A Comparison of Software Tools for the Implementation of Spatial Sounds in Virtual Environments

Yu F. and M. Bouchard
School of Information Technology and Engineering, University of Ottawa, Ottawa, Ontario, K1N 6N5

1.0 Introduction

Since its earliest days, the human-computer interface has been almost entirely visual. Until fairly recently, audio was limited to some kind of altering "beep" for output. Then CD-quality stereo sound appeared in the multimedia-equipped computers. Conventional stereo can easily place a sound in any spot between left and right loudspeakers. However, with true 3-D sound using binaural technology, the source can be placed in any location - right or left, up or down, in the front or in the back [Blau97]. This can be useful when a listener is presented with multiple auditory streams, when information about the positions of events outside of the field of vision is required, or when a listener would benefit from increased immersion in an environment [Dura94]. Current applications of 3-D sound include computer games, videoconference systems, complex supervisory control systems, civil and military aircraft warning systems and computer-user interfaces. More and more, producing 3-D sound is essential to build a virtual environment. In this paper, we review some of the current software tools available for creating the illusion of three-dimensional sounds in virtual environments using binaural technology, and we attempt to evaluate their performance and compare their virtues and shortcomings.

2.0 C++ implementation using HRIRs available from the World Wide Web

To find the sound pressure that an arbitrary source x(t) produces at an ear drum, all that is required is the impulse response h(t) from the source to the ear drum. This is called the Head-Related Impulse Response (HRIR), and its Fourier transform H(f) is called the Head Related Transfer Function (HRTF). The HRTF captures many of the physical cues required for source localization [Bega94]. Once the HRTF is known for the left ear and the right ear, it is possible to synthesize accurate binaural signals from a monaural source. The HRTF is a surprisingly complicated function of four variables: the three space coordinates and the frequency. Systems based on HRTFs are able to produce elevation and range effects as well as azimuth effects [Ming98]. This means that, in principle, they can create the impression of a sound being at any desired 3-D location. In practice, because of person-to-person differences and computational limitations, it is much easier to control azimuth than elevation or range. Nevertheless, HRTF-based systems are fast becoming the standard for advanced 3-D audio interfaces.

A basic and effective spatial audio system is shown in Figure 1, which provides a conceptually simple way to use HRTFs for spatial audio. Basically, it consists of two "convolution engines", each of which can convolve the same audio input stream with a head-related impulse response (HRIR) retrieved from a table of measured values. The outputs of the convolvers go through amplifiers to headphones worn by the listener. The use of headphones eliminates the problem of cross-talk between loudspeakers [Sen97], [Gard98], however it has its own drawbacks such as user fatigue and sounds that sometimes seem to come from inside the head [Bega94]. The first implementation that was tested was thus a simple demo C++ program that convolves an input monaural file with HRIRs and produces a 3D sound stereo output file. The HRIRs that we used in our demo were downloaded from MIT Media Lab web site [Mit]. No attempt to compensate for the response of the headphones or the listener's ear canal was made.

As a whole, the 3D effect of the demo was found to be good. The sound source can be felt moving around the head instead of moving between the two ears. Since no appropriate HRTF transition mechanism (i.e. "crossfading") was added when a sound source was moving from one location to another, 'click' sounds could be heard in the demo.

