

Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science

Bachelor Thesis

Integration of 3D Audio into Virtual Reality

Author: Wolfgang Burgard
Supervisor: Prof. Dr.-Ing. Philipp Slusallek
Advisors: Michael Replinger,
Dmitri Rubinstein
Reviewers: Prof. Dr.-Ing. Philipp Slusallek,
Prof. Dr. Joachim Weickert
Date of Submission: 09. April 2009

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement under Oath

I confirm under oath that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, _____
(Datum / Date)

(Unterschrift / Signature)

Contents

1	Introduction	2
1.1	Motivation	2
1.2	Goals	3
1.3	Structure	4
2	Basics	6
2.1	Scene Graphs	6
2.2	3D Audio	8
2.2.1	3D Audio Generation	8
2.2.2	3D Audio Playback	10
3	Related Work	12
3.1	Standards for 3D Contents	13
3.1.1	Requirements	13
3.1.2	COLLADA	13
3.1.3	U3D	13
3.1.4	X3D	13
3.1.5	Sound Component	14
3.1.6	Conclusion	15
3.1.7	X3D/VRML Browser and Player	16
3.1.7.1	Test Description and Results	16
3.1.7.2	Summary	17
3.2	Scene Graph Runtime Libraries	20
3.2.1	RTSG	21
3.3	Audio/Multimedia software	21
3.3.1	3D Audio Libraries	21
3.3.1.1	FMOD	22

3.3.1.2	Miles Sound System	22
3.3.1.3	OpenAL Soft	22
3.3.2	SDL	23
3.3.3	Audiere	23
3.3.4	Multimedia Middleware Solutions	23
3.3.4.1	Multimedia Processing with Flow Graphs	23
3.3.4.2	Local Operating Frameworks	24
3.3.4.3	NMM	24
3.4	Conclusion	25
4	Architecture	26
5	Integration of 3D Audio into NMM	28
5.1	3D Audio Flow Graphs	28
5.2	Demultiplexing Multichannel Audio	30
5.3	3D Audio Post-processing	31
5.3.1	Objectives	31
5.3.2	Implementation	32
5.3.2.1	Simulation	32
5.4	Buffer Shaping	34
5.5	Integration of OpenAL	35
5.5.1	Objectives	35
5.5.2	Design Pattern	36
5.5.2.1	Motivation	36
5.5.2.2	Problem	37
5.5.2.3	Solution	37
5.5.2.4	Principle	39
5.5.2.5	Application	40
5.5.3	Interfaces	42
5.5.4	Synchronization	43
5.5.4.1	Motivation	43
5.5.4.2	Principle	43
5.5.4.3	Implementation	44
6	Integration of 3D Audio into RTSG	46
6.1	Implementation of X3D Sound Nodes	47

6.2	Sound Renderer for RTSG	49
6.2.1	SceneSoundSystem Architecture	49
6.2.1.1	Motivation	49
6.2.1.2	Design	49
6.2.1.3	Operation Principle	51
6.2.1.4	Features Computations	53
6.2.2	NMMSoundRenderer	61
6.2.2.1	Initialization	61
6.2.2.2	Deinitialization	62
6.2.2.3	Processing Scene Updates	62
7	Conclusion	64
8	Future Work	66
8.1	Distributed Sound Processing	66
8.2	Enhanced Audio Realism	66
8.3	Extensions to X3D Sound Component	67
8.4	Alternative Audio Playback	67
A	Ellipse evaluation	68
B	Scene Documents	71
B.1	Minimalistic Audio Scene	71
B.2	Multichannel Scene	72
	Bibliography	73

List of Figures

2.1	Sample Scene Graph	7
2.2	3D Audio Generation and Playback	9
3.1	X3D Sound Geometry	15
3.2	Pass-through Intensity Attenuation	18
4.1	3D Audio Components	27
5.1	Sample 3D Audio Flow Graph	29
5.2	Demultiplexing Stereo Audio	31
5.3	Sound Propagation	33
5.4	Locale Synchronization	37
5.5	Synchronous Playback of 3D Audio Nodes	39
5.6	Application of Half-Sync/Half-Async pattern	40
6.1	Time-dependent Node Execution	48
6.2	Design of SceneSoundSystem	50
6.3	MultiChannelSplitStrategies	56
6.4	Ellipse Evaluation for X3D Sound Geometry	60
A.1	Ellipse parameters	69

Abstract

This work is about the design and implementation of software to realize audio playback for animated three dimensional (3D) virtual environments. In order to unify the representation and simulation of such virtual 3D environments an standard as X3D is needed. Furthermore, for virtual reality applications the audio playback for such virtual 3D environments is required to be realistic. However, existing X3D software applications have certain limitations considering the realism of their audio playback.

The audio playback software layer presented in this work overcomes these limitations. By employing a flexible multimedia middleware the audio accuracy can be scaled in order to adapt to the available computation and audio playback hardware. An audio library has been integrated into this middleware as one backend for audio playback of virtual 3D environments.

For simulating the virtual environment a framework being conform to the X3D standard has been coupled with the middleware.

Chapter 1

Introduction

1.1 Motivation

Today many entertainment applications like video games require plausible audio playback. Not only the sensation feeling of music and sounds play an important role, but also the orientation supporting effect. For example in a car racing game the music can accentuate thrilling sections of the race and sounds can help the player to locate opponents being outside the current view by listening to the audio playback.

Besides the field of entertainment, **Virtual Reality (VR)** also makes use of audio playback for virtual environments but in this context the audio realism is in the focus. An example for a VR application where audio is in the center of attention is the design of a concert hall [\[46\]](#).

These two examples of applications need different audio accuracy levels due to their different objectives, namely realism and entertainment.

On side of software there has to be flexibility in the degree of realism. This allows to run the software application with different available computation resources. For example it may be required for the racing game that the audio computations can be accomplished by a mobile device with low computational power. Whereas the concert hall simulation is meant to be processed by a whole cluster of computers.

Accuracy of the acoustics in the virtual environment can be adjusted by e.g. toggling simulating of natural sound phenomena. For example a thunderstorm simulation benefits if thunders are audible with a certain delay after the lightnings have been visible. Simulation of such phenomenas is often disregarded by available applications displaying virtual environments as it can be read in chapter 3.

The selection of audio hardware employed for playback also allows different accuracy levels: For example the mobile gaming device uses stereo loudspeakers to play back the audio. By contrast, the concert hall simulation realizes audio playback by a sophisticated audio system consisting of a huge speaker array as required for IOSONO [11] sound playback technology.

The generic solution provides flexibility for both aspects, namely, audio hardware and software and can be adjusted to the application specific needs.

1.2 Goals

The general idea of this work is to provide a software framework having flexible accuracy for audio playback of virtual 3D environments. There is a software layer realizing playback for 3D virtual environments and a software layer simulating the virtual 3D environment. Into the audio playback natural sound phenomenas can be integrated to scale audio accuracy.

The goals of this work are:

1. Examine and discuss existing software and hardware for audio playback of virtual 3D environments.
2. Design and develop a framework to play back audio for virtual 3D environments using appropriate audio hardware. The framework has to be scalable such that different audio accuracy levels can be achieved. Furthermore a transparent access to devices in a network is required to allow distributing audio computations. This enables high accuracy audio playback with several computers as well as reasonable audio playback on low computation devices.

3. Integrate the developed audio software layer into a framework simulating virtual environments.
4. Allow playback of multichannel audio data within the virtual 3D environment

1.3 Structure

In chapter 2 I firstly introduce a structure to represent 3D environments. Subsequently this chapter provides basic knowledge how to produce audio for virtual 3D environments and accuracy levels are discussed. Audio software as well as hardware aspects are examined. Together with chapter 3 describing the evaluation of available software in context of audio for virtual 3D environments goal 1 of this work is addressed.

Furthermore, chapter 3 presents existing software among which suitable ones form the software basis to realize goal 2 and goal 3.

Chapter 4 gives an overview of the components incorporated to produce audio for virtual 3D environments. The interplay of the audio software and the virtual environment simulation software to accomplish goal 3 is explained.

Realization of goal 2 is covered by chapter 5. There an audio library for playback of virtual 3D environments is integrated into the selected multimedia middleware as well as further plugins to enrich realistic audio playback.

The integration of the audio software layer into the framework simulating the virtual environment is described in chapter 6 as well as the way goal 4 is accomplished, namely the implementation of strategies how to handle multichannel audio data.

Chapter 7 summarizes how the goals have been achieved.

An outlook how this work can be extended and ideas to enrich audio playback for virtual environments are described in chapter 8.

In appendix A the formulas to evaluate ellipse equations in polar form in dependency of certain parameters are described. These have been applied for audio computations.

Appendix **B** lists some documents describing virtual environments which have been referenced in this work.

Chapter 2

Basics

In order to realize playback of audio for virtual 3D environments a representation of the virtual environment is required. A convenient structure therefore is presented in section 2.1. Furthermore the audio software needs to perform certain computations and feeds audio playback hardware as described in section 2.2.

2.1 Scene Graphs

A *scene graph* is a data structure arranging elements of a scene conceptual and/or spatially. Scene graphs are a model-centric approach where model geometry, size, appearance, materials, relative locations, and internal relations are expressed as a directed acyclic graph [2, page 7/8]. Data functionality is collected in nodes which can be connected in a certain order by an edge, e.g. from parent to child. A node can have several parents thereby that the node can be referenced. Each reference of a node and the node itself are individual elements of the scene. Figure 2.1 illustrates such a scene graph containing a referenced node.

A motivating example for the employment of a scene graph is a scene of a vehicle described by geometries, a light and a sound as depicted in figure 2.1. Besides the meaningful grouping of the scene elements it is a convenient feature that all children of a certain node can be considered as being placed relative to the parent node. This allows for example to animate several elements in the scene

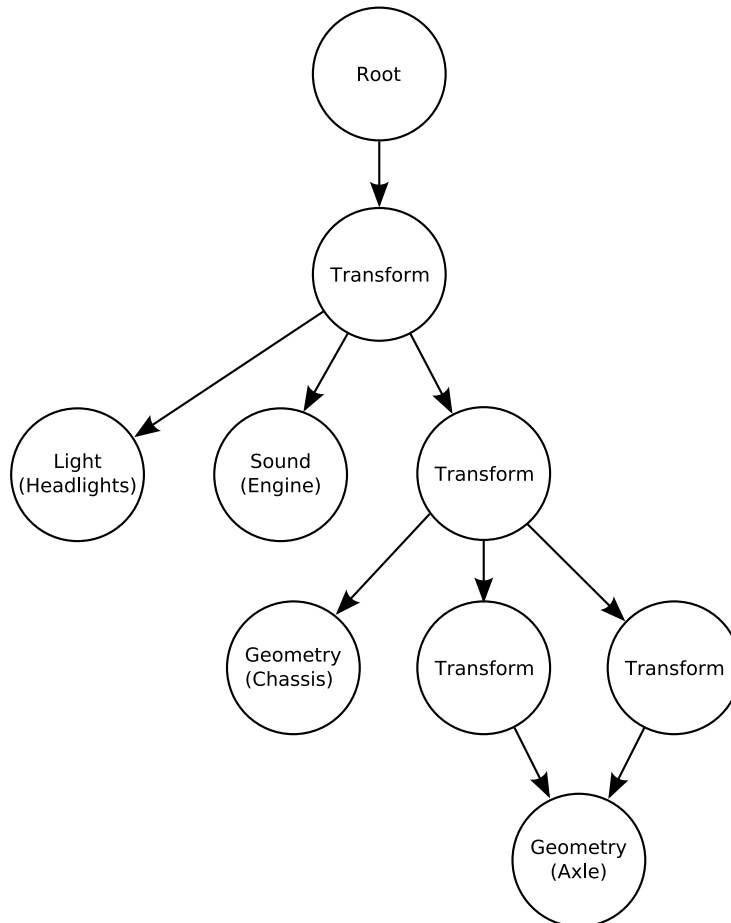


Figure 2.1: Scene graph for a vehicle consisting of geometries, a light and a sound. Light and sound have the same transformation in the virtual environment defined by their parent common **Transform** node. One **Geometry** node represents the geometry of a vehicles axle. Since this node is referenced and has two different parents there are two axles in the scene with individual transformations. The other **Geometry** node comprises the rest of the vehicles geometry being located relative to the axles, light and sound.

by just modifying one transformation. In figure 2.1 modifying solely the top transformation node can move the whole car and not just only one of the axles.

2.2 3D Audio

Within the scope of this work, *3D audio* comprises techniques to generate acoustics of virtual 3D environments and the playback of these virtual acoustics using appropriate audio hardware. The audio playback has the goal to resemble the acoustics of a real environment corresponding to the virtual 3D environment. *(3D) sound rendering* means the process to perform all computations required for 3D audio apart from the playback techniques the audio hardware, like surround loudspeaker systems, applies to reproduce the virtual acoustics in reality.

A sound emitter in the virtual 3D environment which has been processed by 3D audio computations is called *spatialized sound emitter* or *spatialized sound source* within this thesis.

2.2.1 3D Audio Generation

Given a virtual 3D environment, *3D audio generation* refers to the process to create virtual acoustics simulating reality. Typically, at least a listener in the virtual environment and sound emitters located in the virtual environment are taken into account as depicted in the right figure of picture 2.2. The listener can be parameterized by the orientation, the position and the speed. Sound emitters can have parameters like their position, direction into which the sound waves primarily propagate and the movement speed of the sound emitter itself. The greater the distance of a sound emitter to the listener in the virtual environment is the quieter the sound emitter is from the perspective of the listener.

The geometric approach applies *attenuation by distance* for each sound emitter [46]. A *distance model* describes how this distance and the corresponding intensity attenuation are associated. Applying such a distance model can be regarded as minimal accuracy of 3D audio generation and is typically done by 3D audio libraries as described in section 3.3.1. For directional sound sources a *directional model* can be applied which models that the sound source acoustics is not the

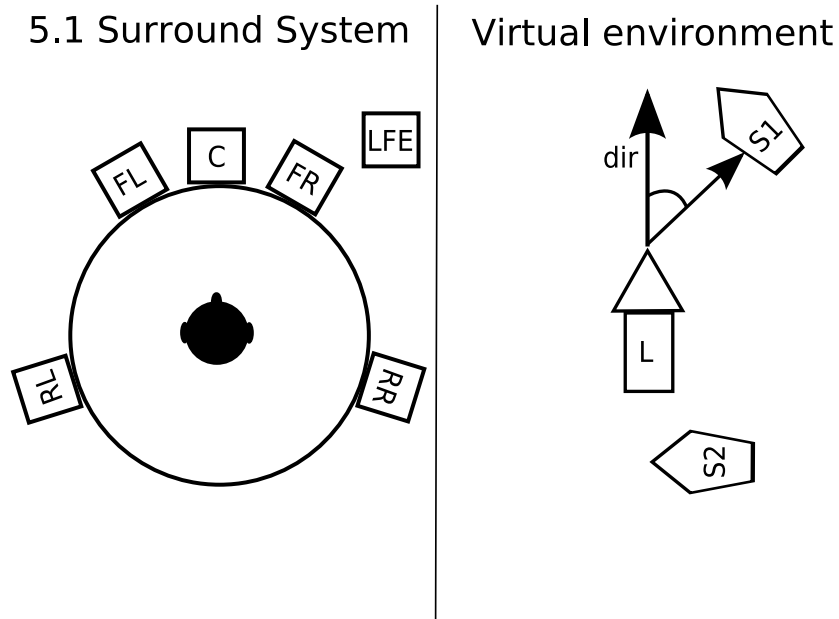


Figure 2.2: Illustration of real and virtual objects in 3D audio playback and generation. The left side illustrates a 5.1 surround loudspeaker-system constellation where the loudspeakers have the same distance to the audience. The center of this constellation corresponds to the sweet spot where the playback accuracy is the highest. The right side illustrates a virtual environment having a listener (L) and sound emitters (S1, S2) having individual positions and directions. The angle between the direction of the listener and the vector to a sound emitter plays an important role for the playback of this sound emitter

same from all directions in virtual environment. The simplest way to model this is to attenuate the intensity depending on the direction from the sound source to the listener in the virtual environment. For example it is intuitive that a directional sound source is perceived louder if the listener is in front of a sound source as perceived from back of the sound source even if the distance is the same.

