Interactive Sound Synthesis for Large Scale Environments

Nikunj Raghuvanshi* Ming C. Lin†

Department of Computer Science
University of North Carolina at Chapel Hill

Abstract

We present an interactive approach for generating realistic physically-based sounds from rigid-body dynamic simulations. We use spring-mass systems to model each object's local deformation and vibration, which we demonstrate to be an adequate approximation for capturing physical effects such as magnitude of impact forces, location of impact, and rolling sounds. No assumption is made about the mesh connectivity or topology. Surface meshes used for rigid-body dynamic simulation are utilized for sound simulation without any modifications. We use results in auditory perception and a novel priority-based quality scaling scheme to enable the system to meet variable, stringent time constraints in a real-time application, while ensuring minimal reduction in the perceived sound quality. With this approach, we have observed up to an order of magnitude speed-up compared to an implementation without the acceleration. As a result, we are able to simulate moderately complex simulations with upto hundreds of sounding objects at over 100 frames per second (FPS), making this technique well suited for interactive applications like games and virtual environments. Furthermore, we utilize OpenAL and EAX™ on Creative Sound Blaster Audigy 2™ cards for fast hardware-accelerated propagation modeling of the synthesized sound.

CR Categories: H.5.5 [Information Interfaces and Presentation]: Sound and Music Computing—Modeling, Methodologies and techniques; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: Sound Synthesis, Rigid-Body Simulation, OpenAL

1 Introduction

Most interactive applications today employ recorded sound clips for providing sounds corresponding to object interactions in a scene. Although this approach has the advantage that the sounds are realistic and the sound-generation process is quite fast, there are many physical effects which cannot be captured by such a technique. For instance, in a typical collision between two objects, the loudness and timbre of the sound is determined by the magnitude and location of the impact forces – a plate sounds very differently when struck on the edge compared to when it is struck in the middle. Consequently, if the collision scenario changes slightly, the sound exhibits a corresponding change. Such subtle effects can add substantial realism to a typical scene by avoiding the repetitiveness common to recorded sound clips. However, developing a system which produces sound using physically-based principles in real time poses substantial difficulties. The foremost requirement is the presence of an efficient dynamics engine which informs the sound system of object collisions and the forces involved. For this work, we have developed a fast and robust rigid-body simulator, but many present day games meet this requirement. Given a dynamics simulator, the main challenge is to synthesize the sound efficiently enough to play in real time while taking only a small portion of the total running time, which is usually dominated by graphics and rigid-body simulation. Typically the sound system can only afford a few hundred CPU cycles per object per sound sample for many interactive applications.

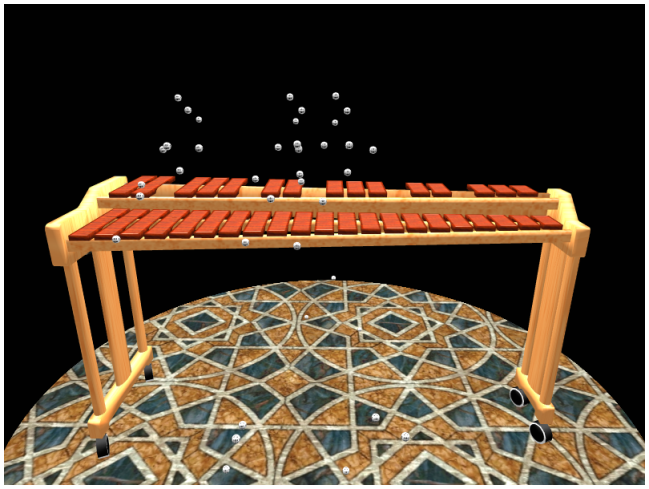

Figure 1: Numerous dice fall on a three-octave xylophone in close succession, playing out the song "The Entertainer" (see the video). Our algorithm is able to produce the corresponding musical tones at more than 500 FPS for this complex scene, with audio generation taking 10% of the total CPU time, on a 3.4GHz Pentium-4 Laptop with 1GB RAM.

Main Results: In this paper, we present an approach which meets the interactive performance requirements outlined above, while ensuring high realism and fidelity of the sound produced. Given an object's geometry and a few material parameters, we construct a spring-mass model approximating the object's surface. We show that although a spring-mass system is a coarser approximation than FEM models used in prior approaches [Chaigne and Doutaut 1997; O'Brien et al. 2001; O'Brien et al. 2002], it is an adequate model to capture the small-scale surface vibrations that lead to the generation of sound in nature. We show how this formulation yields an analytical solution to the equation of motion for the surface of the object.

However, a naive implementation of such an approach can handle only a few (less than ten) sounding objects in real time. We also present several acceleration techniques. The increased computational efficiency is achieved by exploiting auditory perception, which ensures that the resulting degradation in perceived quality is minimal. In addition, the sound quality and the associated computational cost for each object is scaled dynamically in a priority-based scheme which guarantees that the total sound production meets

*nikunj@cs.unc.edu
†lin@cs.unc.edu

stringent time constraints, while preserving the overall aural experience. Our approach has the following characteristics:

- It is based on a discretized physically-based representation that offers simplicity of formulation and ease of implementation;
- It makes no assumptions about the input mesh topology – surface meshes used for physics can be used directly for sound synthesis;
- It is capable of yielding both impact and rolling sounds naturally, without any special-case treatment;
- It enables rich environments consisting of numerous sounding objects, with insignificant difference in the overall audio quality.

We also use OpenAL and EAX™ to provide hardware-accelerated propagation modeling of the synthesized sounds on Creative Sound Blaster Audigy 2™ audio cards which easily produce spatial and environmental sound effects such as distance attenuation and room acoustics. To the best of our knowledge, with the possible exception of methods that rely on physical measurements, no prior work has been demonstrated to handle complex scenarios (e.g. see Figs. 1 and 7) in real time.

Organization: The rest of the paper is organized as follows. We review related work in Section 2. We present the mathematical formulation developed to model the surface vibrations for sound synthesis in Section 3 and describe various acceleration techniques to enable real-time sound generation for a large-scale environment consisting of hundreds of sounding objects in Section 4. In Section 5, we discuss implementation issues and demonstrate the results of our system on complex scenes. Finally, we conclude with possible future research directions.

