Integrating Sounds and Motions in Virtual Environments

JAMES K. HAHN, HESHAM FOUAD, LARRY GRITZ, and JONG WON LEE

Department of Electrical Engineering and Computer Science
The George Washington University
Washington, DC, 20052
hahn@seas.gwu.edu

Abstract

Sounds are often the result of motions of virtual objects in a virtual environment. Therefore, sounds and the motions that caused them should be treated in an integrated way. When sounds and motions do not have the proper correspondence, the resultant confusion can lessen the effects of each. In this paper, we present an integrated system for modeling, synchronizing, and rendering sounds for virtual environments. The key idea of the system is the use of a functional representation of sounds, called *timbre trees*. This representation is used to model sounds that are parameterizable. These parameters can then be mapped to the parameters associated with the motions of objects in the environment. This mapping allows the correspondence of motions and sounds in the environment. Representing arbitrary sounds using timbre trees is a difficult process that we do not address in this paper. We describe approaches for creating some timbre trees including the use of genetic algorithms. Rendering the sounds in an aural environment is achieved by attaching special environmental nodes that represent the attenuation and delay as well as the listener effects to the timbre trees. These trees are then evaluated to generate the sounds. The system that we describe runs parallel in real-time on an eight processor SGI Onyx. We see the main contribution of the present system as a conceptual framework on which to consider the sound and motion in an integrated virtual environment.

1. Introduction

Sounds and motions of objects in a Virtual Environment (VE) are inherently linked to each other. Sounds are, in general, caused by the motions of objects in the environment. The characteristics of sounds are shaped by the characteristics of the objects and their motions.

They reinforce each other to give a coherent perceptual experience. In VE, the primary focus so far has been in spatial localization of sounds using Finite Impulse Response (FIR) filters, usually Head Related Transfer Functions (HRTF) (Wenzel, 1992; Pope and Fehlen, 1993). The link between the motions of objects and the resultant sounds have been minimal.

The problem of generating effective sounds in VE can be divided into three sub-problems: sound modeling, sound synchronization, and sound rendering (Hahn et al., 1993). Sound modeling is an issue that has long been studied in the field of computer music (Mathews, 1969; Vercoe, 1986; Moore, 1990; Dannenberg et al., 1991; Scaletti, 1991). However, the primary consideration in VE is the effective parameterization of sound models so that the parameters being generated from the motion can be mapped to them. Effective sounds can mean realism or effective encoding of information, an area of interest to data sonification. Parameterization and synchronization of sound has been investigated in relation to user interfaces (Scaletti, 1991; Gaver, 1993), data sonification (Scaletti, 1991), and computer animation (Thalmann and Thalmann, 1990; Takala and Hahn, 1992; Hahn et al., 1995; Mishra and Hahn, 1995). Sound rendering refers to the process of generating sound signals from their models within a particular environment, very much like the process of generating images from their geometric models. Problems associated with sound rendering have been studied in the field of acoustics and signal processing. The primary consideration is rendering sounds to keep up with the desirable sampling rate (analogous to the image rendering problems in VE).

In this paper, we present a system for modeling, synchronizing, and rendering sounds for VE (Figure 1). The key idea of the system is the use of a functional representation of sounds, called *timbre trees* (Takala et al., 1993; Hahn et al., 1995). This representation is used to model sounds that are parameterizable. These parameters can then be mapped to the parameters being generated by the motions of objects in the environment. This mapping allows the correspondence of motions and sounds in the environment. Creating the timbre trees for arbitrary sounds is a difficult process and beyond the scope of this paper. We present some heuristics including the use of genetic algorithm to make the process easier. Rendering the aural environment is achieved by attaching special environmental nodes that represent the attenuation and delay (Takala and Hahn, 1992) as well as the listener effects to the timbre trees then evaluating the resultant trees. The system that we describe runs parallel in real-time on an eight processor SGI Onyx. However, we see the contribution of the present system as a conceptual framework on which to consider the sound and motion in an integrated virtual environment. When there was a choice between real-time performance and generality, we went with generality of the solution.