The 3D audio accuracy can be increased among other techniques by incorporating natural sound phenomena like Doppler effects or considering geometries of the virtual environment to simulate the sound muffling effect occurring when a sound emitter is occluded by some geometry from the perspective of the listener. Furthermore, *ray tracing*[43] methods can be applied to find reverberation paths between sound emitters and the listener. Rays are generated emanating from sound sources and followed through the environment until an appropriate set has reached the listener. The problem of these geometric models is that approximation errors and computational complexity increases with larger numbers of reflections and diffractions [46].

Artificial reverberation can be provided by parametric models implemented as digital filters which is commonly used for video games. But this method creates synthetic reverberation effects and can not be considered as a accurate acoustic model [46].

The *finite and boundary element method*[44] approximates solutions for wave equations. In the limit the solution of the wave equation is accurate but it is time and memory consuming and thus not generally used for interactive virtual environment applications [46].

2.2.2 3D Audio Playback

Once the 3D audio for the virtual environment has been generated the virtual acoustics have to be transformed into acoustics in reality. For this conversion several approaches exists with different accuracies employing various audio hardware.

One commonly used technique is *intensity panning* [45] where the intensity of all sound emitters in the virtual environment is individually adjusted for each channel of the audio hardware. The intensity of an individual sound emitter

in a certain channel depends on the relative placement of the sound emitter to the listener in the virtual environment, e.g. on the angle between the listener direction and the vector from the listener to the sound emitters. The right figure of picture 2.2 illustrates this angle and applying intensity panning for the sound emitter S1 causes that it can be heard on the loudspeakers of the 5.1 surround loudspeaker-system ordered by intensity as follows: front right (FR), center (C), front left (FL), rear right (RR) and rear left (RL).

Intensity panning using multichannel audio hardware is superior to the capabilities of monophonic or stereophonic playback [42]. Stereophonic playback using two loudspeakers allows the audience only to locate the sound emitter on a line not in 3D virtual space. Monophonic playback does not reflect any degree of freedom in the placement of the sound source.

Playback using surround loudspeaker-systems has the problem that in a very limited area the acoustics of the virtual environment are approximated reasonably and when leaving this so called *sweet spot* [47] the accuracy ceases rapidly. This makes it impossible to achieve realistic audio playback for a huge audience.

To obtain a higher accuracy level of 3D audio playback for a huge audience *wave field synthesis* [47] audio systems like IOSONO can be employed. The major difference is that the sound waves of the virtual environment itself are reproduced using many secondary loudspeakers and not only the intensity of the loudspeakers simulate directional effects. Wave field synthesis audio systems achieve in the whole area surrounded by these loudspeakers the correct audio playback. This high level of accuracy requires a large array of loudspeakers and heavy processing resources to feed all the channels [46].

Chapter 3

Related Work

In this chapter I present and discuss work related to virtual 3D environments as well as audio and multimedia.

Section 3.1 describes which standards exist to represent and simulate virtual 3D environments. One of these standards is worked out to be suitable for virtual 3D environments containing sound elements. Applications orientated on this standard have been tested with respect to audio playback capabilities.

In section 3.2 I present scene graph libraries with focus on these which conform to the appropriate 3D standard. Such a scene graph library simulates the virtual environment and thus is required to achieve goal 3 of this work.

Libraries to realize 3D audio for virtual environments in real-time are presented in section 3.3.1. Section 3.3.4 presents multimedia middleware solutions among which NMM [20][21] has been selected to integrate one 3D audio library. Employing a middleware is required to allow distributed audio computations.

Finally section 3.4 concludes with the results from the evaluations and names the libraries which are employed to realize the goals of this work.

3.1 Standards for 3D Contents

3.1.1 Requirements

The primary requirement on the standards examined in this section is that they allow to represent 3D content with at least graphics and sound elements in the virtual 3D environment. It has to be possible to animate the objects in the virtual environment in order to create dynamic scenes. Furthermore the standard has to come with a run-time model defining how animations are realized in the virtual environment. Thereby it is not left to the software application to bring live into the scenes which allows to compare applications concerning their audio playback properties. Such a comparison is described in section 3.1.7.

3.1.2 COLLADA

COLLaborative **D**esign **A**ctivity (**COLLADA**)[\[4\]](#)[\[3\]](#) is an intermediate format for representing structured 3D data in XML. This standard does not include a run-time model. Furthermore, it has no representation for sound elements in the scene [\[5\]](#).

3.1.3 U3D

The **U**niversal **3D** (**U3D**) File Format is primarily intended for downstream 3D CAD repurposing and visualization purposes [\[6\]](#). The standard includes an run-time architecture but animated virtual environments with sounds can not be realized since audio contents are not integrated.

3.1.4 X3D

X3D is a royalty-free open standards file format and run-time architecture to represent and communicate 3D scenes and objects using XML[\[9\]](#).

Historically X3D [\[1\]](#)[\[2\]](#)[\[3\]](#) is based on the **V**irtual **R**eality **M**odeling **L**anguage (**VRML**) [\[7\]](#)[\[8\]](#) which is used by many applications in the field of modeling,

computer graphics and virtual reality ¹.

In X3D virtual environments are modeled with scene graphs consisting of nodes. These nodes have fields of certain types like `SFVec3f` (single 3D floating point vector) or `MFVec3f` (set of 3D floating points) describing the state of the node. To realize hierarchies of nodes a field can also be of `SFNode` (single node) or `MFNode` type, i.e. a node can have nodes as children.

The employed scene graph allows to reuse subgraphs by referencing nodes as already described in 2.1. This has for example been done in the scene described by the document in appendix B.2.

A way to enable animations in X3D scenes is to send field-values between nodes by so called *routes*. Additionally it is possible to integrate scripts of a certain script language e.g. JavaScript into a scene.

3.1.5 Sound Component

The sound component of X3D [10] specifies aspects of sound rendering for a X3D scene. Basic objects for sound rendering namely sound sources and a listener can be described in a X3D scene graph. The geometric model for sound sources in the virtual environment is depicted in figure 3.1. One drawback of this model is that the playback of a sound source is sharply cut off when the listener in the scene leaves the outer ellipse.

Further shortcomings of X3D in context of sound rendering are:

- Missing material nodes with acoustic properties, e.g. acoustic reflection parameter
- No access to audio data within the scene. That implies there is no possibility to manipulate the audio data within the scene graph respecting the semantics of X3D. Consequentially sound effects like Doppler shifts can not be simulated in a scene if the sound renderer does not support this feature
- Explicit synchronization of sound sources is not mentioned although it is intuitive that playback for multiple references to the same node has to be

¹although X3D/VRML has to be extended for immersive virtual reality applications [50]

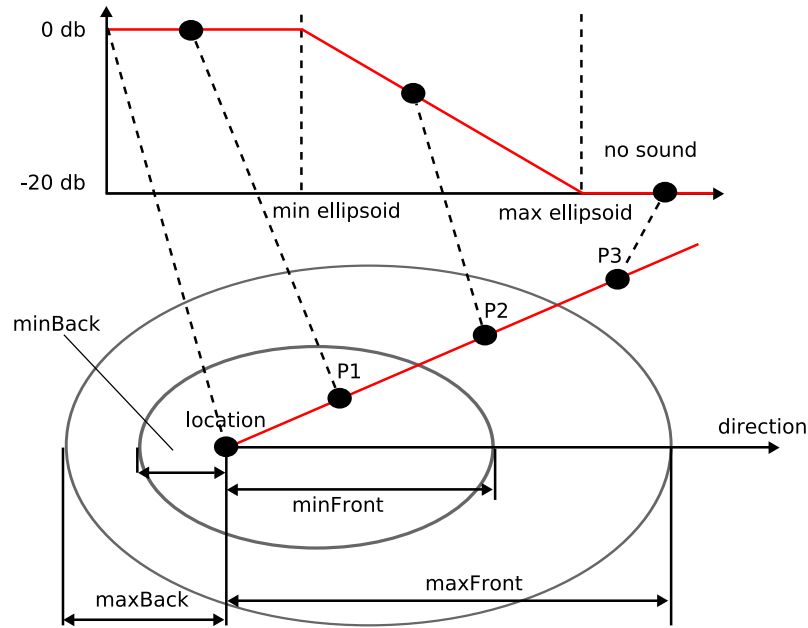


Figure 3.1: Illustration of X3D sound geometry. This is a directional and distance attenuation model determining the intensity of a sound source perceived at a listener in virtual 3D environment. The lower figure shows a sound source in virtual environment having a `location`, `direction` and parameters for describing two ellipses. The inner ellipse, described by `minBack` and `minFront`, defines the region where the sound source can be heard at its maximum intensity. Outside the outer ellipse, extended by `-maxBack` and `maxFront` in `direction` direction, the sound source is not audible at all. Between inner and outer ellipse linear interpolation of the intensity ranging from 0 dB to -20 dB is applied as depicted in upper figure. P1, P2 and P3 are sample positions of the listener.

be synchronized to avoid undesired echoes

- Lack of a precise definition or proposals how multichannel sound sources are to be spatialized. X3D does barely postulate to keep channel separation

3.1.6 Conclusion

Concluding U3D and COLLADA have not integrated any audio elements into their 3D contents. However X3D constitutes an appropriate representation and

run-time architecture for 3D scenes with a sufficient sound component. Drawbacks and advantages of X3D have been shown.

3.1.7 X3D/VRML Browser and Player

3.1.7.1 Test Description and Results

This section presents the sound rendering capabilities of available X3D/VRML software applications. The goal is to find out the state of the art of current X3D/VRML applications in context of sound rendering, i.e. what are desirable audio features of these applications and what can be done better.

The applications have been tested practically by running them on several X3D conform test scenes.

Table 3.1 shows the results of the test series.

The second column lists the methods how sound sources with assigned multi-channel audio data have been treated. Three different methods occurred:

1. **Monohandling** where either the audio data of all channels are mixed into one mono channel. This way a sound source with mono data emerges which can be spatialized. Or for each channel a mono sound source is spatialized with all having the same position.
2. **Pass-through** playback of a multichannel sound source means that the assignment of the audio data to individual channels in the input format corresponds to the assignment of the audio data to the loudspeakers, e.g. audio data in the right channel of the input data remains in the right channel and thus can only be heard from the loudspeaker for right channel playback. Consequentially directional effects in the playback are lost, i.e. the playback is always the same whether the listener in the virtual environment looks at the sound source from front or from back.

Indeed, audio data of different channels are not mixed but the audio data in the individual channels have been attenuated in two ways:

- (a) **Equal attenuated pass-through** (abbreviated by 'Equal atten. pass-trough' in the table 3.1) attenuates the intensity of all chan-

nels equally corresponding to the distance model. That means while the listener of the scene approaches the sound source, the intensities of all channels are attenuated by an equal factor. The direction from which the scene-listener looks at does not influence the playback, e.g. it sounds the same if the sound source is left to the listener or right

- (b) **Individual attenuated pass-through** (abbreviated by 'Individual atten. pass-through.' in the table 3.1) attenuates the channels separately by incorporating the relative² placement of the sound source to the listener. Figure 3.2 illustrates two scenarios resulting in different playback.

The first approach namely downmixing the audio data is not conform to the X3D standard which postulates to keep the channel separation and does not take any advantage of multichannel audio data. Treating the multichannel sources as mono sources with same positions results in exactly the same playback as a spatialized downmixed multichannel source. Equal and individual attenuated pass-through keep the channel separation but spatialization is not comparable to the way it is done with mono sound sources. This makes it hard for the user of the X3D/VRML application to locate the multichannel sound source.

Further criteria under which the applications have been investigated can be found in the table where the 'Sound delay' states if sound playback has been delayed by the time the sound waves need to reach the listener in the virtual environment. 'Sound delay', 'Multichannel playback' and 'Doppler effect' are criteria to evaluate the 3D audio accuracy. 'Audio stream support' tests the flexibility of the application concerning audio data and 'X3D sound geometry' analyses one important aspect of conformance to the X3D standard, namely if the intensity of sound sources is attenuated as depicted in figure 3.1.

3.1.7.2 Summary

Interpreting the table 3.1 yields several insights what current X3D applications are capable of in the sound rendering context. As already mentioned in the beginning of this section these results have been gained in a practical test and it may be the case that with different scenes and/or settings the application behaves

²relative to the listener

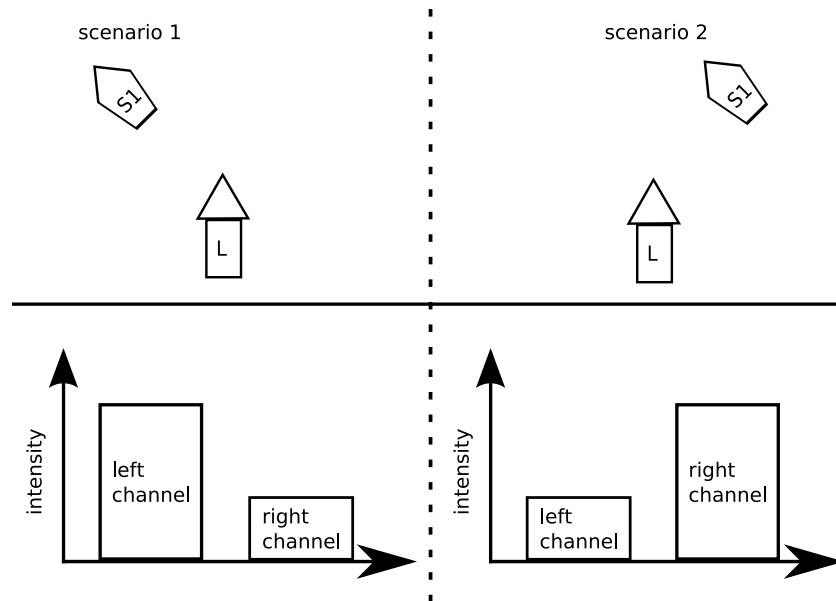


Figure 3.2: The upper figures illustrate virtual environments where in one scenario the sound source is left to the listener and in the other scenario the sound source is right to the listener. Individual attenuated pass-through attenuates the audio data of the stereo input channels separately depending on the relative placement of the sound source to the listener. In **scenario 1** the sound source is left to the listener and thus the left channel audio data are louder as the right channel audio data. In the **scenario 2** it is vice versa. For equal attenuated pass-through the intensities would be the same since the distance of the listener to the sound source is the same in both scenarios

Name	Multi-channel sources	Multi-channel playback	X3D sound geometry (fig. 3.1)	Doppler effect	Sound delay	Audio stream support
Helian [22]	Monohandling	Yes	No	No	No	No
Instant Player [23]	Individual atten. pass-through.	Yes	No	No	No	No ¹
Octaga Player [24]	Equal atten. pass-through	Yes	Yes	No	No	No
Flux Player [25]	Equal atten. pass-through	Yes	No	No	No	Yes
Bitmanagement Contact [26]	Equal atten. pass-through	Yes	No	No	No	Yes
H3DViewer [27]	No sound in test	Yes	Yes	No	No	No
Xj3D Browser [28]	No sound in test	No sound in test	No, similar ²	No sound in test	No sound in test	No sound in test

Table 3.1: X3D/VRML Browser and Player Examination

¹Planned to be supported by non-standard node, see <http://www.instantreality.org/documentation/nodetype/AudioStream/>

²View problems at <http://www.clearwater.com.au/x3d/audio/>

differently but as these scenes conform to the X3D standard and no further sound nodes have been used this section can also be considered as an usability and robustness test of the applications.

