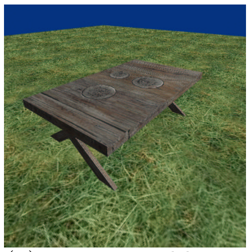
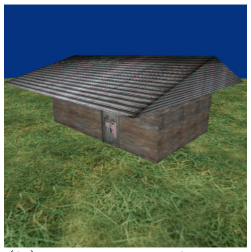
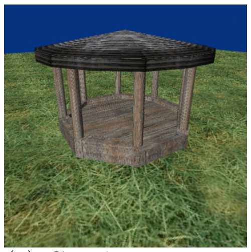
Interactive Simulation of Complex Audio-Visual Scenes

Kees van den Doel, Dave Knott, Dinesh K. Pai

Department of Computer Science
University of British Columbia
Vancouver, Canada


(a) Picnic table with metal plates.


(b) House with metal roof.


(c) Gazebo.

(d) Garbage can.

Figure 1: Several small scenes with sounding objects. Audio for each scene individually can be synthesized in real-time. However putting all the objects together in one scene would overwhelm the CPU unless our mode pruning technique is used.

Abstract

We demonstrate a method for efficiently rendering the audio generated by graphical scenes with a large number of sounding objects. This is achieved by using modal synthesis for rigid bodies and rendering only audible modes. We present a novel real-time algorithm which estimates modal excitations, based on the masking characteristics of the human auditory system, and eliminates modes which are estimated to be inaudible. The efficacy of the algorithm is demonstrated in the context of a simulation of hail falling on sounding objects.

1 Introduction

Interaction sounds are an essential part of computer animation. Even though these sounds are traditionally added by hand by sound designers and Foley artists, there has been recent progress in automatically adding sounds to physically-based computer animation (Doel, Kry, & Pai, 2001; O'Brien, Cook, & Essl, 2001; Hahn, Fouad, Gritz, & Lee, 1998; Hahn et al., 1995; Takala & Hahn, 1992). We review this related work in Section 2.

An advantage of using physically-based modeling over more ad-hoc parametric synthesis methods is that the sounds can in principle be synthesized from physical properties and do not require manual tuning of audio parameters in order to get realistic sounds. This allows the creation of dynamic virtual environments in which objects can be added and their sonic properties determined by their physical properties without requiring the laborious design of separate audio properties.

The modal synthesis approach used by the FoleyAutomatic system (Doel et al., 2001) is attractive for interactive physically-based synthesis of contact sounds, and was demonstrated at SIGGRAPH 2001. We review the background needed in Section 3. The complexity of sound synthesis is linear in the total number active "modes" (or sound sources). However, while this is efficient for a small number of sound-producing objects and interactions, such as in the scenes depicted in Figure 1, complex auditory scenes can quickly overwhelm a simulation. Examples of the kind of complex auditory scenes we have in mind include a shattered window pane falling on a sidewalk and hail falling on a Sunday picnic (see Figure 2).

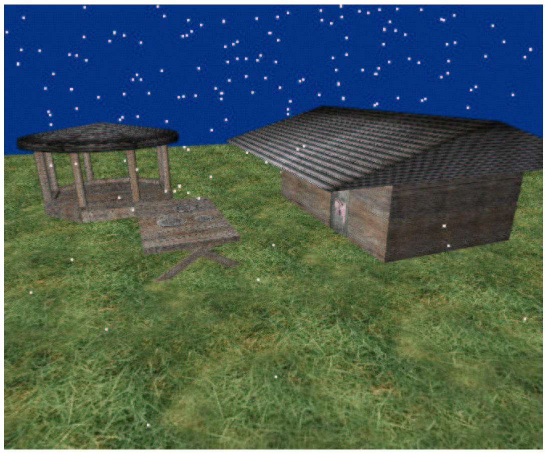


Figure 2: An example of a complex auditory scene. Hail falls on a picnic scene. Sounds are produced by collisions with different objects such as the roof of the house, the picnic table, plates, etc. The sounds depend on the object hit by a particular piece of hail, as well as the location, energy, and the time of each impact. Our technique allows such complex scenes to be simulated at interactive rates. Audio for this scene can not be synthesized in real-time without using our mode pruning technique.

In this paper we show how complex auditory scenes can be simulated at interactive rates on ordinary PC's. We achieve this using a new mode-pruning technique based on human auditory perception that allows us to discard a large number of modes, typically around 90%, without any audible degradation in the sound quality. The discarded modes are masked by other modes and are therefore inaudible. This technique is somewhat analogous to dynamic occlusion culling as used in computer graphics (Sudarsky & Gotsman, 1999). Using the technique described in the paper, we are able to increase the speed of the audio rendering by a factor of about 7-10, depending on the sounds.

To illustrate the technique, we have built a generic hail simulator. We show how to compute particle-based interactions, including determining the timing of the sounds, using graphics hardware. This is accomplished by using a programmable graphics pipeline to calculate both graphics- and audio-related properties of our simulations.

The remainder of the paper is organized as follows. We review other work in sound synthesis for computer animation in Section 2 and go into greater detail about modal synthesis in Section 3 since we extend this approach to complex auditory scenes. In Section 4 we describe our novel mode pruning technique. Section 5 describes our simulator which combines hardware-accelerated particle simulation with sound synthesis. Section 6 discusses the results achieved and we conclude with Section 7.