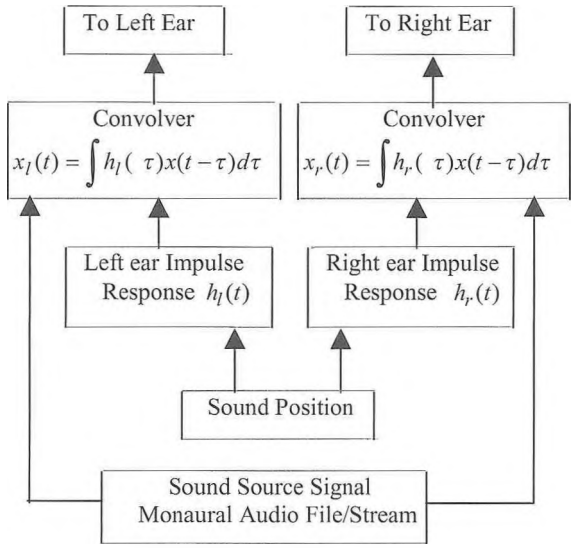

Fig 1. HRTF-Based System

3.0 Real-time Implementation using DirectSound3D™

DirectSound is the wave-audio component of Microsoft's DirectX API [Brad98]. Actually it serves as a middle layer between applications and device drivers, providing numerous sound management capabilities. DirectSound3D (DS3D) is a subset of the DirectSound API calls, that allows for the placement of sounds using a 3-D-coordinate system instead of a simple "left-right" pan. DS3D also automatically calculates things like attenuation due to distance and Doppler shift caused by the relative speed between the listener and the sound source. One demo was implemented to show the functionality of this API. Up to two sounds can be added into the demo, which are represented by two dots in a 2-D plane. The graphical interface of the demo is shown in Figure 2. For each sound source, parameters such as the position, velocity, minimum and maximum distance can be changed individually. The minimum distance is the distance where the volume stops to increase as a listener gets near a sound source, and the maximum distance for a sound source is the distance beyond which the sound does not get any quieter. The user can change the Doppler factor and rolloff factor (i.e. sound attenuation with distance). As in input, the demo uses a Wave monaural sound file. DirectSound3D also provides an additional feature: sound cones. A sound with an amplitude that is the same in all directions at a given distance is called a point source, but in DirectSound3D it is also possible to have sound sources that will only generate sound in a specific region with the shape of a cone.

The 3D sound effect of the DirectSound3D demo was found to be very good, even better than that of the HRTF-based system. Since appropriate crossfading between HRTFs is used by the DirectSound3D API, the 'click' effect which could be heard in the C++ demo of Section 2 was removed.

4.0 Real-time implementation using Java3D™ sound

Java3D is a high-level platform–independent 3-D graphics and sound programming API that can reduce application development time, simplifying all of 3-D graphics and sound programming. A Java3D program creates instances of Java3D objects, which is at least partially assembled from the Java3D class hierarchy, and places them into a scene graph data structure. The scene graph is an

arrangement of 3-D objects in a tree structure that completely specifies the content of a virtual universe, and how it is to be rendered. There are several sound leaf nodes that define the different sounds in a virtual universe [Kevi98]. A BackgroundSound node defines an unattenuated, nonspatialized sound source that has no position or direction. It is useful for playing a mono or stereo music or an ambient sound effect. A PointSound node defines a spatially located sound whose waves radiate uniformly in all directions from some point in space. A piecewise linear curve (defined in terms of pairs consisting of a distance and a gain scale factor) specifies the gain scale factor slope. A ConeSound node object defines a PointSound node whose sound source is directed along a specific vector in space. It is attenuated by gain scale factors and filters based on the angle between the vector from the source to the listener, and the ConeSound's direction vector. There is also a Soundscape node, which defines the attributes that characterize the listener's aural environment, such as gain scale factor, atmospheric rolloff, reverberation, distance frequency filtering and velocity-activated Doppler effect. Multiple Soundscape nodes can be included in a single scene graph.

A simple demo was built to show the functionality of a point sound node in the Java3D API. The point sound source (represented by a color cube), was moved around the listener in the frontal plane. The user could use the mouse or keyboard to navigate in the virtual universe. The 3D sound effect of this implementation was found to be almost as good as the demo of section 3, except that there was a bit more trouble distinguishing above from below.

5.0 Real-time implementation using a sound node in VRML

VRML stands for "Virtual Reality Modeling language". It allows specifying dynamic 3-D scenes through which users can navigate with the help of a VRML browser [Chri97], most of which are plug-ins for Netscape and Internet Explorer. A sound source model in VRML looks like two ellipsoids, one nested inside the other, with the inner ellipsoid sharing a foci with the outer one. The sound source emanates from the shared ellipsoid foci, and goes out in one direction toward the second foci of the outer ellipse. When a listener enters the outer ellipsoid, he/she hears the sound very quietly, and as he/she approaches the inner ellipsoid, the volume increases (the volume drops from the inner ellipsoid to the outside ellipsoid proportionally with the square of distance). Once the listener is inside the inner ellipsoid, the sound becomes ambient, which is to say the volume remains constant, and no 3D effect is computed. No sound is heard outside the outer ellipse.

To compare the sound functionality of VRML with those in Java3D and DirectSound3D, three sound nodes were programmed to move around the listener in horizontal, frontal and median planes, individually. The VRML browser that was used was Internet Explorer™ with the CosmoPlayer™ add-on. The 3D sound effect of the VRML demo was quite good, but the DirectSound3D implementation was slightly better.