Multichannel playback seems to be established and is supposed to be supported by any sound renderer. None of the tested applications handled satisfyingly sound sources with multichannel audio data or implemented simulation of natural sound phenomena like Doppler effect and the above described sound delay. The sound rendering accuracy of these applications is limited. Furthermore, the X3D sound geometry specification is not complied by most X3D browser or player and consequently there are many variations in the behavior of the applications.

3.2 Scene Graph Runtime Libraries

The main requirement on the libraries in this section is that they have to conform to the X3D or VRML standard as these are the only standards for 3D content where sound elements are integrated. The X3D or VRML file formats have to be supported but also the run-time model of these scene graph libraries has to agree with one of these standards. In section 3.1 I give reasons for this. Scene graph libraries as SGL [55], Java3D [56], OpenGL Performer [57], NVIDIA Scene graph [60], OpenSceneGraph [15][14][13] or OpenRM [58][59] have their own representation for virtual environments disagreeing with the standards or not comprising all elements the standards define.

The following scene graph libraries target for conformance to X3D or VRML specification:

OpenVRML provides X3D/VRML parsers, a runtime library and an OpenGL renderer libraries [29].

Avalon constitutes a framework which can be used to realize VR applications [50]

Open ActiveWrl allows to implement parallel working VR applications [62]

H3DAPI is a scene graph API including a API for touch/haptic rendering [27]

X3DLib a X3D library provided with an OpenGL renderer [61]

3.2.1 RTSG

Real-Time Scene Graph (RTSG) is a scene graph framework for supporting VRML/X3D related scene graphs library optimized for real-time ray-tracing with the aim to be conform to the X3D standard [31].

One of the main features of RTSG is it's flexible rendering architecture providing several ways to implement a renderer which also allows rasterization³ or sound rendering as done in this work.

The rendering architecture allows to integrate rendering functionality within the node implementation as it is for example done in H3DAPI [27]. By contrast it is also possible to encapsulate rendering functionality out of the implementation of the nodes into a separate component.

Another advantage in context of rendering is that in RTSG there is no need for full scene graph traversal each frame as done in many other X3D/VRML related software. Instead RTSG provides an event system informing all listeners e.g. a renderer about changes in the scene. This can increase the performance since little or no changes at all in a frame does not imply a full scene graph traversal but only reacting to those events corresponding to the changes.

Furthermore, modifications of scene data which do not depend on the hierarchy in the scene graph, like intensity of a sound source, don't require any traversal.

3.3 Audio/Multimedia software

3.3.1 3D Audio Libraries

This section evaluates libraries considering their 3D audio capabilities. The main requirement on these libraries is the 3D audio playback with surround loudspeaker systems. Furthermore the accuracy of 3D audio generation is examined. In particular Doppler effect has to be simulated.

Besides the goal to find an appropriate 3D audio backend to realize the goals of

³A work with the scope of implementating a renderer using the 3D graphics engine Ogre [49] for RTSG is described in [48]

this work, the techniques of these audio libraries to handle multichannel sound sources is described.

3.3.1.1 FMOD

FMOD 3 [37] is a cross-platform sound engine for playback of audio in virtual 3D world. It supports playback of virtual acoustics via multichannel loudspeaker systems. Spatialization in FMOD includes simulation of Doppler effects, occlusion computations, reverb effects and attenuation by distance. Multichannel audio data are played back in a pass-through manner. It is mainly provided with a commercial license but also a free non-GPL license is available.

3.3.1.2 Miles Sound System

A popular commercial 2D/3D audio library is the **Miles Sound System (MSS)** [36]. The features are playback via multichannel loudspeaker systems, attenuation by distance, environmental reverberation and Doppler effects.

FMOD spatializes multichannel sound sources as default at one position. Optionally one can define an angle which spreads these single channels away from the initially position. The drawback of this method is that one can not define the distance of the speakers to each other and they are always on the floor plane. Furthermore individual placement of the virtual mono channel sources is not possible since all depend on each other.

3.3.1.3 OpenAL Soft

OpenAL Soft [38] is a recent implementation of the OpenAL [39] specification available under the LGPL license.

OpenAL Soft supports 3D audio playback with mono, stereo, 4.0, 5.1, 6.1 and 7.1 loudspeaker systems. Basic 3D audio generation is realized by attenuation by distance accordingly to several models (e.g. linear, inverse square, exponential) and Doppler computations basing on direction, position and especially on the velocities of the listener and sound sources in the virtual environment which are not computed by OpenAL.

Further spatialization techniques like environmental reverberation or occlusion are accessible via the OpenAL effect extension (EFX) [40].

In OpenAL multichannel sound sources are played back in a pass-through manner whereby even the intensity is unaltered, i.e. no attenuation by distance is performed.

3.3.2 SDL

Simple Directmedia Layer (**SDL**) [35] is a cross-platform library for low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video framebuffer. It supports mono/stereo playback with quantization of 8 bit or 16 bits.

This library is not convenient to realize 3D audio playback for virtual environments since it neither performs multichannel sound playback nor provides a plugin mechanism to integrate other backends like an OpenAL library.

3.3.3 Audiere

Audiere [34] is an open source cross-platform high-level audio library. Similar to the SDL it does not provide a plugin mechanism to integrate a 3D audio backend and 3D audio computations would have to be implemented on top of the library.

3.3.4 Multimedia Middleware Solutions

3.3.4.1 Multimedia Processing with Flow Graphs

One concept to realize multimedia middleware processing is encapsulating modules implementing a certain multimedia task into nodes and connecting several of such nodes to a multimedia flow graph. Each node is an independent processing element whereby various connection possibilities are enabled. In such graphs there is a flow direction from *source* nodes to *sink* nodes whereby the first type of nodes represent multimedia elements emitting data like a TV-card or a node reading data from a file on hard disk and nodes of sink type exceptionally con-

sume audio data e.g. a node for audio playback or for writing data to a file. The flow direction defines the direction in which multimedia data pass through the graph. Further node types besides sink and source are:

- *muxer* is a node which processes several input streams and emits one output stream. An example is a node performing audio down-mixing that combines audio data of several channels into one channel
- *demuxer* works vice versa, i.e. it has one input and several outputs e.g. a node separating single channels of a multichannel audio stream like stereo
- *filter* nodes process one input stream and generate one output stream as for example a node performing low-pass filtering would do

Such an modularized and structured concept has several advantages like the flexibility to create several configurations, e.g. exchanging solely a decoder node can enable to support multimedia processing of other format. Flow graphs behavior is conceptional clear since the components are independent and the node principle constitutes a simple way to integrate plug-ins into software making extensions simpler and more flexible.

3.3.4.2 Local Operating Frameworks

The frameworks in this subsection perform the computations and playback of multimedia data on a single computer. Other devices in a network can just act as data source. Microsofts DirectShow [17], Apples QuickTime [18][19] and GStreamer [16] count among this category of multimedia middleware solutions. Since a requirement in this work has been to allow distributed multimedia processing these frameworks can not be employed to achieve the goal 2 as stated in section 1.2.

3.3.4.3 NMM

Network-integrated Multimedia Middleware (NMM) is a multimedia middleware allowing to incorporate several devices in a network for multimedia processing. The transparent access to the devices in a network makes it possible

to use remote processing power to e.g. enable high accuracy sound rendering on low computation power devices. The middleware services of NMM includes synchronized playback distributed over several devices in a network. A advanced feature of NMM is a toolclass `graphbuilder` which automatically constructs flow graphs with appropriate decoders and source nodes. Source and decoder nodes for various formats exist allowing also processing of streamed media data. The flow graph principle in NMM allows easy scaling of multimedia computations.

3.4 Conclusion

The only ISO standards regarding sound elements in a virtual 3D environment are VRML and it's successor X3D. Existing X3D applications had several shortcomings with respect to sound rendering. The sound rendering accuracy is limited since natural sound phenomena are mostly disregarded. Furthermore, multichannel audio data have not been handled convincingly but this is also the case for 3D audio libraries.

Among the 3D audio library could basically all introduced libraries which are OpenAL, FMOD or MSS serve as 3D audio backend since all perform the required 3D audio playback and compute Doppler shifts. OpenAL Soft has been selected because it is free available and one can have insight into the source code. The multimedia middleware NMM fulfills the flexibility requirement to scale 3D sound rendering accuracy, makes distribution possible and OpenAL can be integrated for the basic 3D computations.

The X3D system RTSG represents virtual environments with scene graphs and provides a flexible and performant rendering architecture. Thus the NMM middleware with an OpenAL plugin can be integrated as a sound renderer into RTSG in order to realize 3D audio for virtual environments.

Chapter 4

Architecture

This chapter gives an overview of the components to realize the planned 3D audio playback as depicted in [4.1](#). The roles of `RTSG`, `NMM` and `OpenAL` is explained and how they are brought together.

The simulation of the virtual environment according to the X3D standard is the task of `RTSG`. It handles the X3D scene document, drives the simulation, manages the scene graph, provides a callback mechanism to inform renderers about changes in the scene and a mechanism to visit all X3D Nodes of the scene graph, like traversing. Since `RTSG` has up to now no X3D Sound nodes integrated, the implementation of these nodes is part of this work.

The `SceneSoundSystem` will be a component which constitutes a tool simplifying realization of sound renderers. The tasks are evaluating the scene graph and providing an interface to query all relevant data for sound rendering. Besides supplying basic information of the scene it delivers additional data helping to enhance sound rendering and being conform to the X3D standard. These data are information to meaningfully spatialize multichannel sound sources, velocities of sound sources and the listener required for Doppler computations and intensities of sound sources having applied the X3D attenuation.

A concrete sound renderer making use of the `SceneSoundSystem` is the `NMM-SoundRenderer` which besides the `Ogre renderer` connects the simulation layer with the audio rendering layer. Details of the `Ogre renderer` can be read in [\[48\]](#). Other renderers for `RTSG` are not mentioned in this work since the applica-

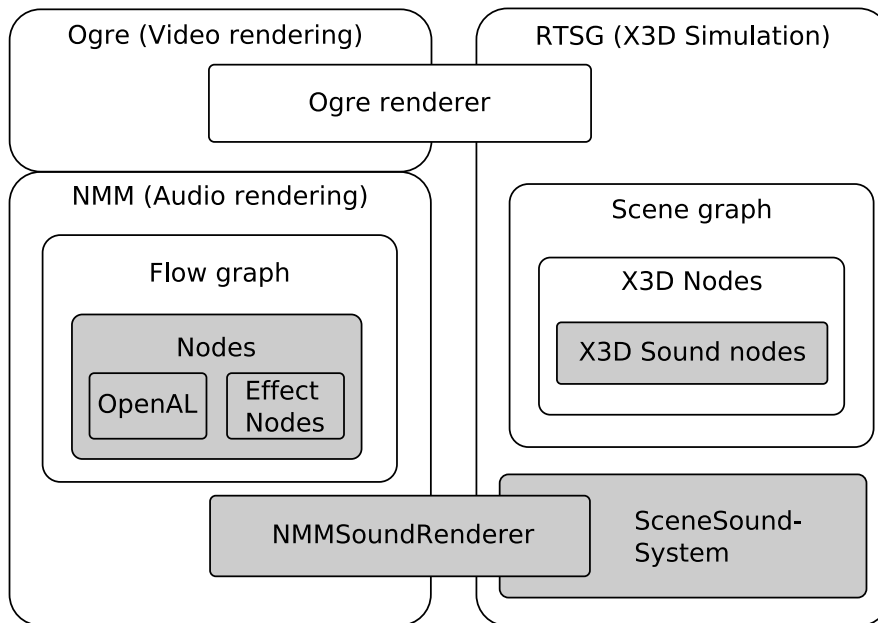


Figure 4.1: Illustration of the components involved in video, 3D sound rendering and X3D simulation. The gray color marks the components which will be implemented or extended within the scope of this work.

tion written within the scope of this work employs the `Ogre renderer` for visual rendering.

The task of the `NMMSoundRenderer` is to play back 3D audio according to the current state of the scene simulation. Therefor the `NMMSoundRenderer` queries scene data from the `SceneSoundSystem` and manages `flow graphs` playing back the spatialized audio data. The flow graphs are constructed during initialization and each time there are new data available from the scene, `NMMSoundRenderer` calls interface functions to access the corresponding nodes of the flow graph to forward these most recent data of the scene. These flow graphs contain already existing NMM nodes like decoders and source nodes but also nodes which have been implemented within the scope of this work. The main node plugin integrates `OpenAL` into NMM which performs 3D audio playback. The 3D audio accuracy has been increased by integrating a sound effect node for delayed sound playback. Since the `NMMSoundRenderer` internally uses flow graphs for 3D audio playback, the sound rendering can be extended by adding further nodes with little effort in future work.

Chapter 5

Integration of 3D Audio into NMM

This chapter is about the integration of the 3D audio library OpenAL into NMM and the implementation of further flow graph nodes to enrich sound rendering. In the first section [5.1](#) an overview of the nodes which are involved into 3D audio playback is given and in subsequent sections I describe their implementation.

5.1 3D Audio Flow Graphs

3D audio flow graphs term flow graphs in NMM containing nodes related to 3D audio playback. These flow graphs can be divided into three blocks: Audio data generation block, 3D audio post-processing block and 3D audio playback block. The audio data generation block contains nodes to obtain raw/decoded audio data for the playback. Several possibilities are imaginable for example reading the audio data from a file encoded in a certain format or receiving a stream of audio data. The 3D audio post-processing block keeps nodes to process these audio data before they are played back in the 3D audio playback block. Nodes in the 3D audio post-processing block can add sound effects to the playback or increase the 3D audio realism. The independency of NMM nodes allows to connect arbitrary many nodes in series within this block whereby scalable rendering accuracy can be achieved.