2 Related Work

In (Doel et al., 2001) it was shown why the modal synthesis technique is an appropriate model for interaction sounds of rigid bodies with contacts. Sound-effects for virtual musical instrument animation were investigated by Cook (Cook, 1995). In (Hahn, Fouad, Gritz, & Lee, 1995; Hahn et al., 1998, 1995; Takala & Hahn, 1992) a number of synthesis algorithms for contact sound are introduced which include a scraping algorithm. A system to render complex environments by reducing the quality of perceptually less important sounds is described in (Fouad & J. K. Hahn, 1996; Fouad, Ballas, & Hahn, 1997).

Synthesizing sounds from finite elements simulations is described in (O'Brien et al., 2001). Automated measurement of modal sound synthesis parameters is described in (Pai et al., 2001).

Synthesis of xylophone sounds, which are similar to the sounds we are interested in, is described in (Chaigne & Doutaut, 1997). In (Essl & Cook, 2000) propagation modeling, a technique closely related to modal synthesis, was proposed to model bowed bars. A number of algorithms for percussion sounds were described in (Cook, 1996).

Psychoacoustics literature has investigated tone-tone masking, both for pure tones and for narrowband noise sources. See for example (Zwicker & Fastl, 1990). Masking models for human hearing resembling ours have been used successfully in perceptual audio coders, see for example (Baumgarte, Ferekidis, & Fuchs, 1995).

In (Lagrange & Marchand, 2001) auditory masking is used in the context of real-time additive synthesis. The algorithm proposed there is very similar to our own. Compression using auditory masking of sinusoidal components is described in (Garcia & J. Pampin, 1999). Additive synthesis with large number of sinusoidal components using the FFT was described in (Freed & Depalle, 1993).

3 Background: Modal Synthesis

When a solid object is struck or scraped or engages in some other contact interaction the rapidly varying forces at the contact points cause deformations to propagate through the solid, causing its outer surfaces to vibrate and emit sound waves. The resulting sound depends on the details of the contact force and on the structure of the object. Examples of musical instruments utilizing solid objects like this are the marimba, the xylophone, and bells.

A good physically motivated synthesis model for solid objects is modal synthesis (Wawrzynek, 1989; Gaver, 1993; Morrison & Adrien, 1993; Cook, 1996; Doel & Pai, 1996, 1998), which models a vibrating object by a bank of damped harmonic oscillators which are excited by an external stimulus. The modal synthesis model is physically well motivated, as the linear partial differential equation for a vibrating system, with appropriate boundary conditions, has as solutions a superposition of vibration modes.

The modal model $\mathcal{M} = \{\boldsymbol{f}, \boldsymbol{d}, \boldsymbol{A}\}$, consists of a vector $\boldsymbol{f}$ of length N whose components are the modal frequencies, a vector $\boldsymbol{d}$ of length N whose components are the decay rates, and an $N \times K$ matrix $\boldsymbol{A}$, whose elements a_{nk} are the gains for each mode at different locations on the surface of the object. The modeled response for an impulse at location k is given by

$$y_k(t) = \sum_{n=1}^{N} a_{nk} e^{-d_n t} \sin(2\pi f_n t), \tag{1}$$

for $t \geq 0$ (and is zero for $t < 0$). The frequencies and dampings of the oscillators are determined by the geometry and material properties (such as elasticity) of the object, and the coupling gains of the modes are related to the mode shapes and are dependent on the contact location on the object. The damping coefficients d_n are related to the $60dB$ decay time by $T_{60} = 3\log(10)/d_n$, where log is the natural logarithm.

The modal model parameters can be obtained from first principles for some simple geometries (Doel & Pai, 1998), or they could be created by "hand and hear", for example using a modal model editor such as described in (Chaudhary, Freed, Khoury, & Wessel, 1998), and for realistic complex objects they can be obtained by fitting the parameters to recorded sounds of real objects (Pai et al., 2001).

A modal resonator bank can be computed very efficiently with an $O(N)$ algorithm (Gaver, 1993; Doel & Pai, 1998; Doel, 1998) for a model of N modes. On a typical desktop computer (1Ghz Pentium III) about 1000 modes can be synthesized in real time at an audio sampling rate of 22050 Hz before saturating the CPU, with a Java implementation. A C implementation is somewhat more efficient, allowing about 2000 modes in real time. In practice the audio synthesis will only be able to use a fraction of the CPU resources, as the graphics, simulation, or game engine will have to run at the same time.

4 Perceptual Pruning for Complex Auditory Environments

Overview

The conventional way to synthesize the sounds of multiple sources is to compute small fragments of digitized sound (called "frames") in separate buffers for each source, and then add or "mix" them. In this manner the CPU time required is just the sum of the times required for each source. The cost of the mixing can usually be ignored or can be performed with hardware in the audio card.

This method makes sense for rendering a piece of music, where the composer has made sure each source or "voice", usually an instrument, is audible. However for natural sounds this is often not the case; some sounds prevent the detection of other sounds. This phenomenon is called "auditory masking" and has been studied extensively for pure tones, narrowband noise, and white noise in the psychoacoustics literature. It should therefore be possible to make the audio synthesis much more efficient if we can determine which parts of the sounds will be masked, and not compute and render those. This approach is similar to techniques used in audio compression techniques, for a review see (Painter & Spanias, 2000).

If we use modal synthesis, or any kind of "additive" synthesis, where a source is composed of many components which are added, we could do an even more fine-grained masking analysis as some modes can be masked by other modes, even within the same object.

Auditory masking is a highly non-linear phenomenon, and the actual masking will be very strongly dependent on the excitations, and on the observation point. For example, an observer on the metal roof will be able to hear a few hail drops falling on the metal plates on the picnic table but during a downpour the roof will make so much noise that the plate will most likely be masked. This means we will have to determine at run time dynamically which modes are audible and which ones are not.

