COLLADA Audio : A Formal Representation of Sound in Virtual Cities by a Scene Description Language

Shih-Han Chan

le cnam

CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS

École Doctorale Informatique, Télécommunication et Électronique
Equipe ILJ – Groupe MIM – Laboratoire CEDRIC

THÈSE présentée par :
SHIH-HAN CHAN
soutenue le : 20 Décembre, 2012

pour obtenir le garde de :
Docteur du Conservatoire National des Arts et Métiers
Informatique / Multimédia

COLLADA Audio:
A Formal Representation of Sound in Virtual Cities by a Scene Description Language

Thèse dirigée par :
NATKIN Stéphane
TOPOL Alexandre

Rapporteurs :
ESTRAILLIER Pascal
JANG Jyh-Shing Roger

Jury :
ARNAUD Rémi
CUBAUD Pierre (Président du Jury)
ESTRAILLIER Pascal
JANG Jyh-Shing Roger
LARCHER Véronique
NATKIN Stéphane
SCHWARZ Diemo
TOPOL Alexandre

Your hearts know in silence the secrets of the days and the nights. But your ears thirst for the sound of your heart's knowledge.

- Kahlil Gibran, *The Prophet*

ACKNOWLEDGMENT

I have always been lucky enough to meet great people who give me a hand to conquer the tasks throughout my adventures. My appreciation to all of those who have practically and/or mentally helped me in any respect:

First and foremost, I am sincerely grateful to my Ph.D. supervisor, Prof. Stéphane Natkin, for giving me a lot of freedom in my work. He enlightened me and guided me through the confusion. With his advice, enthusiasm and encouragement, he has served the most important role in facilitating my growth as a researcher and making this thesis possible.

I would also like to state my gratitude to my co-advisor Dr. Alexandre Topol, who I enjoy having discussions with. I deeply appreciate his efficient working style, positive attitude and brilliant inspirations, which I always find helpful.

I am indebted to my ILJ staffs and colleagues I had the pleasure to work with. I would like to thank Cécile Le Prado and Guillaume Tiger for sharing valuable opinions and knowledge in aspect of sound design and engineering, Guillaume Levieux for helping to sort out problems during the project, and Shuohsiu Hsu for sharing survival tips even before I took the challenge to come to CNAM without knowing how to speak French.

I am grateful to my NTHU master's thesis advisor, Prof. Jang Jyh-Shing Roger, as well as my NCTU college project advisor, Prof. Chuang Jen-Hui, for bringing me to world of entertainment technology and, moreover, encouraging me to step forward to explore the beyond. I am also heartily thankful to my CMU-ETC mentor, Prof. Donald Marinelli, for providing me opportunities to learn and/from play across the globe.

I wish to thank my family, my brother and sister-in-law, and most importantly, my parents Chan-Chen Tsung and Chiang Jui-Ying, for their infinite love and support. When it comes to "compromise", it most likely ends up with them letting me do whatever makes me happy, including pursuing a Ph.D. degree in France.

I wish to thank my companion, *Treasure Bowl*, for your extreme patience with the most dislikable side of me. You are the faith that made me believe I could actually accomplish this. With no doubt I know that in the coming decades, when I look back on these difficult days of writing, it will certainly be your face and voice on my 3.5" phone screen and the window view of the little park from my 24m^2 flat that will first come to mind.

Lastly, I thank myself for being who I am.

Shih-Han Chan @ Paris FR, November 2012

ABSTRACT

Standardized or normalized file formats has been conceived and argued since many years to write, read, and exchange 3D scene descriptions. These descriptions are mainly for visual contents whereas options given for audio compositions of virtual scenes are either lacking or poor. Therefore, we propose to include rich sound descriptions in the COLLADA, which is a standard format for scene descriptions and digital asset exchanges.

Most scene description languages that include a sound description factorize common elements needed by the graphical and auditory information. Both aspects are, for example, described with the same coordinate system. However, as soon as a dynamic description or external data are required, this benefit is lost and all the glue must be done by a programming approach that does not fit designers or authors usual skills. In this thesis, we address this problem and propose to give the bigger role back to the designers even when the scene is dynamic or based on procedural synthesizers. This solution is based on the COLLADA schema in which we have added sound support, scripting capabilities and external extensions. The use of this augmented COLLADA language is illustrated through the creation of a dynamic urban soundscape.

Our work is supported by a collaborated research project, *Terra Dynamica*, which aims to develop 3D urban simulation technology that can be applied to various purposes. The task assigned is to integrate a 3D sound engine into the project to simulate real-time sound behaviors within a dynamic urban scene. In this context, we implement and experiment sound capabilities in COLLADA.

Keywords: COLLADA, scene description language, audio scene graph, soundscape, virtual cities

This page intentionally left blank.

TABLE OF CONTENTS

TABLE OF FIGURES

This page intentionally left blank.

CHAPTER 1.

Introduction

OUTLINE

1.1 Scene Description and COLLADA

Nowadays, almost all applications exploit the senses of sight and hearing of users in order to notify tasks execution and achievement. Visual and aural objects can be generated in different ways, by several techniques, which might be used within the same application. For example, in a video game, graphics can be made of 3D models textured with still (2D bitmap) or moving (2D videos) images and be animated either by key-framed (animation files) or physics (a set of parameters). For the auditory part, on the other hand, a game can play stereo music or synthetized ambiance mixed with spatialized sound sources with different effects. One of the common approaches to associate all these different resources together is to code inside the application the initialization of individual assets and the connection

between them. This can be done either in a fixed way by hard-coding it or by using a "level editor" like those implemented in many popular game engines (e.g., Unity3D, UDK, or CryENGINE). The second way of building a scene by spatializing audio-graphic objects (or assets or prefabs) is undeniably very powerful as it vastly simplifies the work of programmers and their communications needed with game designers. However this method is proprietary and exploits generated files in an unknown binary syntax and is tightly linked to the underlying game engine. It is absolutely not a general description that can be shared between different software solutions to ease the sharing of resources. For this reason, many content creators prefer to use open, loyalty-free scene languages to describe 3D multimedia environments.

Standard interchange format files enable assets to be exchanged across applications and platforms and prevent the loss of data during the transport. It has been many years since the technique was applied to describe virtual audio-graphic three-dimensional scenes. Nevertheless, the descriptions are mainly for visual representations including the information of transforms, textures, and visual effects, whereas there is a lack of options for sound simulations. Acoustic utilities in existing scene description formats are incomplete, out of date, or too sophisticated for practical realization. Although, from the audio aspect, MPEG-4 AudioBIFS provides the most advanced ability, the interpretation of the whole capability of MPEG-4 scene description has never been implemented. Moreover, present standards are usually dedicated to computer music technology (e.g., SDIF and SpatDIF) or web content delivery (e.g., SMIL and X3D) instead of virtual world interactivities. We figure that a scene description that meets our expectation of bringing sound into the simulations of urban environment has not yet been created. Consequently, a novel standard for sound representation is demanded.

COLLADA is a favored option as it is easy to use, exchangeable and transportable among various applications and platforms. The visual contents of virtual agents are able to be described in a COLLADA file, whereas sound capabilities are not. To be consistent, there are two possible solutions for us to add sounds to this virtual muted COLLADA description: First, to describe the whole sound scene independently in a convenient file format that would be generated from the database. Second, to add the sound description correlated with the graphics scene description. Since computing an image viewed and computing a sound heard from a given position in a virtual environment both require a common set of data, it would be disk and memory consuming to duplicate the mutual information. Furthermore, the time required to generate, read and interpret both files would be greater than with a single file in which data is optimized. Therefore we choose the second solution: to add the

sound description in the COLLADA standard which has already support the visual capability.

1.2 Proposition of COLLADA Audio

As we propose to introduce sound capability into the COLLADA schema, there are some principles and goals that we consider. First, we want the COLLADA audio scene to be well-matched in spirit and syntax with the current COLLADA schema so the representation can be coherent and comprehensible. Second, we need to avoid specifying data that are already in the scene description in case of repetition. Third, sound in COLLADA must provide the basic acoustic capabilities defined in MPEG-4 AudioBIFS and support advanced developments in sounds that were not included. Lastly, we suggest the integration of the urban soundscape layers with the scene-graph hierarchy of COLLADA in order to increase a sense of aural depth.

The implementation of COLLADA audio scene design includes both static and dynamic descriptions. The classic static part of the audio scene is an object-oriented composition of numerous audio nodes of which the definition and functionality rely on existing scene description standards, such as MPEG-4 AudioBIFS and AAML. The scene represents a flow graph from bottom to top describing digital signal processing manipulations. There are basically three varieties of the auditory nodes: Audio Stream elements contain sound stream data throughout the signal processing chain; Audio Effect elements provide the control of DSP sound effects; and Audio Scene element specifies an environment where acoustic objects are instantiated and simulated. Since COLLADA has already contained three natures of scene descriptions (i.e., visual, physics, and kinematics), it is important to follow the same design philosophy when defining the audio scene in order to maintain the consistency of the overall structure.

On the other hand, we integrate the event-driven programming technique (i.e., event detection/selection and event handling) into the hierarchical XML schema of COLLADA in order to provide some more dynamic control compared to the traditional capability. The dynamic section of the audio description also takes the advantage of COLLADA programmable units to allow inline or external scripting codes for handling complex objects and interactivities. We suggest that the adaptive descriptions should be available for not only the sound representation but also the other categories of simulations, such as visual and physics. Although, COLLADA is meant to provide a standard format to define 3D assets

but not their runtime semantic, it is not our intention to change its position in the content pipeline. Despite the fact that the behaviors of sound are comprehensively described within the COLLADA script, it is the application side that will determine which contents to be realized.

A traditional application initializes and renders the visual and audio contents independently. Rendering one by one at a time is obligatory since they do not use the same pipeline and the same rendering device. An application loads or generates the correct parameters for each media in the initialization process so it will most likely have to make some changes to the coordinate system and/or duplicate some structures in memory. Hence, associating graphical and acoustic information together in the same resource file would ease the overall audiovisual description of an application. We argue that our proposal is economic in terms of resources needed for implementing an application. The audio profile is organized together with profiles in other categories inside a unified scene description language instead of being described in its own sound description file or being programmed directly inside the application.

The CEDRIC (Computer Science laboratory of the CNAM) is involved in *Terra Dynamica*, a project funded by the French government. The purpose of this three-year project is to bring life to *Terra Numerica*, a static virtual city (Paris in experiment). *Terra Dynamica* aims to develop 3D urban simulation technology that can be applied to various domains, such as realistic simulations of urban transportation, city security, and citizen behaviors, as well as video games and interactive art installations. Bringing life is basically putting animated characters in the city. Nevertheless, making pedestrian and car agents behave properly would not be realistic if they were muted. Even though sounds are often simply noises in a city, they still carry important information for real or virtual citizens. In this context, one of the tasks assigned to the CNAM is the development of a 3D sound engine that simulates real-time sound behaviors within dynamic urban scenes. The investigation mainly focuses on the nature of the urban acoustic scene and its symbolic representation and layout structure, as well as the dynamic synthesis of the soundscape. The virtual city of *Terra Dynamica*, composed by the terrain, roads, buildings or furnishings, is depicted in a database which is translated to a COLLADA document. For this reason, the project provides us a great platform to demonstrate the idea of representing sound in virtual cities by the COLLADA sound capabilities.

1.3 Organization of the Thesis

The next chapter investigates the state-of-the-art technologies and practices with regard to virtual environments, scene representations, and the COLLADA standard. We present the possibility of delivering promising sounds in virtual worlds. There is a discussion of missing functionalities of the current scene description languages for the purpose of reproducing the daily acoustic experience in a variety of virtual environments applications. So we propose to enhance the COLLADA format with the audio capability.

Chapter 3 introduces a novel approach to standardize audio description in virtual cities using the COLLADA schema. The implementation of the audio scene design, including both static and dynamic descriptions, is in line with the proposed design principles. Furthermore, we discuss the use of COLLADA audio during the development of interactive applications with regard to the software tools. For instance the sound engine and game engine that control the playback and spatialization of the sound representation in game worlds. Besides, a scene editor is expected for designers to establish COLLADA audio scenes, and to enable the import and export of the auditory scenes with the COLLADA standard.

In chapter 4, we present several prototypes and a 3D sound engine as the proof of concept to validate the proposed method. The 3D sound engine that we developed has been integrated with the core of *Terra Dynamica* API so can be employed to a wide range of interactive virtual city applications in this project. Moreover, from the point of view of sound design, we examine the feasibility of the support of designer-generated sound contents, such as FMOD events and Pure Data patches.

Lastly in the final chapter, the contribution of this study is concluded. It presents some suggestions for potential future works. Following the conclusion are the appendices for the references of MPEG-4 and COLLADA audio nodes, and common acoustic DSP functions.

For the video/audio demonstrations, please refer to the link below:

http://sites.google.com/site/shihhanchan/thesis/collada-audio

This page intentionally left blank.

CHAPTER 2.

State of the Art

OUTLINE

Before putting forward the proposal of creating a formal representation of sound in virtual cities, it is essential to have a fundamental understanding of the related research works, including their evolution and the cutting-edge achievements. Thus in this chapter, we survey the state-of-the-art technologies and practices.

In the first part of the chapter, the knowledge of virtual urban environment is reviewed with a concentration on the audio aspect. We look into how auditory space can be composed and categorized, from abstract to applicable analysis, based on the soundscape theory; we also study what kinds of roles sounds play and how it is developed in different fields of interactive applications. Furthermore, the generic technologies and techniques applied for building interactive virtual worlds and experiences are investigated.

A virtual scene can be presented not only graphically but also acoustically. Scene description languages are becoming a mainstream to depict the virtual space and the complex dynamic behaviors of the components residing in it. Many of these languages are scene-graph oriented. The second section of this chapter brings together different scene description languages cited explicitly in the sound compositing capabilities. We evaluate the audio capability in major scene description formats, arguing the lack of some sound functionality for interactive simulations.

Lastly, as we suggest supplementing the COLLADA format with an audio scene, we must examine the design philosophy of COLLADA in order to keep the big picture in focus.

2.1 Virtual Environment and Techniques

2.1.1 Overview

Virtual **R**eality (VR) or **V**irtual **E**nvironment (VE) refers to digital computer-based environments that represent the simulation of the real world or the creation of imaginary worlds. Having the capability of user interaction, virtual reality technology can be applied in a tremendous variety of domains, such as entertainment, education, arts, training, product evaluation and testing, and so on. Video games are a representative type of interactive virtual media presenting rich contents like text, graphics, audio, video and animation.

Senses of immersion, engagement, and presence (or so-called "flow") have been believed as the pleasure principle as well as crucial analysis aspects of three-dimensional interactive virtual reality environments [DH00][M03]. That is to say, the player must feel in an open interactive world, and should be driven to the game solution. To solve this paradox, the game industry has invented several techniques derived from the game theory and object oriented specification [VLN02].

The video game industry in particular has contributed to visual and audio experiences that engineers can incorporate into the design of virtual reality systems. During the game development, several professions are involved, including game designers, level designers, **U**ser **I**nterface (UI) designers, artists, sound engineers and designers, programmers (software engineers), and game testers. Technical efforts are specifically across the following disciplines: graphics, sound, physics, **A**rtificial **I**ntelligence (AI), network, gameplay, scripting, UI, input processing, and game tools.

In this section, the knowledge of virtual urban environment is reviewed with a concentration on the audio aspect.

2.1.2 Sound in Virtual Cities

The need for representations of the urban sound comes from acoustic ecology and noise pollution issues. Two main numeric linear mapping tools arise from these researches: sound maps (referencing static field recordings or dynamic soundwalk recordings) and noise maps (representing the sound level distribution and deduced qualitative attributes). Mapped representations are most of the time empirical auditory content overlaid on mapped space. Considering the city as a designed space, its relationship to sound can be understood as a dynamic interaction, first through the analysis and composition paradigms of the soundscape theory, then with regard to the architectural digital acoustic tools and finally as a real-time construction within video games.

2.1.2.1Usages of Sound in Virtual Cities

Sound in virtual cities can be deliberated from an applicable usage policy. In the following, we select five representative applications of sound in virtual cities and analyze the auditory functionality in each of the media.

- Google Earth

Google Earth is a virtual earth representation (globe, map and geographical information) that is recognized worldwide. It lets users fly anywhere on Earth to view satellite imagery, maps, terrain, 3D buildings, from galaxies in outer space to the canyons of the ocean[GEARTH]. Users are allowed to add their own data in Google Earth. Some developers, including the company "Wild Sanctuary", have released a 3D sound layer for the application using a collection of recorded sound sources.

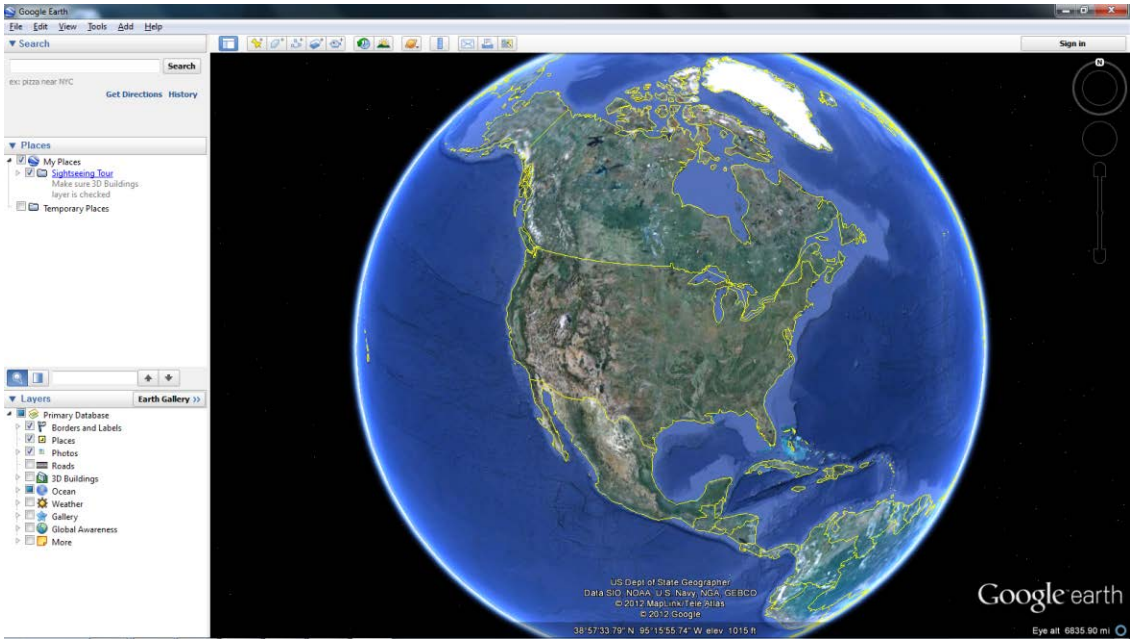


FIGURE 2-1. Google Earth (Google).

- Second Life

Second Life is a well-known multipurpose online virtual world where "residents" (Second Life users) can interact with each other through avatars. *Second Life* supports in-world sound clips (.wav format), which can be played to all residents within range.

FIGURE 2-2. Second Life (Linden Lab).

- Prototype

Prototype is an action-adventure video game developed by Radical Entertainment. It intends to reproduce a dynamic ambience of New York City using an original recorded sound map [M08]. A three-tire soundscape ambience is applied based on the objects inside the game world and their relative densities, populations, and emotional states.

FIGURE 2-3. Prototype (Radical Entertainment).

- Soundwalk

Soundwalk allows combining a localized sound recording with the real sound of the city. It is used for various purposes, such as noise evaluation, soundscape analysis and urban planning, real world walks, musical composition.

FIGURE 2-4. Soundwalk (Soundwalk.com).

- Locustream

Locustream is an "openmike" soundmap employed for several practices, for example, webstream, art installations, raw material for live music.

FIGURE 2-5. Locustream SoundMap (Locus Sonus).

TABLE 2-1 summarizes the usages of sounds in the above example of interactive applications in different genres.

TABLE 2-1. Intersection of different types of virtual urban media and sound functions.

MEDIA / FUNCTIONS	GOOGLE EARTH	SECOND LIFE	PROTOTYPE	SOUND WALK.COM	LOCUS STREAM
IMMERSION	Static recordings	Public spaces, sound art works and private spaces	Music, surround sound, real-time spatialization	Ambient sound, read text	Real-time audio streams
NAVIGATION HELP			3D sound	Audio guide	
INFORMATION	Relative to listenable audio files	Positioning objects (3D sound), relative to audio content	Character behaviors, density / proximity of agents	Historical and culture information, localization	Relative to audio stream broadcasts
SOCIALIZATION		Concerts, adding sounds to avatars, private audio broadcasts			User community
STORYTELLING			Interactive music, dialogues	Sound path divided into sequences	
REUSE OF DATA					Utilization of streams as sources

*Red text shows functions for which users may bring content.

As a conclusion of our analysis, textures, rhythm, and discontinuity are features specific to virtual urban sound spaces. Applications constructed in layers remind us of the classical theory of soundscapes. The dynamic organization of the soundscape is based on the needs created by the gameplay or rules of the media.

2.1.2.2 Sound-Source Clustering

Level **O**f **D**etail (LOD) is a technique widely applied in 3D computer graphics. The norm of managing LOD is to increase the efficiency of graphics rendering by decreasing the complexity (quality) of 3D object visual representation and can still preserve the fidelity of the simplified models. The most common LOD approach is based on mesh polygon reductions of distant or small objects.

Given that more and more interactive virtual cities are involved with an enormous number of graphical actors with sounds, an analogous capability of levels of detail for audio is revealed for memory and disk saving. The process of **S**ound **L**evel **O**f **D**etail (SLOD) for audiovisual scene is similar in concept to the LOD for graphics. One of the SLOD solutions is grouping sound sources by the similarity.

Traditional audio source clustering techniques, as depicted in FIGURE 2-6, group point-sources by their distances to the listener. Each set of sound sources is meant to be replaced by a single representative (usually at the center point of the group), possible with more complex acoustic characteristic (e.g., impulse response). Such impostor sound sources are used to render or spatialize the aggregate audio streams. Some other sound-source clustering strategies have been described on the basis of re-synthesis [TGD04] [SCB11].

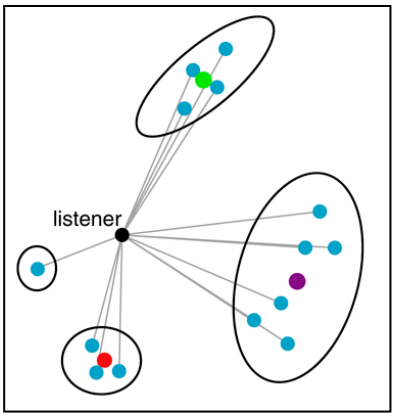


FIGURE 2-6. Paradigm of sound-source clustering techniques [TGD03].

(Blue dots: sound sources; Colored dots: representative.)

2.1.2.3 Soundscape

2.1.2.3.1 Theory

The term soundscape is the combination of sound and landscape. It is coined by Canadian composer and environmentalist, R. Murray Schafer [S77]. According to Schafer, a sonic environment is depicted with three major layers of the soundscape: keynote sounds, soundmarks, and sound signals, creating multidimensional objects. Walter Murch echoes this theory in film sound taxonomy in [WB85], where sound is categorized into background, midground, and foreground respectively.

- Background (Keynote Sounds)

Keynote as a musical term identifies the lowest note upon which a scale is based. In the soundscape theory, it refers to background listening that all other pieces of sounds in the soundscape are associated to. The keynote sounds outline the character of the people living in the certain community, but it may not always be audible. In general, sounds created by nature form the environmental keynote: water, winds, animals, etc. In urban spaces, traffic is considered as the background sound.

- Midground (Soundmarks)

The soundmark is derived from the term landmark. It is a unique audio component specifically to a certain place or region. The audible zone of a soundmark helps to sonically define a community with social and cultural associations. The repeating bell ringing sound in a church is a common soundmark. London's Big Ben and the Steam clock in Gastown in Vancouver are some other classic examples of soundmarks.

- Foreground (Sound Signals)

Signals are foreground sounds that often carries encoded and interpretable messages. Therefore, in contrast to the keynote sounds, they are listened to consciously. The examples would be sounds that are generated as warnings, for example, whistles, vehicle honks, or sirens.

2.1.2.3.2 Applicability

The soundscape concept may be applied, for analytic and creative purposes, to real environments or abstract constructions, including virtual ones [LN07]. The techniques used

to simulate the urban soundscape range from layering of prerecorded sounds to various methods of synthesis [VLS09][VAM10][SCB11].

Due to that the basic soundscape taxonomies are significantly related to human perceptual habits, they are easily applicable for the synthesis and analysis of all such electroacoustical soundscapes, usually referring to virtual acoustic environments [T08]. Today, auditory environment design is not bounded in film soundtracks composing but largely being invested in the digital game industry where complex and detailed soundscapes are the norm, both for realistic and fantasy worlds [T07].

Prototype, a Radical Entertainment game, intends to create a dynamic ambience of New York City [M08]. Not separating the city zone-by-zone with detailed sounds, the desire was alternatively to base the ambience on the objects inside the game world and their relative densities, populations, and emotional states. To do so, they broke the ambience down into three "tiers" based on the concept of soundscape theory.

- The background ambiences were a quadraphonic track of the more distant perspective recordings of Manhattan: a Central Park background and a rooftop background that would fade based on the position of the listener.

- The midground layer was composed of a collection of clustered components in the game world, including grouped pedestrians, vehicles and infected crowds.

- Lastly, foreground ambiences were a composition of sounds from single objects in the environment, played based on the changes of state of these objects.

A smooth transition from foreground through midground to background provides a sense of aural depth of the field while limiting the number of individual foreground sound components thank to the support of the midground ambiences.

2.1.2.4 Sound in Architectural Simulations

It has been believed that spatial audio is an essential element to deliver a more immersive and believable virtual reality experience. A sound is not merely an emission from a source. That is to say, sound waves can be absorbed (attenuated), reflected or refracted by the medium. Audio in soundscape theory, architectures and video games, is characterized by its

strong relationship to space regarding both analytic and creative perspectives. This non-linear relationship exists through time and is developed within many cross-disciplinary works [F06].

In terms of virtual representations, architecture provides advanced acoustic space simulations [FJT02]. As it is dynamically organized in propagation fields (direct, diffuse, critical), the architecture auditory space is related to soundscapes perspective though the aim of acoustic simulation providing an objective auralized simulation based either on deterministic or newer probabilistic models. These models require heavy computation and seem not to be yet effective in the reproduction of large and highly interactive environments.

Numerical simulations of the propagation of the sonic wave in acoustic spaces are presented for architectural design. In some interactive media, such as video games, dynamic factors are then added-up in accordance with the gameplay design. In addition, game sound middleware provides a choice of high-level runtime capabilities, such as aural levels of detail [TGD03], interactive spatialization mixing, and runtime reverberation and filtering [M08].

2.1.2.5 Sound in Video Games

Some video games, such as *Grand Theft Auto*, *Prototype*, *Mirror's Edge*, offer large real-time, interactive and three-dimensional urban environments. Unlike architectural simulations, video game sound design attempts to reproduce the physical phenomenon of sounds but tends to create a credible environment [C08b]. The strategies developed to reach such result rely on the exchanges between the sound engine and the game engine. Sound engines provide some high-level real time capabilities as SLODs, interactive spatialization mixing, runtime reverberation and filtering.

2.1.2.5.1 A Brief History

According to Natkin [N06][GLM02] and Collins [C08a], the evolution of audio technology within games may be divided into three ages. In the first generation of the development, early distribution systems, such as floppy disks and cartridges with read-only memory (ROM), had constrained the sound design by the lack of space for storage and computing power. The earliest video games and home consoles were even silent. *Space Invaders* developed by Taito/Midway first introduced the idea of a continuous background

soundtrack. The tempo of the music was affected by the player's progress in the game, reflecting that soundtrack could dynamically interact with the player.

The majority of 8-bit machines (and early arcade and pinball machines) used sound chips known as **P**rogrammable **S**ound **G**enerators (PSGs), usually limiting sounds to single waveforms without much manipulation ability. Nintendo improved on the 8-bit console sound with the release of **N**intendo **E**ntertainment **S**ystem (NES). The NES used a built-in five-channel sound chip with one waveform for each channel and came prepacked with its famous game *Super Mario Bros*. The 8-bit games produced a type of aesthetic for game music that can still be heard today and has its fans.

Instead of samples, a standardized protocol – MIDI was stored in code, revolutionizing the possibilities for game audio composing. During the mid-1980s, add-on PC sound cards began to develop. Techniques like FM synthesizer allowed game developers to use a broader assortment of instruments and sounds. Although by the early-1990s most FM sound cards supported MIDI, the sound quality was rather poor and disappointing.

The revolution came with the arrival of CD-ROMs available on personal computers and consoles (e.g., Sony PlayStations) offering a wavetable synthesizer and a rough 3D sound generator. With it high-quality recorded audio could be stored, real-time effects were allowed, and adaptive music would react to player and game engine input. However, to control the processor-load and RAM allocated to the sampled sound, synthesis and transformations are limited to the minimum necessary for managing the interactivity. Therefore, the composition was commonly a mix of multiple channels composing, for example, continuous stream of music for non-interactive animation or dialogues, recorded short loops used in ambience, and recorded or generated Foley effects with a simple real-time treatment. The sound design of present prevalent games rests on the same principles. The soundtracks of games like *GTA 3*, *Jack and Dexter*, and *Silent Hill 2* are typical examples in this context.

We are now in the third period in the advancement of sound within games. Computers and video game consoles in the latest generation (the seventh generation), including Microsoft Xbox 360, Sony PlayStation 3 and Nintendo Wii, have eliminated many of the technological difficulties of earlier games. New capabilities and the use of powerful sound cards, such as Creative Labs' Sound Blaster, perform efficient signal processing and real-time synthesis. Complex acoustic simulations as 3D localization and reverberation are involved as well. The potential of interactive and dynamic aspects of games audio has risen and begun to be explored. Games that use music as the primary narrative element, such as *Dance Dance*

Revolution and *Rez*, give interesting examples of the promising evolution aspects for the future progression.

2.1.2.5.2 Design and Production

To have a clear understanding of the modeling tools and sound design procedure, it is necessary to evaluate the production constraints. FIGURE 2-7 illustrates a common workflow of game audio production. In the early stage of the production workflow, sound designers produce auditory contents and classify them into three categories of audio components namely music, voices (dialogues) and sound effects. The actions of integration and tuning are taken in the next step, and then the final task is submitted to the editorial team for approval.

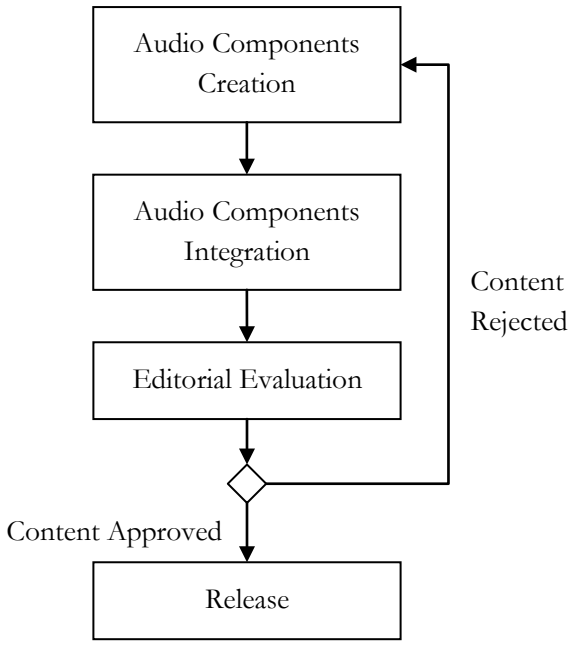


FIGURE 2-7. Game audio workflow [VNL06].

The creation of a game is carried out in four stages of process: Specification and planning, Pre-production, Development, and Validation and testing. The pre-production phase is mainly devoted to the game design, the creation of a significant prototype, and the refinement of cost and planning evaluation [GLN02]. The steps of a functional evaluation (Alpha) and a debugging process (Beta) are considered as the late period of the production.

The essential choices in the matter of sound, including the architecture of the three components and their relationships to game structure and to the images, are defined during the phase of game design [N06]. Unfortunately, as shown in TABLE 2-2, the sound is generally integrated into the game at the late stage of the production, when the other elements have already been constructed. The sound designer must therefore prepare all of the sonic elements without being able to test them in advance. Since there is only little time in prior to a complete content lockdown, any notion of post-production audio is challenged. While sound is now involved in game production a lot earlier, particularly by those with in-house audio teams who are involved in the planning and production of the games from day one, it is in fact the post-production audio phase that is overlooked during the scramble to lock down and stabilize the game [B07].

TABLE 2-2. An audio production model [B07].

Pre-Alpha	**Alpha**	**Beta**	**Sound Beta**	**Gold master candidate**	**Gold master**
(Major features complete)	(Rigorous testing of the features in the game begins)	(Feature complete and tuning complete)	(All sound content and code is finalized, mixed and tuned)	(Submitted to console manufacturers for testing)	(General availability release)

2.1.3 Content Pipeline of Interactive Applications

In general, an interactive application comprises two primary components: The first is the application providing real-time information to the user and the means to interact with it; and the second is the content containing the information which the application navigates through and provides a view to the user.

To separate the data physically from the code would allow creating several products with the same application but with different data. The content is the parts of the interactive application that do not execute managed code. It includes all art assets (e.g., models, meshes,

textures, sprites, effects, terrains, fonts, animations), audio assets (e.g., music, sound effects), and data assets (e.g., tables of levels, character attributes) [XNA]. The content can be generated in a wide variety of ways and stored in a wide variety of file formats.

The real-time application was referred to as the "runtime," and the content for the runtime was stored in the "runtime database." In the game industry, the runtime is called the "game engine."

Digital **C**ontent **C**reation (DCC) tools, such as a 3D model editor, are used by artists to create and update the source data. However, the data structures and algorithms used for modeling may not match with the data that can be processed in real time by the application. Interactive applications need not only the advanced modeling techniques, but also the simpler representation usable in real time. For this reason, compilation techniques were adapted for the content processing to compile the source data and transform it into the runtime data.

The content tends to change frequently in the course of the progress of interactive applications. The process starts with the source data in its original form as a file, and it continues to its conditioning transformations, and finally is encoded to be retrieved and read by the given runtime and often for the given target platform. This link between the content creation and the runtime is termed the "content pipeline." For example, an artist who creates a car model can add the resulting file into the game project, assign the model an id and/or a name, and select an importer and content processor for it. Then a game developer can load the car model in the game scene by its id (or name). This simple flow helps both designers and software engineers focus on their own work without having to spend much time on content transformation.

3D environments are time and money consuming to create and manage. Content creation is reputed to be the largest budget in the cost of developing 3D interactive applications. Therefore, the more that existing content and code for assets (e.g., textures, models, animation, shading, effects, physics) can be shared and reused, the more efficiently production pipelines can be made available. Given the need for larger and more interactive content, developers must spend substantial resources on this technology. The content pipeline is designed to be extensible, so that it can easily support new input file formats and new types of conversion.

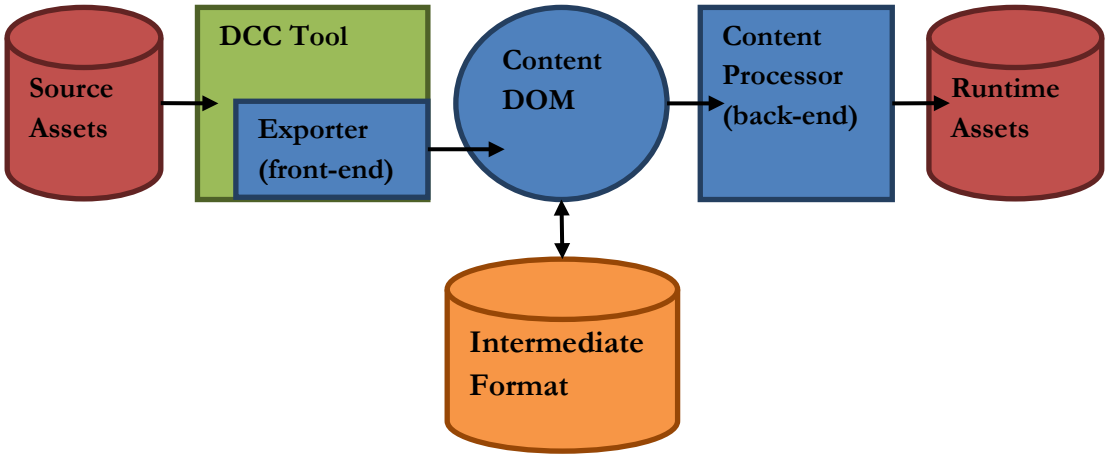


FIGURE 2-8. Content pipeline process.

2.1.4 Architecture for Games

What characterizes a game and more generally a virtual reality system is the fact that the image and sound must be generated in real time, in other words, in time with the player's perceptions.

FIGURE 2-9 schematically shows the mechanisms of production and display for a game. This process is a simulation that synthesizes or generates the images and sounds that the player perceives as a function of his actions. The top of the figure represents game production; the bottom represents what happens when a player is using the game.

Following the previous steps, a programming environment is built to allow game advancements. Visual and aural contents specified in the game design are respectively created by the teams of graphic artists and sound designers. Level designers define the geometry using a standard 3D tool such as Autodesk's Maya or 3ds Max. A scripting language is then used to specify the level in term of objects in the space. Programmers then implement the assets as classes in an object-oriented library using all the facilities of object-oriented programming (e.g., heritage and polymorphism).

Next, we look into the generic architecture and tools used through the process for game making with a specific interest in sound technology.

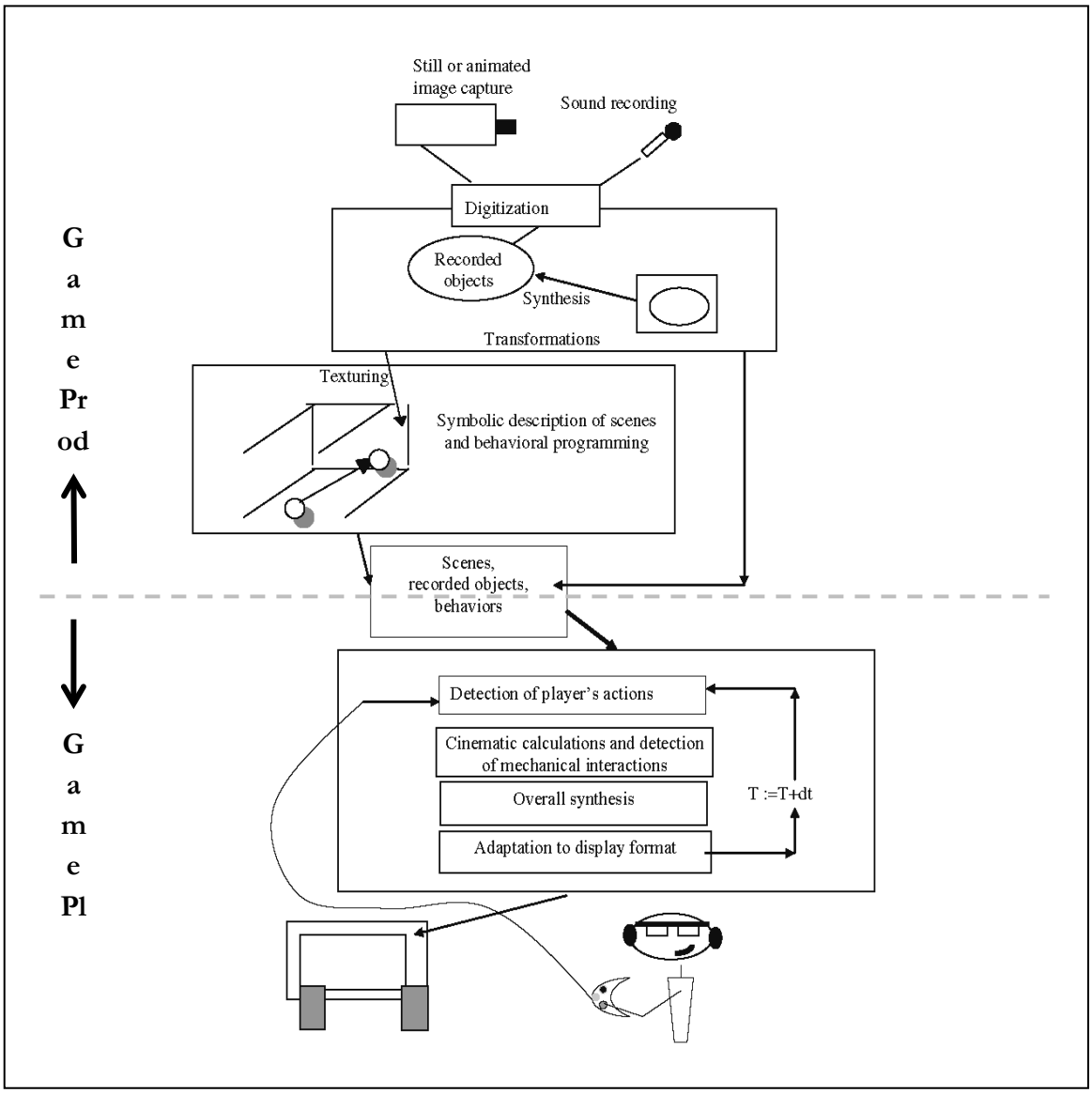


FIGURE 2-9. Mechanisms of production and display for a game [N06].

2.1.4.1Game Engines

The implementation of game is typically dependent on software termed a "game engine", which is a collection of software libraries. Each library in game engine executes a set of functions to perform the dynamic behaviors and interactivities in the game.

<table>
<tr><th colspan="5">Architecture Level</th><th>Examples</th></tr>
<tr><td colspan="5">Level Design
Scripts Editors</td><td>God.move(right, 2); Wait_Event;
onButton.click
God_Anger:=new(thunder)
God_Anger.lightening,
God_Anger.sound</td></tr>
<tr><td colspan="5">Game Classes</td><td>class thunder
methods: lightening, sound</td></tr>
<tr><td colspan="5">General Game Engines</td><td rowspan="2">Create_new_object(God,
god_geometry.vrml,
god_texture.gif,
god_voice.wav</td></tr>
<tr><td>Graphics Engine</td><td>Sound Engine</td><td>Physics Engine</td><td>AI Engine</td><td>Network Engine</td></tr>
<tr><td colspan="5">General Multimedia API (Application Program Interface)
(DirectX, OpenGL, OpenAL…)</td><td>GIMatrixMode();
alsourceplay(source1)</td></tr>
<tr><td colspan="5">Operating System</td><td>Windows, PS3 Monitor…</td></tr>
<tr><td colspan="5">Hardware</td><td rowspan="2">PC, PS3, Xbox 360, Wii</td></tr>
<tr><td colspan="2">Central Processor, memory, etc.</td><td colspan="2">Graphics Accelerator</td><td>Sound Card</td></tr>
</table>

FIGURE 2-10. Software and hardware architecture [N06].

The general-purpose game engines provide a software framework for the game creation. The essential functionality given by a game includes a 3D graphic engine used to display visual rendering results and handle interactive animations, a sound engine that manages sound synthesis, effects and spatialization functions, as well as some more specific libraries like a physics engine for physical system simulations and also an AI engine for gameplay.

These days, many of game engines have a software monitor, which is defined by the level design script, scheduling either on a synchronous mode or asynchronous mode. The synchronous monitor approach is almost always used on console platforms, whereas on PCs, the asynchronous approach is possible.

The use of portable game engines allows minimizing the work to be done to create multi-platform games. Typical example of game engine, such as Unity3D, Crytek's CryENGINE, and Epic's Unreal Engine (via the Unreal Development Kit, UDK), contains a 3D graphics rendering engine, a sound engine that is built in-house or originated from a third-party audio solution, a physics engine and an AI engine. Games developed with these

game engines can be, in principle, ported to cross-platforms including computers, consoles, and/or mobile phones.

Given the demands of faster adding or subtracting features, game development regularly uses scripting languages for enabling tweak-and-run cycles. Scripting languages have the facility to make changes on the fly without recompiling the project. Many game engines include an idiomatic scripting language tailored to the needs of the application user. Each game engine may interpret specifically the authenticate languages. For example, Unity3D has the self-defined UnityScript scripting language based on the fashion of JavaScript but it also recognizes C#; UDK supports its custom object-oriented scripting language UnrealScript (UScript); and CryENGINE uses C++ in combination with Lua.

2.1.4.2 Sound Engines

Game audio projects are commonly built upon a three-layer architecture – creation, integration and scripting, and finally restitution, illustrated in FIGURE 2-11.

The first layer of the structure includes signal processing operators, such as sound synthesizers and real-time filters that can compose and be applied to a sound stream which asks for balance between procedural sound design and wavetable playback. This is the level where programming code or sound editing tools operate directly on audio samples. Digital audio workstations, such as ProTools, Audition, Logic Pro, can be involved in this stage. Produced sound can be in diverse formats (e.g., wav, mp3, ogg, midi), sent to the sound engine for manipulations, such as spatialization or reverberation.

In the middle, the sound engine integrates the acoustic contents and transforms them to runtime assets by spatial audio composing toolsets. Most of the popular audio designing tools, such as Microsoft's XACT, Creative's ISACT, Firelight's FMOD Designer and Audioinetic's Wwise, have a Graphical User Interface (GUI) editor and/or native scripting language available for sound design and manipulations.

The base of the sound engine relies on an audio **A**pplication **P**rogram **I**nterface (API), which provides an interface communicating between higher-level audio libraries and sound cards or other sound output hardware. Corresponding examples to the mid-layer of the audio architecture are XAudio2 (the successor to DirectSound), OpenAL, FMOD and Wwise. This is mostly where game audio programmers are involved.

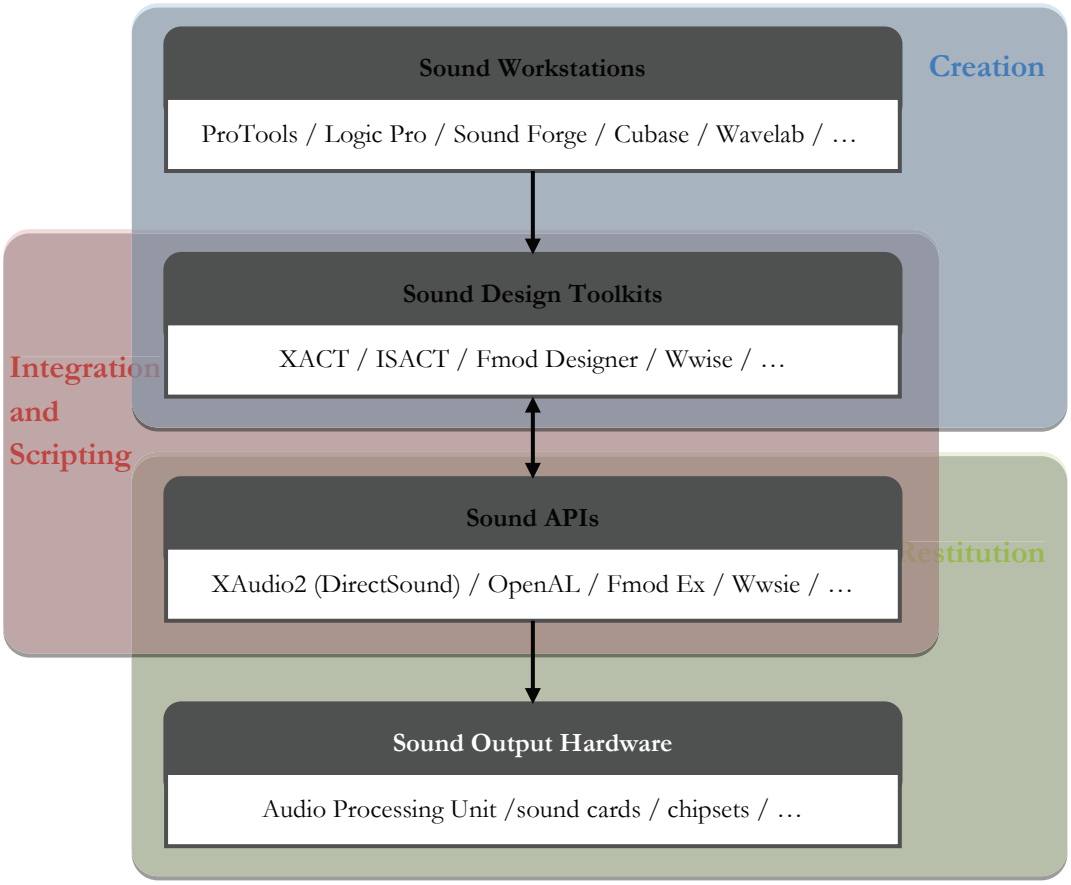


FIGURE 2-11. The three-layer structural of game audio work division.

The borderlines between each layer of the sound engine architecture are sometimes vague since the new generation sound design tools are capable of performing a complex and complete synchronized sound editing. The so-called "Sound Engine" software is basically a real-time mixers and sequencers that can manage most of the mandatory functionalities across the three layers.