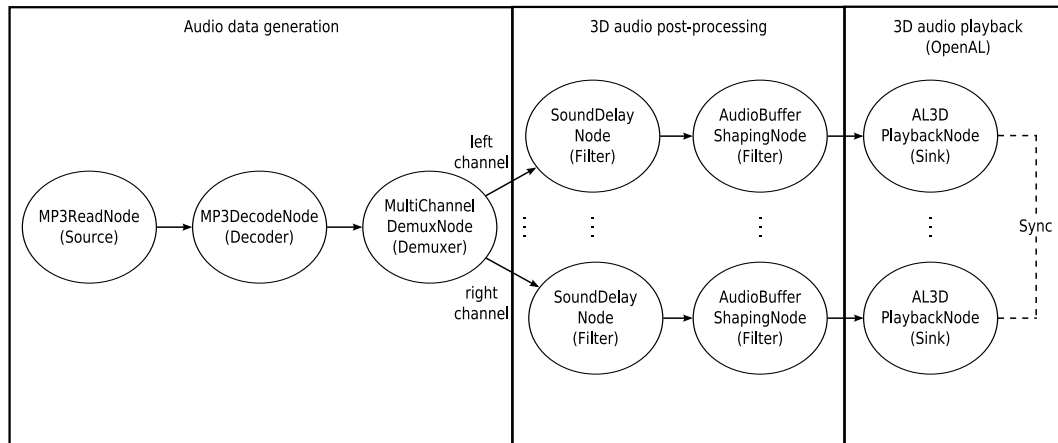


Figure 5.1: Stereo data read and demultiplexed by nodes in the audio data generation block are played back spatialized by OpenAL after being modified by certain post processing nodes. As many sink nodes are created as there are channels in the multichannel audio data. These sink nodes are synchronized in both cases where they are distributed over several devices in a network and in case they are located on a single computing device.

A sample for a 3D audio flow graph is depicted in figure 5.1. This 3D audio flow graph realizes playback of a spatialized stereo sound source. The depicted multimedia flow graph reads in a file in mp3 format and decodes the data into raw audio data. These raw data are afterwards demultiplexed into two streams of raw mono audio data which are post-processed by certain nodes and played back spatialized by OpenAL.

The `MP3ReadNode` being a source node in sense of graph multimedia processing reads the audio data from the mp3 file. These encoded data are then forwarded to the `MP3DecodeNode` applying corresponding algorithms to decode the incoming data and transfer these decoded data to the `MultiChannelDemuxNode`. Within this node the audio data are demultiplexed such that the audio data of the left channel are sent to the one `SoundDelayNode` and the audio data of the right channel are passed to the other `SoundDelayNode`. The `SoundDelayNode` increases 3D audio realism by delaying the playback of incoming audio data until it's internal sound wave propagation simulation determined that the sound waves in the virtual environment have reached the listener in the scene and from that point on incoming audio buffers are accepted and forwarded to the `AudioBufferShapingNodes`. These `AudioBufferShapingNodes` shape the incoming buffers to

a preset size which are received by `AL3DPlaybackNodes` converting these buffers into OpenAL buffers and assigning them to an OpenAL source such that they are played back spatialized.

For each mono sound source in the virtual 3D environment there is one `AL3DPlaybackNode` but a single OpenAL context is doing the 3D audio computations and 3D audio playback for all sound sources. This has been done since OpenAL allows only one context to be rendered by one sound device which implies that all sound sources have to be treated in this single rendering context.

The depicted 3D audio flow graph shows 3D audio playback for one multichannel sound source. In order to enable playback of several (multichannel) sound sources one can build several of these flow graphs or extend the flow graph by adding corresponding nodes. That means the `AL3DPlaybackNodes` can share one OpenAL context among several flow graphs.

5.2 Demultiplexing Multichannel Audio

The `MultiChannelDemuxNode` has one input and several outputs as depicted in figure 5.1. It can be employed to play back multichannel audio data spatialized by `AL3DPlaybackNodes`. In order to be able to play back multichannel audiodata spatialized with `AL3DPlaybackNodes` the `MultiChannelDemuxNode` is actually required since OpenAL plays back multichannel audio data in a pass-through manner. The mono audio data created by the `MultiChannelDemuxNode` can be spatialized by the `AL3DPlaybackNodes` independently.

The task of the `MultiChannelDemuxNode` is to demultiplex multichannel audio data into mono audio data. The current implementation just supports raw audio data assuming the channel data to be stored interleaved. This extraction is done in such a way that in case of stereo for example the audio data for the left channel are sent over one single output and the audio data of the right channel are sent over the other output, i.e. one output stream contains the audio data of one channel. The partial ordering of audio data in a single channel is kept ,too. This is illustrated in figure 5.2.

Since the node knows the quantization which is the size of one sample there is nothing more to compute.

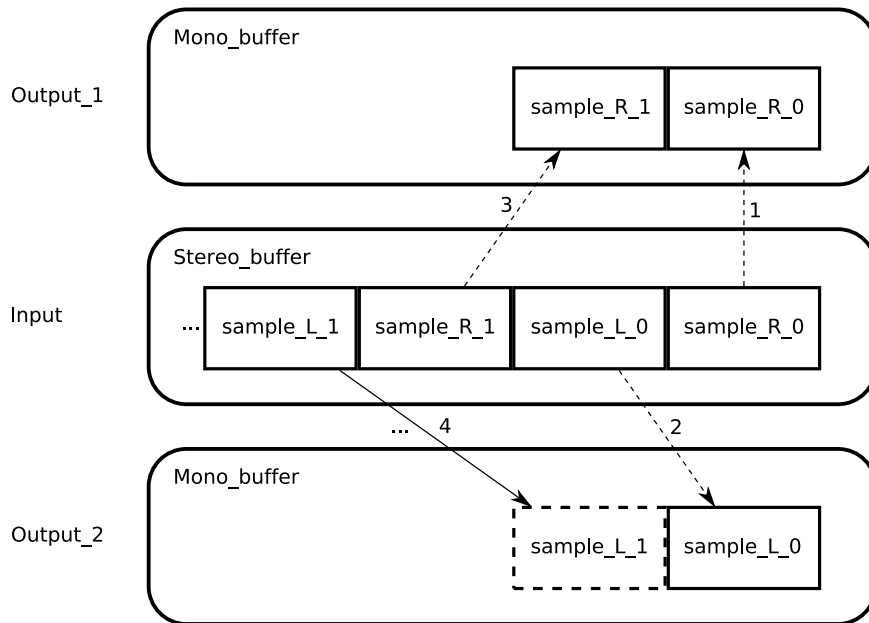


Figure 5.2: Incoming stereo audio data are separated into two mono streams at two outputs. The stereo buffer contains samples for the left channel and the right channel. `sample_R_0` is the first sample of the right audio channel, `sample_L_0` the first audio sample of the left channel, `sample_R_1` the second sample of the right audio channel and so forth. The picture shows that three samples of the `Stereo.buffer` have already been extracted and the 4th sample is copied right now to the `Mono.buffer` of `Output_2`.

5.3 3D Audio Post-processing

The `SoundDelayNode` is a node enriching 3D audio realism by adding a certain sound effect. Further nodes increasing 3D audio accuracy can be implemented similar to this node.

5.3.1 Objectives

Form a technical point of view the `SoundDelayNode` is a filter node having the task to process incoming nodes and generate outgoing nodes.

But it can also be seen in the context of 3D sound rendering since it has a basic 3D environment representation and simulation. The `SoundDelayNode` enriches

3D audio accuracy by simulating how sound waves emitted from a sound source propagate through a medium until they are audible at the listener. Playback at succeeding sink nodes is delayed as long as the sound waves have not reached the listener in the virtual environment. An application scenario is described in section 1.1.

Furthermore, the `SoundDelayNode` simulates the behavior that certain sound waves still reach the listener after the sound source has stopped emitting sound waves.

5.3.2 Implementation

In order to delay playback at succeeding sink nodes the `SoundDelayNode` sends *dummy buffers* containing audio data samples of a 0 amplitude signal resulting in silent playback. Incoming buffers are simply forwarded to the succeeding nodes as soon as sound waves have reached the listener in the virtual environment. Note that the `SoundDelayNode` does not discard any buffers and thus no audio data get lost by inserting this node into a flow graph.

Incoming buffers are forwarded until the sound source in the virtual environment has stopped emitting sound waves and there are no sound waves still on their way to the listener. From that point on dummy buffers are again sent to successive nodes, thus the sound source is not audible again.

5.3.2.1 Simulation

The objects of the 3D environment of this node are:

A listener having a certain position and one sound source parameterized by a position, ellipse parameters defining the sound propagation shape and a direction, besides the speed of sound through the medium surrounding the sound source. The sound propagation has been chosen to be elliptical such that it corresponds to the X3D sound propagation model and it is possible to model a medium where sound does not uniformly propagate in all directions. An example would be a the expansion of sound waves in a storm out of a certain direction: The sound waves do not propagate with the same velocity in orthogonal wind direction as

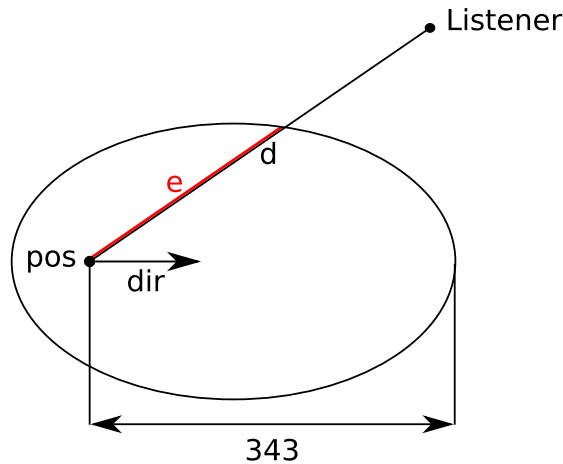


Figure 5.3: This figure depicts the sound wave propagation after one second of sound emitting. The sound waves have expanded in direction `dir` by 343 meters. The ellipse has its left focus at the position of the sound source (`pos`). The expansion in direction to the listener `e` is smaller than the distance `d` between listener and the position of the sound source `pos`. Thus the playback is delayed.

with the wind direction.

Additionally the sound source has a certain start time t_{start} and stop time t_{stop} defining the points in simulation time when the sound source emits sound waves and when it stops emitting sound waves.

The simulation checks in certain update intervals if the playback is supposed to be delayed or not as follows:

1. Determine the current¹ expansion of the sound waves whereby they propagate in direction of the sound source with the full speed through that medium.
2. Test with the ellipse formulas given in the appendix [A](#) whether the listener is inside the current ellipsoid, i.e. test if the expansion of the ellipsoid in direction to the listener is larger than the distance of the listener to the sound source. Figure [5.3](#) illustrates the parameters in this test.
3. If the listener is inside current ellipsoid then start/resume playback else

¹current is relative to the time when the sound source has started emitting sound waves, i.e. the sound waves begin to propagate from that time when the sound source is emitting

delay playback

The same principle can be applied for delayed stopping of playback by simulating the expansion of 'silence' from the sound source.

5.4 Buffer Shaping

The task of the `AudioBufferShapingNode` is to assure that outgoing buffers have a fixed size which can be set via an interface function. This node is required in order to achieve synchronous playback with OpenAL as it can be read in section 5.5.

Additionally, the `AudioBufferShapingNode` sets the *media time* of the forwarded buffers which is the relative time of buffers' audio data in the time interval of the whole audio data. The media time gives succeeding nodes orientation about the progress of the playback.

The media time t_{media_i} of the i -th outgoing buffer is as follows:

$$t_{media_i} = t_{media_{i-1}} + i_{buffer_length} \quad (5.1)$$

where $t_{media_{i-1}}$ is the media time of the buffer sent before and i_{buffer_length} is the length of the outgoing buffers in the corresponding time unit.

Since the size of the buffers is set in bytes one has to compute i_{buffer_length} as follows:

$$i_{buffer_length} = \frac{buffer_{bytes}}{buffer_{quan} * buffer_{freq} * buffer_{chn}} \quad (5.2)$$

where $buffer_{bytes}$ is the predefined size of the outgoing buffers, $buffer_{quan}$ are the number of bytes needed to quantize one sample e.g. 2 bytes for 16 bit quantization, $buffer_{freq}$ is the sample frequency of the audio data and $buffer_{chn}$ corresponds to the number of channels in the audio format e.g. 2 for stereo or 1 for mono.

The shaping of the audio buffers is simply done by waiting for incoming buffers until the `AudioBufferShapingNode` has collected enough bytes to create at least one outgoing buffer. If sufficient data, i.e. at least the predefined number of bytes, are accumulated then repeatedly outgoing buffers, storing as much audio data from the input as they are supposed to contain, are sent individually with the media time computed as stated above until the node has to wait for audio data again.

5.5 Integration of OpenAL

The first section names the tasks of the `AL3DPlaybackNode` mainly being the handling of OpenAL and support for synchronization. The next section described the design of the `AL3DPlaybackNode` and motivates why the Half-Sync/Half-Async pattern has been applied for local synchronization. Distributed playback on several devices in a network can be synchronized by applying the synchronization mechanism of the middleware as described in section [5.5.4](#).

5.5.1 Objectives

The `AL3DPlaybackNode` is a sink node in the sense of multimedia processing with flow graphs. More precise it is a playback node receiving audio buffer and being responsible for the playback of these audio data.

The special property of the `AL3DPlaybackNode` is that it plays back audio data spatialized in a 3D environment by employing the 3D audio library OpenAL. OpenAL does panning and on demand distance attenuation, computes Doppler shifts and performs playback of audio with the help of a sound card and the connected loudspeakers. Within this node the management of the OpenAL objects has to be done, which are:

Device - the sound device used for playback

Context - keeps general settings like speed of sound and a Doppler factor to determine the influence of doppler computations

Source - a spatial representation of a sound source in virtual environment with parameters like position, direction and speed of movement

Listener - the listener in the virtual environment having as primary parameters position, direction and speed of movement

Buffer - a buffer containing audio samples with parameters like number of quantization bits, channels and frequency of the audio data. **Buffers** can be enqueued at a **Source** resulting in spatialized playback of the audio data in the attached buffers

Furthermore synchronous playback of several `AL3DPlaybackNodes` is required to avoid undesired echo effects to obtain smooth playback. Synchronization ensures that the `AL3DPlaybackNodes` which are actually participated in current 3D audio playback have the same progress in playback, i.e. their audio data agree in the media time. On the one side local synchronized playback, i.e. playback on the same machine and on the other side playback distributed over several devices in a network has to be synchronous.

5.5.2 Design Pattern

This section describes and motivates the design of the `AL3DPlaybackNode` where certain subcomponents have been designed accordingly to the Half-Sync/Half-Async software design pattern [54].

5.5.2.1 Motivation

The goal of applying the Half-Sync/Half-Async pattern within this work has been to make local synchronization possible. Local synchronization is indispensable for a smooth playback, especially of 3D audio flow graphs of multichannel sound sources since the multiplexed audio data of the different channels are coordinated to each other.

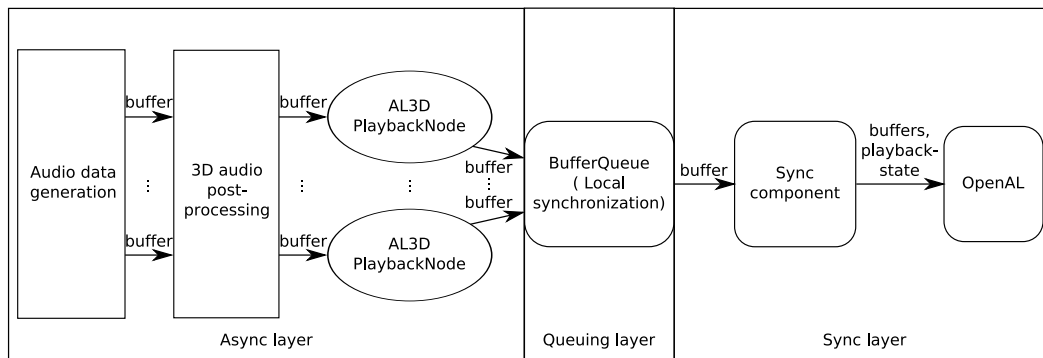


Figure 5.4: Illustration of synchronization on a single computer when communicating with OpenAL. Local synchronization is carried out before transferring buffers received by `AL3DPlaybackNodes` to OpenAL. Additionally, the playback-state (started, stopped and paused) of all local OpenAL Sources is managed by a certain synchronous operating component.