2 Previous Work

The concept of modeling the surface vibrations of objects using discretized physical models in real time was first proposed by Florens and Cadoz [1991], who used a system of masses and damped springs to model 3D shapes and developed the CORDIS-ANIMA system for physically-based sound synthesis. More recently, numerical integration with a finite element approach was proposed as a more accurate technique for modeling vibrations [Chaigne and Doutaut 1997; O'Brien et al. 2001]. These methods had the advantage that the simulation parameters corresponded directly to physically measurable quantities and the results were more accurate. The main drawback was the complexity of formulation and implementation and the low speed of the resulting simulation.

To remedy the performance issue of the above methods, van den Doel and Pai suggested [1996; 1998] using the analytically computed vibrational modes of an object, instead of numerical integration, leading to considerable speedups and enabling real-time sound synthesis. But, since the PDEs governing the vibration of arbitrary shapes are very complicated, the proposed system could only handle simple systems, such as plates, for which the analytical solutions were known. To handle more complex systems which do not admit direct analytical solution, two approaches have been proposed in literature. The first approach, [van den Doel et al. 2001] uses physical measurements on a given shape to determine its vibration modes and their dependence on the point of impact. Later, these modes may be appropriately mixed in a real-time application to generate realistic synthetic sounds. But, arbitrary 3D models have to be physically procured, in order to find their aural properties. In [2002], O'Brien et al. address this problem and propose a method for handling arbitrarily-shaped objects by discretizing them into tetrahedral volume elements. They show that the corresponding finite element equations can be solved analytically after suitable approximations. Consequently, they are able to model arbitrarily shaped objects and simulate realistic sounds for a few objects at interactive rates.

Our work shares some common themes with [O'Brien et al. 2002]. However, we propose a simpler system of point-masses and damped springs for modeling the surface vibrations of the object and it also submits to an analytical solution in a similar fashion, while offering much greater simplicity of formulation and ease of implementation. Furthermore, the complexity of scenes demonstrated in [O'Brien et al. 2002] is low, containing less than 10 sounding objects and the interactions captured are mainly due to impacts. As we will demonstrate in Section 5, our method extends to handling hundreds of objects in real time and is also capable of producing realistic *rolling* sounds in addition to impact sounds.

Often, immersive environments are both visually and aurally complex. The problem of scheduling multiple objects for sound synthesis was first addressed in [Fouad et al. 1997]. They exploited a model of imprecise computation proposed previously in [Chung et al. 1987], and proposed a system in which the objects are iteratively assigned time quotas depending on the availability of resources and priorities of the objects. As described in Section 4, our approach to prioritization and time-quota assignment exploits properties specific to our sound-generation technique, and thus achieves better results using a much simpler scheme. Recently, van den Doel et al. [2004] proposed techniques to synthesize sounds in real time for scenes with a few sounding rigid bodies and numerous particles, by exploiting frequency masking. At runtime, they find emitted frequencies which are masked out by other neighboring frequencies with higher amplitude and do not mix the masked frequencies. We use a different perceptual observation presented in [Sek and Moore 1995], which report that humans are incapable of distinguishing frequencies that are very close to each other. As we will discuss in Section 4, this can be used to prune out frequencies from an object's frequency spectrum as a *pre-processing* step. Our technique leads to better performance and much lesser memory consumption at runtime while ensuring minimal loss in auditory quality.

3 Methodology

Sound is produced by surface vibrations of an elastic object under an external impulse. These vibrations disturb the surrounding medium to result in a pressure wave which travels outwards from the object. If the frequency of this pressure wave is within the range 20 to 22000 Hz, it is sensed by the ear to give us the subjective perception of sound. The most accurate method for modeling these surface vibrations is to directly apply classical mechanics to the problem, while treating the object as a continuous (as opposed to discrete) entity. This results in PDEs for which analytical solutions are not known for arbitrary shapes. Thus, the only avenue left is to make suitable discrete approximations of the problem to reduce the PDEs to ODEs, which are more amenable to analysis. In this section, we show how a spring-mass system corresponding to a physical object may be constructed to model its surface deformation and how it may be analyzed to extract the object's modes of vibration. For ease of illustration, we assume a homogeneous object; inhomogeneous objects may be handled by a simple extension of the approach presented here. Further, we assume that the input object is in the form of a thin shell and is hollow inside. This assumption is motivated by practical concerns since most of the geometry today is modeled for rendering and is invariably only a surface representation with no guarantees on surface connectivity. If a volumetric model is available, the approach outlined in this paper applies with minor modifications. Figure 2 gives an overview of our approach.

3.1 Input Processing

Given an input mesh consisting of vertices and edges, we construct an equivalent spring-mass system by replacing the mesh vertices with point masses and the edges with damped springs. We now

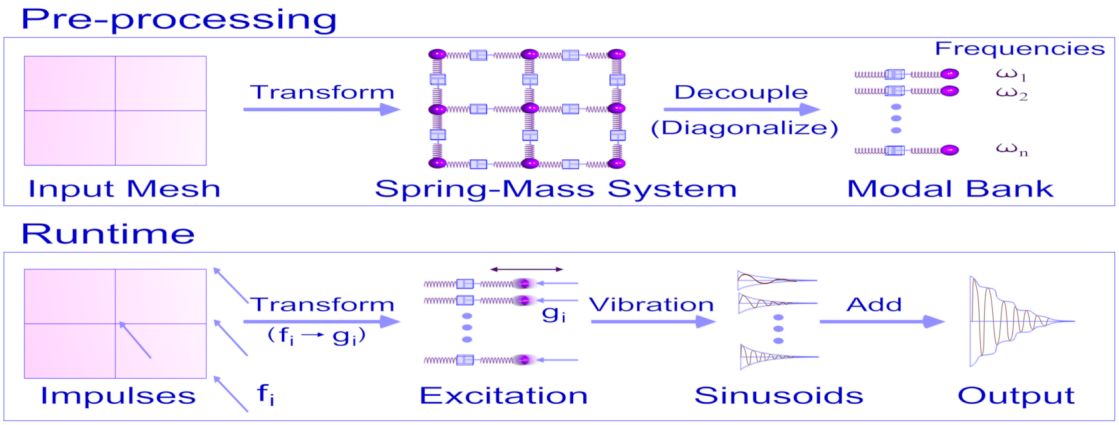

Figure 2: This diagram gives an overview of our approach. In the pre-processing step, each input surface mesh is converted to a spring-mass system by replacing the mesh vertices with point masses and the edges with springs, and the force matrices are diagonalized to yield its characteristic mode frequencies and damping parameters. At runtime, the rigid-body simulator reports the force impulses f_i on a collision event. These are transformed into the mode gains, g_i with which the corresponding modes are excited. These yield damped sinusoids which are suitably combined to yield the output sound signal.