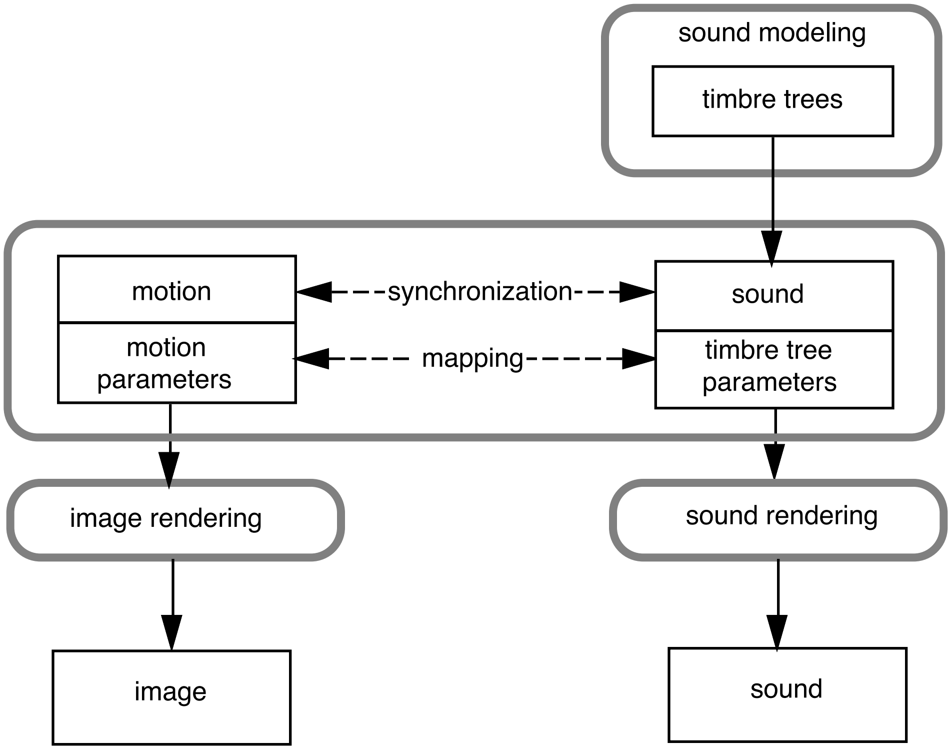


Figure 1. Integrating sound and motion

In Section 2, we describe a method to model parameterizable sounds using timbre trees. In Section 3, methods to map the parameters of timbre trees to motion parameters of virtual objects are described. In section 4, we briefly describe a way to render the timbre trees in a virtual environment. In Section 5, an implementation of a system for evaluating the timbre trees in a virtual environment is described.

2. Sound Modeling

Sound generation has been an active area of research in computer music. Many computer music systems generate sounds by connecting software modules which add, subtract, and multiply sound signals. While these systems are flexible they suffer two major drawbacks in their application to VE. First, the only events modeled by computer music systems are musical in nature and do not naturally correspond to events in a virtual environment. Second, the parameterization of the instruments generated using these systems is geared towards musical performance and therefore is difficult to use in modeling the material and environmental characteristics of an arbitrary VE.

Most sounds currently used in VE are sampled from real sounds or synthesized by MIDI devices. The problems with these approaches are that the sounds cannot be easily parameterized so that they may be correlated to motions. Although the use of real sounds have a place in VE, their use is limiting. Parameterizing real sounds by their attributes such as amplitude and pitch are difficult since they correspond to "reverse engineering" (i.e. we are trying to determine how the sounds were created from the sounds themselves). MIDI synthesizers only allow a limited range of sounds (those that are supported in that specific hardware). What we want are ways to represent sounds that reflect in some way the mechanism responsible for the sounds. Gaver (Gaver, 1993) describes a parameterization based on a heuristic analysis of those physical attributes of a sound that determine a listener's perception of the phenomena producing the sound. This approach was used to give feedback in computer-human interaction. In an immersive environment, this notion of feedback becomes more critical, as well as expansive.

We use timbre trees to represent general parameterizable sounds. The approach allows any sounds (including those involving sampled sounds) to be represented and parameterized. The representation is based on a functional composition description of the sounds. Such an approach has been used in the field of computer music (MUSIC V, Csound, cmusic, and, Fugue) (Mathews, 1969; Vercoe, 1986; Moore, 1990; Dannenberg et al., 1991). However, we are interested in representing general sounds that must be synchronized to motions.

2.1. Functional Composition Using Timbre Trees

Timbre trees are similar to *shade trees* (Cook, 1984) in image synthesis. The main idea behind shade trees is a functional composition that allows the flexible integration of various shading and texturing techniques. The advantage in using a tree structure is the modularity and simplicity of composing an endless variety of techniques. Operations to be performed (dot product, vector normalization, etc.) form the nodes of the tree. Each node operates on parameters that control shading appearances, and in turn produces other appearance parameters. When the entire tree is evaluated, the root returns a final color.

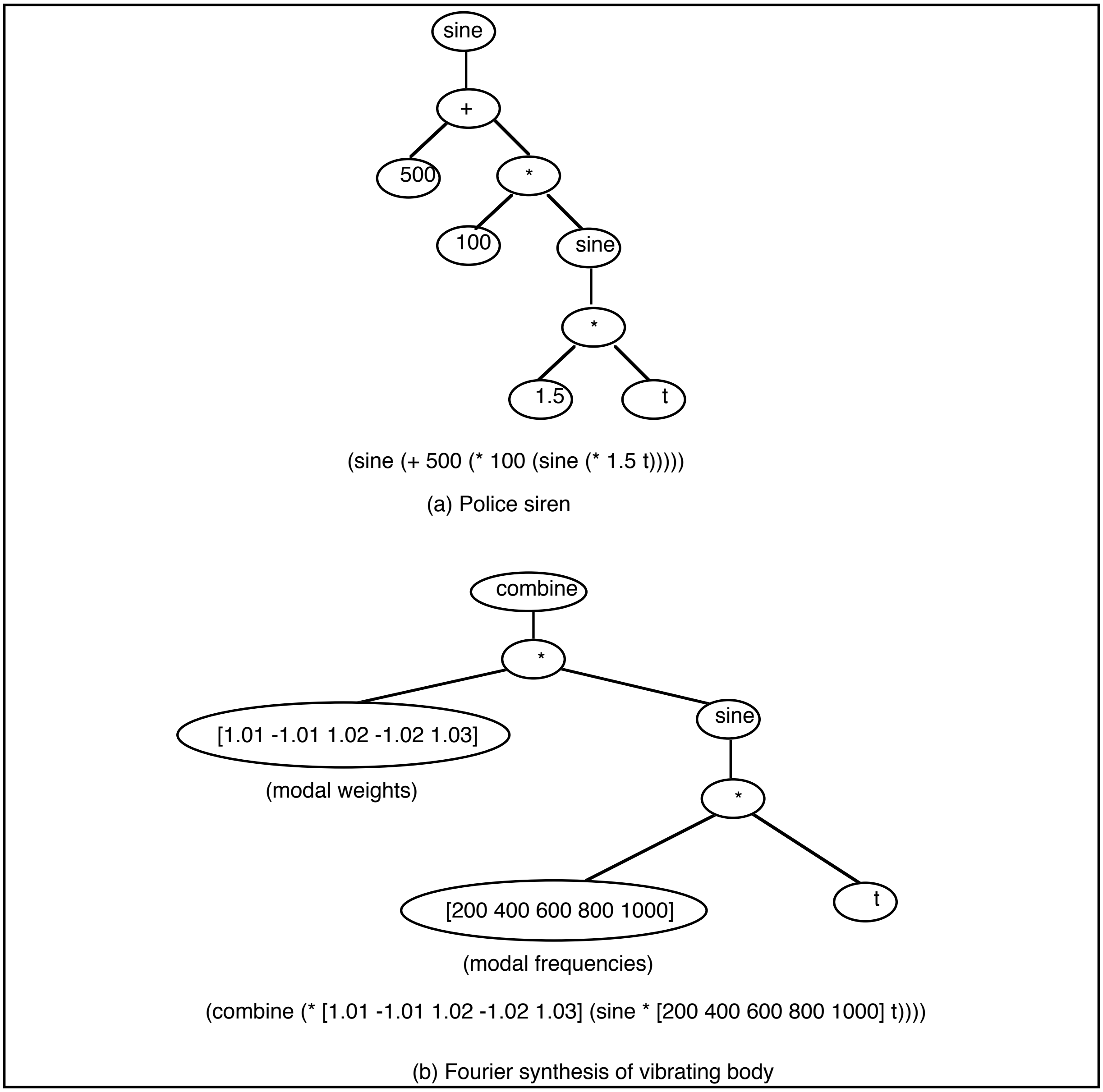
In timbre trees, nodes of the tree operate on other timbre trees, representations of sounds, or parameters. Evaluation of the tree produces sounds with a particular timbre. In object-oriented terminology, a timbre tree with a set of associated parameters can be seen as an abstraction of a class of sounds. The tree, evaluated with a specific set of parameter values, can be seen as a particular instantiation of the class. The user can generate new classes of sounds based on libraries of tree classes and elementary nodes. When parameters are time-varying, associated timbre trees produce sounds that also change with time.