2.2 Scene Representations

2.2.1 Scene Graph

A common method to manage components within an audio-graphic scene is to use the scene graph structure, which is a tree-like data organization that illustrates a virtual space itself and the objects located in that. Each node of the scene graph hierarchy may represent an object, properties of the object (i.e., geometry and physical attributes), transformation of a part of the scene (e.g., location, rotation, scale), the procedural descriptions of the object behaviors which can be triggered by events, or the instructions to be followed by the renderer. The scene graph in virtual-world applications, video games for example, describes a logical relationship between objects so that an object can be considered an extension to another. It may also, but not necessarily, arranges the spatial relationship of the entities.

XML, the e**X**tensible **M**arkup **L**anguage, provides a well-defined framework for structure content, making it a trendy choice for the textural demonstration of scene-graph-based virtual environment. There are XML parsers and text editors for nearly every language on every platform, making the documents easily accessible to almost any application.

Many scene description standards, such as VRML/X3D and MPEG-4, rely on the scene graph to hierarchically organize multiple nodes. Several runtime technologies have been developed to exploit the scene graph approach.

2.2.2 Scene Description Languages

Interactive rich multimedia (i.e., a broad range of digital media choreographing audio, video, text, graphics and synthetic animation in real time) presentations are making their entrance into the world and are being increasingly used for newscasts, education material, entertainment [SVS06]. In this context, a scene description language that enables complex synchronization and authoring interactivity for content production is demanded.

A scene description language refers to any description language to describe an audiovisual presentation composited from several medial components. The language can be

interpreted by a rendering program that will generate, either in real time or in batch, a perceptive (animated image and sound) representation of the scene.

Among an assortment of scene description languages for sounds, there are SMIL, VRML/X3D and MPEG-4 BIFS containing descriptions of both visual and audio, and others like SDIF, SpatDIF, ASDF and XML3DAUDIO composing pure-aural scenes. In the following, we elucidate an overall idea of sound capability in each of these formats.

2.2.2.1 SMIL

Synchronized **M**ultimedia **I**ntegration **L**anguage (SMIL) is a **W**orld **W**ide **W**eb **C**onsortium (W3C) recommended XML markup language to write interactive multimedia presentation. Temporal behaviors and spatial properties of media objects are able to be integrated by using such a format. The latest W3C Recommendation for SMIL is the version 3.0 released in 2008.

The SMIL syntax and semantics are allowed to be reused and further joint with other XML-based standards (e.g., SVG, XHTML, VoiceXML, MusicXML, X3D), for the purpose to have 2D graphics, formatted text or timing representation and synchronization. However, this need for several specifications at a time could lead to some interoperability problems. It could be the case if one wants rich content that mixes 2D/3D graphics, text and video [CD02].

One of the design strategies for integrating relevant functionality with other XML-based languages is along with the notions of modularization and profiling. In SMIL, markup functionality is specified as a collection of semantically-related elements, attributes, and attribute-values in the modularization approach. The modules are then combined to form Language Profiles. SMIL 3.0 specifies 12 functional areas (i.e., Animation, Content Control, Layout, Linking, Media Objects, SmilText, Metainformation, Structure, Timing, Time Manipulations, State, and Transitions), which are then partitioned into 64 corresponding modules. The Media Object modules can be applied to include a single generic media object (i.e., ref) element into a presentation by referencing it with a **U**niform **R**esource **I**dentifier (URI). In addition to the ref element, SMIL defines the following collection of synonyms: animation, audio, img, text, textstream, and video. The Layout modules describe the spatial positioning of media objects in a scene with left, width, right, top, height, and bottom attributes. The AudioLayout module also supports the aural volume control. It is possible to assign a multimedia object to a region causing the region to reproduce its audio portion at

the given relative sound intensity via the use of the `soundLevel` attribute. When used in conjunction with SMIL 3.0 Animation (and if supported by the profile), the value of the attribute may be varied over time [SMIL].

2.2.2.1.1 AAML

The traditional SMIL standard defines and manages only two-dimensional render scenes and yet there is only one controllable parameter in the audio object. That is the sound level. Therefore, Pihkala et al. proposed an extension, which is named **A**dvanced **A**udio **M**arkup **L**anguage (AAML), with 3D audio to SMIL [PL03].

In AAML, a third dimension is supplemented to SMIL's 2D coordinates with a similar layout system by adding front, back, and depth attributes. The AAML namespace introduces six types of new elements and their attributes to SMIL. The first is the `audio3d` element playing 3D audio and locating its sound source (i.e., URL of the audio file) by the region or `left` / `top` / `front` (i.e., X / Y / Z coordinates respectively) attributes. Audio is played at a given sound level within `minDistance` (i.e., the minimum distance) whereas it becomes silent outside the range of `maxDistance` (i.e., the maximum distance); between the two distances is attenuation by that the sound loudness is decreased linearly. The `scale`, `rotate`, and `translate` elements are children of `audio3D` that scales, rotates, and translates the 3D respectively. The `listener` element specifies its reference position in the scene. Its `rolloffFactor` is the amount of the attenuation applied to sounds, based on the distance between the sound source and the listener. In addition, Doppler effects are also of use in AAML. The pitch of the sound is affected by the movement velocity of the listener or the sound object itself. Doppler shift can be increased or decreased with the `distanceFactor` and `dopplerFactor` properties. Finally, the audio environment can be modeled with perceptual parameters, specified in the `environment` element. The attributes of the `environment` in AAML are one-to-one mapped to Creative's **E**nvironmental **A**udio **E**xtensions (EAX) 2.0 assets. It has 25 preset environments that can be chosen from the `preset` attribute. The rest of the attributes can be used to override individual preset parameters.

2.2.2.2 VRML / X3D

The VRML, standing for **V**irtual **R**eality **M**odeling **L**anguage, is a scene graph programming model defined for representing 3D virtual reality as well as a universal interchange format for integrated 3D graphics and multimedia. The latest release of the format was the version 2.0. In 1997, it became an international standard as ISO/IEC 14772, also referred to as VRML97.

The VRML has been superseded by e**X**tensible **3D** (X3D). However, unlike the VRML that is a text-based, X3D expresses scenes in XML file format. Each X3D application represents an interactive 3D time-based audio-graphic space. The display multimedia contents can be dynamically modified through various mechanisms like scripts and routes.

Sounds in X3D are spatialized, giving auditory cues and basic ambient mood of the scene. X3D introduces two kinds of audio nodes, `Sound` and `AudioClip` that are derived from the X3DSoundNode and X3DSoundSourceNode abstract node types respectively. The `Sound` node describes sounds in an actual presentation whereas `AudioClip` provides sound source for use. Each node has several fields which identify the particular behaviors of the object. The semantic declaration of a node includes the name of the node with a list of its fields. Each field's label, type, name, and default values are specified in that order.

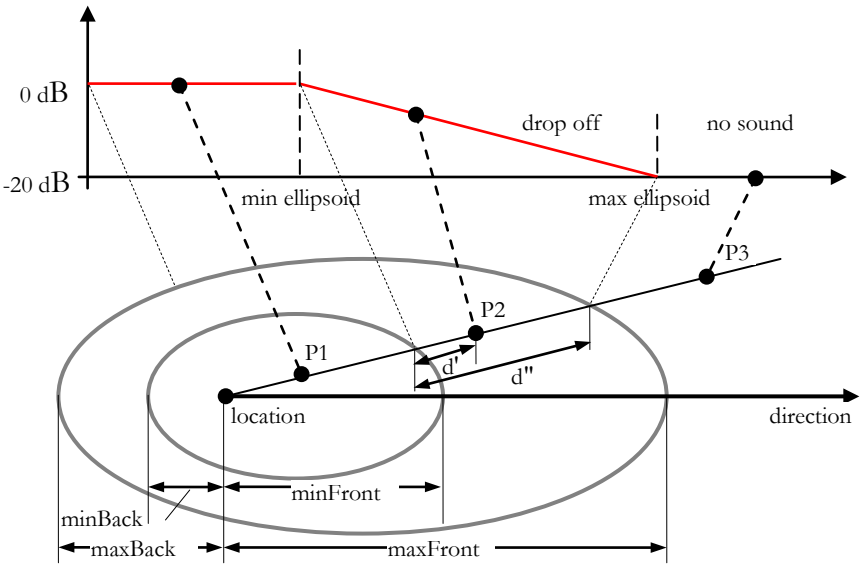


FIGURE 2-12. The ellipsoid sound attenuation model of X3D.

The `sound` node geometry is illustrated in FIGURE 2-12, where the sound is emitted in an elliptical pattern formed by two nested ellipsoids. The loudness attenuation can be obtained from the below equation.

The Equation of Sound Loudness Attenuation:

$$\text{attenuation} = -20 \times (d' / d'') \quad \text{dB} \tag{2-1}$$

, where d' is the distance along the location-to-listener vector, measured from the transformed minimum ellipsoid boundary to the virtual listener, and d'' is the distance along the location-to-listener vector from the transformed minimum ellipsoid boundary to the transformed maximum ellipsoid boundary [P07a].

Topol et al. proposed an enhancement of sound description with a sound-cone definition to "shape" sound sources as well as a room-effect definition for more realistic atmosphere [TS01].

2.2.2.3 MPEG-4

MPEG-4 included object-based audio-video coding for web streaming media and CD distribution, voice and television broadcasting applications, but also digital data storage. Among the existing scene description languages, MPEG-4 provides probably the most advanced scene language considering the sound aspect [S98][SVH99][SYE00]. MPEG-4 consists of several "parts" (standards), among which Part 3 is termed "MPEG-4 Audio" consisting a set of compression technologies for perceptual coding of audio signals. The audio profiles of MPEG-4 includes the Speech, Synthesis, Scalable, High Quality Audio, Low Delay Audio, Natural Audio, Mobile Audio Internetworking, as well as the Main Profile, which is a rich superset of all the other profiles, also containing coding tools supporting natural and synthetic audio. MPEG-4 Audio facilitates a wide variety of applications which could range from intelligible speech to high quality multichannel audio, and from natural sounds to synthesized sounds.

2.2.2.3.1 BIFS

The **BI**nary **F**ormat for **S**cenes (BIFS) is comprised of both 3D and 2D objects interlinked to form the scene graph of a MPEG-4 world. For the reason that it has a great portion of the structure inherited from VRML, BIFS is almost equivalent to the VRML scene graph.

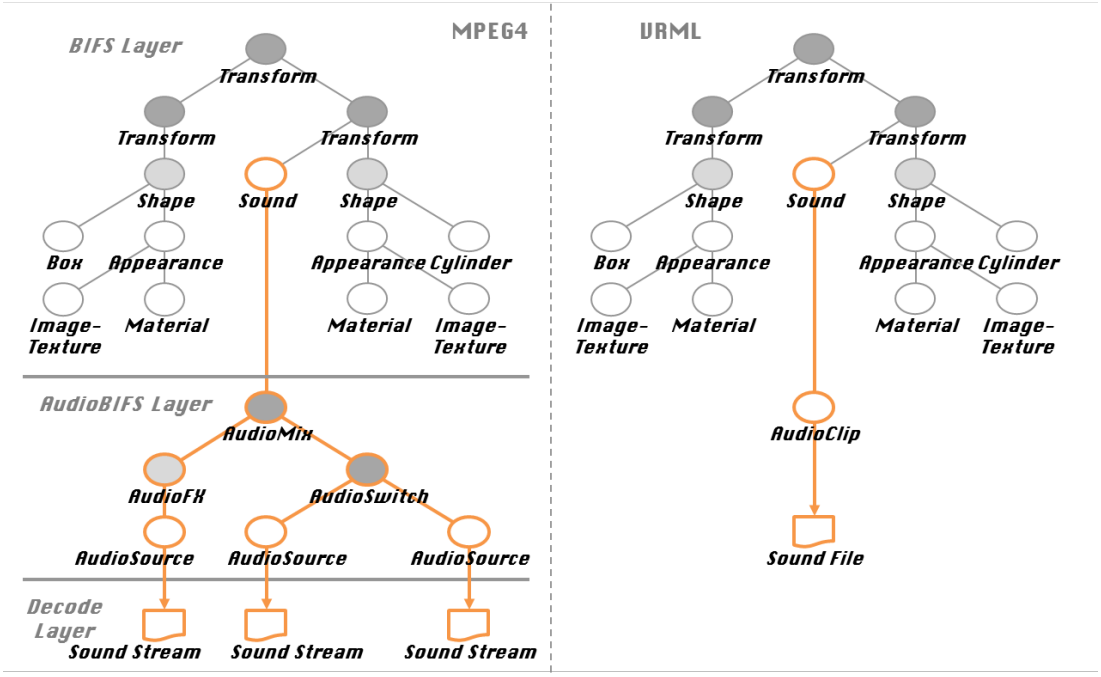


FIGURE 2-13. MPEG-4 and VRML scene graphs with sound nodes [TS01].

In addition to the functions originated from VRML, BIFS has added up distinguishing features so that it can be streamed and can evolve over time if the MPEG-4 server sends modification to the client. This update mechanism consists of two toolsets: BIFS-Command and BIFS-Anim. First, the BIFS-Command allows the manipulations (modification, addition, deletion, replacement) of either a single 2D/3D object or of the entire scene. The second tool, BIFS-Anim, continuously sends an updating stream in a sequence in order to achieve fast animation of components. The updating feature shows how BIFS is stream-dependant; thereby a communication is a request between the bitstreams generators/servers and the MPEG-4 terminals/clients. It requires a heavy on-the-fly calculation of transmission in response to user actions, scene evolution and the renderer. For offline interactive or

automated contents, on the other hand, MPEG-4 has exactly the same paradigm as VRML: the use of ROUTEs to chain behaviors for nodes to others.

BIFS is a binary compressed format; whilst XMT, e**X**tensible **M**PEG-4 **T**extual Format, is an XML-based framework for storing MPEG-4 data. XMT has two levels of textural syntax and semantics. The first variant is XMT-A (XMT-α) providing a deterministic one-to-one mapping to BIFS; it is extended from X3D with X3D-like representations of MPEG-4 specific features such as **O**bject **D**escriptor (OD), BIFS update commands and 2D composition. Secondly, the high-level abstraction of MPEG-4 features based on the W3C SMIL is named XMT-O (XMT-**Ω**), which allows for rapidly creating audiovisual components and specifying their temporal interactions. There is also a standard mapping from XMT-O to XMT-A. The XMT facilitates interoperability with both the Web3D X3D and the W3C SMIL, and provides an exchangeable format between content authors whilst preserving the author's intensions in a high-level textural format [KWC00]. However, it is not a presentation language on its own; it must always be converted to a binary format before it can be transmitted or played back. Shao et al. presented a comparative analysis between SMIL and XMT and had proposed a conversion scheme from the SMIL to the BIFS to take advantage of both formats [SVS06].

2.2.2.3.2 AudioBIFS

Although there is no technical distinction between AudioBIFS and the rest of BIFS, AudioBIFS is considered as an audio subgraph of BIFS. In contrast with the visual scene graph that represents the geometric relationship between graphical objects and the background space, an audio scene represents a signal-flow graph describing digital-signal-processing manipulations [SVH99]. The chain of processing goes from bottom to top in the audio subgraph. Each child node outputs the data to one or more parent nodes above. The leaves and intermediate nodes do not play sounds themselves. Only the result in a sound object, in which an audio sub-tree is rooted, is presented to the listener. Sound objects can also be associated with visual objects in the world for further geometry control. The below diagram illustrates the MPEG-4 Audio system, showing the demux/decode, AudioBIFS, and BIFS layers.

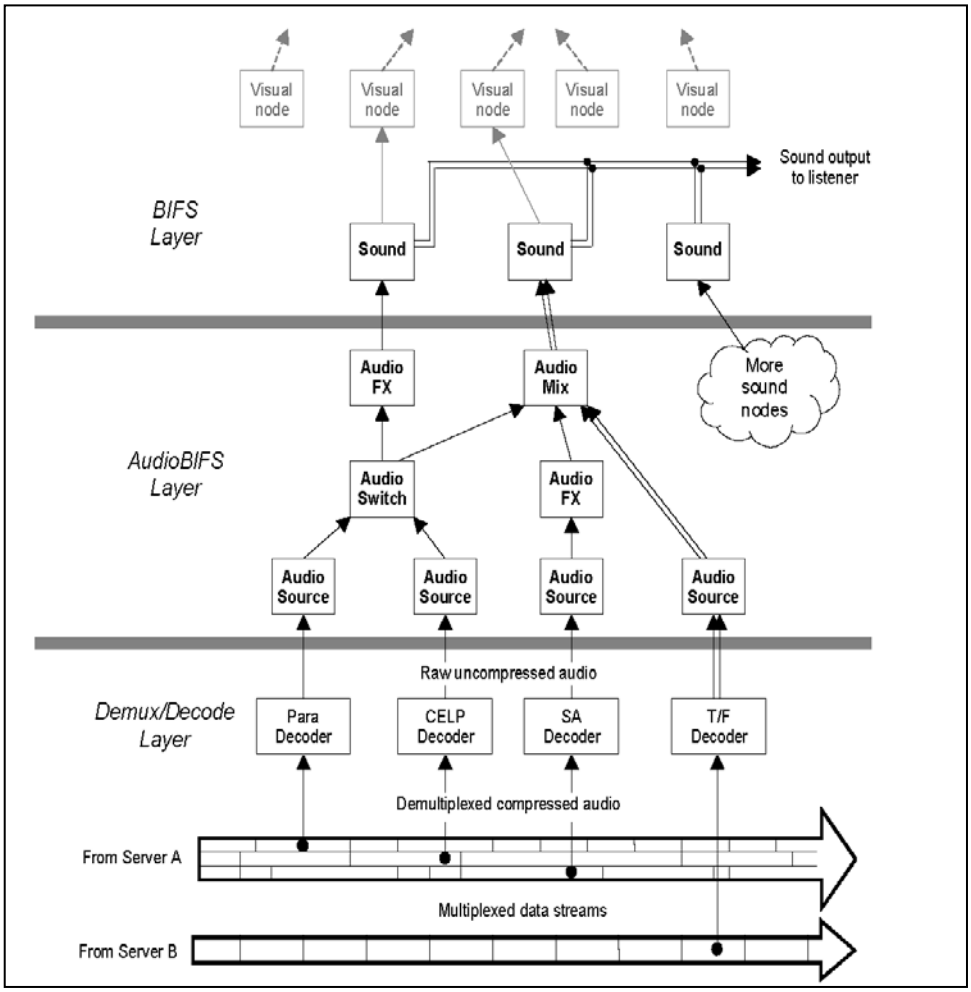


FIGURE 2-14. The MPEG-4 Audio system [S98].

The profiles of MPEG-4 AudioBIFS include Simple 2D Profile, Complex 2D Profile, Audio Profile, and Complete Profile, each of which demands implementation of AudioBIFS nodes. The Complete Profile includes all 2D and 3D visual and audio capabilities of the standard.

AudioBIFS in MPEG-4 Version 1 was standardized in 1999. In this version, eight basic audio-related nodes, include `AudioBuffer`, `AudioDelay`, `AudioFX`, `AudioMix`, `AudioSource`, `AudioSwitch`, `ListeningPoint`, `Sound` and `Sound2D`, were introduced; two nodes from VRML, `AudioClip` and `Sound`, were reused.

Later in 2000, the Version 2 of AudioBIFS (also known as **A**dvanced **A**udio**BIFS** or AABIFS), four new nodes - `AcousticMaterial`, `AcousticScene`, `DirectiveSound`, and `PerceptualParameters`, were proposed for advanced auralization capabilities of audiovisual scenes. The design concept is to create a more natural virtual-reality auditory room that takes into account environmental acoustic effects processing: sound, propagation, reflection, reverberation, attenuation, air absorption, Doppler effect, etc. Two distinct algorithms are presented to approach audio spatialization. The first technique is physical modeling of acoustic environments. That is, to simulate the behaviors of sound in the geometric space. The second technique is perceptual modeling of acoustic environment, in which spatial sound perception is investigated.

In 2004, the most recent update of AudioBIFS Version 3 was unveiled, in which numerous advanced features were added. Among these were `AdvancedAudioBuffer`, `AudioChannelConfig`, `AudioFXProto`, `SurroundingSound`, `Transform3DAudio`, and `WideSound`.

2.2.2.4 SDIF

The **S**ound **D**escription **I**nterchange **F**ormat (SDIF) standard has been created in collaboration by Ircam-Centre Pompidou, Paris, France, CNMAT, University of Berkeley, USA, and the Music Technology Group of the Universitat Pompeu Fabra (MTG-UPF), Barcelona, Spain. It is a widely used standard in the computer-music community for its well-defined and extensible interchange of a variety of sound descriptions including representations of the signal for analysis-synthesis like spectral, sinusoidal, time-domain, or higher-level models, sound descriptors like loudness or fundamental frequency, markers, labels, and statistical models [SDIF].

SDIF consists of a basic data format framework and an extensible set of standard sound descriptions. A simple SDIF file is constructed based on streams, frames and matrices. Each matrix comprises a header with the Signature, Data Type, and Row Count and Colum Count. The matrix content representation can be selected from floating point, integer or text. Moreover, the matrices are written row by row in the file, padded to 4-byte boundaries for the alignment. Then for each frame, the data is stored as one or grouped matrices of arbitrary size and type. Each frame starts with a header containing the means of the Frame Signature that identifies the type of frame data, the Matrix Count that is the number of matrices it contains, the StreamID that is an integer indicating the stream it belongs to, and

lastly the Time tag in seconds relative to the start of the file. The time tag must always be more or equal to the previous frame so that the sequence of frames is sorted in increasing temporal order [BCP08]. The SDIF in conclusion represents all sound descriptions as streams of frames over time.

SDIF-SRL, the SDIF Stream Relationships Language, is an XML-based format for describing the relationships among streams in an SDIF entity (either an SDIF file or SDIF data streamed over a network) [WCF00].

2.2.2.5 SpatDIF

SpatDIF, the **Spat**ial Sound **D**escription **I**nterchange **F**ormat, is an ongoing collaborative cross-project managed by Peters et al. [SP11b][SPATDIF]. A number of researchers and research centers are involved. The general idea is to create a method (semantic and syntactic) as well as best-practice implementations for storing and transmitting spatial audio scene descriptions across real-time and non-real-time audio applications. For controlling spatialization in real-time, SpatDIF uses the **O**pen **S**ound **C**ontrol protocol (OSC). With respect to the initial settings, the SpatDIF Interpreter translates the OSC-messages into low-level commands, tailored to the connected spatial sound renderer.

The structure of SpatDIF namespace is a tired model in two layers. The basic layer contains the minimal information of the audio scene rendering. The core entities and descriptors within this layer must be understood by every SpatDIF compliant renderer. Secondly, the advanced layer contains extensions that provide additional descriptions organized by their functionalities that include transformation instructions or authoring process information (Loop, Interpolation, Trajectory, Geo-Transform and Group Extensions), acoustic modeling information (Distance-Cues, Directivity, Reverb and Doppler Extensions), and presentation setup information (Binaural and Ambisonics Extensions). In addition, a SpatDIF representation consists of two sections - a Meta Section and a Scene Section. The Meta Section serves as information storage zone with descriptions holding general annotation (i.e., comment) and documentation (i.e., author, host, date, session and location) information, as well as the setup of a list of the extensions that will be used within the scene, and the ordering information of how the following Scene Sections will be organized. On the other hand, the Scene Section contains the time-based audio scene rendering instructions in form of entities and their descriptions. The proposed format aims to meet the following goals: Platform Independence, Easily Understandable Syntax,

Extensibility, Free and Open Source, Easy to Connect and also Use of Existing Standards. The development is intended to focus on conceptual rather than technical details. Therefore, existing scene description formats (e.g., SDIF, XML) are proposed to be utilized as data storage solutions for SpatDIF.

2.2.2.6 ASDF

Audio **S**cene **D**escription **F**ormat (ASDF), based on XML, is introduced from the sound recording and reproduction point of view [GS08][GAS10]. This object-based representation of an audio scene is intended to avoid the drawbacks of channel-based storage and to give more efficiency and flexibility for modern high-resolution audio reproduction methods, such as **H**igh-**O**rder **A**mbisonics (HOA) and **W**ave **F**ield **S**ynthesis (WFS). The ASDF is a pure audio scene description without any graphical content. It consists of both static and dynamic feature in the audio scenes for the purpose of real-time user interactivities and artistic performances. The format is still under development in parallel with the **S**ound**S**cape **R**enderer (SSR), which is a versatile software framework for real-time spatial audio reproduction, by Quality and Usability Lab of Deutsche Telekom Laboratories, Technische Universität Berlin, Germany.

2.2.2.7 XML3DAUDIO

XML3DAUDIO, as its name suggests, is a **XML**-based object-oriented scheme for describing and rendering animated **3D AUDIO** scenes where each object has its own set of parameters. It encompasses elementary audio description features (e.g., directivity of sound sources, reflectivity of acoustic material, definition of room reverberation) found in MPEG-4 AABIFS but aims to provide a simpler use than the AABIFS.

Rather than using a traditional scene graph model, the XML3DAUDIO suggests a Scene Orchestra and Scene Score approach to separate the description of scene temporal behaviors from the scene contents. The score defines sequential events and the changes of object parameters. Two parts of the scene score, the initialization score and the performance score, are composed of lines of score. Each line of score gives a single command which is the action being taken on one or more objects. The initialization score is used to set the initial states (e.g., the attributes and parameters) of the scene before it being rendered. Whereas the

performance score holds the scheduling of the scene, the parameters changes as well as the structural relationships that are defined during the course of the scene.

XML3DAUDIO describes complete 3D sound scenes by three categories of descriptors: Scene Content, Scene Timing and Scene Hierarchy. The scene content is defined by a set of elementary objects that are the basic components of the 3D sound scene scheme. The elementary objects include the sound sources, the listener, the virtual surfaces that reflect, absorb and obstruct the sound field, the room reverberations, the medium that defines propagation properties and the 3D audio recordings to create hybrid scenes. The description of the scene content is performed in the scene orchestra. The scene timing descriptors are then performed in the scene score describing the scene temporal behaviors. Lastly, the scene hierarchy descriptors explain the hierarchical relationships between objects. This is to address the fact that complex sound objects are usually made up of several individual elementary objects. In addition to the macro-objects, the acoustic environments can also be defined by the hierarchy descriptor. Only the objects present within the same acoustic space are able to interact with one another [PB02][P06].

2.3 COLLADA

2.3.1 Overview

In this section, we take an overview of COLLADA Digital Asset Exchange schema. COLLADA [AB06][BF08a][BF08b] defines an open standard XML schema from which digital contents of assets can be easily retrieved. COLLADA documents are XML files, usually identified with a .dae (**d**igital **a**sset **e**xchange) filename extension. It was originally created by Sony Computer Entertainment and has become the property of the Khronos standards group.

COLLADA is an intermediate language for transporting data among various interactive 3D applications. It is also a neutral interchange format between several authoring tools, such as geographic information system applications (e.g., Google Earth, ArcGIS), asset repository systems (e.g., 3DVIA, Google Warehouse), web 3D engines (e.g., Papervision3D, Google O3D, Bitmanagement), and game engines (e.g., Unity3D, Ogre3D, Torque 3D). COLLADA is not only widely used in desktop applications, but it's also well suited for working with 3D contents on the web for its inherent use of web technologies (i.e., XML syntax and URI

resource identifiers). The target domain of interactive applications of COLLADA is that in the entertainment industry, where the content is mostly three-dimensional and is game related. Other types of applications that require the same type of technologies will indirectly benefit from it.

The visual scene was the only type of scene in the original design of COLLADA. Later, another domain named physics was introduced to bring physics technology to COLLDA. In the most recent version (1.5.0) of COLLADA standard, the kinematics extension is included. The present COLLADA scene can consist of one visual scene and any number of physics and kinematics scenes. The three designs are independent but there is a mapping between the corresponding components. For the lack of sound description, we propose an audio scene to be specified and implemented in a renderer of COLLADA.

2.3.2 Design Considerations

There are several design goals for the COLLADA schema. While defining a formal representation of sounds using COLLADA, we must keep in mind the principle of the COLLADA schema to meet these objectives. Design goals for the COLLADA schema include the following:

- To liberate digital assets from proprietary binary formats into a well-specified, XML-based, royalty-free, open-standard format.
- To provide a standard common language format so that COLLADA assets can be used directly in existing content tool-chains, and to facilitate this integration.
- To be adopted by as many digital-content users as possible.
- To provide an easy integration mechanism that enables all the data to be available through COLLADA.
- To be a basis for common data exchange among 3D applications.
- To be a catalyst for digital-asset schema design among developers and DCC, hardware, and middleware vendors.

Development of the COLLADA schema involves designers and software engineers in a collaborative design activity. The above design goals, made during the inception of the project, are more important thoughts and assumptions from designers' point of view [BF08b].

2.3.3 Intermediate and Interchange Format

In 3D interactive application production, content creation and management have always been difficult, expensive and time consuming. For this reason, the more content resources can be shared and syndicated, the more efficiently production pipelines can be made available. The rising popularity of COLLADA as an asset exchange and publishing format is to answer this request and trend of content reuse.

COLLADA is designed to be a well-defined intermediate format that could be exported and imported by all the authoring tools. It should be the transport mechanism between numerous tools within the content pipeline. By using this kind of common intermediate representation as a conduit, application developers would not need to write and maintain their own exporters. Instead, they would be able to directly use data suitable in the content tool-chains without learning how to embed utilities inside the primary DCC to facilitate the integration.

COLLADA's objective is to foster the development of a more advanced content pipeline by pursuing better quality content and bringing abundant advanced features to the standard. It encourages the middleware industry to create external tools with particular tasks, such as triangulating, cleaning up the geometry, and remapping texture coordinates. A growing variety of applications are consequently being able to interact with the main DCC tool chosen by the developers. COLLADA ought to play a role as a transport mechanism between the numerous tools in the content pipeline.

The process of transporting digital assets between modeling applications and the runtimes (that is, the "game engines," in the video game industry) has become frequently on account of the growth of 3D games. The growing sophistication of the datasets has further stressed the need for an interchange format. An interchange format aims to enable the data to move freely and losslessly from one tool to another and then to reverse the operation. Developers in the same production can thereby work with a variety of authoring tools. One of the design goals for COLLADA is to support not just the data export from the DCC

tools but also the reimport, although this is limited to the content that can be correctly interpreted by the COLLADA-compatible tools.

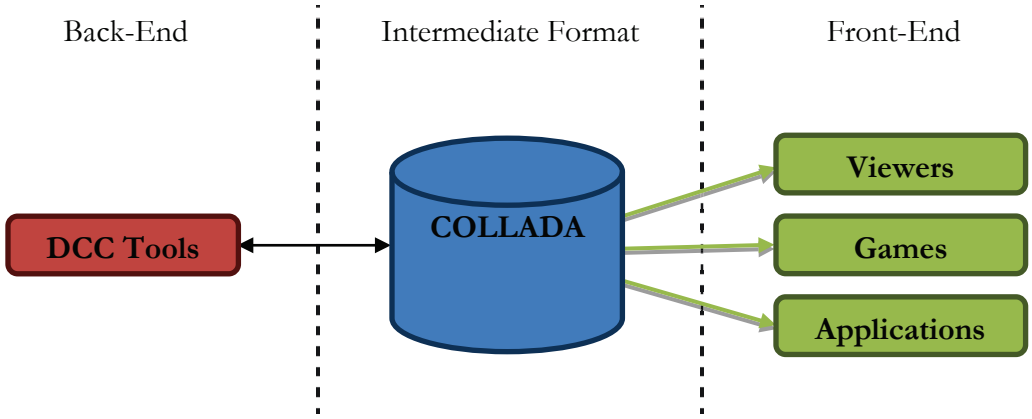


FIGURE 2-15. Intermediate format of choice in the content pipeline.

COLLADA has a mandatory "common" technique storing the data that can be internationally recognized. Ideally, all the "extra" data of a COLLADA document should be kept while going through the process of importing and then exporting from a given tool. This rule is fundamental to enable developers to extend the format in the way they need without having to change the exporter and importer code for all the tools they are using. Storing all the source data in an open standard such as COLLADA would significantly raise the possibility of reusing the assets in the future.

2.3.4 Scenes

The content of an interactive application implies the data that the user can navigate through. The content can have multiple representations stored, partially sharing some of the elements. For example, when the content defines a landscape, different materials are required to characterize the landscape from different views by hours, climates or seasons. The different representations of the data are thus called "scenes" [AB06].

Each COLLADA document can contain one scene. The document and the scene have a one-to-one relationship. This scene represents the instantiation of visual objects, physical

and kinematical simulations that comprise the content created by the artist. The <scene> element may embody a single character, an inanimate rock, or a detailed cityscape used as a level in a video game. The <scene> element is able to instantiate one "visual" scene and one to many "physics" and "kinematics" scenes. It is also able to be extended with the <extra> element.

Specified as follows, the <scene> element has elements that can create instances of specific types:

- the <instance_visual_scene> element that creates an instance of a visual scene;
- the <instance_physics_scene> element that creates an instance of a physics simulation;
- the <instance_kinematics_scene> element that creates an instance of a kinematical simulation.

2.3.4.1 Visual Scene

The visual scene consists of all of the elements that comprise a visual scene graph, including a hierarchy of visible and non-visible objects, attributes, and assets required to render the environment.

The <visual_scene> element acts as the root of the node scene-graph hierarchy. Each <visual_scene> element can optionally have an id attribute and a name attribute for it to be referenced. It can also be extended by <extra> elements. At least one <node> element must be contained in a <visual_scene> element. The node describes the information, such as the cameras, lights, materials, effects and geometries that make up the scene from the visual aspects.

FIGURE 2-16 presents a classic COLLADA sample – the COLLADA Duck. The image on the left displays the XML-based COLLADA Duck document containing visual information; the right presents the 3D COLLADA Duck model rendered from the perspective.

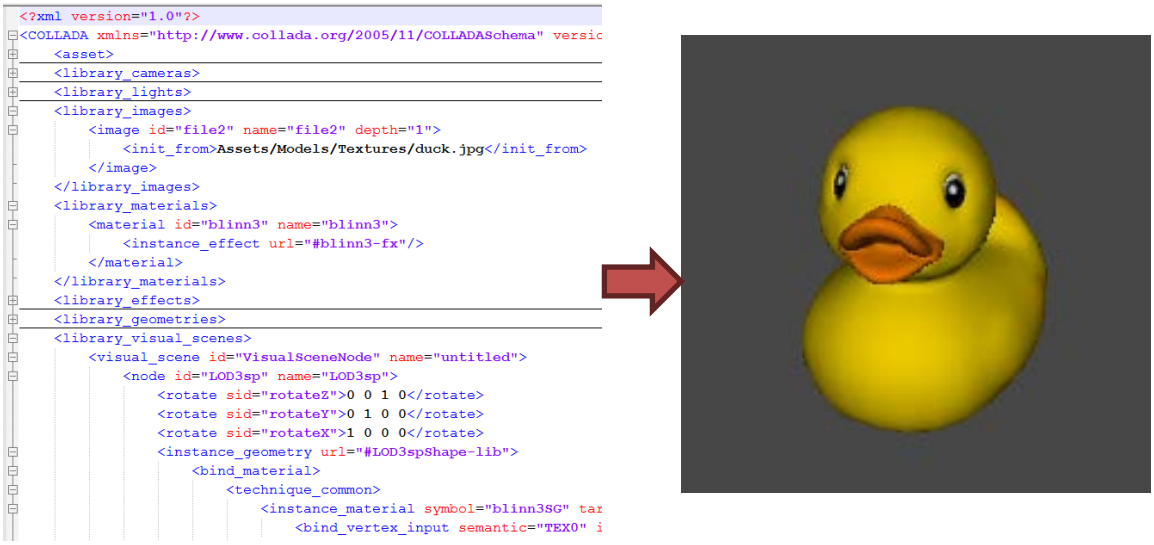


FIGURE 2-16. The COLLADA duck.

(Left: COLLADA document; Right: 3D model rendered in Unity3D.)

Example

The following piece of document shows the designations of layering. In this example, only the nodes ("Node1" and "Node2") that belong to the "city" layer will be rendered.

```
<visual_scene>
   <node id="Node1" layer="street city country">
   <node id="Node2" layer="street city">
   <node id="Node3" layer="country">
   <node id="camera">
      <instance_camera url="#cam"/>
   </node>
   <evaluate_scene>
      <render camera_node="#camera">
         <layer>city</layer>
      </render>
   </evaluate_scene>
</visual_scene>
```

2.3.4.2 Physics Scene

In a 3D application, the data and the processes used for physical simulation of a virtual world are often different than what is used for rendering. The goal of the COLLADA physics design is to provide enough basic constructs required to describe these physics scenes [CV07][BF08b].

Rigid bodies are non-deformable objects defined predominately by geometry shape and physical properties. Their motions can be influenced by Newton's basic laws of physics. Physically based systems usually do not have the perception of scene-graph hierarchy or parent-child relationships used by animated articulated models. Rather, rigid bodies are considered all to be at the same level in a simulation. A constraint links rigid bodies and / or the environment and limits relative motion between the paired rigid bodies. A collection of rigid bodies and constraints are then logically grouped into a physics model to define physical systems. Physics models might be as simple as a single rigid body, or as complex as a character with bones (rigid bodies) that have joints (constraints) connecting them.

There are two approaches to bind the physics and visual scenes:

1. By pointing the parent attribute of an instantiated physics model to the id of a <node> in the visual scene.

This allows a physics model to be instantiated under a specific transform node, which will dictate the initial position and orientation, and could be animated to influence kinematic rigid bodies. Yet by default, the physics model is instantiated under the world, rather than a specific transform node [BF08b].

Example

```
<library_physics_scene>
   <physics_scene id="DefaultPhysicsScene">
      <instance_physics_model url="#pModel" parent="rootNode"/>
   </physics_scene>
</library_physics_scene>

<library_visual_scene>
   <visual_scene id="DefaultVisualScene">
      <node id="rootNode">
         <translate/>
```

```
            <rotate/>
         </node>
      </visual_scene>
   </library_visual_scene>
```

2. By pointing the target attribute of the instantiated rigid body of a physics model to the id of a <node> in the visual scene.

The simulated models in a physics scene are visualized by having their instantiated rigid bodies directly control the placement of nodes in the visual scene. Specifically, at each rendering frame, the location and orientation of each rigid body in the physics scene are applied to the corresponding visible object.

Example

```
<library_physics_scene>
   <physics_scene id="DefaultPhysicsScene">
      <instance_physics_model url="#pModel" parent="#rootNode">
        <instance_rigid_body body="./car/wheel" target="#wheelNode">
           <technique_common>
              <mass>10</mass>
           </technique_common>
        </instance_rigid_body>
      </instance_physics_model>
   </physics_scene>
</library_physics_scene>

<library_visual_scene>
   <visual_scene id="DefaultVisualScene">
      <node id="rootNode">
         <translate>0.3 -1.0 -1.0</translate>
         <rotate/>
         <node id="wheelNode">
            <instance_geometry url="wheelVisualGeometry"/>
         </node>
      </node>
   </visual_scene>
</library_visual_scene>
```

FIGURE 2-17 demonstrates that by targeting the instantiated physics <rigid_body> at a visual <node>, the connection between the two scenes will accordingly be made.

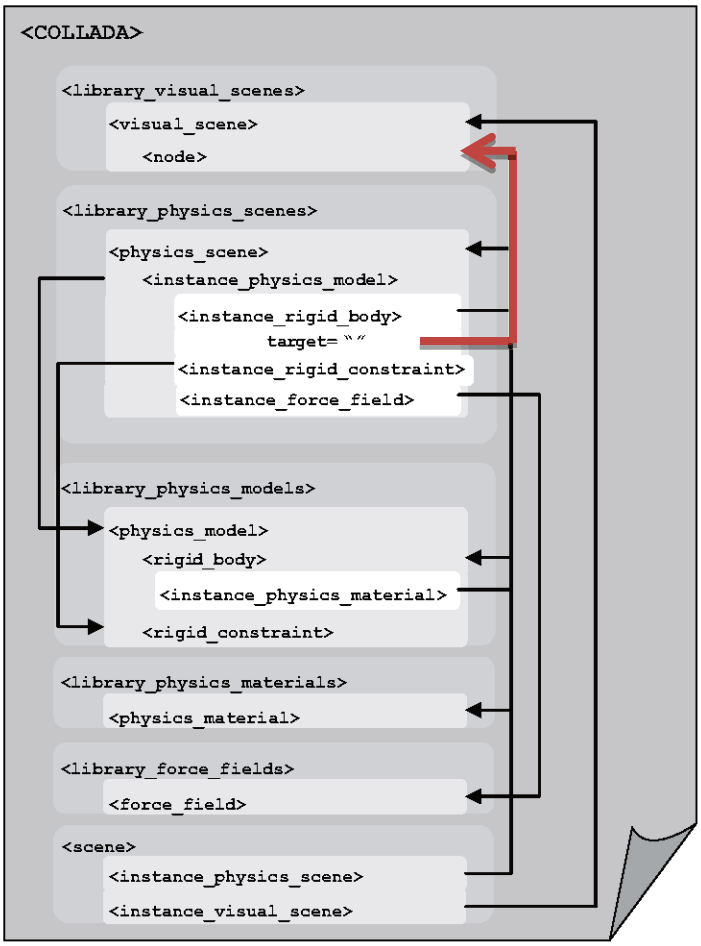


FIGURE 2-17. Relationship of COLLADA elements [AB06].

COLLADA provides a <force_field> as a general way to add forces to the rigid bodies in the scene. The <gravity> is a special case force field that is automatically added to the scene, via the built in physics scene property called gravity.

A rigid body's behavior is completely determined by the physics simulation. Alternatively it can be kinematic, meaning that its position and orientation are controlled by an animation, but it still influences other dynamic bodies in the simulation. COLLADA Physics is capable

of expressing rigid dynamic systems and collision representation. With the physics scene defined, including a set of rigid bodies, constraints, and force fields, the rigid bodies in the physics scene can interact with the physical environment and the other rigid objects in the scene, while obeying the rules of physics.

2.3.4.3 Kinematics Scene

Kinematics descriptions of the COLLADA scene were first presented in COLLADA's latest version 1.5.0. The COLLADA Kinematics is designed to enable to attach kinematical and dynamical properties to objects in a visual scene.

The kinematics portion of COLLADA animation mainly consists of four elements by category: joints, kinematics models, articulated systems and kinematics scenes, described as below.

- Joints

Here allows to define different types of joints, both primitives (`<prismatic>` and `<revolute>`) and compounds, with its axes and limits.

- Kinematics Models

A kinematics model consists of "joints" specified by joint primitives and their arbitrary axes and "links" that are rigid bodies connected through the joints. By one or a chain of kinematic models, nodes in a visual scene can be controlled by a kinematical simulation. Because the kinematics models that are defined in the kinematics scene are completely separate from the geometric appearance, the instance can be bound to elements of an instantiated visual scene [BF08b].

- Articulated Systems

An articulated system defines additional information for a kinematics model. COLLADA kinematics distinguishes articulated systems from two aspects: kinematics and dynamic.

- Kinematics

The <kinematics> element contains additional kinematics information for joints (e.g., locked mode, joint mapping, soft limits, formula) to describe the kinematic behavior of an articulated model and defines the kinematic frames.

- Dynamics

The motion element contains additional dynamics information/limits (e.g., speed, acceleration, deceleration, jerk) to describe kinematic behavior of an articulated model.

- Kinematics Scenes

The kinematics scene defines the current configuration of the scene. While instantiating a kinematics scene in the <scene> element, the links to the visual scene is created.

While the physics part of COLLADA refers to objects that are moved by external forces (e.g., gravity), the kinematics part refers to objects that can move by their own. The definition of the <node> hierarchy by a visual scene allows physics- and kinematics- model instances to reference these nodes.

Similar to the physics scene, there are two methods to bind the kinematics and visual scenes:

1. By pointing the node attribute of a <bind_kinematics_model> to the id of a <node> in the visual scene.

This binds a kinematics model to a node. The description of a kinematics model is completely independent of any visual information, but for calculation the position is important.

Example

```
<scene>
   <instance_visual_scene url="#DefaultVisualScene"/>
   <instance_kinematics_scene url="#DefaultKinematicsScene">
      <bind_kinematics_model node="rootNode">
         <param>scene.kinscene.kinmodel</param>
      </bind_kinematics_model>
   </instance_kinematics_scene>
```

```
</scene>
```

2. By pointing the target attribute of a <bind_joint_axis> to the id of a <node> in the visual scene.

This binds a joint axis of a kinematics model to a single transform of a node. By doing this, a kinematics scene can be possibly synchronized with a visual scene.

Example

```
<instance_kinematics_scene url="#DefaultKinematicsScene">
   <bind_kinematics_model node="rootNode">
      <param>scene.kinscene.kinmodel</param>
   </bind_kinematics_model>
   <bind_joint_axis target="MyNode/joint_0_axis">
      <axis>
         <param>scene.kinscene.kinmodel_joint_axis</param>
      </axis>
      <value>
         <param>scene.kinscene.kinmodel_joint_axis_val</param>
      </value>
   </bind_joint_axis>
</instance_kinematics_scene>
```

2.3.5 Document

2.3.5.1 Basic XML Terminology

COLLADA is an XML-based document type. XML, standing for the e**X**tensible **M**arkup **L**anguage, is well defined by W3C. The well-formed COLLADA framework describes structured content and semantics of documents or datasets, making it a trendy choice for the textural demonstration of scene-graph-based virtual environment. Moreover, the XML

standard has dealt with the issues such as character sets (ASCII, Unicode, Shift_JIS), making any document written in XML instantly universally useful. There are XML parsers and text editors for nearly every language on every platform, to enable the documents easily accessible to almost any application.

A `tag` is a markup construct that begins with "<" and ends with ">." An opening-tag (or start-tag) and a corresponding closing-tag (or end-tag) are referred to as a tag pair. An `element` is a logical document component enclosed by a tag pair, for example, <tagName> content </tagName>. Within the tag pair, the element's content, including its child elements, is specified in texts. In addition to that standard tag pair, there are elements that contain no content information. This kind of empty-element-tag could be represented either by an opening-tag immediately followed by a closing-tag, for instance, <tagName></tagName>, or by a modified-opening tag, which has a slash after the name of the element. For example, <tagName />.

An XML element can have zero or more `attributes` used to specify additional information about this element. An attribute appears within the opening tag of an element and follows the tag name. XML demands that all the XML attributes have a value. That indicates each attribute is a Name-Value pair, with the Name in each pair referred to as the semantic name and the Value surrounded by quotation marks (" ") as the attribute value. For example, <tagName attrName1="attrValue1" attrName2="attrValue2"> content </tagName>.

2.3.5.2 Navigation

The structure of COLLADA documents is hierarchical and properly nested. The document hierarchy creates a parent-child relationship. The parent element contains the nested child element exhibiting aggregation or composition of information. Within a COLLADA instance document, elements can refer to each other by two mechanisms: **U**niform **R**esource **I**dentifier (URI) and **S**coped **ID**entifier (SID).

2.3.5.2.1 URI

The URI addressing refers to the `id` attribute of an element. Many elements that have the `url` and `source` attributes use the URI mechanism to locate instance documents and elements by their `id` attribute values.

The URI internet standard describes syntax for resource names and locations that identify those resources. The **U**niform **R**esource **L**ocator (URL) is the URI syntax that describes the location of a resource. It is commonly used in the **W**orld **W**ide **W**eb (WWW) navigation.

The URI syntax is defined officially by the following hierarchical parts for path separated expressions:

- The scheme part defines the connection protocol such as "http" or "file."
- The authority part defines the server and credentials used to initiate the connection.
- The path part defines the location of the resource on the server.
- The query part started with a question mark ("?") defines additional constraints that identify the resource.
- The fragment part preceded with a hash character ("#") defines a portion of the resource.

When all the parts (disregarding the fragment part) are present, the URL is considered an absolute URL. Otherwise, the URL is considered to be relative on the base URL of the document.

2.3.5.2.2 SID

The SID addressing refers to the `sid` attribute of an element. The `target` and `ref` attributes, `<SIDREF>` and `<SIDREF_array>` elements, and every other attribute or element that contains or references a path to a SID path use this COLLADA-defined scheme to locate and target elements and values within an instance document. The syntax of SID includes the following hierarchical parts:

- The base part defines the unique id attribute of the element that establishes the scope of the navigation. It can also be a dot segment (".") indicating that this is a relative address.
- The path part defines the hierarchical list of identifiers that are unique within the scope of the base identifier. These are optional scoped identifiers (SIDs), each of which is preceded with a literal slash ("/") as a path separator.

- The selector part defines the value member of the element that is addressed.

2.3.5.3 COLLADA Root Element

The <COLLADA> element is the root of the COLLADA document. The first child element of the <COLLADA> is an <asset> that must occur once. The other child elements are optional but must be in a sequence. More precisely, the <COLLADA> element has one <asset> element, zero or more library elements, zero or one <scene> element, zero or more <extra> elements. All of these child elements must occur in the respected order.

The <COLLADA> element requires the <asset> element to provide at least the creation and modification date for this document. The library elements may occur in any order among themselves. Each type of library has a specific name. An optional <scene> element indicates the default scene to be used for the COLLADA document.

2.3.5.4 Libraries and Instances

A COLLADA document is organized as a sequence of several library elements. The various library elements can contain one optional <asset> element and zero or more <extra> elements. Each library can hold only one type of child element that comprehends content in accordance with the type of library. More explicitly, the node named <library_*s> is the library containing a piece of the elements <*>s that can be later instantiated by the <instance_*>. For example, the <node>s, which are defined in the library of nodes: <library_nodes>, can then be instantiated by the <instance_node> in a visual scene. FIGURE 2-17 explains the process of instantiation and relationship of COLLADA elements.