discuss how to assign the spring constants and masses based on the material properties of the object so that the discrete system closely approximates the physical object. The spring constant, k and the particle masses, m_i are given by:

$$\begin{aligned} k &= Yt \\ m_i &= \rho t a_i \end{aligned} \tag{1}$$

where Y is the Young's Modulus of elasticity for the material, t is the thickness of the object surface, ρ is the material density and a_i is the area "covered" by a particle, which is calculated by dividing the area of each mesh face equally amongst all its constituent vertices and summing all the face contributions for the vertex corresponding to the mass in consideration. Note that we did not discuss fixing the spring damping parameters above, which we will return to shortly.

3.2 Deformation Modeling

Once the particle system has been constructed as above, we need to solve its equation of motion in order to generate the corresponding sound. Unfortunately, the resulting system of equations is still mathematically complex because the interaction forces between the masses are non-linear in their positions. However, by making the reasonable assumption that the deformation is small and linearizing about the rest positions, this problem can be cast in the form of a coupled linear system of ODEs:

$$M\frac{d^2r}{dt^2} + (\gamma M + \eta K)\frac{dr}{dt} + Kr = f \tag{2}$$

where M is the mass matrix, K is the elastic force matrix, γ and η are the fluid and viscoelastic damping constants for the material respectively. The matrix M is diagonal with entries on the diagonal corresponding to the particle masses, m_i. The elastic force matrix K is real symmetric, with entries relating two particles if and only if they are connected by a spring. The variable r is the displacement vector of the particles with respect to their rest position and f is the force vector. Intuitively, the terms in the above equation correspond to inertia, damping, elasticity and external force respectively. The specific form of damping used above, which expresses the overall damping matrix as a linear combination of K and M is known as Raleigh damping and works well in practice. For a system with N particles in three dimensional space, the dimensions of all the matrices above is $3N \times 3N$.

This formulation of the problem is well known and is similar to the one presented in [O'Brien et al. 2002]. The main difference in our approach is that the force and inertia matrices are assembled from a spring-mass system which makes the formulation much simpler. The solution to Equation (2) can be obtained by diagonalizing K so that:

$$K = GDG^{-1} \tag{3}$$

where G is a real matrix consisting of the eigenvectors of K and D is a diagonal matrix containing the eigenvalues. For reasons we will explain later, we will henceforth call G the "gain matrix". Plugging the above expression for K into Equation (2) and multiplying by G^{-1} throughout, we obtain:

$$G^{-1}M\frac{d^2r}{dt^2} + \left(\gamma G^{-1}M + \eta DG^{-1}\right)\frac{dr}{dt} + DG^{-1}r = f \tag{4}$$

Observing that since M is diagonal, $G^{-1}M = MG^{-1}$ and defining $z = G^{-1}r$ equation(4) may be expressed as:

$$M\frac{d^2z}{dt^2} + (\gamma M + \eta D)\frac{dz}{dt} + Dz = G^{-1}f \tag{5}$$

Since both M and D in the above equation are diagonal, Equation (2) has been decoupled into a set of unrelated differential equations in the variables z_i, which correspond to individual modes of vibration of the object. The equation for each mode is the standard equation of a damped oscillator and has the following solution for the i'th mode:

$$\begin{aligned} z_i(t) &= c_i e^{\omega_i^+ t} + \overline{c_i} e^{\omega_i^- t} \\ \omega_i^{\pm} &= \frac{-(\gamma\lambda_i + \eta) \pm \sqrt{(\gamma\lambda_i + \eta)^2 - 4\lambda_i}}{2} \end{aligned} \tag{6}$$

where the constant c_i, called the gain for the mode, is found by considering the impulses applied as we will discuss shortly. We use $\overline{c_i}$ to denote the complex conjugate of c_i. The constant λ_i is the i'th eigenvalue in the diagonal matrix, D. The real part of $\omega_i^{\pm}$ gives the damping coefficient for the mode, while the imaginary part, if any, gives the angular frequency of the mode.

3.3 Handling Impulsive Forces

Once an input mesh has been processed as above and the corresponding modes extracted as outlined in Equations (2)-(6), we have all the information needed regarding the aural properties of the object. The sound produced by an object is governed by the magnitude and location of impulsive force on its surface. We model

short-duration impulsive contacts by dirac-delta functions. Given an impulse vector f containing the impulses applied to each vertex of an object, we compute the transformed impulse, $g = G^{-1}f$ in order to evaluate the right-hand side of Equation (5). Once this is done, the equation for the i'th mode is given by:

$$m_i \frac{d^2 z_i}{dt^2} + (\gamma m_i + \eta \lambda_i) \frac{dz_i}{dt} + \lambda_i z_i = g_i \delta(t - t_0) \qquad (7)$$

where t_0 is the time of collision and $\delta()$ is the dirac delta function. Integrating the above equation from a time just before t_0 to a time just after t_0 and noting that $\int_{t_0^-}^{t_0^+} \delta(t - t_0) dt = 1$, we obtain:

$$m_i \Delta \left(\frac{dz_i}{dt} \right) + (\gamma m_i + \eta \lambda_i) \Delta z_i + z_i \Delta t = g_i \qquad (8)$$

Assuming that Δt is very small, and using the fact that the deformation is small compared to the change in velocities, we can neglect the last two terms on the left-hand side to obtain:

$$\Delta \left(\frac{dz_i}{dt} \right) = \frac{g_i}{m_i} \qquad (9)$$

The above gives a very simple rule which relates the change in the time derivative of the mode to the transformed impulse. Referring to Equation (6) and requiring that z_i should stay the same just before and after the collision while $\frac{dz_i}{dt}$ should increase as in Equation (9), the update rule for the mode gain c_i can be shown to be:

$$c_i \leftarrow c_i + \frac{g_i}{m_i \left(\omega_i^+ - \omega_i^- \right) e^{\omega_i^+ t_0}} \ . \qquad (10)$$

Initially, c_i is set to 0 for all modes.