2.2. Timbre Tree Structure

Conceptualizing procedural sounds as trees that represent functions, provides a rich representation for sounds, which can be easily manipulated by both the computer and the sound designer. Each node of the timbre tree may represent one of the following:

- A numerical constant
- A named parameter which may be passed from a motion control system (for example, "*t* (time)," "*fundamental-frequency*," "*angular-momentum*")
- A digitally sampled sound
- A mathematical function with zero or more arguments (In this case, each argument is given by a timbre tree)
- A vector of numerical constants
- Reference to other timbre trees perhaps from a library

Though this representation for timbre trees is easily manipulated by the computer and provides a good conceptual handle for users to think about these structures, we also needed a textual representation which could be used for input/output and for users to be able to read and write. Since the trees represent functions and are recursively defined, we chose a LISP-like language.



(a) Police siren

(b) Fourier synthesis of vibrating body

Figure 2. Timbre trees and corresponding expressions

Standard mathematical functions were implemented, such as +, -, *, /, exp, log, etc. We also implemented several special-purpose functions useful for sound synthesis, such as a number of elementary waveforms (sawtooth, triangle, square, sine, etc.), several types of noise (white noise, Perlin noise (Perlin, 1985), etc.), and some signal processing functions (filtering, convolution, etc.). We provided a mechanism for extending the language to add new functions of arbitrary complexity by coding these functions in C++. Figure 2a shows a simple timbre tree for a frequency modulated siren sound and the associated Lisp expression.

The language also supports vectors and vector operations. The use of vector operations can be quite useful in cases where a single node operates on an array of parameter values. For example, the final sound for a vibrating body can be approximated by a weighted sum of the vibration modes of the body in the frequency domain (Fourier synthesis). A *combine* node sums a vector argument corresponding to the frequencies and weights of the vibration modes and returns a single scalar value. This results in a simpler, more readable, and more manageable tree (Figure 2b).

Evaluation of timbre trees in the temporal domain is much like evaluation of shade trees in the spatial domain. At each sample point in the soundtrack, argument values are bound to timbre tree parameters and evaluation is performed via a postorder traversal of the tree. The output from the root of the tree is the computed value of the sound for that time sample point. For timbre trees that do not have any sampled sound nodes, the evaluation process does not involve re-sampling since the tree is essentially an analytic function. This avoids many of the aliasing artifacts also present in texture mapping.

2.3. Generation of Timbre Trees

The timbre tree structure gives a way to represent sounds but does not directly define a methodology for constructing sounds. Just as there is not just one approach to modeling all objects or textures in computer graphics, there is no one approach for modeling all sounds. The underlying physics could be used to create the tree (e.g. modes of vibrations for simple objects (Takala and Hahn, 1992)) or heuristics could be used to simplify complex phenomena (e.g. Fourier synthesis). Many of the trees that we generated were by trial and error. In general, sound modeling is a very difficult problem which usually relies heavily on the designer's experience and creativity. Modeling arbitrary sounds is beyond the scope of our work. However, we have taken advantage of the timbre tree representation of sounds by using genetic algorithms (GAs) to explore the vast space of possible sounds.

GAs are methods of optimization which have been found to be surprisingly adept at finding global optima in large and high-dimensional search spaces (Koza, 1992; Gritz and Hahn, 1995). Karl Sims used GAs to explore procedural textures, which were also represented by LISP-like expressions (Sims, 1991). From an initial guess, several mutated versions of the timbre tree representation are derived to form the first generation. The trees in the generation are then evaluated and the resulting sounds are played one at a time for the user who rates them for "fitness." The inhabitants of the next generation are derived by choosing one, two, or more parents from the previous generation. The parents are chosen randomly, with probabilities given by the user ratings. The parents are then combined to form a generation of children, which also experience mutations. This cycle continues until the user decides that a tree has been generated which represents the desired sound.