2.3.5.5 Techniques, Profiles, and Extras

COLLADA defines the notion of techniques and profiles. The presence of techniques for a "common" profile is compulsory to make sure COLLADA has a subset that is

universally understood by all COLLADA-compatible tools. The <technique_common> and <profile_COMMON> elements explicitly invoke this profile.

On the other hand, even though the application-specific techniques cannot be interpreted by other tools, it is still written in the same syntax rules to be loadable by other applications.

2.3.5.5.1 Profile

A profile is the first level of COLLADA effect hierarchy. An effect must contain one or more profiles. Each profile is designed for a specific platform, usage scenario, or mixture of APIs. The purpose of profiles is to enable tools or game developers to extend the format by creating a specific section in the content that is clearly marked by a label indicating the nonstandardized part of the content. Profiles represent several sets of functionality for a specific platform. When the runtime content has to create data structures from the source content for a particular target platform, the various profiles can be used as a filter on the COLLADA document. [AB06].

The <profile_*> elements in COLLADA FX encapsulate all platform-specific values and declarations for effects in a particular profile. They define the clear interface between concrete, platform-specific data types and the abstract COLLADA data types used in the rest of the document.

In contrast, the <profile_COMMON> elements in COLLADA FX encapsulate all the values and declarations for a platform-independent fixed-function shader. All platforms are expected to understand <profile_COMMON>. Effects in this profile are designed to be used as the reliable fallback when no other profile is recognized by the current effects runtime.

2.3.5.5.2 Technique

Techniques contain application data and programs, making them assets that can be managed as a unit. The two matters that define the context for a technique are its profile and its parent element in the instance document.

Each <technique_common> specifies source information for the common profile that has to be supported by all COLLADA implementations, whereas <technique> specifies source information for a specific platform or program as designated by the

<technique>'s "profile" attribute. That means the profile attribute of a technique can differentiate it from other alternative techniques in the same scope. Each technique establishes a context for the representation of information that conforms to an associated profile.

All techniques in a specific location represent the same concept, object, or process, but might provide entirely different information for that representation depending on the target application. Techniques generally act as a "switch". If more than one is present for a particular portion of content on import, one or the other is picked, but usually not both. Selection should be based on which profile the importing application can support [BF08b].

In other words, the <technique_common> must be recognized by all the consuming applications; the information it provides is thought of as the reliable fallback by default. If none of the <technique>s specific to the application is appropriate or explicitly chosen, the current runtime must recognize and use the element's <technique_common>.

2.3.5.5.3 Extra

The <extra> element embodies application-specific additional data (i.e., real data or semantic data) related to its parent element, such as OpenGL|ES extensions. Its well-formed XML schema definition enables the <extra> element to contain any type of information, categorized by technique and profile. The extra information can also be managed as an asset.

2.3.5.6 Programmable Units

Currently, profiles in COLLADA FX (effect) manage the combination of fixed-function state, shader assignment, and resource assignment. In the future, more fixed-function pipeline states will be consolidated and replaced by programmable units. [AB06].

In COLLADA FX, shading language code can be located internally or externally. A unique element named <code> is considered as a programmable unit consisting of a fragment of shader code that performs a shading technique (e.g., Phong shading). The <code> element provides an inline block of source code into the <effect> declaration to be used to compile shaders. Inline source code, included as XML schema string data type, must escape all XML identifier characters in case of syntax confusions. On the other hand,

source code or precompiled binary shaders can also be imported into the FX runtime by referencing an external resource in the <include> element.

2.3.6 Level of Detail

Level of detail is a multi-representation management issue, involving increasing or decreasing complexity of a 3D object representation. In scene description, it is usually done by associating additional meta-data with each level for compliant applications to refer to and determine which representation to use. At present, COLLADA has provided several possible mechanisms, including Technique, Scene, Layer, Proxy, and Extra, to describe multiple LODs for visual and physics contents.

2.3.6.1 Technique

There are two types of <technique> elements in COLLADAL. One is the FX <technique> in <profile_*> element for describing one of several methods (LOD versions) for achieving a similar effect. The validation is thought as a best practice of LOD.

Another is the core <technique> in regular elements for declaring the information used to process a particular portion of content on import. The profile attribute of a technique can differentiate it from other alternative techniques in the same scope. Techniques generally act as a "switch". Selection should be based on which profile the importing application can support.

2.3.6.2 Scene

The LOD levels can be defined in separate <visual_scene> or <physics_scene> elements. In this case, each visual or physical scene may represent an individual LOD level. A naming or ordering convention for the scenes is required to distinguish the representations.

2.3.6.3 Layer

Each visual scene may include more than one `<evaluate_scene>` child elements, which declare the information stating how the `<visual_scene>` is to be rendered. The `<evaluate_scene>` element acts as a simple form of command scripting as represented by a sequence of `<render>` child elements. It contains the information needed to process the scene contents for each rendering pass. The `camera_node` attribute refers to a camera as the observer.

Layering and visibility are supported during the evaluation pass. Each `<node>` in the visual scene specifies a `layer` attribute that the node belongs to. Because that `<node>` can belong to multiple rendering layers at once, its `layer` attribute contains not just a single value but a list in which values are separated by whitespace. The `layer` names for each rendering step are matched to the `layer` attribute values of the `<node>`. This mechanism allows the visual scene to select only the matching nodes to be processed through the evaluation pass.

2.3.6.4 Proxy

`Proxy` is an optional attribute for `<instance_node>` elements. The use of this attribute is application defined for LOD. Normally, each `<instance_node>` has an obligatory `url` attribute indicating to a `<node>`, yet applications have an opportunity to resolve the URL in the `proxy` attribute into a `<node>` element. With this option, the application may decide either to use a simple node hierarchy (e.g., a box) or to manage a complex one (e.g., a building) that can recursively contain a nested construct.

2.3.6.5 Extra

COLLADA has the facility to embody arbitrary additional data (real or semantic) by means of `<extra>` elements. An `<extra>` element can contain any type of information, categorized by technique and profile. This provides necessary information for LOD implementation. By defining identified parameters (e.g., type name of LOD) along with proper additional information (e.g., geometry, level, distance) in the `<extra>` tags, selections of LOD representations can be made.

2.3.6.6 Combination

In many situations, LOD capability in COLLADA is done by a combination of the above methods. For example, multiple levels of LOD are defined in <node>s (or their instances) by "layer" or "proxy" attributes, and then more information about how applications decide which path to follow should be stored in the <extra> or <evaluate_scene> elements. Moreover, <technique>s in an <extra> element can be used to characterize the data among different representations and platforms.

2.4 Summary

Development of interactive virtual enlivenment applications, such as video games, is involved with various facets of technologies: audio, visual, physics, AI and so on. Sound in virtual cities is useful, regarding its multifunctional roles and values (environmental, entertaining, emotional, social, cultural…). In addition to visual impacts, sound experience augments the depth of immersion and engagement, delivering more compelling simulated worlds.

Although current technologies (both software and hardware) have enabled audio simulations, a standard format for describing sound in virtual scenes is missing. In the content production pipeline for interactive applications, each brand of development system may have its own different mechanism to master the sound elements in the 3D virtual environment. For instance in some of the game engines, developers can add a game object into the game scene and then attach an audio source component to the game object. Advanced game engines also provide a graphical user interface to assign an audio clip and adjust the parameters for the audio source component. However, these steps have to be repeated object-by-object which is a tedious job and is inefficient for the assessment. Not only that but once switching to another project or game solution, all of the information will be lost, including the link with graphics or physics components. If the application writer intends to reuse the same sound assets, all of these previous works have to be started over again. Some part of the procedure may be facilitated by programming, although in this case, it takes the control over sound designers and demands for extra communications between the programmers and sound designers. Additional information may also need to be specified and documented for references. It turns out that everything would be easier and the process

could be expedited if at the beginning of the application development we have had a scene description standard for representing sounds in virtual environments.

Nonetheless, acoustic utilities in existing scene description languages are conceptually incomplete, out of date, or technically too sophisticated for practical realization. Even though there is a capability to allow dynamic user interactions, it is usually limited to mouse or keyboard controls. Some research ideas, such as sound level of detail (soundscape layering and/or sound-source clustering), have also not yet been covered by scene descriptions. Besides, lots of current standards are mainly designed for purposes (e.g., web contents delivery or digital music composing) other than virtual world interactivities. Therefore, a novel standard format for sound representation in interactive 3D virtual simulations is requested.

COLLADA, an open XML-based language, is chosen not only because it makes it simple to retrieve and organize scene information, but it is also an intermediate plus interchange format that allows digital assets to be written, read, and exchanged among authentic tools. The present COLLADA schema contains visual, physics, and kinematics information but no sound capability yet. Our design of the add-on auditory scene, inspired by the audio scheme in MPEG-4 AudioBIFS, will correspond to the scene graph of the rest of sub-scenes of COLLADA for coherence maintenance. Referred languages like SMIL and X3D have also provided basic and useful information of sound functionality that can be applied to new audio scene description standard in COLLADA.

CHAPTER 3.

Sound in COLLADA

OUTLINE

COLLADA, a XML standard for digital asset exchange, is constructed upon three categories of scenes: visual, physics, and kinematics. As yet there is no auditory information included. For the reason that an ideal scene description standard for describing acoustic phenomena in virtual scenes has not yet been created, we propose an audio capability to COLLADA as a solution of bringing sound in virtual scenes on interactive applications.

In this chapter, we present the scheme of sound capabilities in COLLADA inspired by the MPEG-4 AudioBIFS. In the first, we introduce the concept of audio scene design in COLLADA. Here we discuss the objectives to achieve and policies to follow when designing the sound extensions of COLLADA. The composition of the auditory scene takes into consideration scene graphs and functionalities from both the perspectives of MPEG-4 and current COLLADA. Furthermore, we propose the integration of the city soundscape ambiences with the scene-graph hierarchy of COLLADA, aiming to increase a sense of aural depth. When implementing the sound level of detail, both physical and psychological viewpoints are concerned. It is possible to prioritize particular sound sources while assigning them to soundscape layers or during the process of clustering with the options of semantic filters. The implementation of COLLADA audio scene design is detailed including both static and dynamic descriptions. Finally, we explain how to integrate COLLADA with the game engine in the content pipeline of interactive applications, using COLLADA as an intermediate format to transport digital assets. Moreover, we explicate how the supports of extensive sounds and dynamic scripts may impact on the architecture and workflow of the sound engine.

3.1 Goal and Principle

3.1.1 Overview

In this section, we discuss the design goals and principles for the COLLADA sound descriptions. As we propose to introduce sonic capability into the COLLADA schema, there are some criteria that we consider.

As the same as the MPEG-4 scheme, we suggest that COLLADA Audio must be able to attach sound streams generated by different methods. Sampled files, symbolic sound files, sampled streams or symbolic streams should be available and the choices among which would depend on the type of objects. A pedestrian, for instance, may have his voice in a sample-based file whereas the footstep is a synthesized stream.

Customarily, the sound originates from an URL that leads to a sound asset or a script and will be spatialized by algorithms chosen by the COLLADA player. To leave such an open possibility for rendering the sounds spatialization, some parameters must be given in the description of the scene. As opposed to MPEG-4 which relies heavily on client-server architectures, we intend to minimize communications between stream processors and the sound renderers. To render a spatialized sound with occlusions and reverberation within MPEG-4, the useful parameters must be given in the stream processor but not attached to or described in the MPEG-4 scene directly. Based on the sound renderer selected, these parameters must be computed from some data enclosed in the scene graph to match the visual scene. Hence, a sound stream that does not communicate with either the MPEG-4 client or server which holds the BIFS structure will not be reusable in a different scene. For this main reason, we suggest appending a set of parameters to the COLLADA scene specification so that both physical and psycho-acoustic renderers can produce correct sounds.

Introducing sounds in scene induces an important question about the notion of scene graphs. Traditionally, a graph scene was defined to specify the static part of the scene, in other words, is a typed language but not a procedural one. Yet by nature, sound is dynamic. Indeed the types of dynamics object can be specified, except if we consider that the main goal of such a language is a real-time rendering. The specification of the time and interactive behaviors thus becomes more and more important. Interactions were introduced in VRML and MPEG-4 with simple signals like "mouse click" and the ability to route a signal from a node to another. More complex synchronization scheme was introduced in MPEG-4 based on SMIL (XMT-O) [SVS06]. As a consequence, we propose in COLLADA, complex dynamic behaviors can be either embedded in a block of source code (where the content authors can program features) or point to an external procedure.

Moreover, spatialization problematic will be included in our works as it brings solutions on the subjects of sound level of detail, sound source clustering and reverberation. COLLADA uses several techniques to implement LOD in various usages. The balance must be kept when choosing among approaches to define soundscape layers for different purposes. The task for us is, to find a solution that describes SLOD layers and/or sound-

source clusters by means of the COLLADA schema. To be precise, we need to explain in COLLADA, the sound objects will belong to which soundscape representations in which conditions. Supplementary features are also being conveyed, for example, assigning priorities to sound sources during the process of clustering. No matter if the method we take is based on existing COLLADA techniques or an entirely innovative proposal, it must follow the norm of COLLADA.

According to above mentioned, the COLLADA audio processing flow and functionality are based upon the MPEG-4 AudioBIFS, whereas and the arrangement is established by the current COLLADA syntax rule. Therefore, the composition of the auditory scene must coordinate both the COLLADA schema and AudioBIFS capability. We propose that the sound extension of COLLADA must meet the following guidelines:

- To be compatible in spirit and syntax with the current COLLADA schema.
- To avoid as much as possible specifying data that are already in the scene description.
- To provide the basic acoustic capabilities defined in MPEG-4 and to support advanced developments in sounds that were not included in MPEG-4.
- To facilitate the organization of sounds in layers.

3.1.2 Coherence and Compatibility

The first design criterion is that the sound extension must be compatible in spirit and syntax with the current COLLADA schema. The design strategy for the COLLADA auditory scene follows the same philosophy of visual, physics and kinematics scene graphs in order to maintain the consistency of the overall structure. Moreover, the description has to be well-suited to COLLADA syntax to be straightforwardly recognized by COLLADA-compatible tools in the future. Most important, the rule of sound capabilities must obey the principle of the COLLADA schema to achieve its design objectives (2.3.2 Design Considerations).

3.1.3 Sharing and Reusability

When presenting a novel design into a system, it is encouraging to reuse the existing resources from other techniques in the scope; equally, the notion of the new features is better to be reusable for others. It is a norm to avoid as much as possible specifying data that are already in the scene description. For example, the room geometry should provide some of the data needed to render the acoustic with a physical method. Physical parameters of objects should be used in physical sound synthesis. An object of the visual scene can be associated with an object of the audio scene, sharing parameters like the coordinate systems, the location and orientation or the same geometry. However, this mapping is not mandatory. The auditory scene can rely on perceptive and sound design principles not directly related to the location of the objects in the visual scene. This idea is important for virtual applications like games or art installations.

3.1.4 Configuration and Extensibility

We intend to provide the basic acoustic capabilities defined in MPEG-4 and to support advanced developments in sounds that were not included in MPEG-4, such as clustering of sound sources or real-time synthesis of Foley effects. As a consequence, the following atoms, constructors, libraries and parameters should be either a part of the standard or may be considered as allowed extensions. They must, as well, match with the present COLLADA scene hierarchy.

- Sound Sources

Sound sources can be generated by various methods. Therefore we propose a "sound" node with some parameters like directionality and volume. This node is the equivalent of "light" in the visual scene. We suggest that a sound source may be qualified according to some semantic and staging point of view. This is an instruction given to the render to cluster and localize sources in the spatialization step of the sound pipeline. The file format of a sound node must be specified. For instance, the background of a scene can be a loop on a single surround sound file in Ambisonic B format.

- Acoustic Parameters

Acoustic parameters of an object specifies, either from a physical point of view (acoustic color) or a perceptual perspective (room ambiance), the acoustical characteristics of the object. This should be applied as an acoustic element equivalent to "material" in the visual scene.

- Microphone

Microphone, performed as a "listening-point", is the sound counterpart of "camera" (view-point). It may be mono, stereo or multi-channel (spatialized sound) and has a given directivity. Listening point indicates the spatial position and orientation of the virtual listener in a scene.

- DSPs

- Basic DSP Functions

Basic DSP functions transform a sampled sound file or stream into another one. This is the counterpart of the nodes "effect" and "shaders" in the visual scene.

- Complex DSP Constructors

Constructors of complex DSP are the language used to construct a complex sound node. These include mixers that transform m sound steams into n sound streams, as well as synchronizers, such as clocks and timers, that play, stop, pause, and sequential or parallel playback.

- Synthesizers and Analyzers

Synthesizers transform a virtual sound file or stream into a sampled one; Analyzers transform a sampled sound file or stream into an event, a parameter or a virtual sound stream, as well as time to spectral transformations.

- Sound Rendering Instructions

Instructions for the sound rendering include synthesis, DSP algorithm and parameters, mixing instructions depending on the real acoustic (binaural, transaural, multi-channel 5.1), and what is called in the sequel the audio hierarchy in a particular instruction for a particular type of renderer.

3.1.5 Sound Level of Detail

Considering the large number of audio sources in urban virtual environments, spatialization problematic will be included in our works as it brings solutions regarding sound source clustering and reverberation. COLLADA has offered a number of methods of representing LODs for visual and physics contents. Since the process of SLOD is very similar in spirit to the LOD introduced in computer graphics, we propose to apply the same facility on the audio descriptions in COLLADA.

To organize the auditory scenes based on the integration of audio nodes within COLLADA, we propose to segment the sonic environment of the city into three dynamic layers in relation to, for example, the position of the listener. These layers, based on the soundscape theory, include a real-time synthesized background as well as a midground and a foreground articulated dynamically through a system of SLOD. In other words, the three layers of the soundscape can be considering as three levels of detail for audio. The foreground and midground apply sound to the actors with pre-recorded samples according to a granularity ranging from individuals to groups. The notion of background sound covers the sounds that are not localizable within the listening zone but participating in the overall atmosphere of the city (i.e., background noises, traffic and crowds, and climatic environments). This immersive sound stratum meets the concept of "keynote" as described in soundscape theory.

Seeing that today's interactive virtual environments are usually composed of a significant number of virtual actors presented at the same time in the scene, it is necessary to optimize the amount of sound samples played simultaneously by grouping the sources. The foreground and midground provide SLOD based on the physical conditions of the environment and the semantic description of audio objects. Initially, physical factors determine sets of actors corresponding to sound sources assembled by a traditional approach of "clustering". In the second step, we use the semantic properties of the actors with respect to the application to determine the groupings that will actually be formed. This strategy allows us to vary the priorities of agent clustering in order to adaptively control the mixing and to show or not to show certain elements on the scene depending on the usage scenario. For instance, an explosion and a police siren should be distinctly perceived and localized in an urban security application and cannot be grouped whereas that is not necessarily the case as part of an artistic creation.

As introduced earlier in this thesis, auditory environment design is largely invested in the computer game industry where complex and detailed soundscapes concept is applied for

both realistic environments and fantasy worlds. Video games, for instance *Prototype*, have provided an example of creating a dynamic ambience by dividing it into three tiers of the soundscape where a smooth transition from foreground through midground to background conveys a sense of aural depth of the field. In addition to the SLOD layering approach investigated by *Prototype*, some other sound-source clustering strategies have been described on the basis of geometry and re-synthesis.

3.1.5.1Soundscape Assessment

> Depending on the "acoustic coloration" from the larger environment (geography, climate, wind, water, people, buildings, animals etc.) sound sources create "meanings" to the exposed and block or enable human activities, thoughts, feelings. Therefore, soundscape assessment is engaged in assessing acoustical but also other sensory, aesthetic, geographic, social, psychological and cultural modalities in the context of human activity across space, time, and society.
>
> - B. Schulte-Fortkamp and P. Lercher [FL03]

Elements in different aspects (e.g., acoustical properties, geographical properties, cognitive properties, social relevance) are used to differentiate circumstances of urban ambiences, for instance, the type of architectures (materials, sizes, ages), the type of zones (market places, gardens, coffee shops), the temporality (hours, seasons), the climate (sunny, rainy, windy, snowing), the population density (pedestrians, traffics), and so on. Sound in the urban environment can be comprehended from both spatial and descriptive perspectives.

Soundscape is a useful tool to describe the spatial content of a sound environment, including cities. In addition, it is also a source of information about the current state of the environment [T84]. Soundscape brings clues in regard to the physical nature of the environment and an appreciation of the space surrounding the listener.

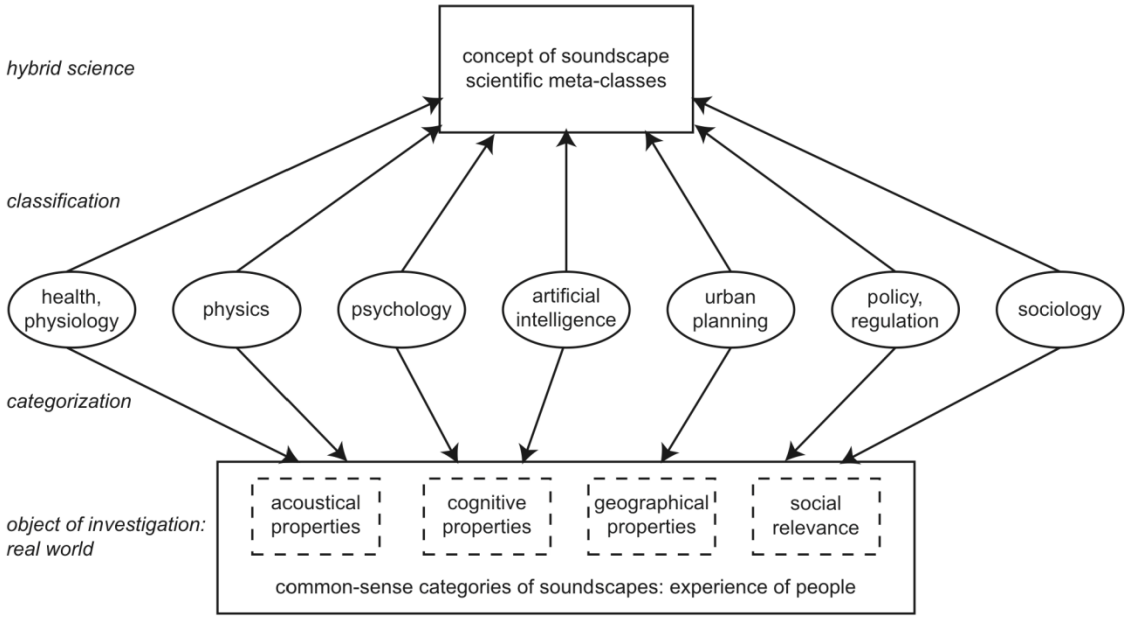


FIGURE 3-1. The organization of Niessen et al.'s hybrid approach to soundscape research [NCD10].

Object identification and categorization based on people's everyday experience in cities have been investigated by several research domains. Soundscape researchers are dedicated to communicate and integrate their study results in shaping the concept of soundscape including both physical descriptions of the sonic environment and the ways (psychological processes) people feel about it. These specific studies allow for understanding how urban soundscape is heard from a qualitative and functional point of view [RD05][NCD10].

3.1.5.2 Sound-Source Prioritizing

> When the subjects and subjects' conceptualizations are the specific objects of investigation, subjects' conceptualizations have to be studied first and the physical descriptions come afterwards as correlated descriptions which do not exhaust the full description of the cognitive object.
>
> – D. Dubois [D03]

The analysis of cognitive representations of acoustic phenomena is conceived as the subjective (i.e., psychological) evaluation of objective measures. Masking thresholds and perceptual criteria (e.g., pitch, loudness) can be dynamically assessed by the psychological system.

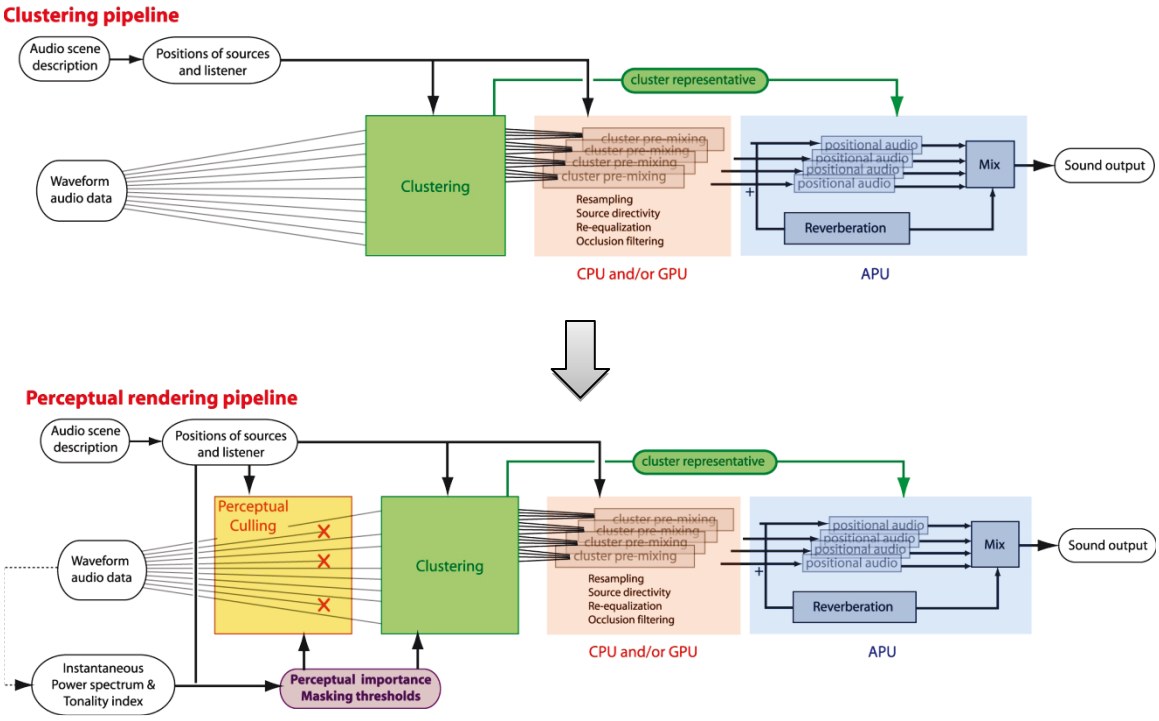


FIGURE 3-2. Audio rendering pipelines with clustering using a combination of APU, CPU and GPU [TGD03].
(Top: a classical audio rendering pipeline with clustering;
Bottom: a perceptually-driven audio rendering pipeline with clustering.)

Given that most of the approaches that include psycho-acoustic knowledge in the audio rendering pipeline have been devoted to speeding up signal processing cost for spatialization purposes, Tsingos et al. proposed a perceptually-driven audio rendering pipeline with clustering, illustrated in bottom of FIGURE 3-2. Compared with the classical clustering, this system concerns subjective judgments of perspective threshold measured along signal characteristics in audio rendering. Perceptual importance helps select better cluster representatives.

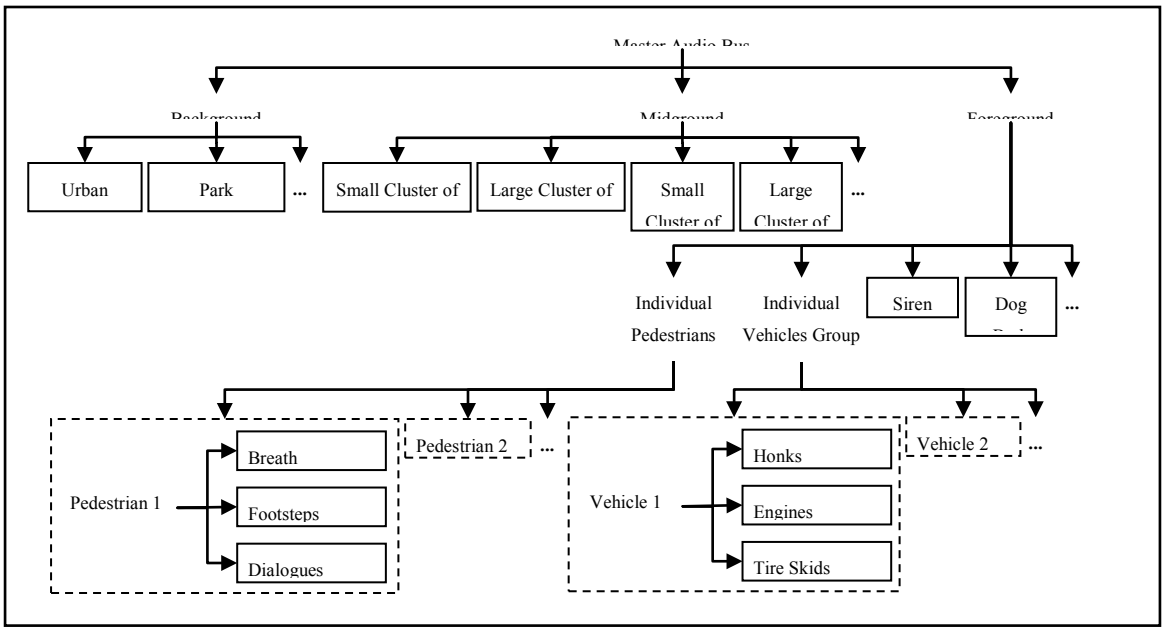


FIGURE 3-3. Example of grouping sound sources.

Generic measurement methods of soundscape characterization are based on geographical facts, such as the distance from the sound sources to the virtual listener or the statistical density of a grouped sound objects within a listening zone. FIGURE 3-3 demonstrates an example of an aural scene in a scene graph presentation organized by geographic relationships among objects. Yet soundscape can be considered as a psychological object, psychological (subjective) assessments, besides perceptual (objective) conceptualizations of physics, should be taken into consideration in classifying soundscape. According to different listening points of interest, it is possible to apply "semantic" filters to spread dynamically individual sources within the soundscape perspective.

In nature, the human perception is able to filter out less desired information (i.e. sending it to the background or midground layer), and to focus on certain sound that we wish to hear more clearly (i.e. presenting it as foreground sound). Take languages for example, linguistic distinctions may be considered as a social/culture aspect of cognitive approach to soundscape. For instance, while traveling aboard where a foreign language is spoken, one can realize immediately when someone is speaking the same language as he does. At the meanwhile, the contents of the conversations that are made in the foreign language can be entirely ignored. The talking noise from crowds can be recognized as midground or even background ambience.

This kind of mechanism works on both visual and aural perspectives and has been frequently utilized in entertainment media. In many video games and movies, sound designers intentionally exaggerate some acoustic characteristics or emphasize some sound effects in a theme to make it sounds even more real than reality itself. For instance, making car acceleration sounds louder in racing games, or making sounds of weapons striking or firing louder in fighting scenes.

Providing a semantic choice, in other words, is offering an option to promote specific objects to a higher-priority position than normal so that we have an opportunity to "focalize" this something particular. That is to say, it is possible to take control of the camera/microphone (point of view/point of listening) and force the audiences to focus on (view/listen to) something particular. For instance, sounds of characters' dialogues, fire alarms, sirens, explosions, or glass window breaking, should have a high priority (i.e., high possibility to be focalized) in a security-oriented application. This is a valuable feature for platforms like *Terra Dynamica*, upon which applications with diverse purposes will be developed.

3.2 Static Scene

3.2.1 Overview

Among the existent scene description languages, MPEG-4 BIFS gives a most well-established auditory scene structure. Inspired by the notion of AudioBIFS, we propose an object-based audio subgraph to the COLLADA scene. In contrast with the visual scene graph that represents the geometric relationship between graphical objects and the background space, an audio scene represents a signal-flow graph describing digital-signal-processing manipulations [SVH99]. The chain of processing goes from bottom to top in the audio subgraph. Each child element outputs the data to one or more parent nodes above. The leaves and intermediate nodes do not play sounds themselves; only the result in an audio object, in which an audio subtree is rooted, is presented to the listener. Audio objects can also be associated with visual nodes.

In addition, we suggest the integration of the soundscape conception into the hierarchical scene with the intention of adding a sense of aural depth. Both objective and subjective evaluations are concerned when implementing the sound level of detail in

COLLADA. It is possible to prioritize particular sound sources while assigning them to soundscape layers or during the process of clustering.

Traditional scene of COLLADA describes its static composition of elements. The basic auditory capability in COLLADA scene is along the lines of the other genres of simulations, representing mostly the static means of objects but not their dynamic activities.

In this section, we introduce the structure of the static part of the COLLADA audio, including a several number of audio nodes and an audio scene comprised of these nodes into a hierarchical order. In accordance with our proposition, COLLADA is divided into four parallel scenes: audio, visual, physics and kinematics. Each sub-scene is independent but the nodes, from one scene to another, can be associated with each other and share properties.

3.2.2 Audio Nodes

The COLLADA auditory scene is constructed by three genres of audio nodes. The first are **Audio Stream** elements that contain sound stream data throughout the signal processing chain. Among these are `<audio_source>`, `<audio_buffer>`, `<audio_delay>`, `<audio_mix>`, and `<audio_switch>`, whose functionalities and semantics are inherited from MPEG-4 AudioBIFS. These elements are intermediate nodes that do not directly send streams to the presenter. The `<sound>` and `<sound_2d>` nodes can also be mapped to those of AudioBIFS, although they are not the roots in the COLLADA audio subtree. What takes place is a node named `<audio_object>` that finishes sound result at the top of each audio subgraph. A novel element, `<sound_ext>` that is particularly designed for external programmed sounds with an extended control of self-defined parameters, is introduced to the sound nodes level.

The second type of audio nodes is **Audio Effect**. Unlike Audio Stream nodes carrying sound data, the Audio Effect elements provide only the control of sound effects that can be applied on Audio Stream elements for delivering DSP effects and/or room reverberations.

The last genre is **Audio Scene** introducing an `<audio_scene>` element that specifies an environment in which acoustic objects are instantiated and simulated. The audio scene embodies the entire set of information that can be auralized from the contents of a COLLADA resource.

The hierarchical structure of COLLADA audio nodes is illustrated in FIGURE 3-4. The elements on the left are the Audio Stream nodes layered in three stages. Auditory data are routed from bottom to top. In the lowest level, there are intermediate nodes and leaves performing mathematical operations on the audio signals they carry; then in the middle, sound nodes served as junctions organizing and passing along the streams and parameters to the root audio object that will present the result to virtual listeners. On the other hand, the Audio Effect nodes, including `<audio_dsp>` that applies DSP effects on individual audio stream nodes and `<acoustic_environment>` that adds perceptual value to the entire audio scene, are designed for "painting" acoustic colors to enhance a sense of persuasive acoustic space.

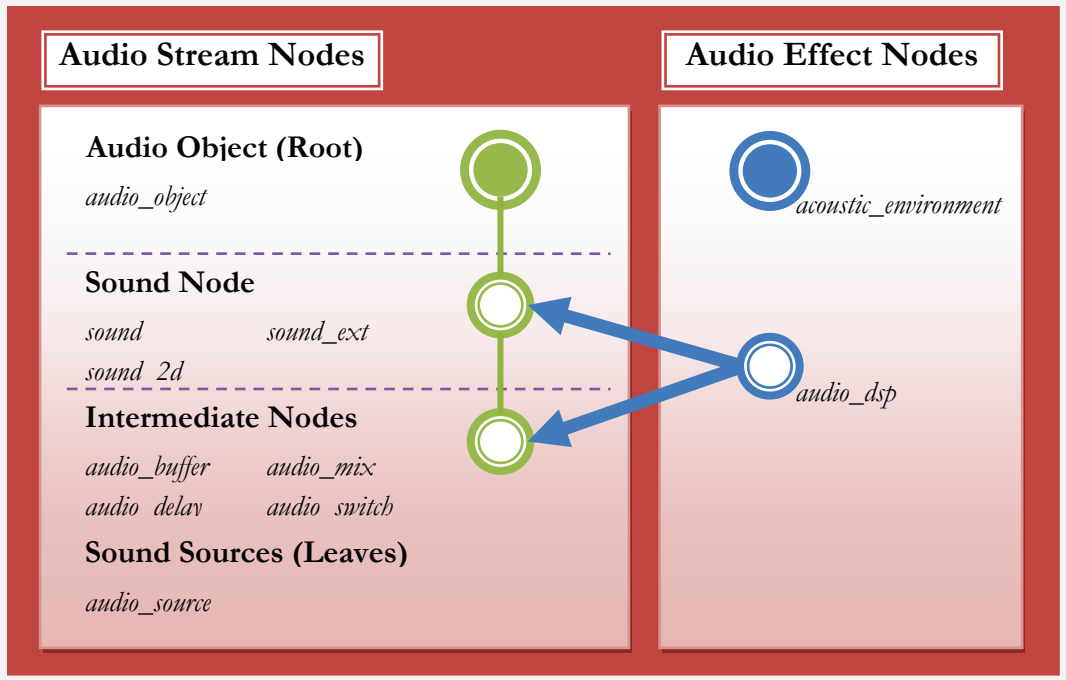


FIGURE 3-4. The hierarchical structure of the audio nodes in COLLADA.

In the following content, we present further information of the functionality of each node and give examples of the implementations. More details of child elements and their usages are listed in APPENDIX B.

3.2.2.1 Audio Stream Nodes

3.2.2.1.1 audio_source

The <audio_source> element, corresponding to AudioSource in MPEG-4 AudioBIFS, adds a sound source into the scene. The current <audio_source> element not only supports the import of common audio file formats, such as WAV PCM, MP3, OGG Vorbis, AAC, or WMA, but other extensive designer-generated sounds, like **FMOD EV**ents (FEV) or **P**ure **D**ata (PD) patches, can also be processed over the usage of the <sound_ext> elements.

Existing COLLADA typed data elements <newparam> and <setparam> are taken to self-define parameters for the extensive sound types. One can use <newparam> to create a new, named parameter of an audio source and assign it a type and an initial value, which can later be reassigned by the <setparam> during the instantiation process.

3.2.2.1.1.1 *instance_audio_source*

The <instance_audio_source> element creates an instance of an object described by an <audio_source> element. Each <instance_audio_source> element has an optional sid and an optional name attributes. Both are a text string that specifies a scoped identifier or a name of this element. The url attribute is required to refer to a local instance or external reference.

Additionally, the <instance_audio_source> can use <setparam> to assign a value to a previously given parameter of the audio source.

An object of an audio source can be instantiated by within any type of Audio Stream node that will be introduced later in this section. This includes intermediate stream nodes (e.g., <audio_buffer>, <audio_delay>, <audio_mix>, <audio_switch>) as well as terminal sound nodes (e.g., <sound>, <sound_2d> and <sound_ext>).

Here is an example of an <instance_audio_source> element that refers to a locally defined <audio_source> element identified by the id "car". The car <audio_source> defines a float parameter "rpm" in <newparam> with a default value 2000.0 which is later modified in 2400.0 by <setparam> within the instance object of the audio source.

Example

```
<library_audio_sources>
   <audio_source id="carFev">
      <init_from>../fev/demo.fev/vehicles/engines/diesel</init_from>
      <newparam sid="rpm">
         <semantic>RPM</semantic>
         <float>2000.0</float>
      </newparam>
   </audio_source>
</library_audio_sources>

<lbrary_sound_exts>
   <sound_ext id="carFevSound">
      <instance_audio_source url="#carFev">
         <setparam ref="rpm">
            <float>2400.0</float>
         </setparam>
      </instance_audio_source>
   </sound_ext>
</lbrary_sound_exts>
```

3.2.2.1.2 audio_buffer

The <audio_buffer> element, corresponding to AudioBuffer in MPEG-4 AudioBIFS, caches the first n seconds specified in the <length> field, of sound from its children for use in interactive playback. It is similar in concept to the VRML node AudioClip, but can be used in broadcast and other one-way streaming media applications.

The child element <num_chan> specifies the number of channels of audio output by this node; and the <phase_group> specifies the phase relations among the various output channels.

3.2.2.1.3 audio_delay

The <audio_delay> element, corresponding to AudioDelay in MPEG-4 AudioBIFS, allows child sounds to be temporally delayed or advanced for synchronization. The length of delay is specified in the <delay> element.

3.2.2.1.4 audio_mix

The <audio_mix> element, corresponding to AudioMix in MPEG-4 AudioBIFS, is used to mix M channels of sounds into N channels through the calculation below.

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} input\ 1 \\ input\ 2 \\ input\ 3 \end{bmatrix} = \begin{bmatrix} output\ 1 \\ output\ 2 \end{bmatrix} \quad (3\text{-}1)$$

, where the value of <matrix> element is [a b c d e f]; <num_inputs> is 3; and <num_chan> is 2. The <num_inputs> field contains the number of input channels to shape the matrix. One is able to control the proportions of the input sounds that are mixed to the output by manipulating the entries specified in the mixing matrix.

3.2.2.1.5 audio_switch

The <audio_switch> element, corresponding to AudioSwitch in MPEG-4 AudioBIFS, selects a subset of the input channels to pass through. In addition to the existing abilities, a new <random> feature is provided to make it possible to select randomly from a number of child nodes (not channels explicitly) to pass through. This is useful, especially when there is an abundance of a certain type of sound, and one doesn't want to choose one source in particular.

In the Example 1, we intend to pick "one" source at random (by setting random field to "1"). Assuming these three sources are all two-channel sounds, if <audio_source> "footstep1" is chosen, the input channels indexed 0 and 1 will be passed through. Giving the value of <which_choice> in "1 1 0 0 0 0" delivers the same result (Example 2).

Example 1

```
<audio_switch>
   <random>1</random>
   ...
   <instance_audio_source url="#footstep1"/>
   <instance_audio_source url="#footstep2"/>
   <instance_audio_source url="#footstep3"/>
</audio_mix>
```

Example 2

```
<audio_switch>
   <random>-1</random>
   <which_choice>1 1 0 0 0 0</which_choice>
   ...
   <instance_audio_source url="#footstep1"/>
   <instance_audio_source url="#footstep2"/>
   <instance_audio_source url="#footstep3"/>
</audio_mix>
```

3.2.2.1.6 sound

The <sound> element, corresponding to Sound in MPEG-4 AudioBIFS, defines sound in a 3D scene. It may also, but not necessarily, be linked with visual nodes in the visual subgraph of the COLLADA scene.

The <sound> node geometry is based on the one defined by VRML/X3D. According to the sound attenuation model in X3D illustrated in FIGURE 2-12, the sound is emitted in an elliptical pattern formed by two nested ellipsoids that are defined by the child elements: <min_back>, <min_front>, <max_back>, and <max_front>. The ellipsoids are oriented by extending the <direction> vector through the sound emitter's <location>. Within the inner ellipsoid, the sound loudness is scaled by the value of the <intensity> and there is no attenuation; between the inner and outer ellipsoids, the sound level is decreased linearly from 0 dB (at the minimum ellipsoid) to -20 dB (at the maximum ellipsoid); outside the outer ellipsoid, no sound is played (Equation

$$\text{attenuation} = -20 \times (d'/d'') \ \text{dB} \qquad (2\text{-}1).$$

The <spatialize> specifies if the audio object is perceived as being directionally positioned relative to the virtual listener. If the <spatialize> element contains the value TRUE, the virtual listener's <direction> and the relative <location> of the <sound> element is taken into account in rendering. The value of the <priority> decides which sounds to be played when there are too many active sound nodes to be played at once because of the limits of the system load. The <priority> value ranges from 0.0 to 1.0 with 1.0 being the highest priority.

3.2.2.1.7 sound_2d

The <sound_2d> element, corresponding to Sound2D in MPEG-4 AudioBIFS, defines sound in a 2D scene. The semantics are identical to those in <sound>, except that <sound_2d> is unavailable in 3D context.

3.2.2.1.8 sound_ext

The <sound_ext> element is an innovation designed to bring extensive sound types into the scope of COLLADA sound schema. In the entertainment production, interactive sounds are typically reproduced via middleware audio toolkits (e.g., FMOD, Wwise) or visual programming languages (e.g., Max/MSP, Pure Data). For this reason, we are dedicated to standardize the description of the design process (i.e., to express it in the fashion of COLLADA scene) with a view to facilitate the communication between the sound content designers and application developers.

Given that the nature of the programmed sound is often complicated beyond the comprehension of the <sound> element, also that the properties of the <sound> element are not always necessarily in use in this case, we coin an element, <sound_ext>, along with an extended control of self-defined parameters. Existing COLLADA typed data objects <newparam> and <setparam> are adoptive for this purpose.

The self-defined parameters can be associated with <newparam> elements inside the <audio_source>s. In each <newparam> element, content authors are allowed to assign the parameter a type and initiate it a value. This value can later be modified through the <setparam> in reference to the corresponding <newparam> or can even be reassigned inside the dynamic script in real-time. The <setparam> appeared within the instance of the extensive sound node may as well enable the retrieval and control of the parameters from its <audio_source> descendants.

Example

```
<library_audio_sources>
   <audio_source id="carWav">
      <init_from>../wav/vehicles/car.wav</init_from>
   </audio_source>
   <audio_source id="carFev">
      <init_from>../fev/demo.fev/vehicles/engines/diesel</init_from>
      <newparam sid="rpm">
```

```
            <semantic>RPM</semantic>
            <float>2000.0</float>
         </newparam>
      </audio_source>
   </library_audio_sources>

   <lbrary_audio_delays>
      <audio_delay id="carDelay">
         <delay>0.5</delay>
         <instance_audio_source url="#carFev"/>
      </audio_delay>
   </lbrary_audio_delays >

   <lbrary_sounds>
      <sound id="carWavSound">
         <instance_audio_source url="#carWav"/>
      </sound>
   <lbrary_sounds>

   <lbrary_sound_exts>
      <sound_ext id="carFevSound">
         <instance_audio_delay url="#carDelay"/>
         <newparam sid="carDelay.carFev.rpm">
            <SIDREF>carDelay/carFev/rpm</SIDREF>
         </newparam>
      </sound_ext>
   </lbrary_sound_exts>
```

3.2.2.1.9 audio_object

The <audio_object> element is the root node in the audio subtree of COLLADA scene. It is the result that will be presented to the listener.

3.2.2.1.9.1 *Profiles*

Thanks to the notion of the <sound_ext>, COLLADA ought to be able to import extensive designer-generated sound formats, including FMOD events and Pure Data patches. However, this capability is limited to specific platforms to which the technique conforms. Therefore, the information of associated profile must be specified. The COLLADA element <profile_*> is employed to implement this idea. As described in 2.3.5.5 Techniques, Profiles, and Extras, the <profile_COMMON> is designed to ensure that the content embraced can be interpreted by all the COLLADA-compatible applications. As opposed to

the common profile, other profiles encapsulate data types specifically conforming to particular platforms or APIs. Extensive sound objects are accordingly declared in this scope.

Example

In the following fragment of COLLADA audio description, two profiles that contain different types of sound sources are provided to the audio object. Applications mean to prioritize profiles based on the API that best matches its platform environment. Even if FMOD is not supported, target platforms must be able to process audio data declared in the common profile.

```
<audio_object id="carAudObj">
   <profile_COMMON>
      <instance_sound url="#carWavSound"/>
   </profile_COMMON>
   <profile_FMOD>
      <instance_sound_ext url="#carFevSound">
         <setparam ref="carDelay.carFev.rpm">
            <float>2400.0</float>
         </setparam>
      </instance_sound_ext>
   </profile_FMOD>
</audio_object>
```

3.2.2.1.9.2 *instance_audio_object*

3.2.2.1.9.2.1 Playback Control

While loading a COLLADA file into a compatible 3D viewer, we expect to instantly see the COLLADA model visualized in the scene. Wherein the audible means, this should not always be the case. In fact, it is very often that an object does not spontaneously make sound but is stimulated by interactions, and the sound level is customarily influenced by the strength of the external force. Therefore, we propose an innovative element in order to enable the playback control of an audio object when instantiating it in the audio scene.

The invention is the `<control>` element, which determines the playback action of an audio object in its initial state. The playback function contained in the `<control>` is an enumeration data type whose value must be one of the following: `PLAY`, `STOP`, or `PAUSE`. If no playback control is specified, by default the object will automatically play the sound defined.

As stated by the dynamic behaviors of sound in nature, dynamical playback control can be realized in further development discussed in 3.3 Dynamic Scene.

Example

```
<!—carAudObj not playing in its initial stage. -->
<instance_audio_object url=”#carAudObj”>
   <control>STOP</control>
</instance_audio_object>

<!—traffic sound plays by default at half loudness. -->
<instance_audio_object url=”#traffic”/>
```

3.2.2.1.9.2.2 Layers

According to the blueprint for the audio scene design described, we intend to integrate the urban soundscape system into the COLLADA audio scene with the intention of enrich a sense of aural depth of the field. Our first attempt is to add a `layer` property to instantiated audio objects in the audio scene.

The `<layer>` for `<node>` elements in the visual scene is a unique attribute which is a list of names indicating which layers that the node belongs to. (See 2.3.4.1 Visual Scene for more details). The analogous idea can be implemented in instantiated audio objects during the simulation to identify the soundscape layers in which the object locates. Layers will simply acts as a selector for soundscape levels in the audio scene. Each instance of audio object can be added to multiple layers by setting a list of whitespace separated layer names, or assigned to the default (non-specific) layer by leaving its layer unspecified.

