Journal of Visualization and Computer Animation, Volume 6, Issue No. 2, pp. 109-123

An Integrated Approach to Motion and Sound

JAMES K. HAHN, JOE GEIGEL[†], JONG WON LEE, LARRY GRITZ,

TAPIO TAKALA[*], AND SUNEIL MISHRA

Department of Electrical Engineering and Computer Science

The George Washington University

Washington, DC, 20052, U.S.A.

hahn|geigel|won|gritz|suneil@seas.gwu.edu

[*]*Department of Computer Science*

Helsinki University of Technology

02150 Espoo, Finland

tta@cs.hut.fi

[†]*Pittsburgh Supercomputing Center*

Pittsburgh, PA 15213, U.S.A.

Journal of Visualization and Computer Animation, Volume 6, Issue No. 2, pp. 109-123

SUMMARY

Until recently, sound has been given little attention in computer graphics and related domains of computer animation and virtual environments, although sounds which are properly synchronized to motion provide a great deal of information about events in the environment. Sounds are often not properly synchronized because the sounds and the phenomena that caused the sounds are not considered in an integrated way. In this paper, we present an integrated approach to motion and sound as it applies to computer animation and virtual environments. The key to this approach is synchronization by mapping the motion parameters to sound parameters so that the sound changes as a result of changes in the motion. This is done by representing sounds using a technique for functional composition analogous to the "shade trees" which we call *timbre trees*. These timbre trees are used as a part of a sound description language that is analogous to scene description languages like RenderMan. Using this methodology, we have produced convincing sound effects for a wide variety of animated scenes including the automatic generation of background music.

KEY WORDS Computer animation Motion Multimedia Sound Soundtrack Virtual environments

INTRODUCTION

Sounds are an integral part of the environment. They are caused by motions in the world and in turn cause changes to the world. Characteristics of sounds produced are directly linked with the phenomenon that caused the sounds. Sounds are also shaped by the environment in which they propagate. Therefore energies that represent the visible and audible spectrum that permeate the world are very much correlated. In computer graphics (image rendering, computer animation, virtual environments, etc.) the concentration has been in rendering the visible spectrum. When sounds have been added, the correlation between the two has been generally weak. Sounds are usually generated independent from the events that actually caused them. The result is that what we see and what we hear are from two different "worlds." The resultant confusion detracts from the total experience.

A large body of work exists in various domains that relate to sound. The issues in sound generation have long been studied in the field of computer music[1-5]. Parameterization and synchronization of sound has been investigated in relation to user interfaces[6,7], data sonification[5] and computer animation[8,9]. What is lacking, however, is a single framework that integrates arbitrary sounds and motions for computer animation and virtual environments. The only previous related work in the graphics literature[10] tried to address this problem. However the important issue that the approach did not address was a general method to parameterize sounds which is essential for establishing the mapping between motion and sound.

In this paper, we propose such a framework and describe a methodology to view sounds and motions in an integrated way (Figure 1). The fundamental approach of this paper is to describe sounds as parameterizable structures. These structures, called *timbre trees*[11], allow sounds to be represented so that the sound parameters correspond in some way to the phenomena that are responsible for creating the sounds. For example, these mappings may be physically-based for object-to-object interactions like scraping or collisions. These mappings may also be purely imaginary such as when the motions are used to generate background music. These parameters allow a timbre tree to span a class of sounds. When a timbre tree is instantiated, the parameters of the trees are bound to the parameters from the motion control system. As the animation proceeds the changing parameters coming from the motion cause changes in the associated sounds. In this way the motions and the sounds that they produce are intimately linked. Environmental effects are simulated by attaching additional nodes to the timbre tree. The final soundtrack is rendered by evaluating the resultant tree.

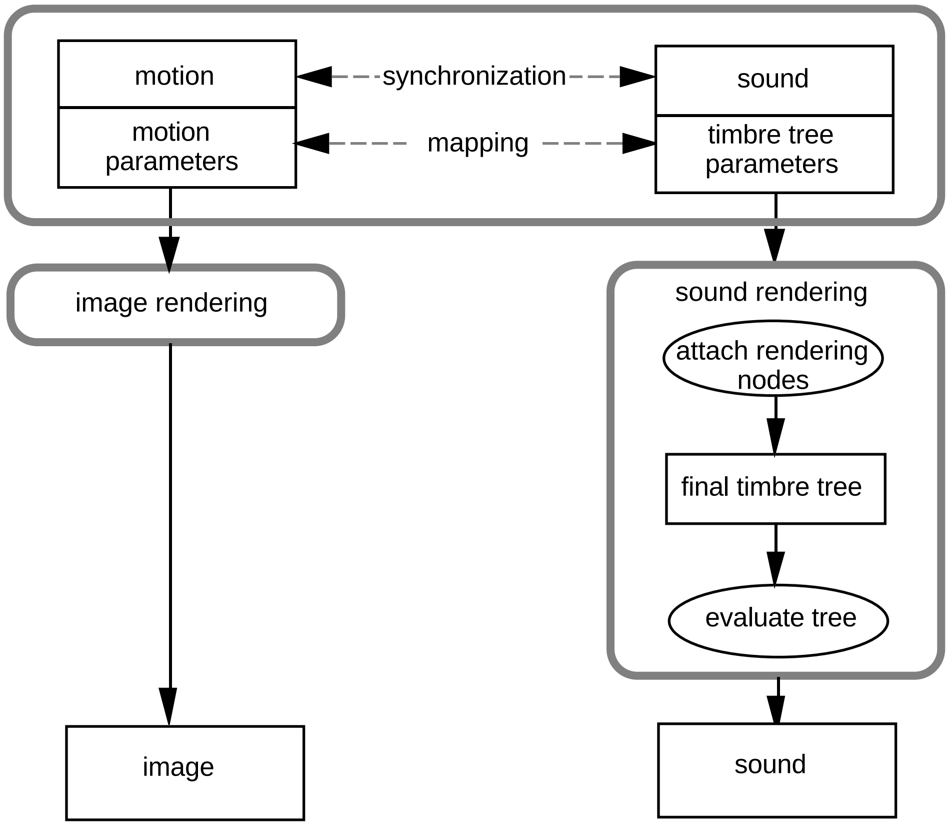


Figure 1. Integrating sound and motion

SOUND GENERATION

The most important part of sound representation and synthesis is the ability to parameterize sounds so that we are able to link these parameters to the motions that are responsible for producing sounds. Sounds can be parameterized by their attributes such as amplitude, pitch, etc. We can use such parameterization for sampled sounds. However, it becomes difficult to map these parameters to motions because it is equivalent to "reverse engineering" (i.e. determining how the motions produced the sounds from the sounds themselves). We can also generate sounds based on an idea of the mechanism behind them. There has been work in synthesizing auditory icons that are parameterizable along the attributes of the events responsible for the generation of sounds in user interfaces[7]. Although the application is directed toward giving the users specific information associated with specific interface events, the approach parallels our work. However, it lacks a general methodology to represent and map sounds to arbitrary motions.

Functional composition of sounds has been explored in a number of computer music systems including MUSIC V, Csound, cmusic, and, Fugue[1-4]. There has also been a number of approaches used for generating sounds such as Fourier synthesis, signal multiplication, filtering (subtractive synthesis), and frequency modulation. These approaches in computer music, although related to our approach, do not address the issue of representing general sounds so that their parameters correspond to the phenomena that caused them. This is essential in order to synchronize the sounds to the motion.

In this paper, we describe a flexible, general, and powerful way to use functional composition to parameterize sounds based on motion using a representation called *timbre trees*.

