the natural magnetic field of the earth can be added to make readings more realiable. The addition of gyroscopes, which measure the speed of angular change, reduces variability even more. Such combinations of sensors are available commercially, and provide very stable orientation information. One noteworthy example is the InertiaCube from Intersense [5], which has a resolution of 0.03° and provides readings at 180Hz.

There are several gloves that exist, which can capture hand shape with great accuracy. The CyberGlove from Immersion [4] captures 22 DoFs relating to the various finger flexions and abductions. Yet position and orientation are not measured, so additional sensors must be used. We note at this point that gloves can be made using combinations of simpler sensors. In fact, as part of this research, a pair of prototype gloves were equipped

with accelerometers, flexion sensors, and force-sensing resistors (FSRs). See Appendix C for a discussion of those prototype gloves.

Cameras offer a less obtrusive style of sensing since nothing needs to be worn by the user. However, achieving low latency data and highly accurate results is not easy. In fact, the only method to reproduce the accuracy of magnetic tracking devices is to wear passive tracking markers on the body for improved recognition. Passive markers imply that no hardware is present on the body; these are simply a specialized material that is easily identified by infrared cameras. The Vicon MX motion capture systems [11] are the industry leader in this type of technology. The cameras can operate at fast as 2000 frames per second (fps), and capture resolutions of up to 4 megapixels. The expense of these systems, along with calibration requirements make this solution less appealing for our purposes. We are, in the end, attempting to provide a generic framework for use by artists. We will therefore explore the use of generic consumer-grade video cameras.

2.2.2 Computer-Vision Based Tracking

Contrary to the methods presented above, video cameras offer sensing of human motion without attaching anything to the user. The difficulty however, is that each frame of the video stream contains additional information of no use, and must be processed to extract the body. This is a challenging task since the human body can exhibit almost arbitrary shape making segmentation difficult, especially against a cluttered or dynamic background. The requirement that all processing be done in real-time further complicates the problem. One simplification that we make is to ignore some parts of the body, and focus only on methods for finding the hands and head.

Tracking human hand position is often performed in order to recognize what is called a *diectic* gesture, which simply means inferring a pointing direction. This problem has been approached by a few researchers. The Pfinder project [86] for example, tracks a human figure in a video sequence to control an interactive application. The algorithm works by finding connected areas of similar colour (blobs) and computing the mean (i.e., centroid position), and the lower-order statistics (i.e., variance in the x and y directions). Together, the variances provide an orientation vector for the blob. Heuristics based on Gestalt principles are used to combine several blobs into a model representing the human user. The blobs are also tracked over time using a Kalman filter, which helps to reduce uncertainty in both the feature extraction and blob-forming processes. This work has been extended by Azarbayejani et al [14] to use a stereo camera setup (sPfinder), which computes low-order statistics of 3-D blobs, and thus produces orientations in three dimensions.

Figure 2–1: The Pfinder system [86]: The left image shows the initial camera view, the middle image shows the segmented user (based on background removal), and the right image shows a rendering of the blobs that make up the tracked human figure.

Jojic et al [47] also work with a stereo camera setup, which they use to build disparity maps then remove the background to leave only pixels that likely represent a person in the environment. These pixels are then modelled using a mixture of Gaussians: one roughly representing the body, and one

for the hand. A statistical method based on expectation maximization is then employed to converge to the best estimation of positions (and variances) for the Gaussian model using heuristics based on the knowledge of human appearance. When the final estimation of the mixture model is returned, the pointing direction can then be found.

One other method worth mentioning uses *proximity spaces* (PS's). Kortenkamp et al [51] use Nishihara's stereo matching algorithm called PRISM [60], which attempts to discover occupied regions of volumetric space. Filled volumes are divided into PS's, where each PS is modelled as a node in a kinematic model of a human. They then define laws that govern the possible relative positions of each PS and their allowed trajectories. For example, PS1 in Figure 2–2 is biased to be higher than PS2, and is connected by a joint that has a preferred angle.

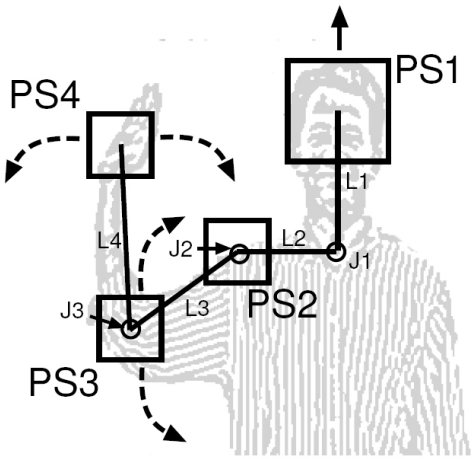


Figure 2–2: 3D model of human built with proximity spaces ([51]).

Once a diectic vector is established, we can then attempt to detect other events with cameras, such as grasping and releasing. Again, this is a difficult problem because hand shape is highly variable, and adequate views cannot always be obtained. However, some researchers have managed to explicitly model the 3-D shape of the hand. For example, Rehg and Kanade

propose *DigitEyes* [70], which models the 27 DoFs of the hand using line features extracted from grey-scale images. Stenger et al. [76] build a model with 37 truncated quadrics that are joined together, and can eventually be used to construct a 3D view of the hand. It is doubtful, however, that these methods can be reliably used in real time by our system. The latency due to continuous image processing and the large computational demand for 3D modelling adds a significant overhead to the system. We will likely use sensors or gloves that are directly attached to the hands to recognize grasping, releasing, and any other simple hand postures.

2.3 The Role of Audio

There has been an increased amount of attention from the audio community attempting to create the same feeling of perceptual immersion found in current visual displays for VEs. Several advances have been made toward the realistic modelling of acoustics, including simulation of environmental effects such as reverberation, reflection, absorption, occlusions, and diffraction. These efforts are starting to be incorporated into several standards, toolkits, and application program interfaces (APIs). This is good news for the developers of simulations and games, who are the target users of these resources. Yet audio for artistic purposes requires a slightly different focus.

Pressing [64] suggests that VEs use audio in roughly three independent ways: information transfer, environmental effects, and artistic expression. This seems to be a natural classification, since there is at least some evidence that our brain processes audio in this manner [62]. Furthermore, current techniques of film production tend to separate audio tracks into music, dialogue, and sound effects. Our framework, though concerned mostly with the topic of artistic expression, must allow for these three types

of audio in some form. This classification will be explored in more detail in the following sections.

2.3.1 Informative Audio

Informative audio includes all types of speech, alerts, audio cues, and other types of notifications. In the context of a virtual world, this may be a voice captured from another participant, generated speech (from artificial agents), pre-recorded audio clips, or synthesized sounds that convey some information. Generally, these types of audio signals are diegetic,[2] and should be *spatialized*, so that their presumed origins in the scene are perceived to come from that direction. However, we would also like to provide the ability to display non-diegetic sounds, for things like audio alerts and notifications. Hence, there needs to be a way to create audio without localizing it in the scene. This is one area where *bending* the rules of physics comes into play. If we had the ability to disable the acoustic modelling of sound, we could provide audio signals that are dissociated from the virtual world.

2.3.2 Environmental Effects

This category includes all audio that places the user within some setting, including sound effects, background music, and auditory ambiance. Typically, these types of sounds are less interactive in nature, and would be used for applications such as video games or simulations to set the "proper mood". We focus on this category the least since such functionality will

[2] The term, *diegetic sound*, is common in the film industry and describes sound that is presumed to be localized at some specific position in the scene. If a character in a movie plays an instrument, this is diegetic, whereas omni-present background music (that the character is unaware of) is non-diegetic.

easily be attainable by implementing the more complicated requirements of the other categories.

However, it should be noted that to depict an environment accurately to the user, all diegetic sound should be processed for environmental acoustics. In nature, the size, shape, and the types of objects within an environment will affect the propagation of sound. Our brains are remarkably adept at uncovering this hidden information, and inferring properties about our surroundings. There are many methods for applying acoustics and reverberation to audio signals. We will discuss this in detail throughout Chapters 3 and 4.

2.3.3 Artistic Audio and Music

Music and artistic audio have only recently been explored within the context of virtual environments, and resulting applications typically focus on virtual simulation of traditional musical instruments rather than navigable 3-D spaces. Perhaps the earliest example, MusicWorld at MIT [31], allowed a user to play a virtual drum set using Polhemus sensors. More recently, Maki-Patola et al [56] have created a virtual xylophone, virtual air guitar, and a virtual synthesizer modelled after the Theremin. Each uses gestures similar to it's real-world counterpart. However their most interesting work is perhaps a virtual membrane that is played with a mallet-like controller. What makes it particularly insightful is the fact that sound is produced as a result of physically modelled oscillations of real-world materials such as leather, wood, and metal. The instrument is hence 'played' by manipulating physical properties such as the dimensions, tension, and damping of the membrane.

This type of modelling, where sonification is inspired by the physical world has also been shown to be effective by Mulder and Fels [58]. They

used a technique called *sound sculpting* [57], where the shape, position, and orientation of virtual objects was used to control sound synthesis. The user would manipulate objects resembling rubber balloons or rubber sheets using two-handed gestures. In the example that the authors provide, the position of the object influences panning, the orientation controls volume, and the overall shape changes the timbre of the sound.

We take inspiration from these ideas, and use the organization of physically modelled objects to influence the resulting sound. Yet rather than considering one virtual object as the instrument, we consider the entire virtual scene as the instrument. This results in more flexible techniques, since interactions between objects can create new sonic effects, and the propagation of audio between objects can be used as another aspect of control.

CHAPTER 3
Understanding the Physics of Sound

The research presented here is largely focused on modelling the behaviour of sound in a virtual environment according to the same laws that exist in nature. It is thus important to have a reliable understanding of the physics behind sound propagation. In this chapter, we will explain several terms related to sound and acoustics.

3.1 Properties of Sound

Sound is a repetitive wave of compressions and rarefactions in a medium (typically that medium is air). The *frequency* can be determined by counting the number of oscillations of the medium per second, and the *amplitude* corresponds to the displacement of the medium. This means that sound exhibits all wave-like phenomena including reflection, refraction, and diffraction. While we assume the reader has some prior knowledge regarding these properties, we provide below a brief introduction to the more important sound behaviours that are referenced throughout this document.

3.1.1 Reflections and Diffraction

The audible wavelength of sound for humans is between 0.02 and 17 metres (for frequencies of 10KHz and 20Hz respectively). In this range, humans will perceive different effects for different frequencies of sound. The reason for this is that reflections only occur when an object is much larger than the wavelength of the sound, while diffraction occurs when the wavelength and object size are roughly the same. Additionally, objects

that are much smaller than the wavelength will not affect the propagation of audio. The numerous reflections that occur in a scene will combine at the listener, which collectively form the acoustic phenomenon we call *reverberation*. These reflections or echoes of the original sound will provide the user with information about the current configuration of the scene.

Since many powerful tools exist for graphical rendering of virtual environments, it may seem like a good idea to use existing models of illumination to render virtual sound. This, however, is not the case. In contrast to 3-D graphics, audio must be computed behind objects and around corners since diffraction allows sound to bend around these entities. Furthermore, the amount of diffraction depends on frequency, with low frequencies (LF) more affected than high frequencies (HF). Thus, we cannot use culling algorithms of computer graphics to ignore parts of the scene that are hidden from the sound source. On the other hand, we may be able to ignore various small objects in the scene since most frequencies will not be affected by their presence. Thus, the computation of audio only requires a coarse representation compared to that of computer graphics.

3.1.2 Temporal Effects

The speed of sound in air is roughly 343 metres per second, which is much slower than light. The propagation time of sound is thus perceptible by humans in most situations. In fact, humans have developed the ability to differentiate sounds with time-of-arrival differences of as little as 7μs [52], which makes timing requirements of acoustic simulation very important. Contrary to 3D graphics, where the travel time of light is usually ignored, audio must be presented to the user with precise timing. Additionally, phase information is important for sound, and has to be taken into account when summing signals that arrive at the listener. Most light sources (with the

exception of lasers) are incoherent waves, allowing their energies simply to be summed together, while sound is coherent, allowing for interference between waves.

3.1.3 Attenuation With Distance

Sound travels through the air as a spherical wavefront, which means that the initial sound source energy spreads over an increasing area as it propagates. In fact, the energy will decrease according to the inverse square law [73] since the area of the sphere of propagation is proportional to the square of its radius. However, scattering and absorption of the wave will also occur due to various particles in air that interact with the sound. Scattering typically affects high frequencies while absorption affects low frequencies.

3.1.4 The Doppler Effect

The Doppler effect (or Doppler shift) is a perceived change of frequency as a sound source and listener move toward or away from each other. When the source is approaching the listener, the emitting sound waves are compressing against previously emitted waves, which results in a higher perceived frequency. Conversely, lower frequencies are perceived when the source is moving away from the listener.

3.1.5 Reverberation

When we consider the combined effect of all sound reflections in a space (a room, concert hall, cave, etc.), we call this *reverberation*. Figure 3–1 shows the simple example of a rectangular room with nothing inside except a theoretical sound source and a theoretical listener.

The direct signal will obviously arrive at the listener first, since the reflections have a greater distance to travel. The sound waves that reflect off the walls and arrive at the listener just after the direct sound are called

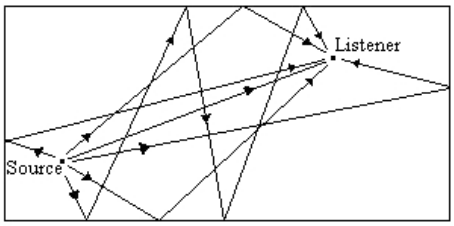


Figure 3–1: Reverberation in a simple room, showing reflection of sound off walls as it travels from source to listener.

1st order reflections. The delay time of these reflections gives the listener some indication of the general size of the room. 2nd order reflections are usually still distinguishable by a listener but after a few bounces, the sound builds up in density and is simply referred to as *reverberant sound*. The time required for this reverberant sound to decay away to 1/1000 of its original amplitude (-60dB) is referred to as the *reverberation time*.

One additional consideration to factor in is the frequency dependence of this decay. Since low frequencies reflect better, they usually decay slower, and each room affects different frequencies in different ways. The materials present in the room can absorb sound to greater extents (e.g., fabrics) than others (e.g., brick). Hence after each reflection, the frequency content of the sound heard will be quite different, and will continue to evolve over time.

All of these features are important psychoacoustic cues that notify a listener about the properties of their environment. Typical modelling of reverberation will hence require the specification of delay times for direct sound, as well as the 1st and 2nd order reflections, the reverberation time, and some filter to simulate the frequency-dependent damping over time.

3.2 Sound Localization

In the late 19th century, Lord Rayleigh performed a series of experiments to explain how humans localize sounds in space[1] [73]. He discovered that the use of two ears (binaural hearing) meant that each ear received a slightly different audio signal. The human head casts a *sound shadow* that causes sounds originating from one side of the head to have a lower intensity on the other side. Humans thus perceive an *interaural level difference* (ILD) that they can use to identify the direction of the sound. Likewise, *interaural timing differences* (ITD) result when sound arrives at one ear slightly before the other.

It should be noted that the ILD and ITD only allow listeners to localize sounds on the azimuthal (horizontal) plane. With small head movements, humans are capable of inferring the elevation as well, but this information is more accurately gathered using a different mechanism: analyzing changes in the frequency composition of sound due to interactions with the human body. Certain frequencies reflect from our shoulders, others diffract around our torso and head, while others are influenced by pinnae (lobes) of our ears. With all these cues, there is a different *spectral colouring* of sounds coming from different heights and directions. In fact, the ILD is also influenced by these spectral effects. Particularly, since low frequencies tend to diffract around objects, they are not shadowed by the head and will hence not result in any noticeable level difference. The *Duplex Theory* of sound localization resulting from Rayleigh's experiments formalizes this principle, and states

[1] Though Rayleigh is widely accepted as the first to explain binaural hearing, similar experiments were performed almost 100 years earlier by Italian scientist Giovanni Venturi.

that humans rely on ILD for sounds above 4KHz while only ITD is used for sounds below 1KHz.

The Duplex Theory and the reverberation cues mentioned in the previous section are the strongest stimuli that allow humans to perceive the direction of a sound source. These factors will thus need to be considered when modelling audio in a virtual environment, which will be discussed in more detail in Chapter 4.

CHAPTER 4
Models and Techniques for Audio in Virtual Environments

In the context of audio for virtual environments, the goal is to model the sound properties described in Chapter 3 using virtual models. A majority of research in this area [15, 75, 59, 48] is concerned with the challenge of presenting sounds to a listener so that they may localize their origins in 3-D space. This problem is known as "audio spatialization", "audio rendering", or "sound imaging", and can be solved in many ways. The techniques tend to differ based on the auditory display being used (i.e., the number of loudspeakers available and their positions relative to the listener). However, these methods only deal with direct sound, which is assumed to travel from the source to the listener without encountering any obstacles. If objects are in the path of the sound wave, it means that reflections, absorption, and diffraction must be modelled as well. This is a markedly smaller area of research, which is often referred to as "virtual acoustics", or "acoustic rendering". In this chapter, we will introduce the most common techniques, and describe how these methods have been adopted by software toolkits.

4.1 Auditory Display Techniques

Given a sound source in 3-D space and listener location, we need to control the output of physical (real-world) loudspeakers such that the emitted audio is perceived to originate in the direction of the various virtual sound sources. To accomplish this, each loudspeaker signal will need to be adjusted to account for the loudspeaker placement. The methods to

accomplish this fall into three main classes: binaural methods, amplitude differencing, and wave field synthesis (WFS).

4.1.1 Binaural Display

Binaural methods aim to reproduce the ILD, ITD, and spectral colouring mentioned in Section 3.2. A sound source position is usually defined in spherical coordinates (ϕ, ψ), which respectively denote the angles of elevation and azimuth to the source. One can imagine that for each (ϕ, ψ) combination, two filters can be defined that simulate the modifications to the two signals received by the ears. The combination of these filters (for every possible set of angles) is referred to as the *head response transfer function* (HRTF). If the HRTF is known for a particular individual, then signals for each ear can be computed to simulate a sound source coming from any direction.