Example

```
<!—carAudObj in the default layer. -->
<instance_audio_object url=”#carAudObj”/>

<!—traffic in the default, midground and background layers. -->
<instance_audio_object url=”#traffic” layer=”midground background”/>
```

3.2.2.1.9.2.3 Link with Visual Nodes

The instance of audio object has an attribute named `parent` that helps to build the bridge between the audio and the visual scenes. The `<audio_object>` element for the audio scene is mostly equivalent to the `<physics_model>` in the physics scene. So the connection of the audio scene with the visual scene in COLLADA is very similar to that of the physics scene. A `<physics_model>` or an `<audio_object>` can be instantiated under a transform by pointing the `parent` parameter of its instance to a certain visual node; the transforms of a node at which the instance of rigid body or the binding sound targets can also be overwritten. More details are described in 3.2.3 Audio Scene.

Below is an example of binding the audio scene to the visual one.

Example

```
<!—Instantiate carAudObj under the visual transform of rootNode. -->
<instance_audio_object url=”#carAudObj” parent=”#rootNode”>
   <profile_COMMON>
      <!—Binds the audio wheelSound to the visual wheelNode. -->
      <bind_sound url=”#wheelSound” target=”#wheelNode”/>
   </profile_COMMON>
</instance_audio_object>
```

3.2.2.2 Audio Effect Nodes

3.2.2.2.1 audio_dsp

Given that the `AudioFX` node has not been practical in our first implementation of COLLADA auditory scene due to the non-support of MPEG-4 SOAL and SASL, we bring up an alternative function, `<audio_dsp>`, for delivering digital signal processing effects.

The functionality of the `<audio_dsp>` element is similar to that of `AudioFXProto` in MPEG-4. In contrast to `AudioFXProto` or `AudioFX` as one of the relaying nodes in the stream processing flow, `<audio_dsp>` contains no sound data but a set of DSP effects together with parameters. The `<audio_dsp>` element can be appended to any COLLADA audio node defined in above (except the `<audio_object>`). Then the

parent node will be able to apply effects to its output according to the effect specified in the <audio_dsp>.

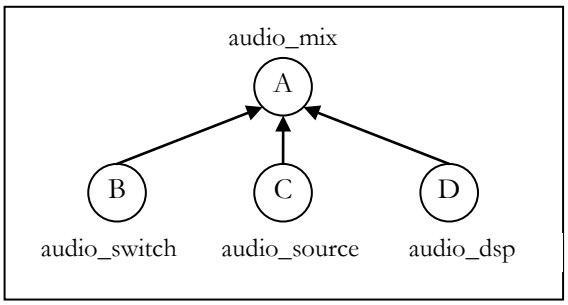


FIGURE 3-5. Example of <audio_dsp> presented in COLLADA.

The <audio_dsp> element in the paradigm above shall be written as follows in the COLLADA syntax:

```
<audio_mix>
   ...
   <instance_audio_switch … />
   <instance_audio_source … />
   <instance_audio_dsp … />
</audio_mix>
```

In this example, A (audio_mix) mixes the inputs from B (audio_switch) and (audio_source), and then adds DSP effects by D (audio_dsp).

As similar to those shaders described in COLLADA <effect> element of COLLADA, the values and the declarations for the various DSPs are typically encapsulated inside the common profile block of the <audio_dsp>. These DSP settings are based on the standard DSP functions used commonly in sound recording and reproduction (See APPENDIX C for more details). Other API-specific profiles may well declare the DSP functions that can be processed using particular DSP algorithm calculated in the corresponding sound APIs, such as FMOD. In each <profile_*>, there are techniques that hold all the necessary information to create an audio effect. Each technique can be used to describe an audio effect in different LOD level or under different game states.

In the following sections, more detailed interfaces and properties of these DSP types are discussed.

3.2.2.2.1.1 *audio_dsp_compressor*

Compressor is a type of amplitude domain effects. Dynamic range compression is the process of compressing dynamic range of an audio signal so that the gain of loud sounds can be reduced and quite sounds can be amplified.

The amount and starting point of gain reduction done to the audio signal is determined by <ratio> and <threshold>. The lower limit of 1:1 is for bypassing [JHB05]; the highest ratio of ∞:1 is considered limiting [K08a]. A limiter is a compressor with a higher <ratio> and usually a fast <attack> time.

The <attack> phase is the time it takes for the compressor to carry its gain reduction to the level determined by the ratio. The <release> phase means the period when the compressor brings the gain back to normal once the signal drops below the threshold.

The <makeup_gain> controls the desired output level with compression active. It is used to recover the gain reduction so that the signal can reach the optimum output level.

3.2.2.2.1.2 *audio_dsp_delay*

Feedback delay allows creating basic time-based delay effects, such as echoes, chorus, etc. The feedback control takes the output from the delay line and feeds it back to the input. The parameter wet/dry <mix> enables to adjust the balance between the loudness of wet (processed) signal and dry (original) signal. A reasonable <feedback> gain and a wet/dry <mix>, cooperative with a moderate <delay_time> can make the sound repeat and become quieter each time it plays back, simulating a reverb effect.

3.2.2.2.1.3 *audio_dsp_equalizer*

A parametric **EQ**ualizer (EQ) is a multi-band variable equalizer that is used to adjust the frequency response of a signal. Three major parameters of tone-shaping are exposed: <center_freq> (f_0), specifying the frequency, at which maximum boost or attenuation occurs; <gain>, determining the amount of boost or attenuation at the center frequency; and <bandwidth> (Δf , of which "Q" is a relation), determining the sharpness of the bandwidth. The Q factor refers the quality of a frequency filter or a damped oscillator. The higher the Q, the narrower and sharper the peak is.

3.2.2.2.1.4 *audio_dsp_filter*

Frequency-based filters cut off a certain range of frequency content in the signal. There are four types of resonating filters provided: `lowpass`, `highpass`, `bandpass` and `bandreject`.

Low-pass filter allow the low frequencies that range from 0 Hz to a -3 dB cutoff frequency (ω_c, `<cutoff_freq>`) to pass through the system. This filter removes the frequency components that are higher than ω_c (high reject) [D00]. High-pass filter let frequencies higher than ω_c pass thorough and blocks lower frequencies (low reject).

Band-pass filters let through a range of frequency. The size of the pass band is determined by the `<bandwidth>`. The frequency components that lie outside the selected range will be cut out by the filter. Band-reject filters, in contrast, prevent a range of frequency from passing through. Only the frequency components outside the band can pass through the system.

Many second-order filters are designed to have a resonance (the "Q"), causing the frequency response around the cutoff frequency to be boosted. The degree of peaking is determined by the `<resonance>` parameter, controlling how much emphasis is given to the area near the cutoff point.

3.2.2.2.1.5 *audio_dsp_reverb*

Reverberation is a frequently used effect for sounds produced in an enclosed or semi-enclosed space where sound is bounced back and forth among the reflecting surface and is absorbed by the medium (e.g., air, walls). A reverb is a collection of these continuously attenuated reflected sound waves (echoes) that reach to the listener's ears so rapidly that they are hardly to be perceived as separate from one to another.

The `<room>` specifies the room effect level; `<hf_room>` (hf-gain) and `<lf_room>` (lf-gain) field control relative room effect high- and low-frequency level.

The `<reflection_level>` (early-reflection) and `<reverb_level>` (late-reverberation) field specify the early reflections and late reverberation volume levels relative to that of the room effect.

The <pre_delay> (reflection-delay) field specifies the delay time of the first reflection; <late_delay> (reverb-delay) specifies the late reverberation delay time relative to the initial reflection.

The <decay_time> field specifies the length of time that room reverberation takes to fade to silence. This feature can be used to simulate the size of the acoustic environment. The <decay_ratio> specifies the high-frequency to low-frequency decay time ratio.

The <hf_reference> and <lf_reference> fields are high- and low-frequency values referenced by other parameters.

Reverberation <density> (modal-density) represents how tightly the reflections in the decay are packed; reverberation <diffusion> (echo-density) specifies the rate at which the density of the reverb increases during the decay.

Example

```
<audio_dsp id="dsp-myComressor">
   <profile_COMMON>
      <technique sid="common">>
         <audio_dsp_compressor>
            <attack>100</attack>
            <ratio>2</ratio>
            <threshold>-20</threshold>
         </audio_dsp_compressor>
      </technique_common>
   </profile_COMMON>
</audio_dsp>
<audio_dsp id="dsp-myCityReverb">
   <profile_COMMON>
      <technique sid="common">>
         <audio_dsp_reverb>
            <room>-1000</room>
            <hf_room>-800</hf_room>
            <lf_room>0</lf_room>
            <reflection_level>-2273</reflection_level>
            <reverb_level>-1691</reverb_level>
            <pre_delay>0.007</pre_delay>
            <late_delay>0.011</late_delay>
            <decay_time>1.49</decay_time>
            <decay_ratio>0.67</decay_ratio>
```

```
            <hf_reference>5000</hf_reference>
            <lf_reference>250</lf_reference>
            <density>100</density>
            <diffusion>50</diffusion>
         </audio_dsp_reverb>
      </technique_common>
   </profile_COMMON>
   <profile_FMOD>
      <technique sid="common">>
         <audio_dsp_reverb>
            <hf_room>-800</hf_room>
            <reflection_level>-2273</reflection_level>
            <reverb_level>-1691</reverb_level>
            <pre_delay>0.007</pre_delay>
            <late_delay>0.011</late_delay>
            <decay_time>1.49</decay_time>
            <decay_ratio>0.67</decay_ratio>
            <diffusion>50</diffusion>
         </audio_dsp_reverb>
      </technique_common>
   </profile_FMOD>
</audio_dsp>
```

3.2.2.2.2 acoustic_environment

The `<acoustic_environment>` element is represented by perceptual parameters. Content authors can use the `<preset>` child element to select from preset environments. The options from the preset list are based on those in Creative Labs' **E**nvironmental **A**udio e**X**tensions (EAX) [PL03] and FMOD's preset reverb properties (originally defined by Creative Labs) [F10b][FMODEX]. The `preset` provides reverberation simulation for a wide range of atmospheres, not merely limited to urban environments. The rest of the child functions enable individual DSP reverb parameters to be overridden to customize the reverb.

Example

```
<library_acoustic_environments>
   <acoustic_environment id="inBathroom">
      <preset>bathroom</preset>
   </acoustic_environment>
```

```
    <acoustic_environment id="inCity">
       <preset>city</preset>
       <decay_time>3.0</decay_time>
       <pre_delay>0.01</pre_delay>
    </acoustic_environment>
</library_acoustic_environments>
```

3.2.2.3 Audio Scene Nodes

3.2.2.3.1 audio_scene

The <audio_scene> element specifies an environment in which sonic objects are instantiated and simulated. The active <audio_scene> is indicated by instantiating them under the main <scene>.

3.2.2.3.1.1 *Semantic Choices*

A COLLADA document can contain various audio scenes, each of which describes a customized scenario comprising one or many audio objects and acoustic environment settings. Each <audio_scene> element is used for specifying information under different game states (e.g., a daytime, nighttime, rainy or sunny) with different versions of soundscape levels. With this feature, audio in COLLADA is capable of describing a wide variety of acoustic circumstances. This shall, as well, provide a semantic choice for different applications in different principles to arrange the priority of the layer represented.

In the following example, ambulance sound effect is usually blended into traffic ambience so there is no independent audio object that represents it. However, this sound signal may deliver urgent messages in a security simulation application. For that reason, we send it to foreground layer, making it stand out to be clearly perceived in this situation.

Example

```
<library_audio_scenes>
   <audio_scene id="DefaultAudioScene">
      <instance_audio_object url="#traffic" layer="midground"/>
   </audio_scene>
```

```
    <audio_scene id="SecuritySimulationAudioScene">
      <instance_audio_object url="#traffic" layer="midground"/>
      <instance_audio_object url="#ambulance" layer="foreground"/>
    </audio_scene>
  </library_audio_scenes>

  <scene>
    <instance_visual_scene url="#DefaultVisualScene"/>
    <instance_audio_scene url="#SecuritySimulationAudioScene"/>
  </scene>
```

3.2.2.3.1.2 *Urban Soundscape Layers*

In order to enrich a sense of aural depth of the field, we integrate the soundscape system into the audio scene. The way we implement it for the audio scene is based on the layering design and evaluation procedure in the COLLADA visual scene cited in 2.3.4.1 Visual Scene.

Like how it is defined in the `<node>` element, the `layer` attribute of `<instance_audio_object>` or `<instance_acoustic_environment>` contains not a single value but a list of names. Matching it with the `layer` name for each rendering pass in the `<evaluate_scene>` can determine the soundscape layer to be evaluated.

An audio scene may instantiate multiple audio objects, but only those that have their `layer` attribute names matched against the `layer` names of the evaluation scene will be processed during the rendering pass. Likewise, a room effect setting for acoustic environment only affects the same-layer objects. The control of which layer to playback can also be adjusted by dynamic scripts in keeping with the game status.

Sounds in different layers may be rendered with different channel setups in accordingly. For example, we use mono for the foreground, stereo for the midground, and surround for the background. By default if no layer is specified, we designate to have all auditory contents be stereo.

In the example below, only footstep and taking sounds that belong to the foreground layer will be passed through the evaluation. Both of them will be played back in mono.

Example

```
  <audio_scene>
```

```
    <instance_audio_object url="#footsteps" layer="foreground"/>
    <instance_audio_object url="#talking" layer="foreground"/>
    <instance_audio_object url="#crowd" layer="midground"/>
    <instance_audio_object url="#traffic" layer="background"/>
    <instance_acoustic_environment url="#city"
                                   layer="foreground midground"/>
    <evaluate_scene>
       <render>
          <layer>foreground</layer>
       </render>
    </evaluate_scene>
 </audio_scene>
```

3.2.3 Audio Scene

3.2.3.1 Scene Graph Hierarchy

The COLLADA audio scene embodies the entire set of information that can be auralized from the contents of a COLLADA resource. The function of the audio scene is inspired from MPEG-4 AudioBIFS. It represents a signal-flow graph, from bottom to top describing digital-signal-processing manipulations by a hierarchical order of audio nodes.

The paradigm below illustrates the hierarchy of the COLLADA scene. The design of the audio scene follows the same philosophy of other visual, physics and kinematics sub-scene graphs to maintain the consistency of the overall structure. Each of these sub-graphs is independent but can be linked with each other and share properties.

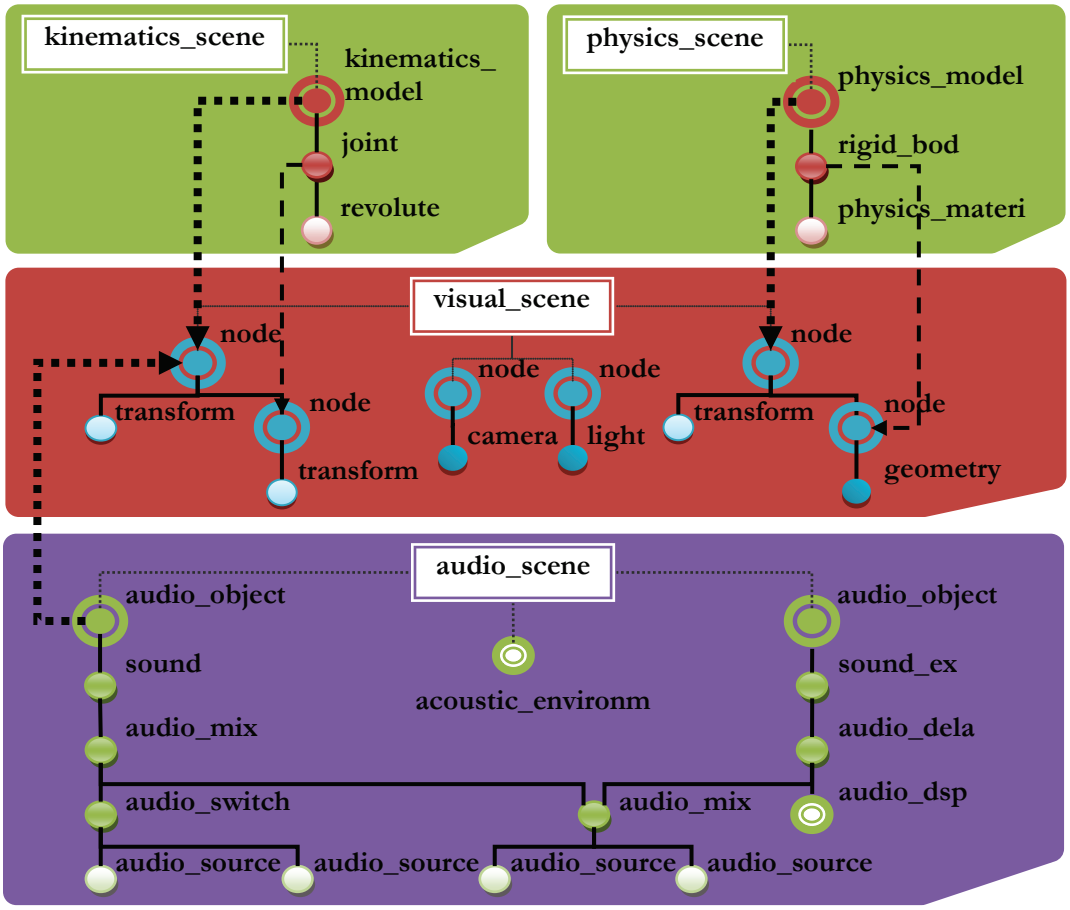


FIGURE 3-6. Paradigm of the scene graph in COLLADA.

3.2.3.2 Link with the Visual Scene

FIGURE 3-7 demonstrates how to link COLLADA elements with each other and how to connect the audio and visual subgraphs. This can be comparable with the diagram in FIGURE 2-17, in which the relationship of COLLADA physics-visual elements is depicted.

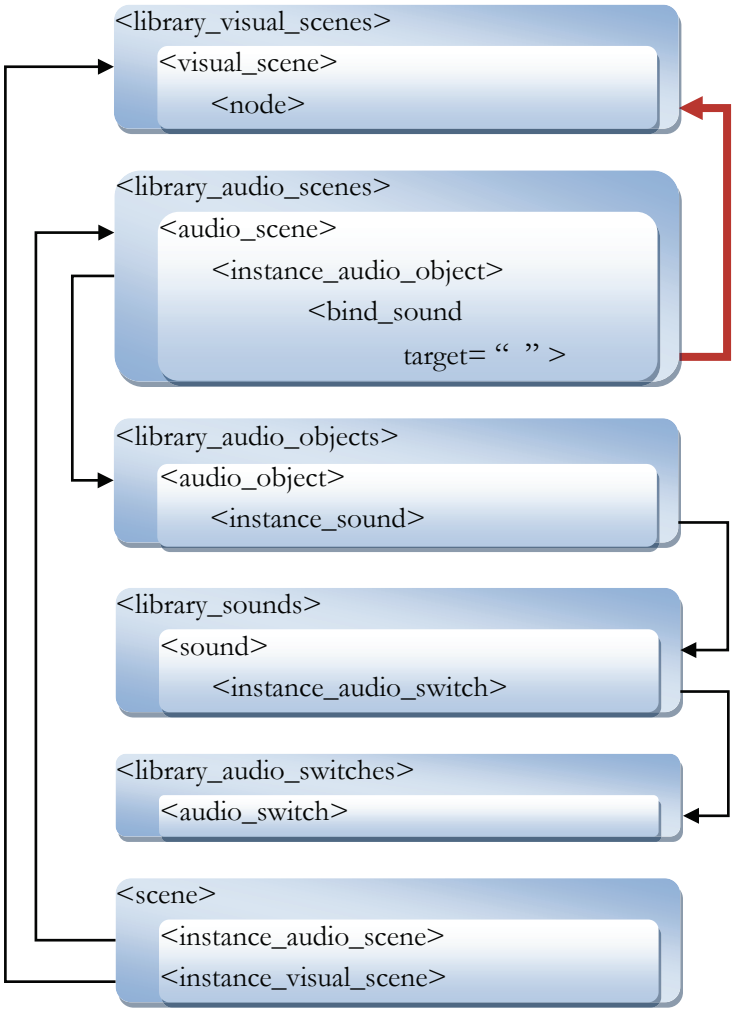


FIGURE 3-7. Relationship of COLLADA audio-visual elements.

Specifically, to keep the consistent design of the audio system in COLLADA, the process of linking the audio scene with the rest of the COLLADA scenes is along the lines of how it works in the physics or kinematics scene. The two approaches to bind the audio with the visual scenes are:

1. By pointing the parent attribute of an instantiated audio object to the id of a <node> in the visual scene.

By doing so, an <audio_object> can be instantiated under a specific transform node (i.e., to dictate the initial position and orientation).

Example

```
<library_audio_scenes>
   <audio_scene id="DefaultAudioScene">
      <instance_audio_object url="#carAudObj" parent="#rootNode"/>
   </audio_scene>
</library_audio_scenes>

<library_visual_scene>
   <visual_scene id="DefaultVisualScene">
      <node id="rootNode">
         <translate/>
         <rotate/>
      </node>
   </visual_scene>
</library_visual_scene>
```

2. By pointing the target attribute of the binding sound of an audio object to the id of a <node> in the visual scene.

Alternatively, the child sound element (i.e. <sound>, sound_2d or <sound_ext>) of <audio_object>/<profile_*> corresponds to the child <rigid_body> node of <physics_model>. During the instantiation of the audio object, a target parameter is given with the bound sound to specify the linkage to the visual representation. It indicates which <node> of the <visual_scene> has its transform influenced by the associated sound.

Example

```
<library_audio_scenes>
   <audio_scene id="DefaultAudioScene">
      <instance_audio_object url="#carAudObj" parent="#rootNode">
         <profile_COMMON>
           <bind_sound url="#wheelSound" target="#wheelNode"/>
         </profile_COMMON>
```

```
        </instance_audio_object>
    </audio_scene>
</library_audio_scenes>

<library_visual_scenes>
    <visual_scene id="DefaultVisualScene">
        <node id="rootNode">
            <translate>0.3 -1.0 -1.0</translate>
            <rotate/>
            <node id="wheelNode">
                <instance_geometry url="#wheelVisualGeometry"/>
            </node>
        </node>
    </visual_scene>
</library_visual_scene>
```

3.2.3.3 Selection of the Soundscape Layers

In each of the audio scenes there are groups of audio objects and acoustic environment settings. Content designers arrange the objects by layers, performing a semantic filter to prioritize sound information to be perceived. That is to say, a COLLADA document can contain more than one audio scene to discriminate against sound in various circumstances. Every single audio scene may respectively represent a tailored theme, specifying information under different game states with different versions of soundscape levels.

In this context, two steps are taken to decide default soundscape layers of the audio scene that will be rendered. Firstly, an audio scene is elected to be instantiated; secondly, one or more soundscape layers are selected by the `<evaluate_scene>` in the audio scene. If no layer is explicitly chosen, all the layers will be rendered.

This mechanism not only follows the standard procedure of analyzing and processing the COLLADA content, but it also adopts our strategy in subjective soundscape layering. In spite of the default scene is all determined inside the COLLADA documents in order to provide applications and tools an indication of the primary scene to load, application writers have the freedom to switch from the default to another possibility and can always modify the details of the presented scene at runtime.

FIGURE 3-8. Two steps to determine default soundscape layers for the representation of COLLADA sound.

3.3 Dynamic Scene

3.3.1 Overview

Complex dynamic behavior is greatly essential to enhance the interactivity of interactive applications. User interaction may directly or indirectly lead to a change of the content. For instance, the virtual representation of the user physically removes the base of a pile of boxes (directly), and then the boxes on top collapse (indirectly).

Every object or piece of world material in a game or film has a sound it would make if you interacted with it [F06]. A sound world is built upon interactivities. As a consequence, we are seeking a scene descriptor with the aim of plotting out the sound behaviors characterized by the dynamics.

Influenced by many different factors, such as the change of time, the location of the listener or user interaction, adaptive audio in virtual cities is able to travel among different perspectives from background via midground to foreground and vice versa. Sound like

motion objects can be transferred from one state into another by an event trigger [C08b]. According to this motive, descriptors for dynamic behavior controls are expected in addition to the static category of audio contents in COLLDA.

COLLADA animation supports dynamic control of visual representation but it is rather linear time-dependent. By tradition, COLLADA provides an extensibility feature termed <extra> for almost all types of COLLADA elements in order to describe arbitrary additional information. In this aspect, information of dynamic performance can be embodied within the scope of the <extra> design. However, because it is difficult to use this kind of classic scene-graph technology to describe such complex dynamic contents management, we propose an alternative solution with a scripting language that enables the description of the dynamic phenomena in the virtual city. This solution is expected to be applied on not only the audio but also the visual, physics and other capabilities in the COLLADA scene.

In this section, we explain how to utilize the extensible schema of COLLADA to deliver dynamic content, and introduce a script language to manage more complicated dynamic nature of interactive applications.

3.3.2 COLLADA Extensibility

The <extra> element represents application-specific additional data (i.e., real data or semantic data) related to its parent element. Its well-formed XML schema definition enables the <extra> element to contain any type of information, categorized by technique and profile. The extra information can also be managed as an asset. In this section, we introduce the extensible technology two new sorts of elements – events and functions, dedicated to help manage the dynamic part of the scene in COLLADA.

3.3.2.1 Events

The status of sound in virtual cities may be influenced by a wide variety of circumstances. It may change according to the environmental factors in nature, such as season, weather or air pressure. Physical impacts in game, like collisions between objects, may also produce sound effects. In addition, the control of sound can be relative to certain gameplay elements,

like timing, as well as the user interactions with input devices, such as the mouse, keyboard or joystick. For this reason, we wanted to define various event elements by genres in order to handle numerous conditions of the scene states.

3.3.2.2 Functions

When an event is triggered, a series of actions in multi-representations may be taken simultaneously in the scene. A `<function>` element is a macro-element to record these complex actions. It typically appears in the instantiated sub-scenes' extra descriptions. The root elements on the top level of the sub-scenes, such as visual nodes or audio objects, can be referenced in functions for dynamical control of transforms, textures, playback, and so on. Each function can then be "called" via the `<call_function>` technique. Once a function is called (usually following an event), the actions declared within this function will be started.

3.3.2.3 Implementation

In this sub-section, we provide two examples to demonstrate how to utilize COLLADA's existing extensibility technique in conjunction with the concept of "event trigger" and "function call" to implement the dynamic evolution.

Example 1

In the first example, when a mouse event – "LEFTCLICK" is triggered, two functions – "ringDoorbellFunc" and "openDoorFunc" will then be called. In other words, when the user click on the visual representation of door, the doorbell will ring (the audio object of the doorbell will play) and the door will open (the visual node of the door will rotate 90 degree on the y-axis).

```
<library_visual_scenes>
   <visual_scene id="DefaultVisualScene">
      <node id="doorNode">
         …
         <extra>
            <technique profile="SHCHAN">
```

```
                <mouse_event type="LEFTCLICK">
                    <call_function url="#ringDoorbellFunc"/>
                    <call_function url="#openDoorFunc"/>
                </mouse_event>
            </technique>
        </extra>
    </node>
</visual_scene>
</library_visual_scenes>

<scene>
    <instance_visual_scene url="#DefaultVisualScene">
        <extra>
            <technique profile="SHCHAN">
                <function id="openDoorFunc">
                    <ref_node ref="doorNode">
                        <rotate sid="rotateZ">0 0 1 0</rotate>
                        <rotate sid="rotateY">0 1 0 90</rotate>
                        <rotate sid="rotateX">1 0 0 0</rotate>
                    </ref_node>
                </function>
            </technique>
        </extra>
    </instance_visual_scene>
    <instance_audio_scene url="#DefaultAudioScene">
        <extra>
            <technique profile="SHCHAN">
                <function id="ringDoorbellFunc">
                    <ref_audio_object ref="#doorbellAudObj">
                        <control>PLAY</control>
                    </ref_audio_object>
                </function>
            </technique>
        </extra>
    </instance_audio_scene>
</scene>
```

Example 2

In the second example, after the game launched for 60 seconds, the function "goBackground" is called, taking the program 10 seconds to transfer the evaluate layer of the audio scene from its initial one to the background.

```
<scene>
   <instance_visual_scene url="#DefaultVisualScene"/>
   <instance_audio_scene url="#DefaultAudioScene">
      <extra>
         <technique profile="SHCHAN">
            <function id="goBackground">
               <interval>10</interval>
               <evaluate_scene>
                  <render>
                     <layer>background</layer>
                  </render>
               </evaluate_scene>
            </function>
         </technique>
      </extra>
   </instance_audio_scene>
   <extra>
      <technique profile="SHCHAN">
         <time_event start="60">
            <call_function url="#goBackground"/>
         </time_event>
      </technique>
   </extra>
</scene>
```

3.3.3 Scripting Language

The content of interactive applications is designed not only for the interaction with the user but also for the interactivity between the elements of the content. Given that the scene graph technology is reaching its limit to manage all the relationships between objects, a script language for describing more complex dynamic behavior for interactive applications is demanded. Consequently, we suggest a high-level language (embedded or external) that

enables scripting of auditory behaviors from sound designers' perspective yet in a programming solution. This script can be treated as an interface glue code between COLLADA and the audio API.

3.3.3.1 Programmable Units

Placing pieces of programming code into COLLADA is in fact not a brand-new design. In COLLADA FX (effect), shading language code can be located internally or externally: The `<code>` element provides an inline block of source code into the `<effect>` declaration to be used to compile shaders; the `<include>` element imports source code or precompiled binary shaders into the FX runtime by referencing an external resource. We intend to implement the dynamic-script system for the audio description based on this technology. Programming code that steers dynamical behaviors or switches SLODs can be either included as XML schema string data type within a block of a `<code>` element, or saved in an external file imported by an `<include>` element at the `<scene>` level within the instance document.

3.3.3.2 DOM and JavaScript

The **D**ocument **O**bject **M**odel (DOM) is a platform- and language-neutral convention for representing a logical arrangement of valid HTML and well-formed XML documents. The public interface of a DOM is specified in its API, enabling the so-called DOM scripting, such as JavaScript, referring programmatically accessing and updating the content and structure of documents.

As well-formed in XML schema, COLLADA has its own defined document object model, created by Sony Computer Entertainment America. The COLLADA DOM is a low-level API that provides a C++ object representation of a COLLADA XML instance document.

The scripting language for dynamic display and interaction with the COLLADA data is a combination of these technologies introduced above. For instance, methods, such as `getNodeById(elementId)` and `getAudioObjectById(elementId)`, which are used to retrieve a specific element and its data are such implementations based upon the

COLLADA DOM method – `getChildren()` of the `daeElement` class, as well as W3C DOM Level 2 method – `getElementById(in DOMString elementId)`. The designation of the properties and methods of objects is meant to be straightforward for developers to understand and further enhance the capability.

3.3.3.3 Implementation

In the following, we introduce several properties and methods to COLLADA objects for the first implementation, and give examples of how they may be applied to code.

Example 1

An `<audio_object>` can be played back automatically or triggered by an event, such as a mouse-click or a collision. Playback methods simply plays, stops or pauses the selected audio object.

By means of the scripting code, the example above of door-clicking can be rewritten as below. It is mandatory to instantiate the audio object before it can be played back. Otherwise, its playback control methods are invalid.

```
<library_audio_objects>
   <audio_object id="bellAudObj">
      …
   </audio_object>
</library_audio_objects>

<library_visual_scenes>
   <visual_scene id="DefaultVisualScene">
      <node id="doorNode">
         …
      </node>
   </visual_scene>
</library_visual_scenes>

<library_audio_scenes>
   <audio_scene id="DefaultAudioScene">
      <instance_audio_object sid="bellIAudObj" url="#bellAudObj">
         …
```

```
        </instance_audio_object>
    </audio_scene>
</library_audio_scenes>

<scene>
    <instance_visual_scene url="#DefaultVisualScene"/>
    <instance_audio_scene url="#DefaultAudioScene"/>
</scene>

<code>
  var scene       : Scene;
  var visSce      : VisualScene;
  var audSce      : AudioScene;
  var doorNode    : Node;
  var bellIAudObj : InstanceAudioObject;

  function clickOnDoor()
  {
     doorNode.Rotate(0, 1, 0, 90);
     bellIAudObj.Play();
  }

  function update()
  {
     doorNode.onClick = clickOnDoor();
  }

  function main()
  {
     scene     = this.getScene();
     visSce    = scene.getActiveVisualScene();
     audSce    = scene.getActiveAudioScene();

     doorNode   = visSce.getNodeById("doorNode");
     bellIAudObj= audSce.getInstanceAudioObjectBySid("bellIAudObj");
  }
</code>
```

Example 2

We expect to be able to apply the solution of dynamic scripting in not only the audio but also the rest of the phases of the COLLADA scenes including visual and physics. In addition to mouse events, the dynamic script also supports collision detections. Moreover, the interaction is not restricted between the same-file elements, but a COLLADA object is also allowed to interact with those from other dae documents.

In this example, we mean to detect collisions with rigid bodies in physical simulation. When the rigid body of the door collides with that of the hand model from the external "actor.dae" file, the door will open and a knocking-door sound will play. These follow-up reactions only happen when this rigid body (collider) has begun touching another rigid body.

```
<code>
  var daeDoc          : DAE;
  var actorDae        : daeRootElement;
  var scene           : Scene;
  var visSce          : VisualScene;
  var phySce          : PhysicsScene;
  var audSce          : AudioScene;
  var doorNode        : Node;
  var actorHandIRBody : InstanceRigidBody;
  var doorIRBody      : InstanceRigidBody;
  var knockIAudObj    : InstanceAudioObject;

  function update()
  {
     doorIRBody.onCollisionEnter(actorHandIRBody) = function ()
     {
        doorNode.Rotate(0, 1, 0, 90);
        knockIAudObj.Play();
     }
  }

  function main()
  {
     scene = this.getScene();
     visSce = scene.getActiveVisualScene();
     phySce = scene.getActivePhysicsSceneById("DefaultPhysicsScene");
     audSce = scene.getActiveAudioScene();

     actorDae = daeDoc.Open("actor.dae");
```

```
        actorHandIRBody = actorDae.getScene().getActivePhysicsScenesByIn
dex(0).getInstancePhysicsModelBySid("handIPModel").getInstanceRigidBody
BySid("handIRBody");

        doorNode    = visSce.getNodeById("doorNode");

        doorIRBody                                                      =
phySce.getInstancePhysicsModelBySid("doorIPModel").getInstanceRigidBody
BySid("doorIRBody);

        knockIAudObj = audSce.getInstanceAudioObjectBySid("knockIAObj");

    }
  </code>
```

Example 3

The `param` property is an array mapped to the parameters of a sound. This is not a property of the `<audio_object>` itself but its child sound node. Depending on the type of the child sound node, it may refer to a different semantic and functionality. For instance, `param[0]` of `<sound>` is its `direction`, whereas it indicates `intensity` for `<sound_2d>`, or the first self-defined parameter for `<sound_ext>`.

In many cases, the parameter values of a COLLADA object changes with its corresponding game component. In this example, we fetch the rpm parameter value of the game component created from the native COLLADA document frame-by-frame, and then update the `param[0]` of the child sound of the audio object "rpmAudObj" with this value. A parameter of an audio object can be referenced by either its name or the index within the parameter array.

```
  <code>
    var car : appComponent;
    var audSce : AudioScene;
    var rpmAudObj : AudioObject;

    function update()
    {
```

```
      var rpmValue : float = car.getParamByName("rpm");
      rpmAudObj.sound.setParamByIndex(0, rmpVal);
      //rpmAudObj.sound.setParamByName("rpm", rpmVal);
   }

   function main()
   {
      car = this.getAppComponent();
      audSce = this.getScene().getActiveAudioScene();
      rpmAudObj = audSce.getAudioObjectByURL("rpmAudObj");
      rpmAudObj.getInstance().Play();
   }
</code>
```

Example 4

This example demonstrates how to fetch the "environment" parameter of the virtual world from the application and assigns it a value. For instance, if the preset value of the desired acoustic environment is "city", the "environment" parameter of the game will be fed an enum name "city" from the enumeration "ENVIRONMENTCODE" to imply that the game world is currently in the environment of a city.

```
<code>
  enum ENVIRONMENTCODE
  {
    generic = 0,
    room,
    bathroom,
    livingroom,
    stoneroom,
    …
  }

  var audSce : AudioScene;
  var iAcoEnv : InstanceAcousticEnvironment;
  var env : ENVIRONMENTCODE;

  function main()
  {
```

```
      audSce = this.getScene().getActiveAudioScene();
      iAcoEnv = audSce.getInstanceAcousticEnvironmentBySid("inCity");
      env = iAcoEnv.getPreset();
      this.getAppComponent().setParamByName("_inEnvironment", env);
   }
</code>
```

Example 5

As discussed in previously, the single audio scene instantiated inside the <scene> element is the default audio scene to be presented. However, the switch of the activation of audio scenes can be simply done by a line of code. Developers must ensure that only one audio scene is active at a time.

```
<code>
   var scene       : Scene;
   var libAudSce   : LibraryAudioScene;
   var audSce1     : AudioScene;
   var audSce2     : AudioScene;

   Function switchToLevel2
   {
      scene.setActiveAudioScene(audSce2);
      // audSce1.active = FALSE;
      // audSce2.active = TRUE;
   }

   function main()
   {
      scene = this.getScene();
      libAudScene = this.getLibraryAudioScenes();
      audSce1 = scene.getActiveAudioScene();
      audSce2 = libAudioScenes.getAudioSceneById("Level2_AudScene");
   }
</code>
```

3.4 Pipeline and Architecture

3.4.1 Overview

Having the design specification of COLLADA audio description presented previously, we are now able to implement the COLLADA auditory scene to represent sound in 3D virtual environments. Including the sound capability, the COLLADA description has been varied in four kinds of scene simulations. Since these partial scenes have equivalent scene-graph structures, each of them may master its own simulation procedure in parallel, and finally aggregate the results together to complete the COLLADA model. Along these lines, when importing a COLLADA file into a 3D virtual scene, the model will not merely be visualized (meshes, textures, visual effects, animations), but it will be auralized by playing back sound effects and will allow physical interactions, such as collisions. The paradigm in FIGURE 3-9 depicts the parallel process of representing COOLADA descriptions in a virtual scene.

In this section, we have a discussion about using COLLADA during the development of interactive applications including video games. We first discuss the decision making in selecting a proper content for applications to use. Subsequently, we present how to implement the COLLADA scene description from a technical point of view, concerning COLLADA's role as a transitional standard for transporting digital assets, and clarify how various software tools are involved in analyzing, processing, and rendering COLLADA in virtual environments. Lastly, the architecture of the proposed sound engine is presented. We explain how the structural design and the process flow can be improved through the implementation of extensive sounds and dynamic scripts facilities.

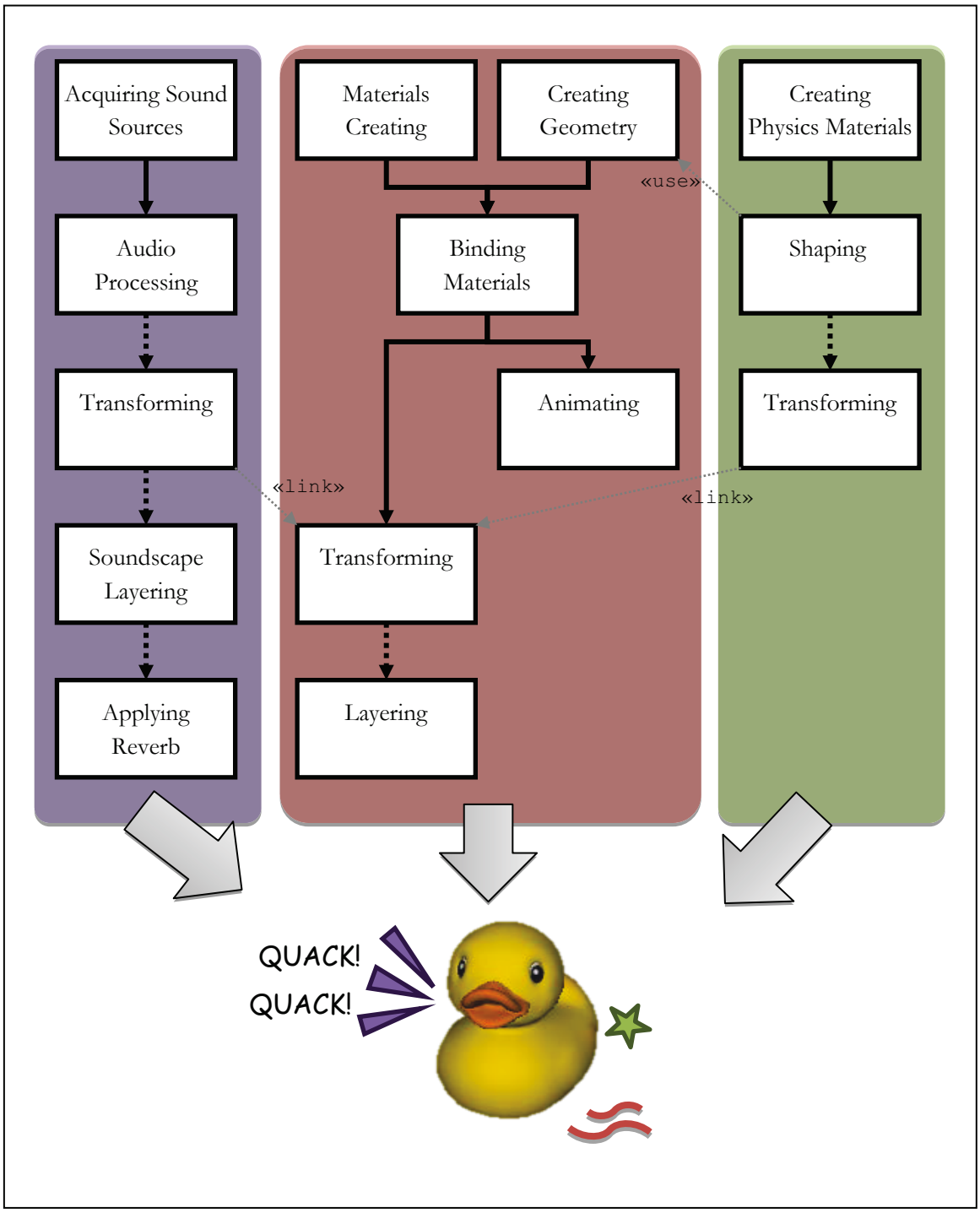


FIGURE 3-9. The realization process of a COLLADA 3D model.

3.4.2 Selection of the API-Specific Profiles

Given that different applications may have their own preference on audio solutions (audio APIs), COLLADA sound capability provides an option for applications to decide which content they want to make audible by means of "profiles". The profile provides the schema to indicate which APIs are necessary to bridge the details of a COLLADA sound and the importing application. For instance, the description embraced in `<profile_FMOD>` is only available for the FMOD-compliant programs. If an application is compatible for more than one target profile, it may have its own decision criteria to rank the alternatives in order of preference. In any case, all applications are expected to realize `<profile_COMMON>`, which is API-agnostic.

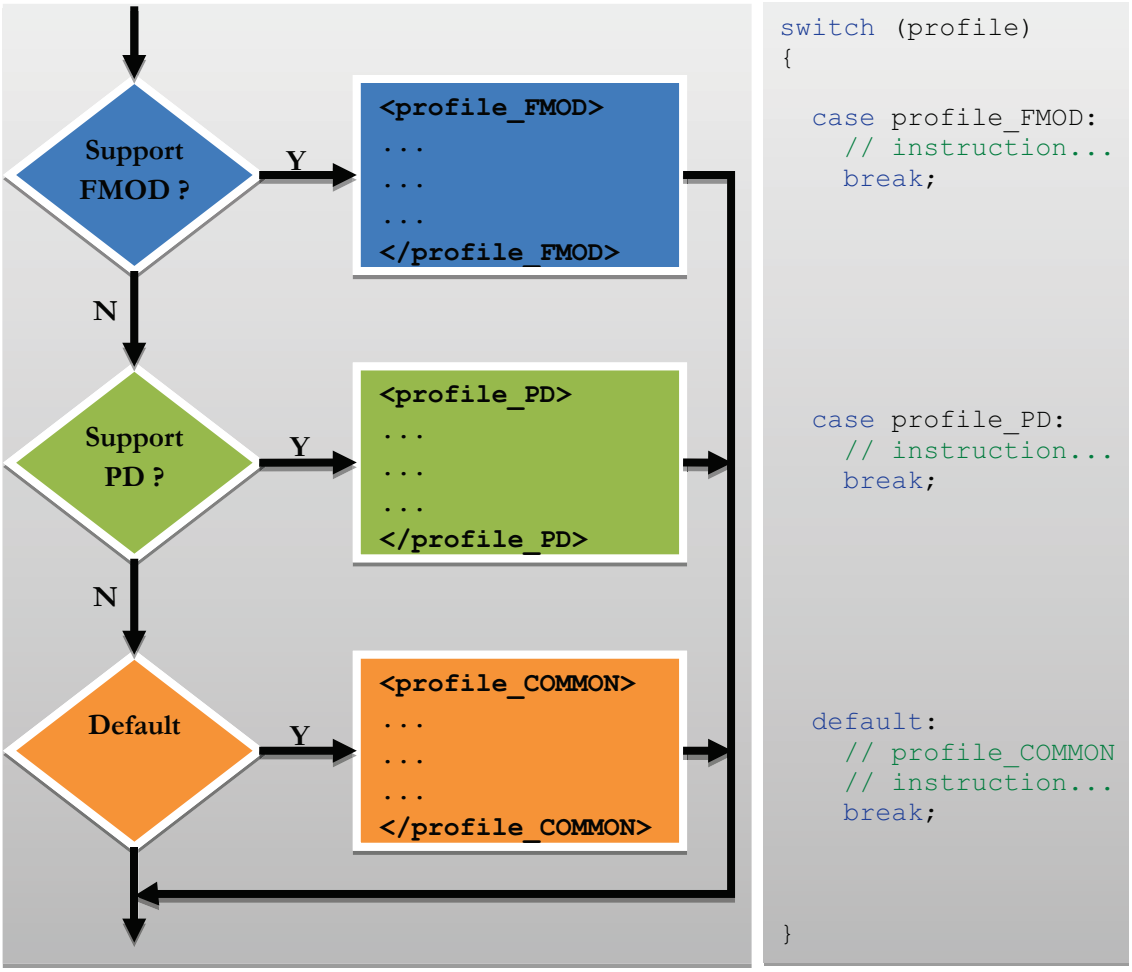


FIGURE 3-10. Paradigm of selecting an API-specific profile using the switch-case method.

3.4.3 COLLADA with Game Development Toolsets

COLLADA acts as an intermediate format in the content pipeline for interactive applications. That means rich contents can be stored in the COLLADA format and then be processed to game assets that are recognizable for the runtime (game engine). At that point, programmers are able to implement the assets as classes in an object-oriented library by means of all the facilities of object-oriented programming. FIGURE 3-11 diagrams the development workflow of interactive applications using COLLADA files.

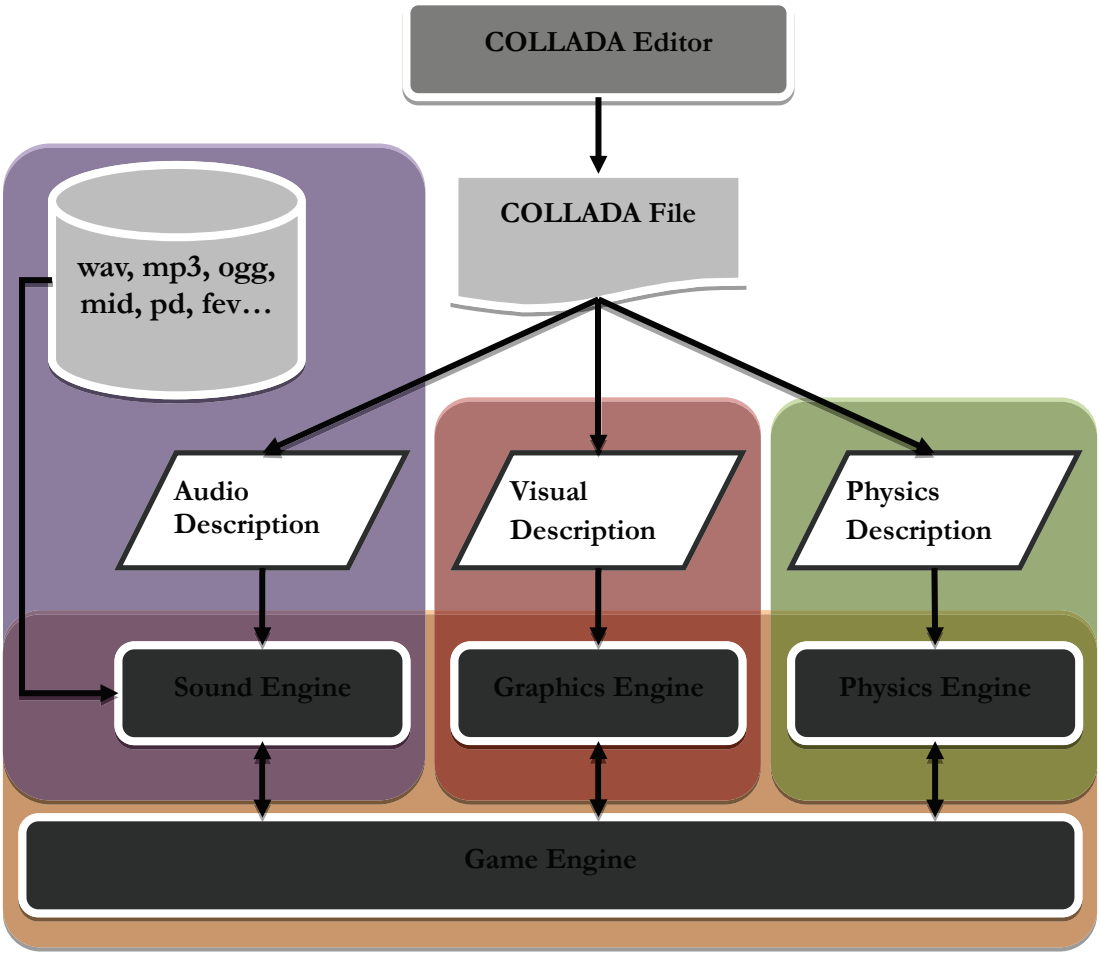


FIGURE 3-11. Workflow of using COLLADA files in the game development.

