OpenSG Symposium (2003)
D. Reiners (Editor)

TRIPS – A Scalable Spatial Sound Library for OpenSG

Tomas Neumann, Christoph Fünfzig and Dieter Fellner

Institute of Computer Graphics, Technical University of Braunschweig, Germany

Abstract

We present a Sound Library that is scalable on several computers and that brings the idea of a self-organized scenegraph to the Sound Library's interface. It supports the implementation of audio features right from the start at a high productivity level for rapid prototypes as well as for professional applications in the immersive domain. The system is built on top of OpenSG[12] which offers a high level of functionality for visual applications and research, but does not come with audio support. We show and compare the effort to implement audio in an OpenSG application with and without TRIPS. Today's audio systems only accept raw 3D-coordinates and are limited to run on the same computer and the same operating system than the application runs on. Breaking these constraints could give developers more freedom and ease to add high-quality spatial sound to their software. Therefore, users benefit from the promising potential OpenSG offers.

Categories and Subject Descriptors (according to ACM CCS): H.5.1 [Multimedia Information Systems]: Audio input/output H.5.5 [Sound and Music Computing]: Systems I.3.7 [Computer Graphics]: Virtual Reality
Keywords: High quality spatial sound, 3D-audio, cluster system, sound API, FieldContainer, rapid prototyping, game engine, immersive system

1. Introduction

Current developments on VR applications in general and of OpenSG[12] applications in particular demand a solution for 3D audio support, as positional sound is more important to both developers and users than ever before[2]. The faster software needs to be developed, the more efficient the interfaces need to be designed. And, the better soundcards get, the more users want to deploy high-quality spatial sound. Finally, if the visual system can benefit from the OpenSG Cluster System[13] then audio performance should also capitalize on it[3].

To have sound within a graphical application in many cases supports and intensifies the user's experience[1] with that software. Simple background music, interface sounds, ambience sounds or interaction-triggered sounds do not distract the user from the main focus, but makes it more realistic, impressive, and effective. Spatial sound improves the user's orientation and acceptance especially in immersive systems and games.

There are some sound engines available and there are some high-level APIs like DirectX for software development on specific operating systems. Just as graphics applications and computer hardware rapidly developed in the last 10 years, so did the sound and multimedia field. But in a Unix-based environment nearly all sound engines lack the ability to benefit from today's consumer soundboards, which have heavily progressed over the last years as well. These sound engines compute 3D sound calculation and mix audio channels in hardware. Modern 4G soundboards, which belong to the so called 4th generation, support high samplerates of 96KHz and a bitrate of up to 24Bit and 8 sound outputs[17]. Complex channel mixing and spatial positioning calculations, which are required to comply with I3DL2[9] specifications, can be supported by sound hardware.

Unix-based operation systems (OS) do not benefit from today's sound hardware because of poor driver support. In that case, spatial sound needs to be calculated in software on the host's CPU. Especially in immersive VR environments, like the DAVE[10] in Braunschweig, the VR server should be free from any unnecessary system load. The ideal solution here would be a sound library that runs on both the VR server as a dummy and a sound server that brings the audio to the sound device. With OpenSG it is even possible to connect computers with different OSs, so the sound server could run on Microsoft Windows and benefit from the newest drivers and hardware support.

Another way to benefit from an OpenSG sound library would be an optimized interface. Instead of dealing and calculating with world-coordinates for the 3D-sound position and updating them every frame, it would be convenient to simply 'glue' a sound to a node in the scenegraph and let it deal with updates. The TRIPS Library provides both features in a comfortable way.

Here, we first give a glimpse into spatial sound techniques and requirements and describe other libraries and their pros and cons. Later, TRIPS is presented from concept to usage and detailed implementation. We show results working with TRIPS on a single PC and in a cluster system using different sound hardware. We conclude with some thoughts and further research.