We achieve this by periodically estimating excitations of the modes of all objects, and using known data on masking characteristics of the human auditory system (Zwicker & Fastl, 1990), we construct plausible masking thresholds for each source, and flag those modes that are predicted to be inaudible. Since the masking threshold depends on the absolute level of playback, for optimal performance, loudness data at the observation point should be fed back into the simulation. If this data is not available we use a level independent masking curve, which is slightly less efficient, or set the level by hand. The modal synthesis is then performed using the "on" modes only. We call this process "mode pruning". This technique has some resemblance to those used in perceptual audio compression algorithms (Baumgarte et al., 1995), where the purpose is to save bits, rather than CPU cycles.

The Algorithm

Modal synthesis computes each audio frame by adding individual modal contributions which, for the special case of a single impulsive excitation, are given explicitly in Eq. 1. For a given mode $y_n(t)$, which is computed in the modal synthesis engine at the audio sampling rate, we estimate the current energy in the mode at the beginning of the next audio frame. The instantaneous energy is ill-defined for a non-stationary signal. It is usually defined for audio signals as the time average over some small time interval T of the power y^2:

$$E_c(t) = \int_{t-T/2}^{t+T/2} y^2(s)ds/T. \qquad (2)$$

If $1/T$ is large with respect to the decay rate d_n of a mode but small with respect to the mode frequency f_n, i.e., many periods fall into the window T but the impulse response does not decay much over T, we can approximate Eq. 2 by the instantaneous sum of the potential and kinetic energy (appropriately discretized) as

$$E_c = (y_n^2(t) + f_s(y_n(t) - y_n(t-1))^2/(2\pi f_n)^2)/2 \qquad (3)$$

where f_s is the sampling rate, and t the last time of the previous audio frame. We have verified numerically that Eq. 3 is a reasonably accurate approximation (within 10%) for damping factors (see Eq. 1) up to about $d = 150/s$, for a window of $T = 5ms$. This time window is appropriate to analyze perception of sound and can be considered as the "pitch detection time" of the human ear (Zwicker & Fastl, 1990). We do not have any audio models with stronger dampings than $d = 15/s$, which allows us to use a larger window size of roughly $T = 50ms$ before damping becomes significant within a window.

We save E_c after computing an audio frame, to be used by the pruner. We then estimate the modal energies for the next frame from the new excitation buffers and add this to the current energy estimate E_c. This assumes no damping of the modes, and therefore overestimates the energy in a mode somewhat. For an audio frame length of $N = 1024$ at a sampling rate of $22050Hz$ the effect of damping is not significant, for dampings up to $d = 15/s$. The total estimated energy per mode is then used to perform the masking analysis.

A masking threshold curve is constructed for each mode. This is the threshold of audibility in decibels of a pure tone of some other frequency f in the presence of the masker.

The first criterion to determine the audibility of a mode is that it must lie above the threshold of hearing (Zwicker & Fastl, 1990), which is well approximated (Terhardt, 1979) in decibels by

$$Th(b) = 3.64(f/1000)^{-0.8} - 6.5e^{-0.6(f/1000-3.3)^2} + 10^{-3}(f/1000)^4,$$

where f is the frequency of the masker in Hertz. For the consideration of masking effects by the human auditory system we consider masking of tones with frequencies F_n and power E_n. Pair-wise masking effects have been studied extensively on humans. We use a masking curve consisting of two straight lines on the Bark frequency scale, see Figure 3. The Bark scale is a non-linear function of the frequency in Hertz which is widely used in psychoacoustics. One Bark corresponds to the width of one critical band. See for example (Zwicker & Fastl, 1990). This spreading function is parameterized by a threshold level a_v, and two slopes s_l and s_u. The shape of the masking curve depends on the actual level of playback, louder sounds producing more high-frequency masking. If the playback level is unknown we use a standard level of 60dB.

The masking effects of overlapping spectral sources can be assumed to add linearly (Green, 1967) in a first approximation, though non-linear addition effects have been used to improve perceptual audio coders (Baumgarte et al., 1995).

The masking curve μ for a given mode of frequency b_m (in Barks) at decibel level L_m is given by

$$\mu(b) = max(Th(b), p_m(b)), \qquad (4)$$

with

$$p_m(b) = L_m - a_v - s_l(b_m - b) \text{ for } b < b_m$$

$$p_m(b) = L_m - a_v - s_u(b - b_m) \text{ for } b \geq b_m$$

with a_v and s_l constant, and

$$s_u = 22 - L_m/5.$$

The lower slope is $s_l = 25dB$. The upper slope of the masking curve is level dependent; louder sounds mask more higher frequencies.

The masking threshold parameter a_v is to be determined empirically, higher values leading to less modes being eliminated. We can use this parameter to make an optimal trade-off between quality and computational resources. Published measurements of tone-tone masking (Zwicker & Fastl, 1990) suggest a value of $20dB$ but this refers to masking in quiet laboratory conditions. In our case we consider mode-mode masking in the presence of all the other modes and we can expect that masking occurs sooner, and we can expect a lower value of a_v to produce no audible artifacts. We found by informal tests on ourselves, using the application and objects described below, that in all cases a value of $5dB$ produced no perceptual change in the audio, and values as low as $2dB$ produced some small audible difference but with little reduction of quality. These tests were performed by simply running the simulation and changing the value of a_v with a slider and listening for differences in audio quality. While this is not a very rigorous approach, the distortions for small values of a_v are quite obvious and can be observed without resorting to elaborate user tests if only order of magnitudes of the value of a_v are required. The optimal parameter a_v has been measured more rigorously with formal user studies for self-masking among the modes of several objects (Doel, Pai, Adam, Kortchmar, & Pichora-Fuller, 2002).