5.5.2.2 Problem

Local smooth playback requires highly precise synchronization with actually no tolerance of time-displacement. In practice little time deviations in the playback have been audible. Time variations caused by costly computations like post-processing or demultiplexing are compensated by the synchronization mechanisms of the middleware. The synchronization of OpenAL playbacks has been done within the `AL3DPlaybackNode`.

OpenAL itself does not provide mechanisms to guarantee synchronous playback of several Sources. It can just ensure synchronous starting, stopping or pausing but playback is asynchronous if one Source runs temporary out of Buffers and other Sources not.

5.5.2.3 Solution

The conceptional solution to ensure synchronous playback at the side of OpenAL is to guarantee that playback of all OpenAL Sources is synchronously started and that OpenAL Buffers are played back synchronously even if OpenAL Sources run temporary out of buffers.

From an abstract point of view there are two layers: A synchronous layer handling

OpenAL **Source** playback and **Buffer** attaching and an asynchronous layer where the **AL3DPlaybackNodes** are located which receive the NMM buffers. A further intermediate layer between the synchronous layer and the asynchronous layer is required to exchange data between these two layers. The presence of these three layers and the requirement of high performance in order to avoid buffer underrun lead to the Half-Sync/Half-Async pattern. Figure 5.4 illustrates the objects in the three layers and the synchronization applied when communicating with OpenAL.

In order to ensure that **Buffers** are synchronously played back at the OpenAL **Sources** dummy buffers are attached to these OpenAL **Sources** which don't have a OpenAL **Buffer** for the current playback time. If a buffer is received for a playback time that is already covered by a dummy buffer then the buffer is discarded.

Figure 5.5 illustrates the scenario where one OpenAL **Source** of a **AL3DPlaybackNode** has received buffers all the time and for the other OpenAL **Source** of another **AL3DPlaybackNode** there had to be attached one dummy buffer. This dummy buffer ensures that the next received buffers can only be played back at the same playback time. Uniform buffer size is required else one can not anticipate all buffers to have the same size and thus the size of the dummy buffer does potentially not equal the size of usual buffers. The **AudioBufferShapingNode** can be employed to ensure that all buffers have the same size.

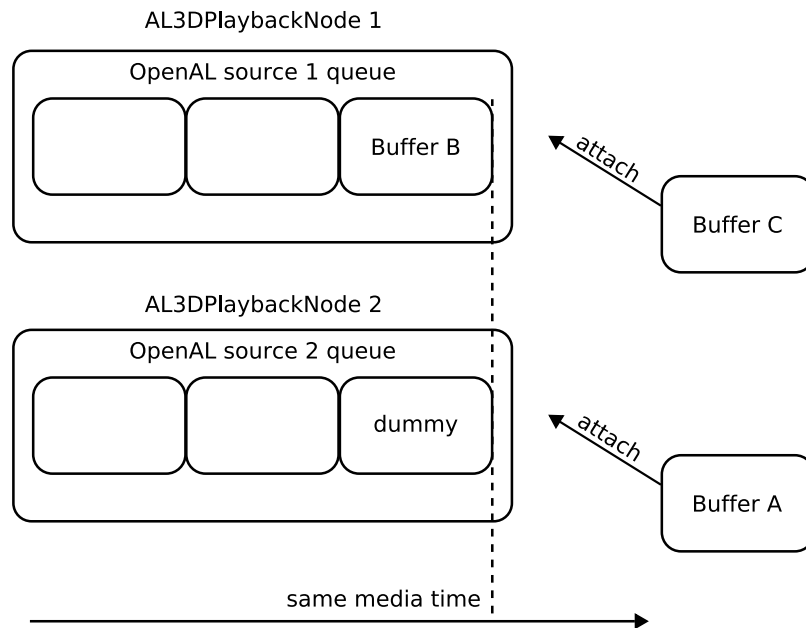


Figure 5.5: Synchronous playback is ensured by inserting dummy buffers if required. Each `AL3DPlaybackNode` has an `OpenAL Source` which has a queue of `OpenAL Buffers` played back subsequently. The dummy buffer ensures that the next arriving buffers `Buffer A` and `Buffer C` are played back at the same time. Without the dummy buffer attached to `AL3DPlaybackNode 2` there would be a chance that `Buffer A` is played back by `AL3DPlaybackNode 2` before `AL3DPlaybackNode 1` has played back `Buffer B` resulting that `Buffer A` and `Buffer C` are played back time-displaced although their contents match in media time

5.5.2.4 Principle

Half-Sync/Half-Async is a concurrency design pattern which decouples asynchronous and synchronous service processing to simplify programming without unduly reducing performance [54]. It has been applied e.g. in the BSD UNIX networking subsystem [53]. The layers of the pattern are as follows:

- **Synchronous task layer** - Services in this layer run in a separate thread and perform high-level processing layers
- **Queuing layer** provides a synchronization and buffering layer. The asynchronous layer enqueue buffers which are independently from the enqueueing dequeued by the component in the synchronous layer.

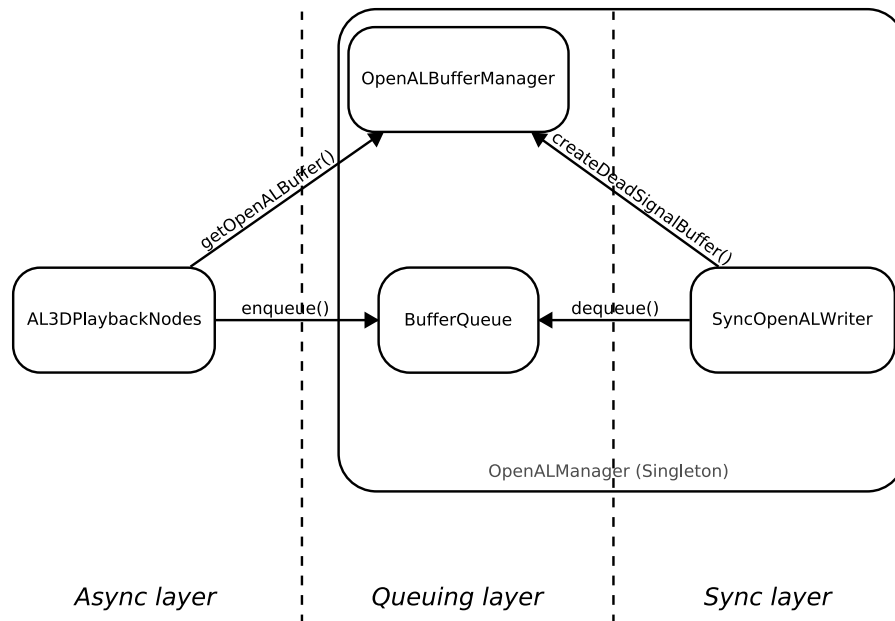


Figure 5.6: The buffer transfer between sync and async layer corresponds to the Half-Sync/Half-Async software design pattern. The primary functions of the pattern are `enqueue()` and `dequeue()` at the interface between these two layer, `BufferQueue`. `enqueue()` is called by all `AL3DPlaybackNodes` and `dequeue()` by the unique `SyncOpenALWriter`. Furthermore administration of OpenAL Buffers is realized by an `OpenALBufferManager`

- **Asynchronous layer** handles extern events like I/O processing
- **External event sources** represent external devices that generate events being processed by the asynchronous layer

5.5.2.5 Application

Figure 5.6 illustrates the design of the `AL3DPlaybackNodes` and depicts the interplay of the subcomponents. The layers of the Half-Sync/Half-Async pattern are also depicted as well as the primary `enqueue()` and `dequeue()` methods realizing the buffer communication between the sync layer and the async layer.

The tasks of the individual components and the communication among them is as follows:

OpenALBufferManager is responsible for the management of OpenAL Buffers.

It keeps account of which OpenAL Buffers have been attached to which OpenAL Sources in order to be able to detach OpenAL Buffers from OpenAL Sources by calling a certain detach function at the SyncOpenALWriter. Furthermore it creates OpenAL dummy buffers/ dead signal buffers. Creation and destruction of OpenAL Buffers is also done within this component. Initially a predefined number of OpenAL Buffers is created which can be increased if required during processing.

BufferQueue constitutes the interface between sync and async layer and stores the OpenAL Buffers enqueued by the AL3DPlaybackNode.

AL3DPlaybackNode is the class implementing the functions of a NMM node.

That means NMM delivers audio buffers to this component, queries the *output delay* describing the delay when audio data are actually played back at this node and awaits initialization and deinitialization of the internal data.

The audio data of received buffers are copied into OpenAL Buffers which are requested from the OpenALBufferManager by calling `getOpenALBuffer()`. This OpenAL Buffer is enqueued into the BufferQueue and the NMM buffer is released.

The initialization and deinitialization of OpenAL is done by calling corresponding functions at the OpenALManager since this singleton allows clear central Device and Context management.

OpenALManager is a singleton managing the OpenAL Device and Context.

Additionally it holds the BufferQueue, the SyncOpenALWriter and the OpenALBufferManager. Synchronization groups being a set of OpenAL Sources which playback has to be locally synchronized by the SyncOpenALWriter are also built at this component. Altogether it keeps all components which are at most once instantiated and performs their initialization and deinitialization.

SyncOpenALWriter makes all OpenAL calls which are related to buffer de/attaching and Source playback state managing. This is done in a separate thread ensuring reliability, high performance and synchronous operation centrally.

The thread is triggered with some tolerance before OpenAL Sources run out of OpenAL Buffers. When being triggered it dequeues OpenAL Buffers from the BufferQueue and attaches all Buffers with the same media time to the corresponding OpenAL Sources. If for a certain OpenAL Source in a synchronization group there are no OpenAL Buffers for the current media time then a dummy buffer is requested from the OpenALBufferManager by calling `createDeadSignalBuffer()` and this dummy buffer takes the place of an usual OpenAL Buffer implying being attached to the corresponding OpenAL Source.

Another task is to compute the output delay on side of OpenAL and of the audio device as I will explain in section 5.5.4.3.

The synchronous operation of OpenAL Source handling and Buffer de/attaching is the key feature of this component. Due to the synchronicity it can be assured that OpenAL Buffers are attached to OpenAL Sources at the same time which is a requirement for echoless playback as depicted in figure 5.5.

The external event sources from the Half-Sync/Half-Async pattern correspond to preceding nodes of the AL3DPlaybackNode. The AL3DPlaybackNode receives buffers as these nodes send audio data.

5.5.3 Interfaces

The AL3DPlaybackNode and the OpenALManager provide certain interface function to set the parameters for 3D sound rendering. The most prominent parameters are the direction, speed and positions of the Listener and Source. Furthermore the intensity and pitch are relevant for a Source. General settings like the speed of sound or the applied distance model can be set. The partitioning is in that way that Listener data and general settings are assigned to the interface of the OpenALManager since then it becomes clear that these settings are global and will effect the rendering of the OpenAL Source of each AL3DPlaybackNode. Attributes which exclusively consider the Source are ascribed to the interface of the AL3DPlaybackNode which avoids the need of an additional argument to identify the OpenAL Source which properties are to be changed.

5.5.4 Synchronization

5.5.4.1 Motivation

Synchronization for playback distributed over several devices in a network avoids undesired echoes in playback, e.g. playback in two rooms where the playback spatially overlaps has to be synchronized. In the context of 3D audio playback this differs from the local synchronization where OpenAL playback is synchronized. Each network-device has it's own OpenAL `Context` and distributed `AL3D-PlaybackNodes` don't know each other.

As an example consider the distributed playback of a file on a local hard disk on one network device and the playback of audio data provided by a stream server in the Internet on the other network device. Synchronization ensures that the playback progress is the same among the playing network devices even if the connection to the stream server is interrupted for a certain time interval. It is legitimate that the playback of one network device is interrupted temporarily but synchronization ensures that all network device which are currently playing have the same progress in playback, i.e. the media time of the audio data is the same.

5.5.4.2 Principle

The synchronization approach of NMM in the example is as follows:

During the time where the connection to the stream server is interrupted the playback of the audio data on the local hard disk continues and the other network device waits for audio data. At the time when the connection to the stream server is again established, the buffers received by the stream server are discarded until the media times correspond to each other in order to catch up with the media time. For example if the connection to the server has been broken for 10 seconds then the at least 10 seconds of audio data are discarded and as many until the time in playback agrees with the playback time of the other network device which has been playing continuously.

5.5.4.3 Implementation

The determination whether buffers are discarded is completely implemented in the middleware. The only remaining computation to be done within synchronized sinks is the output delay. This value is needed in order to start playback synchronized with an active playback. The sink node has to receive audio data at the time when the playback is supposed to start minus the output delay:

$$t_{receive} = t_{start} - i_{delay} \quad (5.3)$$

The output delay for the `AL3DPlaybackNode` is as follows:

$$i_{delay} = i_{device_delay} + i_{source_delay} + i_{BufferQueue} + i_{SyncOpenALWriter} \quad (5.4)$$

where i_{device_delay} is the output delay of the audio device, i.e. the fill status of the sound card buffer, i_{source_delay} is the delay on side of OpenAL caused by attached OpenAL `Buffers` and $i_{BufferQueue}$ and $i_{SyncOpenALWriter}$ are the delays of the buffer keeping components.

This delay describes how long the playback of a new arriving buffer is delayed by audio data already in the playback queue.

OpenAL does not provide a function to query the output delay directly but it can be computed as follows:

$$i_{source_delay} = i_{attached} - i_{played_back} \quad (5.5)$$

where $i_{attached}$ is the length of audio data attached to the OpenAL `Source` and i_{played_back} is the length of already played back audio data. i_{source_delay} is then the length of the audio data which have not been played back.

The delay of the `BufferQueue` and the `SyncOpenALWriter` can be computed as follows:

$$i_{buffer} = \frac{buffer_{bytes}}{freq \cdot 2 \cdot 1} \quad (5.6)$$

where i_{buffer} is the length of the buffered audio data in seconds, $buffer_{bytes}$ corresponds to the number of buffered bytes and $freq$ to the sample frequency of the audio data. The factor 2 in the denominator expresses that OpenAL always uses a 16 bit quantization, i.e. 2 bytes per sample and the factor 1 since only mono audio data are processed.

Chapter 6

Integration of 3D Audio into RTSG

This chapter is about the design and implementation of a sound renderer for the scene graph framework RTSG utilizing the 3D audio flow graphs of NMM to realize playback for a X3D scene.

Sound sources and audio data are represented in the X3D scene graph by certain nodes. Their implementation for RTSG is described in section [6.1](#).

The subsequent section thematizes the implementation of the sound renderer itself beginning with the `SceneSoundSystem` consisting of classes evaluating the scene graph and providing interface functions to query finalized data required for sound rendering. Within the `SceneSoundSystem` strategies are presented which make it possible to meaningful spatialize multichannel sound sources like mono sound sources with any renderer that is implemented for the `SceneSoundSystem`.