2. Spatial Audio Overview

Spatial Audio is calculated for a discrete point in space called the listener[11]. Just like a camera, the listener is defined by a position, up- and lookat-vectors. 3D Sounds that have to be mono sound files, are loaded into a channel of the sound driver. 2D Sounds, like background music, may be stereo and do not have a location in space. Higher samplerates result in better audio quality. The audible result of a 3D mixdown depends on several sound parameters like the position, volume, velocity and loudness. The position is used to calculate the distance to the listener. The volume sets the peak value of a sound. The listener's and the sound's velocity can be used to obtain a Doppler-effect just like when a fire truck drives by. The loudness is defined by minimum and maximum distance values where the volume peak is reached. The newest sound systems also support geometry-related audio calculation like obstruction and occlusion. For each frame, the sound engine calculates the sounds, mixes the channels and streams the audio result to the soundcard's physical line-outs. Headphones still provide the best spatial quality as calculated by the Head-Related Transfer Function (HRTF)[6]. When headphones cannot be used, you get a better 3D audio impression the more output the soundcard provides and the more speakers are connected. Obviously a Dolby digital[7] audio setup (five satellite speakers and a subwoofer) performs better than just a pair of stereo speakers. Figure 1 shows how spatial sounds within a virtual scene get approximated by a speaker set in a real life scenario.

3. Previous Approaches

Typically until now, the audio implementation is the last in the process of developing interactive visual applications[8] (if at all). Often this approach ends with unsatisfying software design, performance loss or poor audio ability. There are some sound engines and high-level APIs that could be used to add audio support to an OpenSG application, but

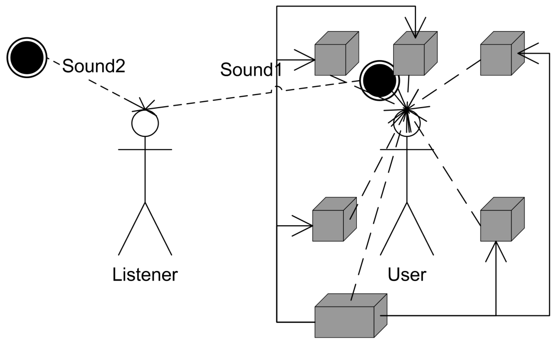

Figure 1: *3D sounds in a virtual scene with the listener and the real life setup with a user in the middle of a 5.1 Dolby digital speaker set.*

some engines are OS- and hardware-dependent like EAX[4] or DirectX. EAX is a hardware-based audio subsystem by Creative in its third generation with positional, reverb and geometric calculation. Creative did not allow licensing and emulation of EAX3 in software, like Sensaura[14] could do with EAX1/2. The full features of EAX3 are only available with the Windows OS. DirectSound is a part of Microsoft's DirectX which can be described as a standardized collection of APIs for multimedia usage. Other sound systems do not take advantage of sound hardware or they simply demand high license fees like $4,000 for a single Miles Sound System. VRJuggler[5] is a complex VR system with the ability to set up a cluster of render clients. Its sound module Sonix offers a simple and generic interface and it is possible to switch underlying sound engines like AudioWorks, Subsynth or OpenAL[15] at runtime. There was no information available whether a sound server can be set up on a connected Windows PC. The advantage of VRJuggler to sit on top of several graphical systems, even OpenSG, means that Sonix needs to accept raw positional parameters when using lowlevel OpenGL in the rendering layer. All systems or APIs listed do not benefit from the semantic structure of a scenegraph or provide an optimized OpenSG interface.