We eliminate the inaudible modes by first removing all modes below absolute threshold and then constructing the upper envelope of all the masking curves defined by Eq. 4 for each source removing all modes that fall below it. If we do this naively by checking all pairs of N modes, the computational cost for N modes will be $O(N^2)$. We can improve the expected cost by first sorting the modes by loudness with cost $O(N \log N)$ and then eliminating modes masked by the loudest first. If we assume the modes are uniformly distributed over frequency range (in Barks) and level (in dB) it is not difficult to show the expected number of maskers is $O(\sqrt{N})$ [1], and the expected cost of the pruning will be $O(N\sqrt{N})$. We found this to be satisfactory for our applications, producing negligible overhead compared to the modal synthesis itself, which is of order $O(f_s N)$, with f_s the sampling rate. Because of the large constant factor f_s the modal synthesis remains the most expensive. It is possible to improve the complexity of the pruning to $O(N \log N)$ by adapting the algorithm presented in (Lagrange & Marchand, 2001).

To illustrate the operation of the pruning within a single object, in Figure 4 we show the raw modes of a complex object, a metal garbage can used in our simulation. We show the masking curve, and the absolute threshold of hearing. This assumes all modes are excited equally, i.e., the contact force has a flat spectrum, with a total loudness of 70dB. Masking effects with $a_v = 5dB$ remove almost 95% of the measured modes.

Pseudo-code

We now present the details of the algorithm in pseudo-code.

0. N = total number of modes of all objects
   T = length of the audio frame.
   L = playback level (obtained from a monitor mike or set manually).
   a_v = masking threshold set by user.
   $MData[]$ = array of mode data from all objects.
   $Mdata[i].f$ = frequency of mode i.
   $Mdata[i].a$ = gain of mode i.
   $Mdata[i].E_c$ = estimated energy of mode i.
   $m(f; f_m, E_m, a_v)$ is the masking curve for a masker with energy E_m and frequency f_m.

1. For every excitation source of impulses (a sample frame of length T, obtained from the particle simulator) compute the energy E_s of the source as the norm of the buffer.

2. For all modes in $MData[]$, estimate the energy E_c in each mode by adding the current energy of the mode as computed with Eq. 3 to aE_s with E_s the energy in the source of the modal object, and a the gain factor of the mode.

4. Calibrate the estimated energies E_c such that the total energy is the observed playback loudness L.

5. Sort $MData[]$ on the E_c values in descending order using a quicksort algorithm.

6. Eliminate modes below threshold of hearing:

```
for i=1 to N
  if MData[i].E_c > Th(MData[i].f)
     MData[i].isOn = true;
     MData[i].isMasker = true; // is potential masker
  else
     MData[i].isOn = false;
     MData[i].isMasker = false;
  endif
endfor
```

7. Do the masking analysis:

[1]Numerical experiments showed the number of maskers to be about $2\sqrt{N}$ for large N.

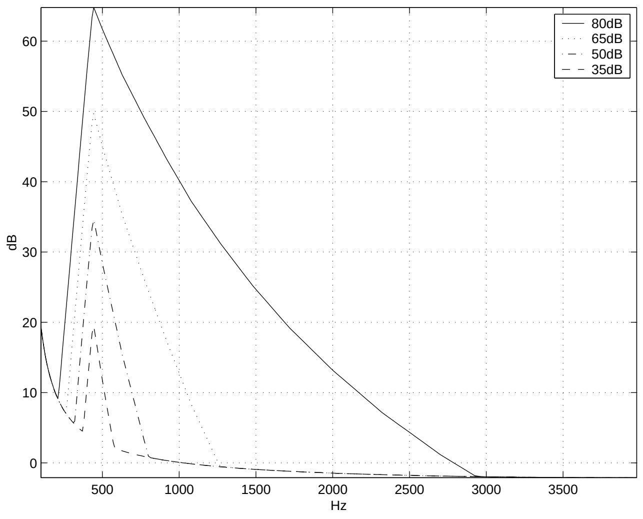


Figure 3: Masking threshold curves for a source at 440 Hz for various loudness levels of the masker, using a_v = 15dB. Any pure tone sound whose frequency-loudness coordinates fall below the masking curve will be inaudible in the presence of the masker.

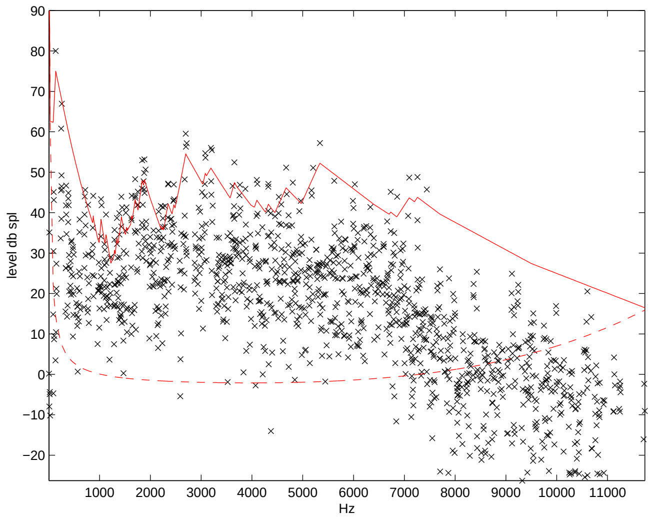


Figure 4: 1361 measured raw modes of the garbage are be reduced to 72 (the modes above the masking curve) at a playback level of 70dB. The masking threshold used was $a_v = 5dB$. The dotted line is the absolute threshold of hearing.

```
for i=1 to N-1
  if MData[i].isMasker
    for k=i+1 to N
      ifMData[k].isOn
        f = MData[k].f;
        th = m(f;MData[i].f,Mdata[i].Ec,av);
        if MData[k].Ec < th
          MData[k].isOn = false;
          MData[k].isMasker = false;
        else if MData[k].Ec < th + av
          MData[k].isMasker = false; // can't be masker
        endif
      endif
    endfor
  endif
endfor
```