This transfer function is however difficult to express mathematically due to the complexity of head shape, and the fact that every human head is different. Some lower level approximation must be made in order to make the computation possible. For example, the head can be modelled as a rigid sphere, allowing for the use of algebraic diffraction equations [69]. Otherwise, a statistical sampling of recorded signals can be used to generate the HRTF. The typical approach is to use a dummy head equipped with special microphones and empirically measure the signal arriving at each ear location for a variety of positioned audio pulses [33]. The Fourier transform of the resulting set of impulse responses provides the HRTF.

This way, one can spatialize a sound simply by filtering two audio signals and rendering them through headphones, or through a stereo speaker setup. Using headphones is obviously the best display method, since the audio signals are isolated to each ear and crosstalk cannot not occur. There

are many methods for crosstalk cancellation [32], but these require that the user remain motionless, otherwise the perceptual effect is ruined.

4.1.2 Amplitude Differencing

The consumer-grade systems that many gamers and cinemaphiles have deployed in their homes use several speakers arranged on a horizontal plane around the user. These display systems are typically called "surround sound", with some standards identifying the number of speakers used. For example, "5.1 channel surround sound" means five specially positioned speakers plus one sub bass speaker, which may be placed anywhere since the low frequencies it produces have poor directional cues. The spatialization is however only *pantophonic* (localizing sounds only in one plane). Previous research [34, 66] has made *periphonic* (fully 3-D) spatialization possible, using speakers distributed both horizontally and vertically around the user. The speakers are usually required to be located equidistantly from the one central point, called the *sweet spot*.

Regardless of the speaker distribution, amplitude differences (and sometimes timing differences) between loudspeakers are used to make sounds appear from a certain direction. Ambisonics [34] for example, encodes audio signals with directional components (x, y, z, w), where w is the gain in a direction specified by the vector (x, y, z). Decoders exist that reproduce the appropriate loudspeaker signals by taking a linear combination of these four channels.

Vector base amplitude panning (VBAP) [66] is a similar method to ambisonics. Speakers are grouped into triplets and a sound source is rendered using only those three speakers rather than the whole array. Figure 4–1 shows such a triplet of speakers and the direction vector, $\mathbf{p}$.

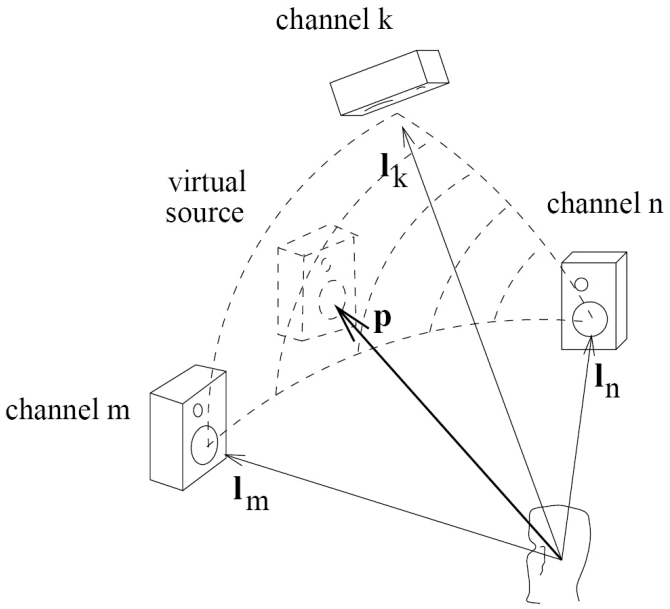


Figure 4–1: VBAP: Vector Base Amplitude Panning [66].

This direction vector can be expressed algabaically with:

$$\mathbf{p} = g_m \mathbf{l_m} + g_n \mathbf{l_n} + g_k \mathbf{l_k} \tag{4.1}$$

where g_m, g_n, and g_k are the gain factors for each speaker and can be solved using simple linear algebra (as long as the speaker positions are known).

The VBAP method allows for the real-time spatialization of sound sources with arbitrary speaker arrangements, yet ideally there should be as many speakers as possible, arranged uniformly around the user. The reason for this is that a perceptual problem occurs when a sound source is located exactly in the middle of a speaker triplet. The amplitude of each speaker will be equal, and though the listener will localize the source in the middle of the triplet, the perceived source will be larger (or more spread out) than when the source is located closer to one of the loudspeakers. This *virtual source spreading* problem can be solved simply by increasing the number of

speakers, or by spreading a source artificially as it becomes aligned with a speaker [67].

Ambisonics, on the other hand, is meant for precisely four, six, or eight symmetrically positioned speakers. Some new methods [30, 28] have provided decoding for standard surround configurations (5.1 channel, etc.), yet this is more of a convenience and deteriorates localization. Furthermore, additional speakers do not improve the quality of ambisonic spatialization as they do with VBAP [68].

4.1.3 Wave Field Synthesis (WFS)

Contrary to amplitude panning techniques, where the sound field is accurately produced only in the sweet spot, WFS attempts to re-create the sound field throughout an entire volumetric space. This requires an array of dozens of small speakers and is computationally expensive compared with the techniques mentioned above. The concept is based on Huygen's principle (1678), which states that a wavefront can be seen as the composition of elementary waves, each of which propagates with the same velocity and wavelength as the original wave. In WFS, each small loudspeaker is responsible for generating one of these elementary waves, which combines with others to re-create the true wavefront of particular sound experience.

The sound field may not be completely accurate in any one spot, since this would require the use of an infinite number of speakers, but the error field is minimized and uniformly distributed. This allows a listener to move about the space without loss of perceptual auditory immersion. That is, there is no sweet spot for WFS systems, and multiple users can even share the same space with accurate sound rendering for each person.

4.2 Virtual Acoustic Effects

So far, the auditory display techniques that have been described allow for the rendering of a virtual sound source so that it may be localized in 3-D space. This however, assumes that the sound waves will not encounter any virtual objects en route to the listener, which is often not the case. In fact, even if there are no objects in between the source and listener, there will likely be reflections from far away objects (or walls) that the listener would hear in the real world. To take into account all of these factors would mean the full modelling of reverberation.

There have only been a few projects that attempt to model these acoustic effects in high detail. For example, the Digital Interactive Virtual Acoustics (DIVA) project [55] uses ray casting (similar to methods in 3-D graphics) to compute reflection paths of audio signals. Additional virtual sound sources are created at the reflection points, and by modelling surfaces with absorption parameters, the authors achieve a realistic response of the environment. Tsingos et al [79] augment this technique by also modelling the diffraction of sound around the edges of 3-D objects using the uniform theory of diffraction (UTD).

If ray casting is too computationally expensive for a particular application, there are simplifications that one can employ. For example, environmental audio extensions (EAX) [2] can provide reverberation effects given only a simple description of a room. This can include the width, height, depth, and perhaps a description of the materials present (e.g., hardwood, carpet, concrete). Based on these parameters, a filter is applied to the audio signal, which creates the appropriate delays and attenuations to simulate the reflections of such a room. EAX also supports the reverberation modifiers described in Table 4–1 (occlusion, exclusion, and obstruction).

Occlusion	The source and listener are separated by a wall. Depending on the material and thickness, some sound energy will transfer through the wall. but both direct and reverberant sound will be severely attenuated.
Exclusion	The source and listener are in separate environments, but the path from the source to listener is not blocked. For example, the listener may be able to see the source through a window or doorway. In this case, the direct sound is unaffected, but the reverberant sound is attenuated depending on the size of the opening.
Obstruction	The source and listener are in the same environment, but the path from the source to the listener is blocked. The direct sound in this case is attenuated, yet reverberant sound is left unaffected.

Table 4–1: Reverberation modifiers (EAX).

These modifiers provide a simple mechanism to simulate more complicated reverberation effects simply by adjusting the gain of the direct and reverberant sound components.

4.3 Available Toolkits and APIs

In the case of real-time applications such as games and virtual reality systems, several toolkits and APIs are available, including Microsoft DirectX [7], OpenAL [8], X3D [3], and MPEG-4 [74]. These toolkits allow developers to move sound sources around a virtual scene interactively and have them properly rendered. Unfortunately, these toolkits are not sophisticated enough for highly interactive sound applications, particularly for those geared towards artists and musicians. They offer only simple mechanisms to integrate spatial audio into applications, and often have an impoverished audio representation.

For instance, most APIs have no method to specify the directivity of sounds and instead consider all sources as omni-directional. In the cases where directional sounds are supported, these are typically implemented

with linear attenuation applied as a function of angle. For example, the directivity of sound sources in X3D is modelled with two ellipsoids (as shown in Figure 4–2). The inner ellipsoid indicates the commencement of attenuation based on distance, which proceeds linearly until full attenuation occurs at the outer ellipsoid. There is no support for complex radiation patterns that are emitted by traditional musical instruments, or the cardioids that are commonly found in audio equipment.

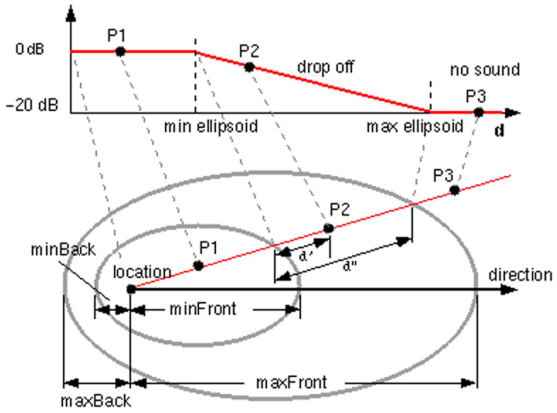


Figure 4–2: The X3D sound node model [3].

Furthermore, most APIs only support a single listener located at one position in space, and lack the possibility to perform *unnatural* sound propagation. In general, it is difficult to manage more complex sonic interactions between objects that may be important to artists. For example, it might be interesting to copy signals to different areas of the scene, attach two sets of headphones to a soundcard for two different users, or boost the Doppler Effect for dramatic purposes.

One noteworthy standard for describing interactive audio scenes is AudioBIFS from MPEG-4 (ISO 14496-1) [74]. BIFS, which stands for Binary Format for Scene Description, is in an extension of VRML and X3D, but with a focus on describing audiovisual scenes in a compact and object-oriented fashion that ultimately leads to streamable interactive content. The

representation of 3-D audio is quite interesting and serves as inspiration for the work presented here. AudioBIFS borrows the scene-graph hierarchy from VRML to organize nodes, but introduces specialized audio processing nodes such as “Sound”, “AudioSource”, “AudioFX”, and “ListeningPoint”. These can be attached to graphical objects anywhere within the scene, allowing developers to create scenes for interactive mixing and musical control within a 3-D context.

The work presented in this document uses an open-source library called OpenSceneGraph (OSG) [9] to manage the arrangement of objects in space and efficiently control the geometric transformations performed on them. The main difference between our system and AudioBIFS, is that we provide the ability for dynamic control and reconfiguration of the entire scene in real-time. BIFS on the other hand are binary in format and cannot change dynamically. The scene must be authored in advance and then compiled for distribution. Interactivity is accomplished by allowing certain fields to be *exposed* and updated by a content server. The developer must pre-define all possible interactions during the authoring phase. In our system, there is no differentiation between an authoring mode or interactive mode; they are one and the same.

CHAPTER 5
A Novel Paradigm for Spatial Control of DSP

The techniques described in Chapter 4 are mainly used for realistic simulations of 3-D audio. That however, is not the focus of the work presented here. We instead try to use the spatial arrangement of sound as a control interface for creating music or artistic sonic material. Realistic audio propagation is only a starting point, and users can choose to depart from that as desired. As far as we know, this is a novel paradigm for dealing with sound. The features that we introduced earlier encapsulate this new way of thinking, and will be expanded upon throughout this chapter:

1. Use an *immersive environment* with merged physical/virtual geometry.
2. Rely heavily on *physical interaction* with sound in space.
3. Employ a *3-D control interface* based on familiar spatial activity.
4. Provide *generic 3-D audio nodes* that perform audio processing.
5. Allow for *rule-bending* of the acoustic model of sound propagation.

5.1 Immersion

In order to deploy this new method of audio control, the concept of immersion is fundamental. We track the user's physical position and model his or her actions in virtual space. In fact, several aspects of the user's world such as speakers, microphones, walls, and furniture can be included in the virtual representation. The geometry between the physical and virtual worlds is merged so that all spatial relations are kept intact. An object one metre away in the real world will appear one metre away in the virtual world. This merging is often called *augmented reality* (AR) by the research community, where real-world objects are registered in 3-D, and presented to

the user in a modified way. For example, an AR system might use a camera to show a user the world in front of him/her (via a head-mounted display), yet parts of the video scene have been identified and labelled. Our proposed system could be used in this sense, except since we focus on audio instead of video and would tend to label the scene with auditory markers.

The key factor for our design is that every component of the real world auditory display (loudspeakers, microphones, etc.) should should be represented in the virtual 3-D world. These components may be displayed either visually, sonically, or typically both. Hence, a user who is *inside* of this world is offered a literal rendering of the audio scene and is provided with direct and real-time feedback of the current state of the application.

It should be noted that accurate tracking of the user is needed to realize this system. The user's head position and orientation must be passed to the engine in order to achieve accurate rendering. Particularly important are physical audio devices such as headphones or microphones that are worn by the user. If other components in the real world are moving and have some importance in the audio scene, then they too should be registered and updated in the engine.

5.2 Physical Interaction With Sound

With this sense of immersion, interaction with the virtual world becomes very *physical*. In fact, this physicality can convey two separate ideas. First, we are dealing with a notion of direct physical activity by the human body to affect sounds. Rather than creating layers of abstraction through sliders, pedals, or other controllers, we would like artists and musicians to use natural human motion to control audio processing. This means that part of our system will include tracking of the human body, and

we will map the user's posture and gestures to meaningful outcomes in the virtual world.

The second concept related to physicality is then the notion of modelling real-world physics. We want as much as possible to model things such as the kinematics of moving entities, and the physical properties of sound propagation. Users are familiar with these phenomena and will hence need less of a learning curve to understand the possible interactions in the virtual world.

5.3 3-D Control Interface

To facilitate learning and ease-of-use, it is important that all interaction with the environment be achieved through truly three-dimensional spatial activity (moving, turning, pointing, grabbing, etc.). For example, rather than controlling the level of an audio signal with a slider, a user could simply reach out, grab an object, and move it closer or further away. Such an action effectively controls the gain of an audio signal, since sound naturally decays with distance. Likewise, by moving around the space and orienting one's head in different directions, a user can change the mix of signals arriving at their destination. When a scene is composed of spatially positioned sound objects that coexist with the user, the progression of a musical piece is achieved simply by travelling through virtual space. These control concepts are easy to learn since humans use similar interactions in their everyday world. We thus build on existing knowledge of human spatial understanding.

5.4 Generic 3-D Audio Nodes

The most important concept that differentiates our work from others is perhaps the generalization of all sound-related objects in the virtual

environment. Particularly, we do not use the notion of *listener* versus *sound source* that is typically found in the literature. Instead, all sound-related objects in a scene are referred to as *soundNodes*. Given the appropriate parameters, these can portray a listener, sound source, or any arbitrary DSP processor. The reason for this is that a soundNode can act as a *source* (emitting sound), a *sink* (absorbing sound), or both at the same time. The latter case is particularly important in the context of musical systems since we can allow a node to absorb sound, apply DSP to that captured audio, and then emit the result back into the scene.

This ability to apply DSP at specific 3-D locations is not often included in spatial audio systems, and allows for many novel applications to be designed. One can build a "spatial instrument" using a number of sound generators and filters scattered throughout a virtual scene. The audio signals can be generated at particular locations in space, then travel successively through various filters or processors. The result can be captured at multiple locations in space, and relayed to additional filters, or simply sent to loudspeakers so that users can hear the mix.

It is necessary to understand that we are using 3-D space as the medium for signal flow in a DSP application. Instead of using patch cables or wires to connect different DSP devices, we are using the continuous flow of sound through 3-D space. Unlike conventional analog or digital audio systems, there are no logical connections between different components, and no knobs or sliders to control levels. Rather, audio signals are processed (attenuated, delayed and filtered) according to the laws of physics. Manipulating the relative distance and orientation between soundNodes becomes the method of controlling audio processing. Thus, the principal operators

for mixing (e.g. bussing, panning, filtering) become spatial in nature (e.g. translation, rotation, scaling).

This, of course, works better for some audio operations than others. Gain, for example, is most naturally controlled by increasing or decreasing the distance between nodes. Filter coefficients, on the other hand, seem to be controlled best by the rotation of a node. There are many audio processing operations that do not lend themselves to such a physical analogy. For example, adjusting the timbre of a sound is a difficult operation, often involving several parameter manipulations at once. However, using spatial mappings as a starting point for building novel musical controllers will hopefully reduce the effort required to master simple operations, and users can concentrate on the more complicated interactions.

5.5 "Bending" Reality

Earlier we mentioned that we rely on the physics of sound propagation to serve as the medium for interaction within the virtual 3-D space. This includes simulation of sound decay with distance and angle, diffraction, reverberation, and Doppler shift. However, for the purpose of musical creation and performance, the user (or rather, musician) may not desire truly accurate models of sound propagation. Rather, he/she may wish to exaggerate or diminish certain acoustic properties for artistic purposes. For example, the Doppler effect is often emphasized in sound tracks for cinema because it adds dramatic effect. Also, to help manage timing in musical pieces, perhaps the delay that occurs when sound travels a certain distance would not be desired.

One other important feature is to allow for the bending of reality only for specific purposes, and not globally for the entire scene. For example, some nodes may be used for effects and should exhibit the Doppler effect,

while other nodes may be used to managed pitched sound material. In this latter case, the Doppler effect should probably be disabled so that a frequency shift does not occur, and the tonal quality of the music is maintained.