It is hard to compare sound systems under a Unix-based OS with those running on Windows. Pushed by industry developments in driver architecture, consumer soundcard's abilities and innovative hardware, Windows sound systems are always two steps ahead. Besides scattered soundcard support, there is no Unix-based implementation for audio obstruction and occlusion. The only two platform that offer obstruction and occlusion with FMOD[16] are Windows and the Xbox. EAX2/3 and reverb effects are also only available on Windows, Xbox, Sony's Playstation 2 and Nintendo's GameCube. Even if spatial positioning calculation can be done fast in software, Linux sound drivers, like OSS, still lack the newest techniques for sufficiently mixing multiple audio channels. The result is a poor spatial quality that could be simply verified by the human user. We chose FMOD as the underlying audio engine, because of the following obvious reasons. FMOD is widely platform independent, currently supporting seven systems. Its free for noncommercial use and it supports the most optimized sound APIs under Windows like DirectX or EAX3. It also supports multi-listeners, e.g. for split screen application, and multi dynamic DLL linking, e.g. to support more than one soundcard at the same time. Most consumer soundcards are not able to cascade their driver models so still a cluster solution is needed. FMOD is able to offer sufficient audio performance under Linux for stereo output, because a Dolby digital output is not yet possible with software calculation.

There are stringent rules how and where to implement audio within an application. First, the sound engine has to be initialized, then sounds are loaded. Changes of their sound parameters and playback commands are often completely spread over the code. Whenever a sound moves its new position has to be delivered in world coordinates to the sound engine. When all changes in a frame are done the sound engine is forced to update and calculate the sound. The corresponding call often is located at the end of the display routine. Four issues need to be pointed out in this context:

1. The listener and each sound position needs to be calculated into world coordinates and passed to the sound engine's interface.
2. The sound engine generates commands, which are directly conducted to the sound driver and then to the soundcard. The sound and the spatial quality depend on the computer's soundcard and OS the application runs on.
3. Currently, Linux is not a prevailing sound platform.
4. Cascading most consumer soundcards can currently only be achieved with cluster systems.

4. TRIPS Concept

The TRIPS sound library consists of two different OpenSG FieldContainers: The first is a TRIPS Audio Node, which holds the full audio context with the listener's position and depends from the camera, samplerate and other audio fields. And secondly a TRIPS Sound Object, which holds specific sound properties, like the sound file and a beacon to a scenegraph node, which is used for the spatial sound positioning and velocity. All TRIPS Sounds are organized in a soundlist multifield within the audio node. Figure 2 shows the different TRIPS FieldContainers and how they are bound to the OpenSG scenegraph. The TRIPS audio node does not have to be a child of the root node. It may sit next to it. The TRIPS sound objects also do not interfere with the scenegraph, but each of them is linked to a node with a beacon singlefield. The fact that they are part of OpenSG is important to benefit from the OpenSG cluster system, which is described later.

Most of the TRIPS internals are hidden from the user without limiting the flexibility for specific applications. So, with minimal additional code TRIPS adds complex sound to an OpenSG application.

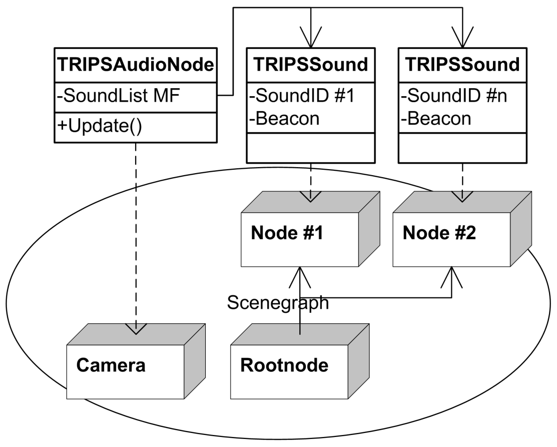

Figure 2: *Diagram of TRIPS FieldContainer and how they connect to the nodes in the scenegraph.*

4.1. TRIPS Usage

TRIPS provides an easy-to-use OpenSG interface for basic spatial sound support. With just a few lines of code the audio context is created and updated while sounds are being loaded and subsequently played. A TRIPS sound object is able to calculate its own world coordinates and velocity with the help of the node referenced by the beacon field. If that node moves within the scene or even in the scenegraph, the TRIPS sound object keeps track of its updated position and velocity. The example in Figure 3 demonstrates how a 3D sound of a car horn is bound to a moving car node in the scene, and 2D background music moving with the Listener. Only the Camera pointer is passed to the TRIPS audio node to calculate the correct listener vectors and update the audio engine. The sample code in Figure3 shows what is only needed to have background music and a fully working spatial sound with the OpenSG SimpleSceneManager class.