Visual and aural contents specified in the game design are respectively created by the teams of graphic artists and sound designers. As there already exist many COLLADA-

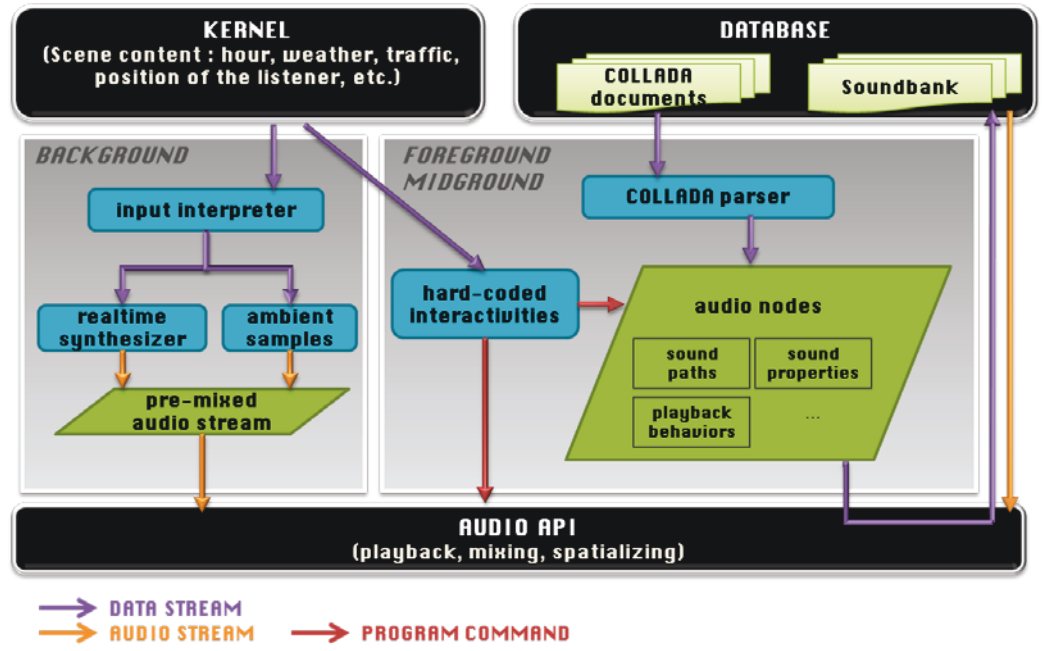
supported (i.e., allowing for importing and exporting COLLADA files) 3D computer graphics tools, such as Maya and 3ds Max, for making animations, images, models and visual effects, it is suggested to have a comparable editor for sound designers to create audio contents and export them into the COLLADA format.

Scene descriptions contained in a COLLADA file can be retrieved and parsed into runtime assets by categories and then managed by assorted "engines" (implemented with code libraries and APIs).

General driven game engines provide a programming environment where game elements in different categories (e.g., graphics, animations, physics, AI, gameplay, networking, memory management, multi-threading) are integrated and communicate with each other for application developments. The leading game engines, such as Unity3D, UDK and CryENGINE, are comprised of a set of software libraries, each of which executes a series of functions to perform the dynamic behaviors and interactivities in the game. The core functionality typically includes a 3D graphics rendering engine, a sound engine, a physics engine, and some other specific types of modules. Games developed with these game engines can be, in principle, ported to cross-platforms including computers, consoles, and/or mobile phones.

The proposed sound engine architecture diagram is depicted in FIGURE 3-12, the upper of which is the old version of the sound engine architecture where the control of real-time synthesis for background ambience is operated separately from COLLADA management. In this version of design, interactivities (user interactions, game state and parameter changes) relative to sound behaviors are mostly hard-coded, costing heavy communication load between programmers and designers and losing the flexibility of development. The plan in the bottom explains how enriched COLLADA can integrate sound resources by embodying extensive sounds (including Pure Data patches and FMOD event objects) in the COLLADA scheme, and further simplify the procedure and communication by specifying interactions in script manipulation.

BEFORE



AFTER

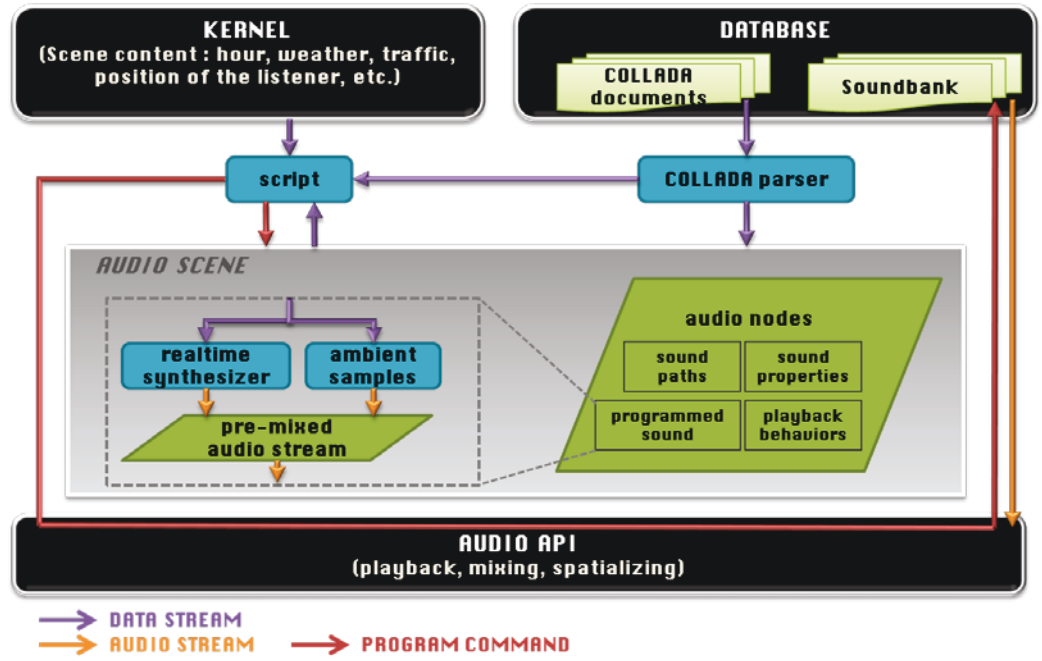


FIGURE 3-12. Integration of COLLADA in sound engine architecture.

Unlike delivery formats like X3D that are intended to contain the information of interactivity for interactive applications, COLLADA provides a standard language to describe 3D assets but not their runtime semantic [AP07]. As such, definition and implementation of how the content will be used is left to the application author. A COLLADA-compatible tool, such as Papervision3D or Unity3D, usually has its own solution to compile source data stored in COLLADA into runtime assets. It is not our intention to change its position occupied in the production pipeline. As a result, even though all the intelligence is comprehensively described within COLLADA script, the effort of realization shall be made by the importing application.

3.5 Summary

In this chapter, we discuss the concept and the implementation of audio scene design in COLLADA. Complying with the design principles and goals, we present an enhanced audio capability in COLLADA standard. The audio scene constructs upon the same structure of existing COLLADA scene graphs, grabbing materials from which to avoid data duplications. The COLLADA audio describes both static and dynamic sound information. The classic static part of the audio scene is a hierarchical composition of numerous audio nodes of which the definition and functionality are based on authentic scene description standards, such as MPEG-4 AudioBIFS and AAML. TABLE 3-1 sorts the essential auditory nodes proposed in COLLADA. On the other hand, the dynamic section is an initiative that utilizes the approaches of "event trigger" and "function call" to integrate the paradigm of event-based programming into the scene-graph XML-oriented structure of COLLADA. We also obtain the advantage of COLLADA programmable units to allow inline or external scripting codes for handling complex dynamic objects and interactivities. We suggest that the more adaptive descriptions should be not restricted to the audio simulation but also available for the representations of graphics, physics, AI, and so on.

Moreover, the soundscape layering technique is applied to the COLLADA audio scene graph. The implementation of the SLOD uses the existing COLLADA mechanism offered for the LODs of visual and physics scenes so the approach would meet the standard analyzing and processing procedure for the COLLADA content. The concept and design of the urban soundscape ambiences is a collaboration research work done with Tiger et al. More details will be discussed in Tiger's Ph.D. Thesis [TPHD].

TABLE 3-1. COLLADA audio nodes.

COLLADA Audio Nodes	Reference	Description
<audio_source>	AudioSource (MPEG-4)	It adds a sound source into the scene. FMOD events and Pure Data patches are acceptable in addition to common audio formats.
<audio_buffer>	AudioBuffer (MPEG-4)	It records a segment of sound to be used in interactive playback.
<audio_delay>	AudioDelay (MPEG-4)	It allows child sounds to be temporally delayed or advanced for synchronization.
<audio_mix>	AudioMix (MPEG-4)	It mixes M channels of sounds into N channels through a simple matrix calculation.
<audio_switch>	AudioSwitch (MPEG-4)	It selects a subset of the input channels to pass through. The new <random> function enables the option to randomly select sources to pass through.
<sound>	Sound (MPEG-4)	It defines sound in a 3D scene
<sound_2d>	Sound2D (MPEG-4)	It defines sound in a 2D scene
<sound_ext>	New	It is used with the <audio_source> to define extensive designer-generated types of sound in a 3D scene.
<audio_object>	New	It is the result of COLLADA sound that will be presented to the listener
<audio_dsp>	AudioFXProto (MPEG-4)	It delivers a DSP unit (effect).

		In contrast with `AudioFXProto`, it contains no audio stream but the setting of a DSP function.
`<acoustic_environment>`	`environment` (AAML)	It specifies perceptual parameters that model the audio environment.
`<audio_scene>`	New	It embodies the entire set of information that can be auralized from the contents of a COLLADA resource.

The anticipated representation of sound in virtual cities is designed upon the principles of MPEG-4 and COLLADA. Here we examine the execution of the plan by comparing the audio capabilities implemented with the initial goals.

- Sound Sources

An audio scene in COLLADA is defined by the layers of soundscape. Layers are composed of `<audio_object>`s, each of which contains a sound (i.e., `<sound>`, `<sound_2d>`, or `<sound_ext>`) with parameters. Soundscape layers together with sound sources can be organized by objective measures (i.e., graphical facts: distance, density) or subjective evaluations (i.e., semantic choices: social, culture).

- Acoustic Parameters

The `<acoustic_environment>` element provides a room reverberation simulation by perceptual parameters.

- Microphone

Even though there is no specific element that represents the microphone, most auditory nodes are still able to determine whether or not there are important phase relationships (e.g., mono, stereo, or multi-channel) between the channels via the use of the `<phase_group>` flag. In addition, the `<spatialize>` child element of the `<sound>` node can specify whether the sound should be spatialized in the 3D scene; and the `<direction>` orients the audible filed.

- DSP

Some suggested audio nodes are used for processing purpose. For instance, the `<audio_mix>` element can mix M channels of sounds into N channels through a simple matrix calculation, and the `<audio_delay>` allows sounds to be temporally delayed or advanced for synchronization. Above all, the `<audio_dsp>` provides a set of pre-defined DSP effects. Individual DSP parameters can be overridden to customize the effect.

According to the analysis, we conclude that the essential features of the promising sound description standard have been covered in our proposition of the audio scene in COLLADA. The aural depth can be demarcated by projecting the soundscape of city ambiences with the scene-graph hierarchy of COLLADA. Moreover, the provision of extensive sound types and dynamic description may deliver further possibilities beyond the original design goals. Advanced functionalities that have not been accomplished can be considered extensions. The schema must meet the design principles of assessment.

In the next step we want to validate the technical feasibility of the proposed audio description in COLLADA. We learn valuable lessons through the implementation of several prototypes, the collaboration with sound designers, and the development of a 3D sound engine integrated into the *Terra Dynamica* project detailed in the next chapter.

More prescribed specification of COLLADA audio nodes can be referred in APPENDIX B, written in accordance to the syntax rules for COLLADA extensions. A coherent and complete proposal for COLLADA Audio will be officially presented to the Khronos Group.

This page intentionally left blank.

CHAPTER 4.

Experiments and Evaluation

OUTLINE

The ultimate objective of this research is to bring dynamic sound into interactive virtual reality applications. By using the scene language defined in this thesis, descriptions of aural phenomena and everyday auditory experiences can be standardized and then reproduced in 3D virtual environments.

In the preceding chapters of this thesis, we have presented a novel audio scene composed of auditory elements to the COLLADA scene description standard based on the proposed design concept. The sound capability includes both classic static conformations and novel dynamic behaviors. In this chapter, we learn the feasibility of implementing the COLLADA audio from both technical and sound design perspectives through several early prototypes, sound design approaches, and finally a portable sound engine.

4.1 Early Implementation

4.1.1 Overview

In the first stage of the work, we developed several prototypes of virtual worlds, using COLLADA models under diverse programming environments. The prototypes have rather simple scenarios with a few **N**on-**P**layer **C**haracters (NPCs) either stationary at specified locations or moving around. Users control an avatar in a first-person or third-person view to navigate through a 3D map to explore the sound world. The idea is to achieve the following goals:

- To test and evaluate the COLLADA compatibility of various 3D game engines and graphics toolkits.
- To demonstrate that COLLADA schema can provide a quick and easy accessibility and control to the audio content.

4.1.2 COLLADA Compliant Game Engines

The target domain of interactive applications of COLLADA is that in the entertainment industry, where the content is mostly three-dimensional and is game related. At present,

increasing number of game engines (free or proprietary release) and 3D computer graphics software have begun to support COLLADA format.

During the early stage of the research and development, no sophisticated visual representation was needed. The original intention was to rapidly prototype a three-dimensional visual world application with COLLADA-based contents. Therefore, three major graphics rendering and/or game engines – Ogre3D, Panda3D and Unity3D were selected for the first implementation according to personal programming experience. Under each environment, three immobile sounding COLLADA models (the classic COLLADA duck, cat, and cow) are spread in a simulated terrain. The user in a third-person view uses keyboard keys to control his/her direction and movement, to locate himself/herself on the map and to discover the sound sources by listening.

FMOD was chosen to be the sound system consistently for the experiments, given its high popularity and compatibility. OpenAL was another available audio rendering tool for most of the prototyping environments. The following elucidates the integration of the COLLADA, sound engine and game engines for more details.

4.1.2.1Ogre3D

The **O**bjected-Oriented **G**raphics **R**endering **E**ngine (abbreviated as OGRE3D or OGRE) is an open-source real-time scene-oriented 3D graphics rendering engine. As its name states, it only provides real-time 3D rendering capabilities, leaving developers the space to implement the rest of the game-required functionalities.

Ogre is plugin-oriented making it possible for application writers to customize much of its behaviors by registering new add-ons (plugins, tools and supplemental libraries). Therefore, even though Ogre does not natively support COLLADA, there are several COLLADA plugins available. The Ogre COLLADA plugins act as an interface between an Ogre application and the COLLADA APIs (e.g., OpenCOLLADA, FCollada, and COLLADA DOM), which are libraries to facilitate loading, reading and writing COLLADA files under programmatic control.

The figure below is a screenshot of the prototype using Ogre as the graphics rendering tool. Due to the fact that Ogre-COLLADA plugins had a rather poor support at this stage, we chose to use simple cube objects in various sizes and colors to graphically represent three different actors that produce different sounds correspondingly.

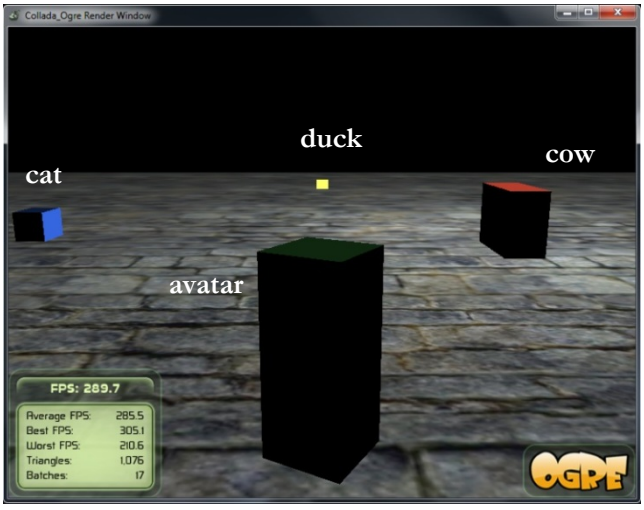


FIGURE 4-1. Screenshot of the early game prototype using Ogre3D.

4.1.2.2 Panda3D

Panda3D is a free, open-source game engine. It utilizes FCollada, a C++ library offering support for COLLADA interoperability, for converting COLLADA models to the Panda3D's native format .egg (through "dae2egg" convertor) or directly loading .dae files into Panda3D. The version used was Panda3D 1.7.2 with Python as the game-development language. Panda3D can be configured to use either OpenAL or FMOD sound system.

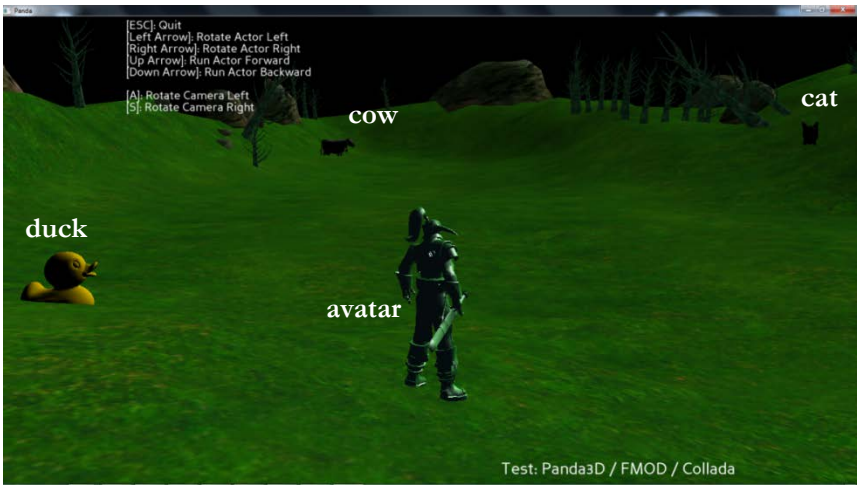


FIGURE 4-2. Screenshot of the early game prototype using Panda3D.

4.1.2.3 Unity3D

Among the three selected game engines, Unity3D (abbrev: Unity) has the most complete support of COLLADA .dae files. The way to import COLLADA in Unity is simply to drag and drop a COLLADA file as a new asset into a Unity project. Then the mesh with the material will be managed as a Unity object in the project. The program is runnable on Unity Pro versions 3.3.0 and higher with OpenAL and/or FMOD libraries. It was written mainly in C# language supplemented by JavaScript for some movement controls.

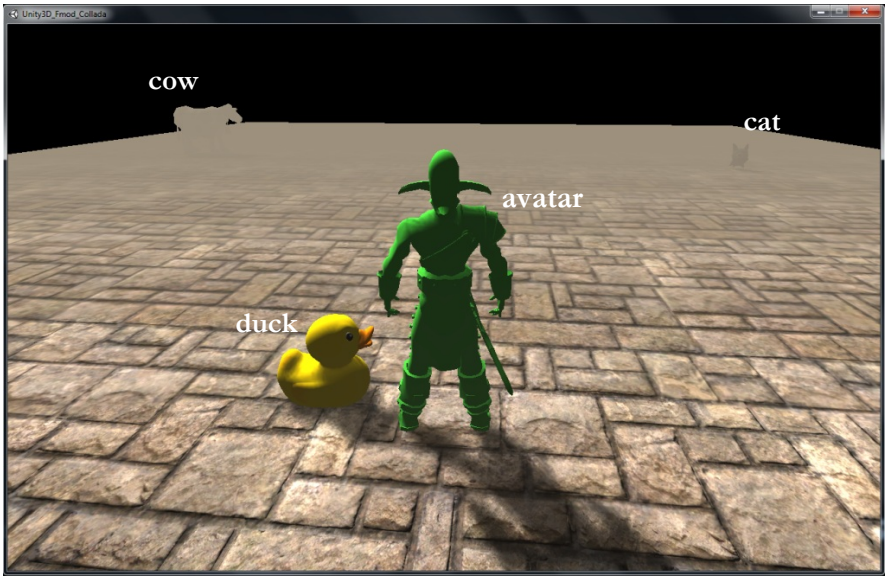


FIGURE 4-3. Screenshot of the early game prototype using Unity3D.

Appreciated for its comprehensive integration with COLLADA, Unity allowed us to rapidly prototype interactive COLLADA applications. We thus built later a couple of more mockups using advanced scenarios of both fixed and roaming actors. In the succeeding prototypes, we change the camera view from the third person to the first.

FIGURE 4-4 is a snapshot of a simple demo built in Unity3D. The theme is three COLLADA vehicles making noise (the blue is steady; the reds are running) in Rue Sufflot in Paris. The models of the buildings and streets are provided by the project *Terra Dynamica*. The Sufflot area is also the main scene for this project.

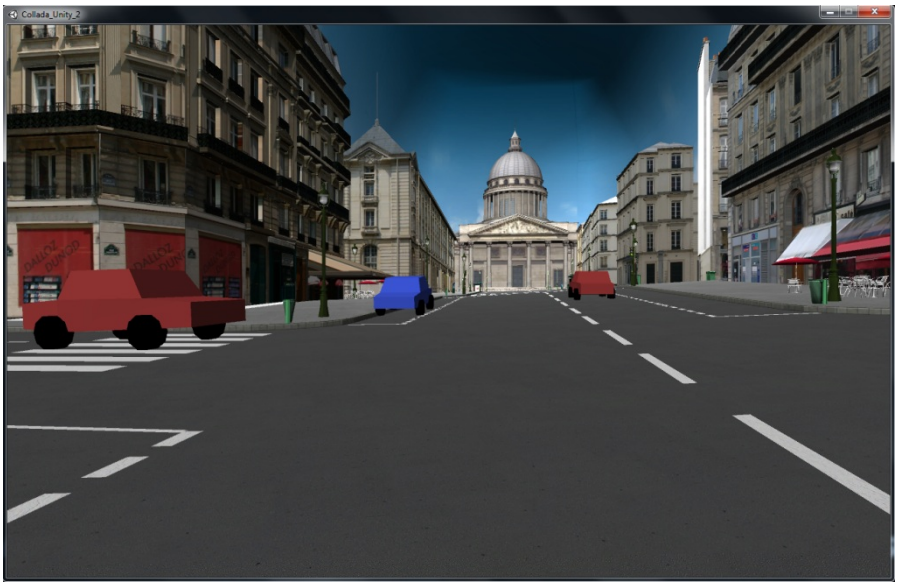


FIGURE 4-4. Screenshot of the game prototype with moving NPCs using Unity3D.

4.1.2.4 Summary

According to the study of game engines and the implementation of prototypes, we learn that game-related tools have been increasingly compliant with the COLLADA format. Compared to free and open-source game engines, proprietary ones (freeware or commercial) provide relatively more thorough support for the functionality (in terms of visual and physics) of COLLADA. Unity3D performs the highest efficiency among the selected game engines during the first investigation and evaluation. Other advanced COLLADA-compatible game engines, such as UDK and CryENGINE, make them a good choice for developing a higher-quality visual world application.

4.1.3 Audio Content Control

In the early prototype implementation, we intended to keep the audio scene simple in COLLADA. Instead of complex contents, the audio scene contained only several audio source nodes having information of their file paths and names, as well as the corresponding audio objects with some basic parameters and playback behaviors, such as pitch, volume, and loop. The primary objective was to address that with the scene graph hierarchy of the

auditory scene acoustic units in COLLADA were easily retrieved and organized. Without having to modify the programming codes, a COLLADA model might still emit alternative sounds by simply swapping the audio object presented or adjusting its parameter values. Most of all, the prototypes demonstrated audio objects could be linked with visual nodes, sharing the coordinate system and transform. In the demonstrations, spatial sounds were attached to models respectively so characters would travel along with sound in the virtual world.

4.2 Sound Design

4.2.1 Overview

In the entertainment production, interactive sounds are typically reproduced via middleware audio toolkits (e.g., FMOD, Wwise) or visual programming languages (e.g., Max/MSP, Pure Data). For this reason, we are dedicated to standardize the description in COLLADA of the sound design process for external extensions so as to facilitate the communication between the sound content designers and application developers. The solution proposed gives the bigger role back to the designers even when the scene is dynamic or based on procedural synthesizers.

Currently two extensive sound presentations are suggested to be included in COLLADA Audio: Pure Data patches and FMOD events. In this section, we discuss their design and production procedures connected with COLLADA Audio. Through the actual implementations, the integration of external programmed sounds into COLLADA Audio descriptions is proved to be not merely theoretical but also practical from both sound designing and programming perspectives.

4.2.2 Interactive Sound Moving to Procedural Production

The classic way of designing sounds for interactive and/or virtual objects is to create the appropriate audio material aside from the virtual environment [C08b]. These sounds are created from genuine recordings or based on soundbanks or synthesized. Sometimes the

ending result is a mix of all of these techniques. Tools used to design the sound (e.g., filters, samplers) are frequently included in a **D**igital **A**udio **W**orkstation (DAW) environment (e.g., ProTools, Logic). These production environments were originally conceived for musical or audio documentary production as well as for linear audiovisual production.

The designed sounds are then included into the virtual environment. This can be done by the mean of either a generic sound engine (e.g., FMOD, Wwise) or a specifically developed solution. Within the sound engine, the sound designer defines the rules and behaviors applied to the playback and synthesis of sounds. These rules are based on the description of interactive events which unique ids are linked between the sound engine and the virtual environment with a dedicated API.

The sound is then tested within the virtual scene. If the result does not fit the expectations, the whole creation pipeline may be started over from DAW to integration. Another downside regarding this pipeline is inherent to the creation of the environment itself: as the graphic objects are developed, their needs in terms of sound might change. Hence sound production is separated from the visual production in time: sound design is often one of the last steps in the production process.

However, recent studies about sound design tend to focus on different strategies interlinking visual and audio. The improvements are reaching towards procedural ways of generating interactive soundscapes. Multiple goals are pursued with these strategies. One of the first intends is to create a non-repetitive design so that the listener does not have the feeling of hearing twice the same soundscape; using procedural audio is also a way to reduce memory consumption by switching from sample playback to sound synthesis [MCW06] or to use refined strategies of audio playback such as granular [P11] and concatenative synthesis [B10][SBV06].

Fields of application range from interactive installation to video games. On this matter, video games are completing a full circle as sound was originally synthesized within chips [F07]. Tools dedicated to real time synthesis are showing up in major video game sound engines such as FMOD and Wwise. Furthermore, dedicated procedural plugins for environmental sound design such as AudioGaming's *AudioWeather* [1]are released. Research in the field of interactive procedural audio is also leading to synthesis solutions [VAM11] and sound description languages [V09].

[1] *AudioWeather* is a middleware introduced by the company AudioGaming. It uses pure procedural audio to generate the sound in real time for video games weather system.

4.2.3 Pure Data

In the following, a sound design pipeline is proposed based on the COLLADA sound description exposed above. This pipeline relies on the close dialogue between Pure Data patches and COLLADA files and allows sound design for an interactive project to be developed closely to the graphic assets. We will first discuss the actual evolution of sound design techniques for interactive environments, then the linking of COLLADA files with **P**ure **D**ata (PD) and finally the architecture of the basic Pure Data patch dedicated to the COLLADA <sound_ext>/<audio_source> element.

4.2.3.1 Proposal for Interactive Sound Design

Our proposal is to simplify the sound design pipeline described in above by means of the COLLADA sound description of the virtual environment. We intend to link this description to the sound design tools. We also want this sound design architecture to communicate with any sound API and visual rendering engine so that it is possible to use to produce audio in a wide range of applications and environments. Hence our proposed architecture remains independent from the choice of sound APIs and graphics rendering engines (e.g., Unity3D, UDK, Ogre3D).

As for a sound design tool, we propose to use Pure Data, which is an open source graphical programming environment developed by Miller Puckette for audio, video, and graphical processing. Pure Data has been used in variety of artistic installations and interactive computer music and multimedia creations. It is a rich tool for audio design [F10a] allowing many different treatments and techniques of synthesis [P07b]. Many other solutions for graphical programming exist, such as Reaktor, Max/MSP, or Bidule. Pure Data appears easier to integrate in our project and provides a large amount of libraries letting us program the main features we need.

Linking between Pure Data and COLLADA is done through the <audio_source> node described above. The node points at the file path of the attached Pure Data patch and describes the essential characteristics of the sound, just like the <audio_source> node.

The <audio_object> is the processing conclusion presented to the listener. It is possible to carry a 3D sound <sound>, 2D sound <sound_2d>, or extensive sound <sound_ext>. The <audio_source> being at the lowest level of the audio

COLLADA description, a sound of `<audio_object>` may be composed of several `<audio_source>` nodes. From a sound design point of view, this architecture makes layering easier regarding audio-object-oriented sonification. Furthermore, the editing of a complex audio object does not lead to the destruction of the sound attached to it but to the edition of the joint audio layers.

4.2.3.2 Pure Data Design

As Pure Data patches are attached to the `<audio_source>` nodes, their design is intimately linked to its functionalities. They also receive data from the dynamic script described in

The below paradigm shows the connections coming in and out of the standard Pure Data patch attached to an `<audio_source>` and a dynamic script.

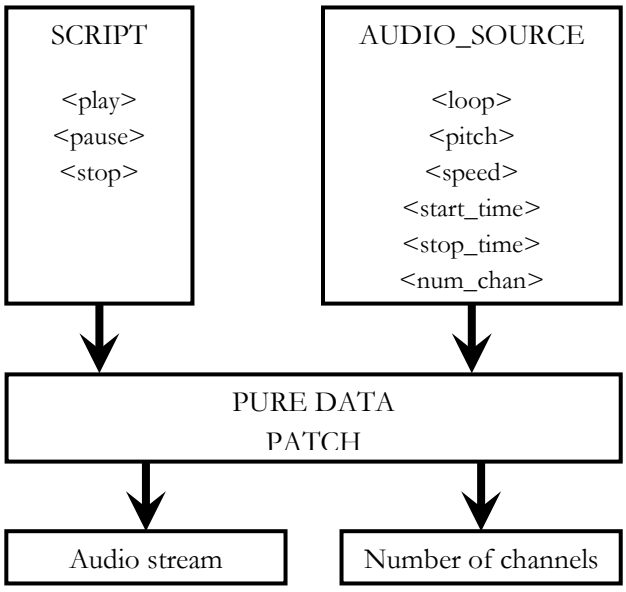


FIGURE 4-5. Flows coming in and out of the standard pure data patch.

The COLLADA `<audio_source>` element describes the following functions:

- `<loop>` defines the looping behavior of the sound a value of `0` sets the sound to loop forever and any other value sets the number of time the sound is played.

- `<pitch>` controls the payback pitch shifting of the sound, it is specified as a ratio where 1 indicates the original bitstream pitch.
- `<speed>` controls the playback speed of the sound, values other than 1 indicate multiplicative time-scaling by the given ratio.
- `<start_time>` specifies, in seconds, the starting point of playback.
- `<stop_time>` specifies, in seconds, the stopping position of playback.
- `<num_chan>` describes how many channels of audio are there in the decoded stream.

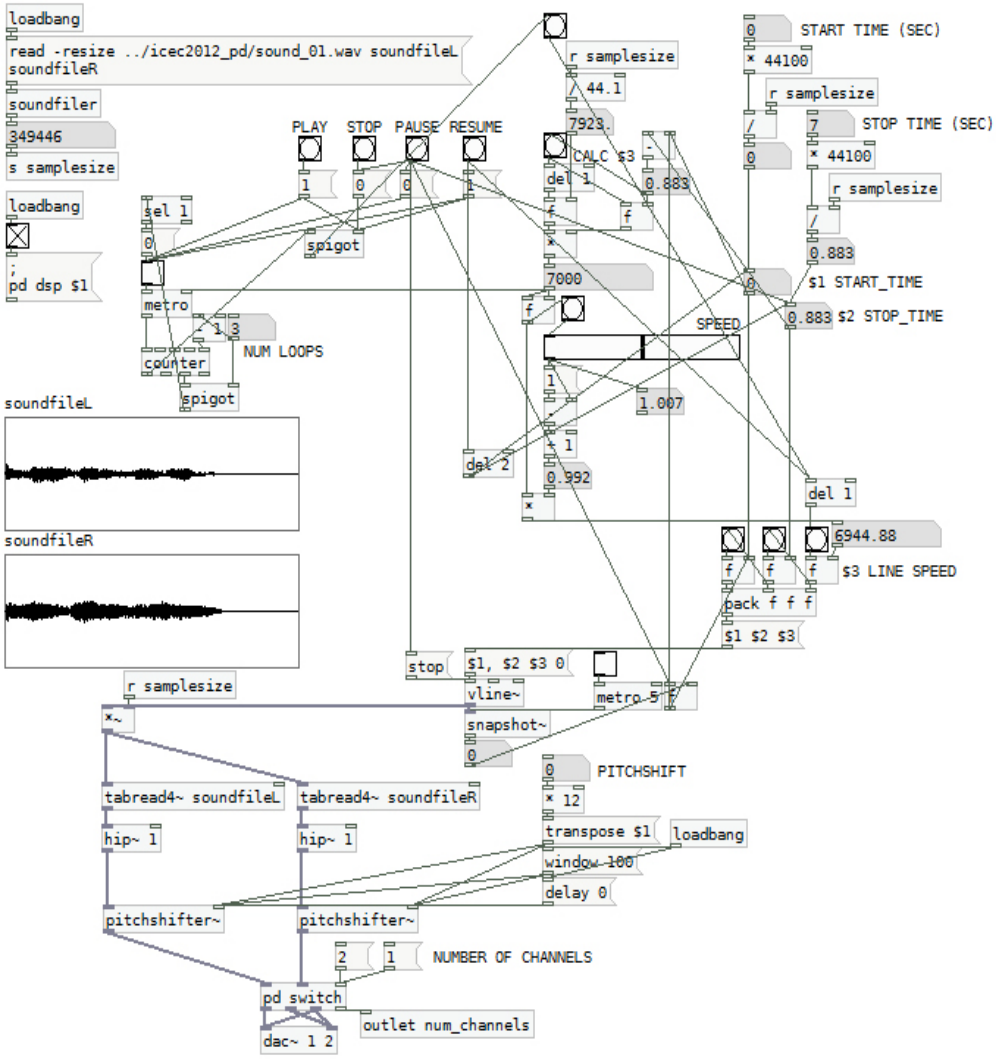


FIGURE 4-6. Basic Pure Data patch for the <audio_source> nodes.

Plus, dynamic scripting in COLLADA allows for a patch to receive Play, Stop, Pause and Resume information so that the sounds are triggered accordingly to the dynamic events of the scene.

Pure Data patches output two streams: the audio stream of the patch and the number of channels used within it. The number of channels information is used by the nodes <audio_mix> and <audio_switch> for processing purpose.

We implement these functionalities inside the basic Pure Data patch initiated in the <audio_source> node. Ratio and time values are rescaled to match their description inside the Pure Data patch.

It gets then possible to playback the complete sound attached to one <audio_source> or a part of it, at a selected pitch and speed. These functionalities are close to granular synthesis requirements [T98] so further possibilities should be considered besides simple audio playback.

4.2.3.3 Communication between Pure Data and Client Code

Pure Data is a real-time patcher programming language. It is possible to be extended by writing externals (object-classes) and patches (abstractions) working with Pure Data. There have been several works, such as "flext" and "libpd", dedicated to the communication between Pure Data and programming code. Flext is a C++ programming layer for cross-platform development of extensions to the Pure Data and Max/MSP real-time multimedia systems [G04]. Libpd, developed by Brinkmann et al., liberates raw audio from GUI and drivers, allowing for straightforward exchange of control and messages between PD and the client code into which it is embedded [BKL11][LIBPD]. Libpd turns PD into an audio synthesis and processing library, letting PD be more accessible across a greater range of cross-platform gadgets, including mobile phone apps, games, web pages, and art projects It can be used under a variety of systems (e.g., Windows, Linux, Mac, Android and iOS) and with multiple programming languages (e.g., Java, Objective C, C++, C#, Python and Lua), making it a decent glue tool to facilitate the capability of using Pure Data for creating sound experience in COLLADA-based applications.

4.2.4 FMOD

In addition to Pure Data, we propose to include FMOD events description in COLLADA via the use of `<audio_source>` and `<sound_ext>`.

FMOD is a sound middleware product from Firelight Technologies. It is a programming audio library and toolkit made for the creation and playback of interactive audio used in games and software applications. The FMOD sound system comprises three main parts: FMOD Ex (the low-level sound engine), FMOD Event System (higher level application layer on top of FMOD Ex), and FMOD Designer (sound designer tool).

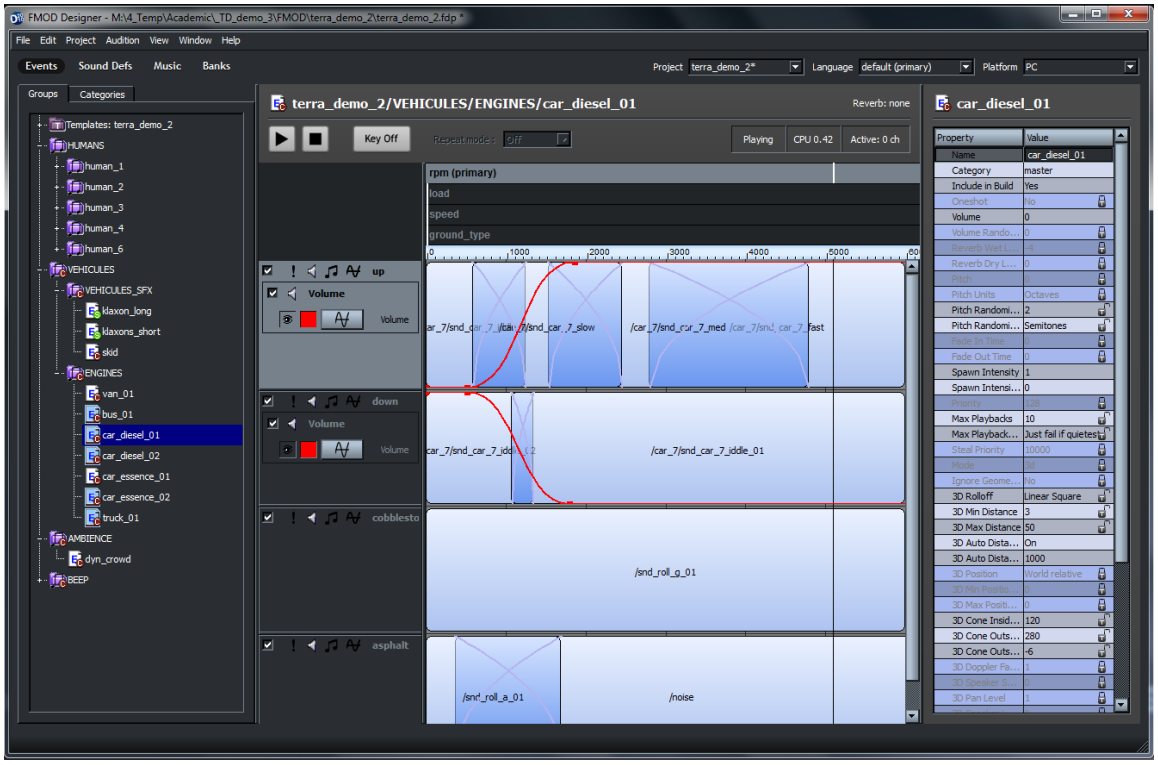


FIGURE 4-7. Creation of FMOD events in FMOD Designer.

During the development cycle, sound designers use the FMOD Designer tool to create an FMOD project for authoring complex audio events and music for playback. FMOD Designer provides an event editor for sound manipulations such as blending, volume contouring, pitch shifting, and spatializing. Each FMOD event contains a set of properties and parameters (with the default value and the range) which can be dynamically controlled

from the programming side at runtime. The data of the events including their parameters created by sound designers can all be stored in the fashion of the COLLADA format. To be precise, we use <audio_source>, the lowest-level node of the COLLADA audio scene graph, to navigate to the FMOD event. The locator path is a sequence of segments includes the file folder, filename of the event project, event group name, and finally the name of the event. The <audio_source> may then be progressed through the processing flow via an extensive sound element <sound_ext>. The procedure is similar to what is explained for PD patches. The parameters of the events can be managed through the use of <newparam>, <setparam> and dynamic scripts.

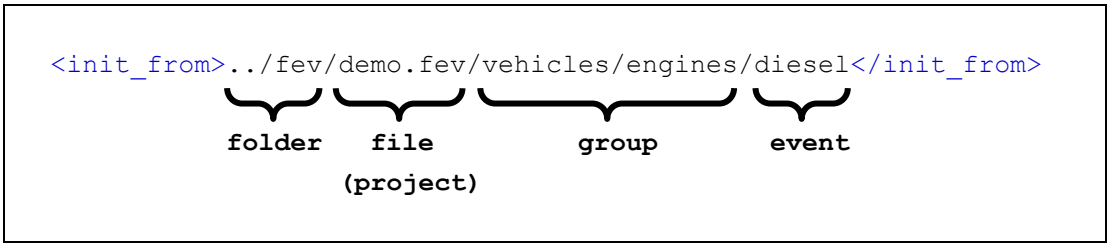


FIGURE 4-8. Example of the naming rule for FMOD events in COLLADA.

Successively, programmers are able to retrieve the information of FMOD events from the COLLADA document and utilize FMOD Event System to play back content generated with the FMOD Designer tool. This puts back a lot of creative power in the hands of the sound designers and simplifies life for the programmers.

Below is a segment of content from a COLLADA document. In this example, two types of sound source formats, WAV and FEV are given. All the programs are expected to be able to interpret the content embraced in the common profile, here that means to handle WAV formats. The choice of an FMOD event can be optionally chosen by the importing application that is FMOD-compliant for playing a diesel-engine sound effect.

Example

```
<library_audio_sources>
   <audio_source id="carWav">
      <init_from>../wav/vehicles/car.wav</init_from>
   </audio_source>
```

```
    <audio_source id="carFev">
       <init_from>../fev/demo.fev/vehicles/engines/diesel</init_from>
       <newparam sid="rpm">
          <semantic>RPM</semantic>
          <float>2000.0</float>
       </newparam>
    </audio_source>
 </library_audio_sources>

 <lbrary_sounds>
    <sound id="carWavSound">
       <instance_audio_source url="#carWav"/>
    </sound>
 <lbrary_sounds>

 <lbrary_sound_exts>
    <sound_ext id="carFevSound">
       <instance_audio_source url="#carFev"/>
       <newparam sid="carFev.rpm">
          <SIDREF>carFev/rpm</SIDREF>
       </newparam>
    </sound_ext>
 </lbrary_sound_exts>

 <lbrary_audio_objects>
    <audio_object id="carAudObj">
       <profile_COMMON>
          <instance_sound url="#carWavSound"/>
       </profile_COMMON>
       <profile_FMOD>
          <instance_sound_ext url="#carFevSound">
             <setparam ref="carFev.rpm">
                <float>2400.0</float>
             </setparam>
          </instance_sound_ext>
       </profile_FMOD>
    </audio_object>
 <lbrary_audio_objects>
```

4.3 Terra Dynamica and Sound Engine

4.3.1 MAC and SoundController

We have been engaged with a consortium of academic and industrial partners in the project *Terra Dynamica*, of which the goal is to develop 3D urban simulation technology that can be applied to a wide range of fields. The virtual city of *Terra Dynamica* is built upon a database translated to a COLLADA document, making it a great platform for us to demonstrate the representation of sound in virtual cities descripted by COLLADA.

The core Kernel (namely MAC) of *Terra Dynamica* encapsulates a plurality of internal modules performing the calculations of the simulation including representation of the environment, creation and destruction of virtual actors, physics engine, navigation, AI, scenario scripting. While the project requests audio uses within five main fields of applications (game and art, urban design and architecture, safety analysis, city transportation, proximity services), one of the tasks assigned to us is the development of a 3D sound engine that simulates real-time sound behavior within a dynamic urban scene.

The sound engine, termed "SoundController", is considered an external module like other application-centric controllers that can be pushed into the kernel sequencer for the hyper process (state-machine) control. In this way it can be synchronized with the game world state. The SoundController is built upon the architecture described in 3.4 Pipeline and Architecture. It includes a parser to load COLLADA files, parse the sound content of each document and convert it into a tree (scene-graph construct) of the audio scene. The AudioObjectManager contained within the sound module is used to manage (construct/destruct and control playback/volume) the collection of audio objects. It uses audio trees to create audio objects in three types according to their sound source formats.

The sound engine is built upon the FMOD sound system, mostly for the playback control of foreground sound signals and midground soundmarks. Pure Data is the use of producing dynamic city background ambience. The environmental sound in the background PD patch can be influenced on-the-fly by a number of game factors, such as the number of actors (pedestrians and vehicles) within the scene, or the position and the angle of the listening point. There are several solutions to integrate Pure Data with the SoundController. For the moment, it is made by a running instance of Pure Data that exchanges messages with the SoundController via TCP/IP socket ports. Nevertheless, using the libpd library in

the sound engine project for a direct access to the PD patch is considered a more efficient approach.

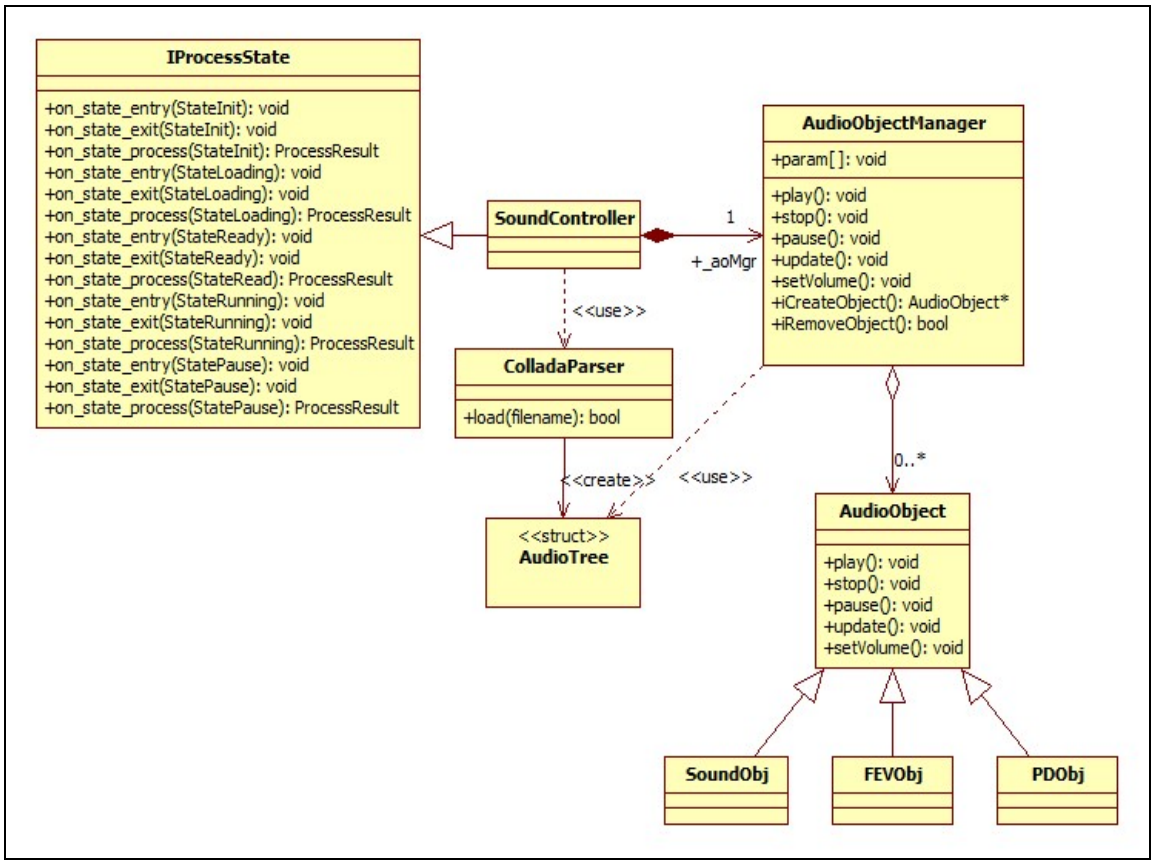


FIGURE 4-9. Class diagram of the sound engine execution.