4 Real-time Sound Synthesis

In this section, we describe how the mathematical formulation presented in the previous section is utilized to efficiently generate sound in real time. First, we describe a naive implementation and then discuss techniques to increase its efficiency.

Assume that there exists a rigid-body simulator which can handle all the dynamics. During a collision event, the sound system is informed of the object that undergoes the impact and the magnitude and location of impact. This impact is processed as described in Section 3.3 to result in the gains for the different modes of vibration of the object, where the gain for the i'th mode being c_i. The equation for a mode from the time of collision onwards is given by (6). The amplitude contribution of a mode at any moment is proportional[1] to its *velocity* (and not position). This is because the pressure contribution of a particle is determined by its velocity and the mode velocities are linearly related to the physical velocities of the particles. The mode velocity is found by taking a differential of Equation (6) with respect to time:

$$v_i = \frac{dz_i}{dt} = c_i \omega_i^+ e^{\omega_i^+ t} + \overline{c_i} \omega_i^- e^{\omega_i^- t} \qquad (11)$$

For generating each audio sample, we need to evaluate the above equation for all vibration modes of the object, which is quite inefficient. As mentioned in [O'Brien et al. 2002], the simple observation that $e^{i\omega(t+\Delta t)} = e^{i\omega t} e^{i\omega \Delta t}$ offers some gain in performance since generating a new audio sample just requires a single complex multiply with the previous value. However, the efficiency is still not sufficient to handle a large number of objects in real time. We

[1]The constant of proportionality is determined based on the geometry of the object and takes the fact into account that vibrations in the direction of the surface normal contribute more to the resulting pressure wave than vibrations perpendicular to the normal. We do not describe it in detail here as it is not critical to the approach being presented.

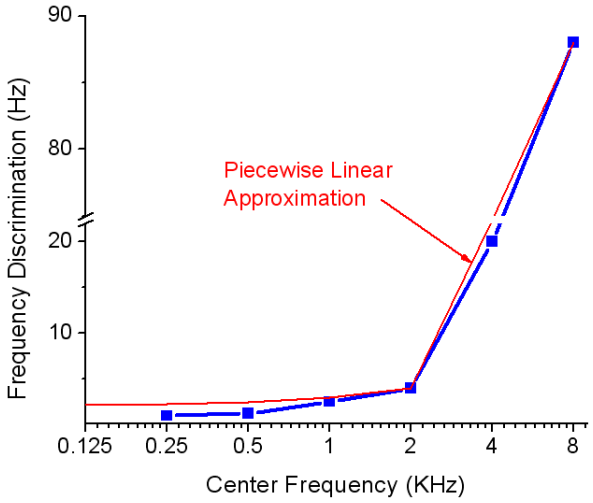

Figure 3: This plot shows frequency discrimination in humans as a function of the center frequency. Note that the human capacity to discriminate between frequencies degrades considerably for frequencies in the range 2-22 KHz, which forms a bulk of the human audible range. We use this fact to guarantee that no more than 1000 modes need to be mixed for any object in the worst case, *irrespective* of its geometric complexity. In most cases the actual number is much smaller, in the range of a few hundreds. The red curve shows the piecewise linear approximation of this curve that we use.

may estimate the running time of the system as follows: A simple spring-mass system with N particles has $3N$ modes, and the above operation needs to be repeated for each mode for each audio sample. Assuming a sampling rate of 44100 Hz, the number of floating-point operations (FLOPS) needed for this calculation for generating audio samples worth t seconds is:

$$T = 3N \times 4 \times 44100t \ FLOPS \ . \qquad (12)$$

Considering that the typical value of N is about 5000 or higher, producing sound worth 1 second would take 2646 MFLOPS. Since today's fastest processors operate at a few thousand MFLOPS [Dongarra 2005], the above processing would take about a second. Given that this estimated amount of time is for just one object and a typical scene may contain many such objects, such an approach is clearly not fast enough for interactive applications. Furthermore, for many real-time environments such as games and virtual environments, only a very small fraction of the actual time can be allocated for sound production. Thus, in the rest of this section, we will discuss techniques to increase the efficiency of the proposed base system to enhance its capability in handling scenarios with a large number of sounding objects at interactive rates.

From Equation (12), it is clear that the running time is proportional to the number of modes being mixed and the number of objects. Next, we present acceleration techniques for sound simulation by reducing the number of modes per object: "Mode Compression" and "Mode Truncation", and by scaling the audio quality of different objects dynamically with little degradation in perceived sound quality.

4.1 Mode Compression

Humans have a limited range of frequency perception, ranging from 20 to 22000 Hz. It immediately follows that modes with frequencies lying outside this range can be clipped out and need not be mixed. However, there is another important fact which can lead to large reductions in the number of modes to be mixed. A perceptual study described in [Sek and Moore 1995] shows that humans have a limited capacity to discriminate between nearby frequen-

cies. Note that this is different from frequency masking [Zwicker and Fastl 1990] in which one of two *simultaneously* played frequencies masks out the other. Rather, this result reports that even if two "close enough" frequencies are played in succession, the listener is unable to tell whether they were two different frequencies or the same frequency played out twice. The authors call the length of the interval of frequencies around a center frequency which sound the same, the "Difference Limens to Change" (DLC). Figure 3 shows a plot of the DLC against center frequencies ranging from .25 to 8 KHz. Interestingly, the DLC shows a large variation over the audible frequency range, getting very large as the center frequency goes beyond 2 KHz. Even at 2 KHz, the DLC is more than 1 Hz. That is, a human subject cannot tell apart 1999 Hz from 2000 Hz.