TRIPS can be run in several settings. The PC that the application runs on can generate the sound locally, can be connected to a dedicated sound server with the cluster system, or both. Depending on the host's OS and the sound device both speaker settings and used sound drivers can be changed. Different configurations can be set through environment variables such as audio type, audio mode, driver selection or connected speaker setup. By changing just two variables, the same application can first generate local stereo sound on a Linux PC and then trigger a Windows sound server, which can produce full quality spatial sound at the connected Dolby digital speakers.

4.2. TRIPS Details

The TRIPS Audio Node (see Figure 4) holds the complete audio context and all other data that affects the sound engine or the usage of the cluster system. The listener position,

```
// audio globals
TRIPSAudioNodePtr   audio;
TRIPSSoundPtr       music, car;

int main(int argc, char **argv){
[..]
  osgInit(argc,argv);
  audio = TRIPSAudioNode::create();
  // creates global AudioNode "audio"
  audio->init(false);
  addRefCP(audio);
  // raises Ref-Counter of audio

  music = audio->addSound(
    "background.wav"); // 2D filename
  // start playback looped
  music->setVolume(100);
  music->play(TRIPSSound::LOOP);

  car = audio->addSound(
    "media/horn.wav", // 3D filename
    carnode,          // Beacon
    TRUE);            // automatic velocity
  // start playback every 10 seconds
  car->play(TRIPSSound::EVERY, 10);
[..]
}

void display ( void ){ // each frame
[..]
  // get camera from SimpleSceneManager
  CameraPtr cam = ((mgr->getWindow())
    ->getPort()[0])->getCamera();

  audio->update(cam); // TRIPS Update
}

void key(unsigned char k, int x, int y){
  switch(k){
    case 27:
      subRefCP(audio);
      exit(1);
  }
}
```

Figure 3: *Complete Sample Code to setup TRIPS with a looped 2D- and a moving 3D-Sound*

the listener lookat- and up-vectors and velocity are private member variables and can only be changed by calling the Update()-method with a camera pointer. The addsound() and subSound() methods change the soundlist multifield. Info() gives some details about available sound devices and lists the number of hardware voices. The Audiomode can be set via an environment variable to either be ACTIVE or MUTE. This switches the local sound processing. If a cluster system is running, the important fields get synchronized, as explained later in more detail.

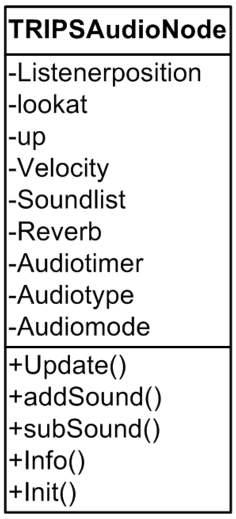

Figure 4: *TRIPSAudioNode FieldContainer.*

A TRIPS Sound object (see Figure 5) offers methods to control the playback of a sound. A sound can be played once, in a loop, every discrete seconds, or randomly within limits, as triggered all by the Play()-method and playmodes like: ONCE, LOOP, EVERY or RANDOM (see also Fig. 3). It can be paused and un-paused using Pause() and may also be stopped. This is done with a sound-to-play field, a playmode and two additional parameter fields, which all get synchronized the same way a cluster system works and then get parsed in the changed()-method. The method setVolume() changes the peak volume and setMinMax() changes the distance this peak is reached and where the sound fades out.

4.3. TRIPS changed()-Method

In OpenSG, changes to data fields must be encapsulate between beginEditCP() and endEditCP() to mark them as manipulated. On a cluster system but also on a single host OpenSG calls a changed()-method, which can then trigger special behavior somewhere else, both on the same PC but also on a connected PC. The TRIPS play command does not call the sound engine directly but it triggers the changed()-method on the same host and on connected PCs. This way the play command travels from the application server to the sound server and starts the playback there on its local soundboard. This way not just a single, but several sound servers can be added to the cluster. Each of these sound servers can have a different listener position as specified through an offset to the camera position.