Using this process, we have been able to produce entire classes of sounds (e.g. bee-like sounds like mosquitos, chain saws, etc.). These classes are clearly derivatives of the original sound but which defy description and would have been difficult to conceive without the aid of GAs.

3. Correspondence to Motion

We refer to the system that controls and updates the positions of objects and users in the VE as the *motion control* system, very much like that in computer animation. The motion may be caused by the user (gestures, locomotion, etc.), by simulation (physically-based modeling (Hahn, 1988)), by behavioral modeling, or by kinematic techniques like key-framing (Watt and Watt, 1992). Some *sound events* are automatic based on physics. For example, when objects come in contact with each other (collision, rubbing, etc.) sounds should be generated. In the case of physically-based motion control, the parameters that are needed to characterize the resultant sounds, like impulse due to collision, come directly from the simulation. In other cases where only simple kinematic information like trajectory is present, needed information like velocity and acceleration can be calculated. Some sound events do not have a direct physical correspondence. For example, we may want to indicate distances between objects with sounds. In behavioral modeling, we may want to indicate "emotions" of objects with sounds. Oftentimes, a creative mapping that does not have any physical basis can be very effective in reinforcing certain correspondences. Sounds tend to affect the listener in a more subconscious and impressionistic way than visual cues allowing this freedom. Such abstract mappings can also be used for sonification (Scaletti, 1991) to express abstract data.

Timbre trees are usually associated with virtual objects in the scene. As the objects move in the environment, the associated timbre trees move with them. When sound events occur, the corresponding timbre trees are instantiated. Timbre trees are instantiated at a particular point in time and space by mapping parameters from the motion control system to the parameters associated with the timbre trees. This mapping depends to a great extent on how the particular timbre tree was constructed since this determines what parameters are available. There are some generic sound parameters like amplitude or delay that can be attached to any timbre tree.

3.1 Mapping Motion Parameters to Timbre Tree Parameters

For physically-based timbre trees, the mapping is obvious. For heuristically generated trees, the construction process often suggest a natural mapping. For example, in the case of the heuristic collision sound of an object striking a drum, the timbre tree is given by:

```
(* loudness
        (combine
                (*
                        (sine (* (rvector 200 20 10000) t))
                        (damp (* (rvector 200 20 10000) (* damping t))))))
```

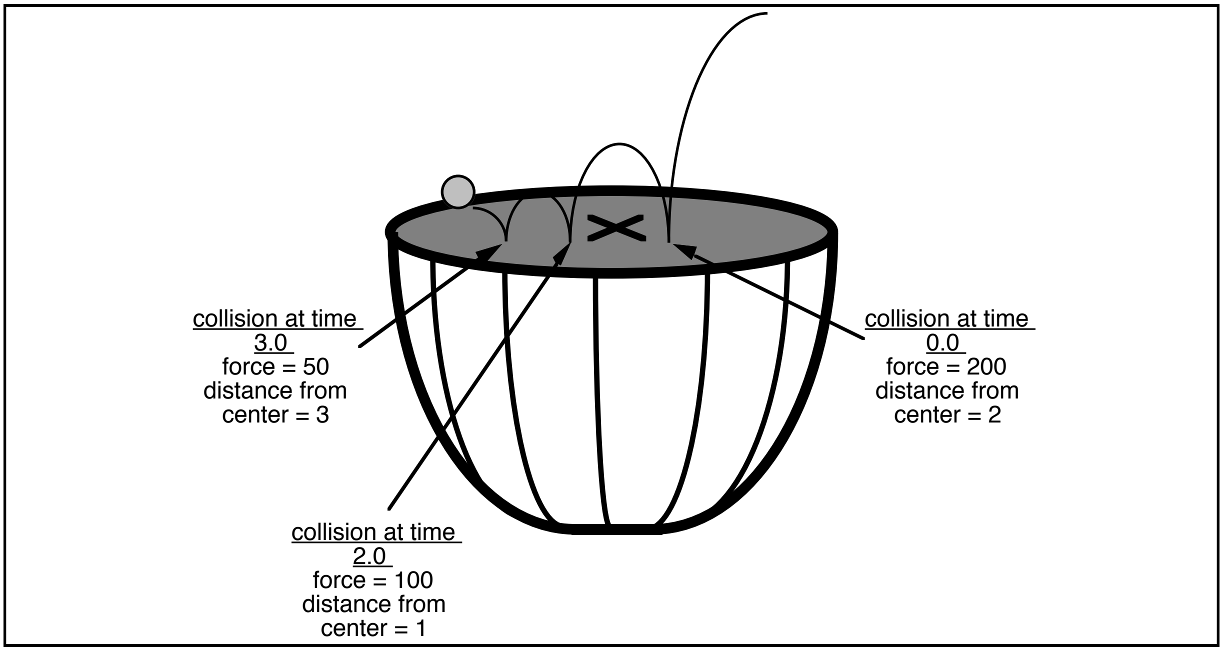


Figure 3. Object bouncing on a drum

This timbre tree uses the Fourier synthesis technique described in Figure 2b where amplitude value (loudness) and damping rate (damping) are given as parameters. The values for these parameters can be supplied by a physically based motion control system where objects are considered as rigid bodies. In this case, loudness can be mapped to the impact force with which the object strikes the drum's surface and the damping can be mapped to the distance from the center of the drum to the point of impact (Figure 3). The timbre tree is instantiated every time the object strikes the drum. These mappings give the correct appearance of the correspondence between sound and motion without considering the complexities of drum surface vibrations.