We use the above fact to drastically reduce the number of modes that are mixed for an object. We linearly approximate the DLC curve with a piecewise linear curve shown as the red line in Figure 3. The approximation has two segments: one from 20 Hz to 2 KHz and another from 2 KHz to 22 KHz. As we show in the figure we overestimate the DLC slightly. This increases the performance further and we have observed minimal loss in quality in all the cases we have tested. The main idea behind our compression scheme is to group together all the modes with perceptually indistinguishable frequencies. It can be easily shown that if the above mentioned linear approximation to the DLC curve is used and indistinguishable modes clustered at the corresponding frequency centers, the maximum number of modes that need to be mixed is less than 1000. It is important to note that this is just the worst case scenario and it happens only when the frequency spectrum of the object consists of all frequencies from 20 to 22,000 Hz, which is very rare. For most objects, the frequency spectrum is discrete and consequently, the number of modes after mode compression is much smaller than 1000, typically in the range of a few hundreds.

We now describe the details of our technique. Recall the gain matrix from Equation (3), G. The gain matrix has a very simple physical interpretation: Rows of the matrix correspond to vertices of the object and columns correspond to the different modes of vibration (with their corresponding frequencies). Each row of G lists the gains for the various modes of vibration of the object, when a unit impulse is applied on the corresponding vertex. It is clear from the above discussion that all the mode gains within a row of G which correspond to modes with close frequencies need to be clustered together. This is achieved by replacing the gain entries for all such modes by a single entry with gain equal to the sum of the constituent gains. Since a mode corresponds to a whole column, this reduces to summing together columns element-wise based on their frequencies. The complete procedure is as follows:

- Sort the columns of G with the corresponding mode frequencies as the key. [2]
- Traverse the modes in increasing order of frequency. Estimate the DLC, Δ at the current frequency using the piecewise linear curve shown in Figure 3. If the current frequency and next frequency are within Δ of each other the two mode frequencies are indistinguishable, replace the two columns by their element-wise sum.

Below, we enumerate the main advantages of this scheme:

1. The running time is constant instead of linear in the number of vertices in the object. For example, if the input mesh is complex with 5,000 vertices, the number of modes mixed is bounded by 1000 instead of the earlier $3N = 15,000$ which is a substantial performance gain.

[2]This step is usually not needed as most linear algebra packages output the eigenvector matrix sorted on the eigenvalues

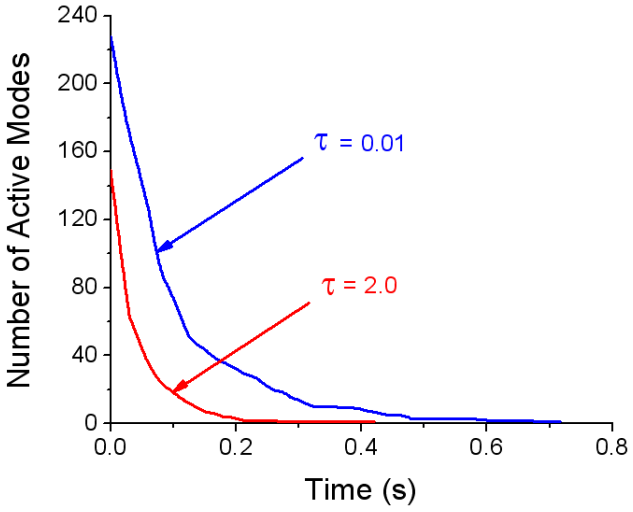

Figure 4: This graph shows the number of modes mixed vs time, for a xylophone bar just after it is struck in the middle. τ is the mode truncation threshold. A higher value of τ leads to more aggressive truncation of modes with low amplitude, leading to savings in terms of the number of modes mixed. In this case, $\tau = 2.0$ results in about 30% gain in efficiency over $\tau = 0.01$ which only truncates modes with near-zero amplitude. The sound quality for both the cases is nearly identical.

2. Since this scheme requires just the frequencies of the different modes, the whole processing can be done as a pre-process without requiring any extra runtime CPU cycles.

3. From the above mentioned procedure, it is clear that the number of columns in the matrix G, which is the same as the number of modes, is now bounded by 1000 instead of the earlier value of $3N$. Since this matrix needs to be present in memory at runtime for transforming impulses to mode gains, its memory consumption is an important issue. Using this technique, for an object with 5000 vertices, the memory requirement has been reduced from 225 MB to less than 15 MB, by more than a factor of 15.

4. Most objects have a discrete frequency spectrum with possibly many degenerate frequencies. Due to numerical inaccuracies while diagonalizing the elastic force matrix and the approximations introduced by the spring-mass discretization, these degenerate frequencies may appear as spurious distinct modes with near-equal frequencies. Obviously, it is wasteful to treat these as distinct modes. It is our observation that most of the times these modes' frequencies are close enough so that they are naturally summed together in this scheme.

4.2 Mode Truncation

The sound of a typical object on being struck consists of a transient response composed of a blend of high frequencies, followed by a set of lower frequencies with low amplitude. The transient attack is essential to the quality of sound as it is perceived as the characteristic "timbre" of the object. The idea behind mode truncation is to stop mixing a mode as soon as its contribution to the total sound falls below a certain preset threshold, τ. Since mode truncation preserves the initial transient response of the object when τ is suitably set, the resulting degradation in quality is minimal. Figure 4 shows a plot of the number of active modes with respect to time for a xylophone bar struck in the middle for two different values of τ: .01 and 2. These values are normalized with respect to the maximum sample value which is 65536 for 16-bit audio. The first case with

$\tau = .01$ performs essentially no truncation, only deactivating those modes which have near-zero amplitude. Note that with $\tau = 2$ the number of modes mixed is reduced by more than 30%. Also, the number of active modes floors off much earlier (.2 secs compared to .6 secs). It is important to note that this results in little perceptible loss in quality.