8. Compute the next audio frame using standard modal synthesis with the “on” modes.

9 GOTO 1.

5 Particle Simulation and Collision Detection

The sound synthesis algorithm can be applied to any kind of situation where many objects create sounds simultaneously. In order to test the algorithm described in the above in the context of a graphical simulation we have chosen to simulate the sound and visuals of hail falling on multiple objects. This provides a suitably complex environment to test these ideas in practice, and also provides an example of how to integrate the audio algorithm in a simulation.

In many real-time graphics applications, the bottleneck for overall speed is the CPU, with the graphics processing unit (GPU) often sitting idle for a significant amount of time. This is certainly the case for applications such as ours, in which a large percentage of the computer's processing power is typically used for simulation and collision detection, in addition to audio synthesis of the scene's acoustic properties.

We attempt to alleviate the load on the computer's CPU by offloading as much computation as possible to the graphics hardware. This is done with the aid of the programmable vertex engine described in (Lindholm, Kilgard, & Moreton, 2001). Using a special construct, the *Impact Map*, which we describe below, we map the computation of impact events which drive the audio synthesis onto a problem the GPU can handle. An additional complication is that we need to compute the impact times within the accuracy of the audio rate (at least 22050Hz for high quality sound) which is much higher than the graphics frame rate.

Particle Simulation

For our graphical scenes, we simulate particles of the type described in (Reeves, 1983). The particles' motion is completely determined by initial conditions, with the exception of the calculation of impacts with (and response to) interfering bodies, and is given in closed form by a parametric equation $\boldsymbol{p} = \boldsymbol{g}(t)$, where t is the global simulation time and $\boldsymbol{p}$ is the instantaneous three-dimensional position.

The initial attributes of each particle, such as position and velocity, are passed as data to the programmable vertex engine. In addition, particles are “regenerative” in the sense that they are created with an initial state, simulated for a finite period of time and then expire, at which time they are reborn with possibly altered starting conditions. The advantage is that once a particle's parameters are specified the GPU can compute its trajectory without any further involvement of the CPU.

Collision Detection

We allow particles to impact with colliders, which are objects whose surfaces are defined implicitly by some formula $f(\boldsymbol{p}) = 0$, where $\boldsymbol{p}$ is a point on the surface. The collision time of a particle impacting the surface can be found by solving $f(\boldsymbol{g}(t)) = 0$ for t.

As an example, in a typical scene we will perform collision detection of particles moving only under the influence of gravity against bounded planar sections such as quadrilaterals and elliptical discs. Thus, given constant acceleration $\boldsymbol{a}$, and initial position and velocity $\boldsymbol{x_0}$ and $\boldsymbol{v_0}$, the instantaneous position of a particle is given by $\boldsymbol{p}(t) = \boldsymbol{g}(t) = \boldsymbol{a}t^2/2 + \boldsymbol{v_0}t + \boldsymbol{x_0}$. The unbounded surface of the collider is defined by $f(\boldsymbol{p}) = \boldsymbol{N} \cdot (\boldsymbol{p} - \boldsymbol{p_0}) = 0$, where $\boldsymbol{N}$ is the surface normal of the collider and $\boldsymbol{p_0}$ is any point on the surface. The collision time is then given implicitly by $f(\boldsymbol{g}(t)) = \boldsymbol{N} \cdot (\boldsymbol{a}t^2/2 + \boldsymbol{v_0}t + \boldsymbol{x_0} - \boldsymbol{p_0}) = 0$, which is a quadratic equation in t. Solving a quadratic equation using vertex programs is relatively easy, and is also a scalar operation, meaning that using vector math we can test for collisions with up to four colliders simultaneously.

Using similar math, we can calculate intersection of the motion of unaccelerated particles moving in straight lines with second order surfaces such as natural quadrics. Again, this is simple since it also results in a quadratic equation for t.

We detect a collision by comparing the computed collision time with the current simulation time. We can also process impacts even further, in order to detect collisions with more complex objects. This is done by associating with each collider one or more functions $c(\boldsymbol{p}) > 0$, that test whether or not the impact point matches some criterion. For example, intersections with a circular disc require $c(\boldsymbol{p}) = r - |\boldsymbol{g}(t) - \boldsymbol{q}| > 0$, where r and $\boldsymbol{q}$ are, respectively, the radius and center point of the disc.

For graphical display, we eliminate particles that have already intersected a collider by assigning them a reference alpha color and directing the graphics hardware to not render any fragment that has this alpha value.

Using Graphics Hardware to Drive Audio Synthesis

Using the graphics hardware to compute impact events for the audio portion of the simulation is considerably harder than using it to simulate particles for purely graphical purposes. To do so, we must find some way to associate impacts computed on the GPU with events that drive the audio synthesis. The difficulty is that the hardware graphics pipeline is essentially a one-way street. The only destination for resultant data from calculations performed on the GPU is the framebuffer. [2] The output data is therefore stored as a two-dimensional array of pixel values, each of which is interpreted as a set of three or four values, usually used to represent color channels. The precision of these values is commonly limited to eight bits per channel.