Representation using Timbre Trees

Timbre trees are analogous to shade trees[12] 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. Timbre trees operate in a similar fashion. Nodes of the tree operate on other timbre trees, representations of sounds (including sampled sounds), or on external parameters. Standard mathematical functions as well as several special-purpose functions useful for sound synthesis, such as a number of elementary waveforms (sawtooth, triangle, square, sine), several types of noise (white noise, Perlin noise[13], etc.), and some signal processing functions (filtering, convolution) have been implemented.

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. Thus, the user can easily generate new classes of sounds based on heuristics and/or libraries of trees and elementary nodes. By evaluating a tree with a time dependent set of parameters, the characteristics of the sound can be changed "on the fly."

Growing Timbre Trees

Timbre trees provide a convenient way for animators to create sounds. We illustrate a typical design process through an example for generating a timbre tree that represents the buzzing sound of behaviorally-controlled bees (Figure 2). We use a general idea of how bees actually make sound with their wings to construct a plausible procedural representation. A bee would tend to push its wings down slowly, then lift them up quickly, repeating this process at some frequency. Thus, our first guess is a sawtooth wave of a particular frequency. To avoid an instrument-like pure tone, the frequency is continuously deviated about its mean, so it is represented by a varying frequency given by a sum of a mean frequency and a high frequency noise. Even a noisy tone is dull for a bee, if the mean frequency is constant. A living creature's continuously varying activity can be simulated by defining the mean frequency as a nominal frequency plus a low frequency noise resulting from a behavioral, second-order Markov process. One particular tree represents an instance of a sound (e.g. one bee performing a particular activity), but the tree structure itself represents an entire class of sounds (e.g. different kind of bees, different insect sounds, or even chain saws).

The example illustrates an important goal of the sound generation and representation system. Since the timbre tree of the sound was derived by the animator from a general idea of how the sound was produced, the parameters associated with the tree can readily be mapped to parameters of the motion responsible for the sound. It is not the ability to generate sounds that is important but the ability to parameterize the sounds so that it can be mapped to the motion.

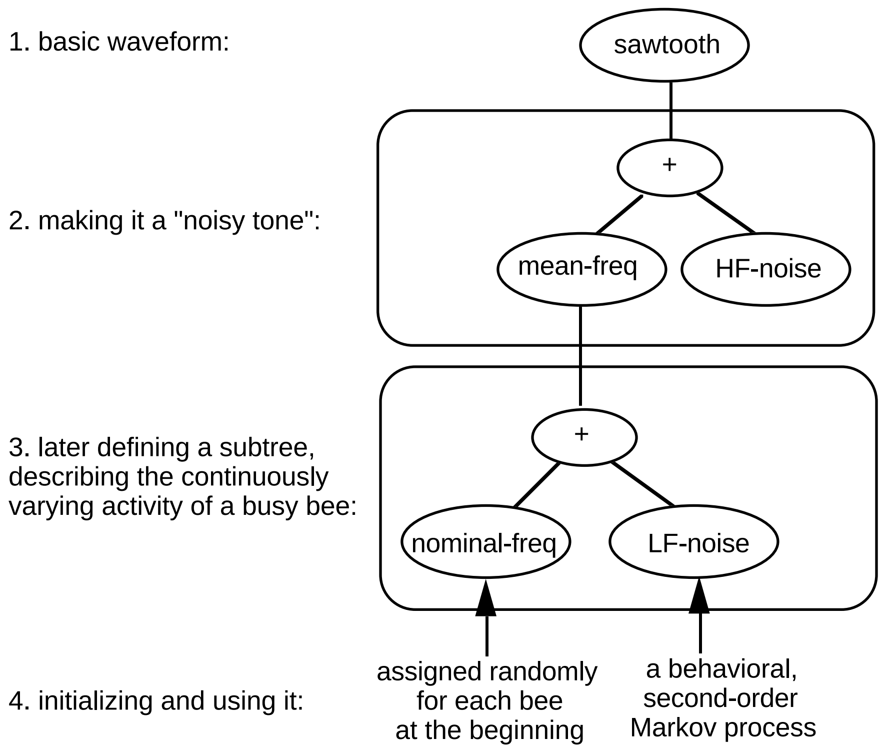

1. basic waveform:
sawtooth
2. making it a "noisy tone":
+
mean-freq
HF-noise
3. later defining a subtree, describing the continuously varying activity of a busy bee:
+
nominal-freq
LF-noise
4. initializing and using it:
assigned randomly for each bee at the beginning
a behavioral, second-order Markov process

Figure 2. Timbre tree design process for bee sounds

One simple but extremely general timbre tree is one for Fourier synthesis. In this case, a number of frequencies are added together to generate the desired sound. Such a synthesis was shown to be effective in cases where the modes of vibration are simple such as guitar strings[11]. More sophisticated modal analysis can be applied based on physical properties. The advantage of this approach is that we can easily map physical attributes to sound parameters. For example, there exists simple models for the modal analysis of the frequency components of hollow tubes[14]:

$$f_n = \frac{\pi K}{8L^2}\sqrt{\frac{E}{\rho}}[3.011^2, 5^2, 7^2, \ldots, (2n+1)^2]$$

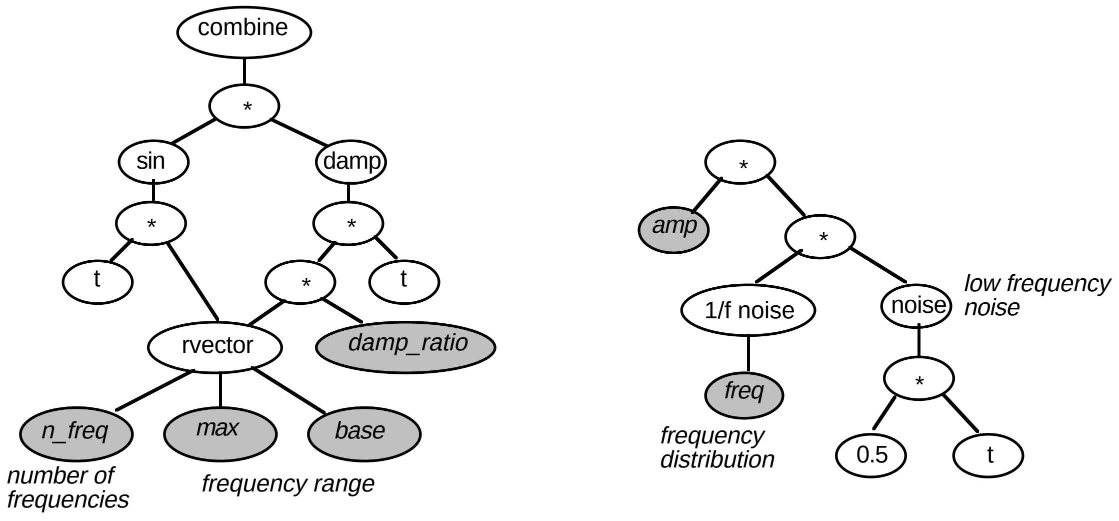
where L is the length, E the elasticity, ρ the density, and K is the radius of gyration which is the square of the tube's outer diameter over the inner diameter. We used such a formula to generate wind chime sounds (EXAMPLES).

For shading calculations in image rendering, an ad hoc heuristic such as Phong illumination may not be physically correct, but is fast and accurate enough for most purposes. Similarly for sound rendering, rigorous physical simulation is not practical for many sounds, nor is it pragmatic, since even an astute listener may not recognize the difference between a physically simulated sound and a heuristic procedural sound. For this reason, timbre trees for many common sounds could be developed by deriving a heuristic from a rough notion of how the sound is actually produced by a real physical system. These heuristics could be used to determine the parameterization for the sounds so that they can be mapped to the motion events.