3.2. Mapping User Gestures to Timbre Tree Parameters

Much of the motions in VE are due to the interactions of the user with the objects in the environment. Of these interactions, some of the sounds are generated as a result of the surface textures of objects. Figure 4 shows a procedural texture of wood. As one scrapes this surface with an object (e.g. finger nail or pen), sounds should be generated whose signal characteristic corresponds to the microscopic grain texture. Figure 5 shows the signal generated from intersecting the two dimensional wood texture with the trajectory of the scraping object given as a diagonal line in Figure 4.

Figure 4. Procedural texture of wood and trajectory of scraping

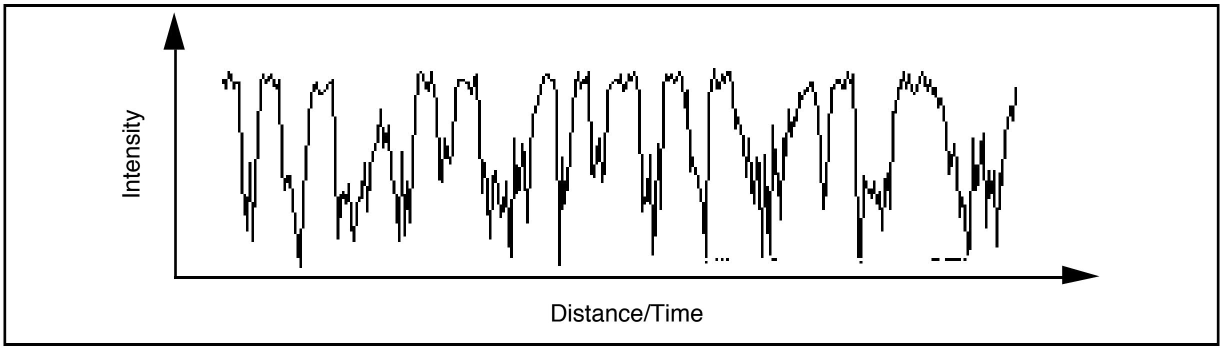


Figure 5. Intersection of texture and trajectory

The signal is filtered by the geometry and material characteristics of the object that is used to scrape the surface. Using an analogy of the phonograph needle, smaller, harder objects are able to pick up smaller microscopic surface features. Larger, softer objects act as low-pass filters. The resultant signal is globally scaled (in the time axis) by the speed with which the surface is scraped. The overall amplitude is scaled by the normal force with which the surface is scraped to arrive at the final sound. Figure 6 shows the timbre tree that represents this process. The sounds generated should be a function of the texture, the scraping object, the speed, and the force with which the surface is scraped.

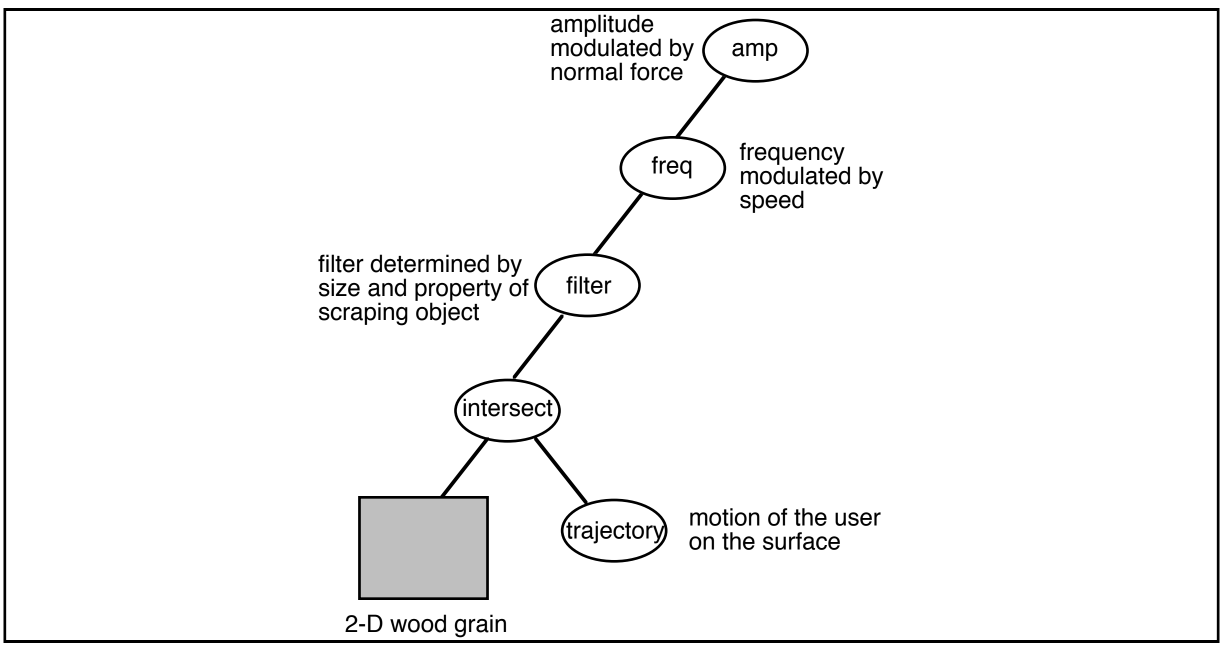
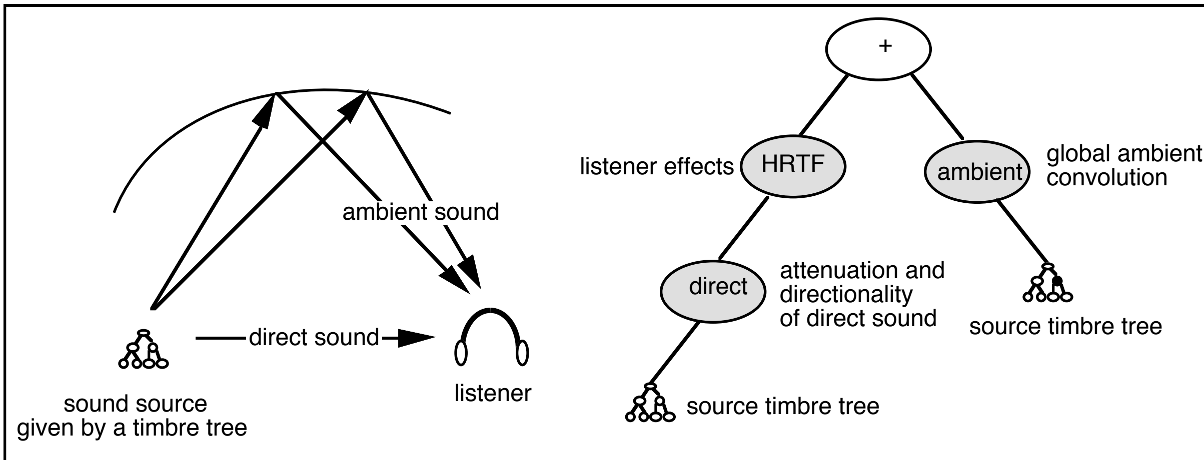