When we are using the vertex programs to calculate impact events, we do not submit the entire geometry of the particle to the graphics hardware. Instead, we ask the GPU to render a single point primitive with the same simulation parameters as the particle, with the exception that its position corresponds to the particle's center of mass. Furthermore, we do not render this point primitive as part of the graphical scene. Instead, we redirect rendering to another graphics context which we will refer to hereafter as the impact map. Since we do not wish the impact map to be visible, the ideal rendering content is an off-screen rendering surface (also known as a pbuffer). For systems that do not support this feature, we can render to the rear (invisible) portion of the main framebuffer, which we clear between rendering the impact map and the graphical scene.

The impact map holds a two-dimensional representation of the locations at which particles have impacted with colliders. At each simulation step, the GPU vertex program computes whether or not each particle has impacted a collider between the previous and current frames of the animation. In the case of impact with a planar collider, if such an impact has occurred, then the exact impact location is calculated. This impact location is then transformed into the space of the collider, normalized to fit the viewport of the impact map, and the point primitive is drawn at that location. If no impact occurred, the point primitive is transformed to a location outside of the impact map's viewport and therefore culled during fragment generation.

When we detect a collision and render it into the impact map, we can also record some auxiliary data about the impact. This data is stored in the color channels. For instance, given three color channels, we may choose to encode the identity of the collider that was impacted, the exact impact time, and the velocity with which the particle hit the collider.

As an example, in our picnic scene the energy with which the hailstones strike the buildings or table is calculated in the

[2]There are alternate render targets such as texture maps, but the information is still accessed in graphics hardware memory as a pixel array

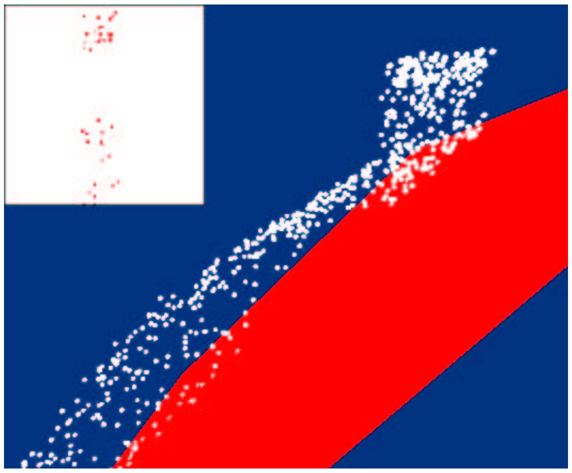


Figure 5: A scene showing a multitude of particles striking and bouncing off of a single planar section. The impact map (inset) describes a two-dimensional parameterization of impact locations and characteristics. It is computed in real time using programmable graphics hardware.

vertex program and delivered through the first color channel. The probability distribution of hailstone energy follows a $1/E$ law. Many natural stochastic phenomena follow $1/f$ distributions (West & Shlesinger, 1989). We experimented with other distributions such as a box function but the $1/E$ distribution sounded more natural.

Note that it is possible for more than one collision to map into the same pixel of the impact map. For many applications, such as rendering "splat" textures or even our audio synthesis, this is acceptable since the results of one impact may mask another. Alternatively, recent advances in programmable graphics hardware may allow us to use the pixel shading stage of the graphics pipeline to combine impact data in more complex ways.

After rendering the impact map, we read the pixels from the rendering surface and parse through them looking for impacts. If an impact is found, then we excite the relevant modes of the model associated with the collided object. In addition, we may choose to post-process the impact point or its associated data in some manner. For instance, we may choose to associate multiple modal models with a single collision object, as in the case of our picnic table with plates, or some other object that has varying modal models depending on the impact location.

6 Results

The mode pruning algorithm was implemented in JASS (Doel & Pai, 2001), a unit generator based synthesis class hierarchy written in Java. A stand-alone audio testing application was made to synthesize the sound of hail falling on multiple objects in real-time. The excitations were modeled as random impulses. Audio was generated for several values of the masking threshold parameter a_v and compared aurally to determine the effect of this. The CPU load generated under various conditions using the algorithm was also monitored in this manner.

The testing environment consists of a scene with 9 complex objects: 4 metal roof panes, 3 metal plates of various

Object	modes	reduced
Roof pane 1	108	93
Roof pane 2	165	128
Roof pane 3	216	176
Roof pane 4	237	206
Plate 1	239	97
Plate 2	239	105
Plate 3	239	98
Table	100	60
Garbage Can	1361	72
Total	2873	1066

Table 1: Modal models used for simulation. Shown are the raw modes and the modes after taking self-masking within one object into account, assuming $a_v = 5$ and a flat excitation spectrum.

a_v	K	CPU load	quality
0	3%	30%	poor
2.5	6%	33%	good
5	10%	38%	best
20	23%	75%	best

Table 2: Performance of the mode-pruning algorithm for various values of the masking threshold a_v. We indicate the number of surviving modes K, the CPU load and the quality of the sound, where "best" indicates it sounds the same as the full synthesis with all the modes. The audio frame buffer length was $T = 46ms$. The audio sampling rate used was 22050Hz.

sizes, a wooden table, and a metal garbage can. Modal models for some of the objects (roof panes and plates) were constructed mathematically by solving for the modal frequencies for rectangular and circular plates. The material properties determined the damping of the modes using a model based on the modulus of elasticity as described in (Doel & Pai, 1998). The other objects were modeled by parameter extraction from recorded impulse responses as described in (Pai et al., 2001). In Table 1 we list the objects and the number of modes for each of them. The second column indicates the number of modes reduced by self-masking on each object, using a masking threshold of $a_v = 5dB$.