The last subsection is about the implementation of the `NMMSoundRenderer` making use of the `SceneSoundSystem` to query scene data and managing NMM flow graphs realizing 3D audio playback.

6.1 Implementation of X3D Sound Nodes

The nodes to represent sound sources and audio data in a X3D scene are as follows:

X3DSoundNode A abstract node constituting the base for sound nodes

Sound is the spatial representation of a sound emitter with the X3D ellipsoid sound geometry model. The main parameter are `direction`, `location`, `intensity` and the ellipses descriptors `minBack`, `minFront`, `maxBack` and `maxFront`. The base node of the Sound node is the `X3DSoundNode`

X3DSoundSourceNode serves as abstract node for all nodes representing audio data

AudioClip Represents audio data which locations are stored in the node field `url`. This field `url` contains several URLs and the first with valid audio data is taken as current URL. It is a concrete `X3DSoundSourceNode`

MovieTexture having `X3DSoundSourceNode` and `X3DTexture2DNode` as base nodes is both a texture node and keeps audio data in certain `urls`.

A **Sound** node which has a `X3DSoundSourceNode` as it's children in the `source` field represents a object in the scene emitting audio data of the `X3DSoundSourceNode` with the spatial assignment defined by it's fields values.

The fact that the sound renderer has not been implemented within the sound nodes reduces the their implementation tremendously.

For the nodes **Sound** and **SoundNode** nothing algorithmic had to be implemented.

The implementation of the `SoundSourceNode` is limited to inform sound renderers whether the node is active or not meaning that reading out the data represented by this node is paused or running. The determination whether a node is active or not depending on the fields `startTime`, `stopTime` and `loop` is specified by the X3D standard as illustrated in picture [6.1](#).

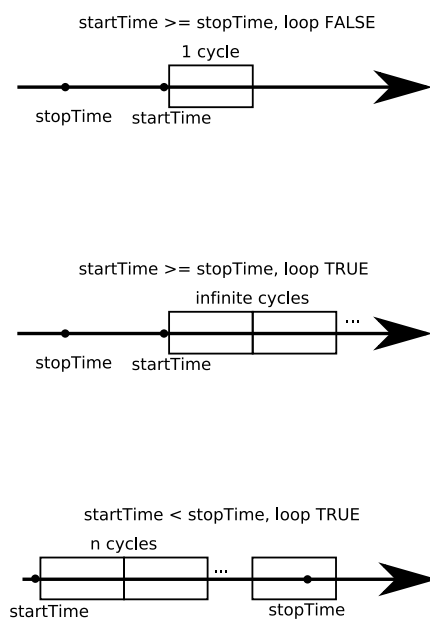


Figure 6.1: Illustrates time-dependent node execution.

The first figure shows the parameters resulting in the playback of one cycle where a cycle defines a complete read out of the audio data. The second figure shows an infinite long playback of the audio data by looping the read out. Finally the last figure shows the playback of a fixed number of cycles where the last cycle is not a fully read out but is interrupted when the `stopTime` is reached.

6.2 Sound Renderer for RTSG

6.2.1 SceneSoundSystem Architecture

6.2.1.1 Motivation

The primary goal of the `SceneSoundSystem` is to process all data from scene graph which are required for basic sound rendering and providing interface methods to query these data allowing to implement a sound renderer without needing to care about tasks like traversing over scene graph and processing updates. It is a set of tools classes which can be used by a sound renderer but they don't have to be applied. RTSG is independent of the `SceneSoundSystem` and a sound renderer can still be implemented as a usual renderer but then it has to evaluate the scene graph itself.

Since all presented 3D audio libraries did not spatialize multichannel audio data satisfyingly the `SceneSoundSystem` provides strategies which makes it possible to spatialize multichannel with any of these 3D audio libraries.

A further feature is the the determination of the attenuated intensity accordingly to the X3D sound model for all sound sources. Thereby the sound renderer can deactivate internal distance and directional attenuation computations and take the intensity agreeing with the X3D model. This is performed on demand allowing to still use another sound model and avoid unnecessarily computations. This simplifies to develop further sound renderers for RTSG which are conform to the X3D standard.

The last bonus computations performed by the `SceneSoundSystem` for a sound renderer is the automatic determination of speed of the listener in the scene and sound source. These values can for example be taken into account for Doppler calculations which are supported by all 3D audio libraries which have been presented in this work.

6.2.1.2 Design

Figure 6.2 illustrates the design of the `SceneSoundSystem`. On the left there is the application using a `SceneSoundSystemRenderer` depicted which can make certain

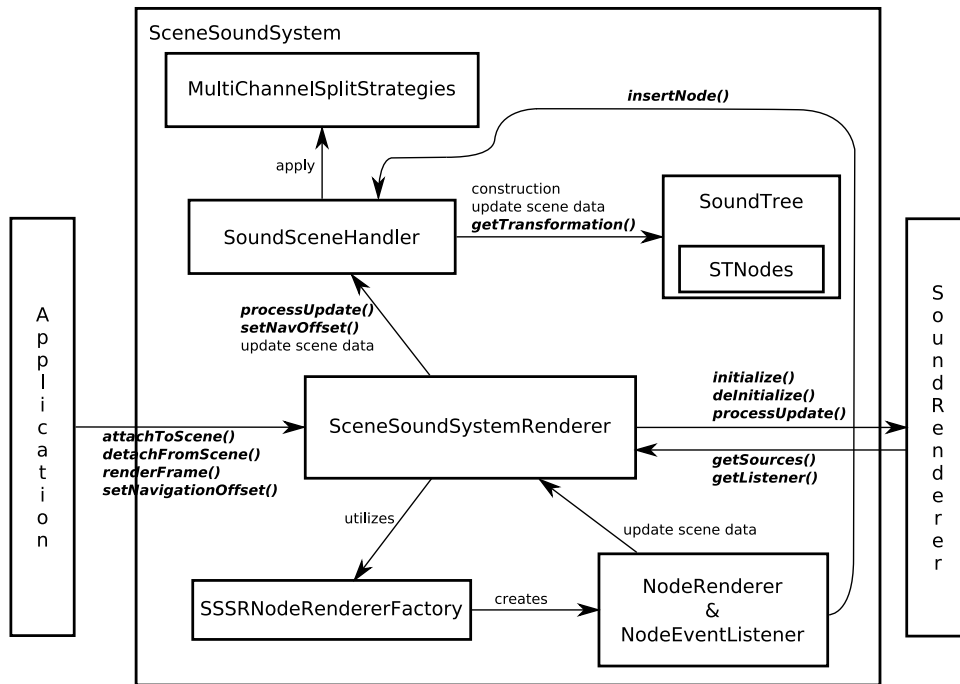


Figure 6.2: Design of SoundSceneSystem. Bold italic descriptions mean function calls and normal typed text describes the interaction between components

function calls like `detachFromScene()` and `attachToScene()` which attaches or detaches the `SceneSoundSystemRenderer` from the X3D scene. Further functions are `renderFrame()` informing the `SceneSoundSystemRenderer` that the current scene state has to be rendered and `setNavigationOffset()` telling the `SceneSoundSystemRenderer` how the user navigated in the scene.

The `SceneSoundSystemRenderer` constitutes an abstract renderer in the sense of RTSG being encapsulated of node implementations. It utilizes a factory (`SSSRNodeRendererFactory`) to create `NodeRenderers`. These `NodeRenderers` create for each node and each reference to a node they are supposed to render a representative (`STNode`) in the `SoundTree` by calling `insertNode` and register themselves as a `NodeEventListener`. By this registration these as callback handlers they are informed about changes of their node and can call corresponding update functions at the `SceneSoundSystemRenderer`.

The `SceneSoundSystemRenderer` forwards these updates to the `SoundSceneHandler`. This `SoundSceneHandler` is responsible to build, update and evaluate the `SoundTree` constituting a reduced representation of the scene graph contain-

ing for these nodes a representative which are relevant for sound rendering. The motivation for applying this representation of the scene even though RTSG keeps the complete scene graph is on the one hand to avoid visiting nodes or even sub-graphs which are not relevant for sound rendering like light, material or animation nodes. On the other hand `SoundTree` is a set of undirected trees enabling bottom up traversals allowing to traverse beginning at nodes, representing a sound source, up to their root node to accumulate the transformations. This increases the performance especially for complex scenes in particular because typically the number of subgraphs containing sound nodes is relative small compared to the whole scene graph.

By avoiding references of nodes in the `SoundTree` a node can have at most one parent and thus the structure is a tree. This allows to store for example at the leaf nodes one transformation which accumulates all the transformations from the node to it's root. Thereby performance can be increased since it is not required to perform traversal from this node to it's root if the transformations have not changed. This optimization can be implemented in future work.

Additionally, the `SoundSceneHandler` applies `MultiChannelSplitStrategies` to obtain data to spatialize multichannel sound sources.

Finally, on the right side of the graphic the `SoundRenderer` is depicted being a concrete sound renderer derived from the `SceneSoundSystemRenderer`, like the `NMMSoundRenderer`. The functions this `SoundRenderer` implements are called by the `SceneSoundSystemRenderer` for de- and initialization as well as notifying that the scene state has to be rendered.

6.2.1.3 Operation Principle

6.2.1.3.1 Initialization

The application creates a scene from a X3D scene document and attaches this scene to the renderers like the `SceneSoundSystemRenderer` by calling `attachToScene()`. Within this function the `SoundTree` is constructed by calling the RTSG traversal routine and the first `SoundSceneHandler::processUpdate()` is called to evaluate the initial scene state.

The traversal routine of RTSG creates the `NodeRenderers` for the individual

nodes by utilizing the `SSSRNodeRendererFactory`. The `NodeRenderers` of the `SceneSoundSystem` create during this traversal a representative in the `SoundTree` for each node and each node reference. That means nodes are not referenced in the `SoundTree` and one obtains a tree.

Additionally they register themselves as `NodeEventListeners` in order that they are informed about changes in the scene.

Finally `initialize()` of the `SoundRenderer` is called preparing the audio backend for rendering.

6.2.1.3.2 Deinitialization

The deinitialization sequence is similar to the initialization starting with the Application calling `detachFromScene()` causing the `SceneSoundSystemRenderer` to firstly deinitialize the `SoundRenderer`, afterwards arranging that the `SoundSceneHandler` clears internal data structures and finally activating the RTSG rendering routine with the task to remove data which have been set by the `NodeRenderers`.

6.2.1.3.3 Update Handling

The application arranges that RTSG does one simulation step causing that `NodeEventListeners` receive update events expressing changes in the scene. These events are forwarded to the `SceneSoundSystemRenderer`. From there the data are given to the `SoundSceneHandler` which updates the corresponding `STNodes`. Up to now there has not been any notification to the `SoundRenderer` and the `SceneSoundSystem` will still deliver the scene data of the last simulation step when calling `getSources()`. The method `getSources()` returns the data of all sound sources in the X3D scene.

This notification has to be triggered by the Application by calling `SceneSoundSystem::renderFrame()` which informs that all events of the simulation step have been dispatched and rendering has to be done with these data.

`SceneSoundSystem::renderFrame()` works as follows:

The `SceneSoundSystemRenderer` calls `processUpdate()` at the `SoundSceneHandler` to achieve that the new data of the `STNodes` are evaluated by performing

traversals in the `SoundTree`. Within this function the feature computations are performed as well.

Subsequently the `processUpdate()` method is called to notify the `SoundRenderer` about the changes. These new scene data can be queried by calling `getSources()` and `getListener()` where `getListener()` returns the data of the listener in the X3D scene.

6.2.1.4 Features Computations

6.2.1.4.1 MultiChannelSplitStrategies

6.2.1.4.1.1 Application

The goal of the `MultiChannelSplitStrategies` is to provide data for the scene allowing to spatialize multichannel sound sources with a sound renderer especially if the sound renderer is limited to spatialize mono sound sources meaningfully. The basic idea is to create a set of mono sound sources for each multichannel sound source. These so called *mono splits* are supposed to emit the audio data of individual channels of the multichannel audio data. This is the suggestion how to make use of the mono splits but the way how the mono splits are used is up to the `SoundRenderer` of the `SceneSoundSystem`.

The task the `MultiChannelSplitStrategies` take over is the localization of these mono splits. Therefore it accesses additional data assigned to the `Sound` nodes of the scene, e.g. geometries in the scene. The number of mono splits created depends on the employed strategy and the additional data at the `Sound` node. The `SoundRenderer` has to come up with techniques how to treat the situation where there are less mono splits than channels in the audio data. Solutions are for example to assign multiple channels to individual mono splits or to leave the channels out for which no mono splits exist. In the case there are more mono splits than channels one could reuse the channels in a round robin manner such that each mono split has audio data.

These mono splits neither exist in the scene graph of RTSG nor in the `SoundTree` and the `SoundRenderer` can still use the unmodified multichannel sound sources. This avoids that sound rendering alters the scene for other (sound-) renderers.

6.2.1.4.1.2 Conceptual Strategies

The first two strategies handle geometric data to determine the position of mono splits. It is presumed that the geometries have been assigned to the multichannel sound source.

The last method presented simply takes a set of fixed positions assigned to the multichannel sound source as locations of the mono splits. Advantages and drawbacks of all methods are worked out and it depends on the requirements which method is the best to be applied.

1. Multiple Geometries

This method takes a set of geometries in the scene and understands the global position of these geometries as global positions of mono splits. The number of geometries determines the number of mono splits created for the multichannel sound source.

This has the advantage that the exact positions of the mono splits in the final scene is imaginable during scene modeling. It is ideal if sound emitters of the scene are punctual, e.g. a loudspeaker in virtual world.

The drawbacks of this method is the requirement of multiple geometries and that the scene-object, which is supposed to contain the mono splits, must be divided into several geometries namely at least for each mono split one geometry in order to obtain reasonable placement.

2. Bounding Volumes

If the requirement having several geometries is not fulfilled then the method in this section can help out to locate mono splits. Here just a single geometry is processed to determine the mono split placements.

The key idea is to locate the mono splits inside a bounding volume of the geometry. The motivation for this approach is to have a fully automatic placement simplifying scene authoring. The bounding volume has not necessarily to be computed by the `SceneSoundSystem` but either the scene document has a bounding volume defined which allows more control of the mono split placement by the author or the visual rendering system has already computed the bounding box for the corresponding geometry.

For the placement of the mono splits inside the bounding volume several

ways are imaginable. One approach is to move a loudspeaker constellation inside this bounding volume, e.g. the 5.1 surround loudspeaker system constellation depicted in figure 2.2. For 2D or 1D constellations further disambiguation has to be done accordingly to certain conventions.

Considering the placement for a 5.1 surround loudspeaker system a reasonable simple approach is to apply the transformations which have been applied to the geometry in the scene also to the constellation plane. The initially placement of the constellation plane could be in the origin parallel to the ground of the scene.

Another approach is to place the mono splits at the corners of a bounding box with a predefined ordering.

The drawback of this method is that the absolute positions of the mono splits are not necessarily imaginable before rendering the scene especially if the bounding volume is determined at run-time for complex geometries.

3. Predetermined Position

The last way to specify the position or relative placement of the mono splits is to define them in the scene document itself.

Here translations/vectors are assigned to the multichannel sound source and define the position of a mono split as one translated position relative to the multichannel sound source.

This methods has the advantage that the absolute position is well predictable but there is a shortcoming on automatism and a relation of sound sources to geometry objects is missing.

6.2.1.4.1.3 Design and Implementation

The design of the `MultiChannelSplitStrategies`, depicted in figure 6.3, allows to integrate various strategies.