For sounds generated by the interactions of physical objects, we have constructed a timbre tree for a class of physical objects and excitation modes (e.g. colliding, sliding, rolling)[15]. For collision sounds, we can use the Fourier synthesis technique. The modes of vibrations are sufficiently complex that they should be represented as a statistical distribution:

$$\text{signal}(t)=\sum_{i=1}^{n} e^{-c\omega_i t}\sin(\omega_i t)$$

where the ω_i correspond to a set of random frequencies and c a damping term. A timbre tree that corresponds to this is given in Figure 3a.



(a) Timbre Tree for collision sound

(b) Timbre Tree for wind sound

Figure 3. Timbre tree examples. The darkened nodes represent parameters

In this tree, the number of frequency components (*n_freq*), the range of the random frequencies (given by *base* and *max*) and a damping ratio (*damp_ratio*) are given as parameters. The specialized node rvector returns a static vector of random values within the given range. The decay rate of each frequency component is set by the node damp as being proportional to the frequency. This is based on the observation that energy dissipation within the material is proportional to the velocity. Thus high frequency vibrations that undergo high speed deformations die out sooner. A combine node sums a vector argument corresponding to the frequencies and weights of the vibration modes and returns a single scalar value. Each of these parameters map to events and physical attributes. Different combinations of parameter values result in a surprising variety of sounds within this class including bell sounds, wood-like sounds, and metallic sounds (like the striking of a cymbal).

Using a similar technique, we generated wind sounds by multiplying a 1/f noise signal with a low frequency noise function (Figure 3b). The amplitude parameter (*amp*) can map to wind force from an animation. The frequency distribution (*freq*) can be used to generate different types of wind sounds.

We found that deriving these heuristics was somewhat of a learned (and developable) skill, much as it is for writing procedural shaders in image rendering. Once a user had developed several of these sounds and understood how combinations of functions lead to particular families of sounds, it became easier to develop new sound families and instances.

Many times in the development of timbre trees (especially when using a heuristic approach), we would find that we had constructed a timbre tree which was obviously an incorrect instance of the class that we wanted. For example, at one point we had a perfect mosquito sound, but what we really wanted was a bee. Or we would have a timbre tree which would sound almost right, but would be incorrect in a way which eluded quantification. Or perhaps we would have the correct instance, but wanted to explore other sounds of that class. To help us in these situations, we developed tools that make use of evolutionary algorithms as a form of computer-aided search through these function classes.

Genetic programming (GP) is a method of optimization which utilizes evolutionary concepts such as reproduction, mutation, and natural selection to discover useful computer programs or formulas[16]. Sims has used a kind of GP to explore procedural textures, which were represented by LISP-like expressions[17]. Since timbre trees can also be represented as LISP-like programs, we can extend this technique to the sound domain. Using this technique, we have been able explore sounds within a class (like mosquitos, chain saws, etc.) by varying the parameter values within a particular timbre tree. We have also produced entirely different timbre trees with the same set of parameters that represent different classes of sounds.

SYNCHRONIZATION

The link between the image and sound domains is defined via synchronization. Synchronization involves not only timing (when sounds are to occur), but also the mapping of motion parameters to those of sounds (how sounds are to be shaped). Once a sound class has been defined and its parameters determined using a timbre tree, synchronization is achieved by instantiating individual sounds by binding the sound class parameters with appropriate parameters from a motion control system. For cases where a sound varies continuously with the motion, we introduce the notion of a time dependent variable. These variables represent signals which are continuously changing in time. Supplying a time dependent variable as an instantiation value to a sound class results in a sound which will also vary continuously, synchronized to the continuous change of the signal represented by the variable.

In using time dependent variables, the values that define the time varying signal are usually calculated by the motion control at each animation frame. Since the temporal sampling granularity of sound is much finer than that of images (8000Hz - 44100Hz as compared to 30Hz), interpolation is used to find values of the time dependent variable for times between animation frames.

Sonic Scene Description Language

Using a Sonic Scene Description Language, an animator or a motion control system can describe what is occuring in the sound domain. The description language is analogous to scene description languages such as RenderMan[18]. Once created, this description is fed to a renderer which produces the final soundtrack. Thus the synchronization process which is represented by the sonic scene description is independent of the rendering process.

We have developed such a language that allows for the definition of sound classes using timbre trees, the instantiation of individual sounds, the starting and stopping of these instantiated sounds, the definition of time dependent variables, and the specification of their keyframed values. Physical object data that may be useful in sound rendering (e.g. the positions, orientations, and attributes of microphones, sound sources, and other objects in 3-D space) can also be specified using the language. Like most visual scene description languages, our language includes both a textual and a functional interface. Thus the sonic scene can be described using a text file or a program written in C++. Using the functional interface, the sonic scene may be output directly from a motion control system.

For example, the sonic scene for a simple animation of two colliding cubes is described using the functional interface in Figure 4. Functions beginning with `Sr` are utility routines used to specify the sonic scene. Note that these routines can be incorporated directly into the application performing the motion control. This example is typical of how these routines might be used in a physically based animation system. The timbre tree presented in Figure 3a is used as the definition for the class of collision sounds. Although each of the cubes is of a different material (one is wood, the other metal), the same tree can be used for creating the sounds generated by either of the cubes. At each collision detected by the motion control, a collision sound is instantiated, started and thus introduced into the sonic scene. Different instances of the collision class are created by supplying the tree with different sets of instantiation parameters dependent upon the material of each cube. The force of the collision, returned by the motion control, guides the amplitude of the collision sounds. Note that, although not illustrated, time dependent variables may also be supplied as arguments during sound instantiation.

The sonic scene description, when interpreted, will produce a series of timbre trees, one for each sound instantiation. These trees will essentially be clones of the tree defining the class except that leaf nodes, representing timbre tree parameters, will be replaced with nodes representing the appropriate argument values. Time dependent variables are represented as *keyframe nodes* which perform the proper interpolation of key values. A specialized *timer node* is placed at the root of each instantiated tree and acts as a switch, indicating when the sound is to be turned on and off. These trees will later be combined and evaluated during the rendering phase.

Mapping motion parameters to music

We can extend the methodology to synchronize background musical soundtracks with motion-control parameters. This could mean generating motion from sounds or creating soundtracks from the motion, which is the approach that we explored. Our aim was to generate soundtracks that are both interesting, and intrinsically based on the motion parameter values. Such a technique has been previously explored[19], however, in this previous work, the very limited correlation between the motion and sound was specified manually.