5. Results

High-quality spatial sound can be achieved with some costs. Often the sound driver performs additional calculations. In

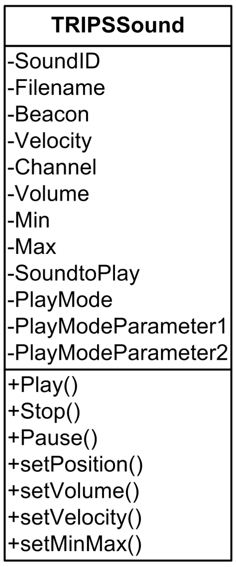

Figure 5: *TRIPSSound FieldContainer.*

order to quantify how they affect the system as a whole, the overall performance was measured in a number of test runs. An interesting question is the impact of the sound's referenced node depth within the scenegraph and calls to the sound engine on the results.

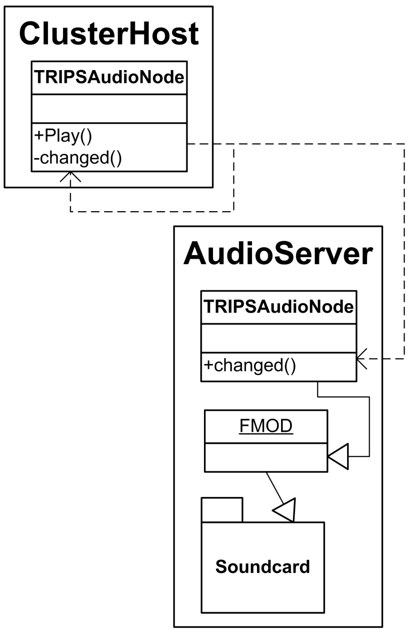

Figure 6: *Example how the play()-command triggers the changed()-method locally and on a cluster PC.*

5.1. Benchmark Scene

The benchmark scene is composed of 64 spheres, randomly falling through a cubic volume, whose tessellation contains about 330,000 triangles. Each sphere node, which moves every frame, can have a TRIPS sound bound to it. The spheres can be inserted in the scenegraph at two different depths, to test the cost of additional matrix multiplications. The idle function and any GLUT interaction were disabled and the test went on for 3,000 frames. The tests were run on three different systems and on two different platforms.

As a first result, we found that sound calculation with and without specific hardware support cannot be compared because of the limited speaker output. It is not really adequate to compare a TRIPS hardware Dolby digital sound setup with six channels to a stereo output computed in software under Linux. There is a remarkable difference in audio quality that no benchmark can show, and further more the OSS sound driver blocked the start of the application for up to 30 seconds.

5.2. Overall Performance

Figure 7 shows the render time to produce the 3,000 frames with a Creative Audigy soundcard with 0 to 64 3D sounds. The quality spatial sounds have a negative impact, while the stereo mix in software shows a quite stable performance. Both tests were run on a Windows AMD Athlon-XP 1.4Ghz system.

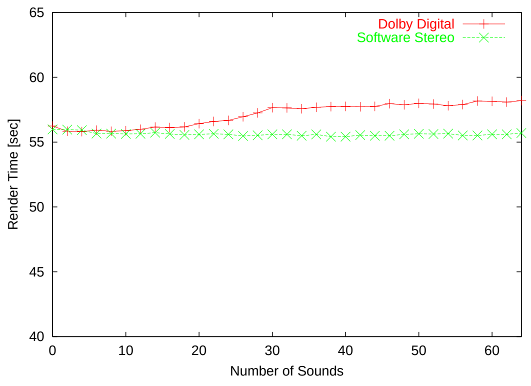

Figure 7: *Performance of Dolby digital sound with hardware support versus software calculated stereo.*