It was found that values of a_v as low as $5dB$ produced no audible difference compared to a full synthesis with all modes, as far as we could tell by listening ourselves. When a_v was reduced to $2.5dB$ the sound was audibly different though still of reasonable quality. Reducing a_v down to $0dB$ produced a marked decrease in quality. In Table 2 we summarize the results. The overhead of the pruning took 8% of the CPU resources at an audio frame size of $T = 46ms$. Using perceptually transparent pruning (i.e., no audible artifacts compared to a full synthesis) the CPU load is 38%, compared to 280% for a full modal synthesis, which can not run in real-time.

We have also integrated the audio with our particle simulation using all objects described above. The user is allowed to move a cloud around with the mouse and set the amount of hail generated with a slider. The audio is rendered synchronously with the particle simulation. With sliders the user is allowed to change the audio mode pruning masking threshold parameter. A graphical display of the average percentage of the number of total modes used is also displayed. It can be seen that the number of modes depends on the hail density, the location of the hail shower (i.e., it may miss some objects and as a result less modes will be excited), and on the masking threshold. A threshold value as low as $5dB$ produces no audible difference compared to a setting of $100dB$, though a much larger percentage of modes is pruned. When a_v is decreased below $5dB$, the sound becomes audibly different, as some audible modes have been omitted.

7 Conclusions

We have presented a novel algorithm to render complex sounds from many objects simultaneously using modal synthesis. This was achieved by considering the audibility of modes as predicted by a simplified model of human hearing, and performing this analysis at run-time at a relatively slow rate compared to the audio sampling rate. The modes predicted to be inaudible then do not have to be synthesized for the next audio frame. This speeds up the audio synthesis by a factor 7-9 for a representative application we have implemented: hail falling on a picnic scene. Further improvements can probably be achieved by using a more accurate model of auditory masking than the simple model we have used.

The masking parameter a_v, which determines how many modes are eliminated, was determined roughly by changing its value in real-time with a slider and listening to the sounds ourselves. The resulting value was consistent with formal user studies performed previously by some of us which measured the value of a_v for impact sounds of single objects.

We have also shown a method for using graphics hardware to compute modal excitation events associated with our sample environments. The impact map was introduced as a technique for detecting and reporting impacts of particles with interfering objects.

Our dynamic masking algorithm can also be applied to environments with not just impulsive excitations, but also sliding and rolling interactions. In this case we have to adjust the excitation estimation procedure as the contact forces will no longer have a flat spectrum when modeled with our techniques as described in (Doel et al., 2001), but this is straightforward. We do not expect this to change the value of a_v, as excitations with a non-flat spectrum can be emulated by a flat-spectrum excitation with properly modified gain coefficients.

The position of the user with respect to the audio sources, which we have taken to be fixed, can be incorporated easily dynamically by changing the modal gain parameters as a function of the location of the observer. Refinements of the masking algorithm could take into account the relative positions of the sources; presumably spatially separated sources will produce less masking than sources which are close together.

We believe a similar technique can also be used in the context of other real-time audio synthesis techniques including sample-playback based systems. In this case the details of the masking analysis would be somewhat different, but one could for example perform an FFT based analysis and consider inter-source masking per critical band.

References

Baumgarte, F., Ferekidis, C., & Fuchs, H. (1995). A Nonlinear Psychoacoustic Model Applied to the ISO MPEG Layer 3 Coder. In *Preprint 4097, 99th AES Convention, New York, October 1995.* New York.

Chaigne, A., & Doutaut, V. (1997). Numerical simulations of xylophones. I. Time domain modeling of the vibrating bars. *J. Acoust. Soc. Am.*, *101*(1), 539–557.

Chaudhary, A., Freed, A., Khoury, S., & Wessel, D. (1998). A 3-D Graphical User Interface for Resonance Modeling. In *Proceedings of the international computer music conference.* Ann Arbor.

Cook, P. R. (1995). Integration of physical modeling for synthesis and animation. In *Proceedings of the international computer music conference* (pp. 525–528). Banff.

Cook, P. R. (1996). Physically informed sonic modeling (PhISM): Percussive synthesis. In *Proceedings of the international computer music conference* (pp. 228–231). Hong Kong.

Doel, K. v. d. (1998). *Sound synthesis for virtual reality and computer games.* Unpublished doctoral dissertation, University of British Columbia.

Doel, K. v. d., Kry, P. G., & Pai, D. K. (2001). FoleyAutomatic: Physically-based Sound Effects for Interactive Simulation and Animation. In *Computer graphics (acm siggraph 01 conference proceedings)* (pp. 537–544). Los Angeles.

Doel, K. v. d., & Pai, D. K. (1996). Synthesis of Shape Dependent Sounds with Physical Modeling. In *Proceedings of the International Conference on Auditory Display 1996.* Palo Alto.

Doel, K. v. d., & Pai, D. K. (1998). The Sounds of Physical Shapes. *Presence*, *7*(4), 382–395.

Doel, K. v. d., & Pai, D. K. (2001). JASS: A Java Audio Synthesis System for Programmers. In *Proceedings of the International Conference on Auditory Display 2001.* Helsinki, Finland.