We apply transformations to motion parameters to produce a musical score which can be considered another example of a sonic scene description language. Performance information is supplied within the score, but the actual musical output is dependent on the nature of the rendition of the score. Any motion control parameters, or combination of motion control parameters may be employed in generating a musical score. In general, the introduction of more complex mappings creates scores that tend to reflect the motions of objects less distinctly, but adds subtlety and character to the music produced. These resultant scores often appear far more "composed" in the traditional sense, than simply reflecting the motion in some automated way. The transformations used may be based on heuristics, or on simple musical or physical rules or dependencies. Similarly, any musical parameters or structures may be mapped to the motion parameters. Minimally, only musical pitches need be output in order to produce a score. In addition, some concept of instrumentation is necessary. The obvious methodology is to assign one part to each object for which parameter values are available.

```
/*
 * declare a collision sound class defined by the timbre tree
 * found in file figure3a.tt
 */
SrSoundClass collision = SrTimbreTree ("figure3a.tt");

/*
 * animation loop - t is a global clock used in guiding the
 * simulation...frame rate is 30 frames/sec
 */
for (float t = 0.0 ; t < simulation_length; t += 0.0333) {

   /*  motion of each cube is determined and collision(s)
       detected by motion control. Force of each collision is obtained */
   motion_and_collision_detection (...);

   for each collision {
      float force = obtain_collision_force (...);

      if (wooden_cube_collision) {
          /* instantiate a wood collision sound.  Wood sound is created by
             summing many frequencies that damp out quickly */
          SrSound wood_collision =
            SrInstantiateSound (collision,                  // sound class
                           "amp",       SR_CONSTANT, force,  // amp depends on force
                           "n_freq",    SR_CONSTANT, 500,    // n freq = 500
                           "max",       SR_CONSTANT, 10000,  // max_freq = 10000
                           "base",      SR_CONSTANT, 5000,   // base freq = 5000
                           "damp_ratio",SR_CONSTANT, 2.6,    // damp ratio = 2.6
                           NULL);                        // end of parameter list

          /* start up this instantiated sound at time corresponding to
             current frame */
          SrStartSound (wood_collision, t);
      }

      if (metal_cube_collision) {
          /* instantiate a metal collision sound.  Metal sound is created by
             summing a handful of frequencies that damp out slower */
           SrSound metal_collision =
             SrInstantiateSound (collision,                  // sound class
                            "amp",       SR_CONSTANT, force,  // amp depends on force
                            "n_freq",    SR_CONSTANT, 50,     // n_freq = 50
                            "max",       SR_CONSTANT, 1000,   // max_freq = 1000
                            "base",      SR_CONSTANT, 500,    // base freq = 500
                            "damp_ratio",SR_CONSTANT, 1.0,    // damp ratio = 1.0
                           NULL);                        // end of parameter list

          /* start up this instantiated sound at time corresponding to
             current frame */
          SrStartSound (metal_collision, t);
      }
   }  // end of for each collisiom
   ....
}  // end of animation loop
```

Figure 4. Sonic Scene Description for the collision of two cubes

We allowed our transformations to map motion parameters to any semitone in the Western-based diatonic scaling system. By using simple transformations, however, we could also force the music to obey musical rules such as conformation to a modal scale (e.g. the diatonic minor scale in C). By keeping these compositional constraints as transformations, the system is freed of musical bias at the lowest level, while still allowing rule-based composition if desired.

Updating musical pitches (which are mapped to time dependent variables) continuously in time is both unrealistic and unnecessary. It is more sensible to sample every quarter, or half second. By producing scores with equal length notes, associated constructs such as instrument tempos can be handled simply by slurring notes to produce longer durations of constant pitches.

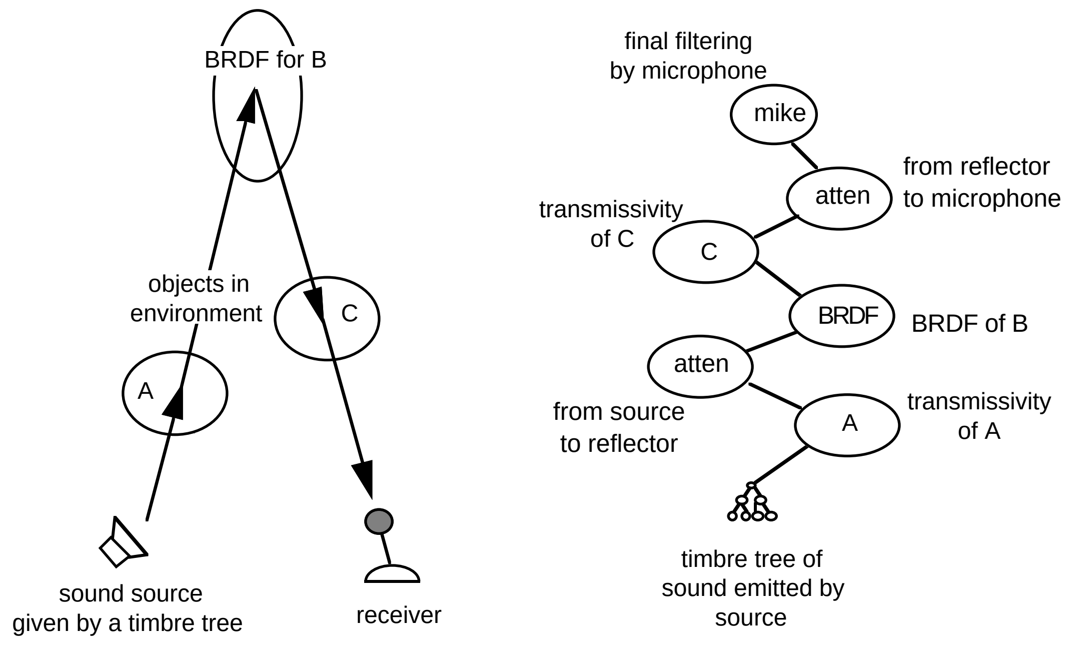
The musical scores produced from the transformations are immediately identifiable to musicians. This however is only one method of performing, or rendering, the soundtrack. Using MIDI devices, the output may be generated directly, without the need for an explicit musical score. We have used such techniques to produce background music for a number of animations (EXAMPLES).

RENDERING

The process of rendering sound in a spatial environment involves a series of modifications to the sound. These are due to transformations from object space to microphone space, auralization, and environmental effects. The transformation of sound from the object space to microphone space is result of attenuation and delay due to the distance between them[11]. This can be expressed as attenuation/delay nodes in the timbre tree.

A good deal of the work has been done in the area of auralization in virtual environments. Several methods have been successfully implemented ranging from the simulation of Head Related Transfer Functions (HRTFs) using Finite Impulse Response (FIR) filters based on empirical data[20] to heuristics based on simple psychoacoustical principals[21]. All of these methods share a common approach in that they filter a sound signal based on the position of a sound source in a virtual space. Thus the directional effects of a listening device, whether it be a set of stereo directional microphones or the ear represented by a set of HRTF filters, can simply be seen as yet another timbre tree node.

The method used to create the environmental effects is defined by an "environmental" node and could range from simple heuristics to a sophisticated treatment of acoustical theory. One approach is to trace the sound energy in the environment using representations of objects that are specifically designed for sound tracing since light waves and sound waves "see" different representations of objects. For example, reflection, refraction, and transmission of each object can be represented by 3-dimensional bidirectional reflectance distribution functions (BRDFs) which are functions of the shape and material characteristics of the object.



(a) Environmental effects (b) Timbre tree with specialized rendering nodes

Figure 5. Environmental nodes appended to the timbre tree

By attaching these nodes to the timbre trees that represent individual sound and combining the resultant trees, an entire scene to be rendered can be represented as a single timbre tree. This tree will represent the sound heard by a given sound receiver and, when evaluated and sampled, will result in the generation of the final soundtrack. 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, evaluation is performed by a post order traversal of the tree. The output from the root of the tree is the computed value of the sound for that time sample point. Figure 5 illustrates the use of specialized rendering nodes in describing a sonic scene to be rendered.

EXAMPLES

We have made a number of animations to illustrate our approach. The basic philosophy that was used in constructing timbre trees as well as the mapping between sound and motion was not to restrict ourselves to the mappings implied by strict physical simulations.

The first example illustrates a process of synchronization that automates the function of a "foley artist"[22]. In this case, we wanted to map the parameters of a physically-based motion of a coin rolling on the floor to a sampled sound. The parameters that we had for the sound were amplitude and pitch. We mapped the angular velocity to the amplitude based on the observation that the energy of the interaction of the coin with the floor is proportional to the angular velocity. We also mapped the surface normal to the amplitude. Physical analysis of the sound produced as the coin starts to oscillate and comes to rest is rather complicated. However, this mapping gives the synchronization that we expect. The final soundtrack was produced using a composition of the two mappings. Figure 6 shows the frames from the animation as well as the amplitude variation of the sound produced.