6.0 Comparison of implementations and conclusion

Different ways to implement 3-D sounds in virtual environments have been studied through demo building. The 3D sound performance was only measured by a simple subjective listening evaluation, but the 3-D effect of all demos was quite impressive with or without visual cues. There are some obvious implementation differences between the different methods. It is quite complicated and time-consuming to build a HRIR-based system with the C language. But such a system has a high flexibility: extra acoustic characteristics such as echoes and diffraction could be added, and also different sets of HRIRs could be used for each individual to improve the performance. This cannot be done with other techniques. Also the exact spatialization algorithm that is used in the other techniques is unknown. The most interesting aspect of Microsoft DirectSound is that it provides a standard interface for the developers, which is supported by almost all the hardware manufacturers. This is important because software-emulated 3-D sounds are computationally expensive, and DirectSound can provide hardware acceleration. On PCs with audio cards supporting DirectSound acceleration, the host CPU consumption by the 3-D sound system will thus not be a problem.

The DirectSound3D API can deal with all kinds of input sound files. On the other hand, Java platforms only support the following audio file formats: AIFF, AU and WAV, linear or u-law PCM encoded. But the most attractive aspect of Java3D is that it is an API of Java, which is a platform-independent language. Once it is compiled, it can be run anywhere. Java3D also provides more control on the attributes of the acoustic environment to be rendered, for example with the Soundscape nodes. This is unique among all the techniques tested. Since VRML is just a modeling language used to describe 3-D scenes, it is the easiest way to implement 3-D sounds into a virtual environment. But it has to pay for these advantages with low flexibility and less efficiency. Also the implementation of spatialization in a VRML sound model is browser dependent. So it will be very difficult to allow consistent, reliable, high-quality 3-D audio on all platforms, using VRML. This platform-dependency may be the most serious disadvantage of VRML sound nodes.

To conclude, the best language or implementation tool to integrate 3-D sounds into a virtual environment depends on the application itself. If the efficiency is crucial for your application, DirectSound3D is a good choice, just as many computer game developers did. If your application is required to be platform independent, Java3D may be used instead. But if it is required to reduce the development time to its minimum, VRML is an interesting alternative, since it is the easiest one to use. And do not forget the most important rule of all in 3-D sound implementation: the content is the king.

References

[Bega94] Begault, D. (1994). *3-D Sound for Virtual Reality and Multimedia* Academic Press, Boston, MA, 1994.

[Blau97] Blauert, J. (1997). *Spatial Hearing* 2nd edition MIT Press, Cambridge, MA.

[Brad98] Bargen, B. and Donnelly, T.P. (1998) *Inside DirectX (Microsoft Programming Series)*. Microsoft Press.

[Chri97] Marrin, C. and Campbell, B. (1997) *Teach Yourself VRML 2 in 21 Days*. SAMS Net.

[Dura94] Begault, D.R. and Erbe, T. (1994). "Multichannel Spatial Auditory Display for Speech Communications," *J. Audio Eng. Soc.*, Vol. 42, No. 10, pp. 819 – 826.

[Gard98] Gardner, W.G. (1998). *3-D Audio Using Loudspeakers*, Kluwer Academic, Norwell, MA.

[Kevi98] Sowizral, H., Rushforth, K. and Deering, M. (1998). *Java3D API Specification*. Addison Wesley Longman

[Ming98] Ming Z., Kah-Chye T. and Er M.H., (1998) "Three-dimensional Sound Synthesis Based on Head-Related Transfer Functions," *J. Audio Eng. Soc.*, Vol. 46, No. 10, pp. 836 –844.

[Mit] http://sound.media.mit.edu/KEMAR.html

[Sen97] Sen M.K. and Canfield, G.H. (1997) "Dual-Channel Audio Equalization and Cross-talk Cancellation for 3-D Sound Reproduction," *IEEE Trans.On Consumer Electrics*, Vol. 43, No. 4, pp. 1189 – 1196.

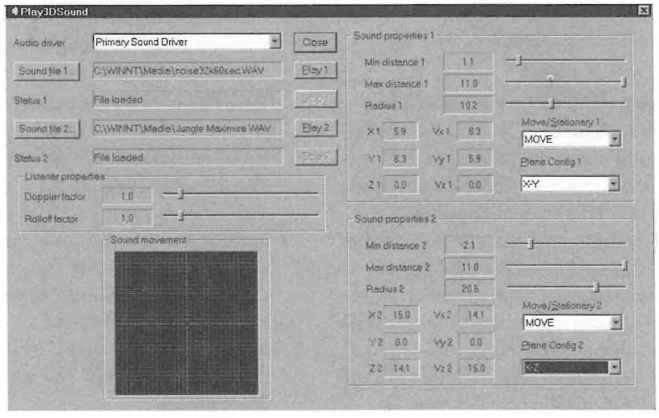

Fig. 2 Demo for the DirectSound API