Figure 6. Timbre tree corresponding to scraping a wood texture

4. Rendering

Rendering sounds shares many similarities to rendering images (Takala and Hahn, 1992). First, the sound energy being emitted needs to be traced within the environment. The sound reaching the listener then needs to be processed to take into account the listener effects such as interaural delay, shoulder reflection, head shadowing, and pinna response. These effects can be expressed as convolution filters known as Head Related Transfer Functions (HRTF) (Wenzel, 1992; Pope and Fehlen, 1993). These processes can be seen as additional *environment nodes* that can be attached to the original timbre trees. The resultant trees are then evaluated at sampled frequencies to generate the final sounds. This corresponds to rasterization that occurs in image rendering. The whole process of rendering sounds can be seen as a rendering pipeline analogous to image rendering pipeline. Just as in image rendering, we consider each stage of the pipeline as a transformation attached to the original timbre tree. The advantage is that we can concatenate all of the rendering transformations and that one transformation can be applied to the original timbre tree. In practice, since real-time digital signal processors (DSP) for spatial placement and listener effects exist (e.g. Crystal River Engineering's Acoustitron), some parts of the pipeline can be processed separately.



(a) Environmental effects

(b) Final timbre tree with environment nodes

Figure 7. Sum of direct and ambient terms

Tracing sound energy within an acoustic environment can be quite complex comparable to tracing light energy within a visual environment to render images (taking into consideration the differences in wavelength and speed). We have used forward ray tracing of major paths of sounds for computer animations (Takala and Hahn, 1992). However, due to the computational overhead of a VE (including real-time evaluation of the timbre trees), we have lumped all global effects into one ambient term in our initial implementation. This is similar to the use of ambient term in image rendering. The ambient term is calculated by associating each partitioned spaces in the virtual environment (Funkhouser and Sequin, 1993) with certain acoustical properties. During a walk through of a building, for example, one would expect each room entered to exhibit its own unique acoustical personality. The rendering of a sound is performed within the context of the space the sound is to be heard. The acoustic characteristics are simulated currently by simply causing reverberation that is a function of the space geometry and the reflectivity. Using heuristics, and simplified acoustic models of sound passing in enclosed volumes, basic environmental effects due to room size, and material composition have been provided. The final sound heard is a sum of the ambient term as well as a term that corresponds to the direct path of the sound from the source to the listener. Figure 7 shows a timbre tree with all the environmental nodes. We do not currently consider listener effects although we are planning to use real-time DSP to handle the functionality of that node.

5. Evaluating Timbre Trees in a Virtual Environment

A system for evaluating timbre trees in a virtual environment has been integrated into a real-time audio system for VE applications called the Virtual Audio Server (VAS) (Fouad and Hahn, 1996) (Figure 8). The VAS system was developed as a testbed for studying real-time audio in virtual environments. In that regard the generality of the system was of the utmost importance in its conception. The system supports multiple clients connected to the sound server. The digital audio output of the system can be directed to any audio output device for playback, typically some form of real-time spacialization device such as the Acoustitron.

5.1. Timbre Trees In VE

The primary requirement of any interactive use of timbre trees is that they be evaluated in real-time. The evaluation of a timbre tree to produce a single sample is a relatively expensive process since it usually involves a number of floating point operations. Producing a digital sound signal using timbre trees requires evaluation of the tree at the sampling rate of the sound signal which is minimally 22kHz if extreme aliasing is to be avoided. This places a large computational burden on the machine.