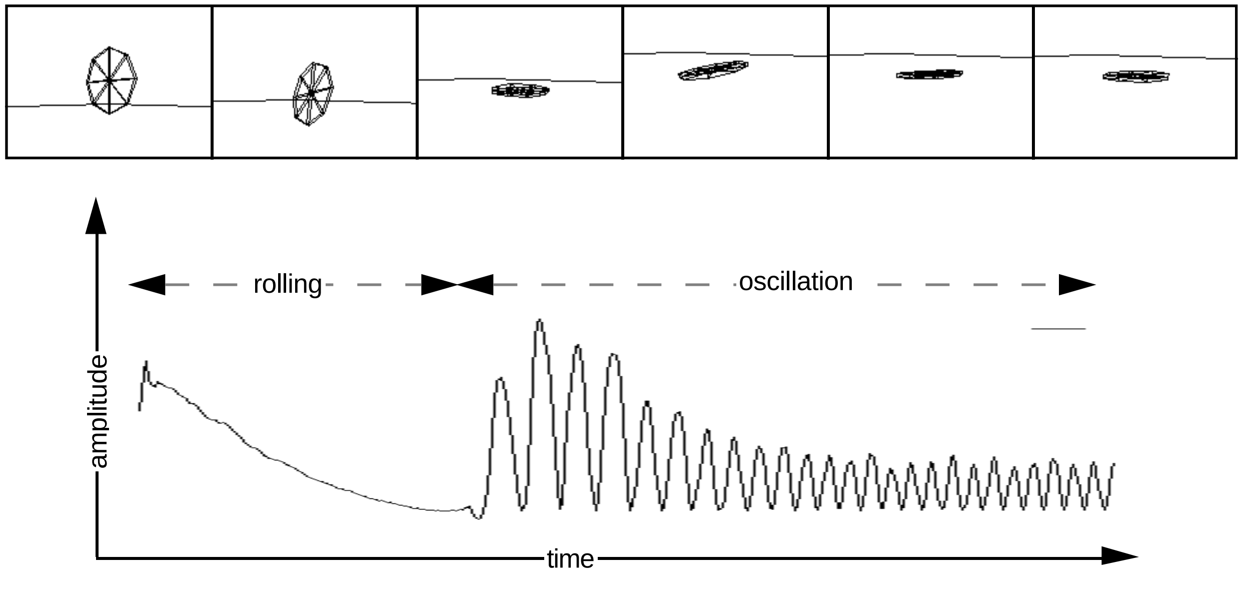


Figure 6. Frames from the animation and the sound produced by a rolling coin

Figure 7 shows a frame from an animation where five chimes are being tossed about by winds animated using physically-based modeling. Three types of sounds were generated: the chimes themselves, the wind, and the background music. Using the timbre tree for a wind chime discussed previously, chime sounds are produced for collision events that occur within the simulation. For the wind sound, we used the wind timbre tree presented. Here the mapping is less direct as the three dimensional vector field of the wind, which has no visible motion, has to be mapped to the scalar wind force of the timbre tree. We chose to map the total wind force felt by each of the chimes to the wind force of timbre tree.

Figure 7. Frame from an animation of wind chimes

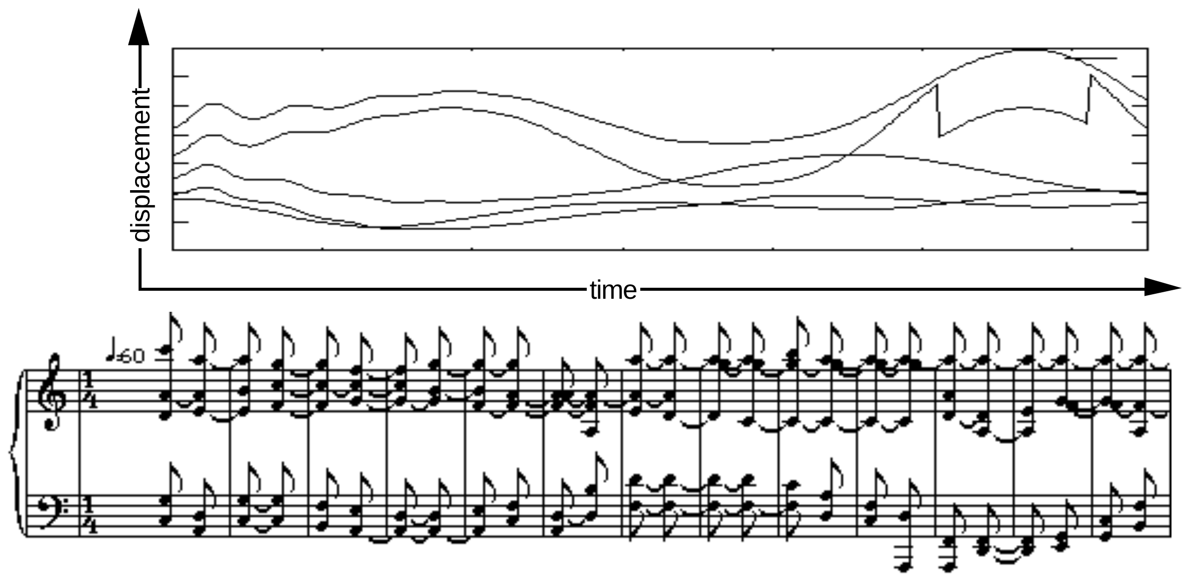


Figure 8. Motions of the physically-based wind chimes and the musical score they produced

The musical score for the background music was generated by mapping the motions of each of the five chimes every half second to individual notes (Figure 8). The top figure shows the actual displacements of the chimes over time and the bottom figure shows the score generated by the system. The scale corresponds to the discretization of the mapped parameters along pitch (vertical) axes while the timing and duration of the notes corresponds to the discretization along the time (horizontal) axes.

The motion for the famous animated lamp learning to limbo (Figure 9) was produced by an experimental motion control system based on genetic programming. In listening to the squeaky hinges of a real lamp, we noticed that the hinges produced a raspy sound whose pitch and amplitude varied with angular velocity. Because of this, we chose a frequency and amplitude modulated sawtooth wave. Since squeaky joints seem to have different sounds when opening and closing, the hinge has two squeak sounds associated with it, one for when the angular motion is in a clockwise direction and another for when the motion is in the counter clockwise direction. The base collision sound was produced with a Fourier synthesized timbre tree whose amplitude is mapped to the collision force (Figure 10). The music for the animation was produced by mapping the angular displacement of the hinges to notes of a blues scale.