Doel, K. v. d., Pai, D. K., Adam, T., Kortchmar, L., & Pichora-Fuller, K. (2002). Measurements of Perceptual Quality of Contact Sound Models. In *Proceedings of the International Conference on Auditory Display 2002.* Kyoto, Japan.

Essl, G., & Cook, P. R. (2000). Measurements and efficient simulation of bowed bars. *J. Acoust. Soc. Am.*, *108*(1), 379–388.

Fouad, H., Ballas, J., & Hahn, J. (1997). Perceptually Based Scheduling Algorithms for Real-Time Synthesis of Complex Sonic Environments. In *Proc. int. conf. auditory display.* Palo Alto, USA.

Fouad, H., & J. K. Hahn, J. K. (1996). A framework for integrating sound into virtual environment interfaces. In *IS&T/SPIE Symposium on Electronic Imaging: Science & Technology.* San Jose, CA.

Freed, A., & Depalle, X. (1993). Synthesis of Hundreds of Sinusoidal Partials on a Desktop Computer without Custom Hardware. In *Proceedings of the international conference on signal processing applications and technology.* Santa Clara.

Garcia, G., & J. Pampin, J. (1999). Data Compression of Sinusoidal Parameters using Psychoacoustics Masking. In *Proceedings of the international computer music conference.* Beijing, China.

Gaver, W. W. (1993). Synthesizing Auditory Icons. In *Proceedings of the ACM INTERCHI 1993* (pp. 228–235).

Green, D. M. (1967). Additivity of Masking. *J. Acoust. Soc. Am.*, *41*(6).

Hahn, J. K., Fouad, H., Gritz, L., & Lee, J. W. (1995). Integrating sounds and motions in virtual environments. In *Sound for Animation and Virtual Reality, SIGGRAPH 95 Course 10 Notes.* Los Angeles.

Hahn, J. K., Fouad, H., Gritz, L., & Lee, J. W. (1998). Integrating Sounds and Motions in Virtual Environments. *Presence*, *7*(1), 67–77.

Hahn, J. K., Geigel, J., Lee, J. W., Gritz, L., Takala, T., & Mishra, S. (1995). An integrated approach to motion and sound. *Journal of Visualization and Computer Animation*, *28*(2), 109–123.

Lagrange, M., & Marchand, S. (2001). Real-time additive synthesis of sound by taking advantage of psychoacoustics. In *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-01).* Limerick, Ireland.

Lindholm, E., Kilgard, M. J., & Moreton, H. (2001). A User-Programmable Vertex Engine. In *Computer graphics (acm siggraph 01 conference proceedings).* Los Angeles, USA.

Morrison, J. D., & Adrien, J.-M. (1993). Mosaic: A framework for modal synthesis. *Computer Music Journal*, *17*(1).

O'Brien, J. F., Cook, P. R., & Essl, G. (2001). Synthesizing Sounds from Physically Based Motion. In *Siggraph 01* (p. 529-536). Los Angeles.

Pai, D. K., Doel, K. v. d., James, D. L., Lang, J., Lloyd, J. E., Richmond, J. L., & Yau, S. H. (2001). Scanning physical interaction behavior of 3D objects. In *Computer Graphics (ACM SIGGRAPH 01 Conference Proceedings)* (pp. 87–96). Los Angeles.

Painter, T., & Spanias, A. (2000). Perceptual Coding of Digital Audio. *Proceedings of the IEEE*, *88*(4), 451–460.

Reeves, W. T. (1983). Particle systems - a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, *2*(2), 91 – 108.

Sudarsky, O., & Gotsman, C. (1999). Dynamic scene occlusion culling. *IEEE Transactions on Visualization and Computer Graphics*, *5*(1), 13 – 28.

Takala, T., & Hahn, J. (1992). Sound rendering. *Proc. SIGGRAPH 92, ACM Computer Graphics*, *26*(2), 211–220.

Terhardt, E. (1979). Calculating Virtual Pitch. *Hearing Research*, 155–182.

Wawrzynek, J. (1989). VLSI models for real-time music synthesis. In M. Mathews & J. Pierce (Eds.), *Current directions in computer music research.* MIT Press.

West, B. J., & Shlesinger, M. (1989). On the ubiquity of 1/f noise. *Inernational Journal of Modern Physics B*, *3*(6), 795–819.

Zwicker, E., & Fastl, H. (1990). *Psychoacoustics.* Berlin: Springer-Verlag.

8 Figure captions

Figure 1: Several small scenes with sounding objects. Audio for each scene individually can be synthesized in real-time. However putting all the objects together in one scene would overwhelm the CPU unless our mode pruning technique is used.

Figure 2: An example of a complex auditory scene. Hail falls on a picnic scene. Sounds are produced by collisions with different objects such as the roof of the house, the picnic table, plates, etc. The sounds depend on the object hit by a particular piece of hail, as well as the location, energy, and the time of each impact. Our technique allows such complex scenes to be simulated at interactive rates. Audio for this scene can not be synthesized in real-time without using our mode pruning technique.

Figure 3: Masking threshold curves for a source at 440 Hz for various loudness levels of the masker, using $a_v = 15$dB. Any pure tone sound whose frequency-loudness coordinates fall below the masking curve will be inaudible in the presence of the masker.

Figure 4: 1361 measured raw modes of the garbage are be reduced to 72 (the modes above the masking curve) at a playback level of 70dB. The masking threshold used was $a_v = 5dB$. The dotted line is the absolute threshold of hearing.

Figure 5: A scene showing a multitude of particles striking and bouncing off of a single planar section. The impact map (inset) describes a two-dimensional parameterization of impact locations and characteristics. It is computed in real time using programmable graphics hardware.