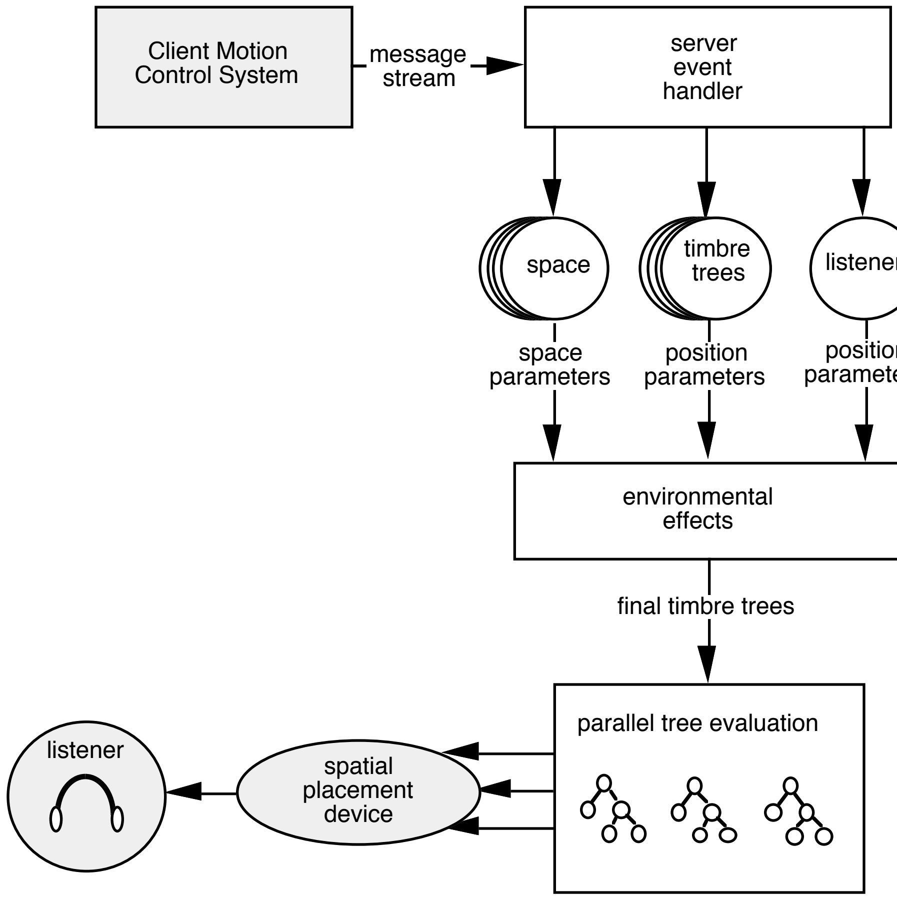


Figure 8: VAS system architecture

Compounding the problem of the computational requirements of a timbre tree is the requirement of multiple, concurrently executing trees. It is not unreasonable to expect that many sounds may be active in a virtual environment at the same time.

The use of timbre trees in computer animation gives us the convenience of a priori knowledge of simulation results due to the deterministic nature of scripted animation. This allows the parameterization of the tree based on keyframed parameters. While some of the motion within a VE may be keyframed, the non-deterministic nature of the user's realtime interaction with the environment requires a more dynamic approach: the running simulation must communicate parameters to the tree instantaneously as events occur.

5.2. System Architecture

The VAS system was developed in an object-oriented paradigm. The class *timbre tree* forms a class hierarchy from which the nodes comprising a timbre tree are derived. *Listener* has associated with it the position, orientation, as well as a number of other listener attributes.

The idea of partitioning a VE into distinct acoustical spaces is modeled by the class *space*. *Space* objects allow the motion control system to define spaces in the VE and associate distinct acoustical properties to each of those spaces. The definition of these properties includes a timbre tree object representing the background ambient sound and the environmental effects that the space has on sounds occurring in that space.

The VAS system was constructed as a client/server architecture where the clients are VE motion control systems. This approach was chosen for its generality as well as for load balancing. The computational burden of both the VE simulation and the sound generation would be too heavy for one machine. A client/server architecture allows the computational burden to be distributed across machines according to the suitability of the machine's architecture to the task.

Clients communicate with the *server event handler* of VAS through a message passing protocol using TCP/IP. There are, in general, three categories of events that are communicated to the server: object or listener movement within the environment; sound activation and deactivation events; and finally events indicating a parameter value change. As discussed in relation to timbre trees, the mechanism of parameterized sound allows for dynamic control over sound based on some user defined mappings. The VAS system provides a general mechanism for the parameterization of any representation of sound. The root class *timbre tree* provides the mechanism for maintaining and dynamically updating parameter/value pairs. The interpretation of these parameters depends on the derived classes of *timbre trees*. These resultant trees may then have environmental nodes attached, with listener attributes and space attributes as parameters.

The instantiated timbre trees are then evaluated in parallel. The result of this evaluation is a digital sample stream which can then be passed to hardware devices for spatial placement (e.g. Acoustitron).

5.3. Parallel Evaluation of Timbre Trees

In order to achieve real-time performance with the capability of handling multiple concurrent timbre trees, parallelism was employed. The parallelism was achieved using the light weight process or thread mechanism provided by SGI's IRIX operating system (Silicon Graphics Inc.). The system is implemented on an SGI Onyx with eight processors. Each active sound is assigned one or more threads which are responsible for evaluating the timbre tree in real-time. Multiple threads are required when the complexity of the timbre tree precludes its evaluation by a single thread in real-time. In those cases a dynamic work allocation scheme is used to coordinate the effort among the cooperating threads. The sample stream is divided into blocks. The next unallocated block is posted in shared memory, and the first idle thread claims the next free block of samples and updates shared memory with the next free block.

Timbre trees are evaluated by each thread in blocks corresponding to 10 ms at the output sampling rate of the system. The size of the work blocks determines the granularity of the parallelism. The lower bound of the block size is due to the overhead of synchronization and communication. Through experimentation with the block size, we have found that at block sizes below 100 ms, the system's performance begins to degrade considerably. The upper bound of the block size is due to the minimum perceivable latency in the sound signal produced. The system cannot send a block to the output device until it has been completed. It is therefore necessary that the block size fall below the minimum perceivable latency. A sample block equivalent to 100 ms produces an acceptable amount of delay.

6. Conclusion

Sounds are an important and integral part of VE. However, the research so far has concentrated primarily on localization of sounds in the environment. What has been missing is the correspondence between motions and the resultant sounds. We have outlined a system for integrating motion and sound in a virtual environment. The key to this approach is to express sounds as parameterizable structures using timbre trees. This allows users to easily design new timbre trees based on elementary nodes or libraries of timbre trees. When the timbre tree parameters are mapped to motion parameters, the desired correspondence is realized.