The abstract class `MultiChannelSplitStrategy` is the base class for all strategies. Classes deriving from this class have to implement a `split()` method which is supposed to return a set of positions for the mono splits accordingly to the implemented algorithm.

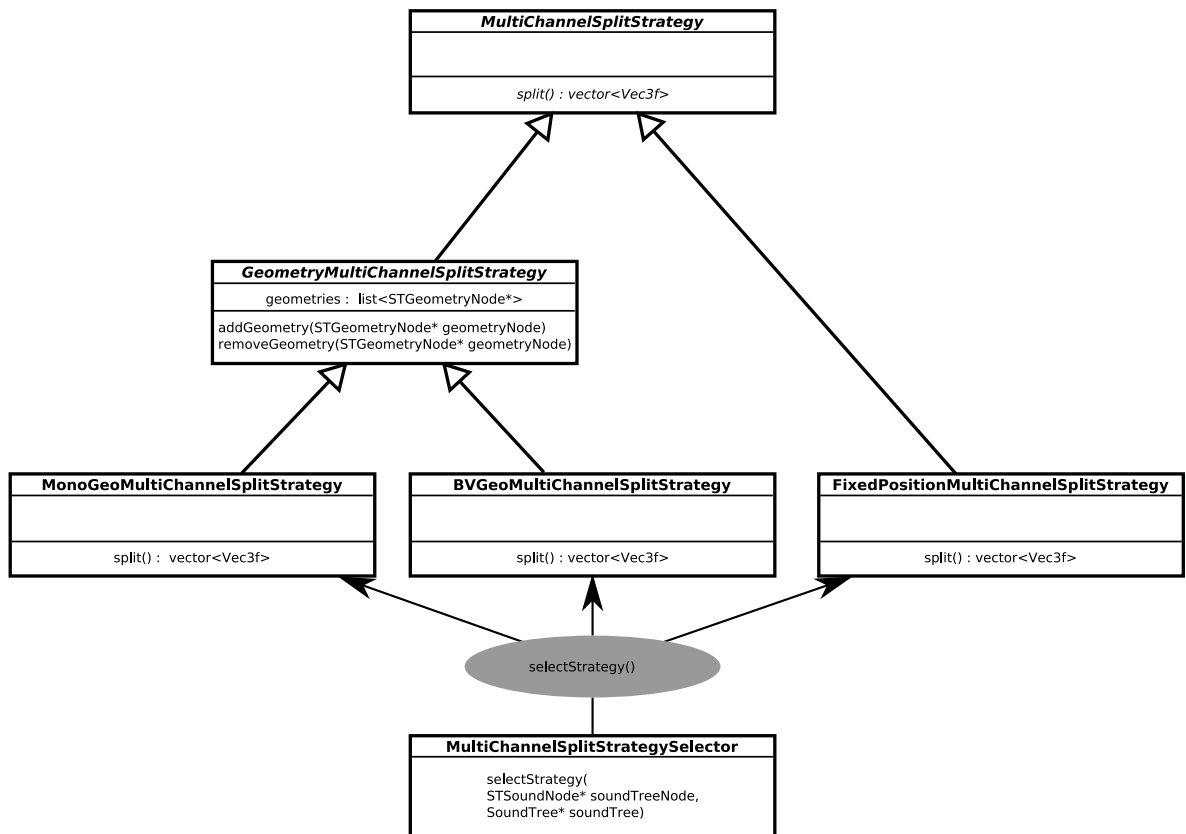


Figure 6.3: Design of MultiChannelSplitStrategies

One concrete strategy could e.g. be the `FixedPositionMultiChannelSplitStrategy` which interprets certain vectors assigned to a `Source` as predetermined positions.

Strategies considering geometric objects associated to a `Sound` node for the determination of the mono split positions have to derive from the `GeometryMultiChannelSplitStrategy` class. This class contains a vector of geometric nodes in the `SoundTree` which can be added or removed by corresponding functions.

An example for such a `MultiChannelSplitStrategy` is the `BVGeoMultiChannelSplitStrategy` which computes the bounding box of one geometric object assigned to a `Sound` node and applies a certain algorithm to localize mono splits.

The `MultiChannelSplitStrategySelector` is a class to create depending on the data assigned to a `Sound` node an appropriate strategy to localize mono splits. Selection can be done as follows:

If any geometric objects are assigned to a `Sound` node then these geometries are added to the `GeometryMultiChannelSplitStrategy` by calling `addGeometry()`. In the case only one geometry is given a `BVGeoMultiChannelSplitStrategy` is created else `MonoGeoMultiChannelSplitStrategy`. If vectors are assigned to the `Sound` node then a `FixedPositionMultiChannelSplitStrategy` is selected.

The concrete `MultiChannelSplitStrategy` which has been implemented within the scope of this work is the `MonoGeoMultiChannelSplitStrategy`. It implements the algorithm understanding the global position of each geometric objects as the global position of a mono split.

That means if the additional data of a `Sound` node contain a set of references to geometric nodes then the positions of these geometric objects in the scene are determined and returned as position of the mono splits. A minimalistic scene where two `Boxes` have been associated to a `Sound` node can be found in the appendix at [B.2](#). The two resulting mono splits have the exact position of the two boxes after applying this strategy.

The implementation of `MonoGeoMultiChannelSplitStrategy::split()` simply queries the transformed position of the geometric objects at the `SoundTree`. The `SoundTree` has done traversals from the geometric node to the root in order to obtain the global position.

6.2.1.4.2 Speed determination

The `SceneSoundSystem` determines for the listener in the scene as well as for all sound sources and their mono splits the velocity vectors which can be used by a 3D audio backend to e.g. simulate Doppler effects.

The speed at simulation time i of these moving objects is simply determined as:

$$v_i = \frac{pos_i - pos_{i-1}}{sim_time_i - sim_time_{i-1}} \quad (6.1)$$

where pos_i is the position vector (3D) of the moving object at simulation time i , pos_{i-1} is the position vector of the moving object at simulation time $i - 1$, $sim_time_i - sim_time_{i-1}$ is the time difference between the simulation steps.

In order to avoid abrupt movements one can average over the current velocity vector and speed vectors from previous simulation steps. The weighting for the average is as follows:

$$w_0 = \frac{n \cdot (n + 1)}{2} \cdot \frac{1}{n^2} \quad (6.2)$$

$$w_i = \frac{n - i}{n^2} \quad (6.3)$$

where w_0 weight the current speed and w_i weights the speed i simulation steps ago. n is the number of speed vector over which averaging is computed.

The weights are monotonically decreasing which has the effect that the more recent a speed is the greater is the influence in the average. The weights add up to 1 such that the velocity is not scaled in the end:

$$\begin{aligned}
w_0 + \sum_{i=1}^{n-1} w_i &= \frac{n \cdot (n+1)}{2} \cdot \frac{1}{n^2} + \sum_{i=0}^{n-1} \frac{n-i}{n^2} \\
&= \frac{\frac{n \cdot (n+1)}{2} + \sum_{i=0}^{n-1} n-i}{n^2} \\
&= \frac{\frac{(n-1) \cdot n}{2} + \frac{n \cdot (n+1)}{2}}{n^2} \\
&= \frac{\frac{2 \cdot n^2}{2}}{n^2} \\
&= 1
\end{aligned} \tag{6.4}$$

6.2.1.4.3 X3D intensity computation

Figure 6.4 illustrates the X3D sound geometry model. The `SceneSoundSystem` computes the attenuated intensity of all sound sources according to this model on demand. This is done as follows:

Let d be the distance of the listener to the sound source. There are three cases to distinguish:

- $d < rMin$ → the listener is inside the inner ellipse and thus the attenuation factor is 1, i.e. no attenuation
- $d > rMax$ → attenuation factor is 0, i.e. no audible playback since the listener is outside the outer ellipse
- $rMin < d < rMax$, which means the listener is between the inner and the outer ellipse and interpolation has to be applied. The interpolation corresponding to the X3D system is as follows:

$$fac = \frac{d - rMin}{rMax - rMin} \tag{6.5}$$

Multiplying this value with -20 db yields a linear ramp between 0 db and -20 db as postulated by the X3D specification for sound sources which inner ellipse has at least a size of 1/10 of the outer ellipse.

If this proportion is not fulfilled then a inverse square intensity drop-off is to be modeled. Mathematically this can be described as

$$20 \cdot ((fac - 1)^2 - 1) \tag{6.6}$$

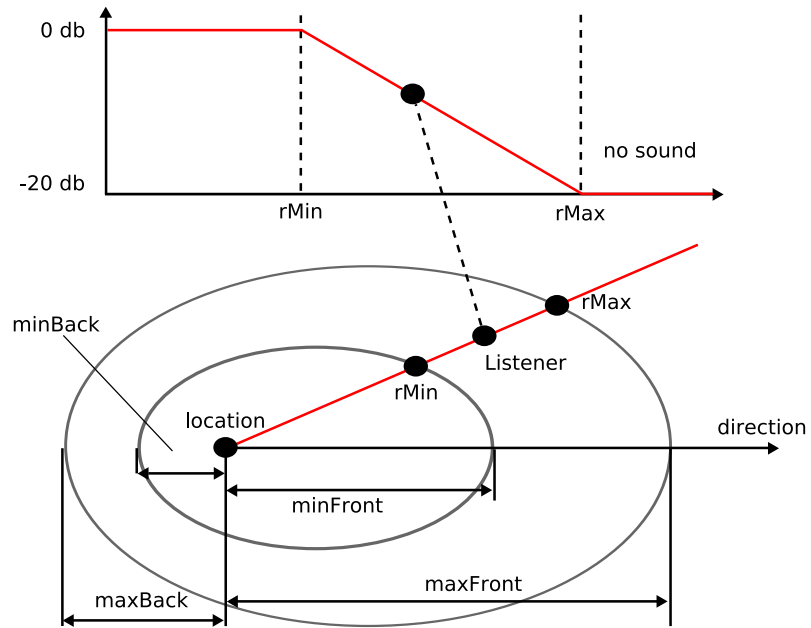


Figure 6.4: Illustration of the X3D sound model. The lower figure shows a sound source in virtual environment having a `location`, `direction` and parameters for describing two ellipses. The inner ellipse, described by `minBack` and `minFront`, defines the region where the sound source can be heard at it's maximum intensity. Outside the outer ellipse, extended by `-maxBack` and `maxFront` in direction `direction`, the sound source is not audible at all. Between inner and outer ellipse linear interpolation of the intensity ranging from 0 dB to -20 dB is applied as depicted in upper figure. `rMax` is the extension of the outer ellipse in direction to the listener and `rMin` the extension of the inner ellipse in that direction.

which is a quadratic function shifted by 1 to the right moved down by 1 and afterwards scaled by 20. In the range from 0 to 1 (which are the possible values for *fac*) the function has a quadratic fall off from 0 to -20. Understanding this value in db scale agrees to the X3D specification.

The values for `rMin` and `rMax` can be determined by applying the ellipse equation in polar form as described in [A](#) for the inner ellipse and the outer ellipse.

6.2.2 NMMSoundRenderer

The objectives of the `NMMSoundRenderer` are to setup 3D audio flow graphs in NMM and keep them up to date with the data from the X3D scene.

The `NMMSoundRenderer` plays the role of a concrete `SoundRenderer` for RTSG within the `SceneSoundSystem` depicted in figure [6.2](#). Every `SoundRenderer` making use of the `SceneSoundSystem` is supposed to be derived from the `SceneSoundSystemRenderer` class and has to implement the abstract methods `initialize()`, `deInitialize()` and the method within rendering is supposed to actually happen namely `processUpdate()`.

6.2.2.1 Initialization

The initialization is mainly the construction of 3D audio flow graphs having a form as depicted in figure [5.1](#). Therefore it firstly queries the initial scene data from the `SceneSoundSystemRenderer` by calling `getSources()`

For each (multichannel) sound source a flow graph is constructed in the following way:

The tool object `graphbuilder` of NMM is configured in such a way that it automatically selects the appropriate decoders for the URL of the audio data and add a `MultiChannelDemuxNode` as sink node. In case the audio data are mono this `MultiChannelDemuxNode` does not execute any relevant computations. But this case can be handled in the same way where multichannel audio data are played back:

The flow graph is extended by adding a `SoundDelayNode`, an `AudioBuffer-`

`ShapingNode` and finally an `AL3DPlaybackNode` in this order for each output at the `MultiChannelDemuxNode` . These nodes are consecutively connected to obtain the final 3D audio flow graph.

The playback for a X3D node and references to this node is synchronized as well as the playback of mono splits for a multichannel sound source.

One `AL3DPlaybackNode` represents one mono split or one mono sound source.

After creation the flow graphs are paused to ensure that playback is only done if the corresponding sound nodes are active. Note that during the construction of the flow graphs OpenAL has been initialized.

Finally the `processUpdate()` function is called arranging that the 3D environment of OpenAL and the `SoundDelayNode` are adapted to the X3D scene state as described in section [6.2.2.3](#).

6.2.2.2 Deinitialization

The deinitialization code destructs all flow graphs implying that OpenAL is deinitialized.

6.2.2.3 Processing Scene Updates

Within the `processUpdate()` basically the parameters for 3D sound rendering are set at the `AL3DPlaybackNode` and `SoundDelayNode` by utilizing their interfaces. In order to obtain the current scene data `getSources()` and `getListener()` are called at the `SceneSoundSystemRenderer`. Here the flow graphs are un-/paused depending on the state of the sound nodes in the scene, too. Within this function the positions of the mono splits are set at the `AL3DPlaybackNodes` if present. If no mono splits have been computed then the `AL3DPlaybackNodes` get the position of the (multichannel) source. In either way for each mono channel which is demuxed by the `MultiChannelDemuxNode` a separate `AL3DPlaybackNode` is created and its scene data are set. This prevents in all cases that OpenAL plays back in pass-through manner.

In order to obtain the X3D sound model the OpenAL distance attenuation is deactivated and at each `AL3DPlaybackNode` the attenuated intensity computed

by the `SceneSoundSystem` is set as intensity of the sound source. Furthermore the speed of the listener and the sound sources determined by the `SceneSoundSystem` are used to enable Doppler effects in OpenAL.

Chapter 7

Conclusion

Within the scope of this work current available applications displaying virtual environments have been evaluated in order to obtain an overview of their sound rendering capabilities. Some shortcomings have been revealed like the disregard of natural sound phenomena. Furthermore, in this examination it became clear that the current approaches to handle multichannel audio data in virtual 3D environments are not convincing. Playback hardware like stereo or 5.1 surround loudspeaker systems have been introduced as well as 3D audio playback software. Thereby goal 1 of this work has been achieved.

Goal 2 has been realized by employing the multimedia middleware NMM and integrating the 3D audio library OpenAL into the middleware. Audio accuracy can be increased by incorporating further 3D audio nodes like the `SoundDelayNode` into playback. The transparent access to devices in a network allows distributed 3D audio computations. Since NMM is independent of OpenAL alternative 3D audio software can be integrated for realizing even more accurate 3D audio generation and also playback by e.g. employing a wave field synthesis audio system as written in section [2.2.2](#).

Goal 3 is achieved by integrating NMM into the X3D framework RTSG. This framework simulates the virtual 3D environment described by a X3D scene document. The integration is done by designing and developing a sound renderer making use of a optional architecture evaluating scene data.