The *Terra Dynamica* Kernel API is distributed in a package of kernel data, services, and a number of internal modules. With this we are able to monitor the simulation progression of the virtual city Paris through command prompt but with neither 3D graphics nor sound feedback. A full version of the demonstration can be done by incorporating with our SoundController plus the graphics rendering engine (ThalesView) developed in-house by the consortium member Thales Group. The framework of this scheme is shown in the below diagram. Additional modules can be tailored and added on to applications with specific purposes.

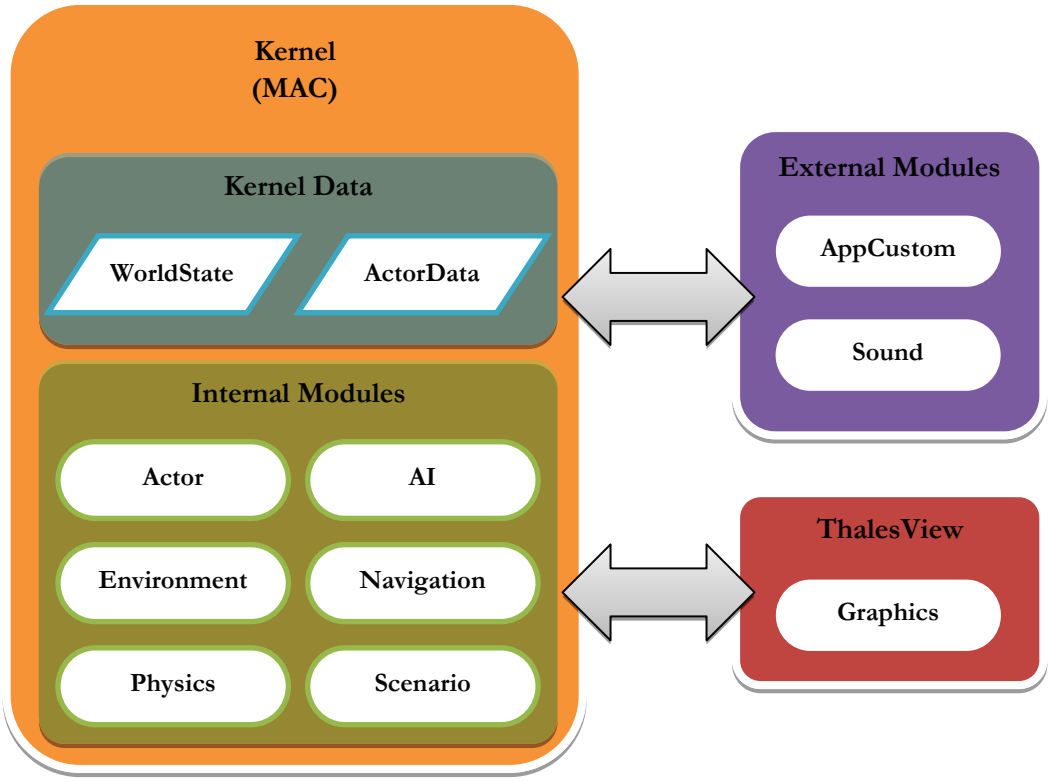


FIGURE 4-10. The framework of the *Terra Dynamica* system.

4.3.2 Terra Dynamica Demo

The project *Terra Dynamica* had presented its demos in *Futur en Seine*[2] in 2011 and 2012. These demos demonstrate a lively virtual metropolitan locality of Sufflot in Paris. Simulated actors include various types of pedestrians (genders, ages, businesses) and vehicles (engines, functions). Each actor comes along with a sound effect reflecting its appearance and activity. The sounds attached to individuals are created by means of FMOD events with a set of parameters whose values dynamically change according to the actors' behaviors. In contrast to these foreground signals is the background ambience produced by a pure data patch. The environment sound can be adaptively affected by the position (mostly the altitude) of the

[2] *Futur en Seine* is an international festival organized by Cap Digital, a member of the project *Terra Dynamica*. The ten-day event shows each year the latest digital innovations to French and international professionals and to the general public.

listener (camera) and the statistic density of actors within the listening zone. For example, when the camera is in the avatar height (first-person view), surrounded by the crowds, the background noise is generally loud giving an atmosphere of a busy city. When the camera is moving to a higher and higher level, the whole city noise is getting weaker, calmer, and with less details.

FIGURE 4-11. Screenshots of the *Terra Dynamic* simulation from different viewpoints using ThalesView.

4.3.3 Integration with Game Engines

In order to make the reproduction flexibly compatible with different rendering solutions, we integrated the external modules into the kernel and built it as a plugin (MAC.dll) that can be used in wide-ranging game engines. In this manner, the selected game engine can operate its native services, fed with data from the MAC.

The approach has been proved technically functional in Unity3D and CryENGINE. Given that both SoundController and CryENGINE or Unity3D choose FMOD as their audio solution, we enable a reallocation of FMOD in SoundController, forcing the SoundController to use the FMOD Event System that is employed by the game engine to eliminate the system conflict.

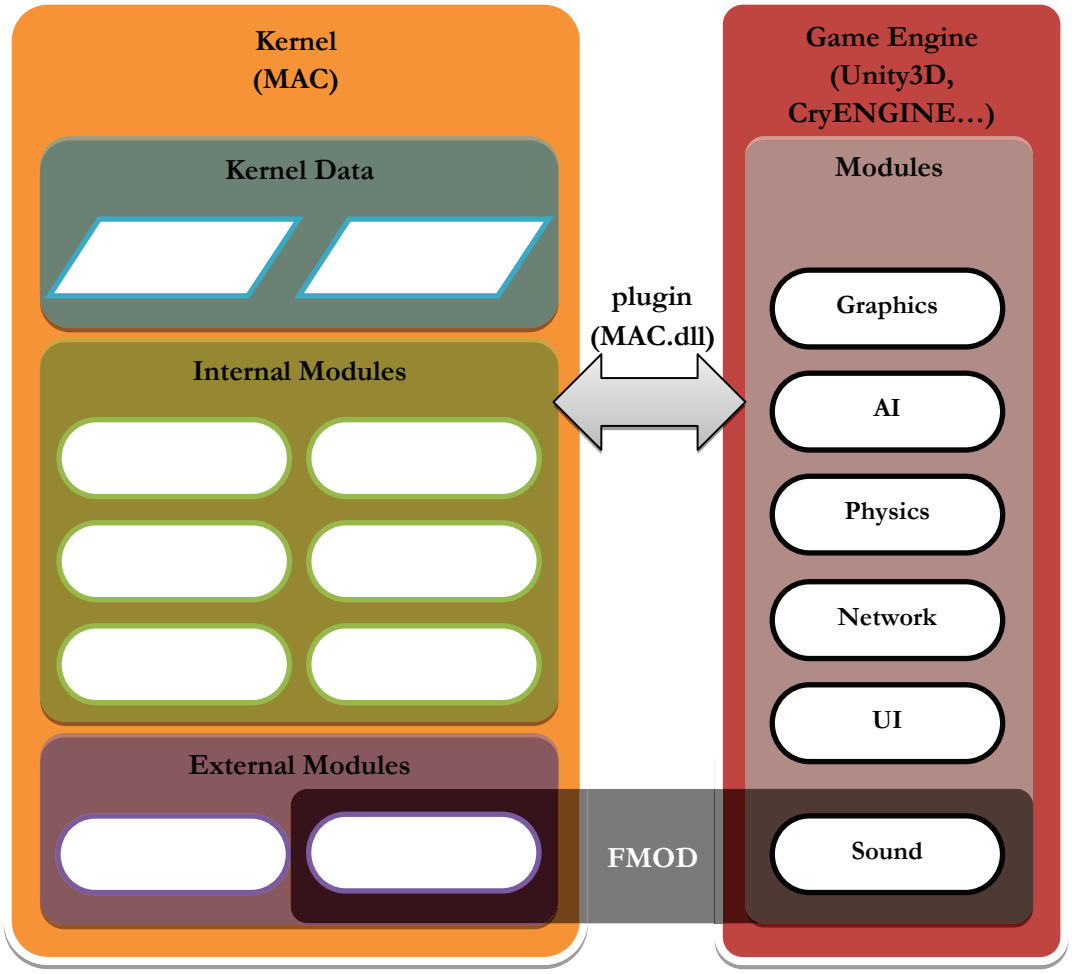


FIGURE 4-12. Integration of MAC and SoundController with the game engine.

Thanks to this integration, the MAC (and most important, the sound engine) is portable so that the virtual city of *Terra Dynamica* is now able to be implemented on game engines chosen from the developers. What is more is that the simulation is not limited anymore to the prospects provided by MAC, but it can also be a fusion of typed models from the *Terra Dynamica* with the assets (e.g., meshes, materials, animations, bounding spheres, sounds) from the game engine. Also, the game engine will have an access to the SoundController since they share the same FMOD sound system.

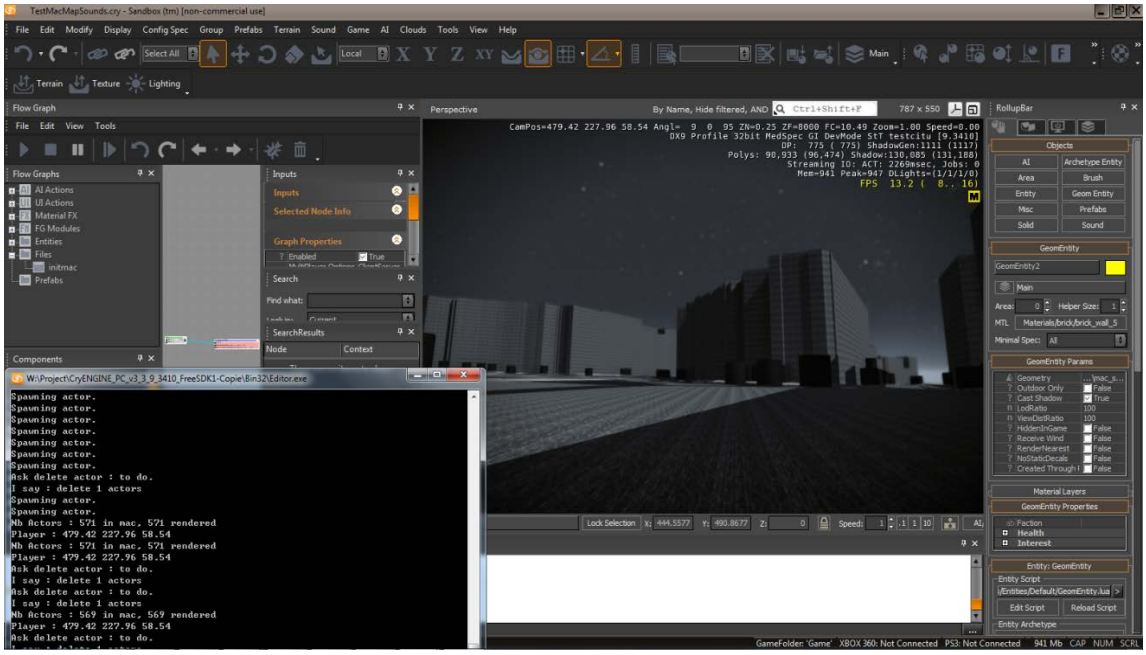


FIGURE 4-13. The working environment of the *Terra Dynamic* simulation using CryENGINE.

4.4 Summary

In this chapter, we validate the feasibility of using the COLLADA standard to describe sound in virtual cities. The examination takes in three stages: the early prototype implementation, sound design (the use of extensive audio formats), and the development of a 3D sound engine.

In the early phase of the development, a several number of prototypes were made with assorted graphics renderers or game engines. In these prototypes, OpenAL or FMOD sound

system was selected for audio rendering and playback control. This early work aided us evaluating the compatibility of COLLADA with numerous game engines and demonstrating the accessibility and control of the COLLADA audio content.

Furthermore, we examined the usefulness of including the extensive sound descriptions in COLLADA. One of the suggested sound design pipelines established a close interchange between FMOD events and COLLADA files and gave back the sound designers more creative power within interactive projects. The sound designers we have worked with have found that with this unique format for a rich sound design, the production chain is simpler and their work is less dependent upon the integration done by the programmers. Another proposal was to merge COLLADA sound description with Pure Data sound production, making an efficient way of quickly building interactive audio scenes. Scripting and external linking capabilities added in our proposal would permit to describe even richer functionalities. Regarding the sound design aspects, further developments should be conducted in terms of sound granulation. Random values of pitch shifting, grain amplitude, densities and envelopes would increase the sensation of non-repetitive soundscape. So far, the sound description properties applied for sample playback functions. Adding descriptive parameters applicable to pure synthesis, such as additive synthesis, would allow extended design functionalities. These parameters could consist in high level description of spectral characteristics. Also, semantic descriptors could be sonified with corpus based concatenative synthesis. The aim is to create generic Pure Data concatenative patches dedicated to classes of `<audio_source>` nodes, for example, footsteps and engines. The experiment of sound design is a collaboration work done with the sound designers Tiger et al. More information will be available in Tiger's Ph.D. Thesis [TPHD].

Finally, we have put together the experience we gained in the previous steps to build a 3D sound engine (SoundController) with the support of COLLADA audio capability for urban sound simulations. The sound engine is created based on the FMOD sound system and produces dynamic city background ambience by Pure Data. This portable sound engine is a plugin that can be integrated with the *Terra Dynamica*'s kernel API (MAC), using ThalesView or other game engines for graphics rendering. An advanced demonstration of a virtual city has been made by a technical integration of MAC, SoundController, and CryENGINE. The progressive program covers the functionalities of SoundController, which is dependent on COLLADA to describe acoustic behaviors of virtual actors, and keeps the option of utilizing the advantages and features of CryENGINE for developing interactive applications of virtual cities.

Indeed, it is difficult to formally validate our proposal of sound in COLLADA since there is no evident metrics that can be managed to compare it with other traditional solutions. Nevertheless, we claim that our proposal is economic in terms of resources needed for implementing an application. The audio profile is organized together with profiles in other categories inside a unified scene description language, which allows the sharing of data to reduce the duplications.

The simulation core of *Terra Dynamica* joint with our sound engine will serve a platform for developing applications in a wide range of domains, such as video games, simulations of transportation, security, and military. From the delivery of these applications, we will have valuable information about how the sound engine based on the COLLADA audio description can be helpful for building an interactive virtual 3D audiovisual scene. The project *Terra Dynamica* had presented a showcase of virtual Paris city in *Futur en Seine* 2011 and 2012 with the sound supported by the SoundController. The presentation had been proved a success, appreciating sounds for delivering a more convincing immersive atmosphere to create a compelling experience in virtual environments.

This page intentionally left blank.

CHAPTER 5.

Conclusion and Future Work

This thesis presents a formal representation of sound in virtual cities using the scene description language COLLADA.

In this scope, we introduce an object-oriented tree-like audio scene based on the knowledge of mainly MPEG-4 AudioBIFS and other referred scene description formats. The COLLADA audio scene is a composition of auditory nodes hierarchically representing a sound signal chain. From bottom to top, it describes digital-signal-processing manipulations. Each child element outputs the data to one or more parent nodes. The leaves and intermediate nodes of the audio tree do not play sounds themselves, but only the result in the root will be presented to the listener. Most of the acoustic elements of COLLADA inherit the name and functionality from those of MPEG-4 or AAML standards, whereas some are improved with new features or are an entirely inventive. The sound in COLLADA facilitates the import and process of both common audio file formats (e.g., WAV, MP3, OGG) and designer-generated sounds (e.g., FMOD events, Pure Data patches) with the support of self-defined parameters.

Moreover, because the process of level of detail for audio is similar in spirit to that in computer graphics, the same facility on the sound descriptions is expected in order to provide the applications with more information for classifying sound sources. The soundscape layers of sound in COLLADA are specified accordingly with the utilization of the existing techniques (i.e., technique, scene, layer, and extra) used for demonstrating

multiple LODs for visual and physics contents. This mechanism not only follows the standard procedure of analyzing and processing the COLLADA content, but also adopts the strategy in semantic filtering. In addition, even though the default scene is being defined inside the COLLADA documents in order to provide applications and tools an indication of the primary scene to load, application writers have the freedom to switch scenes from one to another and can always alter the details of the scene presented at runtime.

The above structure built upon the basic sound capability constructs the static part of the COLLADA auditory scene like how the other categories of COLLADA sub-scenes (i.e., visual, physics, and kinematics simulations) are referred to static means of the objects in the scene but not their dynamic behaviors. Considering the dynamic nature of sound in urban environment, the enhanced COLLADA standard with sound capability includes as well dynamic representations. Two solutions are applied to achieve the desired result: Firstly we specify an event-trigger condition, incorporating with the common function-call method to transfer sound from one state to another. This technique takes advantage of the extensibility of COLLADA, retaining its hierarchical scene graph structure. Beyond the fixed-function state, the second solution relies on "programmable units". This can be done by embedding a block of inline source code or referencing to an external resource that programs complex dynamic part of COLLADA sound description. However, the capability of dynamic description is not expected to change the position of COLLADA as an intermediate language of choice in the production pipeline. Although the enriched COLLADA defines the format of 3D assets, the decision of realizing the content is still made by the application writers.

The proposed audio scene covers the essential atoms and libraries for the standard, including sound sources, acoustic parameters, listening point, basic DSP functions and complex DSP constructors; incomplete implementation are considered as allowed extensions. In accordance with the principle of resources reusability and exchangeability, elements in different categories can be associated together and sharing data in our design, which is cost-effective for preventing duplications of assets.

Indeed, it is not easy to precisely evaluate the proposition of COLLADA audio since there is no evident metrics that can be used to compare it with other traditional solutions. Nonetheless, we have implemented numerous applications to demonstrate the technical feasibility. The simple sound description in COLLADA was validated through some early experiments with various game engines and sound APIs. More advanced demonstrations have been done by an implementation of a 3D sound engine constructed upon the FMOD sound system. The 3D sound engine has been employed in the project *Terra Dynamica*, a

dynamic virtual city, consolidated with the *Terra Dynamica* core kernel API and its authorized graphics rendering engine (ThalesView) or developer-selected game engines (e.g., CryENGINE, Unity3D). Furthermore, the sound designers we have worked with have found that with this unique format for a rich sound design, the production chain is simpler and their work is less dependent upon the integration done by the programmers. There is also a plan to submit our proposal of sound branch of COLLADA to the Khronos Group, which is the official organization in charge of the COLLADA standard.

The 3D sound engine we have developed is a programming project with several thousands of lines of code. Nevertheless, this is just a basic proof of concept since we have not technically implemented and optimized all the functionalities that we have proposed for the COLLADA audio scene. In the coming future, we expect more specific descriptions for sound analyzing/synthesizing methods as well as recording/rendering instructions will be included in the acoustic standard. Furthermore, the soundscape layering skill we introduce in COLLADA does not give much hint for the decision making process of sound sources clustering. Although the algorithm of sound level of detail is out of the scope of this thesis, it could make an interesting question of whether if the information of SLOD approaches (in physical or psychological aspect) can be provided in this kind of scene-graph structure and how; and if not, how far can this idea be pushed? Despite it was said that scene graph had hit its limit of describing complex dynamic behaviors and interactivities of objects, we hope our proposal of the merge of the event-driven programming methods (i.e., event detection/selection and event handling) with scene graph hierarchy opens another door to solve this issue. In addition, the scripting language we propose has only scratched the surface with the concept. As Arnaud and Barnes have suggested, more fixed-function pipeline states will be consolidated and replaced by programmable units in the future [AB06]. It is expectable that a more well-established scripting language is required and will play a bigger role in the future standard.

Sound in COLLADA is anticipated to be a use for other forms of interactive applications, such as interactive music or art installations. In addition, COLLADA uses web technologies ("XML" for the syntax and "URI" for the resource identifiers), making it a good fit for web-based implementations. Further than being applied in native interactive applications (e.g., computer or console games), it is well suited for 3D web content delivery [T10]. COLLADA and X3D (a universal delivery format for web applications) can be used together as a powerful tool set for developing their Web and enterprise applications [AP07]. Some experiments have proved that a partial COLLADA parser can be implemented in

JavaScript and suggested binding the two Khrons standards – COLLADA and WebGL[3] together for a promising web experience of 3D imaging. It is thus expectable that in the near future an analogous web standard that provides the native access to the hardware-accelerated audio will be defined (e.g., OpenAL or OpenSL ES for the Web) and can be allied with the COLLADA sound description to enable immersive 3D web sound delivery. As support for HTML5 and relative technologies grows, it is likely more and more 3D web applications will be willing to use COLLADA directly as an input format.

As soon as the standard of sound in COLLADA is established, it can always be extended by adding advanced features, for instance the "acoustic materials" (acoustical colors) that specifies auditory characteristics (e.g., reflection and transmission properties) in order to model acoustic effects of the environment in a physical approach.

In spite of that options are provided to interlink the audio scene with the visual one in COLLADA, the connection is not mandatory. The proposed interactive sound description is possible to be standalone as a pure audio capability document. The manuscript that contains sound only can be beneficial for engaging non-diegetic sounds or surrounding ambiences. In this context, our proposal sets up a foundation of a formal way to describe sounds in virtual cities. The knowledge and skills are transferable thus it opens up new possibilities to standardize audio scene description in any text format and not limited to the COLLADA schema.

[3] The **W**eb **G**raphics **L**ibrary (WebGL), based on OpenGL ES 2.0, is a low-level JavaScript API for rendering interactive 2D and 3D graphics for web applications without the use of plugins. It is exposed through the HTML5 Canvas element as Document Object Model interfaces.

APPENDIX A

MPEG-4 AudioBIFS Nodes

This appendix covers the elements that compose MPEG-4 AudioBIFS.

A.1. AudioBIFS Version 1

A.1.1.AudioBuffer

The `AudioBuffer` node records a segment of sound to be used in interactive playback. The short audio samples recorded in this node are made available to the SA (**S**tructure **A**udio) decoder attached to the `AudioSource` for use in the synthesis process.

```
AudioBuffer {
   exposedField    MFNode      children            []
   exposedField    SFBool      loop                FALSE
   exposedField    SFFloat     pitch               1.0
   exposedField    SFTime      startTime           0
   exposedField    SFTime      stopTime            0
   exposedField    SFFloat     length              0.0
   Field           SFInt32     num_chan            1
   Field           MFInt32     phaseGroup          [1]
   eventOut        SFTime      duration_changed
   eventOut        SFBool      isActive
}
```

A.1.2. AudioClip

The `AudioClip` node in MPEG-4 AudioBIFS is inherited from VRML. The use of this node should be only for VRML backward compatible applications.

```
AudioClip {
   exposedField    SFString    description         ""
   exposedField    SFBool      loop                FALSE
   exposedField    SFFloat     pitch               1.0
   exposedField    SFTime      startTime           0
   exposedField    SFTime      stopTime            0
   exposedField    MFString    url                 []
   eventOut        SFTime      duration_changed
   eventOut        SFBool      isActive
}
```

A.1.3. AudioDelay

The `AudioDelay` node allows child sounds to be temporally delayed or advanced for synchronization. The `delay` field specifies the length of delay.

```
AudioDelay {
   eventIn         MFNode      addChildren
   eventIn         MFNode      removeChildren
   exposedField    MFNode      children            []
   exposedField    SFTime      delay               0
   Field           SFInt32     num_chan            1
   Field           MFInt32     phaseGroup          []
}
```

A.1.4. AudioFX

The `AudioFX` node allows arbitrary signal-processing effects defined via structure audio tools to be applied to the inputs. The `orch` command buffer contains a tokenized block of signal-processing written in SAOL (**S**tructure **A**udio **O**rchestra **L**anguage) [SV99]; the script command buffer may contain a tokenized score for the given orchestra written in SASL (**S**tructure **A**udio **S**core **L**anguage), if needed. The `params` field allows scene-graph-level interaction or events to affect the process of sound generation in orchestra.

```
AudioFX {
   eventIn         MFNode          addChildren
```

```
   eventIn          MFNode             removeChildren
   exposedField     MFNode             Children           []
   Field            SFCommandBuffer    Orch               ""
   Field            SFCommandBuffer    Score              ""
   exposedField     MFFloat            Params             []
   Field            SFInt32            num_chan           1
   Field            MFInt32            phaseGroup         []
}
```

A.1.5. AudioMix

The AudioMix node is used to mix M channels of sounds into N channels through a simple matrix calculation. One is able to control the proportions of the input sounds that are mixed to the output by manipulating the entries specified in the mixing matrix.

```
AudioMix {
   eventIn          MFNode      addChildren
   eventIn          MFNode      removeChildren
   exposedField     MFNode      children           []
   exposedField     SFInt32     numInputs          1
   exposedField     MFFloat     matrix             []
   Field            SFInt32     num_chan           1
   Field            MFInt32     phaseGroup         []
}
```

A.1.6. AudioSource

The AudioSource node adds a sound source into an AudioBIFS scene. It attaches an audio decoder, of which identified by the MPEG-4 audio standard, to scene graph. AudioSource shares the same time-sensitive and playback-control fields with AudioClip. The numChan field specifies how many channels of audio are in the decoded bitstream. The phaseGroup flag determines whether or not there are important phase relationships between multiple channels of source sound.

```
AudioSource {
   eventIn          MFNode      AddChildren
   eventIn          MFNode      removeChildren
   exposedField     MFNode      Children           []
   exposedField     MFString    url                []
   exposedField     SFFloat     Pitch              1.0
   exposedField     SFFloat     Speed              1.0
```

```
   exposedField    SFTime      StartTime          0
   exposedField    SFTime      StopTime           0
   field           SFInt32     NumChan            1
   field           MFInt32     PhaseGroup         []
}
```

A.1.7. AudioSwitch

The AudioSwitch node selects a subset of the input channels to pass through by setting the element to 1 in the whichChoice field. For example, [0 1 1 0] means the second and third channels can pass through.

```
AudioSwitch {
   eventIn         MFNode      AddChildren
   eventIn         MFNode      removeChildren
   exposedField    MFNode      Children           []
   exposedField    MFInt32     WhichChoice        []
   field           SFInt32     NumChan            1
   field           MFInt32     PhaseGroup         []
}
```

A.1.8. ListeningPoint

The ListeningPoint node indicates the spatial position and orientation of the virtual listener in a scene. If no ListeningPoint is given, the listener is located at the active ViewPoint position.

```
ListeningPoint {
   eventIn         SFBool      set_bind
   exposedField    SFBool      Jump               TRUE
   exposedField    SFRotation  Orientation        0 0 1 0
   exposedField    SFVec3f     Position           0 0 10
   field           SFString    Description        ""
   eventOut        SFTime      BindTime
   eventOut        SFBool      IsBound
}
```

A.1.9. Sound

The Sound node in MPEG-4 AudioBIFS is inherited from VRML Yet, whereas the VRML Sound can only refer to an AudioClip which is a sample sound file given through a url, the Sound node in MPEG-4 supports a mix of sound samples (AudioMix), symbolic sound files (MIDI), real-time sampled or symbolic streams of sounds. At each step of the mix, some audio effects can be applied to the corresponding stream (through the AudioFX nodes), depending on local or global parameters, properties of the scene, as well as real time events. For example, a Sound node could attach a join of a synthetic wind with a sampled music, the level of which would be according to the current level of dramaturgy (on-the fly computed parameter).

```
Sound {
   exposedField    SFVec3f     Direction         0 0 1
   exposedField    SFFloat     Intensity         1.0
   exposedField    SFVec3f     Location          0 0 0
   exposedField    SFFloat     MaxBack           10.0
   exposedField    SFFloat     MaxFront          10.0
   exposedField    SFFloat     MinBack           1.0
   exposedField    SFFloat     MinFront          1.0
   exposedField    SFFloat     Priority          0.0
   exposedField    SFNode      Source            []
   Field           SFBool      Spatialize        TRUE
}
```

A.1.10.Sound2D

The Sound2D node defines a BIFS sound in a 2D scene. The semantics are identical to those in the Sound, except that Sound2D is unavailable in 3D context.

```
Sound2D {
   exposedField    SFFloat     intensity          1.0
   exposedField    SFVec2f     location           0 0
   exposedField    SFNode      source             []
   Field           SFBool      spatialize         TRUE
}
```

A.2. AudioBIFS Version 2 (AABIFS)

A.2.1. AcousticMaterial

The AcousticMaterial node specifies reflection and transmission properties for surfaces of an object. It is similar to the Material node for visual objects but has additional acoustical properties.

```
AcousticMaterial {
   field            SFFloat     reffunc            0.0
   field            SFFloat     transfunc          1.0
   field            MFFloat     refFrequency       []
   field            MFFloat     transFrequency     []
   exposedField     SFFloat     ambientIntensity   0.2
   exposedField     SFColor     diffuseColor       0.8 0.8 0.8
   exposedField     SFColor     emissiveColor      0 0 0
   exposedField     SFFloat     shininess          0.2
   exposedField     SFColor     specularColor      0 0 0
   exposedField     SFFloat     transparency       0
}
```

A.2.2. AcousticScene

The AcousticScene node restricts the acoustic rendering process to a 3D listening area with the center and size fields. Only when the virtual listener locates within the region, the surfaces on that area will be modeled. AcousticScene contains several reverb parameters employed to apply late reverberation to sounds.

```
AcousticScene {
   Field            SFVec3f     center             0 0 0
   Field            SFVec3f     size               -1 -1 -1
   Field            MFTime      reverbTime         0
   Field            MFFloat     reverbFreq         1000.0
   exposedField     SFFloat     reverbLevel        0.4
   exposedField     SFTime      reverbDelay        0.5
}
```

A.2.3. DirectiveSound

The DirectiveSound node defines a directivity (can be frequency-dependant) sound source. This allows a finer control of the modeling of sound propagation between the source and the listening point that includes transmission in the medium (distance dependant attenuation, air absorption, and propagation delay), modeling of acoustic effect of the environment (sound reflections, propagation through objects, and reverberation). The physical approach to acoustic environment modeling is set to be a default. Yet by specifying a PerceptualParameters node within the perceptualParameters field of a DirectiveSound, the PerceptualParameters node can be attached to the DirectiveSound node to apply as well the perceptual approach.

```
DirectiveSound {
   exposedField    SFNode      source                []
   exposedField    SFNode      perceptualParameters  []
   exposedField    SFVec3f     direction             0 0 1
   exposedField    SFFloat     intensity             1.0
   exposedField    SFVec3f     location              0 0 0
   exposedField    SFBool      spatialize            TRUE
   exposedField    SFBool      roomEffect            FALSE
   field           MFFloat     angles                0.0
   field           MFFloat     directivity           1.0
   field           MFFloat     frequency             []
   field           SFFloat     speedOfSound          340.0
   field           SFFloat     distance              1000.0
   field           SFBool      useAirabs             FALSE
}
```

A.2.4. PerceptualParameters

The PerceptualParameters node makes possible to be attached with a DirectiveSound sound source for the simulation of virtual room effects by the perceptual approach to acoustic environment modeling.

```
PerceptualParameters {
   exposedField    SFFloat     sourcePresence        1.0
   exposedField    SFFloat     sourceWarmth          1.0
   exposedField    SFFloat     sourceBrilliance      1.0
   exposedField    SFFloat     roomPresence          0.0
   exposedField    SFFloat     runningReverberance   1.0
   exposedField    SFFloat     Envelopment           0.0
   exposedField    SFTime      lateReverberance      1.0
   exposedField    SFFloat     heaviness             1.0
```

```
   exposedField    SFFloat     liveness              1.0
   exposedField    MFFloat     omniDirectivity       1.0
   exposedField    MFFloat     directFilterGains     1.0 1.0 1.0
   exposedField    MFFloat     inputFilterGains      1.0 1.0 1.0
   exposedField    SFFloat     refDistance           1.0
   exposedField    SFFloat     freqLow               250.0
   exposedField    SFFloat     freqHigh              4000.0
   exposedField    SFTime      timeLimit1            0.02
   exposedField    SFTime      timeLimit2            0.04
   exposedField    SFTime      timeLimit3            0.1
   exposedField    SFFloat     modalDensity          0.8
}
```

A.3. AudioBIFS Version 3

A.3.1. AdvancedAudioBuffer

The AdvancedAudioBuffer node is used to correct, extend and replace AudioBuffer.

```
AdvancedAudioBuffer {
   eventIn         MFNode      addChildren
   eventIn         MFNode      removeChildren
   exposedField    MFNode      children              []
   exposedField    SFBool      loop                  FALSE
   exposedField    SFFloat     pitch                 1.0
   exposedField    SFTime      startTime             0
   exposedField    SFTime      stopTime              0
   exposedField    SFTime      startLoadTime         0
   exposedField    SFTime      stopLoadTime          0
   exposedField    SFInt32     loadMode              0
   exposedField    SFInt32     numAccumulatedBlocks  0
   exposedField    SFInt32     deleteBlock           0
   exposedField    SFInt32     playBlock             0
   exposedField    SFFloat     length                0.0
   Field           SFInt32     numChan               1
   Field           MFInt32     phaseGroup            [1]
   eventOut        SFTime      duration_changed
   eventOut        SFBool      isActive
}
```

A.3.2. AudioChannelConfig

The AudioChannelConfig node describes how to pass one or more channels of input sound samples through into mono, stereo, or multi-channel outputs. Since there is an ambiguous definition between the phaseGroup field and the AudioChannelConfig node, the phaseGroup fields specified in other nodes shall be ignored.

```
AudioChannelConfig {
   eventIn          MFNode      addChildren
   eventIn          MFNode      removeChildren
   exposedField     MFNode      Children                    []
   exposedField     SFInt32     generalChannelFormat        0
   exposedField     SFInt32     FixedPreset                 0
   exposedField     SFInt32     fixedPresetSubset           0
   exposedField     SFInt32     fixedPresetAddInf           0
   exposedField     MFInt32     channelCoordinateSystems    []
   exposedField     MFFloat     channelSoundLocation        []
   exposedField     MFInt32     channelDirectionalPattern   []
   exposedField     MFVec3f     channelDirection            []
   exposedField     SFInt32     ambResolution2D             1
   exposedField     SFInt32     ambResolution3D             0
   exposedField     SFInt32     ambEncodingConvention       0
   exposedField     SFFloat     ambNfcReferenceDistance     1.5
   exposedField     SFFloat     ambSoundSpeed               340.0
   exposedField     SFInt32     ambArrangementRule          0
   exposedField     SFInt32     ambRecombinationPreset      0
   exposedField     MFInt32     ambComponentIndex           []
   exposedField     MFFloat     ambBackwardMatrix           []
   exposedField     MFInt32     ambSoundfieldResolution     []
   field            SFInt32     NumChan                     0
}
```

A.3.3. AudioFXProto

The AudioFXProto node implements a tailored subset of functionality available through the AudioFX node. While the AudioFX node normally requires a Structured Audio implementation that is high computational power, this subset allows player without Structured Audio capability to use the standard audio effects identified by means of predefined values of the protoName field. The standard audio effects include PROTO audioChorus, audioCompressor, audioEcho, audioEqualizer, audioFilter, audioFlange, audioNaturalReverb, audioReverb, audioSpeedChange, audioStereoBase, and audioVirtualStereo.

```
AudioFX {
PROTO audio "Name" [
   exposedField    MFNode            audioFXChildren     []
   exposedField    MFFloat           audioFXParams       []
   exposedField    SFInt32           audioFXnumChan      1
   exposedField    MFInt32           audioFXphaseGroup   []
   ]
DEF "Name" AudioFX {
   eventIn         MFNode            addChildren
   eventIn         MFNode            removeChildren
   exposedField    MFNode            Children            []
   field           SFCommandBuffer   Orch                ""
   field           SFCommandBuffer   Score               ""
   exposedField    MFFloat           Params              []
   field           SFInt32           num_chan            1
   field           MFInt32           phaseGroup          []
   }
}
```

A.3.4. SurroundingSound

The SurroundingSound node attaches sound to a scene, thereby giving it spatial qualities and relating it to the visual content of the scene. This includes multi-channel signals that cannot be spatially transformed with the other sound nodes because of their restrictions to the specification of the phaseGroup field and the spatialize field.

```
SurroundingSound {
   exposedField    SFNode      Source                []
   exposedField    SFFloat     Intensity             1.0
   exposedField    SFVec3f     Location              0 0 0
   exposedField    SFFloat     Distance              0.0
   exposedField    SFFloat     distortionFactor      0.0
   exposedField    SFRotation  Orientation           0 0 1 0
   exposedField    SFBool      isTransformable       TRUE
}
```

A.3.5. Transform3DAudio

The Transform3DAudio node enables 3D audio subgraph to be adapted in 2D scenes.

```
Transform3DAudio {
   eventIn         MFNode      AddChildren
   eventIn         MFNode      removeChildren
   exposedField    MFNode      Children                  []
```

```
   exposedField    SFFloat     thirdCenterCoordinate        0.0
   exposedField    SFVec3f     rotationVector               0 0 1
   exposedField    SFFloat     thirdScaleCoordinate         0.0
   exposedField    SFVec3f     scaleOrientationVector       0 0 1
   exposedField    SFFloat     thirdTranslationCoordinate   0.0
   exposedField    SFRotation  coordinateTransform          1 0 0 -π/2
}
```

A.3.6. WideSound

The `WideSound` node relates an audio subtree to the rest of BIFS. It attaches sound and to a scene so that the sound is spatialized with a determinable widening for non-phase-related signals from its descendant audio nodes.

```
WideSound {
   exposedField    SFNode      Source                 []
   exposedField    SFNode      perceptualParameters   []
   exposedField    SFVec3f     Direction              0 1 0
   exposedField    SFFloat     Intensity              1.0
   exposedField    SFVec3f     Location               0 0 0
   exposedField    SFBool      Spatialize             TRUE
   exposedField    SFBool      RoomEffect             FALSE
   exposedField    SFInt32     Shape                  0
   exposedField    MFFloat     Size                   []
}
```

This page intentionally left blank.

APPENDIX B

COLLADA Audio Reference

This appendix covers the elements that compose COLLADA Audio.

B.1. audio_source

Attributes

Name/example	Type	Description
id	xs:ID	A text string containing the unique identifier of this element. This value must be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

Child Elements

Name/example	Description	Default	Occurrences
<loop>	Specifies how many times that the sound shall restart once playback is over. When set to -1, the sound will loop forever.	0	0 or 1
<pitch>	Controls the playback pitch. It is specified as a ratio, where 1 indicates the original bitstream pitch; values other than 1 indicate pitch-shifting by the given ratio.	1.0	0 or 1
<speed>	Controls the playback speed. It is specified	1.0	0 or 1

	as a ratio, where 1 indicates the original speed; values other than 1 indicate multiplicative time-scaling by the given ratio (i.e. 0.5 specifies twice as fast).		
<start_time>	Specifies a time at which to start the audio playing.	0	0 or 1
<stop_time>	Specifies a time at which to turn off the sound. When set to 0, the sound continues until it is finished or plays forever.	0	0 or 1
<num_chan>	Describes how many channels of audio are in the decoded bitstream.	1	0 or 1
<phase_group>	Specifies whether or not there are important phase relationships between multiple channels of source sound. The values in the array divide the channels of audio into groups; if phase_group[i] = phase_group[j] then channel i and j are phase related. Channels for which the phase_group value is 0 are not related to any other channel.	None	0 or 1
<newparam>	Creates a new parameter from a constrained set of types.	N/A	0 or more
<init_from>	Initializes the sound from a URL (for example, a file).	None	0 or 1
<instance_audio_dsp>	Instantiates an <audio_dsp> element.	N/A	0 or 1

B.1.1. instance_audio_source

Attributes

Name/example	Type	Description
sid	sid_type	A text string value containing the scoped identifier of this element. This value must be unique within the scope of the parent element. Optional.
name	xs:token	The text string name of this element. Optional.
url	xs:anyURI	The URL of the location of the <audio_source> element to instantiate. Required. Refers to a local instance using a relative URI fragment identifier that begins with the "#" character. The fragment identifier is an XPointer shorthand pointer that consists of the URI of the element to instantiate.

		Refers to an external reference using an absolute or relative URL when it contains a path to another resource.

Child Elements

Name/example	Description	Default	Occurrences
<setparam>	Assigns a concrete value to a predefined parameter of the instantiated audio source.	N/A	0 or more
<extra>	Provides additional information that is not defined in COLLADA about or related to the <instance_audio_source> element.	N/A	0 or more

B.2. audio_buffer

Child Elements

Name/example	Description	Default	Occurrences
<loop>	Specifies how many times that the sound shall restart once playback is over. When set to -1, the sound will loop forever.	0	0 or 1
<length>	Specifies how much sound to record.	0.0	0 or 1
<pitch>	Controls the playback pitch. It is specified as a ratio, where 1 indicates the original bitstream pitch; values other than 1 indicate pitch-shifting by the given ratio.	1.0	0 or 1
<start_time>	Specifies a time at which to activate playback.	0	0 or 1
<stop_time>	Specifies a time at which to turn off the sound.	0	0 or 1
<num_chan>	Specifies the number of channels of audio output by this node.	1	0 or 1
<phase_group>	Specifies the phase relationships among the various output channels.	[1]	0 or 1
<duration_changed>	Specifies the duration of the audio object.	N/A	0 or 1
<is_active>	Indicates whether playback is active or not.	N/A	0 or 1
<instance_audio_source>	Instantiates an <audio_source> element.	N/A	0 or 1
<instance_audio_buffer>	Instantiates an <audio_buffer> element.	N/A	0 or 1
<instance_audio_delay>	Instantiates an <audio_delay> element.	N/A	0 or 1
<instance_audio_mix>	Instantiates an <audio_mix> element.	N/A	0 or 1
<instance_audio_switch>	Instantiates an <audio_switch> element.	N/A	0 or 1

<instance_audio_dsp>	Instantiates an <audio_dsp> element.	N/A	0 or 1

B.3. audio_delay

Child Elements

Name/example	Description	Default	Occurrences
<delay>	Specifies the length of delay.	0	0 or 1
<num_chan>	Specifies the number of channels of audio output by this node.	1	0 or 1
<phase_group>	Specifies the phase relationships among the various output channels.	None	0 or 1
<instance_audio_source>	Instantiates an <audio_source> element.	N/A	0 or 1
<instance_audio_buffer>	Instantiates an <audio_buffer> element.	N/A	0 or 1
<instance_audio_delay>	Instantiates an <audio_delay> element.	N/A	0 or 1
<instance_audio_mix>	Instantiates an <audio_mix> element.	N/A	0 or 1
<instance_audio_switch>	Instantiates an <audio_switch> element.	N/A	0 or 1
<instance_audio_dsp>	Instantiates an <audio_dsp> element.	N/A	0 or 1

B.4. audio_mix

Child Elements

Name/example	Description	Default	Occurrences
<num_inputs>	Specifies the number of input channels. It should be the sum of the number of channels of the children.	1	0 or 1
<matrix>	Specifies the mixing matrix which relates the inputs to the outputs. matrix is an unrolled num_inputs x num_chan matrix .	None	0 or 1
<num_chan>	Specifies the number of channels of audio output by this node.	1	0 or 1
<phase_group>	Specifies the phase relationships among the various output channels.	None	0 or 1
<instance_audio_source>	Instantiates an <audio_source> element.	N/A	0 or 1
<instance_audio_buffer>	Instantiates an <audio_buffer> element.	N/A	0 or 1
<instance_audio_delay>	Instantiates an <audio_delay> element.	N/A	0 or 1
<instance_audio_mix>	Instantiates an <audio_mix> element.	N/A	0 or 1
<instance_audio_switch>	Instantiates an <audio_switch> element.	N/A	0 or 1
<instance_audio_dsp>	Instantiates an <audio_dsp> element.	N/A	0 or 1

B.5. audio_switch

Child Elements

Name/example	Description	Default	Occurrences
<random>	Specifies how many sources would be randomly selected to pass though. The default is -1, meaning particular passed-through channels should be indicated in the <which_choice> element. If the value of random is a positive integer or zero, <which_choice> would have no effect.	-1	0 or 1
<which_choice>	Specifies which channels should be passed through.	None	0 or 1
<num_chan>	Specifies the number of channels of audio output by this node.	1	0 or 1
<phase_group>	Specifies the phase relationships among the various output channels.	None	0 or 1
<instance_audio_source>	Instantiates an <audio_source> element.	N/A	0 or 1
<instance_audio_buffer>	Instantiates an <audio_buffer> element.	N/A	0 or 1
<instance_audio_delay>	Instantiates an <audio_delay> element.	N/A	0 or 1
<instance_audio_mix>	Instantiates an <audio_mix> element.	N/A	0 or 1
<instance_audio_switch>	Instantiates an <audio_switch> element.	N/A	0 or 1
<instance_audio_dsp>	Instantiates an <audio_dsp> element.	N/A	0 or 1

B.6. sound

Child Elements

Name/example	Description	Default	Occurrences
<direction>	Orients the elliptical audible field.	0 0 1	0 or 1
<intensity>	Defines the loudness of the sound. The value of the field ranges from 0.0 to 1.0, and this value specifies a factor that scales the normalized sample data of the sound source during playback.	1.0	0 or 1
<location>	Specifies the location of the sound in the 3D scene.	0 0 0	0 or 1
<max_back>	Defines an elliptical audible field.	10.0	0 or 1
<max_front>	Defines an elliptical audible field.	10.0	0 or 1
<min_back>	Defines an elliptical audible field.	1.0	0 or 1
<min_front>	Defines an elliptical audible field.	1.0	0 or 1

<spatialize>	Specifies whether the sound should be spatialized in the 3D scene.	True	0 or 1
<instance_audio_source>	Instantiates an <audio_source> element.	N/A	0 or 1
<instance_audio_buffer>	Instantiates an <audio_buffer> element.	N/A	0 or 1
<instance_audio_delay>	Instantiates an <audio_delay> element.	N/A	0 or 1
<instance_audio_mix>	Instantiates an <audio_mix> element.	N/A	0 or 1
<instance_audio_switch>	Instantiates an <audio_switch> element.	N/A	0 or 1
<instance_audio_dsp>	Instantiates an <audio_dsp> element.	N/A	0 or 1

B.7. sound_2d

Child Elements

Name/example	Description	Default	Occurrences
<intensity>	Defines the loudness of the sound. The value of the field ranges from 0.0 to 1.0, and this value specifies a factor that scales the normalized sample data of the sound source during playback.	1.0	0 or 1
<location>	Specifies the location of the sound in the 2D scene.	0 0	0 or 1
<spatialize>	Specifies whether the sound should be spatialized in the 2D scene.	True	0 or 1
<instance_audio_source>	Instantiates an <audio_source> element.	N/A	0 or 1
<instance_audio_buffer>	Instantiates an <audio_buffer> element.	N/A	0 or 1
<instance_audio_delay>	Instantiates an <audio_delay> element.	N/A	0 or 1
<instance_audio_mix>	Instantiates an <audio_mix> element.	N/A	0 or 1
<instance_audio_switch>	Instantiates an <audio_switch> element.	N/A	0 or 1
<instance_audio_dsp>	Instantiates an <audio_dsp> element.	N/A	0 or 1

B.8. sound_ext

Child Elements

Name/example	Description	Default	Occurrences
<newparam>	Creates a new parameter from a constrained set of types.	N/A	0 or more
<instance_audio_source>	Instantiates an <audio_source> element.	N/A	0 or 1
<instance_audio_buffer>	Instantiates an <audio_buffer> element.	N/A	0 or 1
<instance_audio_delay>	Instantiates an <audio_delay> element.	N/A	0 or 1
<instance_audio_mix>	Instantiates an <audio_mix> element.	N/A	0 or 1

<instance_audio_switch>	Instantiates an <audio_switch> element.	N/A	0 or 1
<instance_audio_dsp>	Instantiates an <audio_dsp> element.	N/A	0 or 1

B.9. audio_object

Child Elements

Name/example	Description	Default	Occurrences
<asset>	Defines asset-management information.	N/A	0 or 1
Profile	At least one profile must appear, but any number of any of following profiles can be included: • <profile_COMMON> • <profile_FMOD> • <profile_PD>	N/A	1 or more
<extra>	Adds additional user-defined information to the <audio_object>.	N/A	0 or more

Child Elements for <audio_object>/<profile_*>

Name/example	Description	Default	Occurrences
<newparam>	Creates a new parameter from a constrained set of types.	N/A	0 or more
<instance_sound>	Instantiates a <sound> element.	N/A	0 or 1
<instance_sound_2d>	Instantiates a <sound_2d> element.	N/A	0 or 1
<instance_sound_ext>	Instantiates a <sound_ext> element.	N/A	0 or 1
<extra>	Adds additional user-defined information to the <audio_object>/<profile_*>	N/A	0 or more

B.9.1. instance_audio_object

Attributes

Name/example	Type	Description
sid	sid_type	A text string value containing the scoped identifier of this element. This value must be unique within the scope of the parent element. Optional.
name	xs:token	The text string name of this element. Optional.

url	xs:anyURI	The URL of the location of the <audio_object> element to instantiate. Refers to a local instance or an external reference. Required.
layer	list_of_names_type	The names of the layers to which this instantiated audio object belongs. For example, a value of "foreground midground" indicates that this object belongs to both the layer named foreground and the layer named midground. The default is empty, indicating that the object doesn't belong to any specific layer. Optional.