CHAPTER 6
The Software Framework

Motivated by the spatial DSP paradigm described in Chapter 5, a new software and hardware framework was developed in order to satisfy the level of control required. The traditional methods, toolkits, and APIs described in Chapter 4 do offer control mechanisms for interactivity, yet there are many limitations in their functionality. For example, though sounds can be moved individually in real-time, their acoustic parameters (directivity pattern, roll-off, etc.) often cannot be manipulated dynamically. Rather, these parameters remain fixed and are usually pre-defined for the entire scene. Furthermore, virtual sound sources tend to only represent external audio streams (sound files, line-in, etc.) rather than signal processing objects. It is fundamental to our paradigm that a soundNode be able to collect sound, process it, and re-emit the result into the scene. Lastly, the ability to apply *rule-bending* to the laws of physics is not supported by traditional APIs, and we assert that this is an important feature to allow for artistic and musical creation.

In this chapter we introduce the software framework that has been designed to support these additional features. It is written in C++, relying heavily on the use of the PureData programming language, and the OpenSceneGraph graphics toolkit. The codebase is quite large, so only the most important classes will be described here. Particularly, we will discuss the classes: *soundEngine*, *soundNode*, and *soundConnection*. These are the essential pieces that realize our vision.

6.1 Preface: About PureData and OpenSceneGraph

Before we begin to describe the framework that was developed, it is important to introduce the PureData (Pd) patcher-based programming language. This is a real-time graphical programming environment designed for control of audio processing. It resembles the Max/MSP system [6] but has an open-source license and is intended to be simpler and more portable. There is a large community of music-focused programmers, who are continually developing new DSP and control methods using this language. Hence, we believe that it is the ideal candidate to avail a powerful tool to artists in an extensible fashion. Readers who are not familiar with the structure of Pd should read Appendix A, since we will refer to related terms such as *patches*, *abstractions*, and *externals* throughout the rest of this document.

OpenSceneGraph (OSG) on the other hand, is a graphics toolkit that provides a high-level interface to OpenGL. Rather than writing and optimizing low-level graphics calls, OSG allows a developer to concentrate on the organization and interaction of 3-D content. We foresee that a typical scene will be composed of several independent 3-D models such as people, musical instruments, audio equipment, and architectural elements such as furniture, walls, and floors. The spatial interactions between these elements will likely be high-level, involving simple behaviours like translation, rotation, and scaling. We anticipate that the need to control these models at the vertex level will be rare, although the ability to animate various components of these models will be required - for example, articulating the body parts of a human avatar. With these constraints in mind, we have adopted OSG, mainly because of the *scene graph* data structure, which

provides a hierarchical organization to virtual scenes (discussed in more detail in Section 6.3.2).

In order to use these two resources together, we use the API provided with Pd to create custom *externals* to suite our purposes. An external encapsulates any C/C++ code and provides a patchable object for use in the Pd programming environment. The soundEngine, soundNode, and soundConnection are all realized as such Pd externals, which users can place on their patches and connect messages as desired. These externals contain several methods that manage custom C++ classes, including those from the OSG toolkit. The soundNode, for example, extends the `osg::Referenced` class so that OSG can maintain reference counting and automatic garbage collection. It also contains instances of other classes (`osg::Node`, `osg::Group`, `osg::PositionAttitudeTransform`, etc.) that are used to manage the scene graph and the associated parameters and methods.

The additional benefit of using the OSG graphics toolkit rather than a simpler standalone scene graph library is that the scene can be rendered visually as well as sonically. This is a very important feature, since graphics provide the additional feedback that makes this system a true 3-D interface. For more information about OSG and the scene graph data structure, readers are encouraged to read Appendix B before proceeding with the rest of this chapter.

6.2 The soundEngine

The soundEngine is the overall manager of the virtual scene and allows for the creation (and destruction) of soundNodes and soundConnections. Table 6–1 lists the various methods that are available.

create [nodeID]	Creates a soundNode with the given nodeID
destroy [nodeID]	Destroys the soundNode with the given nodeID
connect [sourceID] [sinkID]	Create a soundConnection between the sourceID and sinkID
disconnect [sourceID] [sinkID]	Destroy the soundConnection between the sourceID and sinkID
saveXML [filename]	Save the current scene (including all soundNodes, soundConnections, and all parameters) to an .xml file
loadXML [filename]	Load a scene from an .xml file
clear	Clears the current scene
debug	Prints debugging information to the Pd status window

Table 6–1: The commands recognized by the soundEngine.

It should be noted that the soundEngine's *create* and *connect* methods are simply helper functions. A user can also just place a patchable soundNode object on a Pd canvas, and it will be properly registered with the soundEngine and placed on the scene graph. The helper functions are however convenient since with one command, they create all the appropriate Pd objects and connect them as necessary.[1]

6.3 The soundNode

The fundamental building block of our framework is an entity called the *soundNode*. This is a data structure containing parameters that model the spatial characteristics of an audiovisual element, as well as logical parameters relating to control and DSP. A list of these parameters is identified and described in Table 6–2. Notice that each node has traditional

[1] This is accomplished using internal messages to Pd [38], which allow for dynamic creation and connection of Pd patches without using the mouse and keyboard.

Administration:

id	The unique ID of the node
parent	The ID of this node's parent
children[]	A linked list of children nodes (redundant data structure)
dsp	The DSP abstraction which computes audio for this node
maps[]	A linked list of mapping abstractions that define control behaviours

Spatial Parameters:

pos = (x, y, z)	The 3-D position of the node in its local coordinate system
scale = (x, y, z)	The scale of the object in three dimensions
radn = (θ, ϕ, ψ)	3-D vector representing direction of radiation (sound source orientation)
sens = (θ, ϕ, ψ)	3-D vector representing direction of sensitivity (sound sink orientation)
radnRolloff	A function or look-up table returning radiation values for various angles of incidence
sensRolloff	A function or look-up table returning sensitivity values for various angles of incidence
radnFactor	A distortion factor that affects the sampling of the radnRolloff
sensFactor	A distortion factor that affects the sampling of the sensRolloff

Display Parameters:

gfx	The filename of the 3-D graphical model to display
state	The animation index, from [0-1]
directionToggle	In the case of graphical objects with only one possible orientation, this flag indicates whether to use radn or sens for display
radnFlag	Indicates whether the radiation wireframe mesh is visible
sensFlag	Indicates whether the sensitivity wireframe mesh is visible
labelFlag	Indicates whether a 3-D text label is visible
lightFlag	Indicates whether the spotlight is enabled
VUmeterFlag	Indicates whether the VU meter is visible
laserFlag	Indicates whether the laser beam is visible
ambient = (R, G, B)	The colour of the ambient component of the spotlight
diffuse = (R, G, B)	The colour of the diffuse component of the spotlight
specular = (R, G, B)	The colour of the specular component of the spotlight

Table 6–2: List of all soundNode parameters.

spatial parameters such as position and orientation; there are also several parameters specific to our implementation.

As mentioned earlier, a soundNode can be either a *source* (which radiates sound), a *sink* (which absorbs sound), or both of these at the same time. The case where a soundNode represents both a source and sink is particularly important in the context of musical interaction since this is how signal modification is realized. The node labelled B in Figure 6–1 shows this dual-function node, while node A is purely a source, and nodes C and D are purely sinks.

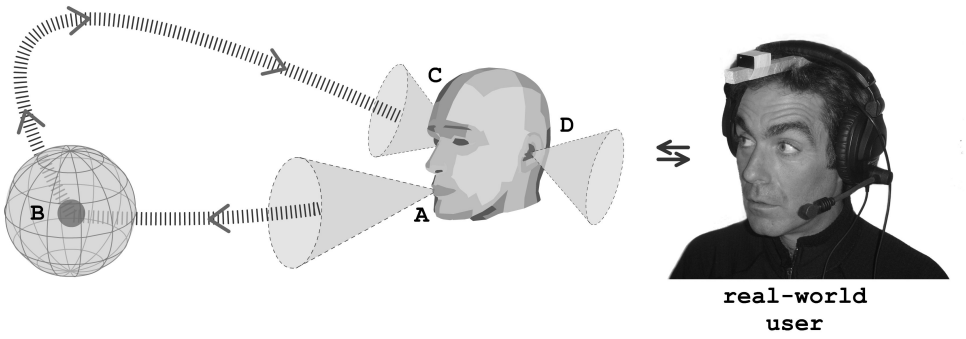


Figure 6–1: Example of a simple scene modelled with soundNodes.

Source nodes in our representation are similar to those found in many other systems. They typically represent input from microphones, sound files, or sound synthesis. The concept of sink nodes is however more rare. Most traditional systems only render the audio scene for one listener at one position in space. In our representation, the concept of a listener is more abstract since it is just a particular type of sink node (or group of sinks). Sound from the virtual scene is gathered there, and buffered to the soundcard so that they may be heard. This generalization allows for several interesting scenarios. For example, multiple listeners with user-specific audio rendering can occupy the environment at the same time by defining a number of sink nodes to represent each listener's auditory display. If a

user is wearing headphones to listen to the scene, then two sound sinks are used to represent each earpiece. If the listener is using a surround speaker setup to listen to the scene, then sound sinks are defined for each physical loudspeaker. Figure 6–2 shows how we might change the virtual scene in the case that surround speakers are used. Each sound sink collects sound from the direction in which a physical loudspeaker is located with respect to the user. For example, the front-right sink in the surround setup will collect sound from the front and right of the user's current position.

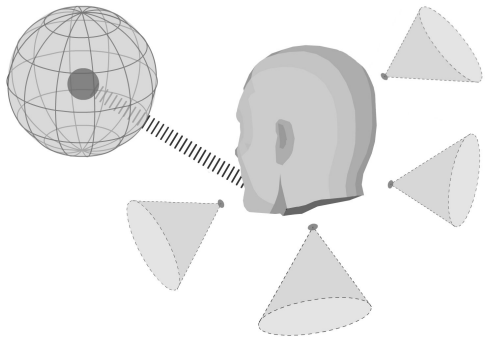


Figure 6–2: Example of a simple scene using a surround display rather than a headphone display.

It is important to note one difference between Figures 6–1 and 6–2. In the former case, the user's head orientation must be tracked and the sinks representing the headphone speakers must move relative to the head. This is not true for the surround setup, where the user's head rotation does not influence the position of the surround loudspeakers. In fact, the user's head could arbitrarily move around within the space, getting closer or further away to some of those surround speakers, or even outside and beyond the speakerfield. This difference identifies two requirements for our framework. First, a mechanism for translating and rotating several nodes at once must be available. This is accomplished by organizing the nodes in a scene-graph, and is discussed in Section 6.3.2. Second, we must be able

to track the user's physical head position and orientation and use those offsets to correct the audio display. This is accomplished using sophisticated tracking hardware and algorithms, as described in Section 7.1.

6.3.1 Directivity

The dual functionality of every soundNode requires a few specific parameters to be defined (re: Table 6–2). For instance, we note that a soundNode must have two orientations: one to indicate the direction of radiation (*radn*), and one to indicate the direction of sensitivity (*sens*). A soundNode can thus accept signals from one direction and emit them in another. Also associated with each orientation, is a parametric directivity (or *rolloff*) pattern that specifies how sound intensity falls off as you orient away from those direction vectors.

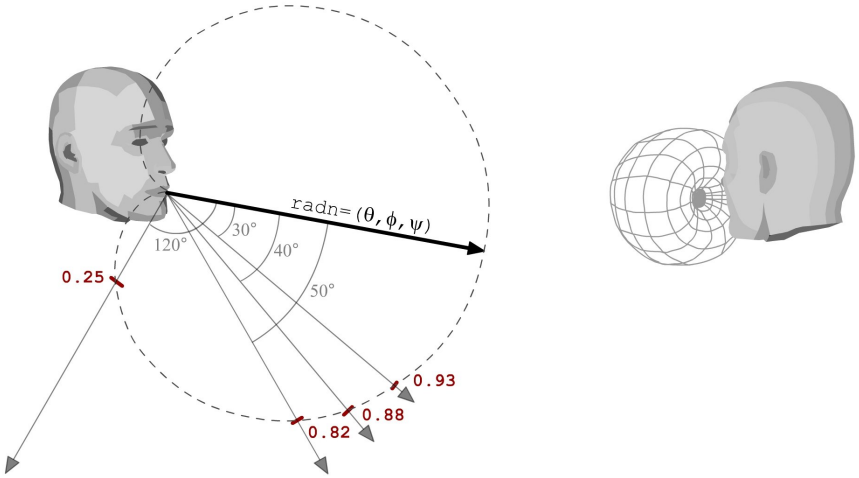


Figure 6–3: Directivity of a soundNode is shown. The sound radiates most strongly in the direction specified by *radn*, and falls off according to a cardioid-shaped rolloff function.

To illustrate, Figure 6–3 shows a source-type soundNode and its radiation. The sound signal will radiate with full gain (value of 1.0) along the direction specified by the *radn* vector, and fall off according to the *radnRolloff* function. Hence, we can think of a rolloff as some function or

procedure that provides gain values given an *angle of incidence* (α) to the direction vector.

In the case of Figure 6–3, the rolloff function is a cardioid[2] shape, graphed in polar coordinates in Figure 6–4, and is defined by the following equation:

$$\text{radnRolloff}\,(\alpha) = \left[\frac{(1+\cos(\alpha))}{2}\right]^{\gamma} \tag{6.1}$$

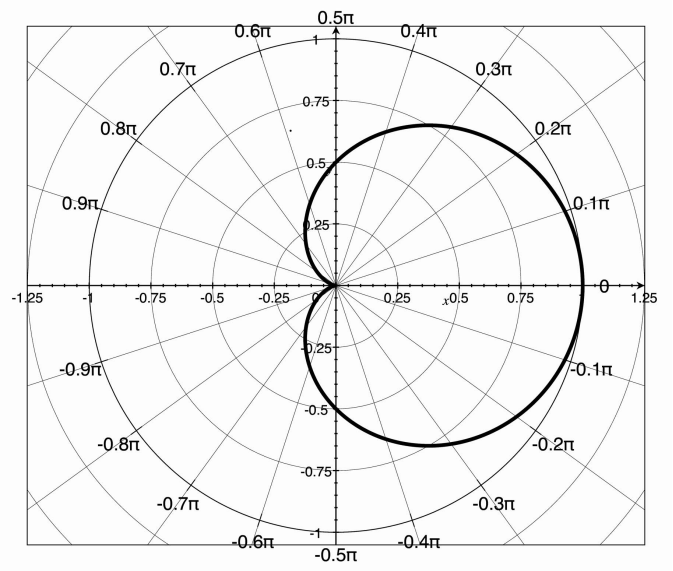


Figure 6–4: The cardioid function of Equation 6.1 when $\gamma = 1$.

Any such rolloff function can be defined by the user, or specified by a look-up table if a desired effect cannot be described algebraically. For best results however, the function should evaluate to 1.0 if the angle of incidence is 0 radians, and should be a valid monotonically decreasing function to the range of π radians. The reason for these requirements is that there is also an

[2] It is worthwhile to note that such directivity patterns are commonly found in the field of acoustics. Most directional microphones exhibit sound sensitivity similar to cardioids, and many traditional musical instruments have lobed radiation patterns [73].

additional control parameter called *radnFactor* (or *sensFactor* in the case of a sink-type node). This parameter allows for the distortion of a rolloff function, essentially by stretching or squishing it.

As an example, we see a rolloff factor, γ, in Equation 6.1. If $\gamma = 1.0$, then the function is evaluated as a normal cardioid without any distortion. However, when $\gamma = 0$, the shape flattens out resulting in omni-directional radiation, and when $\gamma > 1.0$ a hyper-cardioid shape is produced. This ability to change the directivity from omni-directional to tightly-focused with just one parameter is extremely useful to a performer.

6.3.2 Node Hierarchy

As mentioned earlier, there must be a method of interconnecting soundNodes so that their geometric properties are shared. For example, in Figure 6–1, we note that nodes A, C, and D all share a common geometric reference. If the user's head moves or rotates, then all three of these nodes should move and rotate together. To accomplish this, we borrow the graph structure from the field of 3-D graphics known as a *scene graph*. This is a tree-structured graph where spatial transformations applied to a node are automatically propagated to its children.

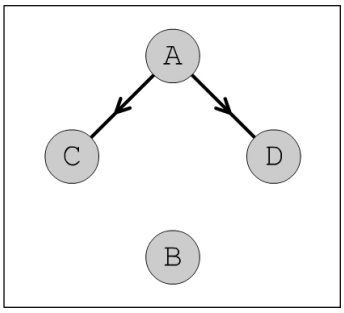


Figure 6–5: The scene graph associated with Figure 6–1.

Figure 6–5 shows the scene graph associated with the simple example presented in Figure 6–1. If the user rotates to speak in a different direction

and the source node A is updated as a result, then the location of the sink nodes representing the user's ears (C and D) will rotate accordingly because their positions are defined relative to their parent. Node B however is not affected since it has no geometric relation to the user's head.

To keep track of the scene graph, we see from Table 6–2 that each soundNode has a pointer to a *parent*, and a linked list of *children* pointers. It may seem redundant to store both of these parameters, but there are cases where certain updates are made more efficient if these are known.

It may be interesting to note that scene graph traversal is done quite differently by the audio engine than by the typical OSG graphical renderer. In the case of graphics, parts of the scene graph might be pruned if they are not in the viewing region, which saves CPU cycles. For audio, pruning is not possible since sound travels in all directions and has unlimited influence. Furthermore, a graphical traversal must be done for every rendered frame, whereas in the audio case, the scene graph is only a data structure and does not need continuous evaluation. The only time a scene graph needs to be traversed is when a parameter is changed, and in fact only the subgraph from the changed node down to the leaf nodes needs to be updated.