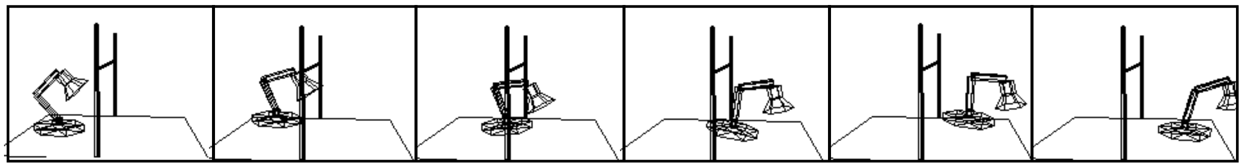


Figure 9. "L*xo Learns to Limbo"

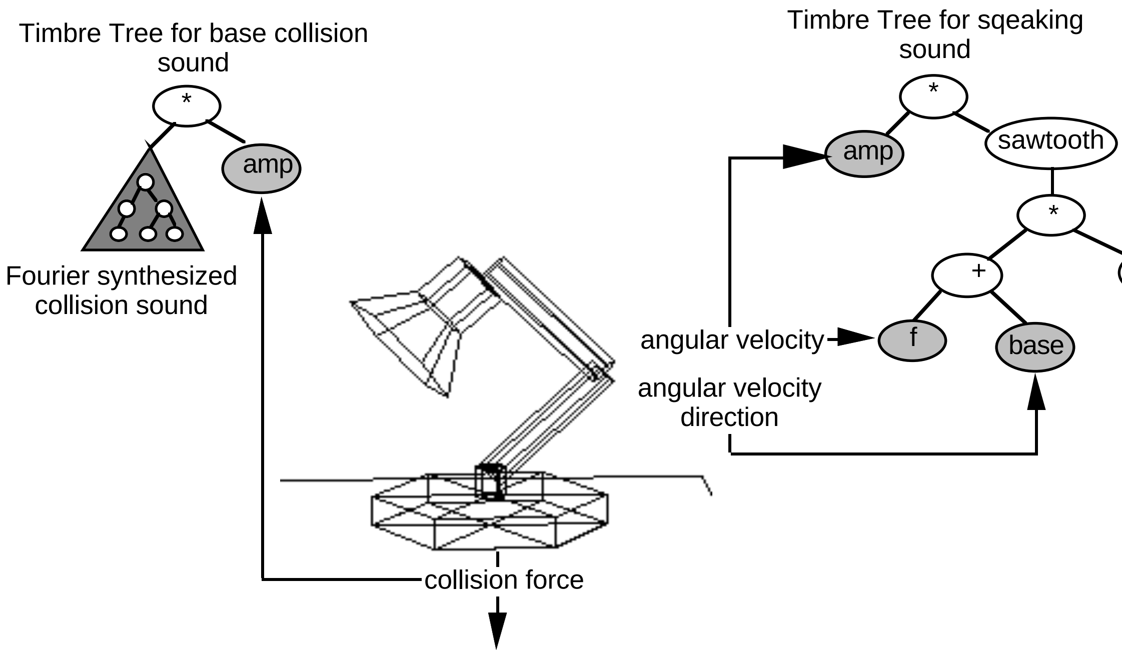

Timbre Tree for base collision sound
*
amp
Fourier synthesized collision sound
Timbre Tree for sqeaking sound
*
amp
sawtooth
*
+
t
f
base
angular velocity
angular velocity direction
collision force

Figure 10. Mapping motion of lamp to timbre trees

CONCLUSION

In the past, sounds have been given only a cursory review in the computer graphics community and have been added to graphics almost as an afterthought. It has become obvious that the aural senses and the added impact of synchronized sound and images will play a bigger role in providing a more complete experience. In this context, it is important that the sound design and its use occur in conjunction with (and with the same level of importance as) the generation of motions and images.

Sound has been a part of many domains such as computer music, human-computer interaction, sonification, virtual environments, and recently in computer animation. The body of knowledge amassed from these disciplines may give the impression that the work, as it relates to computer graphics, is largely finished. But the quality and the effort needed to synchronize sound to motion in computer animations and virtual environments attest to the fact that much work is needed in integrating the domains of sounds and images. It is the purpose of this research to tackle this integration problem. According to this philosophy, we have developed a system of sound representation and synthesis using timbre trees, synchronization by mapping motion parameters to sound parameters, and rendering by adding additional nodes to the final timbre tree.

Future extensions include more coupling between parameters of the motion and the parameters of the timbre tree. For example, the motions of deformable objects can be used to generate sounds (e.g. a flag snapping in the wind). Parameterizing sampled sounds require more studies in what constitutes "generic" qualities of sounds so that they can be mapped to the events that caused the sounds. The system can be extended to real-time by the use of a MIDI based system where the sound generation is handled by dedicated hardware (e.g. a synthesizer or sampler). We are in the process of developing such a real-time sound system to be used in virtual environment applications.

ACKNOWLEDGEMENTS

We would like to thank the Naval Research Laboratory (NAVY N00014-94-K-2009) and NASA Goddard Space Flight Center (NCC 5-43) for their support. We would also like to extend warm appreciations to the students in the Computer Graphics Laboratory at the George Washington University for all their enthusiasm and support.

REFERENCES

1. M. Mathews, *The Technology of Computer Music*, MIT Press, MA, 1969.

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

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

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

5. C. Scaletti, "The Kyma/Platypus Computer Music Workstation" in *The Well-Tempered Object: Musical Applications of Object Oriented Software Technology*, Stephen Travis Pope, ed. MIT Press, 1991.

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

7. W. Gaver, "Synthesizing Auditory Icons," *Proc.of INTERCHI*, 1993.

8. J. Lewis, "Automated Lip-Synch: Background and Techniques," *The Journal of Visualization and Computer Animation*, Vol. 2, No. 4, pp. 118-122.

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

10. T. Takala and J. Hahn, "Sound Rendering," *Proce. of SIGGRAPH'92*, *ACM Computer Graphics*, Vol. 26, No. 2, pp. 211-220.

11. T. Takala, J. Hahn, L. Gritz, J. Geigel, and J. W. Lee, "Using Physically-Based Models and Genetic Algorithms for Functional Composition of Sound Signals, Synchronized to Animated Motion," *International Computer Music Conference (ICMC)*, Tokyo, Japan, Sept. 10-15, 1993.

12. R. Cook, "Shade Trees," *Proc. of SIGGRAPH'84, ACM Computer Graphics*, Vol. 18, No. 3, pp. 195-206.

13. K. Perlin, "An Image Synthesizer," *Proc. SIGGRAPH'85*, *ACM Computer Graphics*, Vol. 19, No. 3, pp. 287-296.

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

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

16. J. Koza, *Genetic Programming*, MIT Press, Cambridge, MA, 1992.

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

18. S. Upstill, *The RenderMan Companion*, Pixar/Addison-Wesley, 1989.

19. J. Nakamula, T. Kaku, T. Noma and S. Yoshida, "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, 1993.

20. E.M. Wenzel, "Localization in Virtual Acoustic Displays," *Presence: Teleoperators and Virtual Environments*, Vol. 1, pp. 80-107, 1992.