5.3. Update Test

Figure 8 shows the time needed in the 3,000 frames sequence to calculate the new positions for all sounds and the calls to FMOD for updating the listener position and all sounds managed by the TRIPS library. The sound position is derived from the world volume of the beacon node. Calculation of this world volume requires the to-world matrix, which is the product of all transformation matrices above. In this test the spheres hang in a depth of 3 and 9 in the scenegraph. The sound calculation in software scales linear to the number of sounds. The FMOD calls for hardware support take longer and deviate at 32 sounds because the Audigy soundcard only supports 32 true hardware sounds. Note that in software mode only stereo output is produced. All tests ran on a Windows AMD Athlon-XP 1.4 system.

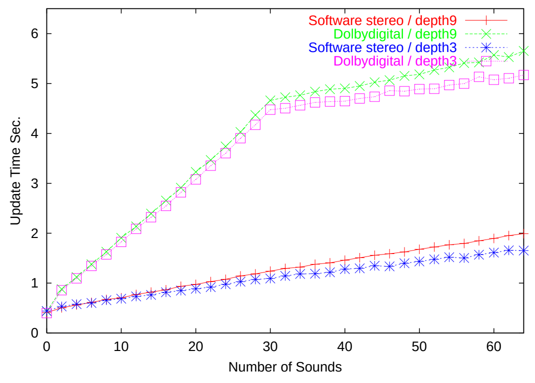

Figure 8: *The Update Test shows different scenegraph depths and extra load for hardware support.*

5.4. Cluster Test

Figure 9 again shows the render time of 3,000 frames. This time it is a Linux DualP3 1Ghz that normally hosts the OpenSG cluster system for a VR system of eight render servers to run the DAVE. In both tests one dedicated soundserver running on Windows was connected. Even with the low load that stereo software mode generates, as seen in Figure 8, the render time is slightly better with no local sound and a connected sound server. To clear any nonessential load from the cluster host is a good argument for a dedicated sound server. The host then does not calculate any transform matrices for the sounds or sets any calls to the sound engine.

6. Conclusion and Future Work

It has been shown how easy it is to add audio and spatial sound to an OpenSG application using TRIPS. Loading, playing and binding a sound to a scene node is simple. The developer does not have to worry about updates to the listener or sound positions and velocities, but one single TRIPS code line can take care of this. This was possible only by bringing the sound interface to a higher semantic level by defining sound positions with beacon nodes. TRIPS also ensures high-quality spatial sound on a local PC running on Windows and it offers the same quality when a dedicated sound server is connected to the OpenSG cluster system. But high-quality spatial sound like Dolby digital does not come for free. The sound driver can consume considerable time

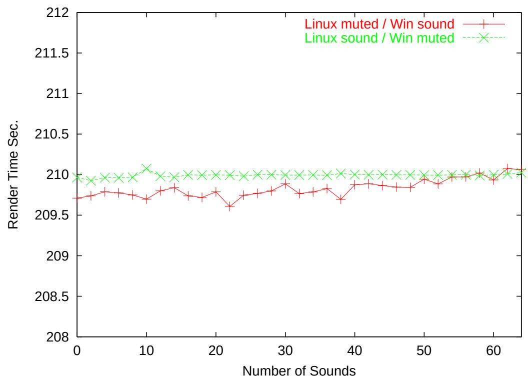

Figure 9: *The Cluster test shows a slight performance advantage with a dedicated sound server.*

which has negative effects on the overall performance. If the future brings even more realistic and complex spatial sound calculations, this will cause more load on both the CPU and the system. The effect of different node depths in the scene can be clearly seen. This effect could be easily remedied by an optimization where a flag is set when a node has changed its position. This would save a to-world matrix calculation per sound as well as the corresponding call to the sound engine.

It is a remarkable result that using the OpenSG cluster system the additional load to run stereo spatial audio in software is marginal. A cluster system running on Windows would experience more performance loss when producing high-quality spatial sound. Still the combination of TRIPS and OpenSG offers to move this load transparently to a dedicated sound server.