6.3.3 DSP

The ultimate aim of this framework is to help artists to organize DSP processes in order to build complex interactive musical systems. For this reason we keep the DSP methodology quite generic, allowing a user to specify almost arbitrary audio processing. The soundNode's *dsp* parameter is simply a pointer to a Pd abstraction, and the user is free to build any type of processing within that abstraction. The one constraint is that the abstraction must have only one input and one output (i.e., a monophonic patch). This may seem like a significant constraint, but polyphonic DSP

can easily be accomplished by grouping nodes together in a scene graph and copying audio signals between them. This will be discussed further in the next section, where we will also explain how connections between soundNodes are formed, and how DSP graphs are formed. For now, we just remember that soundNodes can absorb sound, perform DSP, and re-emit the result. This feature will allow for any complicated multi-voice instruments to be created, though careful attention will need to be paid regarding *how* the nodes are connected (via soundConnections).

6.4 The soundConnection

As mentioned earlier, a musical performance context requires powerful mechanisms to control the audio propagation between nodes. Rather than forcing an identical propagation model for every node, we construct a directed *soundConnection* between every source-sink pair in the scene. This soundConnection is in fact a data structure, containing various parameters that describe how sound travels between those nodes, allowing a user to specify whether the sound should be delayed, whether it should decay with distance, and whether it should be filtered to simulate diffraction and absorption. Furthermore, soundConnections need not exist for every possible pairing in the scene. They are only created as needed, to specify exactly which soundNodes can feed others with sound. If a soundConnection does not exist between two nodes, then the physical modelling of sound propagation between those nodes is not computed, and precious CPU cycles are saved.

As an example, consider again the scene depicted in Figure 6–1. There are exactly three soundConnections present: $(A \rightarrow B)$, $(B \rightarrow C)$, and $(B \rightarrow D)$. There is no connection from $(A \rightarrow C)$ or $(A \rightarrow D)$, because in this example, the user wants to hear only the processed voice, and not

the direct (raw) signal. If desired, the user could create such connections and mix in some of the raw signal. Such an effect would be similar to how we hear ourselves in nature. Likewise, feedback may be desirable from an artistic perspective, so the user may wish to create feedback loop potential among certain nodes. We thus start to see the power of the connection mechanism, since it allows the user to specify exactly how audio travels within the environment.

6.4.1 The soundConnection Features

In addition to creating and destroying connections, the user can also adjust several *soundConnection features*, as described in Table 6–3. Particularly, the intensity of each feature can be controlled as a percentage (usually from 0-100%, though values greater than 100% are sometimes possible). This ability allows for some interesting effects. For example, the user can *teleport* a sound signal between two very distant nodes by setting the effects of Distance Decay and Doppler to 0%. This ability is important since (as mentioned in Section 6.3.3) soundNodes can be used to organize polyphonic audio processing. For those purposes, the user will likely want to *copy* a signal to several nodes without any delay or decay being applied.

Distance Decay	The attenuation of the sound signal with distance
Angular Rolloff	The influence of the rolloff functions
Doppler Effect	A variable delay as a function of distance
Proximity Effect	A low-shelf filter for small distances (close proximity)
Absorption Filter	A low-pass filter simulating air absorption as a function of distance
Incidence Filter	A low-pass filter to simulate frequency-dependent attenuation as a function of angle
Reverb	A filter with an impulse response representative of the current scene size

Table 6–3: Features of a soundConnection.

It is important to note that these features allow for bending the rules of physics, as discussed earlier. Sometimes it may be useful to construct a scene that is not perceptually accurate, by enabling or disabling some of the connection features. For example, when "Angular Attenuation" is turned on, it may be impossible to hear something directly behind a sink. This would never occur in nature, but could be interesting since providing listeners with hyper-sensitive hearing might allow for improved auditory navigation, or new musical listening experiences. In the case of a source node, the angular attenuation feature is essential if the user wants to use the system as an effects mixer. It allows a sound signal to travel along a tightly constrained direction, which permits users to simulate the connections that might have been made with patch cords in the studio.

Another example involves the propagation time when sound travels from node to node. This travel time causes a variable delay, based on the distance between the nodes, and also affects the frequency content of the sound due to Doppler shift. It is often beneficial to eliminate the delay so that musicians can maintain a cohesive rhythm, and hear a synchronized mix even if they are closer to one instrument than another. Furthermore, the resulting frequency changes might degrade musical passages with rich harmonic content. Thus, fast-moving objects that contain tonal sound content should probably have Doppler disabled, while objects used for spatialization, simulation, or sound effects should include Doppler.

6.4.2 Computation of Audio Propagation

Once a sound connection is established between two nodes, the audio signal must be filtered and attenuated according to the arrangement of the nodes and the various connection features that the user has specified. For each connection, a handler exists that recomputes an attenuation value

and the filter coefficients whenever the position, orientation, or connection features are changed.

Such changes can occur quite frequently because several objects may be animated and continuously changing position. For this reason, we have two different rates at which computations are made. There is an *audio rate* (typically 44100 Hz), at which audio samples are convolved with various filters. The *control rate* on the other hand is much slower, operating in the range of 10-100 Hz. Updates to the parameters of a soundNode only occur at this slower rate, and users can adjust this rate to ensure that enough time is available for the re-computation of audio propagation. The audio is not affected by slowing down the control clock since it is handled by an independent scheduler. Users can thus ensure real-time performance even with complicated scenes.

The computation is relatively simple. We emulate the natural propagation of sound as described in Chapter 3, though some features such as proper diffraction and reverberation modelling have not yet been fully implemented. The parameters that we saw in Table 6–2 contain all the information that we need for the computation, but we show this diagrammatically in Figure 6–6.

This figure shows two connected nodes: source A and sink B, with direction vectors specified by $\mathbf{A}_{\text{radn}}$ and $\mathbf{B}_{\text{sens}}$, respectively. This notation is necessary to differentiate the *radiation* of sound from A while node B is instead *sensitive* to sound. For another connection, A may be acting as a sink, in which case vector $\mathbf{A}_{\text{sens}}$ would be used to describe its directional sensitivity.

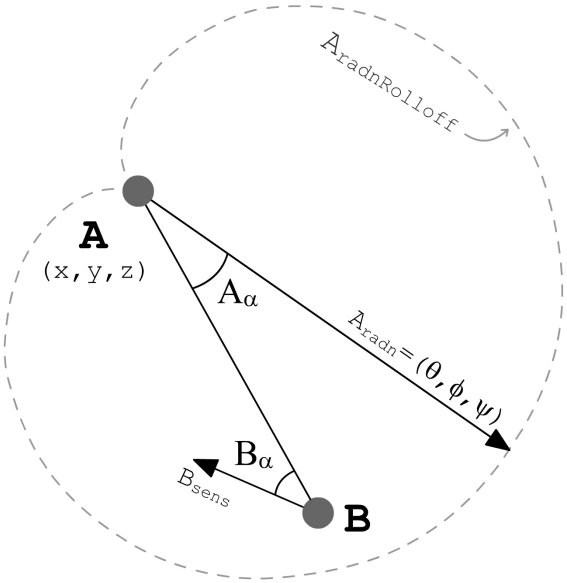


Figure 6–6: A soundConnection involving two soundNodes, showing various parameters needed for the computation of audio propagation.

Attenuation based on distance and angle:

The most basic model of sound propagation between nodes is to attenuate the signal according to distance and angle. For distance, sound decays exponentially due to the fact that it travels as a spherical wavefront, so we compute a distance gain value as:

$$G_{\text{dist}} = \frac{1}{(1 + |\mathbf{B} - \mathbf{A}|)^{\frac{\beta}{50}}} \tag{6.2}$$

This gain value will always be in the range of [0,1], and represents the amount by which the amplitude of the signal should be scaled to simulate decay of sound due to distance. The additional control parameter, β, helps to control the steepness of the exponential decay. Note that β is in fact controlled by the Distance Decay parameter of the soundConnection. When $\beta = 100\%$, it results in an exponent of 2.0, which is in fact identical to the *inverse square law* [73] that sound observes in nature. Meanwhile, a value of $\beta = 0\%$ effectively cancels the affect of distance decay since no

matter what the distance may be, the gain will always be 1.0. Lastly, it should be noted that percentages above 100% can be used for even sharper decay than what is common in nature, allowing users to use distance as an important control parameter.

Attenuation based on angle has already been discussed in Section 6.3.1, but will now be described in more detail. First recall that the *angle of incidence* (denoted by α) describes the difference between a node's radiation/sensitivity direction, and the direction of a connected sink/source. This can be computed for the source as follows:

$$A_\alpha = \cos^{-1}\left(\frac{\mathbf{A}_{\text{radn}} \bullet (\mathbf{B}-\mathbf{A})}{|\mathbf{A}_{\text{radn}}|\,|\mathbf{B}-\mathbf{A}|}\right) \tag{6.3}$$

and the sink incidence would thus be:

$$B_\alpha = \cos^{-1}\left(\frac{\mathbf{B}_{\text{sens}} \bullet (\mathbf{B}-\mathbf{A})}{|\mathbf{B}_{\text{sens}}|\,|\mathbf{B}-\mathbf{A}|}\right) \tag{6.4}$$

Recall that each node has a rolloff function that returns an attenuation value based on this incidence value. If the above incidence values are evaluated by their respective rolloff functions, two more gain values are:

$$G_{\text{radn}} = A_{\text{radnRolloff}}\,(A_\alpha) \tag{6.5}$$

and

$$G_{\text{sens}} = B_{\text{sensRolloff}}\,(B_\alpha) \tag{6.6}$$

The final attenuation due to distance and angle is simply the product of these various gain values:

$$G_{\text{final}} = G_{\text{dist}} G_{\text{radn}} G_{\text{sens}} \tag{6.7}$$

Spatial filtering:

It should be noted that the attenuation values computed above are applied to the *entire* sound signal. However, this is not an accurate model of sound decay in nature, where some frequencies are attenuated differently due to various spatial factors. For example, high frequencies are quite directional and are usually much quieter behind the source while low frequencies tend to wrap around and are heard equally in all directions. These behaviours must therefore be modelled using filters that are parametrized by geometric properties.

The loss of high frequencies behind an object can be simulated with the use of a low-pass filter (that we call an *incidence filter*), whose cutoff frequency decreases with the angle of incidence. The shape of the object will also play an important role, since audio wavelengths that are roughly the same size as an object will diffract and bend around the object. We use a soundNode's *gfx* and *scale* parameters to determine its shape. Many methods from 3-D graphics can be used to extract meaningful information, such as the convex hull (bounding 3-D contour), the average radius, or the volume that the object occupies.

High frequencies are also lost due to air absorption as the distance between the source and sink increases. This absorption or scattering is simulated with an *absorption filter*, which again is a low-pass filter where the cutoff frequency decreases inversely with distance.

A common effect known to sound recording engineers, the *proximity effect*, models the boost of low frequency energy when the distance between source and sink is very small. This is accomplished with the use of a low-shelf filter, but only when when a threshold closeness is attained. Though this is a side effect of directional microphones in the real world, it serves

as a good cue for spatial proximity. Furthermore, it is a familiar effect that most people will recognize, particularly performing musicians.

Another spatial effect that we consider is a simulation of reverberation by filtering with an impulse response function. While this impulse function should ideally be computed directly from the scene geometry, this has not yet been implemented in an automatic fashion. For now, the user must adjust a few parameters (1^{st} and 2^{nd} order reflection gains, reverberation gain, room size, reverberation time, etc.) to achieve a desired effect. In future work, we plan to examine ray casting techniques similar to the DIVA project [55] mentioned earlier, where additional sound sources would be automatically created at important reflection points.

The balance of directional sound, spatial filtering, and reverberant sound will greatly add to the user's spatial awareness. The fact that our framework allows the user to tune each of these components is quite interesting. For example, a user can diminish directivity effects (angular roll-off and incidence filtering) when the distance between source and sink is larger than the reverberation radius. The reverberant sound would thus dominate and the sense of directivity would decrease, which is a common psychoacoustic effect. On the other hand, reverb may be heavily exaggerated or made to contradict directivity for an interesting artistic effect.

Doppler Shift:

One final computation relates to the travel time of sound between source and sink. If either of the nodes are moving in space, this travel time changes, which results in an apparent shift in the sound frequency. This effect, known as *Doppler shift*, is reproduced using a variable delay embedded in each connection. The delay is is computed based on the speed

of sound in air (approximately 343 m/s) and the distance between the source and sink of the connection.

6.4.3 DSP graphs

The soundConnection object ultimately allows a user to organize soundNodes into graphs that describe how signals are processed. These *DSP graphs* are directed, starting from a source node and terminating at one or more sink nodes. They are also potentially cyclic graphs (since any source can also be a sink), meaning that recursive filtering can easily be developed. Figure 6–7 shows the DSP graph associated with the simple example that was presented back in Figure 6–1. The scene graph is also shown so that readers do not confuse these two representations.

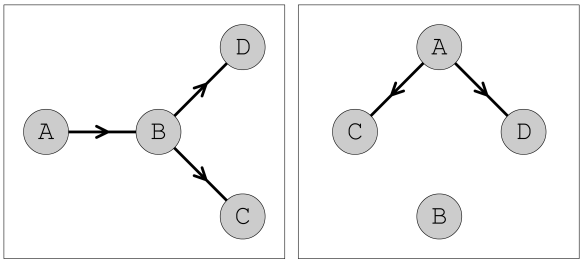


Figure 6–7: The DSP graph (left) and scene graph (right) associated with Figure 6–1.

Although DSP graphs have little to do with 3-D arrangement, they serve as a valuable method to visualize the audio processing chain. In future work, we aim to develop a (2-D) interface to easily manage the creation/destruction of soundConnections based on this graph structure. Users will be able to simply click and drag to create soundConnections between nodes, thus allowing for dynamic rearrangement of the DSP chain in addition to the spatial control parameters used in the 3-D interface.

6.5 Graphical Representation

The ability to add a rich visual component to 3-D audio works is desireable from an artistic point of view. For this reason, we provide a graphical engine that supports convincingly believable content. Designers can develop 3-D graphics using various commercial applications (e.g., 3D Studio Max, Maya, Blender), and incorporate these graphics into the virtual scene using OSG import plugins. Each soundNode has a *gfx* parameter that indicates the filename of such a graphical model.[3]

In our framework, graphics are not only used as an artistic medium, but are essential components of the editing and authoring environment. When dealing with such a large amount of data and many control parameters, it is important to have suitable feedback and meaningful visualization. By providing a graphical rendering of the virtual scene, a user can visually perceive the spatial arrangement of soundNodes, when doing so by ear might be difficult or impossible. This must, however, be done with care, ensuring that a tight coupling exists between audio and visual components, since this correspondence plays an important psychological role. A mismatch of 4 degrees is perceptible by professionals while 15 degrees is noticeable by a novice listener [50].

We also provide several visual debugging tools that can help a user to understand the audio processes that are taking place. This includes text labels, VU meters, and graphics to indicate the directivity of a soundNode. Each of these features can be turned on or off by setting the appropriate

[3] Depending on the authoring application being used, this filename can be a .3ds, .obj, .flt, or even an .osg file. The latter case is very powerful, since it allows us to attach an entire subgraph of models to one soundNode.

flags (see Table 6–2 for a list of flags). Figure 6–8 shows a scene where some of these features have been activated for particular nodes.

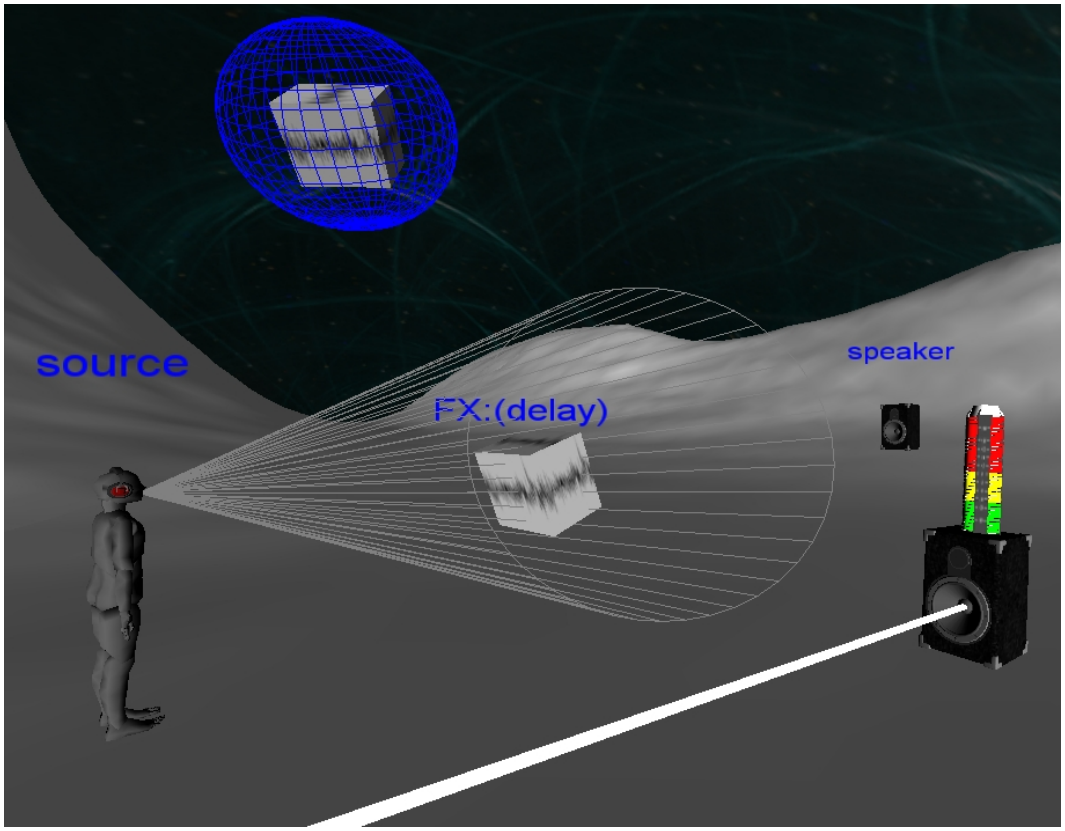


Figure 6–8: A screenshot of a scene that illustrates use of the debugging graphical features. The radiation wireframe mesh has been activated for the avatar to show the focus of the rolloff function. Conversely, the higher of the two FX nodes has a sensitivity mesh that shows omni-directional sensitivity. There are several text labels in the scene. The closest loudspeaker has a laser beam indicating its direction, and a VU meter that indicates the current sound intensity.