The details of the technique are as follows: Assume that an object has just undergone a collision and the resulting mode gains c_i have been calculated as given by Equation (10). From this time onwards until the object undergoes another collision, Equation (11) gives a closed-form expression for the mode's contribution to the sound of the object. This can be used to predict exactly when the mode's contribution drops below the threshold τ. The required "cutoff time", t_i^c is such that for all times $t > t_i^c$:

$$c_i \omega_i^+ e^{\omega_i^+ t} + \overline{c_i} \omega_i^- e^{\omega_i^- t} < \tau \tag{13}$$

Using the fact that for any two complex numbers x and y, $|x + y| \leq |x| + |y|$ it can be shown that,

$$t_i^c \leq \frac{1}{-Re(\omega_i^+)} \ln \left(\frac{2\,|c_i|\,|\omega_i^+|}{\tau} \right) \tag{14}$$

Using the above inequality, the cutoff times are calculated for all the modes just after a collision happens. While generating the sound samples from a mode, only one floating point comparison is needed to test if the current time exceeds the cutoff time for the mode. In case it does, the mode's contribution lies below τ and consequently, it is not evaluated.

4.3 Quality Scaling

The two techniques discussed above are aimed at increasing the efficiency of sound synthesis for a single object. However, when the number of sounding objects in a scene grows beyond a few tens, this approach is not efficient enough to work in real time and it is not possible to output the sound for all the objects at the highest quality. It is critical in most interactive applications that the sound system have a graceful way of varying quality in response to variable time constraints. We achieve this flexibility by scaling the sound quality for the objects. The sound quality of an object is changed by controlling the number of modes being mixed for synthesizing its sound. In most cases of scenes with many sounding objects, the user's attention is on the objects in the "foreground", that is, the objects which contribute the most to the total sound in terms of amplitude. Therefore, if it is ensured that the foreground sounds are mixed at high quality while the background sounds are mixed at a relatively lower quality, the resulting degradation in perceived aural quality should be reduced.

We use a simple scheme to ensure higher quality for the foreground sounds. At the end of each video frame, we store the sum of the vibrational amplitudes of all modes for each object, which serve to determine the object's priority. At the next video frame, all objects are sorted in decreasing order based on their priority and the total time-quota for sound-generation divided among the objects as a linearly decreasing ramp with a preset slope, S. After this, all objects are processed in their priority order. For each object, its quality is first scaled so that it can finish within its assigned time-quota and then the required modes are mixed for the given time period. If an object finishes before its time-quota has expired, the surplus is consumed greedily by the next higher priority object. The slope, S of the ramp decides the degree to which the foreground sound quality is favored over a degradation in background sound quality. The case with $S = 0$ corresponds to no priority scheduling at all, with the time-quota being divided equally among all objects. The converse case with $S = \infty$ corresponds to greedy consumption of the time-quota. That is, the whole time-quota is assigned to the highest priority object. After the object is done, the remaining time, if any, is assigned to the next highest priority object and so on.

4.4 Putting Everything Together

To illustrate how all the techniques described above are integrated, we present a summary of our approach.

Pre-processing

- Construct a spring-mass system corresponding to the input mesh. (Section 3.1)
- Process the spring-mass system to extract the gain matrix, G and the (complex) angular frequencies of the object's modes of vibration: ω_i^+ and ω_i^-. (Section 3.2, Eqns. (3) and (6))
- **Mode Compression:**
  Aggregate columns of G based on frequencies of the corresponding modes, as described in Section 4.1.
- Store the resulting gain matrix along with the (complex) constants ω_i^+ and ω_i^- for modes correspond to the columns of G after compression. Note that ω_i^- need not be stored in case ω_i^+ has a non-zero imaginary part since in that case $\omega_i^- = \overline{\omega_i^+}$.

Runtime Processing

- Load the gain matrix and mode data for each object.
- Begin simulation loop:

  1. Run rigid-body simulation
  2. For each object, O:
     - **Collision Handling:**
       If the rigid-body simulator reports that O undergoes a collision event, update its gain coefficients as per Equation (10) using the collision impulse and its location. (Section 3.3)
     - **Mode Truncation:**
       Compute cutoff times t_j^c for each mode based on the mode truncation threshold, τ. (Section 4.2, Equation (14))
  3. **Quality Scaling:**
     Sort objects based on amplitude contribution, assign time-quotas and compute the number of modes to be mixed for each object. (Section 4.3)
  4. **Sound Synthesis:**
     For each timestep at time t and for each object, O:
     - Consider all modes permitted by the current quality setting which satisfy $t_j^c > t$. Sample and summate all these modes as described at the beginning of this section. This is O's contribution to the sound.
     - Output the sum of all objects' sample contribution as the sound sample for time t.

  End simulation loop

5 Implementation and Results

In this section we present results to demonstrate the efficiency and realism achievable with our approach.

5.1 Rigid Body Simulation

We have implemented the algorithm and acceleration techniques presented in this paper using C++ and OpenGL. Our rigid-body simulator extends the technique presented by Guendelman et al. [2003] to incorporate DEEP [Kim et al. 2002] for fast and more accurate penetration depth estimation, instead of sample-based estimation using distance fields. It also uses a more complex friction model presented by Mirtich and Canny [Mirtich and Canny 1995], which results in more robust contact resolution.

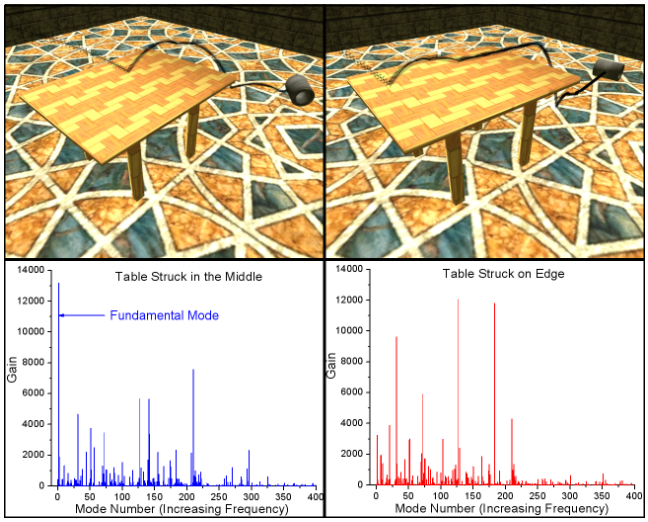

Figure 5: A metallic cylinder falls onto a wooden table, in the middle (left) and on the edge (right) and rolls off. The bottom part shows the corresponding frequency spectra for the two cases. Note that for the case on the left, most of the impulse is transferred to the low frequency fundamental mode while for the case on the right, the impulse is mostly transferred to higher frequency modes.