The developer can create quite complex audio scenes with the very lean interface of TRIPS. If desired, more features can easily be used by simply adding FMOD commands directly to the application, if TRIPS runs actively on the local host. Another possibility is to create trigger fields in order to let them get synchronized with the help of the changed()-method and to execute the FMOD calls on a remote PC.

In the future, TRIPS will be extended by adding more sound features, hopefully without losing the advantage of easy usage. But new sound hardware will bring new benefits to the user and offers more realistic audio processing, by taking into account the 3D geometry of the scene for mixing the audio or by mixing together different sound rooms with different reverb effects.

A DAVE setup or a powerwall-like setup bring an immersive graphical impression of a virtual scene to the user. With TRIPS both setups can be transferred to the audio field. Several sound servers with different listener positions could present an immersive sound impression to the user, because several sets of Dolby digital speakers could be used. This would then be done in a parallel setup, with two or three speaker sets ordered vertically. Alternatively a set can be rotated by 90 degrees in order to have the sets ordered horizontally and a vertically. In analogy to a visual powerwall, an audible powerwall is possible by placing several sound servers and their speakers next to each other with the correct listener offset thereby creating a continuous sound impression.

Acknowledgements

We gratefully acknowledge the support by the German Federal Ministry of Education and Science (**BMB+F**) in the OpenSG PLUS project under grant 01 IRA 02G and the hardware support by **Creative** and **Terratec**.

References

1. Durand R. Begault. 3-d sound for virtual reality and multimedia. Technical Report NASA/TM-2000-0000, NASA, 2000. 1

2. NVIDIA Technical Brief. NVIDIA nForce platform processors audio processing unit, 2002. http://www.nvidia.com/docs/lo/2027/SUPP/APU-_TechBrief_71502.pdf. 1

3. David A. Burgess and Jouke C. Verlinden. An architecture for spatial audio servers. Technical Report 93-34, GVU Center, 1993. 1

4. Creative. EAX, 2000. http://eax.creative.com/developers/. 2

5. Carolina Cruz-Neira. VRJuggler, 2001. http://www.vrjuggler.org. 2

6. Gastier D. J. and F. L. Wightman. A model of head-related transfer functions based on principal components analysis and minimum-phase reconstruction. *Journal of Acous. Soc. Am.*, 91(2):1637–1647, 1992. 2

7. Dolby. Dolby-digital technical information, 2003. http://www.dolby.com/digital. 2

8. V. Gal, C. Le Prado, J.B. Merland, S.Natkin, and L. Vega. Processes and tools for sound design in computer games. In *Proceedings International Computer Music Conference*, September 2002. http://deptinfo.cnam.fr/Enseignement-/DESSJEUX/infoeleves/ICMC20025.pdf. 2

9. Interactive Audio Interest Group. Interactive 3d audio rendering guidelines level 2.0, September 1999. http://www.iasig.org. 1

10. Sven Havemann. Höhlenzeitalter. *iX Magazin für professionelle Informationstechnik, Heise-Verlag*, (11), November 2002. http://www.graphics.tu-bs.de/dave. 1

11. Juriaan D. Mulder and Edoh H. Dooijes. Spatial audio in graphical applications. Technical Report CS-R9434, CWI Netherlands, 1994. 2

12. Dirk Reiners, Gerrit Voß, and Johannes Behr. OpenSG - basic concepts. In *Proc. of OpenSG Symposium 2002*, 2002. http://www.opensg.org. 1

13. Markus Roth. Integration paralleler rendering-verfahren für lose gekoppelte systeme in OpenSG. In *Proc. of OpenSG Symposium 2002*, 2002. http://www.opensg.org. 1

14. Sensaura. Sensaura website, 2001. http://www.sensaura.com. 2

15. Loki Software. OpenAL specification and reference, 2001. http://www.openal.org/info. 2

16. Firelight Technologies. FMOD, 2003. http://www.fmod.org. 2

17. VIA. Via envy audiochip, 2003. http://www.viatech.com/en/Digital 1