Child Elements

Name/example	Description	Default	Occurrences
Profile	At least one profile must appear, but any number of any of following profiles can be included: • <profile_COMMON> • <profile_FMOD> • <profile_PD>	N/A	1 or more
<control>	Controls the playback of the audio object when instantiated. Valid enumeration values are: • PLAY • STOP • PAUSE	PLAY	0 or 1
<extra>	Adds additional user-defined information to the <instance_audio_object>.	N/A	0 or more

Child Elements for <instance_audio_object>/<profile_*>

Name/example	Description	Default	Occurrences
<bind_sound>	Binds a <sound> element to a node in the visual scene.	N/A	0 or 1
<bind_sound_2d>	Binds a <sound_2d> element to a node in the visual scene.	N/A	0 or 1
<bind_sound_ext>	Binds a <sound_ext> element to a node in the visual scene.	N/A	0 or 1
<extra>	Adds additional user-defined information to the <instance_audio_object>/<profile_*>	N/A	0 or more

B.10. audio_dsp

Child Elements

Name/example	Description	Default	Occurrences
<asset>	Defines asset-management information.	N/A	0 or 1
<newparam>	Create a new parameter from a constrained set of types recognizable by all platforms.	N/A	0 or more
Profile	At least one profile must appear, but any number of any of following profiles can be included: • <profile_COMMON> • <profile_FMOD> • <profile_PD>	N/A	1 or more
<extra>	Adds additional user-defined information.	N/A	0 or more

Child Elements for <audio_dsp>/<profile_*>

Name/example	Description	Default	Occurrences
<asset>	Defines asset-management information.	N/A	0 or 1
<newparam>	Create a new parameter from a constrained set of types recognizable by all platforms.	N/A	0 or more
<technique> (DSP)	Declares a technique for this audio dsp. See the below table for child element details.	N/A	1 or more
<extra>	Adds additional user-defined information.	N/A	0 or more

Child Elements for <audio_dsp>/<profile_*>/<technique>

Name/example	Description	Default	Occurrences
<asset>	Defines asset-management information.	N/A	0 or 1
DSP elements	Create a DSP effect from the following set of types: • <audio_dsp_compressor> • <audio_dsp_delay>, • <audio_dsp_equalizer>,	N/A	0 or 1

	• <audio_dsp_filter>, • <audio_dsp_reverb>.		
<extra>	Adds additional user-defined information.	N/A	0 or more

B.10.1. audio_dsp_compressor

Function	Type	Default value	Range	Unit
attack	float	50.0	10.0 … 200.0	seconds
makeup_gain	float	0.0	0.0 … 30.0	dB
ratio	float	3.0	1.0 … 60.0	ratio
release	float	50.0	20.0 … 1000.0	seconds
threshold	float	0.0	-60.0 … 0	dB

B.10.2. audio_dsp_delay

Function	Type	Default value	Range	Unit
delay_time	float	500.0	10.0 … 5000.0	seconds
feedback	float	0.5	0.0 … 1.0	ratio
mix	float	0.5	0.0 … 1.0	ratio

B.10.3. audio_dsp_equalizer

Function	Type	Default value	Range	Unit
bandwidth	float_array	[1.0 1.0 1.0 1.0 1.0 1.0]	0.2 … 5.0	octave
center_freq	float_array	[80.0 160.0 500.0 2000.0 5000.0 12000.0]	20.0 … 22000.0	Hz
gain	float_array	[0.0 0.0 0.0 0.0 0.0 0.0]	-20.0 … 20.0	dB
num_band	int	6	0 … 16	N/A

B.10.4. audio_dsp_filter

Function	Type	Default value	Range	Unit
type	int	0	0 … 2	{lowpass, highpass, bandpass, bandreject}

bandwidth	float	2400.0	1.0 … 22000.0	Hz
cutoff_freq	float	5000.0	1.0 … 22000.0	Hz
resonance	float	1.0	0.0 … 10.0	N/A

B.10.5. audio_dsp_reverb

Function	Type	Default value	Range	Unit
room	float	-1000.0	-10000.0 … 0.0	mB
hf_room	float	0.0	-10000.0 … 0.0	mB
lf_room	float	0.0	-10000.0 … 0.0	mB
reflection_level	float	-10000.0	-10000.0 … 1000.0	mB
reverb_level	float	0.0	-10000.0 … 2000.0	mB
pre_delay	float	0.02	0.0 … 0.3	seconds
late_delay	float	0.04	0.0 … 0.1	seconds
decay_time	float	1.0	0.1 … 20.0	seconds
decay_ratio	float	0.5	0.1 … 2.0	ratio
hf_reference	float	5000.0	1000.0 … 20000.0	Hz
lf_reference	float	250.0	20.0 … 1000.0	Hz
density	float	100.0	0.0 … 100.0	percent
diffusion	float	100.0	0.0 … 100.0	percent

B.11.acoustic_environment

Child Elements

Name/example	Description	Default	Occurrences
<preset>	Specifies the preset environment name, one of the following: “generic”,”paddedcell”,“room”,“bath room”,“livingroom”,“stoneroom”,“auditorium”,“concerthall”,“cave”,”arena”,”hangar”,”carpetedhallway”,“hallway”,”stonecorridor”,”alley”,”forest”,“city”,”mountains”,”quarry”,”plain”,“parkinglot”,”sewerpipe”,”underwater”.	“generic”	1
<env_diffusion>	Specifies the environment diffusion.	1.0	0 or 1
<room>	Specifies the overall room effect level.	-1000.0	0 or 1
<hf_room>	Controls relative room effect high-	-100.0	0 or 1

	frequency level.		
<lf_room>	Controls relative room effect low-frequency level.	0.0	0 or 1
<reflection_level>	Specifies the early reflections volume levels relative to that of the room effect.	-2602.0	0 or 1
<reverb_level>	Specifies the late reverberation volume levels relative to that of the room effect.	200.0	0 or 1
<pre_delay>	Specifies the delay time of the first reflection.	0.007	0 or 1
<late_delay>	Specifies the late reverberation delay time relative to the initial reflection.	0.011	0 or 1
<decay_time>	Specifies the length of time that room reverberation takes to fade to silence. This feature can be used to simulate the size of the acoustic environment.	1.49	0 or 1
<decay_hf_ratio>	Specifies the high-frequency to mid-frequency decay time ratio.	0.83	0 or 1
<decay_lf_ratio>	Specifies the low-frequency to mid-frequency decay time ratio.	1.0	0 or 1
<hf_reference>	Specifies the high-frequency values referenced by other parameters.	5000.0	0 or 1
<lf_reference>	Specifies the low-frequency values referenced by other parameters.	250.0	0 or 1
<density>	Represents how tightly the reflections in the decay are packed.	100.0	0 or 1
<diffusion>	Specifies the rate at which the density of the reverb increases during the decay.	100.0	0 or 1

B.12. audio_scene

Child Elements

Name/example	Description	Default	Occurrences
<asset>	Defines asset-management information.	N/A	0 or 1
<instance_audio_object>	Instantiates an audio_object element.	N/A	1 or more
<instance_acoustic_environment>	Instantiates an audio_object element.	N/A	0 or more
<evaluate_scene>	The <evaluate_scene> element	N/A	0 or more

	declares information specifying how to evaluate this audio_scene.		
<extra>	Adds user-defined and multi-representable information to the <audio_scene>.	N/A	0 or more

This page intentionally left blank.

APPENDIX C
Common DSP Functions

This appendix describes some common DSP functions.

C.1. Compressor

Compressor is a type of amplitude domain effects. Dynamic range compression is the process of compressing dynamic range of an audio signal so that the gain of loud sounds can be reduced and quite sounds can be amplified.

The amount and starting point of gain reduction done to the audio signal is determined by ratio and threshold. For instance, a ratio of 4:1 means that if input level is 4dB exceeding the <threshold>, the output level will be 1 dB above the <threshold>. The gain has been reduced by 3dB. When input level is 8dB exceeding the threshold, the output level will be 2dB above the threshold with 6dB gain reduction.

FIGURE C-1 shows the two primary procedures of dynamic range compression. On the left is a Downward Compression process that decreases loud sounds level over a certain threshold and leaves loud sounds unaffected, whereas the right-hand-side process is known as an Upward Compression that increases quiet sounds level while sounds louder than the threshold remain unchanged [RGG09].

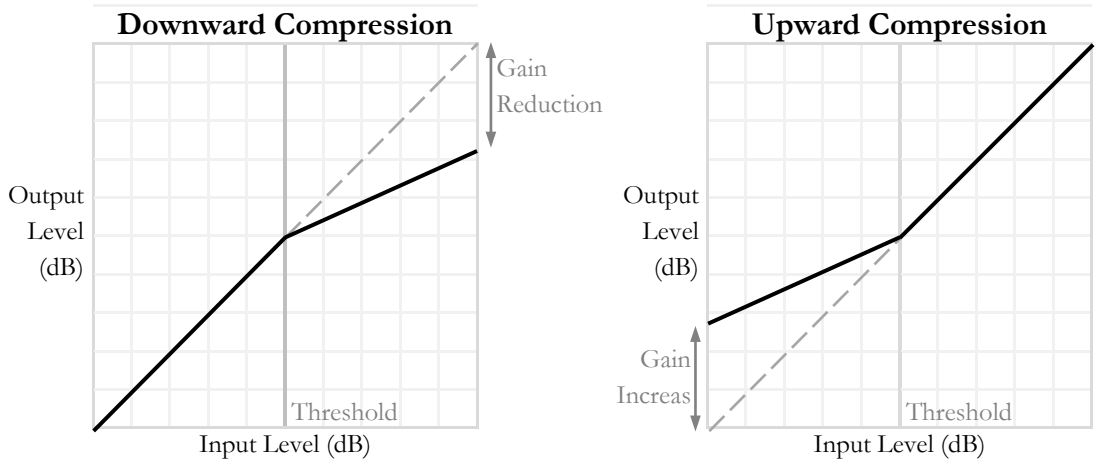


FIGURE C-1. Two main methods of dynamic range compression.

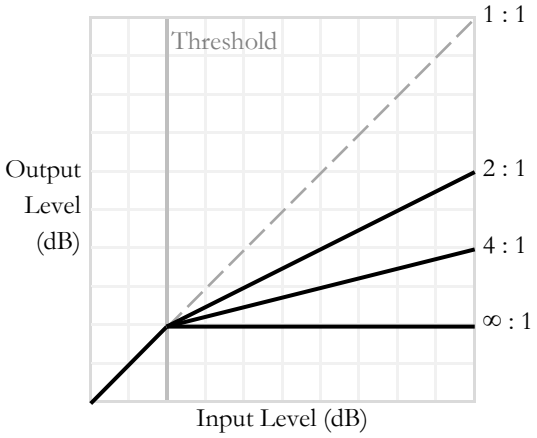


FIGURE C-2. Different compression ratios for DSP compression process.

C.2. Feedback Delay

Feedback delay allows creating basic time-based delay effects, such as echoes, chorus, etc. The feedback control takes the output from the delay line and feeds it back to the input.

FIGURE C-3 diagrams typical feedback delay process, using the common feedback delay algorithm as shown in the equation below:

$$\boldsymbol{output} = (1 - < mix >) \times \boldsymbol{input} + < mix > \times delay \tag{C-1}$$

, where

$$delay_{sample} = < delay_{time} > \times sample_{rate}$$

$$delay[n] = delayBufer[n - delay_{sample}]$$

$$\boldsymbol{delayBuffer}[n] = \sum_{i=0}^{N-1} < feedback >^{i} \times (\boldsymbol{input}[n - i \times \boldsymbol{delay}_{sample}])$$

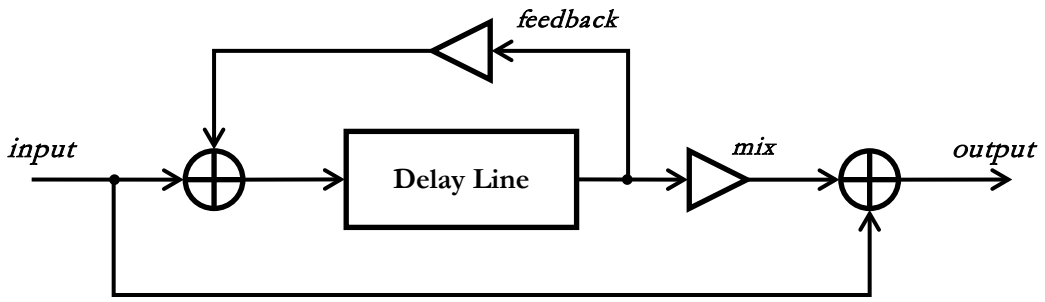


FIGURE C-3. The feedback delay processor.

C.3. Parametric Equalizer

A parametric equalizer (EQ) is a multi-band variable equalizer that is used to adjust the frequency response of a signal. The Q factor given by the below formula refers the quality of a frequency filter or a damped oscillator.

$$Q = \frac{\boldsymbol{Center\ Frequency}}{\boldsymbol{Bandwidth}} = \frac{f_0}{\Delta f} = \frac{f_0}{f_2 - f_1} \tag{C-2}$$

Following the definition of a decibel (dB), a power level change of 50% corresponds to:

$$\sqrt{0.5 Power} \cong 0.707 \mathbf{Amplitude}$$

$$\mathbf{Power\ in\ decibel} = 20\log\left(\frac{Output\ amplitude}{Input\ amplitude}\right) = 20\log\left(0.\frac{701}{1}\right) = -3\mathbf{dB} \quad \text{(C-3)}$$

Given that 3 dB is where that the power is reduced by half (Equation 5.3), bandwidth is derived from the points (f_1 and f_2) on the EQ curve that are 3 dB above or below the amount by which the central frequency (f_0) has been cut or boosted. The central frequency is at the midpoint (in octaves) between the lower and upper frequencies. That is, if the bandwidth is two octaves, the center frequency is one octave above the lower frequency and one octave below the upper frequency.

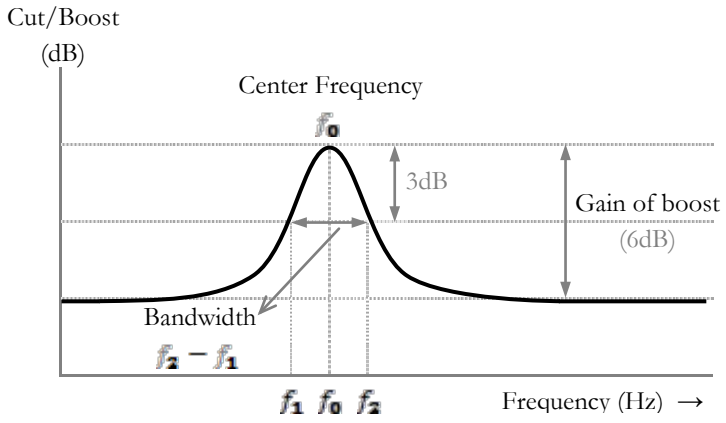


FIGURE C-4. A frequency boosts using a parametric EQ.

C.4. Frequency-based Filter

Frequency-based filters cut off a certain range of frequency content in the signal. There are four types of resonating filters provided: lowpass, highpass, bandpass and bandreject.

The gain curve of low-pass and high-pass filters is graphically depicted in FIGURE C-5. FIGURE C-6 illustrates the gain curve of band filters.

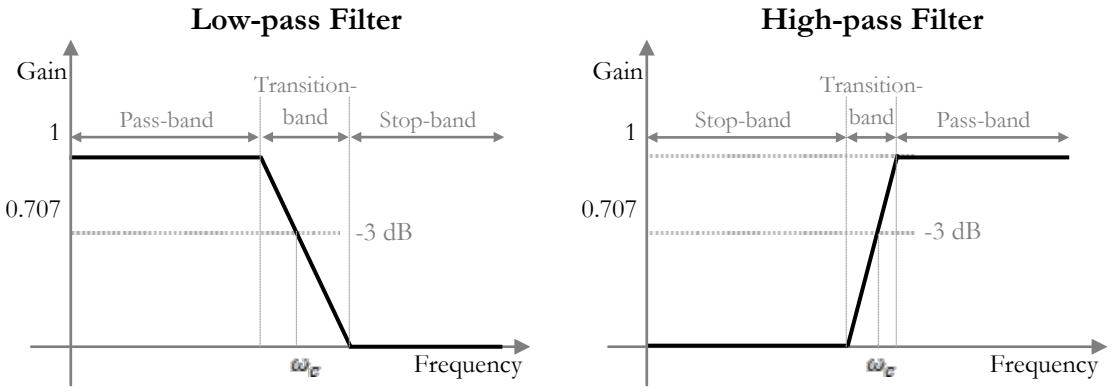


FIGURE C-5. Gain curve of low-pass and high-pass filters.

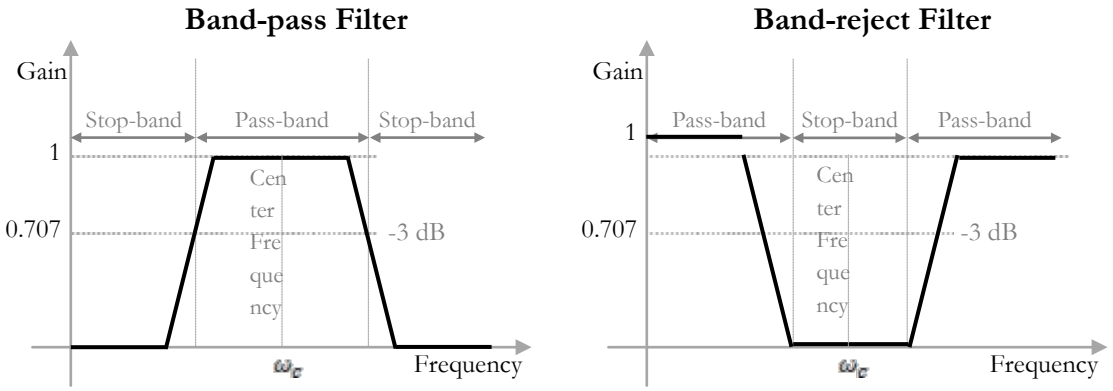


FIGURE C-6. Gain curve of band filters.

This page intentionally left blank.

BIBLIOGRAPHY

[S77] R. Murray Schafer: *The Tuning of the World*. Random House Inc. 1977. Republished as *The Soundscape: Our Sonic Environment and the Tuning of the World*. Destiny Books. ISBN: 978-0-89281-455-8. 1993.

[T84] B. Truax: *Acoustic Communication*. Ablex Publishing. 1984. The 2nd Edition. Greenwood Press. ISBN: 978-1-56750-536-8. 2001.

[WB85] E. Weis and J. Belton.: *Film Sound: Theory and Practice*. Columbia University Press. ISBN: 978-0-23105-637-3. 1985.

[R96] C. Roads: *The Computer Music Tutorial*. The MIT Press. ISBN: 978-0-26268-082-0. 1996.

[S98] E. D. Scheirer: *The MPEG-4 Structured Audio Standard*. In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing. 1998.

[T98] B. Truax: *Real-time Granular Synthesis with a Digital Signal Processor*. Computer Music Journal, 12 (2). 1998.

[SV99] E. D. Scheirer and B. L. Vercoe: *SAOL: The MPEG-4 Structured Audio Orchestra Language*. Computer Music Journal, 23 (2). 1999.

[SVH99] E. D. Scheirer, R. Väänänen and V. Houpaniemi: *AudioBIFS: Describing Audio Scenes with the MPEG-4 Multimedia Standard*. IEEE Transactions on Multimedia, 1 (3), pp.237-250. 1999.

[D00] J. P. Déziel: *Applied Introduction to Digital Signal Processing*. Prentice Hall. ISBN: 978-0-13775-768-8. 2000.

[DH00] Y. Douglas and A. Hargadon: *The Pleasure Principle: Immersion, Engagement, Flow*. In Proceedings of the 11th ACM on Hypertext and Hypermedia (HYPERTEXT00), pp. 153-160. 2000.

[KWC00] M. Kim, S. Wood and L. Cheok: *Extensible MPEG-4 Textual Format (XMT)*. In Proceedings of the ACM Workshops of Multimedia, pp.71-74. 2000.

[SW00] D. Schwarz and M. Wright: *Extensions and Applications of the SDIF Sound Description Interchange Format*. In Proceedings of the International Computer Music Conference (ICMC), pp.481-484. 2000.

[SYE00] J. Signès, Y.Fisher and A. Eleftheriadis: *MPEG-4's Binary Format for Scene Description*. Signal Processing of Image Communication, 15, pp.321-345. 2000.

[WCF00] M. Wright, A. Chaudhary, A. Freed, S. Khoury, A. Momeni and D. Wessel: *An XML-based SDIF Stream Relationships Language*. In Proceedings of the International Computer Music Conference (ICMC), pp.186-189. 2000.

[TS01] A. Topol and F. Schaeffer: *Enhancing Sound Description in VRML*. In Proceedings of the International Computer Music Conference (ICMC). 2001.

[CD02] C. Concolato and J. Dufourd: *Comparison of MPEG-4 BIFS and Some Other Multimedia Description Languages*. Workshop and Exhibition on MPEG-4 (WEPM). 2002.

[CK02] K. Cha and S. Kim: *Interactive Authoring Tool for Extensible MPEG-4 Textual Format*. Workshop Notes of Semantic Authoring Annotation and Knowledge Markup 15th European Conference on Artificial Intelligence (ECAI). 2002.

[DHM02] R. Dachselt, M. Hinz and K. Meißner: *CONTIGRA: An XML-based Architecture for Component-Oriented 3D Applications*. In Proceedings of the 7th International Conference on 3D Web Technology (Web3D02), pp.155-163. 2002.

[FJT02] T. Funkhouser, J. Jot and N. Tsingos: *"Sounds Good to Me!" Computational Sound for Graphics, Virtual Reality, and Interactive Systems*. In SIGGRAPH 2002 Course Notes.

[GLM02] V. Gal, C. le Prado, J.B. Merland, S. Natkin and L. Vega: *Processes and Tools for Sound Design in Computer Games*. In Proceedings of the International Computer Music Conference (ICMC). 2002.

[GLN02] V. Gal, C. le Prado, S. Natkin and L. Vega: *Writing for Video Games*. In Proceedings of the 5th Virtual Reality International Conference (Laval Virtual, VRIC02). 2002.

[PB02] G. Potard, I. Burnett: *Using XML Schemas to Create and Encode Interactive 3-D Audio Scenes for Multimedia and Virtual Reality Applications*. In Proceedings of the 4th International Workshop on Distributed Communities on the Web (DCW), pp.193-203. 2002.

[SRJ02] M. Sung, S. Rho and J. Jang: *A SMIL-based Multimedia Presentation Authoring System and Some Remarks on Future Extension of SMIL*. In Proceedings of the Packet Video Conference. 2002.

[D03] D. Dubois: *Perception, Representation and Knowledge: Acoustic Phenomena between Noise and Sounds*. In Proceedings of the 34th Spanish Congress on Acoustics (TecniAcustica). 2003.

[FL03] B. Schulte-Fortkamp, P. Lercher: *The Importance of Soundscape Research for the Assessment of Noise Annoyance at the Level of the Community*. 2003.

[M03] A. Mcmahan: *Immersion, Engagement, and Presence: A Method for Analyzing 3D Videogames*. In The Video Game Theory Reader, pp. 67-86. ISBN: 978-041596-579-8. 2011.

[PL03] K. Pihkala and T. Lokki: *Extending SMIL with 3D Audio*. In Proceedings of the International Conference on Auditory Display (ICAD), pp.95-98. 2003.

[HDM03] H. Hoffmann, R. Dachselt, K. Meissner: *An Independent Declarative 3D Audio Format on the Basis of XML*. ICAD, pp.99-102. 2003.

[TGD03] N. Tsingos, E. Gallo and G. Drettakis: *Breaking the 64 Spatialized Sources Barrier*. Gamasutra.com. 2003.

[BA04] J. Bresson, C. Agon: *SDIF Sound Description Data Representation and Manipulation in Computer Assisted Composition*. In Proceedings of the International Computer Music Conference (ICMC). 2004.

[G04] T.Grill: *flext – C++ Layer for Pure Data and Max/MSP Externals*. In the 2nd Linux Audio Conference. 2004.

[TGD04] N. Tsingos, E. Gallo and G. Drettakis: *Perceptual Sound Rendering of Complex Virtual Environments.* ACM SIGGRAPH, 23 (3), pp.249-258. 2004.

[VH04] R. Väänänen and J. Huopaniem: *Advanced AudioBIFS: Virtual Acoustics Modeling in MPEG-4 Scene Description.* IEEE Transactions on Multimedia, 6 (5), pp.661-675. 2004.

[ISO05] *Information Technology — Coding of Audio-Visual Objects — Part 11: Scene Description and Application Engine.* ISO/IEC FDIS 14496-11:2005 (E). 2005.

[JHB05] R. Jeffs, S. Holden and D. Bohn: *Dynamics Processors — Technology & Application Tips.* RaneNote 155. Rane Corporation. 2005.

[RD05] M. Raimbault and D. Dubois: *Urban Soundscapes: Experiences and Knowledge.* Cities, 22 (5), pp.339-350. ISBN: 02642751. 2005.

[V05] R. Väänänen: *3-D Audio and Virtual Acoustics.* 2005.

[AB06] R. Arnaud and M. C. Barnes: *COLLADA*: *Sailing the Gulf* of 3D Digital Content Creation. A K Peters, Ltd. 2006.

[F06] A. J. Farnell: *Sound Synthesis for Games.* Sounding Out 3: Sunderland University. 2006.

[L06] J. Lindbergh: *Implementation* of a *COLLADA Scene-Graph.* Master Thesis, Luleå University of Technology. 2006.

[MCW06] A. Misra, P. R. Cook and G. Wang: A New Paradigm for Sound Design. In Proceedings of the 9th International Conference on Digital Audio Effects (DAFx-06), pp.319-324. 2006.

[N06] S. Natkin: *Video Games & Interactive Media: A Glimpse at New Digital Entertainment.* ISBN: 978-1-56881-297-7. A K Peters, Ltd. 2006.

[P06] G. Potard: *3D-Audio Object Oriented Coding.* Ph.D. Thesis. University of Wollongong. 2006.

[SBV06] D. Schwarz, G. Beller, B. Verbrugghe and S. Britton: *Real-time Corpus-based Concatenative Synthesis with CataRT.* In Proceedings of the 9th International Conference on Digital Audio Effects (DAFx-06). 2006.

[SVS06] B. Shao, L. M. Velazquez, N. Scaringella, N. Singh and M. Mattavelli: *SMIL to MPEG-4 BIFS Conversion.* In Proceedings of the 2nd International

Conference on Automated Production of Cross Media Content for Multi-Channel Distribution (AXMEDIS), pp.77-84. 2006.

[VNL06] O. Veneri, S. Natkin, C. le Prado and M. Emerit: *A Game Audio Technology Overview*. In Proceedings of the 3rd Sound and Music Computing Conference (SMC06). 2006.

[AP07] A. Arnaud and T. Parisi: *Developing Web Applications with COLLADA and X3D*. 2007.

[B07] R Bridgett: *Post-production Sound: A New Production Model for Interactive Media.* The Soundtrack, 1(1), pp.29-39. 2007.

[CV07] E. Coumans and K. Victor: *COLLADA Physics*. 2007.

[F07] A. Farnell: *An Introduction to Procedural Audio and its Application in Computer Games*. Audio Mostly Conference. 2007.

[J07] J. Jot: *Efficient Description and Rendering of Complex Interactive Acoustic Scenes*. In Proceedings of the 10th International Conference on Digital Audio Effects (DAFx-07), pp.99-100. 2007.

[LN07] C. Le Prado and S. Natkin: Listen Lisboa: *Scripting Languages for Interactive Musical Installations*. In: Proceedings of the 4th Sound and Music Computing Conference (SMC07), pp. 50-56. 2007.

[MBT07] T. Moeck, N. Bonneel, N. Tsingos, G. Drettakis, I. Viaud-Delmon, D. Alloza: *Progressive Perceptual Audio Rendering of Complex Scenes*. In Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games (I3D07), pp. 186-196. 2007.

[P07a] M. Pohja: *X3D and VRML Sound Components*. 2007.

[P07b] M. Puckette: *The Theory and Technique of Electronic Music*. World Scientific Publishing. 2007.

[PFM07] N. Peters, S. Ferguso and S. McAdams: *Towards a Spatial Sound Description Interchange Format (SpatDIF).* Canadian Acoustics, 35 (3). 2007.

[T07] B. Truax: *The Analysis of Electroacoustic Music as Soundscape*. Electroacoustic Music Studies 07, De Montfort University. 2007.

[BCP08] J. Burred, C. Cella, G. Peeters, A. Röbel and D. Schwarz: *Using the SDIF Sound Description Interchange Format for Audio Features*. International

Conference on Music Information Retrieval (ISMIR), pp.427-432. 2008.

[C08a] K. Collins: *From Pac-Man to pop music: Interactive Audio in Games and New Media*. Ashgate Publishing, Ltd. ISBN: 978-0-75466-200-6. 2008.

[C08b] K. Collins: *Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design*. MIT Press. ISBN: 978-0-26203-378-7. 2008.

[BF08a] M. Barnes and E. L. Finch: *COLLADA – Digital Asset Schema Release 1.4.1 Specification (2nd Edition)*. Sony Computer Entertainment Inc. 2008.

[BF08b] M. Barnes and E. L. Finch: *COLLADA – Digital Asset Schema Release 1.5.0 Specification*. Sony Computer Entertainment Inc. 2008.

[GM08] G. Gatzsche and F. Melchior: *Spatial Audio Authoring and Rendering: Forward Research Through Exchange*. In Proceedings of the International Computer Music Conference (ICMC), pp.303-304. 2008.

[GS08] M. Geier and S. Spors: *ASDF: Audio Scene Description Format*. In Proceedings of the International Computer Music Conference (ICMC). 2008.

[ISO08] *X3D Architecture and Base Components Edition 2*. ISO/IEC 19775-1.2:2008. 2008.

[K08a] J. Kadis: *Dynamic Range Processing and Digital Effects*. 2008.

[K08b] G. S. Kendall: *Interchange Formats and the Art of Spatial Audio*. In Proceedings of the International Computer Music Conference (ICMC). 2008.

[M08] S. Morgan: *Dynamic Game Audio Ambience: Bringing Prototype's New York City to Life*. Gamasutra.com. 2008.

[P08a] N. Peters: *Proposing SpatDIF - The Spatial Sound Description Interchange Format*. In Proceedings of the International Computer Music Conference (ICMC). 2008.

[P08b] S. T. Pope: *Interchange Formats for Spatial Audio*. In Proceedings of the International Computer Music Conference (ICMC). 2008.

[T08] B. Truax: *Soundscape Composition as Global Music: Electroacoustic Music as Soundscape*. Organised Sound, 13 (2), pp.103-109. 2008.

[VGN08] O. Veneri, S. Gros and S. Natkin: *Procedural Audio for Game using GAF*. 2008.

[VP08] O. Veneri and Y. Planqueel: *Create a Scalable and Creative Audio Environment: Middleware Project PLAY ALL*. Game Developers Conference (GDC08). 2008.

[BEJ09] J. Behr, P. Eschler, Y. Jung, M. Zöllner: *X3DOM: a DOM-based HTML5/X3D integration model*. In Proceedings of the 14th International Conference on 3D Web Technology (Web3D09), pp.127-135. ISBN: 978-1-60558-432-4. 2009.

[PLS09] N. Peters, T. Lossius, J. Schacher, P. Baltazar, C. Bascou and T. Place: *A Stratified Approach for Sound Spatialization*. In Proceedings of the 6th Sound and Music Computing Conference (SMC09), pp.23-25. 2009.

[RGG09] D. Reese, L. Gross and B. Gross: *Audio Production Worktext: Concepts, Techniques, and Equipment*. Elsevier Science. ISBN: 978-0-24081-098-0. 2009.

[V09] O. Veneri: *Architecture d'un Intergiciel pour la Création Sonore dans les Jeux Vidéo*. Ph.D. Thesis, Conservatoire National des Arts et Métiers. 2009.

[VLS09] A. Valle, B. Lombardo and M. Schirosa: *Simulating the Soundscape through an Analysis/Resynthesis Methodology*. In Auditory Display, 6th International Symposium, CMMR/ICAD, pp.330-357. 2009.

[B10] W. Brent: *A Timbre Analysis and Classification Toolkit for Pure Data*. In Proceedings of the International Computer Music Conference (ICMC). 2010.

[CLN10] S. Chan, C. Le Prado, S. Natkin and G. Tiger: *A Sound Engine for Virtual Cities*. In Proceedings of the 9th International Conference on Entertainment Computing (ICEC2010), pp.443-445. 2010.

[F10a] A. Farnell: *Designing Sound*. MIT Press. 2010.

[F10b] Firelight Technologies Pty, Ltd.: *FMOD Designer 2010 User Manual*. 2010.

[GAS10] M. Geier, J. Ahrens and S. Spors: *Object-based Audio Reproduction and the Audio Scene Description Format*. Journal Organised Sound, 15 (3), pp.219-227. 2010.

[NCD10] M. Niessen, C. Cance and D. Dubois: *Categories for Soundscape: Toward a*

Hybrid Classification. INTER-NOISE 2010, 92 (6), pp.1-14. 2010.

[SKR10] K. Sons, F. Klein, D. Rubinstein, S. Byelozyorov and P. Slusallek: *XML3D: Interactive 3D Graphics for the Web*. In Proceedings of the 15th International Conference on 3D Web Technology (Web3D10), pp.175-184. ISBN: 978-1-45030-209-8. 2010.

[T10] R. Turkowski: *Enabling the Immersive 3D Web with COLLADA & WebGL*. 2010.

[VAM10] C. Verron, M. Aramaki, R. Kronland-Martinet and G. Pallone: *A 3D Immersive Synthesizer for Environmental Sounds*. IEEE Transactions on Audio, Speech, and Language Processing, 18 (6), pp.1550-1561. 2010.

[X10] XML3D.ORG: *XML3D 0.4 Specification*. 2010.

[BKL11] P. Brinkmann, P. Kirn, R. Lawler, C. McCormick, M. Roth and H. Steiner: *Embedding Pure Data with libpd*. In the 4th international Pure Data Convention. 2011.

[BS11] J. Bresson and M. Schumacher: *Representation and Interchange of Sound Spatialization Data for Compositional Applications*. In Proceedings of the International Computer Music Conference (ICMC). 2011.

[CLN11a] S. Chan, C. Le Prado, S. Natkin, G. Tiger and A. Topol: *Sound in COLLADA*. In Proceedings of the 10th International Conference on Entertainment Computing (ICEC2011), pp.235-246. 2011.

[CLN11b] S. Chan, C. Le Prado, S. Natkin, G. Tiger and A. Topol: *Sound in Virtual Cities: The Project TerraDynamica*. Versatile Sound Models for Interaction in Audio–graphic Virtual Environments Workshop at the 14th International Conference Digital Audio Effects (DAFx-11). 2011.

[DUO11] R. Diankov, R. Ueda, K. Okada and H. Saito: *COLLADA: An Open Standard for Robot File Formats*. The 29th Annual Conference of the Robotics Society of Japan International Session. 2011.

[P11] L. J. Paul: *Granulation of Sound in Video Games*. In Proceedings of the AES 41st International Conference. 2011.

[SCB11] D. Schwarz, R. Cahen, F. Brument, Hui Ding and Christian Jacquemin: *Sound Level of Detail* in Interactive Audiographic 3D Scenes. In Proceedings of the International Computer Music Conference (ICMC).2011.

[SCS11] D. Schwarz, R. Cahen and N. Schnell: *Descriptor-Based Texture Synthesis Control in Interactive Audio–Graphic 3D Scenes by Activation Profiles.* Versatile Sound Models for Interaction in Audio–graphic Virtual Environments Workshop at the 14th International Conference Digital Audio Effects (DAFx-11). 2011.

[SP11a] SpatDIF Project: *Scene Examples for SpatDIF V 0.2.* 2011.

[SP11b] SpatDIF Project: *SpatDIF Specification V 0.2.* 2011.

[C12] E. Coumans: *Bullet 2.80 Physics SDK Manual.* 2012.

[CNT12] S. Chan, S. Natkin, G. Tiger and A. Topol: *Extensible Sound Description in COLLADA: A Unique File for a Rich Sound Design.* In Proceedings of the 9th Advances in Computer Entertainment Technology Conference (ACE2012). 2012.

[PLS12] N. Peters, T. Lossius, J. C. Schacher: *SpatDIF: Principles, Specification, and Examples.* In Proceedings of the 9th Sound And Music Computing Conference (SMC12). 2012.

[SS12] K. Sons and P. Slusallek: Demo: *XML3D – Interactive 3D Graphics for the Web.* 21st International World Wide Web Conference (WWW2012). 2012.

[CEM] COLLADA Extensibility Mechanisms: http://collada.org/mediawiki/index.php/Extension

[CEP] COLLADA Extensions Portal: http://collada.org/mediawiki/index.php/Portal:Extension_directory

[FMODEX] Firelight Technologies Pty. Ltd.: *FMOD Ex Documentation.*

[GEARTH] Google Earth: http://earth.google.com

[GLIBPD] GitHub/libpd: https://github.com/libpd

[LIBPD] Libpd: http://libpd.cc/

[MAAPI] Mozilla Audio Data API: https://wiki.mozilla.org/Audio_Data_API

[OM6.5] OpenMusic 6.5 User Manual – SDIF: http://support.ircam.fr/forum-ol-doc/om/om6-manual/co/SDIF.html

[PV3D] Papervision3d - Open Source realtime 3D engine for Flash: https://code.google.com/p/papervision3d/

[SDIF] Main SDIF site: http://sdif.sourceforge.net/

[SMIL] SMIL W3C Recommendation: http://www.w3.org/TR/2008/REC-SMIL3-20081201/

[SPATDIF] Wiki of SpatDIF: http://redmine.spatdif.org/projects/spatdif/wiki

[TPHD] G. Tiger: Ph.D. Thesis, Conservatoire National des Arts et Métiers. 2009.

[WEBAUD] W3C Editor's Draft – Web Audio API: http://dvcs.w3.org/hg/audio/raw-file/tip/webaudio/specification.html

[XNA] MSDN Library - XNA Game Studio Express – Programming Guide: http://msdn.microsoft.com/en-US/library/bb203887(v=xnagamestudio.10)

Annexe E

Résumé en Français

OUTLINE

E.1. Introduction

Actuellement, la quasi-totalité des applications mettent à contribution les canaux visuels et auditifs de l'utilisateur afin de lui notifier l'exécution et l'accomplissement de tâches. Une telle application initialise et effectue le rendu de ses deux médias indépendamment car ils n'utilisent pas les mêmes fichiers supports ni les mêmes informations. Il en résulte que les mêmes informations nécessaires pour les 2 médias sont chargées indépendamment et dupliquées en mémoire pour convenir à chacune des 2 API de rendu qui sera utilisées. Par exemple la librairie graphique OpenGL utilise des transformations géométriques qui n'affectent pas les positions des sources sonores ou de l'écoutant dans la libraire sonore OpenAL. Lier les informations auditives et visuelles ensemble dans un même fichier support permettrait de faciliter la description globale d'une application audiovisuelle.

Les objets visuels et audio peuvent être générés de différentes manières, avec différentes méthodes, qui peuvent être utilisés dans la même application. Par exemple, dans un jeu vidéo, les graphismes peuvent être obtenus grâce à des modèles 3D texturés avec des images fixes (bitmap 2D) ou animées (vidéos 2D) et être animés soit par images clés (fichier d'animation) ou par animation procédurale (un ensemble de paramètres). Pour la partie audio, un jeu vidéo peut jouer une musique stéréo ou une ambiance synthétique mixée avec le rendu de sources sonores spatialisées auxquelles on peut associer différents effets. L'approche usuelle pour lier tous ces éléments est de coder individuellement l'initialisation des ressources et les liens existants entre elles. Cela peut être fait de manière très bas-niveau dans le code de l'application ou bien en utilisant un outil de conception de scène (*scene designer*) comme ceux accessibles dans les moteurs de jeux populaires. Cette dernière manière de construire une scène en spatialisant les objets audiovisuels est très puissante puisqu'elle facilite grandement le travail d'intégration du programmeur et permet au *level designer* d'être autonome pour la création des niveaux de jeu. Cependant, cette solution s'appuie sur des technologies propriétaires et exploite des fichiers générés avec un format binaire inconnu et donc exclusivement utilisables dans le moteur de jeu. On n'obtient donc pas avec cette méthode une description générale qui peut être partagée par différentes solutions logicielles. Pour cette raison, certains créateurs de contenus préfèrent utiliser des langages de description de scène ouverts pour décrire leurs environnements multimédia 3D.

Des formats de fichier libre de droits permettent d'échanger des ressources entre applications et plateformes et donc d'empêcher la perte de données lors des conversions de fichiers nécessaires dans d'autre cas. Cela fait de nombreuses années que cette technique s'applique à la description de scènes audiovisuelles en 3D. Cependant, les capacités de

description sont principalement pour les aspects visuels. Lorsque la description de l'acoustique d'une scène est présente elle est incomplète, démodée ou bien trop sophistiquée pour une réalisation pratique. De plus, les standards de description audio actuels sont généralement liés aux domaines de la musique générée par ordinateur ou à la délivrance de contenus web plutôt qu'aux environnements virtuels interactifs. Nous nous sommes aperçus qu'un langage de description de scène répondant à notre volonté d'ajouter du son dans un environnement urbain simulé restait à inventer. Par conséquent, nous proposons un nouveau standard de représentation sonore.

COLLADA est notre alternative préférée car ce langage est simple, échangeable et transportable entre applications et plateformes. Les aspects visuels dynamiques sont décrits dans les fichiers COLLADA alors que les capacités sonores ne les sont pas. Nous avons donc décidé pour minimiser la redondance des informations et donc de leur traitement d'ajouter dans les fichiers COLLADA une branche son corrélée avec le visuel. De cette manière l'ensemble de données communes pour le rendu de l'audio et de l'image n'est pas redondé mais décrit une unique fois pour l'un et lié à l'autre.

Dans la section suivante, nous présenterons plus en détails les techniques permettant de décrire des environnements virtuels et plus particulièrement les langages de description de scène parmi lesquels COLLADA. Puis, nous décrirons les ajouts que nous avons fait à la syntaxe COLLADA pour prendre en compte la description sonore, qu'elle soit statique ou dynamique (dépendante ou non de paramètres externes). Avant de conclure, nous présenterons les différentes expérimentations qui nous ont permis d'évaluer notre proposition d'un point de vue conception sonore. En particulier, nous nous attarderons sur les développements faits dans le cadre du projet Terra Dynamica.

E.2. Etat de l'art

Le cadre expérimental de notre travail est celui des environnements urbains. C'est plus précisément la genèse de notre étude puisque l'on recherchait une méthode pour adjoindre une description sonore à un environnement urbain visuel précédemment développé. De multiples solutions se sont offertes à nous pour sonoriser un environnement 3D. Les solutions étudiées se séparent en deux catégories, les produits commerciaux que l'on peut détourner pour notre propre besoin et les solutions techniques basées sur le paradigme de graphe de scène.

E.2.1. Environnements virtuels urbains

Les notions de Réalité Virtuelle (RV) ou d'Environnements virtuels se réfèrent à des environnements numériques créés par ordinateur qui représentent la simulation du monde réel ou la création de mondes imaginaires. Intégrant la capacité d'interaction avec l'utilisateur, les technologies de RV peuvent être appliquées dans une grande variété de domaines, tels que le divertissement, l'éducation, les arts, la formation, l'évaluation et le test de produits, etc. Les jeux vidéo forment un genre très représentatif de médias interactifs virtuels.

Les sentiments d'immersion, d'engagement et de présence ont été vus comme composants de plaisir ainsi que comme aspects cruciaux dans l'analyse des environnements interactifs en trois dimensions [DH00][M03]. C'est-à- dire que le joueur doit se sentir dans un monde ouvert et interactif, et devrait être accompagné vers la solution du jeu. Pour résoudre ce paradoxe, l'industrie du jeu a inventé plusieurs techniques dérivées de la théorie des jeux et la spécification orientée objet [VLN02].

Le besoin de représentations du son urbain provient de problème d'écologie acoustique et de pollution sonore. Deux principaux outils de cartographie numériques linéaires résultent de ces recherches : les cartes sonores (enregistrements statiques sur le terrain ou les enregistrements dynamiques d'itinéraires sonores) et des cartes de bruit (représentant la distribution du niveau sonore et d'attributs qualitatifs déduis). Les représentations cartographiées sont la plupart du temps un contenu auditif empirique superposé sur l'espace mappé. Considérant la ville comme l'espace, sa relation au son peut être comprise comme une interaction dynamique ; d'abord à travers les paradigmes d'analyse et de composition de la théorie des paysages sonores, puis en ce qui concerne les outils d'architecture acoustiques numériques et enfin comme une construction en temps réel au sein des jeux vidéo.

L'analyse que nous avons pu faire des applications représentatives du son dans des villes virtuelles (Google Earth augmentée d'une couche sonore, Second Life, le jeu Prototype [M08], Soundwalk et Locustream) montre que les textures, le rythme et la discontinuité sont des fonctionnalités spécifiques de ce domaine d'application sonore. Par ailleurs, l'organisation dynamique du paysage sonore est basée sur les besoins créés par le gameplay ou par les règles du média.

Le terme *soundscape* que l'on utilise pour nommer les paysages sonores urbains est la combinaison de son (*sound*) et de paysage (*landscape*). Il a été inventé par le compositeur canadien R. Murray Schafer [S77]. Selon lui, un environnement sonore est dépeint à l'aide de trois couches : les signaux sonores, les marqueurs sonores et les thèmes sonores. E. Weis et J.

Bolton utilisent dans leur taxonomie sur les sons de films ces mêmes catégories renommées respectivement premier plan, plan médian et arrière plan [WB85]. Cette classification qui peut être faite de manière sémantique ou selon des critères de distances est particulièrement utilisée dans la gestion du niveau de détails sonores pour aller au-delà du simple *clustering* de sources sonores [TGD04][SCB11].

FIGURE E-7. La carte sonore de Loocustream.

E.2.2. Les langages de description de scènes

Une méthode commune pour gérer un environnement audio-vidéo est d'utiliser un graphe de scène qui est une organisation des données sous forme d'arbre qui décrit le monde virtuel et les objets qui le composent. Chaque nœud de la hiérarchie peut représenter des propriétés d'un objet, des transformations géométriques, des comportements qui peuvent être déclenchés par des événements ou des instructions de rendu. De nombreux standards tels que VRML/X3D ou MPEG4 s'appuient sur les graphes de scène pour organiser hiérarchiquement les différents éléments. Les contenus multimédia interactifs riches (i.e., tous les supports numériques interactifs mixant audio, vidéo, texte, graphiques et animation de synthèse) sont de plus en plus utilisés dans des domaines allant de l'information, au matériel pédagogique, en passant par les jeux vidéo [SVS06]. Dans ce contexte, un langage de description de scène qui permet la synchronisation inter media complexe et la création de contenus interactifs est nécessaire.

La notion de langage de description de scène fait référence à tout langage de description apte à décrire une présentation audiovisuelle composite à partir de plusieurs éléments média. Ce langage peut être interprété par un programme de rendu qui va générer, que ce soit en temps réel ou en batch, une (image animée et son) perceptive représentation de la scène.

Parmi les principaux langages de scène gérant les sons, citons SMIL, VRML/X3D et MPEG-4 BIFS qui gèrent des descriptions de fois visuelles et audio, et d'autres comme SDIF, SpatDIF, ASDF et XML3DAUDIO ne gérant que le son. Dans ce paragraphe, nous proposons une vision globale des capacités sonores de chacun de ces formats et discutons des fonctionnalités manquantes pour apte à répondre à la majorité des besoins actuels.

Synchronized Multimedia Integration Language (SMIL) est un langage de balise du World Wide Web Consortium (W3C) basé sur XML destiné à la description des présentations multimédia interactive. Il est apte à intégrer des Comportements temporels et spatiaux d'objets multimédias. La dernière recommandation du W3C pour SMIL est la version 3.0 sortie en 2008.

La syntaxe et la sémantique SMIL peuvent être réutilisées et combinés avec les autres standards basés sur XML (par exemple, SVG, XHTML, VoiceXML, MusicXML, X3D), dans le but de traiter des graphismes en 2D, des textes mis en forme ou la représentation du temps et de la synchronisation. Toutefois, ce mélange de description peut conduire à des problèmes d'interopérabilité. Ce pourrait être le cas si l'on traite un contenu riche qui mêle graphiques 2D/3D, texte et vidéo [CD02].

La norme SMIL définit et gère seulement des scènes de rendu en deux dimensions. Il n'y a qu'un seul paramètre contrôlable dans l'objet audio, c'est le niveau sonore. Pihkala et al. ont proposé une extension, qui est dénommée Advanced Audio Markup Language (AAML), supportant l'audio 3D pour SMIL [PL03]. Avec AAML, une troisième dimension est ajoutée aux coordonnées 2D de SMIL avec un système de mise en page similaire en ajoutant devant, le dos et les attributs de profondeur.