Goal 4 is accomplished by an architecture to generate mono sound sources for

multichannel sound sources. Each mono sound sources is associated with an individual channel of the multichannel audio data. Thereby the audio data of the multichannel sound source can be played back spatialized in the virtual 3D environment even if the 3D audio library itself can not spatialize multichannel sound sources. The positions of the mono sound sources are determined by evaluating geometric data in the scene. Further approaches to localize the mono sound sources have been presented.

Chapter 8

Future Work

8.1 Distributed Sound Processing

The middleware that is employed by the implemented sound renderer allows to outsource 3D audio computations to several devices in a network. This capability could be exploited by the sound renderer for realizing audio playback for X3D scenes. As further extension several audio hardware connected with computers in a network can be used to perform 3D audio playback for one X3D scene. For example two 5.1 surround loudspeaker systems would yield 10.2 surround sound for a X3D scene.

8.2 Enhanced Audio Realism

Basic spatialization is supported by the sound renderer implemented in this work. Advanced spatialization techniques also incorporate the environment of the sound sources. Thereby phenomena like occlusion or reflection of sound waves can be simulated. Furthermore constructive or destructive interference phenomena can increase audio realism.

8.3 Extensions to X3D Sound Component

Within the scope of this work the nodes specified by the X3D standard have been implemented. Solely using these nodes can be a limiting factor since for example material nodes with acoustic parameters are missing. Such nodes can be increases into the X3D framework RTSG to allow simulating scenes with appropriate acoustic properties.

In order to dynamically modify the audio samples in the scene nodes with access to the audio data could be integrated. This allows for example to dynamically compose a piece of music in a scene by mixing available audio samples.

Another extension to X3D would be an alternative representation of sound data. For scenes of physical experiments a representation of sound by means of the sound waves itself could be interesting. The node can contain a set of sound waves described by their frequency, phase and amplitude. Such a representation of sound waves is advantageous if intending to simulate inference phenomena.

8.4 Alternative Audio Playback

The sound renderer for the scene graph framework RTSG uses 3D audio flow graphs with OpenAL as 3D audio backend. The middleware NMM allows to integrate further 3D audio backends. Thereby alternative 3D audio approaches like finite and boundary element method can be integrated. By designing and developing a further abstraction layer the sound renderer can make use of arbitrary 3D audio nodes determined during run-time.

Appendix A

Ellipse evaluation

This section describes how to determine the extension of an ellipsoid in direction of a certain vector v . Figure A.1 describes the parameters of the ellipse and the variables in the computation. The ellipsoid (3D) is obtained by rotating the ellipse (2D) around the vector *direction*. The formulas and definitions which have been used in this section can be found at [32].

The given values are: *front*, *back*, *direction*, *location* and v

The goal of this computation is to determine the value of r with the given values.

The equation for r of an ellipse in polar form relative to the left focus is:

$$r = \frac{p}{(1 - \varepsilon \cdot \cos(h))} \quad (\text{A.1})$$

where ε is the so called *eccentricity* and p is the *half parameter*.

The eccentricity can be computed as

$$\varepsilon = \frac{e}{m} \quad (\text{A.2})$$

with e being the *linear eccentricity* defined as [33]

$$e = \sqrt{m^2 - n^2} \quad (\text{A.3})$$

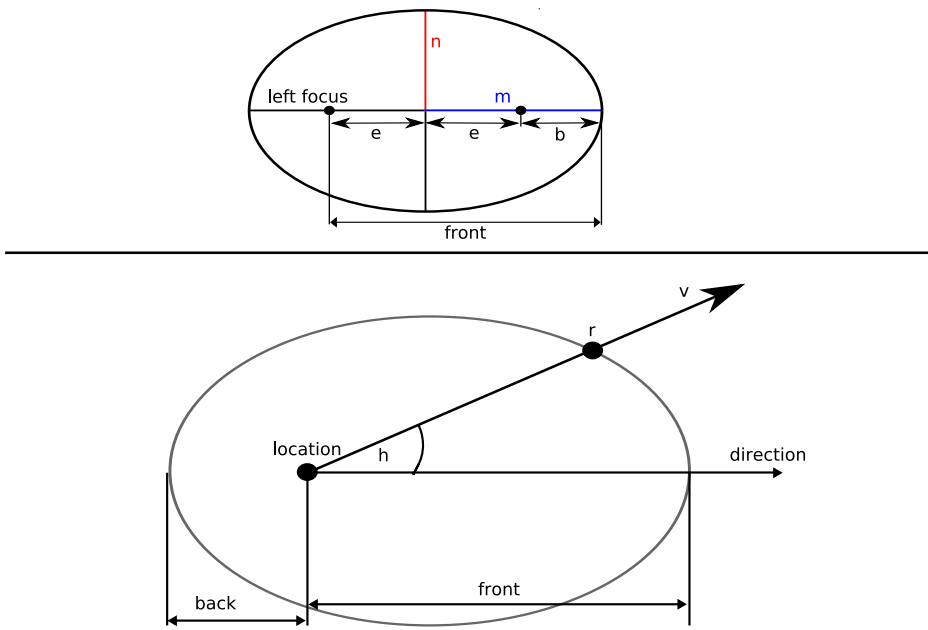


Figure A.1: The upper figure shows the half major m and the half minor n as well as the linear eccentricity e being the distance from the center of the ellipse to one of the foci. The lower figure shows a cut through the ellipsoid having its left focus at `location` in 3D space. The extension along `direction` is `front` and along `-direction` is `back` where `direction` is a 3D vector and `front` and `back` are scalar quantities. The ellipsoid has a extension of r in direction of a certain vector v . h is the angle between v and `direction`.

where it is the distance from the center of the ellipse to one of the foci.

From figure [A.1](#) one can see that

$$\begin{aligned}
 front &= 2 \cdot e + b \\
 &= 2 \cdot e + back \\
 \Leftrightarrow e &= \frac{front - back}{2}
 \end{aligned} \tag{A.4}$$

p can be expressed in dependency of the m and n :

$$p = \frac{n^2}{m} \tag{A.5}$$

where m and n can be computed as follows (see also [[2](#), pages 344/345]):

$$m = \frac{front + back}{2} \tag{A.6}$$

which is half the extension of the ellipse along *direction*

$$n = \sqrt{front \cdot back} \tag{A.7}$$

can be derived as follows:

Using the definition of e from [\(A.3\)](#) yields

$$e = \frac{c}{m} \Rightarrow n = \sqrt{m^2 - e^2} \tag{A.8}$$

Now plugging in the formulas [\(A.6\)](#) and [\(A.4\)](#) yields

$$n = \sqrt{\left(\frac{front + back}{2}\right)^2 - \left(\frac{front - back}{2}\right)^2} \tag{A.9}$$

$$= \sqrt{front \cdot back} \tag{A.10}$$

The remaining unknown in equation [\(A.1\)](#) is h which can be computed as follows:

$$h = a \cos\left(\left\langle \frac{direction}{\|direction\|}, \frac{v}{\|v\|} \right\rangle\right) \tag{A.11}$$

Appendix B

Scene Documents

The X3D standard supports several data formats to describe the virtual environments. The scenes in this section are encoded in VRML file format.

B.1 Minimalistic Audio Scene

This document in VRML file format describes a scene having a sound source at location (1, 2, 1) represented by the **Sound** node. The sound source has a **intensity** of 0.9 and the ellipse of the X3D sound model (see figure 3.1 in chapter 3) is described by the fields **maxBack**, **maxFront**, **minBack**, **minFront** and **direction**. The audio data, represented by the **AudioClip** node, are stored in a wav file defined by the **url** field. The **loop** field having the value **TRUE** specifies that the playback is repeated once it has finished.

```
1 #VRML V2.0 utf8
2 Sound {
3     source AudioClip
4     {
5         url "mono.wav"
6         loop TRUE
7     }
8
9     intensity 0.9
10    location 1 2 1
11    direction 1 0 2
12
13    maxBack 4
14    maxFront 12
15    minBack 2
```

```
16     minFront 6
17 }
```

B.2 Multichannel Scene

The scene described by this document in VRML file format contains a sound source and geometric objects associated with the sound source which can be used to decompose the stereo sound source into mono sound sources by applying `MultiChannelSplitStrategies` as described in paragraph 6.2.1.4.1. These geometries are boxes, one placed in the origin of the virtual environment and the other placed at (8, 0, 4). The `metadata` field of the `Sound` node associates the sound source with the geometries by referencing to nodes which have been defined previously by the `DEF` keyword.

```
1  #VRML V2.0 utf8
2  Transform
3  {
4      translation 8 0 4
5      children DEF Speaker1 Shape
6      {
7          geometry Box{ }
8      }
9  }
10
11 # geometric representation of other mono source
12 DEF Speaker2 Shape
13 {
14     geometry Box{ }
15 }
16
17 Sound
18 {
19     source AudioClip
20     {
21         url "muxed_stereo.wav"
22         loop TRUE
23     }
24     metadata MetadataSet
25     {
26         value [USE Speaker1 USE Speaker2]
27     }
28 }
```

Bibliography

- [1] <http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/>
X3D Specification ISO/IEC 19775:2004
- [2] DON BRUTZMAN AND LEONARD DALY *X3D: Extensible 3D Graphics for Web Authors*, 2007
- [3] DR. RÉMI ARNAUD AND TONY PARISI *Developing Web Applications with COLLADA and X3D*, March 2007
- [4] <http://www.collada.org>
Collada
- [5] MARK BARNES AND ELLEN LEVY FINCH *COLLADA - Digital Asset Schema Release 1.5.0 Specification*, April 2008
- [6] ECMA INTERNATIONAL *Standard ECMA-363 Universal 3D File Format*, June 2007
- [7] <http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/>
VRML Specification
- [8] HUI-JUN REN AND DA-KUN ZHANG *Virtual Scene based on VRML and Java*, December 2007
- [9] <http://www.web3d.org/about/overview/>
X3D overview

- [10] <http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/Part01/components/sound.html>
Sound component of X3D Specification ISO/IEC 19775:2004
- [11] <http://www.iosono-sound.com/>
IOSONO audio system
- [12] <http://sourceforge.net/projects/osgal/>
Open scene graph audio library
- [13] DIRK REINERS *OpenSG: A Scene Graph System for Flexible and Efficient Realtime Rendering for Virtual and Augmented Reality Applications*, 2002
- [14] DIRK REINERS, GERRIT VOSS AND JOHANNES BEHR *OpenSG: Basic Conce*, February 2002
- [15] <http://www.openscenegraph.org/projects/osg>
Open scene graph
- [16] <http://www.gstreamer.net/>
GStreamer
- [17] <http://msdn.microsoft.com/> Microsoft Developer Network
- [18] <http://developer.apple.com/quicktime/> QuickTime
- [19] APPLE INC. *QuickTime Overview*, August 2005
- [20] <http://www.networkmultimedia.org/>
Network-integrated Multimedia Middleware
- [21] MARCO LOHSE, MICHAEL REPPLINGER, FLORIAN WINTER AND PHILIPP SLUSALLEK *Network-Integrated Multimedia Middleware (NMM)*, October 2008
- [22] <http://www.niallmoody.com/heilan/>
Helian X3D browser
- [23] <http://www.instantreality.org/home/>
Instant reality - Instant Player

- [24] <http://www.octaga.com/joomla/index.php>
Octaga Player
- [25] <http://www.mediamachines.com/developer.php>
Flux Player
- [26] http://www.bitmanagement.com/products/bs_contact_vrml.de.html
Bitmanagement Contact
- [27] <http://www.h3dapi.org/>
H3D API and H3D Viewer
- [28] <http://www.xj3d.org/>
Xj3D Browser
- [29] <http://openvrml.org/>
OpenVRML
- [30] DMITRI RUBINSTEIN *RTSG - Design and Implementation of a Scene Graph Library based on Real-Time Ray Tracing*, September 2005
- [31] ILIYAN GEORGIEV, DMITRI RUBINSTEIN, HILKO HOFFMAN AND PHILIPP SLUSALLEK *Real Time Ray Tracing on Many-Core-Hardware*, October 2008
- [32] <http://de.wikipedia.org/wiki/Ellipse>
Wikipedia - Ellipse
- [33] THEO KÜHLEIN *Mentor-Repetitorien, Band 27, Analytische Geometrie II*, 1967
- [34] <http://audiere.sourceforge.net/>
Audiere
- [35] <http://www.libSDL.org/>
Simple Directmedia Layer
- [36] <http://www.radgametools.com/miles.htm>
Miles Sound System
- [37] <http://www.fmod.org/>
FMOD music and sound effects system

- [38] <http://kcat.strangesoft.net/openal.html>
OpenAL Soft
- [39] <http://connect.creativelabs.com/openal>
OpenAL
- [40] CREATIVE LABS, INC. *OpenAL Effects Extension Guide*, 2006
- [41] <http://www.creative.com/>
Creative Labs Inc.
- [42] LAURI SAVIOJA *Modeling Techniques for Virtual Acoustics*, August 2000
- [43] PETER SVENSSON *The Early History of Ray Tracing in Room Acoustics*, June 2008
- [44] J.P. COYETTE, G. VANDERBORCK AND W. STEICHEN *A coupling procedure for modeling acoustic problems using finite elements and boundary elements*, May 1994
- [45] SHARAF HAMEED AND VILLE PULKKI *Modeling of Coloration of Virtual Sound Sources in Listening Rooms*, June 2004
- [46] THOMAS FUNKHOUSER, JEAN-MARC JOT AND NICOLAS TSINGOS *'Sounds Good to Me' Computational Sound for Graphics, Virtual Reality, and Interactive Systems*, 2002
- [47] THOMAS SPORER *Wave Field Synthesis - Generation and Reproduction of Natural Sound Environments*, October 2004
- [48] BENJAMIN SCHUG *An Extensible Rendering Interface and an OpenGL Renderer for the RTSG Scene Graph Library*, September 2008
- [49] <http://www.ogre3d.org>
Ogre 3D graphics rendering engine
- [50] JOHANNES BEHR, PATRICK DÄHNE AND MARCUS ROTH *Utilizing X3D for Immersive Environments*, 2004
- [51] FREDERICK P. BROOKS, JR. *What's Real About Virtual Reality*, November/December 1999

- [52] DAVID MURPHY AND FRANCIS RUMSEY *A Scalable Spatial Sound Rendering System*, 2001
- [53] DOUGLAS C. SCHMIDT AND CHARLES D. CRANOR *Half-Sync/Half-Async An Architectural Pattern for Efficient and Well-structured Concurrent I/O*, September 1995
- [54] DOUGLAS SCHMIDT, MICHAEL STAL, HANS ROHNERT UND FRANK BUSCHMANN *Pattern-oriented Software Architecture Volume 2 - Patterns for Concurrent and Networked Objects*, 2001
- [55] <http://sgl.sourceforge.net/>
SGL
- [56] <http://java.sun.com/javase/technologies/desktop/java3d/>
Java3D
- [57] <http://www.sgi.com/products/software/performer/>
OpenGL Performer
- [58] <http://www.openrm.org/>
OpenRM
- [59] E. WES BETHEL *White Paper: OpenRM Scene Graph Thread Safety and Multistage Rendering*, July 2001
- [60] http://developer.nvidia.com/object/nvsg_home.html
NVIDIA Scene Graph
- [61] <http://libx3d.sourceforge.net/>
X3D lib
- [62] <http://open-activewrl.sourceforge.net/>
Open ActiveWrl