5.2 Position Dependent Sounds

As discussed earlier, the main advantage of using physically-based sounds over recorded audio is the ability to capture effects such as the magnitude of impacts between objects and more importantly, the subtle shift in sounds on striking an object at different points. Figure 5 shows a scene with a metallic cylinder tossed onto a wooden table. Both the table and cylinder are sounding. The figure contrasts two cases: the first case, shown on the left, depicts the cylinder striking the table near the middle and rolling off, while in the second case it strikes the table near the edge. We discuss the rolling sound in the next subsection, and will discuss the impact sound here. Since the table-top is in the form of a plate, we would expect that striking it on the edge would transfer a larger fraction of the impulse to higher frequencies, while striking it in the middle should transfer most part of the impulse to the fundamental mode of vibration, leading to a deeper sound. To verify this, we plotted the frequency spectra for the two cases just after the cylinder makes first impact with the table. The corresponding plots for the two cases are shown in the lower part of the figure. The case on the left shows a marked peak near the fundamental mode while the peak is completely missing in the second case. Conversely, the second case shows many peaks at higher frequencies which are missing in the first one. This difference clearly demonstrates that the sound for the two cases is markedly different, with the same qualitative characteristics as expected. Another important point to note is that this technique does not require the meshes to be highly tessellated to capture these effects. The table consists of just 600 vertices and the cylinder 128 vertices.

5.3 Rolling Sounds

In addition to handling impact sounds, we are able to simulate realistic rolling sounds without requiring any special-case treatment for sound synthesis. This is in part made possible because of the rigid-body simulator we have developed, which is able to handle contacts in a more graceful manner than most impulse-based simulators. Figure 6 shows the impulses on the cylinder and the corresponding audio for the case shown in the right side of Figure 5. The cylinder rolls on the table after impact, falls to the ground and rolls on the floor for sometime. The initial rolling sound, when the cylinder is on the table, has a much richer quality. The sound of the

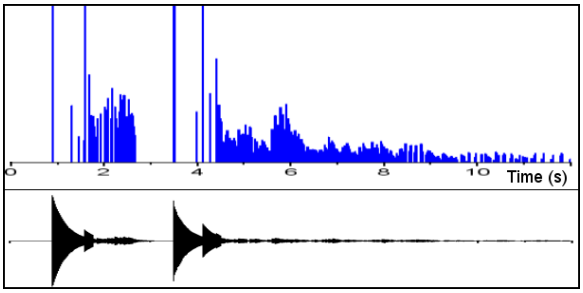

Figure 6: A plot of the impulses on a cylinder versus time for the scene shown on the right in Figure 5 and the corresponding audio samples. The peaks correspond to impacts while the numerous low-amplitude impulses correspond to rolling forces.

table as the cylinder rolls over it conveys a sense of the cylinder's heaviness, which is only partly conveyed by the sound of the impact. The cylinder, although uniformly tessellated, is very coarse, with only 32 circumferential subdivisions. Figure 6 shows the impulses applied on the cylinder against time. The peaks correspond to impacts: when the cylinder falls on the table, and when it falls to the ground from the table. Note that the audio waveform shows the corresponding peaks correctly. The period of time stretching from 6 to 8 seconds consists of the cylinder rolling on the floor and is characterized by many closely-spaced small-magnitude impulses on the cylinder as it strikes the floor again and again due to its tessellation. To test how important the periodicity of these impulses was for the realism of rolling sounds, we found the mean and standard deviation of the interval between these impulses from the data presented in Figure 6. The mean time between the impulses was 17 ms with a standard deviation of 10 ms. The fact that the standard deviation is more than 50% of the mean demonstrates that the impulses show very little periodicity. This suggests that the periodicity of collisions is not critical for the perceived realism of rolling sounds.

5.4 Efficiency

We are able to do audio simulation for complex scenes in real time using our approach. Figure 7 shows a scene with 100 metallic rings falling simultaneously onto a wooden table and undergoing elastic collision. All the rings and the table are sounding. Each ring is treated as a separate object with separate aural properties. The rings consist of 200 vertices each. Figure 8 shows the audio FPS[3] for this simulation against time for the one second interval during which almost *all* the collisions take place. The application frame rate is 100 FPS. Note that this is not the raw data but a moving average so that the short-range fluctuations are absorbed. The plot on the bottom is the base timing without using any of the acceleration techniques described in Section 4. The audio in this case is very choppy since the audio generation is not able to keep up with the speed of rendering and rigid-body simulation. With mode truncation and mode compression, the performance shows significant improvement. However, after initially starting at about 200 FPS, the frame rate drops in the latter part where the maximum number of collisions happen. With quality scaling in addition to mode compression and truncation (shown by the top-most curve), the frame rate exhibits no such drop, continuing to be around 200 FPS. This is because quality scaling gives priority to sound generation for those rings which just underwent collision, while lowering the quality for other rings which may have collided earlier and are contributing less to the overall sound. This illustrates the importance of quality scaling for scenarios with multiple collisions. It is important to note that although this example sufficiently demonstrates the capability

[3] An audio frame is defined as the amount of sound data sufficient to play for a duration equal to the duration of one video frame.