The *directionToggle* Parameter:

As mentioned earlier, a soundNode contains two different orientations (radiation and sensitivity). Any loaded 3-D model is assumed to be rigid, with only one possible orientation. For this reason, a *directionToggle* parameter exists that tells the graphical engine whether the *radn* or *sens* vector should be used to orient the graphical model. It may seem restrictive

to allow only rigid objects to be attached to a soundNode, yet it is still possible to convey articulation or deformation in other ways. For example, one could use two different soundNodes arranged in a scene graph so that they appear to be one graphical object capable of articulation. Otherwise, animations can be used (see below) to allow for any arbitrary deformation of shape.

Directivity Tools:

We provide the ability to visualize the directivity of soundNodes with a *laser beam*, *wireframe mesh*, or *spotlight*. Each approach has its respective benefits and drawbacks. The laser beam only shows the main direction vector of a sound node (i.e., the one specified by the *directionToggle* parameter), while a wireframe mesh can be shown for both the radiation and sensitivity. The wireframe mesh is a direct rendering of the current rolloff function, with the appropriate shrinking/stretching applied by the distortion factor.

The spotlight, though computationally expensive, can act as an appealing analog to the behaviour of sound as it propagates. Light has similar decay properties (in terms of both distance and angle), and can thus serve as an indicator of directivity, but in a more visually appealing way than a wireframe mesh. This feature can even be used during performance so that the audience can perceive the directivity of sound in space. Unfortunately, this feature can only visualize monotonically decreasing rolloff functions.

Animations:

Custom animations are possible using the keyframe animation feature popular in most 3-D modelling applications such as Maya or 3D Studio Max. Currently, there is limited support for exporting these to OSG, so designers must either encode their animations using the FBX format,[4] or use the OSGExp plug-in [46] for 3D Studio Max.

Regardless of how it is exported, the animation will be realized by nodes embedded in the scene graph. There are three ways to include such animations so that they are recognized by our framework. The simplest method uses the `osg::AnimationPath` node, which essentially stores transformation matrices for various time indices, each of which describe a model's position, orientation, and scale at a specific point in time. OSG can then interpolate a matrix for any other time index given this list. Using this animation feature, models can be made to orbit, spin, or follow some predefined path. They cannot, however, exhibit mesh deformations or changes in shape. For that type of animation, an `osg::Sequence` node must be used.

The osg::Sequence node allows several models to be attached beneath it, each of which represent one frame of an animation. The `osg::Sequence` node controls which child node is visible at any specific point in time. The benefit of this type of animation is that arbitrary deformations can be handled, yet the drawback is that each frame is a unique model, whose description requires space and memory. An `osg::Switch` node also exists, which is identical to the `osg::Sequence` node, except that multiple (or even

[4] Autodesk FBX is a free platform-independent authoring and interchange format that provides access to content from most 3-D vendors [13].

all) children can be visible at the same time, and there are no automatic controls to play the animation. The showing and revealing of nodes must be controlled manually.

Given these types of animations, our framework provides one simple parameter to control their playback. The *state* parameter allows a user to assign a number (in the range of [0,1]) to specify which frame of the animation should be shown. This number is automatically scaled to match the current length of the animation, thus simple control signals can be designed by users to play the animations in a desired way. This provides a powerful mechanism for sonigenic display, where audio parameters such as energy, envelope, or beat detection can be rendered visually.

VU Meter:

For each soundNode, an animated intensity meter is also available that displays the current sound energy contained within the node. This is similar in appearance to a home stereo VU meter, with colouring (green for low intensities, yellow from -12dB to 0dB, and red above) to indicate the strength of the audio signal being sent or received by the soundNode.

Text Labels:

The last and most simple visual feature is the 3-D text label. This graphic appears just above the bounding box of the soundNode's subgraph, and ignores the node's orientation since it always faces the camera for better legibility. The text can be set with any content, so users can describe each soundNode with a meaningful statement.

6.6 Example Using Pd Patches

Given the framework described so far and the simple example seen earlier in Figure 6–1, we now consider how one can build this scene using

Pd patches. Various message boxes need to be defined and the appropriate object boxes need to be instantiated. Figure 6–9 shows a screenshot of this patch.

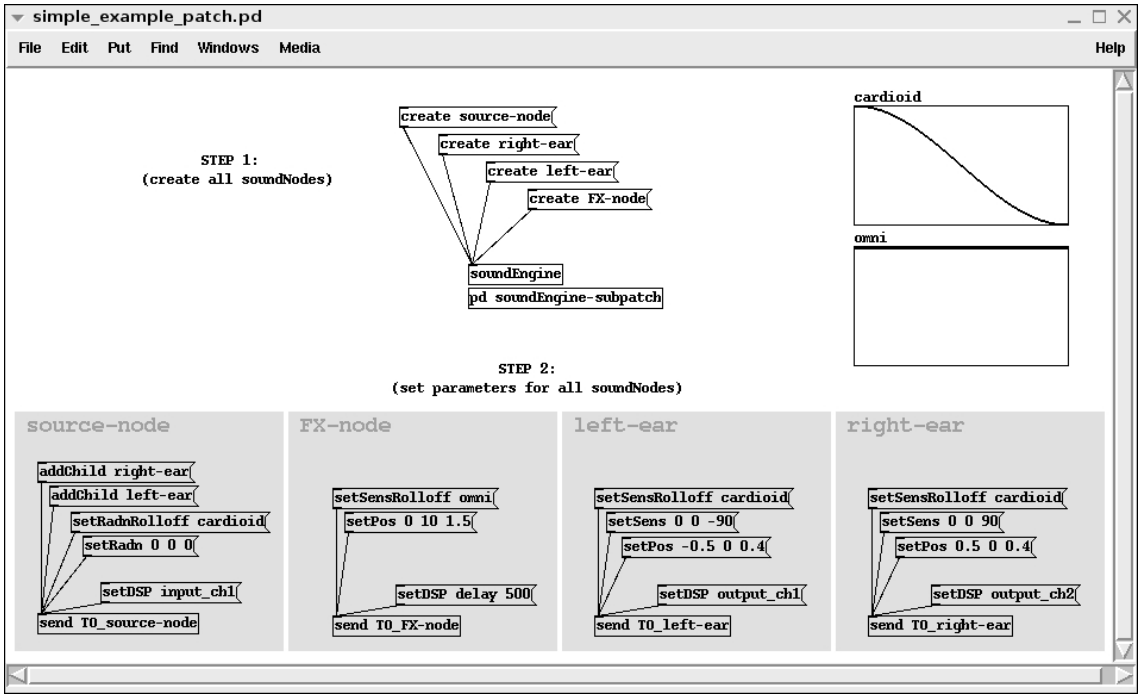


Figure 6–9: The Pd patch associated with Figure 6–1.

The construction of the scene begins by sending messages to the soundEngine to create each of the four soundNodes (source-node, right-ear, left-ear, and FX-node). In doing so, the soundEngine will dynamically create the appropriate soundNode objects and place them on the *soundEngine-subpatch* canvas. The soundEngine conveniently sets up the appropriate objects to receive messages for each soundNode. These receive elements are prefixed with "TO_" Thus, to update parameters of the source-node, one can simply send a message to "TO_source-node".

After the soundNodes have been created, parameters are set so that the nodes behave like the scene in Figure 6–1. Note for instance that the right-ear and left-ear soundNodes have been added as children to the source-node.

This means that the ear positions and orientations will be described in a local coordinate system relative to the source-node. The left-ear position of $(-0.5, 0, 0.4)$ is not an absolute position, but is rather the offset from the position of the source-node, locating the ear 0.4 units above the source-node and 0.5 units to the left.[5] Each ear also has cardioid sensitivity pointing at 90° away from the direction in which the source is radiating its sound.

The DSP abstraction for each ear is defined to send the captured audio signal to the soundcard's output channels (presumably to a pair of connected headphones), while the DSP abstraction for the source-node accepts the soundcard's input channel (presumably a microphone) and emits the result into the scene. Figure 6–10 shows the source-node's DSP abstraction. Note that the inlet is ignored, and a sound signal is only connected to the outlet of the patch. This is indicative of a source-type soundNode. A sink-type soundNode would on the other hand accept a connection from the inlet and ignore the outlet. The FX-node uses both; it takes the incoming signal, delays it, and outputs the result into the scene, thus acting both as a source and sink.

6.7 Control and Parameter Mapping

One of the trickiest aspects of creating powerful musical interactions is the mapping of control parameters to musical effect. Many of today's so-called computer music interfaces either end up being too simple to achieve virtuosic playability in the long term, or too complicated to learn in the short term. Wessel and Wright [81] have made this point clear, and suggest mapping simple controls to higher-level musical behaviours. These would

[5] Note that OSG uses a right-handed coordinate system with +Z up.

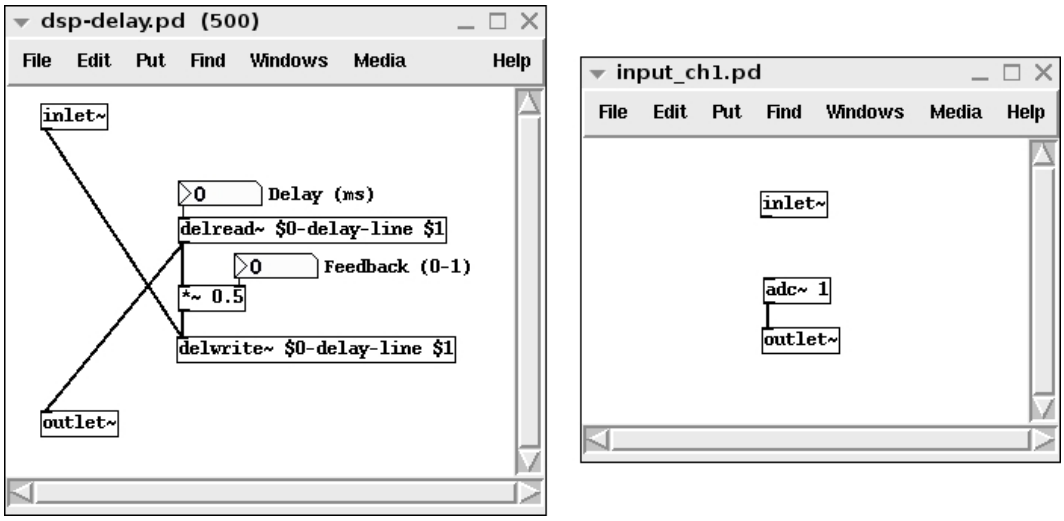


Figure 6–10: Two sample DSP abstractions: A delay patch that takes the delay time as an argument (left), and a patch to read input from the sound-card (right).

include control metaphors such as navigation through timbre space, or any multidimensional synthesis control. Hunt and Kirk [42] likewise suggest that *holistic control* of a parameter space is better than sequential control using one-to-one mappings. They also mention the need for a *performance mode*, where the system "stays constant whilst the focus is on the improvement of the player". Our system takes inspiration from these ideas by considering 3-D space as the mapping medium.

Since users are both geometrically and perceptually immersed within the virtual world, they have the ability to explore complicated parameter configurations in a holistic manner. Multimodal feedback is provided so that users can easily learn how the system reacts to certain actions, and there are typically many methods to achieve similar results. We offer a *performance mode*, where a user hopefully abandons the keyboard and mouse, and relies on more encompassing sensing technologies such as gloves and cameras. In this mode, users immerse themselves in the rendered 3-D world, and do not deal directly with the Pd programming environment.

They simply move around, use their gloves or other sensors, and their effects are rendered on the screen(s) and through the loudspeakers. The *editing mode* on the other hand involves direct interaction with Pd object boxes and GUI elements. The distinction between these two modes is however fuzzy, since a controller (glove, sensor, etc.) is in fact realized as a Pd object or series of Pd messages. The same effect can often be accomplished using both modes.

The mappings between controllers and their musical effects exist on several levels. Particularly, one can think of *spatial mappings* and *sonic mappings* as two different classes of parameter routing. Spatial mappings define control related to the transformations of 3-D objects, including methods for translating, rotating, and scaling objects. This is typically accomplished using some external sensor or input device, where the sensor data is captured and used to update the parameters of a particular soundNode. Sonic mappings on the other hand, define how those positions and orientations affect a sound signal.

It should be noted that sensor data can be mapped to directly affect sonic outcomes, without changing any spatial parameters. For example, the orientation of one's hand could change the pitch of a sound. However, we have found that those types of mappings are not as effective, and result in an interface that is difficult to learn. This is due to the fact that less feedback is provided, and the benefit of human spatial understanding is ignored. This leads to a steeper learning curve since the resulting behaviours needs to be memorized, whereas the mapping of spatial parameters to sonic effects seems to be quicker and easier to understand.

There are many inherent sonic mappings that already exist in the system due to the modelling of sound propagation. For example, the

distance between objects implies a decay of the sound signal that travels between them, but a user could define several other mappings as well. The sound generated by a node could increase in pitch as it increases with height, or the orientation of a node could indicate the play position in a sound file to create a virtual audio scrubber. The possibilities are vast, hence we need to provide users with the appropriate tools to realize all of these combinations.

Similarly to providing arbitrary DSP possibilities, we allow for almost any mappings to be defined using abstractions created in Pd. These are simple Pd patches that can intercept messages (from sensors, or any soundNode parameter update) and re-route them as the user sees fit. For example, in Figure 6–11 we see a patch that intercepts data received from an orientation sensor and uses it to control the radiation direction of a soundNode. The other patch in that figure takes the current height of a soundNode to control the delay time of the DSP.

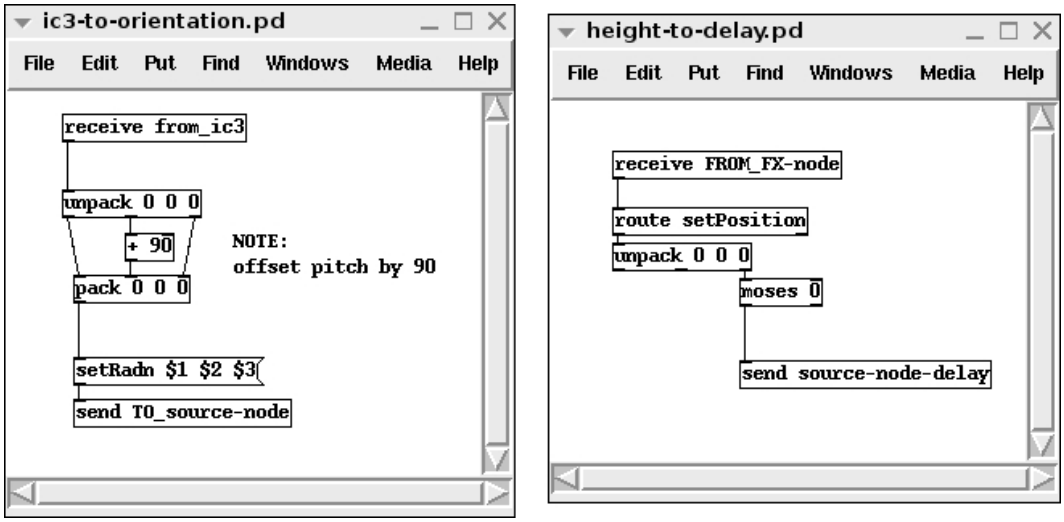


Figure 6–11: Two mapping abstractions defined in Pd. The first (left) routes orientation sensor data to the radiation direction of a soundNode. The other (right) routes the height of a node to the delay time of its corresponding DSP abstraction (note that the *moses* object simply ensures that the value does not drop below zero).

It should be noted that it is not necessary to intercept a message within a mapping. Rather, a *generative mapping* can exist where a metronome or clock can be used for control. For example, the orientation of a soundNode can continuously change over time so that it spins on its axis, or a clock could be used to advance a soundNode along a path in space.

6.8 The Distributed soundEngine

It should be noted that the framework proposed will often be distributed over a small network of computers. This is not always necessary, since a user could design an application for just one laptop and a set of headphones, but if immersion is desired then multiple screens are needed and hence multiple rendering machines need to be deployed. Figure 6–12 shows a sample setup for an immersive application.

As it turns out, OSG provides some efficient mechanisms for distributed rendering by employing clever algorithms on the scene graph data structures. By keeping track of the bounding region for each node's subgraph, OSG can prune entire sections of a scene that do not need to be rendered. A virtual camera views only a certain portion of a virtual scene. Anything not in the viewing volume will not need to be computed, including transform accumulation, lighting effects, and shading. The only thing that we need to do, is set the camera view for each machine, and an efficient distribution of rendering is realized.

As a result, we have written a custom OSG application that does not require Pd, and can be executed on a number of rendering machines. This application includes a daemon that listens and responds to network messages from a master controller. Messages can, for example, be sent to set the camera view. Yet more importantly, messages are sent that construct and maintain the state of all soundNodes in the virtual world.