The biggest problem with the use of timbre trees is the computational load. Optimizations to the code, such as table lookup techniques, have enabled us to evaluate moderately complex trees in real-time. While these results are encouraging, we are still only capable of evaluating few trees concurrently. We are also investigating "levels of detail" for sound based on cost/benefit criteria equivalent to those in image rendering (Funkhouser and Sequin, 1993). We feel that the ultimate logical solution to the problem is the sound equivalent of the hardware image rendering pipeline. The system that we have presented in this paper suggests an important step in developing such a system.

7. Acknowledgements

We would like to thank the members of the George Washington University Computer Graphics Laboratory and the Laboratory for Advanced Computer Applications in Medicine. Special thanks go to Joe Geigel and Tapio Takala of the Helsinki University of Technology. Partial support for this work was provided by the Naval Research Laboratory (N00014-94-K-2009) and the Electronics and Telecommunications Research Institute (ETRI).

8. References

Blattner, M., Smikawa, D., and Greenburg, R. (1989). Earcons and Icons: Their Structure and Common Design Principles. *Human-Computer Interaction*, Vol. 4, No. 1, pp 11-44.

Borish, J. (1985). An Auditorium Simulator for Domestic Use. *J. Audio Eng. Soc*, Vol 33, No. 5, pp.330-340.

Chamberlin, H. (1985). *Musical Applications of Microprocessors.* Hayden Book Co.

Cook, R. (1984). Shade Trees. *Proc. SIGGRAPH'84*, Vol.18, No.3, pp.195-206.

Dannenberg, R., Fraley, C., and Velikonj, P. (1991). Fugue: A Functioal Language for Sound Synthesis. *IEEE Computer*, Vol.24, No.7, pp.36-42

Fletcher, N. and Rossing, T. (1991). *The Physics of Musical Instruments*. Springer-Verlag.

Fouad, H. and Hahn, J. (1996). A Framework for Integrating Audio in Virtual Environments. *Proc. IS&T/SPIE Symposium on Electronic Imaging: Science & Technology*.

Funkhouser, T. and Sequin, C. (1993). Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments. *Proc. SIGGRAPH'93*, pp. 247-254.

Gaver, W. (1993). Synthesizing Auditory Icons. *Proceedings of INTERCHI*.

Gritz, L. and Hahn, J. (1995). Genetic Programming for Articulated Figure Motion. *Journal of Visualization and Computer Animation*, Volume 6, Number 3, pp. 129-142.

Hahn, J. (1988). Realistic Animation of Rigid Bodies. *Proc. SIGGRAPH'88*, ACM Computer Graphics, Vol.22, No.3, pp.299-308

Hahn, J., Gritz, L, Darken, R., Geigel, J., and Lee, J. (1993). An Integrated Virtual Environment System. *Presence: Teleoperators and Virtual Environments*, Vol. 2, No. 4, pp 353-360.

Hahn, J., Geigel, J., Lee, J., Mishra, S., Gritz, L., and Takala, T. (1995). An Integrated Approach to Motion and Sound. *Journal of Visualization and Computer Animation*, Volume 6, Issue No. 2, pp. 109-123.

Mishra, S. and Hahn, J. (1995). Mapping Motion to Sound and Music in Computer Animation and VE. Invited Paper, *Proceedings of Pacific Graphics*.

Koza, J. (1992). *Genetic Programming*. MIT Press.

Mathews, M. (1969). *The Technology of Computer Music*. MIT Press.

Moore, F. (1990). *Element of Computer Music*. Prentice Hall, Englewood Cliffs, NJ.

Nakamura, J, Kaku, T., Noma, T., and Yoshida, S. (1993). Automatic Background Music Generation Based on Actors' Emotion and Motions. *Proceedings of the First Pacific Conference on Computer Graphics and Applications*, Vol. 1, pp.147-161.

Perlin, K. (1985). An Image Synthesizer. *Proc. SIGGRAPH'85*, Vol.19, No.3, pp.287-296.

Pope, S. and Fehlen, L. (1993). The Use of 3-D Audio in a Synthetic Environment: An Aural Renderer for a Distributed Virtual Reality System. *Proc. IEEE VRAIS '93*, pp. 176-182.

Scaletti, C. (1991). The Kyma/Platypus Computer Music Workstation in S. Pope (Ed.). *The Well-Tempered Object: Musical Applications of Object Oriented Software Technology*. MIT Press.

Silicon Graphics Inc. Parallel Programming on Silicon Graphics Computer Systems. Version 1.0, Documentation number 007-0770-010.

Sims, K. (1991). Artificial Evolution for Computer Graphics. *Proc. SIGGRAPH'91*, Vol.25, No.3, pp. 319-328.

Takala, T. and Hahn, J. (1992). Sound Rendering. *Proc. SIGGRAPH'92*, Vol.26, No.2, pp.211-220.

Takala, T., Hahn, J., Gritz, L., Geigel, J., and Lee, J. (1993). Using Physically-Based Models and Genetic Algorithms for Functional Composition of Sound Signals, Synchronized to Animated Motion. *International Computer Music Conference (ICMC)*.

Thalman, N. and Thalman, D. (1990). *Synthetic Actors in Computer Generated 3D Films*. Springer-Verlag.

Vercoe, B. (1986). *Csound: A manual for the Audio Processing System and Supporting Programs*. MIT Media Lab.

Watt A. and Watt, M. (1992). *Advanced Animation and Rendeting Techniques*, Addison-Wesley.

Wenzel, E. (1992). Localization in Virtual Acoustic Displays. *Presence: Teleoperators and Virtual Environments*, 1, 80-107.

Zaza, T. (1991). *Audio design: Sound Recording Techniques for Film and Video*. Prentice-Hall.