21. S. T. Pope and L. E. Fehlen, "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.

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

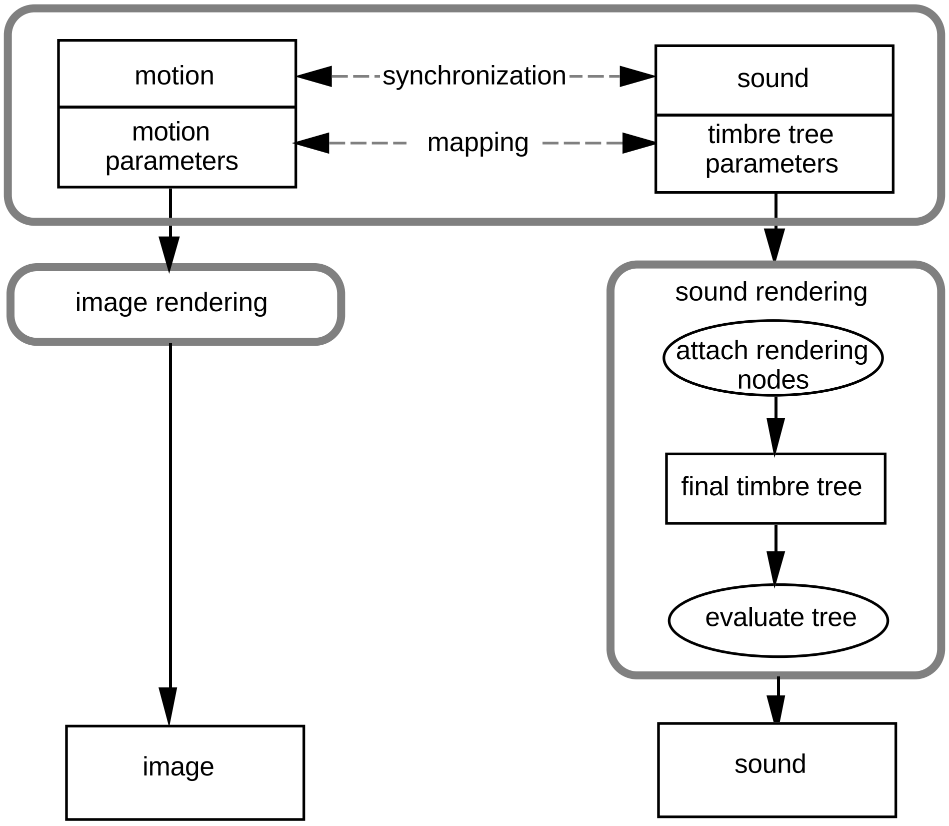


Figure 1. Integrating sound and motion

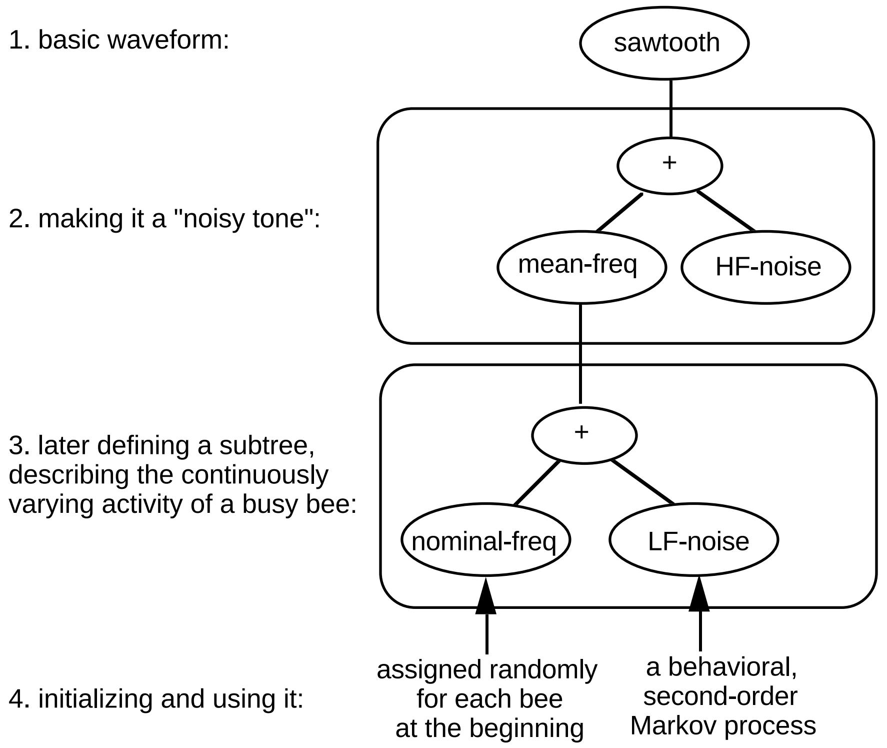
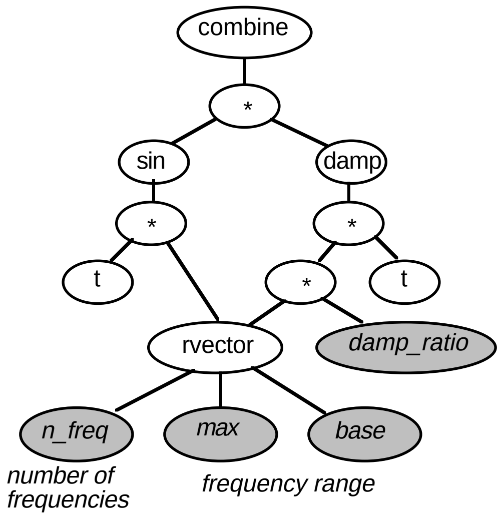
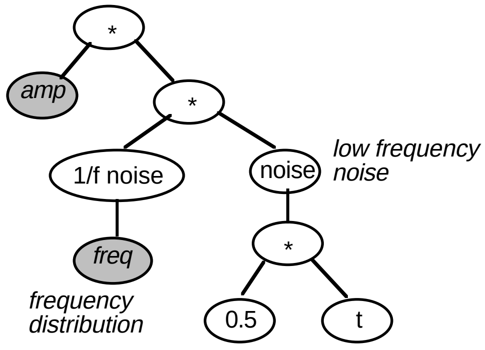

1. basic waveform:
sawtooth
2. making it a "noisy tone":
+
mean-freq
HF-noise
3. later defining a subtree, describing the continuously varying activity of a busy bee:
+
nominal-freq
LF-noise
4. initializing and using it:
assigned randomly for each bee at the beginning
a behavioral, second-order Markov process