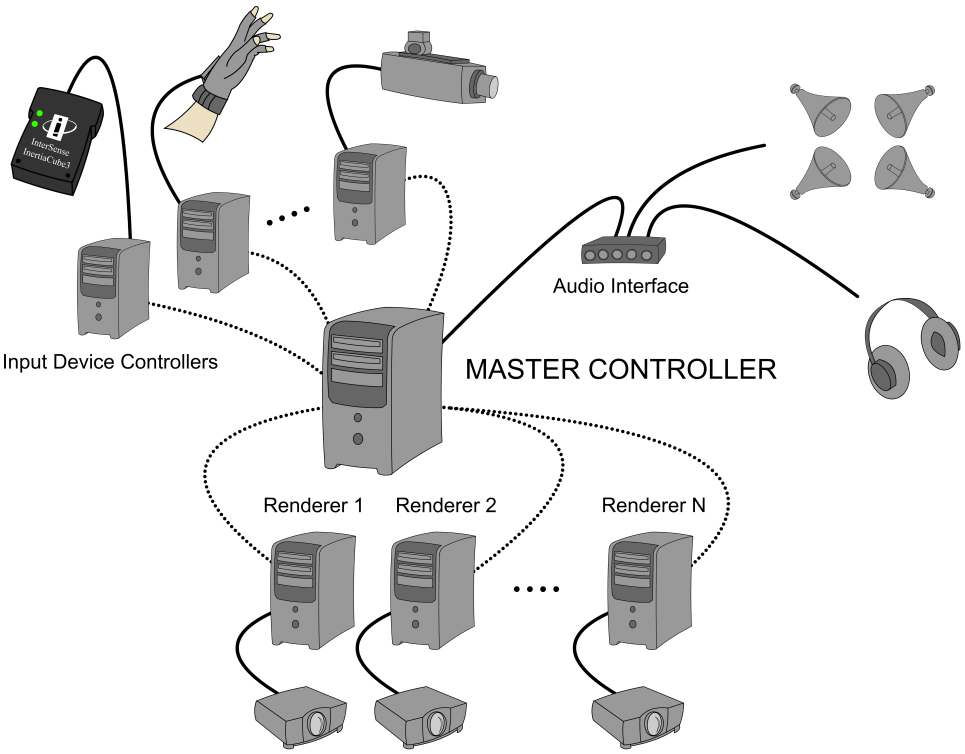


Figure 6–12: Overview of how the framework might be deployed in hardware to support an immersive environment. Note that solid lines represent physical connections while dotted lines represent network connections via TCP/UDP.

A parallel thread is spawned during the program's instantiation that continually monitors a socket for incoming UDP messages. This thread, which we can refer to as a *mailman* keeps track of a number of *mailboxes*, each pertaining to a unique soundNode. Whenever a new node is created by the master controller, the mailman is told to create a new mailbox for the given ID. Then, when parameter updates are sent, they are stored in the mailbox until the next frame cycle is computed. As an example, consider the following sequence of messages:

```
newSoundNode head
head setPosition 10 0 -10
```

The first message has the special token, `newSoundNode`, which tells the mailman to create a mailbox called `head`, then a new soundNode with that

ID is added to the scene graph. The second message begins by specifying a mailbox ID, which identifies where the remainder of that message is to be stored. Messages are stored within mailboxes as a FIFO queue. When the scene graph is traversed, each node has a callback function that is called, which checks its respective mailbox for messages and performs the various operations. The stored messages indicate a handler to call, and any additional arguments. In the case of our example, the OSG function `setPosition(10,0,-10)` will be called.

6.9 Putting it all together

With all of the pieces described above, it is possible to create a distributed system that renders both auditory and visual display for multiple participants. Each user is represented as a number of soundNodes, which move together through a virtual scene, since they are grouped in a scene graph. A number of sink nodes are defined to represent the physical loudspeakers of a user's sound system, and a source node is defined to allow a realtime audio feed into the scene. OSG provides camera objects, which can be attached to any node in a user's subgraph so that a subjective visual rendering is also available.

The scene can also contain many DSP nodes, which process audio at localized 3-D positions. Any type of processing can be realized by creating a DSP abstraction in Pure Data, thus providing a highly configurable tool for artistic creation. Many different control methods can also be developed using mapping abstractions, which allows data from input devices to be used for manipulation of parameters of the soundNodes. Together, these pieces support the creation of a virtual world, that can function as an audiovisual performance interface.

CHAPTER 7
Interaction Techniques

Until now, this discussion has only been concerned with virtual world representation, and not physical world representation. As previously mentioned, the ultimate goal is to deploy the system in an immersive virtual environment (IVE), where the user feels submerged in the world and receives no percepts that violate the congruence between physical and virtual reality. To accomplish this, we must have an accurate model of the user in the real world, and establish a similar virtual representation for use in interaction.

Once a model of the human user is available, many interesting interaction possibilities become available since the user shares one continuous space with various interactive elements. Hence, rather than using traditional WIMP (windows, icons, menus, pointers) interfaces, we would like to provide natural and efficient control techniques based on real human interaction. Of particular interest to our framework, a user will want to select various virtual objects (soundNodes) in the scene, change their spatial arrangement (translation/rotation), and perhaps change other parameters that do not necessarily have anything to do with 3-D space (e.g., select a rolloff function, change DSP type). Additionally, users will want to navigate around the scene, perhaps simply for exploration, or it may be necessary to move to a precise position in order to experience a particular sonic effect. A definitive solution for these challenges has not yet been discovered, yet various tracking systems and several interaction techniques have been prototyped. The rest of this chapter will elaborate on these investigations.

7.1 User Tracking

Earlier in Section 2.2, we presented some of the sensing and tracking technologies that are commonly used in IVEs. For the framework presented here, we have yet to include a standardized user tracking system along with our codebase. The main reason for this is that most of the methods discussed need a fixed and calibrated camera setup, while our goals include transportability and easy deployment. One exception is the work by Azarbayejani et al [14], which includes auto-calibration as part of the tracking system.

Combining several of those methods can also be quite difficult since each has a preferred type of image sequence. The methods used for tracking hand shape usually require the hand to be the central focus of the image, while diectic gesture recognition requires a more distant view. We in fact would like to have the entire human figure present in the image so that we can use the sequence also to display video avatars of users in the virtual world.

Our current tracking approach is thus relatively simple, with the obvious intention to make it more robust in the future. Grasping and releasing is not detected, since we intend to have other sensors available for this (e.g., the gloves described in Appendix C). For positional tracking, we have a front-view and top-view perspective of the scene. We use a combination of background removal via image differencing (the foreground image can be used for avatar display), and skin colour segmentation to extract meaningful features. Three blobs of connected pixels are formed, which represent each hand and the user's head. Similar to Pfinder [86], we compute the centre of mass, and low-order statistics for these blobs, thus providing estimates of current body posture. Uncertainties do however

arise due to problems of occlusion, varying illumination, false positives due to skin-like colours (e.g. wood surfaces), and poor pixel connectivity. For reliable tracking and minimization of noise, the CONDENSATION algorithm [44] has been explored to model these uncertainties, as well as the Kalman-filtering approach used by Pfinder. This method is not entirely accurate, since the resulting diectic vector passes through the head and hand rather than along the forearm. However, we found that with proper feedback on the screen, users were able to adjust to the paradigm.

Detection of the head position is adequately performed by computer vision, but we also need an accurate orientation, which is currently not possible with our tracking system. Furthermore, we note that the image processing required to deduce orientation with better accuracy will increase latency. Listeners often move their heads slightly to help localize sounds, and studies have shown that latencies greater than 70ms will decrease the ability to localize sounds [18]. For this reason, we do not use cameras for measuring of head orientation, and rather use a dedicated orientation sensor: the InterSense InertiaCube3 [5].

7.2 Virtual Object Selection

The targetting of 3-D objects using a 2-D screen is a relatively simple task. If the system can recognize diectic (pointing) gestures made by the user, then a 3-D ray can be defined that describes the pointing direction in the virtual coordinate system. OSG has built-in methods for *picking* 3-D objects from the scene given this ray (which is usually called a *picking ray*). A list of objects with which the ray intersects is returned, and typically we choose the first (closest) object in that list as the intended target.

Picking with one finger is however not the only technique that can be used to target an object. Pierce et al [63] describe numerous gestural

primitives, which can be seen in Figure 7–1. The "head crusher" for example, creates a picking ray that travels through the midpoint of the two fingers, while the "lifting palm" creates a picking ray that is offset slightly above the actual pointing direction of the hand. These methods are perhaps better than the "sticky finger" technique (essentially what we described above), since the object is not obscured by the user's hand. The "framing hands" method on the other hand could be used for targetting multiple targets, which would be an attractive feature to have.

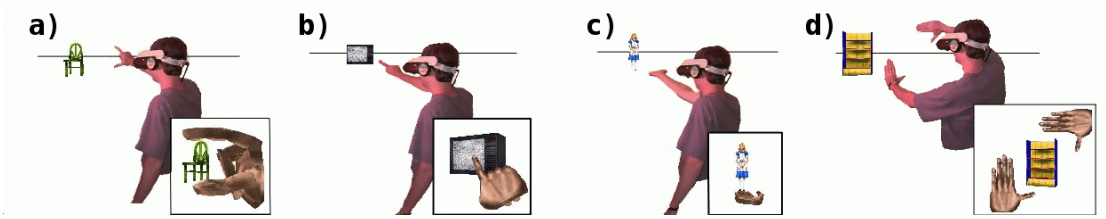


Figure 7–1: Interaction techniques proposed by Pierce et al [63]: a) head crusher, b) sticky finger, c) lifting palm, d) framing hands.

Once an object is targeted, the next step is to identify whether or not it should be considered as *selected*. An additional gesture, such as grasping, could be performed to confirm the selection. Of course, the system would need to be able to recognize such a gesture. Our tracking system however cannot recognize different hand postures (unless gloves are used), so a time threshold can be used, requiring the hand to remain locked on a target for a certain period before it is considered selected.

7.3 Virtual Object Manipulation

The techniques proposed by Pierce et al. provide for the selection of a virtual object. Once that occurs, a spatial modification gesture comes quite naturally. For example, the "framing hands" method can scale an object by moving the hands apart or closer together once it has been selected. Rotating the hands can also rotate the object, and moving the hands results

in a translation of the object through virtual space. In fact, by using two hands, several of these transformations can be performed at the same time. The user can quite easily scale, rotate, and even translate an object at the same time.

This benefit of using bimanual (two-handed) interaction has been shown by many researchers (e.g., [49, 39, 20, 54]). Particularly, Buxton and Myers [20] note that performance of two hands in a compound task is better than one hand. Leganchuk et al [54] note that less cognitive load is imposed when using two hands, thus allowing for more complex tasks to be performed without increasing the gestural complexity. Many researchers also note that there is a structure to using two hands that should be followed to ensure success.

7.3.1 Bimanuality

The disparity in the utility of the left and right hands, which is often described as "handedness", needs to be considered when designing bimanual interfaces. Generally, one *preferred hand* is favoured over the other for a variety of tasks. Guiard [36] has proposed a widely accepted paradigm called the *Kinematic Chain Model* to describe the asymmetry and division of labour between hands. He develops some rules that govern this phenomenon, which outline how two hands should be used:

- *Preferred-to-non-preferred reference:* the preferred hand finds its spatial references in the results of non-preferred hand motion.
- *Asymmetric scales of motion:* the preferred hand has higher temporal and spatial scales of motion. The non-preferred hand has more infrequent, coarse movements.

- *Non-preferred hand precedence:* the non-preferred hand moves first to set a frame of reference, then the preferred hand applies actions relative to that reference.

To illustrate, consider the example of writing text on a piece of paper. The non-preferred hand will first hold and position the paper (providing the reference and moving first). After positioning, the writing movements will typically be more frequent, precise, and span a greater spatial scale while the positioning movements will be rare and coarse in nature.

7.3.2 The Pieglass: A Bimanual Menu Widget

The spatial modifications mentioned earlier (translating, rotating and scaling using the "framing hands" technique) have been shown to be effective by several researchers [20, 27]. We hence aim to use those gestures for all spatial tasks. There are, however, other non-spatial parameters of a soundNode that have no natural corresponding gestures. Rather than forcing the user to learn an abstract gestural vocabulary, we propose the use of a menu to organize and present the options to the user.

In previous work [17], we designed a bimanual menu widget called a *pieglass*, which is a variation of the toolglass metaphor proposed by Bier et al [16]. It is also based on pie menus, which were proposed by Hopkins [40] as a replacement for traditional linear menus. Instead of being linearly placed, menu items are distributed radially around a central point, thus minimizing the distance to any menu option. The pieglass is also semi-translucent, so users can see through the widget to locate target objects behind. This is an important feature, since menu options are applied only to the 3-D object that is directly behind the widget. The pieglass also has the ability to open with several hierarchical layers (sub-pieglasses) so that more complicated menus can be realized. For instance, a user might select

the “DSP” wedge, which opens another pieglass with options for “Input Channels”, “Output Channels”, “FX”, and “Loops”. The “FX” wedge can then be chosen, which opens another pieglass containing all the effects that can be assigned to a soundNode.

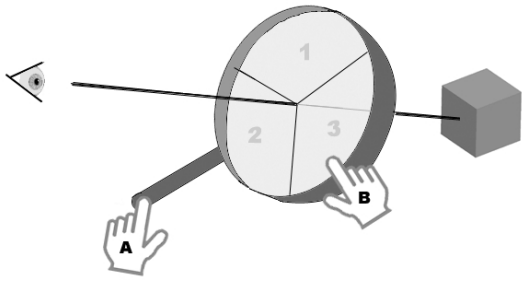


Figure 7–2: The pieglass metaphor: The hand labelled “A” controls the position of the widget and by doing so, selects a target object within the *crosshairs* of the widget. The hand labelled “B” navigates through menus and selects a modifier that is applied to the target object.

A diagram illustrating the interaction technique of the pieglass can be seen in Figure 7–2. The widget floats on the screen just in front of the camera, while the non-preferred hand controls its position (setting the frame of reference). An object is targeted when it is lined up in the *crosshairs* of the widget, then the preferred hand is used to navigate through the menu and select wedges, thus applying new parameters.

This metaphor allows for many interesting interactions. For example, once several sub-pieglasses are opened, they do not need to close after a selection is made. Rather, the user could open the “FX” pieglass mentioned above, and sequentially assign effects to multiple nodes one after another. There is no need re-open each menu option again, which allows the user to work more efficiently.

It should be noted that the pieglass has not yet been fully implemented within the framework described here. It still remains as prototype research, written in pure OpenGL. The transfer of this widget to OSG should not be

difficult, and will likely facilitate the editing and authoring of virtual worlds created with our framework. We are eager to pursue this investigation in our future work.

7.4 Navigation

Navigation is an important type of interaction that must be considered when using our framework. Users are able to create virtual worlds of arbitrary size and will need to change their viewpoint to interact efficiently with objects that are located in various regions. It is possible to use gestures to control the virtual viewpoint. Ware and Osborne [80] suggest three metaphors to accomplish this: holding a camera in one's hand, holding the world in one's hand, and using the hand to control a virtual flying vehicle. The first two methods have a direct and absolute mapping from hand position to camera position. To move to a certain part of the virtual world, a user simply moves his or her hand to the corresponding part of the real world. That spot will never change, allowing quick navigation to specific areas. The problem is that the world needs to be finite and somewhat small in size to provide reasonable navigation. The virtual flying vehicle however allows for a very large (and even infinite) world since the user's hand position maps to velocity rather than position. That is, the offset of the hand in a certain direction will indicate the camera velocity in that direction.

We find that even though velocity-based navigation is slower and requires better accuracy from the user, it is the most natural technique. We do not however use the user's hand to control the velocity since hands are reserved for interaction and manipulation of soundNodes. Instead, we use the user's head position. By moving within the physical world, users can set

the velocity of the camera in a certain direction. We call this the *Human Joystick Metaphor*.

This is a particularly useful metaphor when using OSG, since a camera can exist anywhere in the scene graph, even as the child of a soundNode. This means that the camera coordinates can be expressed in terms of the local coordinate system of a parent, and moving forward results in a translation along the direction vector of the soundNode. Many interesting behaviours can be accomplished with this technique. For example, a camera can be attached to a node orbitting in space. The human joystick movements that the user performs will simply offset the camera from the current path, so moving backwards will allow us to follow the soundNode as it orbits. Additionally, we have created a *lock-to-parent* mode for the joystick metaphor, which forces the camera to orient its view vector towards the parent. This allow the user to move away in any direction, while always keeping the parent node in sight.

CHAPTER 8
Sample Applications

The framework that we have described, when combined with user tracking hardware, can be used to create a number of interesting applications and installations. Obviously, one can use this framework to spatialize sounds for 3-D environments. However, the flexibility of the propagation paradigm allows us to exaggerate or diminish certain acoustic properties for artistic purposes. We believe that it is not always necessary to use perceptually accurate models of 3-D audio, and that several new applications can be discovered by bending the rules that govern reality. The following are a few ideas[1] that we have started to develop. These are not yet fully functional applications, yet the basic interactions have been tested.

8.1 Active Listening

In the real world, sounds arrive at our ears from all directions, and it is up to the human perceptual system to differentiate and localize them. We are extremely good at this task, being able to isolate a single conversation in a room full of people. This ability, known as the *Cocktail Party Effect*, has been shown to rely heavily on accurate 3-D sound information [12]. Thus, 3-D audio simulation has been a large focus of research when considering cluttered environments with many sound sources. Air traffic control is a perfect example, where transmitted conversations from pilots can be

[1] Several of these applications were included in a previous publication [85], with co-authors Zack Settel and Jeremy R. Cooperstock.

localized in a 3-D space according to their relative position from the airport [72].

One area where the Cocktail Party Effect breaks down is when the intensity of many nearby sources is much greater than some source that is further away. In such a case, it is extremely difficult to pick out this distant audio signal. However, the ability of our system to bend the laws of physics means that we can *focus* hearing in a particular direction when needed. The user can simply change the sink node sensitivity, so that any sound that is not directly ahead will be severely attenuated. Additionally, the user can diminish the effect of distance attenuation so that distant sounds are heard more clearly.

This ability to focus one's listening opens the door to various novel applications. Obviously, mission-critical systems such as air traffic control might not benefit due to the fact that some sounds might be entirely blocked from hearing. However, we see great potential in the realm of music, games, and other artistic applications. For example, consider listening to a huge group of musicians, perhaps an African percussion ensemble. Each individual in the group might be playing a different phrase with a unique pattern. Such polyrhythmic music means that an outsider who is listening to the entire ensemble will often perceive only one pattern at time. Often visual cues will adjust the listener's perception because he or she can track a particular performer's hands hitting a drum, and pick out a different rhythm. Moreover, the listener's position with respect to the group will also change this perception. By moving from one side of the ensemble to another, the apparent rhythm can change dramatically.