Le VRML, Virtual Reality Modeling Language, est un modèle de programmation de graphe de scène définie pour représenter la réalité virtuelle en 3D ainsi que d'un format d'échange universel pour les graphiques et objets multimédias intégrés. La dernière version du format est la 2.0. En 1997, VRML est devenu une norme internationale ISO / IEC 14772, également appelée VRML97.

Depuis cette date VRML a été remplacé par X3D (eXtensible 3D). Contrairement à VRML qui est repose sur une syntaxe propre, X3D exprime des scènes au format de fichier XML. Chaque application X3D représente un espace interactif 3D temporel et audiovisuel. Le rendu des contenus multimédia peut être modifiés dynamiquement par divers mécanismes comme les scripts et les routes (propagation d'évènements).

Les sons, dans X3D, indices sonores et ambiance, sont spatialisés. X3D introduit deux types abstraits de nœuds audio, Sound et AudioClip. Le nœud Sound décrit le flux sonore généra par le rendu alors que AudioClip est la description d'usage des sources sonores. Chaque noeud possède plusieurs champs qui identifient les comportements particuliers de l'objet. En 2001, Topol et al. ont proposé une amélioration de la description sonore avec une définition précise du cône de "Shape" ainsi qu'une amélioration de la notion de son ambiant [TS01].

MPEG-4 est originellement une norme de description des contenus multimédia interactifs et de protocoles de haut niveau. Elle visait des applications dans la vision de la télévision connectée telle qu'elle était envisagée dans les années 90. MPEG-4 inclus le codage d'objets audio-vidéo pour les médias en streaming web et la distribution de CD, les applications de diffusion de la voix et de la télévision, mais aussi le stockage des données numériques. Parmi les langages de description de scène actuels, MPEG-4 fournit probablement le langage la scène la plus avancée du point de vue des contenus sonores [S98] [SVH99] [SYE00]. MPEG-4 se compose de plusieurs «parties» (normes), dont la partie 3, «MPEG-4 Audio», est composé d'un ensemble de technologies de compression pour le codage perceptif des signaux audio. MPEG-4 Audio s'applique à une large variété d'applications qui peuvent aller du discours intelligible au son muticanal de qualité et des sons naturels aux sons synthétisés.

Le format binaire pour les scènes (BIFS) est composé d'objets à la fois en 2D et 3D reliés entre eux pour former le graphe de scène d'un monde MPEG-4. Une grande partie de la structure est héritée de VRML, et donc, BIFS est presque équivalent au graphe de scène VRML. Bien qu'il n'existe aucune distinction technique entre AudioBIFS et le reste du BIFS, AudioBIFS est considéré comme un sous-graphe audio de BIFS. En contraste avec le graphe de scène visuelle qui représente la relation géométrique entre les objets graphiques et de l'espace d'arrière-plan, une scène audio représente un graphique du signal décrivant manipulations, de traitement du signal numérique [SVH99].

Cette norme permet de décrire une salle d'audition virtuelle qui prend en compte l'environnement acoustique et le traitement des effets: la propagation, la réflexion, la réverbération, l'atténuation, l'absorption de l'air, effet Doppler, etc... Deux algorithmes distincts sont présentés pour aborder la spatialisation audio. La première technique est la modélisation physique des environnements acoustiques. La seconde technique est la modélisation de perception de l'environnement acoustique, où la perception spatiale du son est étudiée.

Dans la prolongation des BIFS, The Sound Descriptif Interchange Format (SDIF) est un standard largement utilisé dans la communauté informatique musicale pour l'échange d'une variété de descripteurs sonores [FADE]. SpatDIF, le descripteur de son spatial. [SP11b] [SPAT]. L'idée générale est de créer une méthode (sémantique et syntaxique) ainsi que les meilleures pratiques mises en œuvre pour stocker et transmettre des descriptions de scènes sonores spatiales entre les applications audio en temps réel et non en temps réel.

La structure de SpatDIF repose sur un modèle en deux couches. La couche de base contient les informations minimales sur le rendu de la scène audio. Les entités de base et les descripteurs au sein de cette couche doivent être compris par tous systèmes de rendus conforment à la norme SpatDIF. La couche supérieure définit des extensions qui fournissent des descriptions supplémentaires organisées par fonctionnalités. Elles incluent des instructions de transformation ou des informations sur la description de premier niveau, des informations de modélisation acoustique (et des informations de configuration de restitution (Binaural et extensions Ambisonics).

Audio Description de la scène Format (ASDF), basé sur XML, repose sur la nature de l'enregistrement sonore considéré du point de vue de la reproduction [GS08] [GAS10]. Cette représentation basée sur les objets d'une scène audio est destinée à éviter les inconvénients de stockage basés sur un canal défini et de donner plus d'efficacité et de flexibilité pour les méthodes de reproduction audio à haute résolution modernes, comme le Haut-commande Ambisonics (HOA) et Wave Field Synthesis (WFS). L'ASDF est une description de la scène audio pure, sans aucun contenu graphique. Il se compose de fonctions statiques et dynamiques dans les scènes audio tournée vers l'interactivité de l'utilisateur et des performances artistiques. Le format est toujours en cours de développement en parallèle avec le moteur de rendu SoundScape (SSR).

XML3DAUDIO, comme son nom l'indique, est un système orienté objet basé sur XML pour la description et le rendu de scènes AUDIO animés en 3D dans laquelle chaque objet a son propre ensemble de paramètres. Elle englobe des fonctionnalités élémentaires de

description audio (par exemple, la directivité des sources sonores, la réflectivité de matériau acoustique, la définition de la réverbération de la pièce) qui se trouvent en MPEG-4 AABIFS mais vise à fournir une utilisation plus simple que les AABIFS.

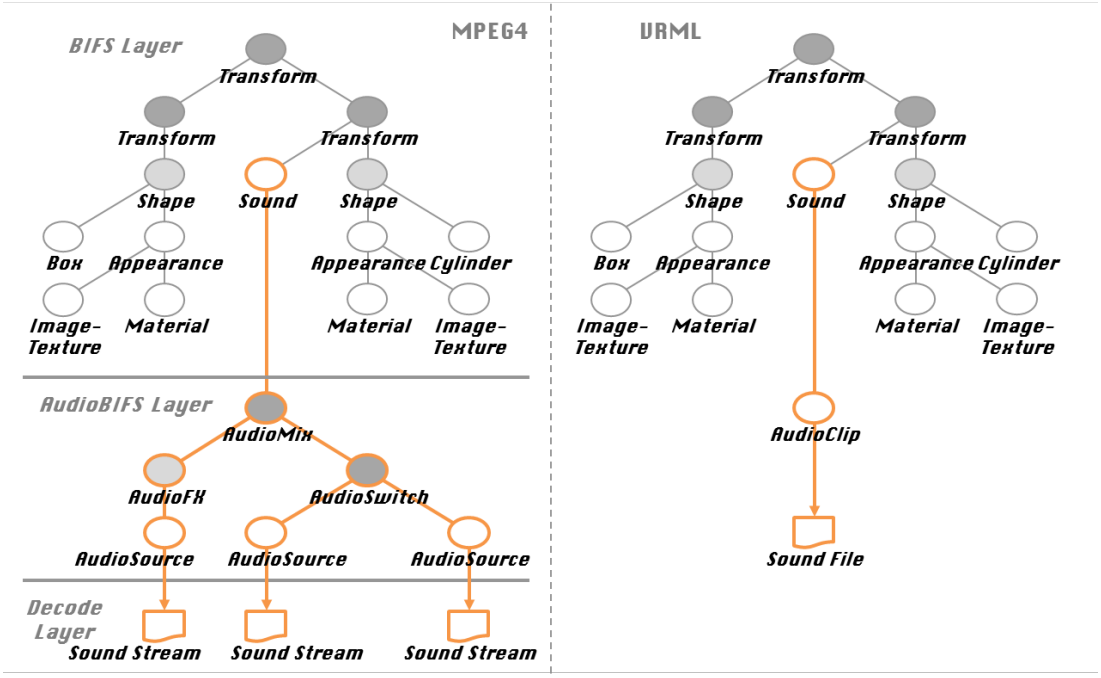


FIGURE E-8. Une scène MPEG4 et VRML [TS01]

E.3. Le son dans COLLADA

E.3.1. Principes généraux du schéma COLLADA

COLLADA est un langage intermédiaire pour transporter des données entre les différentes applications 3D interactives. C'est également un format d'échange neutre entre plusieurs outils d'édition, telles que les applications d'information géographique (par exemple, Google Earth, ArcGIS), des systèmes de stockage de ressources multimédia (par exemple,

3DVIA, Google Warehouse), les moteurs 3D Web (par exemple, Papervision3D, Google O3D, Bitmanagement), et moteurs de jeux (par exemple, Unity3D, Ogre3D, Torque 3D).

COLLADA est non seulement largement utilisé dans les applications de bureau, mais il est aussi adapté pour travailler des contenus 3D pour le web reposant sur son utilisation propre des technologies web (i.e., la syntaxe XML et Resource Identifiers URI). Le domaine cible des applications interactives de COLLADA, c'est l'industrie du divertissement, en particulier le jeu vidéo, dont le contenu est essentiellement en trois dimensions. D'autres types d'applications qui nécessitent le même type de technologies bénéficieront indirectement de COLLADA.

Initialement, seule la scène visuelle faisait partie de la conception de COLLADA. Plus tard, un autre domaine appelé physique a été introduit pour gérer les moteurs physiques à partir d'un modèle COLLDA Une extension cinématique est incluse dans la version la plus récente (1.5.0) de la norme COLLADA,.

Actuellement, La scène COLLADA peut comporter une scène visuelle et un nombre quelconque de scènes physiques et la cinématique des scènes. Les trois modèles sont indépendants mais il existe une correspondance entre les composants correspondants basée sur les nœuds constructeurs et leurs systèmes de coordonnées.

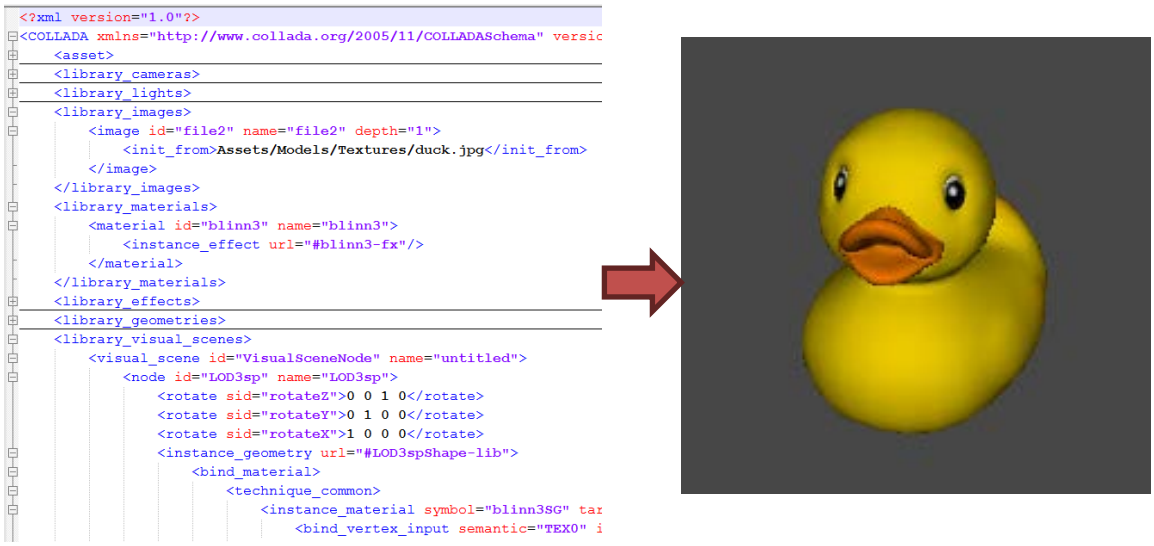


FIGURE E-9. La description d'un canard en COLLAD.

E.3.2. Objectifs de notre proposition

Le cœur de nos travaux est de proposer une scène audio définie et pouvant être mises en œuvre dans un moteur de rendu de COLLADA.

Le principe de cette scène doit répondre à plusieurs objectifs et rester dans l'esprit le principe du schéma COLLADA. Les objectifs de conception du schéma sonore de COLLADA sont les suivantes:

- C'est une extension du standard qui doit rester dans la forme et le fond cohérent et compatible avec cette norme.
- Il doit fournir un format de langage standard, compatible avec le standard actuel de sorte que les contenus COLLADA existants puissent être utilisés et que le son puisse y être facilement ajouté.
- Il doit éviter de devoir décrire des données que l'on peut déjà préciser dans le standard actuel, en particulier la géométrie de la scène sonore.
- Il doit supporter les fonctionnalités sonores des langages de scènes existant, en particulier MPEG4.
- Il doit être également capable de supporter les technologies de synthèse, de traitement et de rendu sonore développée les plus récentes.

Parmi les langages de description de scène existants, MPEG-4 BIFS propose dans doute la structure de scène auditive la mieux fondée. En nous inspirant du concept des AudioBIFS, nous proposons un graphe audio pour les objets de la scène COLLADA. En contraste avec le graphe de scène visuelle qui représente la relation géométrique entre les objets graphiques et l'espace d'arrière-plan, une scène audio représente une représentation symbolique du signal décrivant la synthèse et les transformations du signal numérique [SVH99]. La chaîne de traitement va de bas en haut dans le sous-graphe audio. Chaque élément fils émet les données vers un ou plusieurs noeuds parents du niveau supérieur. Les feuilles et les nœuds intermédiaires ne sont donc pas des sources sonores; seul le résultat produit par un objet audio, racine d'une sous-arborescence audio, est audible. Bien entendu les objets audio peuvent être associés à des nœuds visuels.

En outre, nous proposons de pouvoir prendre en compte la structure du paysage sonre, sous la forme de plans, de profondeur sonore et de niveau de détail dans la description de la scène. Ces concepts doivent pouvoir être utilisé dans un sens « acoustique » objectif (distance des sources à l'auditeur) mais également subjectif. Il doit être possible d'établir des priorités particulières pour certaines sources sonores tout en les attribuant à des plans du paysage sonore ou en fonction d'un processus de regroupement des sources (clustering) qui traite le niveau de détail.

E.3.3. La scène statique

Selon notre proposition, COLLADA est divisé en quatre scènes parallèles: audio, vidéo, de la physique et de la cinématique. Chaque sous-scène est essentiellement une description statique composée de noeuds hiérarchisés. Elles sont indépendante les unes des autres, mais les nœuds, d'une scène à l'autre, peuvent être associés et partager certaines propriétés.

Dans ce paragraphe, nous présentons la structure de la partie statique de notre proposition audio pour COLLADA. Nous décrivons les nœuds audio et la scène audio composée de ces nœuds ordonnés hiérarchiquement.

La scène COLLADA audio est construite selon trois classes de nœuds audio.

La première sont des éléments de flux audio qui génèrent et stockent les données des flux sonores tout au long de la chaîne de traitement du signal. Parmi ceux-ci on trouve `<audio_source>`, `<audio_buffer>`, `<audio_delay>`, `<audio_mix>` et `<audio_switch>`, dont les fonctionnalités et la sémantique sont héritées de MPEG-4 AudioBIFS.

Effet Audio constitue la seconde classe de nœuds audio. Contrairement aux nœuds de flux audio transportant des données sonores, les éléments d' Effet Audio fournissent le contrôle des effets sonores qui peuvent être appliqué sur les flux audio tels que des transformation DSP et / ou la réverbérations et la spatialisation.

La dernière classe est dénommée Scène Audio. On y introduit un noeud `<audio scène>` qui spécifie l'environnement dans lequel les objets acoustiques sont instanciés et simulées. La scène audio englobe l'ensemble des informations qui peuvent être sonorisés à partir du contenu des ressources COLLADA.

La structure hiérarchique des nœuds audio de COLLADA est illustrée à la figure suivante. Les éléments sur la gauche sont les nœuds de flux audio structurés en trois couches. Les données audio sont acheminées de bas en haut. Au niveau le plus bas, les noeuds et des feuilles intermédiaires effectuent des opérations mathématiques sur les signaux audio. Au niveau intermédiaire les nœuds sonores servent de jonctions et de structuration organisent en passant le long des flux les paramètres à l'objet audio racine qui présentera le résultat à l'auditeur. Sur la partie droite, les nœuds d'effets audio, y compris `<audio_dsp>` qui s'applique effets DSP sur les flux audio et `<acoustic_environment>` qui ajoute une qualité perceptive à toute la scène audio, sont conçus pour décrire les couleurs acoustiques afin d'obtenir une sensation d'espace sonore convaincant.

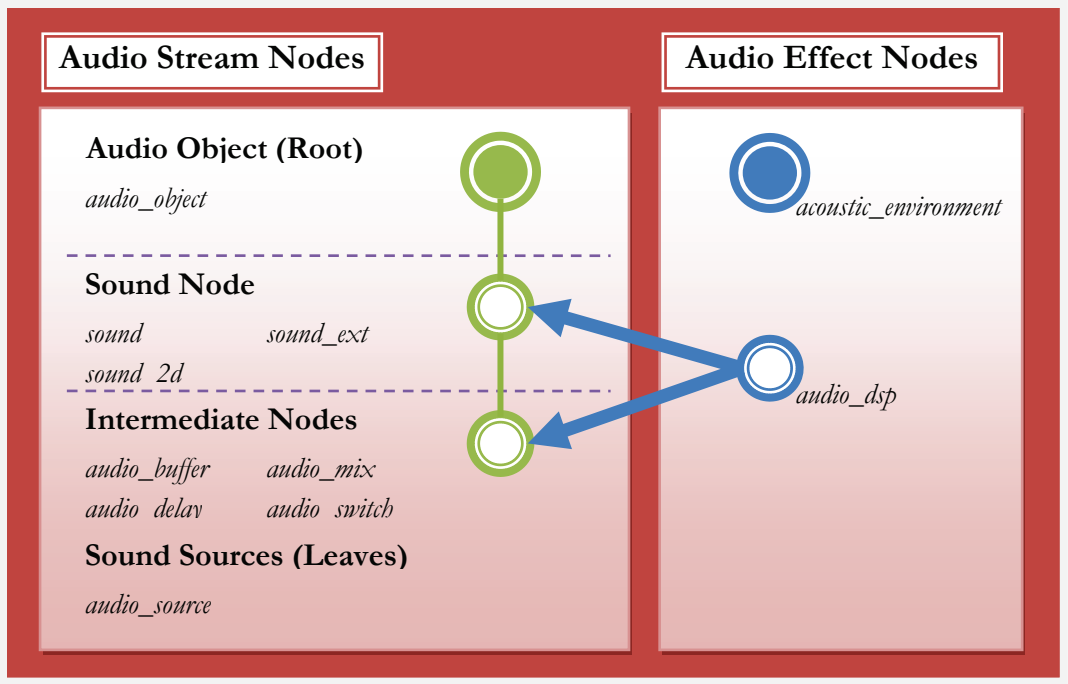


FIGURE E-10. La structure hiérarchique des noeuds COLLADA.

E.3.4. La scène dynamique

E.3.4.1. Principes généraux

Alors que les graphes de scènes sont utilisés pour des applications interactives très complexes, comme les jeux vidéos, leur capacité de description de comportement dynamique est souvent très limitée. Par exemple le principal événement de VRML est lié à la gestion de la souris. Nous avons désiré proposer un traitement plus avancé des comportements dynamiques sonores.

A chaque objet d'un mode fictionnel est associé un certain nombre de sons qu'il produirait lors d'une interaction [F06]. Un univers sonore est donc construit sur l'interactivité. En conséquence, nous sommes à la recherche d'un descripteur de scène permettant de décrire les comportements sonores dynamiques.

Influencé par de nombreux facteurs, tels que les échelles chronologiques, l'emplacement de l'avatar ou la nature des interactions, la gestion adaptative du son dans les villes virtuelles doit être capable de se placer entre les différents plans du fond au premier plan via le plan moyen, et vice versa. L'état « sonore » d'un objet peut changer dynamiquement lors du déclenchement d'événements [C08B]. Les descripteurs pour les contrôles de comportement dynamique dans COLLDA sont donc nécessaires et relativement complexes.

COLLADA animation prend en charge le contrôle dynamique de la représentation visuelle, mais ceci correspond essentiellement à des animations linéaires (cinématiques). Il était également possible d'utiliser une fonctionnalité d'extensibilité de COLLADA appelée <extra>. Elle peut être appliquée à presque tous les types d'éléments COLLADA afin de décrire des informations supplémentaires quelconque. Mais dans la syntaxe et l'esprit cette possibilité est orientée vers la description d'attributs. Il est donc difficile d'utiliser ce type de technologie de scène graphique classique pour décrire une gestion complexe de contenu dynamique. Nous proposons donc une solution alternative reposant sur un langage de haut niveau (intégré ou externe). Ce langage de script peut être considéré comme une interface entre COLLADA et l'API audio. Cette solution peut être appliquée non seulement sur l'audio, mais également le visuel, la physique et d'autres capacités de la scène COLLADA.

E.3.4.2. Les événements

Le statut du son dans des villes virtuelles peut être influencé par de nombreux facteurs. Il peut s'agir, par exemple, des facteurs environnementaux comme la saison, la météo ou la pression d'air. Les relations physiques, comme les collisions entre les objets, peuvent également produire des effets sonores. En outre, il dépend de nombreux aléas externes, essentiellement les actions du joueur. Pour cette raison, nous avons proposé une classification des évènements qui n'est pas limitative et tente d'être exhaustive

E.3.4.3. Les Fonctions

Quand un événement est déclenché, il peut produire une série d'actions simultanées dans la scène. Le noeud `<function>` est un macroélément qui permet de décrire ces complexes actions. Une fonction a la visibilité de tous les descripteurs de son sous arbre. Ceci permet, par exemple, d'altérer une texture, les paramètres d'un DSP ou une acoustique. Chaque fonction peut être appelée par un `<call_function>`. Une fois qu'une fonction est appelée (généralement à la suite d'un évènement), Les actions déclarées dans cette fonction sont exécutées.

E.3.4.4. Exemple

Le tableau suivant est un exemple d'utilisation des fonctions dynamique. Un `<audio_object>` peut être lu automatiquement ou déclenchée par un événement, comme un clic souris ou une collision. Les méthodes de lecture peuvent jouer, arrêter ou interrompre l'objet audio sélectionné.

Ceci est illustré dans l'exemple. Lorsqu' un événement de souris - "LeftClick" est déclenché, deux fonctions - "ringDoorbellFunc" et "openDoorFunc" sont appelées. En d'autres termes, lorsque l'utilisateur clique sur la représentation visuelle de la porte, la sonnette sonne (l'objet audio de la sonnette joue) et la porte s'ouvre (le nœud visuel porte pivote de 90 degrés sur l'axe des y).

```
<library_visual_scenes>
```

```
    <visual_scene id="DefaultVisualScene">
       <node id="doorNode">
          …
          <extra>
             <technique profile="SHCHAN">
                <mouse_event type="LEFTCLICK">
                   <call_function url="#ringDoorbellFunc"/>
                   <call_function url="#openDoorFunc"/>
                </mouse_event>
             </technique>
          </extra>
       </node>
    </visual_scene>
</library_visual_scenes>

<scene>
    <instance_visual_scene url="#DefaultVisualScene">
       <extra>
          <technique profile="SHCHAN">
             <function id="openDoorFunc">
                <ref_node ref="doorNode">
                   <rotate sid="rotateZ">0 0 1 0</rotate>
                   <rotate sid="rotateY">0 1 0 90</rotate>
                   <rotate sid="rotateX">1 0 0 0</rotate>
                </ref_node>
             </function>
          </technique>
       </extra>
    </instance_visual_scene>
    <instance_audio_scene url="#DefaultAudioScene">
       <extra>
          <technique profile="SHCHAN">
             <function id="ringDoorbellFunc">
                <ref_audio_object ref="#doorbellAudObj">
                   <control>PLAY</control>
                </ref_audio_object>
             </function>
          </technique>
       </extra>
    </instance_audio_scene>
</scene>
```

E.4. Mise en œuvre et évaluation

E.4.1. Introduction

A partir de la spécification de COLLADA audio présentée précédemment, nous sommes maintenant en mesure de mettre en œuvre la scène auditive COLLADA pour représenter son dans les environnements virtuels 3D. Ce travail à pour objectif démontrer la faisabilité de notre architecture. Nous avons commencé par deux implantations test sou Ogre et Unity. Deux versions finalisées sous Thalesview et CryEngine.

E.4.2. Pipeline et architecture

Les constituants de COLLADA, dans notre proposition sont organisés en quatre classes: le visuel, la physique, la cinématique et le son. Les rendus de ces différents éléments, et en particulier le son, par rapport aux autres, sont relativement indépendants. Dans la pratique ce parallélisme se traduit au niveau le plus haut dans l'architecture d'implantation et au niveau le plus bas l'utilisation de processeurs spécialisés La figure suivante illustre le processus parallèle de traitement des descripteurs COOLADA d'une scène virtuelle.

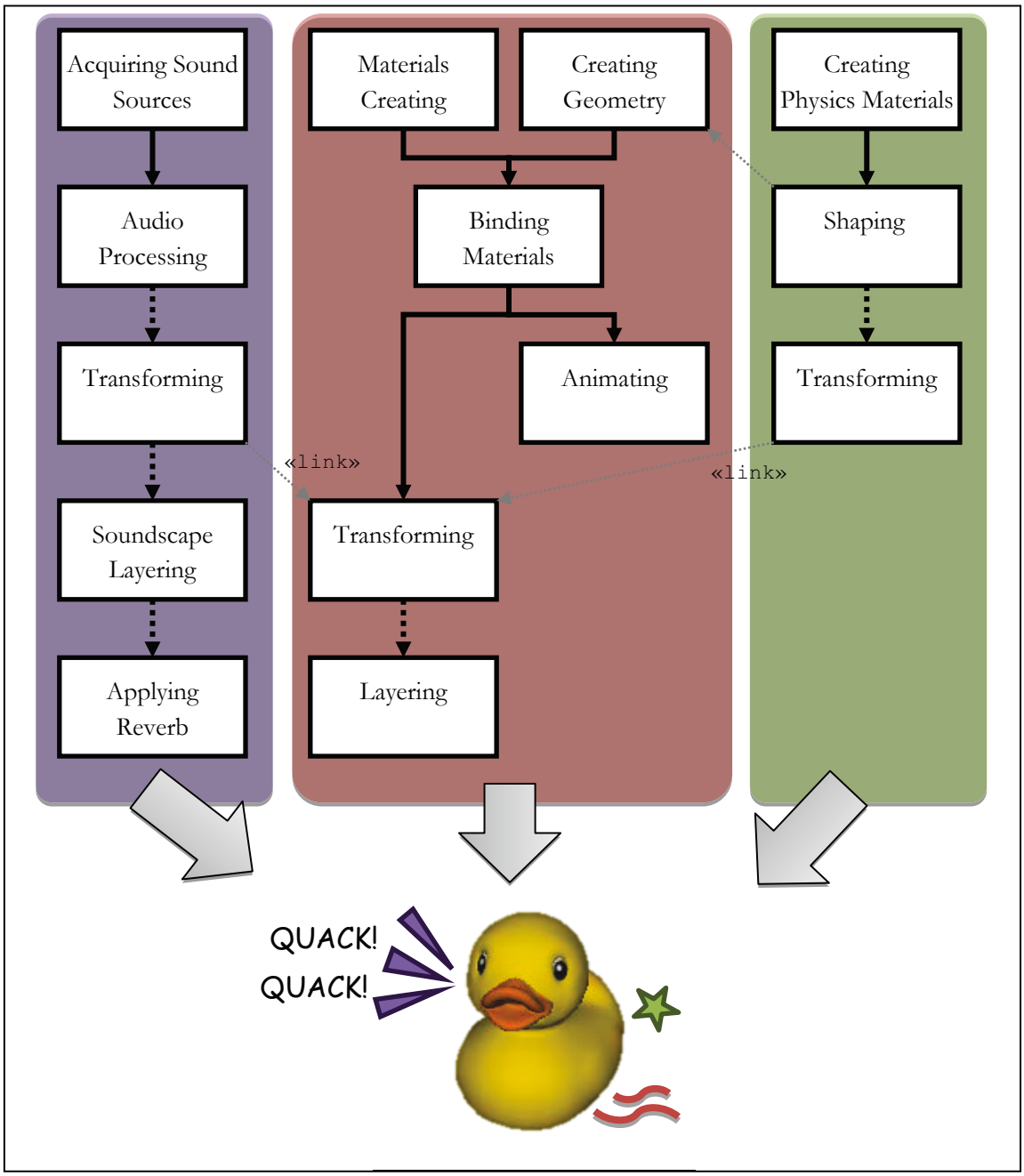


FIGURE E-11. Parallélisassions des processus de rendu d'un modèle COLLADA.

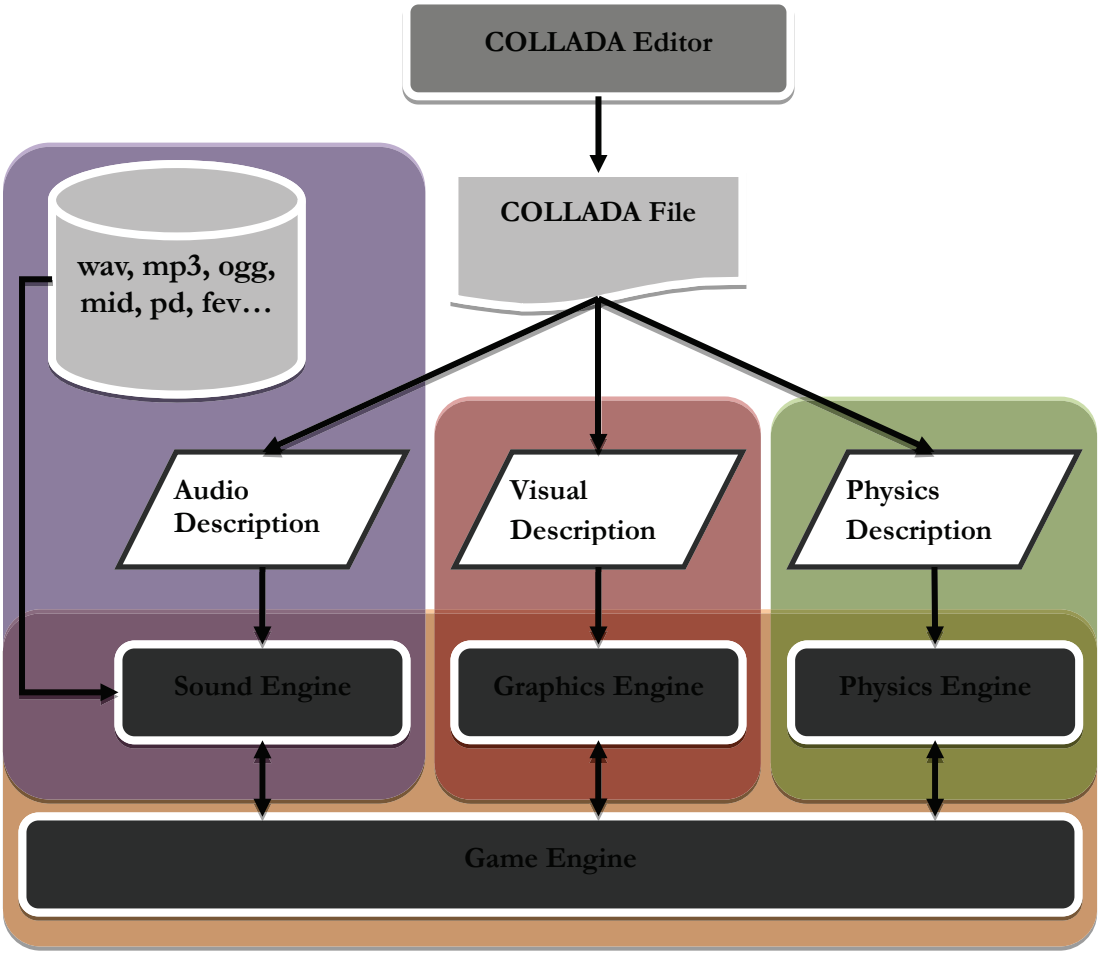


FIGURE E-12. Workflow de l'utilisation de fichiers COLLADA dans le développement de jeux.

La source de la description de scène est l'éditeur qui est un outil destine aux artistes praticiens du domaine concerné. Par exemple pour la création graphique et l'animation cette fonction est assurée par des outils graphiques informatiques, tels que Maya et 3ds Max, pour faire des animations, des images, des modèles et des effets visuels. Ils peuvent importer et exporter des fichiers COLLADA et offre une création et un mode de mise au pont des contenus graphique.

Pour les applications de jeux vidéo ces descriptions de scènes, contenues dans un fichier COLLADA, peuvent être réutilisé dans un moteur de jeu qui possède souvent son éditeur. Ceci permet de compléter et d'adapter la scène pour des applications interactives.

Il est indispensable de un éditeur comparable pour les concepteurs sonores pour créer des contenus audio et les exporter au format COLLADA. La nature de l'interface de saisie, adaptée à la composition spatiale et au sound design est un problème de recherche en soi.

Les moteurs de jeux principaux, tels que Unity3D, Unreal Engine et CryENGINE fournissent un environnement de programmation permettant de créer les éléments de jeu. Ils sont constitués d'un ensemble de bibliothèques logicielles, dont chacun exécute une série de fonctions pour gérer les comportements dynamiques et interactifs. Chaque bibliothèque gère des fonctions (moteurs dédiés) relatives à une technologie utilisée (par exemple, le graphique, l'animation, la physique, l'IA, les réseaux). Elles sont intégrées et communiquent les unes avec les autres. Les fonctionnalités de base comportent un moteur de rendu graphique 3D, un moteur sonore, un moteur physique. Le moteur de jeu génère un exécutable qui comprend, outre le code développé et les bibliothèques, un moniteur temps réel d'exécution. Les principaux moteurs peuvent générer des codes dépendant de la plate-forme cible. Par conséquent les jeux développés avec ces moteurs de jeu peuvent être, en principes, portés sur plusieurs plates-formes (ordinateurs, les consoles et / ou téléphones mobiles).

La scène COLLADA est analysée et ses différents éléments sont transformés en formats internes qui sont utilisés par les moteurs dédiés correspondants. COLLADA agit donc comme un format intermédiaire dans le pipeline de contenu pour les applications interactives. Cela signifie que les contenu complexes peuvent être stockés dans le format COLLADA et ensuite traitées à l'exécution de l'application par le moteur de jeu. Cette mécanique de traitement des données est illustrée par la figure précédente.

Le schéma de l'architecture du moteur son que nous proposons est illustré par la figure suivante. La partie supérieure de ce qui est importé d'un moteur sonore existant et le contrôle de la synthèse en temps réel pour une ambiance de fond est exploité séparément de gestion COLLADA, qui se trouve dans la partie droite de la figure.

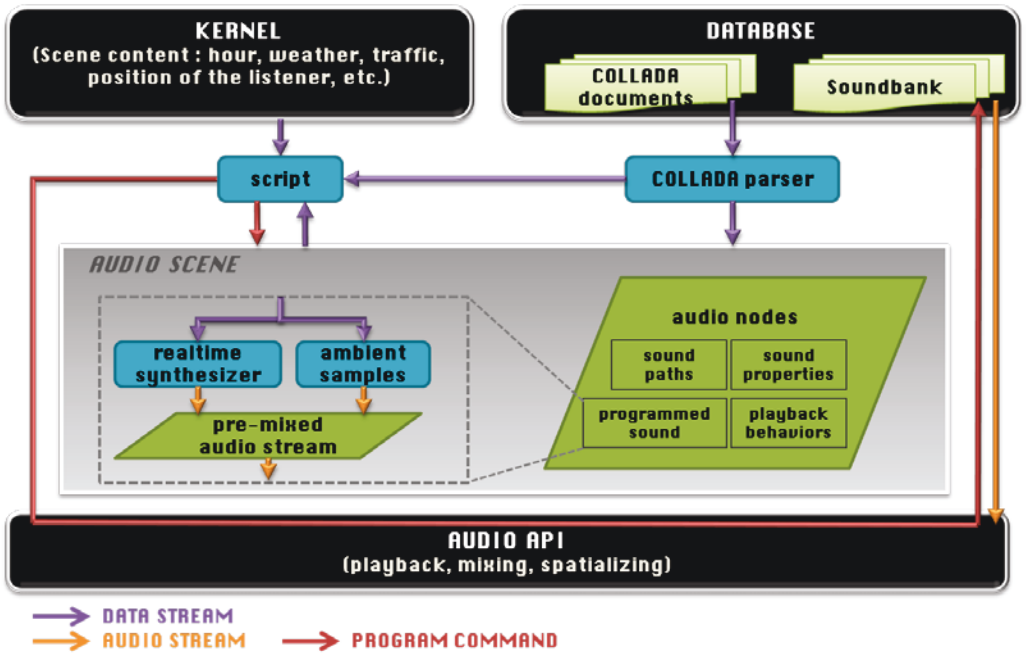


FIGURE E-13. Intégration de COLLADA dans l'architecture du moteur sonore.

Cette structure permet d'éviter de programmer la totalité de l'interactivité sonore dans le moteur de jeu en reportant dans les scripts interactifs sonore issus d'un éditeur son. Le plan dans le fond explique comment COLLADA enrichi peut intégrer les ressources sonores en encapsulant des fonctions sonores étendues (y compris les patchs Pure Data et objets d'événements FMOD).

Cependant, contrairement aux formats de livraison comme X3D qui sont destinés à contenir des descripteurs procéduraux et de rendu pour des applications interactives, COLLADA fournit un langage standard pour décrire les éléments 3D, mais ne décrit pas leur sémantique d'exécution [AP07]. Cette définition et sa mise en œuvre (façon dont le contenu sera utilisé) sont laissées à l'auteur et aux outils de rendu. Un outil compatible COLLADA, comme Papervision3D ou Unity3D, a généralement sa propre solution pour compiler les données source stockées dans COLLADA sous une forme exécutable. Il n'est pas de notre intention de modifier cette position occupée par COLLADA dans le pipeline de production. En conséquence, même si toute l'intelligence est décrite en détail dans les COLLADA script, l'effort de réalisation est effectué par l'application importatrice (le moteur de jeu en l'occurrence).

E.5. Implantation du moteur son

E.5.1. Projet Terra Dynamica

Ce travail a été en partie réalisé dans le cadre du projet TerraDynamica Le but de ce projet FUI, est de donner vie à TerraNumerica, une ville virtuelle statique. Cette ville virtuelle, Paris dans l'expérience, est utilisée pour de nombreuses applications allant de la sécurité urbaine aux installations artistiques. Piloté par Thalès, le projet est porté par un consortium mêlant partenaires académiques et industriels, sur une période allant de 2010 et 2013. Donner de la vie à une ville virtuelle consiste essentiellement à y ajouter des personnages animés. Néanmoins, la crédibilité de piétons ou de voitures est très limitée en l'absence de son. La conception d'un moteur sonore interactif, d'un synthétiseur d'ambiances sonores urbaines ainsi que l'installation artistique « Le Promeneur Ecoutant » font partie des tâches assignées au CNAM dans ce projet. Dans le projet l'audio est utilisée dans les cinq principaux domaines d'applications (jeu et l'art, le design urbain et de l'architecture, de l'analyse de la sécurité, le transport interurbain, services de proximité).

La ville virtuelle de Terra Dynamica est construite à partir d'une base de données au format COLLADA, ce qui en fait une plateforme idéale pour nous de démontrer la représentation en COLLADA du son dans les villes virtuelles.

Le noyau logiciel de (dénommé MAC) de Terra Dynamica encapsule une quantité de modules internes qui effectuent les calculs de la simulation, y compris la représentation de l'environnement, la création et la destruction des acteurs virtuels, le moteur physique, la navigation, l'intelligence artificielle, le scénario de simulation.

Le moteur son, appelé "ControleurAudio", est considéré comme un module externe comme les autres contrôleurs orientés applications qui peuvent être appelés dynamiquement dans le séquenceur du noyau pour le moniteur d'exécution. De cette façon, il peut être synchronisé avec l'état du monde virtuel. Le ControleurAudio est construit sur l'architecture décrite en au paragraphe précédent. Il comprend un analyseur des fichiers COLLADA, qui analyse le contenu sonore de chaque document et le convertit en un arbre (construction scène graphique) de la scène audio. L' « AudioObjectManager » contenu dans le module sonore est utilisé pour gérer (construction / destruction et la lecture / de contrôle du volume) la collection d'objets sonores. Il utilise des arbres audio pour créer des objets sonores des trois types en fonction de leurs formats source.

Le moteur son est construit au dessus du système audio FMOD, principalement pour le contrôle de la lecture de signaux sonores de premier plan et les repères sonores de second plan. Pure Data est utilisé pour la production du fond sonore urbain ambiant. Dans ce cadre, le bruit de l'environnement peut être influencée à la volée par un certain nombre de facteurs, tels que le nombre d'acteurs (piétons et véhicules) dans la scène ou la position et l'angle du point d'écoute. Il existe plusieurs solutions pour intégrer Pure Data avec le ControleurAudio. Pour l'instant, elle est faite en utilisant une instance active de Pure Data qui échange des messages avec le ControleurAudio via les ports de socket TCP / IP.

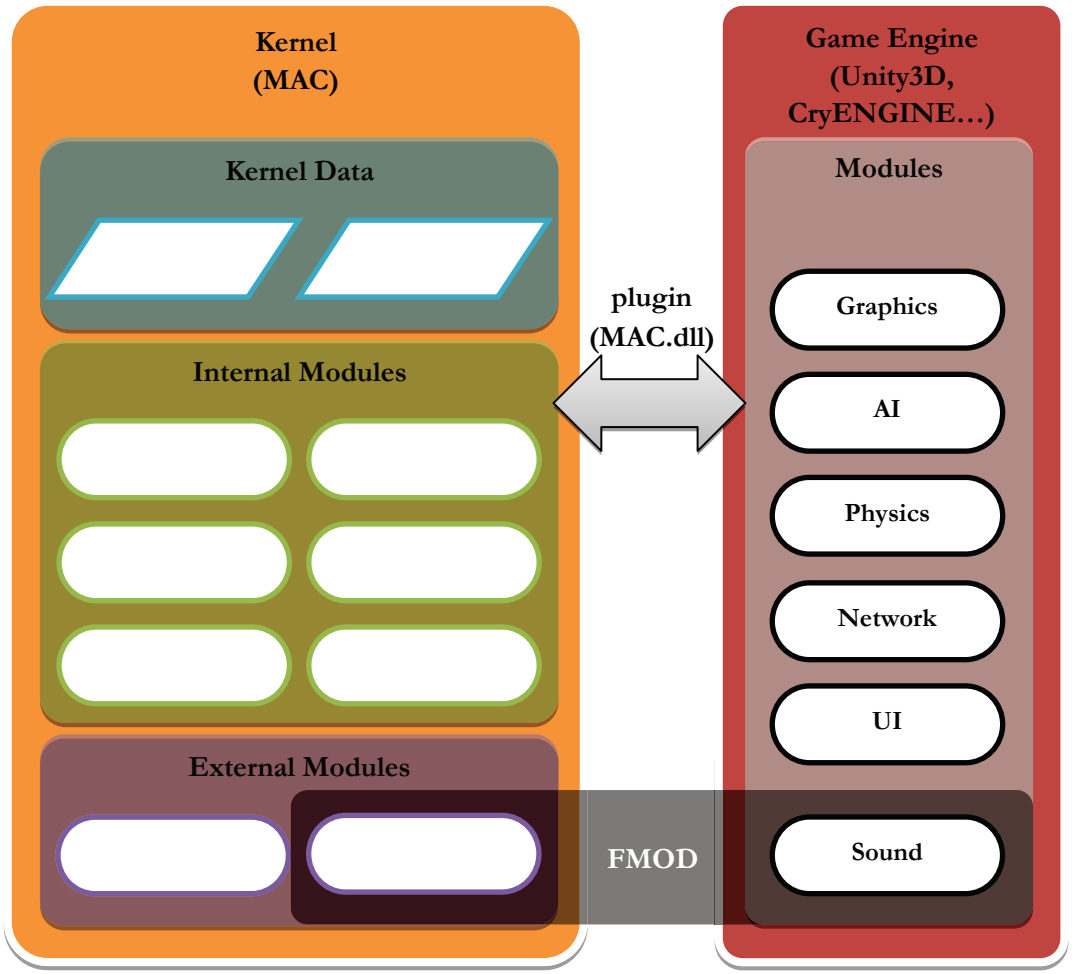


FIGURE E-14. Diagramme de classe de l'exécution du moteur sonore.

E.6. Travaux futurs

Cette thèse présente une représentation formelle du son dans les villes virtuelles à l'aide de la description de scène langue COLLADA.

La description sonore de base dans COLLADA a été validée par de premières expériences avec différents moteurs de jeu et les API sonore. La version la plus avancée a été réalisée par l'implémentation d'un moteur de son 3D construit sur le système audio FMOD. Elle a été utilisée dans le projet Terra Dynamica, une ville virtuelle dynamique.

Ns travaux est prévu pour être utilisés pour d'autres formes d'applications interactives, telles que la musique interactive ou des installations artistiques. En outre, COLLADA utilise les technologies web ("XML" pour la syntaxe et "URI" pour les identificateurs de ressources), ce qui en fait un bon choix pour les implémentations basées sur le Web. Outre l'utilisation dans les applications interactives natives (jeux par exemple, sur ordinateur ou console), COLLADA est bien adapté pour la livraison de contenu Web 3D [T10]. COLLADA et X3D (un format de distribution universelle pour les applications web) et peut être utilisés comme un ensemble d'outil puissant pour développer leurs applications d'entreprise et Web [AP07]. Certaines expériences ont prouvé qu'un analyseur partiel de COLLADA peut être mis en œuvre en JavaScript et en liant les deux normes Khrons - COLLADA et WebGL3. Il est donc prévisible que dans un proche avenir un standard web analogue qui fournira l'accès natif à l'audio et accélération matérielle sera définie (par exemple, OpenAL ou OpenSL ES pour le Web) et pourra être liée à la description de son COLLADA pour permettre la description de son web immersif et 3D. Comme le support de HTML5 et les technologies liées se développent, il est probable que de plus en plus d'applications web 3D seront prêtes à utiliser COLLADA directement comme format d'entrée.

Notre proposition peut être étendue en ajoutant des fonctionnalités avancées, comme par exemple les "matériaux acoustiques" (couleurs acoustiques) qui spécifie les caractéristiques auditives (propriétés, par exemple, de réflexion et transmission) pour traiter modèle acoustique par une approche physique.

Les mécanismes que nous avons proposé pour lier les scènes audio avec visuelles de COLLADA ne sont pas toujours nécessaires. La description sonore interactive proposée peut être utilisée de façon autonome. En particulier la structuration en plan sonores permet de gérer sons non diégétiques ou d'ambiances. Dans ce contexte, notre proposition peut servir de base formelle pour décrire les sons dans des villes virtuelles. Enfin le fond des

mécanismes proposés peut être étendu à d'autre standard ou d'autres langages et ne se limite donc pas au schéma COLLADA.

This page intentionally left blank.

Résumé:

Depuis de nombreuses années, des formats de fichier standardisés ont été conçus pour écrire, lire et échanger des descriptions de scènes 3D. Ces descriptions sont principalement faites pour des contenus visuels; les options accessibles pour les compositions audio des scènes virtuelles sont, dans les meilleurs des cas, pauvres et dans les pires, manquantes. C'est pourquoi nous proposons d'inclure une description sonore riche dans le COLLADA qui est un format standard pour d'échange d'assets numériques. La plupart des langages de description qui incluent une description sonore factorisent les éléments communs aux informations visuelles et sonores. Ces deux aspects sont par exemple décrits dans le même système de coordonnées. Cependant, dès lors qu'une description dynamique ou que des données externes sont requises, toutes les liaisons doivent être faites de manière programmée. Dans cette thèse, nous tentons de résoudre ce problème et nous proposons de donner plus de puissance créative aux sound designers même lorsque les scènes sont dynamiques ou basées sur de la synthèse procédurale. Cette solution est basée sur le schéma COLLADA dans lequel nous avons ajouté la description sonore, des capacités de scripting et des extensions externes. L'utilisation de ce langage COLLADA ainsi augmenté est illustrée à travers la création d'un paysage sonore urbain.

Mot clés: COLLADA, langage de description de scène, graphe de scène sonore, soundscape, villes virtuelles

Abstract:

Standardized file formats has been conceived since many years to write, read, and exchange 3D scene descriptions. These descriptions are mainly for visual contents whereas options given for audio compositions of virtual scenes are either lacking or poor. Therefore, we propose to include a rich sound description in the COLLADA, which is a standard format for exchanging digital assets. Most scene description languages with a sound description factorize common elements needed by the graphical and auditory information. Both aspects are, for example, described with the same coordinate system. However, as soon as a dynamic description or external data are required, all the glue must be done by a programming approach. In this thesis, we address this problem and propose to give more creative power in the hands of sound designers even when the scene is dynamic or based on procedural synthesizers. This solution is based on the COLLADA schema in which we add the sound support, scripting capabilities and external extensions. The use of the augmented COLLADA language is illustrated through the creation of dynamic urban soundscape.

Keywords: COLLADA, scene description language, audio scene graph, soundscape, virtual cities