Figure 2. Timbre tree design process for bee sounds

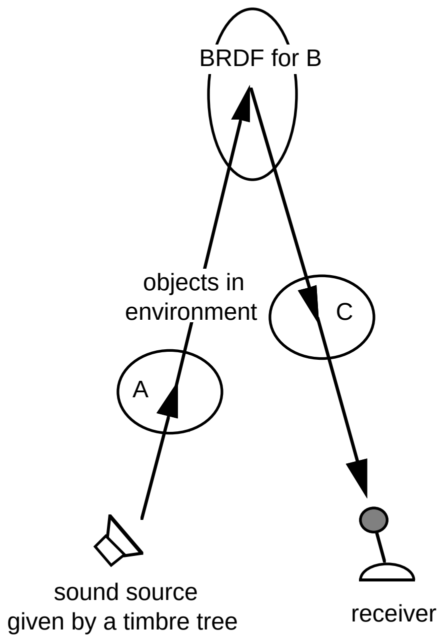


(a) Timbre Tree for collision sound

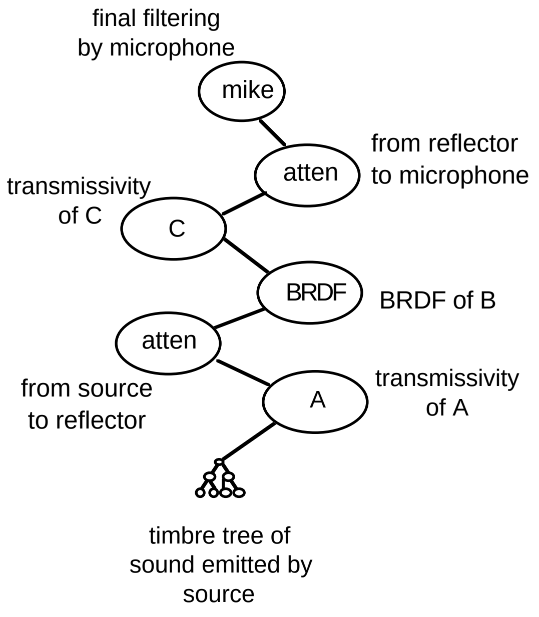


(b) Timbre Tree for wind sound

Figure 3. Timbre tree examples. The darkened nodes represent parameters

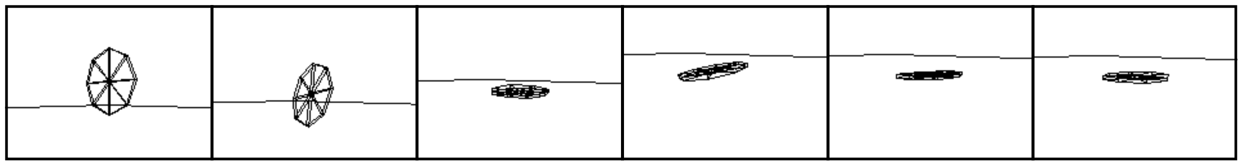


(a) Environmental effects



(b) Timbre tree with specialized rendering nodes

Figure 5. Environmental nodes appended to the timbre tree



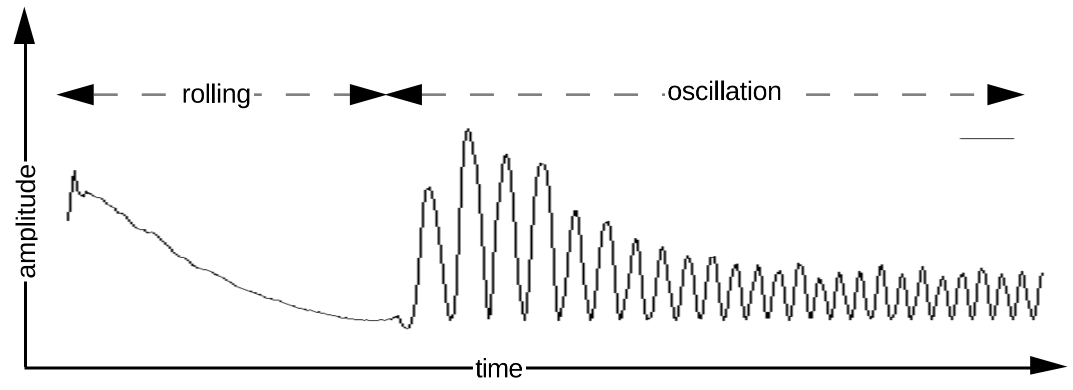


Figure 6. Frames from the animation and the sound produced by a rolling coin

Figure 7. Frame from an animation of wind chimes

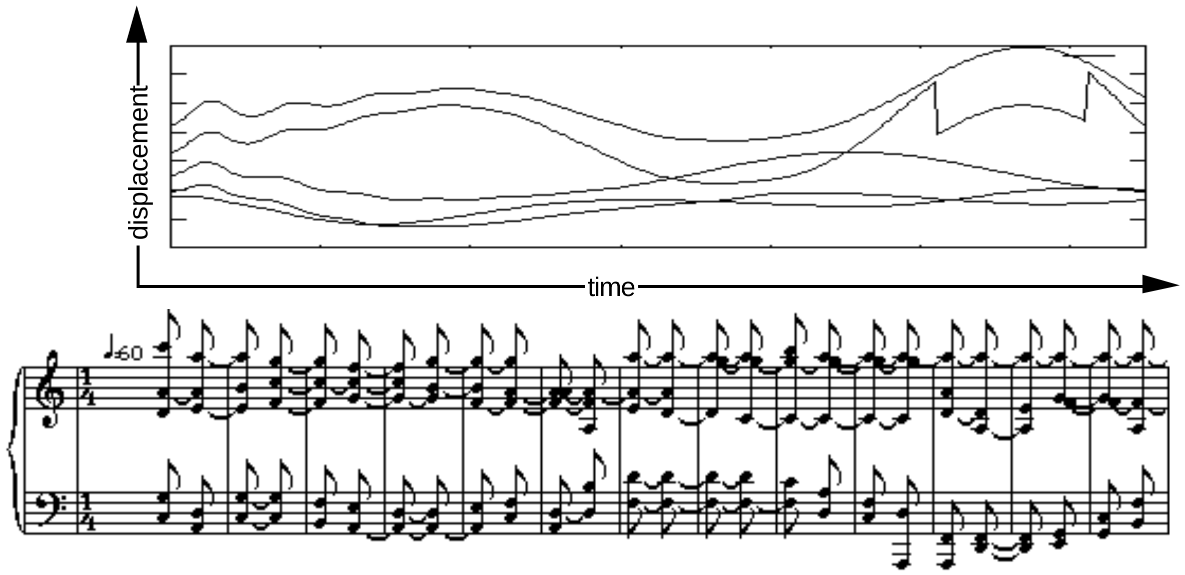


Figure 8. Motions of the physically-based wind chimes and the musical score they produced

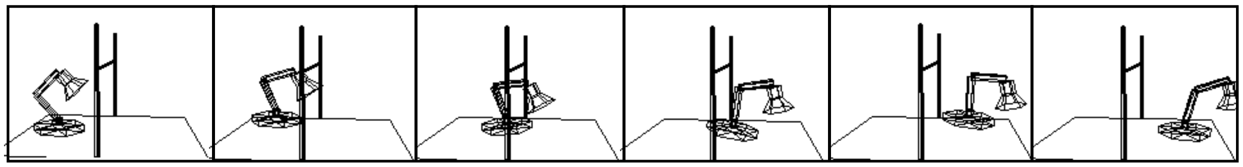


Figure 9. "L*xo Learns to Limbo"

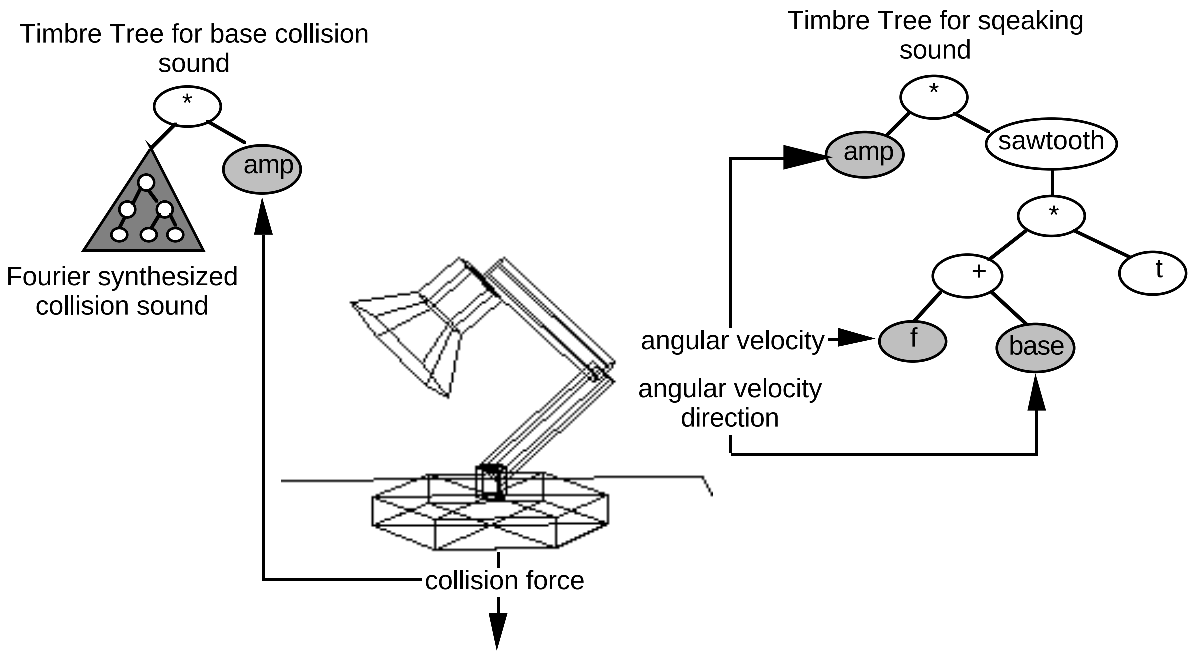

Timbre Tree for base collision sound
*
amp
Fourier synthesized collision sound
collision force
Timbre Tree for sqeaking sound
*
amp
sawtooth
*
+
t
angular velocity
f
base
angular velocity direction

Figure 10. Mapping motion of lamp to timbre trees