With our framework, we can reproduce and even enhance this listening experience while maintaining the subtle and natural interaction that

takes place. Each member of the ensemble would be represented as a soundNode, with a specific location in space. There could be a graphical avatar associated with each node so that the listener can also visualize the percussionist and see all hand movements. The position of the listener's ears would be modelled in space with associated positions and orientations. For example, headphones could be equipped with an orientation sensor and computer vision could be used to track the listener's position.

As the listener moves and turns in space, the experience of the music constantly changes based on the proximity and orientation to various source soundNodes. By *rolling* one's head to the side, the listening focus may be narrowed or widened, thus adding an additional element of listening control. Given the polyrhythmic nature of the music, the range of potential listening experiences is quite large. When listeners' sessions are recorded, great differences can be noted among the recordings, pointing to a potential authorship tool for creating remixes.

8.2 Multichannel Audio Displays and Mix-downs

The above mentioned ability to locate a listener within fields of sound sources lends extremely well to applications of audio mixing, and particularly mix-downs. Unlike conventional interfaces for mixing, which usually involve sliders and panning sounds in 2-D with some type of joystick interface, our interface intrinsically provides the ability to mix-down recordings using 3-D spatial arrangement. In this case, the so-called listener can be thought of as one or more *virtual microphones*, capturing sound sources in their proximity and spooling them to disk. The sound sources could be the independent tracks of a multi-track recording. For example, the drum track would be represented as one sound source while the lead guitar could be another, each placed in a specific area of 3-D space. This would

allow the listener to move closer to a particular node at the appropriate moment in a musical piece, such as when the guitar would have a solo. The quality of the mix is thus achieved by the organization of source sounds (tracks), and the movement of virtual microphones among them.

The particular format of the mix (5.1, stereo, etc.) is determined by the number of virtual microphones. For example, a 5.1 channel mix-down would use an array of six virtual microphones that collects the audio scene at a location in space. These microphones (sink-type soundNodes) are oriented to capture sound in their respective directions (e.g., the centre microphone would have 0° rotation, the right microhone would have 30°, the right rear would have 110°, etc.). It is worth noting that unlike conventional surround mixing environments, which localize sound in a horizontal plane (pantophonic), our environment's virtual microphones filter sounds based on their angle of incidence, offering fully 3D (periphonic) audio reproduction. Additionally, a virtual microphone array can be itinerant, and move dynamically though the recording space during mix-down, thus opening up additional artistic possibilities to the mixing engineer.

8.3 3-D Audio Dipping

Many modern performers assemble musical pieces from samples and loops of existing material. This usually involves gear such as turntables, samplers, and sequencers that are triggered using keys, knobs, and sliders. This control paradigm is quite effective, allowing a single performer to produce intricate musical material. The performer simply ensures that all of the devices are synchronized (perhaps by some external clock signal or by ensuring that all material has the same tempo). Then using sliders, he or she can fade in one or more of the signals.

This style of control can also apply to material generated by computer music processes or synthesizers. Wessel and Wright [81] have defined this technique as "Dipping", where the musical processes are silent by default, but the performer can *dip* in with some controller to make the processes heard.

Similar to Active Listening, the 3-D Audio Dipping is an application to which our framework is particularly well disposed. We start by placing source soundNodes around the user, ideally so that each node is equidistant. The sound contained in the nodes could be a number of phase-locked loops to ensure synchronization. The performer then focuses a collector (sink-type soundNode) in the direction of the sound that he or she wishes to dip into. For example, a 6 DoF sensor, mapped to the position and orientation of a sink node could be coupled to each of the performer's hands. Using another parameter such as the roll (or twist) of the hand, the user could also decrease or increase the focus so that fewer or more nodes can be dipped into.

Additionally, by performing an editing gesture (via glove controller, modal button, etc.), the performer could even relocate and swap soundNodes during performance, thus extending musical range. Such ability could prove quite useful, since the performer could group certain nodes together in space so that they are always dipped into at the same time. Nodes could also be moved closer or further away which would effectively control the gain of the emitted sound.

8.4 Spatial Routing to DSP Effects

In the examples above, we considered a number of source nodes in the scene that can be selectively listened to, or dipped into by a tracked human listener. It is also possible to flip the situation around, and have

many sink nodes present in the scene, which are all absorbing sound from real-world instruments. For example, we could equip a saxophone with a wireless microphone and an orientation sensor in order to capture its audio signal and always know its orientation in space. We can then define a number of sink nodes, each with a particular effects-processing unit such as reverb, delay, flanging, or ring modulation. When the sax player points the instrument in a certain direction, the sound is proportionally radiated to one or more effects-processing nodes in space according to their soundConnection features. By *rolling* the horn, the radiation is altered, fanning out to reach more effects nodes, or the inverse. The output of the effects nodes can simply be bussed to a mixer, or better yet, the resulting sound can be re-emitted into the scene so that it interacts with other effects nodes or is captured by a virtual microphone.

This setup is analogous to musicians who use pedals to send their instrument's audio to an effects boxes for processing. With our framework, however, we replace the real-world effects boxes, pedals, and connecting cables with virtual entities.

8.5 Audio Games

There has been some focus on development of audio-only games, particularly for visually-impaired users, and several recent attempts have been made to develop these games for all types of users [78, 71]. Röber and Masuch [71] describe several potential games. For example, *Mosquitoes* simulates a number of insects flying toward a user from all directions, and the user must aim bug spray in their direction to fend off attack. Spatial audio cues are used, in combination with head tracking and a Polhemus Stylus pen for determining pointing direction. Another example, called *AudioFrogger*, requires the user to cross a street without being hit by

oncoming traffic. The only cues for localizing vehicles are audio-based. The authors note that the exaggeration of sound effects, with Doppler in particular, can greatly aid the user's perception and interaction.

Although we have not yet created such games, our framework could be used to develop these and other interactive sonic worlds in a more versatile way than current audio APIs. As mentioned earlier, most APIs are built to supplement audio effects for 3-D visual applications, and hence their audio representation is usually not very robust. For audio-only gaming, it will be imperative to control the propagation of sounds with the utmost ability. It should be possible to adjust the effects of Doppler, filtering, and decay for each individual soundNode in the scene. For example, in the *AudioFrogger* game, one could boost the gain and enhance the Doppler shift for vehicles that are on a direction collision course with the player, while inconsequential vehicles would have diminished gain and Doppler.

8.6 Telepresence and Remote (Collaborative) Performance

The Shared Reality Environment at McGill University [24] allows people who are geographically separated to share one continuous virtual world. A user stands in a room equipped with immersive displays, cameras to capture movements, and a microphone to capture audio. Meanwhile, another user in an analogous space located somewhere else in the world will be able to interact with the user in Montreal as though they were in the same place. This situation is often referred to as *telepresence*, and our framework inherently allows for this type of feeling. When using our system, the users' bodies are already modelled in the virtual world in order to allow for proper rendering and interaction with virtual objects. We also have video captures of each person since cameras are used for tracking. Hence all the pieces are there to render the users (both visually and acoustically) as if

they are located at specific positions in the virtual world. In fact, recalling Figure 6–12, we can see that most of the data is already travelling through network connections in order to manage the distribution of rendering. Those network connections are typically local, but could theoretically travel anywhere in the world to support remote users.

This architecture allows for many interesting scenarios, where users can interact together with the same soundNodes. There are possibilities for remote performance where users can jam together, and the fact that the users are sharing one virtual instrument introduces notions of *collaboration*. Most remote performance systems simply allow musicians to play together using traditional musical instruments. There is little influence that one user can have over the other. Using our framework however, one user can be manipulating various soundNode parameters while another user is simultaneously sending sound in its direction. In effect, the environment becomes one large, distributed, multi-user instrument.

Another possible use of the Shared Reality Environment would be to perform virtual concerts, where audience members can visit and navigate around the world while one (or more) performers do the actual interaction. This would allow users with simple hardware (just one computer, headphones, and a joystick or mouse for navigation) to experience a 3-D audio performance from the comfort of their own homes, and leads to interesting new forms of performance art and musical expression.

There is however one very challenging issue that needs to be mentioned when discussing remote musical performance, and that is the issue of *latency*. There will be several delays introduced from the time a performer makes a motion to the time that the result is expressed musically at the remote location. The sensor acquisition (cameras, gloves, etc.) takes time

to digitize and process, the encoding, packetizing and transmission through various routers and switches over the internet takes a significant amount of time, and then a decoding process must be used at the receiving end before the final signals can be rendered. Although latency can be minimized by using several technological tricks [82], there will always be some latency that cannot be avoided. We are limited by the speed of light through fibre optics, and the topology of the network. A performance between Montreal and Vancouver for example, will need to cope with a network ping time of almost 100ms [25].

Although these delays seem small, synchronization between players is crucial for musical performance. The Distributed Immersive Performance (DIP) project at USC, has performed many experiments to determine what latencies are tolerable by musicians [22, 23]. They found that latencies of more than 50ms become intolerable for certain types of music. Obviously, some music is possible with greater latencies, especially more ambient music without strong tempo requirements. However, synchronization of highly percussive and rhythmic music can often not be synchronized even at 50ms.

We thus feel that the collaborative performance application mentioned above would be best deployed in close geographic proximity, while remote musical performance on the other hand could be visited by audience members anywhere on the globe. The difference is that the latter application does not require synchronization, and can thus be rendered on the receiving end with arbitrary latency.

CHAPTER 9
Conclusions and Discussion

We have described a new paradigm for interacting with sound in 3-D and for creating spatially organized audio processing. This paradigm is supported by a software framework, allowing a user to create virtual worlds that function as musical interfaces and interactive sonic applications. By combining tracking technologies capable of inferring a user's body position, we can employ several natural interaction techniques for manipulating the virtual world. When all of these features work together, they provide a powerful multimodal environment that can lead to many novel and interesting applications.

The power of the framework is realized by the sophisticated representation of soundNodes and their acoustic propagation model. Providing users with the ability to control directivity of sound with high accuracy, and allowing the rules of physics to be bent are both essential factors that are necessary for using virtual environments in a musical context. The traditional APIs and toolkits mentioned earlier are too focused on simple spatialization tasks, and cannot be used as effectively for musical purposes. Furthermore, the fact that soundNodes can act as both sources and sinks allows for DSP graphs to be designed, which can accomplish extremely complicated audio processing tasks. This enables users to create virtual worlds that simulate almost any piece of equipment that can be found in a sound studio (effects processors, synthesizers, mixing consoles, etc.).

Typical musical tasks can thus be accomplished using completely new metaphors, since they take place in continuous 3-D space, and do not need

to be logically organized with patch cords and sound modules. Users will tend to think of spatial relationships, rather than the positions of sliders or knobs, as the factor that influences the sound. This is powerful idea, since humans have a great sense of spatial understanding that they have acquired from the real world. Feedback from the system is thus more natural and subconscious, and less cognitive load is imposed on the performer when trying to keep track of the state of an application. For instance, a listener does not have to see a soundNode to know its position if the sound emitted can be heard. Hence the performer can focus more on the music and less on the interface.

Ultimately, we feel that this architecture will allow a performer to accomplish more than is possible using traditional audio equipment, especially if the task requires several controls to be operated in real time. We do need to verify this with experimental procedures, which is one of the future directions that is anticipated for this project. Other work will focus on user tracking and control methods. The vision-based tracking method described earlier does yield a coarse model of the user's body, but much higher accuracy and stability is required to make gesture-based interaction viable for extended use. The integration of gloves with vision-based tracking will make up for some of the annoyances caused by the tracking system. However, a solution to make the gloves wireless is needed before they can be used effectively in an immersive environment.

Regardless of the challenges still in place due to inadequate tracking, several interesting applications have already been prototyped that bypass these problems. The "3D Audio Dipping" and "Spatial Routing to DSP Effects" applications do not require a full model of the user and can be deployed using only one (or two) InertiaCube sensors. The "Mixdown"

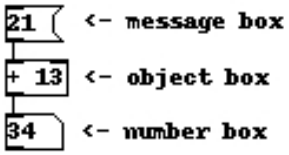
and "Active Listening" applications can also be performed outside of the immersive environment, using just one laptop computer. These applications are well disposed to headphone-based auditory displays, requiring that the InertiaCube be attached to the headphones, and navigation through the scene can be accomplished using a mouse or joystick.

We thus already see the potential of this new paradigm for providing novel musical tools. In the next year, we plan to release an open source version of the framework, called *audioTWIST*, as part of the Open Territories project [53]. This project will hopefully expand as new developers begin to add their own ideas and creations. In time, we expect that several artists will be able to create novel works of art, cutting-edge performances, and interesting installations using the framework. We are eager to see how this new way of thinking about music and sound in 3-D will evolve.

APPENDIX A
APPENDIX: PureData

PureData (Pd) is a graphical programming environment, created in 1996 by Miller Puckette [65]. It is geared particularly towards audio processing applications, yet it can also be used effectively for generic control systems, and graphical processing [29]. Programming with Pd is similar to Opcode's Max/MSP [6], and involves placing various objects onto a *canvas* and connecting them with *patch cords*. A collection of canvases with their connected objects can be saved as a file, which is often referred to as a *patch*. These patches can be created so that they too can be placed onto a canvas and become interconnected with other objects and patches. When a patch is used in such a way, we refer to it as an *abstraction* - mainly due to the fact that it can exist in multiple places with unique memory.

Pd is a real-time programming system, which means that as soon an object is placed on the canvas, it becomes operational. However, the object must be connected (with a patch cord) to another object to make for any meaningful operation. There are two possible types of connections: control connections, and audio connections. *Control objects* output sporadic signals only when a control event is triggered. This occurs for example, when a user clicks on a box, but can also be caused by an incoming message from another control object. Consider the example below:



The patch does nothing until a user clicks on the message box. An event is triggered that causes the message (the floating point number of 21) to be sent from the *outlet* of the message box to the *inlet* of the object box. The object box adds 13 to any incoming message and outputs the result, which then goes to a number box that displays the result. Control events thus propagate until they reach an object that is not connected to anything else, or is designed not to propagate the message any further.

Audio objects on the other hand (which always have a tilde appended to their name; e.g., osc~, dac~) continuously carry out computation. Pd arranges all audio objects into a linear order in memory, then processes blocks of an audio signal through this list at a constant rate. Typically blocks of 64 samples are processed at 44100 Hz, which results in a computation every 1.45 milliseconds. Control signals are often used to change the way an audio object behaves, for example, changing the frequency of an oscillator. Audio signals, however, cannot be fed directly into a control object. Some intermediate audio object must be used that takes an audio signal as input and outputs control signals at an explicit rate.

To create objects with such custom behaviours, users can do one of two things. They can create an abstraction, complete with inlets and outlets so that it can be used on a patch, or a custom object can be programmed in C using the PureData API. These custom objects, which are called *externals*, are compiled into dynamic libraries and are loaded by Pd during runtime whenever such an object is created. It is this feature of Pd that provides the flexibility and extensibility that we needed in order to develop our framework.

Further documentation of Pd can be found at www.puredata.info.

APPENDIX B
APPENDIX: OpenSceneGraph

OpenSceneGraph (OSG) is an "open source, cross platform graphics toolkit for the development of high performance graphics applications such as flight simulators, games, virtual reality and scientific visualization" [9]. It accomplishes this by providing high-level classes and methods to manage OpenGL. Yet most importantly, it provides an object-oriented data structure known as a *scene graph* to help with the organization of 3-D content.

A scene graph is simply a tree-shaped data structure that organizes the logical and spatial elements of a 3-D scene. Each element is represented as a node in a tree, including things like models, transformations, lights, textures, and cameras. The interesting thing about this organization is that it provides an efficient method to propagate information to groups of objects. Particularly, any spatial transformations performed on a parent node will be automatically propagated to all children nodes. Scene graphs are thus useful for representing rigid structures, and managing hierarchical scenes where groups of objects are animated together.

Consider the example of a typical dining table. The table top might be the parent node for four children nodes, each or which represent a leg of the table. Each child has a position that is relative to the parent, indicating its offset from the centre of the table. In fact, the child is not directly attached to the parent, but rather, an intermediate *transform node* exists in between the parent and child to realize the offset. Transform nodes include the `MatrixTransform`, `DOFTransform`, and `PositionAttitudeTransform`, which

are structures that store a 4x4 matrix used to translate, rotate, and scale the subtree attached below.

When rendering a scene, OSG traverses the tree from the root to the leaves. When it encounters a transform node, it multiplies it with the stacked transformation matrix to take into account the new position, orientation and scale. Hence if we translate or rotate the table top, then all legs will automatically translate or rotate with it since they contain the accumulated matrix of all parent nodes up to the root level. When the traversal finally encounters a model (e.g., a table leg), it then renders the model at the position described by the current transformation matrix. It should be noted that the table leg needs only to be stored in memory one time and can be reused multiple times in the scene, thus making the engine fairly efficient.

The propagation of operations to child nodes holds not only for spatial commands, but also for logical commands such hiding the group, pruning it from the scene, or assigning OpenGL state modes/attributes.

The OpenSceneGraph website (www.openscenegraph.org) has much more documentation related to the functionality and performance of OSG. Readers are encouraged to use this resource if more information about this toolkit is desired. Also, it should be noted that this project requires minimally OpenSceneGraph version 1.0.

APPENDIX C
APPENDIX: Building a Glove Controller

Part of the research presented here included an investigation into building a low-cost glove controller.[1] That is, a collection of sensors that would capture the posture of the hand, particularly to recognize gestures such as pointing and grasping. Figure C–1 shows a diagram of the hand [61], with the 23 possible degrees of freedom (DoFs) that we could try to capture.

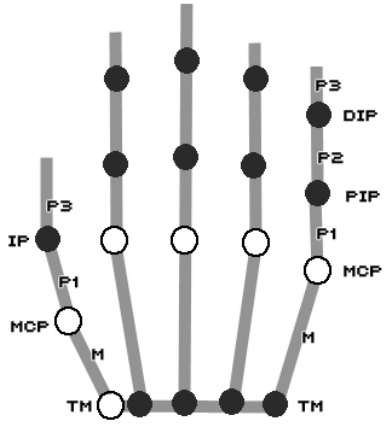


Figure C–1: Diagram of a hand [61]. Shaded circles depict joints with 2 DoFs while non-shaded depict joints with only 1 DoF. P1, P2, P3, and M are respectively the proximal, middle, distal, and metacarpal phalanxes. TM, MCP, IP, PIP, and DIP are respectively the trapexio metacarpal, metacarpo phalangeal, interphalangeal, proximal interphalangeal, and distal interphalangeal joints.

To keep sensors to a minimum, the choice was made to place one flexion sensor along the back of each finger, spanning from the proximal phalanx (P1) toward the tip of the finger. This would capture the flex of

[1] This glove controller was originally developed at McGill University's Music Technology Area, with guidance from Marcelo Wanderly.

the proximal interphalangeal (PIP) joint, which is good at representing finger posture since it exhibits the greatest change in flex for gestures such as grabbing and pointing. Note that if the goal was to use flexion as an excitation gesture, it might be better to use the flex of the metacarpo phalangeal (MCP) since a tapping-like gesture will typically keep the PIP constant while only moving the MCP. With this in mind, we kept sensor placement quite versatile with the use of Velcro. Sensors can be moved at any time to capture motions of other parts of the hand.

A dual axis accelerometer is mounted on the back of each hand, and measures the acceleration of both lateral (side to side) and medial (forward and back) hand motion. Thrusting motions can be easily captured and used as selection gestures, and sideways swinging and quick oscillations of the hand can indicate deselection. Furthermore, since there natural acceleration due to gravity, this sensor can also measure tilt of the hand when there is no actual displacement. More accurately, this corresponds to the DoFs known as pitch and roll:

$$pitch = A\sin(\frac{A_M}{g})$$

$$roll = A\sin(\frac{A_L}{g})$$

where A_L and A_M are lateral and medial acceleration values and g is acceleration due to gravity.

On the tips of each finger, several more DoFs have been incorporated by adding pressure triggers that can be used for additional selection gestures. By touching one's fingers together, a discrete event can be generated to toggle or trigger certain processes. Also, they may be used as clutches to identify when a gesture is initiated and then terminated.

The pressure triggers use force sensing resistors (FSRs) rather than simple switches. Thus they are also able to provide valuable parametric data related to how much force exists between the fingertips, and can be used to control other continuous parameters. The sensors are placed on all fingers except for the thumb, since it was assumed that each finger sensor would be triggered by the thumb.

All of the above mentioned transducers are types of variable resistor, thus some simple circuits were constructed to ensure that each sensor had a similar voltage swing. This circuitry is housed on back of the hand, with a versatile patching bay where different sensors may be connected or replaced on the fly. Figure C–2 shows the fully constructed prototype.

Figure C–2: Gloves with all sensors attached. Each finger is equipped with a bend sensor, and an FSR on the fingertip. The accelerometer and all circuitry is housed in the box that sits on the back of the hand.

Once the sensors are chosen and appropriately placed, the control voltages must in some way be converted to digital signals and made available within a Pd patch. The first interfaced used was the AtoMIC Pro (developed at Ircam), which converts analog voltages into MIDI data. This device can convert up to 32 analog inputs of 0-5V at a rate of 1075 μs per active channel. For one glove, 10 channels are activated (4 FSRs + 4 Flex

+ 2 acceleration), which results in a latency of 10.75 ms, and thus gives a sampling rate of 92Hz.

This device is however quite big, and requires several wires to be directly connected to the glove. Therefore, efforts are being made to find a smaller I/O board, hopefully with an on-board Bluetooth or RF wireless transmitter. This will free the user to walk around the space without being encumbered by wires.

References

[1] Ascension Flock Of Birds. http://www.ascension-tech.com/products/flockofbirds.php.

[2] Creative EAX. http://soundblaster.com/eax/.

[3] Extensible 3D (X3D) ISO standard (ISO/IEC 19775:2004). http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/.

[4] Immersion CyberGlove. http://www.immersion.com/3d/products/cyber_glove.php.

[5] Intersense. http://www.intersense.com/.

[6] Max/MSP. http://www.cycling74.com/products/maxmsp.

[7] Microsoft DirectX. http://www.microsoft.com/windows/directx/default.aspx.

[8] OpenAL. http://www.openal.org/.

[9] OpenSceneGraph. www.openscenegraph.org.

[10] Polhemus Liberty. http://www.polhemus.com/liberty.htm.

[11] Vicon MX Motion Tracking. http://www.vicon.com/products/viconmx.html.

[12] B. Arons. A review of the cocktail party effect. *Journal of the American Voice I/O Society*, 12:35–50, July 1992.

[13] Autodesk®. Universal 3D Asset Exchange - The Autodesk®FBX®Format. http://www.autodesk.com/fbx.

[14] A. Azarbayejani and A. Pentland. Real-time self-calibrating stereo person tracking using 3-D shape estimation from blob features. In *Proceedings of the International Conference on Pattern Recognition (ICPR '96) Volume III-Volume 7276*, page 627. IEEE Computer Society, 1996.

[15] D. R. Begault. *3-D sound for virtual reality and multimedia*. Academic Press Professional Inc., 1994.

[16] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. DeRose. Toolglass and magic lenses: The see-through interface. In *Proceedings of SIGGRAPH '93*, pages 73–80, 1993.

[17] Y. Boussemart, F. Rioux, F. Rudzicz, M. Wozniewski, and J. R. Cooperstock. A framework for 3D visualisation and manipulation in an immersive space using an untethered bimanual gestural interface. In *VRST '04: Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 162–165, New York, NY, USA, 2004. ACM Press.

[18] D. S. Brungart and A. J. Kordik. The detectability of headtracker latency in virtual audio displays. In *Proceedings of International conference on auditory displays (ICAD 2005)*, pages 37–42, 2005.

[19] W. Buxton. The haptic channel. In R. M. Baecker and W. Buxton, editors, *Readings in Human-Computer Interaction*, pages 357–365. Morgan Kaufmann Publishers, 1987.

[20] W. Buxton and B. Myers. A study in two-handed input. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, volume 27, pages 321–326, 1986.

[21] S. K. Card, J. D. Mackinlay, and G. G. Robertson. The design space of input devices. In *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 117–124, New York, NY, USA, 1990. ACM Press.

[22] E. Chew, A. A. Sawchuk, R. Zimmerman, the Tosheff Piano Duo (V. Stoyanova & I. Tosheff), C. Kyriakakis, C. Papadopoulos, A. R. J. François, and A. Volk. Distributed immersive performance. In *Proceedings of the 2004 Annual National Association of the Schools of Music (NASM 2004)*, 2004.

[23] E. Chew, R. Zimmermann, A. A. Sawchuk, C. Papadopoulos, C. Kyriakakis, C. Tanoue, D. Desai, M. Pawar, R. Sinha, and W. Meyer. A second report on the user experiments in the distributed immersive performance project. In *Proceedings of the 5th Open Workshop of MUSICNETWORK: Integration of Music in Multimedia Applications (MUSICNETWORK 2005)*, 2005.

[24] J. R. Cooperstock. Shared reality environment, March 2004. http://www.cim.mcgill.ca/sre/.

[25] J. R. Cooperstock. Interacting in shared reality. In *Proceedings of HCII 2005: The 11th International Conference on Human-Computer Interaction*, 2005.

[26] C. Cruz-Neira, D. J. Sandin, and T. DeFanti. Surround-screen projection-based virtual reality: The design and implementation of the CAVE. In *Proceedings of SIGGRAPH '93*, pages 135–142. ACM Press, 1993.

[27] L. D. Cutler, B. Froehlich, and P. Hanrahan. Two-handed direct manipulation on the responsive workbench. In *Symposium on Interactive 3D Graphics*, pages 107–114, 191, 1997.

[28] J. Daniel, J. Rault, and J. Polack. Ambisonics encoding of other audio formats for multiple listening conditions. In *Proceedings of the 105th Audio Engineering Society Convention. Preprint #4795*, 1998.

[29] M. Danks. Real-time image and video processing in GEM. In *Proceedings of the International Computer Music Conference*, pages 220–223, 1997.

[30] R. Elen. Ambisonics: The surround alternative. In *Proceedings of the 3rd Annual Surround Conference and Technology Showcase*, 2001.

[31] M. Friedmann, T. Starner, and A. Pentland. Device synchronization using an optimal linear filter. In *SI3D '92: Proceedings of the 1992 Symposium on Interactive 3D graphics*, pages 57–62, New York, NY, USA, 1992. ACM Press.

[32] W. G. Gardner. *3-D Audio Using Loudspeakers*. Kluwer Academic Publishers, Norwell, MA, 1998.

[33] W. G. Gardner and K. D. Martin. HRTF measurements of a KEMAR. *J. Acoustics Soc. of America*, 97(6):3907–3908, 1995.

[34] M. Gerzon. Periphony: With-height sound reproduction. *J. Audio Eng. Soc.*, 21(1):2–10, 1973.

[35] M. Gross, S. Wurmlin, M. Naef, E. Lamboray, C. Spagno, A. Kunz, E. Koller-Meier, T. Svoboda, L. Van Gool, S. Lang, K. Strehlke, A. V. Moere, and O. Staadt. blue-c: a spatially immersive display and 3D video portal for telepresence. *ACM Trans. Graph.*, 22(3):819–827, 2003.

[36] Y. Guiard. Asymmetric division of labor in human skilled bimanual action: The kinematic chain as a mode. *Journal of Motor Behavior*, 19(4):486–517, 1987.

[37] J. J. Heiss. The future of virtual reality: Part two of a conversation with Jaron Lanier. http://java.sun.com/features/2003/02/lanier_qa2.html, 2003.

[38] D. Henry. Pd internal messages. http://puredata.info/community/pdwiki/PdInternalMessages.

[39] K. Hinckley, R. Pausch, D. Proffitt, and N. Kassell. Two-handed virtual manipulation. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 5(3):260–302, 1998.

[40] D. Hopkins. The design and implementation of pie menus. *Dr. Dobb's J.*, 16(12):16–26, 1991.

[41] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 693–702. ACM Press, 2002.

[42] A. Hunt and R. Kirk. Mapping strategies for musical performance. In M. Wanderley and M. Battier, editors, *Trends in Gestural Control of Music*. IRCAM - Centre Pompidou, 2000.

[43] K. Isakovic, T. Dudziak, and K. Kchy. X-rooms. In *Proceeding of the seventh international conference on 3D Web technology*, pages 173–177. ACM Press, 2002.

[44] M. Isard and A. Blake. Condensation – conditional density propagation for visual tracking, 1998.

[45] J. Jacobson, M. S. Redfern, J. M. Furman, S. L. Whitney, P. J. Sparto, J. B. Wilson, and L. F. Hodges. Balance nave: a virtual reality facility for research and rehabilitation of balance disorders. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 103–109. ACM Press, 2001.

[46] R. S. Jensen and J. Jessurun. OSGExp: An open source exporter from 3DS Max to OpenSceneGraph. http://osgexp.vr-c.dk.

[47] N. Jojic, T. Huang, B. Brumitt, B. Meyers, and S. Harris. Detection and estimation of pointing gestures in dense disparity maps. In *Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition*, page 468. IEEE Computer Society, 2000.

[48] J. M. Jot. Real-time spatial processing of sounds for music, multimedia and interactive human-computer interfaces. *Multimedia Systems*, 7(1):55–69, 1999.

[49] P. Kabbash, I. S. MacKenzie, and W. Buxton. Human performance using computer input devices in the preferred and non-preferred hands. In *Proceedings of SIGCHI conference on Human factors in computing systems*, pages 474–481, 1993.

[50] S. Komiyama. Subjective evaluation of angular displacement between picture and sound directions for HDTV sound systems. In *Journal of Audio Engineering Society*, volume 37, pages 210–214.

[51] D. Kortenkamp, E. Huber, and R. P. Bonasso. Recognizing and interpreting gestures on a mobile robot. In *AAAI/IAAI, Vol. 2*, pages 915–921, 1996.

[52] C. Kyriakakis. Fundamental and technological limitations of immersive audio systems. *Readings in multimedia computing and networking*, pages 69–79, 2001.

[53] La Société des arts technologiques. The Open Territories project. http://tot.sat.qc.ca.

[54] A. Leganchuk, S. Zhai, and W. Buxton. Manual and cognitive benefits of two-handed input: An experimental study. *ACM Transactions on Computer-Human Interaction*, 5(4):326–359, 1998.

[55] T. Lokki, L. Savioja, R. Väänänen, J. Huopaniemi, and T. Takala. Creating interactive virtual auditory environments. *IEEE Comput. Graph. Appl.*, 22(4):49–57, 2002.

[56] T. Maki-Patola, J. Laitinen, A. Kanerva, and T. Takala. Experiments with virtual reality instruments. In *Proceedings of NIME*, pages 11–16, 2005.

[57] A. Mulder, S. S. Fels, and K. Mase. Mapping virtual object manipulation to sound variation. In *Proceedings of the Information Processing Society of Japan, SIGMUS*, pages 63–68, Tokyo, Japan, Dec 1997.

[58] A. G. E. Mulder, S. S. Fels, and K. Mase. Design of virtual 3D instruments for musical interaction. In *Proceedings of the 1999 conference on Graphics interface '99*, pages 76–83, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[59] M. Naef, O. Staadt, and M. Gross. Spatialized audio rendering for immersive virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 65–72. ACM Press, 2002.

[60] H. K. Nishihara. Practical real-time imaging stereo matcher. *OptEng*, 23(5):536–545, September 1984.

[61] V. Pavlovic, R. Sharma, and T. S. Huang. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):677–695, 1997.

[62] I. Peretz. Auditory agnosia: A functional analysis. In *Thinking in sound: The cognitive psychology of human audition*, pages 199–230, Oxford, 1993. Clarendon Press.

[63] J. S. Pierce, A. S. Forsberg, M. J. Conway, S. Hong, R. C. Zeleznik, and M. R. Mine. Image plane interaction techniques in 3D immersive environments. In *Proceedings of the 1997 Symposium on Interactive 3D graphics*, pages 39–43. ACM Press, 1997.

[64] J. Pressing. Some perspectives on performed sound and music in virtual environments. *Presence*, 6(4):482–503, 1997.

[65] M. Puckette. Pure Data. In *Proceedings of the International Computer Music Conference*, pages 269–272, San Francisco, 1996.

[66] V. Pulkki. Virtual sound source positioning using vector base amplitude panning. *J. Audio Eng. Soc.*, 45(6):456–466, 1997.

[67] V. Pulkki. Uniform spreading of amplitude panned virtual sources. In *Proceedings of the 1999 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 1999.

[68] V. Pulkki. Spatial sound generation and perception by amplitude panning techniques. *Ph.D. Thesis, Helsinki University of Technology, Espoo, Finland*, 2001.

[69] W. M. Rabinowitz, J. Maxwell, Y. Shao, and M. Wei. Sound localization cues for a magnified head - implications from sound diffraction about a rigid sphere. *Presence: Teleoperators and Virtual Environments*, 2(2):125–129, 1993.

[70] J. Rehg and T. Kanade. DigitEyes: Vision-based human hand tracking. Technical Report TR CMU-CS-93-220, CMU, 1993.

[71] N. Röber and M. Masuch. Leaving the screen: New perspectives in audio-only gaming. In *Proceedings of International conference on auditory displays (ICAD 2005)*, pages 92–98, 2005.

[72] A. Ronald, M. Daily, and J. Krozel. Advanced human-computer interfaces for air traffic management and simulation. In *Proceedings of 1996 AIAA Flight Simulation Technologies Conference*, pages 656–666, 1996.

[73] T. D. Rossing. *The Science of Sound (2nd edition)*. Addison-Wesley, 1990.

[74] E. Scheirer, R. Väänänen, and J. Huopaniemi. AudioBIFS: Describing audio scenes with the MPEG-4 multimedia standard. In *IEEE Trans. Multimedia*, volume 1, pages 237–250, 1999.

[75] G. Steinke. Surround sound - the next phase : An overview. *J. Audio Eng. Soc.*, 44(1):651.

[76] B. Stenger, P. R. S. Mendonça, and R. Cipolla. Model-based hand tracking using an unscented Kalman filter. In *Proc. British Machine Vision Conference*, volume I, pages 63–72, Manchester, UK, September 2001.

[77] I. E. Sutherland. The ultimate display. In *Proceedings of IFIP*, pages 506–508, 1965.

[78] D. S. Targett and M. Fernström. Audio games: Fun for all? All for fun? In *Proceedings of International conference on auditory displays (ICAD 2003)*, pages 216–219, 2003.

[79] N. Tsingos, T. Funkhouser, A. Ngan, and I. Carlbom. Modeling acoustics in virtual environments using the uniform theory of diffraction. In *Proceedings of SIGGRAPH '01*, pages 545–552. ACM Press, 2001.

[80] C. Ware and S. Osborne. Exploration and virtual camera control in virtual three dimensional environments. In *SI3D '90: Proceedings of the 1990 Symposium on Interactive 3D graphics*, pages 175–183, New York, NY, USA, 1990. ACM Press.

[81] D. Wessel and M. Wright. Problems and prospects for intimate musical control of computers. In *Workshop at NIME*, 2001.

[82] W. Woszczyk, J. R. Cooperstock, J. Roston, and W. Martens. Shake, rattle, and roll: Getting immersed in multisensory, interactive music via broadband networks. *Journal of the Audio Engineering Society*, 53(4):336–344, 2005.

[83] M. Wozniewski, Z. Settel, and J. R. Cooperstock. A framework for immersive spatial audio performance. In *New Interfaces for Musical Expression (NIME), Paris*, pages 144–149, 2006.

[84] M. Wozniewski, Z. Settel, and J. R. Cooperstock. A paradigm for physical interaction with sound in 3-D audio space. In *Proceedings of International Computer Music Conference (ICMC)*, 2006.

[85] M. Wozniewski, Z. Settel, and J. R. Cooperstock. A spatial interface for audio and music production. In *International Computer on Digital Audio Effects (DAFx)*, 2006.

[86] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland. Pfinder: Real-time tracking of the human body. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(7):780–785, 1997.