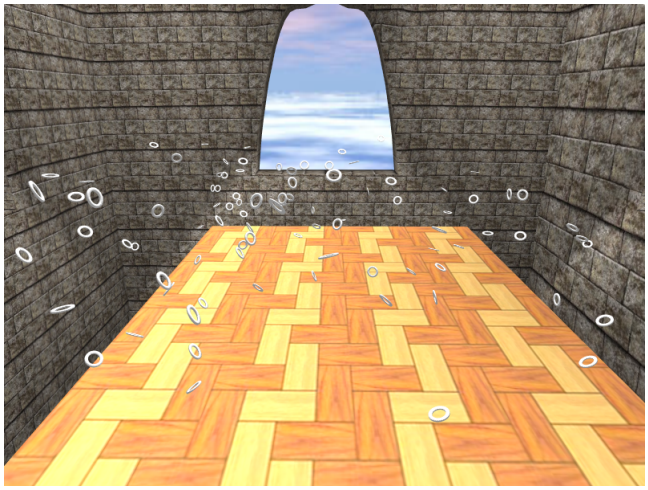

Figure 7: More than 100 metallic rings fall onto a wooden table. All the rings and the table are sounding. The audio simulation runs at more than 200 FPS, the application frame rate being 100 FPS. Quality Scaling ensures that the perceived sound quality does not degrade, while ensuring steady frame rates (See Figure 8)

of the system to maintain steady frame rates, it is improbable in a real application, since there are about 100 collisions within a second. This is the reason why the CPU utilization is high (50%). A more common scenario would be as shown in Figure 1, which has a much lower CPU utilization (10%).

To illustrate the realistic sounds achievable with our approach, we designed a three-octave xylophone shown in Figure 1. The image shows many dice falling onto the keys of the xylophone to produce the corresponding musical notes. The audio simulation for this scene runs in the range of 500-700 FPS, depending on the frequency of collisions. The dice have been scripted to fall onto the xylophone keys at precise moments in time to play out any set of musical notes. Because of the efficiency of the sound generation process, the overall system is easily able to maintain a steady frame rate of 100 FPS. Also, there are situations in which many dice fall on different keys within a few milliseconds of each other, but the sound quality exhibits no perceptible degradation. Although we have not tuned the xylophone keys to match the exact frequency spectrum of a real xylophone, the resulting sound is realistic and captures the essential timbre of the instrument. The material parameters for the xylophone were taken from [Chaigne and Doutaut 1997].

6 Conclusion

We have presented a physically-based sound synthesis algorithm with several acceleration techniques for rendering a large-scale scene consisting of hundreds of interacting objects in real time, with little loss in perceived sound quality. This approach requires no special mesh structure, is simple to implement, and further takes advantage of existing hardware acceleration. We plan to extend this framework to auditory display of sliding sounds, explosion noises, breaking sounds, and other more complex audio effects that are difficult to simulate at interactive rates.

References

CHAIGNE, A., AND DOUTAUT, V. 1997. Numerical simulations of xylophones. i. time domain modeling of the vibrating bars. *J. Acoust. Soc. Am. 101*, 1, 539–557.

CHUNG, J. Y., LIU, J., AND LIN, K. J. 1987. Scheduling real-time, periodic jobs using imprecise results. In *Proc. IEEE RTS*.

DONGARRA, J. J. 2005. Performance of various computers using standard linear equations software (linpack benchmark report). Tech. rep., Knoxville, TN, USA.

FLORENS, J. L., AND CADOZ, C. 1991. The physical model: modeling and simulating the instrumental universe. In *Represenations of Musical Signals*, G. D. Poli, A. Piccialli, and C. Roads, Eds. MIT Press, Cambridge, MA, USA, 227–268.

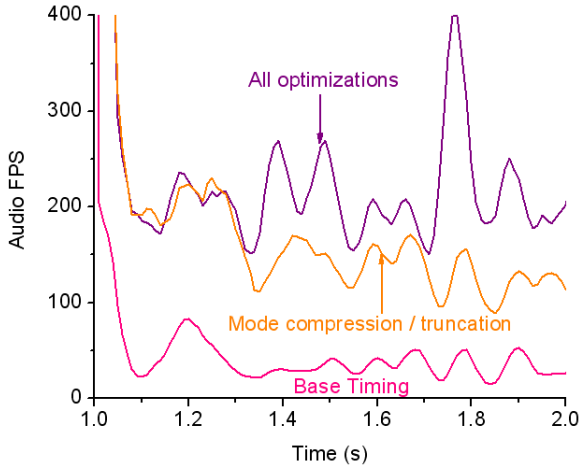

Figure 8: This graph shows the audio simulation FPS for the scene shown in Figure 7 from time 1s to 2s, during which almost all the collisions take place. The bottom-most plot shows the FPS for an implementation using none of the acceleration techniques. The top-most curve shows the FPS with mode compression, mode truncation and quality scaling. Note how the FPS stays near 200 even when the other two curves dip due to numerous collisions during 1.5-2.0s.

FOUAD, H., BALLAS, J., AND HAHN, J. 1997. Perceptually based scheduling algorithms for real-time synthesis of complex sonic environments. In *Proc. Int. Conf. Auditory Display*.

GUENDELMAN, E., BRIDSON, R., AND FEDKIW, R. 2003. Nonconvex rigid bodies with stacking. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH) 22*, 871–878.

KIM, Y. J., LIN, M. C., AND MANOCHA, D. 2002. DEEP: an incremental algorithm for penetration depth computation between convex polytopes. *Proc. of IEEE Conference on Robotics and Automation*, 921–926.

MIRTICH, B., AND CANNY, J. 1995. Impulse-based simulation of rigid bodies. In *1995 Symposium on Interactive 3D Graphics*, P. Hanrahan and J. Winget, Eds., ACM SIGGRAPH, 181–188. ISBN 0-89791-736-7.

O'BRIEN, J. F., COOK, P. R., AND ESSL, G. 2001. Synthesizing sounds from physically based motion. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 529–536.

O'BRIEN, J. F., SHEN, C., AND GATCHALIAN, C. M. 2002. Synthesizing sounds from rigid-body simulations. In *The ACM SIGGRAPH 2002 Symposium on Computer Animation*, ACM Press, 175–181.

SEK, A., AND MOORE, B. C. 1995. Frequency discrimination as a function of frequency, measured in several ways. *J. Acoust. Soc. Am. 97*, 4 (April), 2479–2486.

VAN DEN DOEL, K., AND PAI, D. K. 1996. Synthesis of shape dependent sounds with physical modeling. In *Proceedings of the International Conference on Auditory Displays*.

VAN DEN DOEL, K., AND PAI, D. K. 1998. The sounds of physical shapes. *Presence 7*, 4, 382–395.

VAN DEN DOEL, K., KRY, P. G., AND PAI, D. K. 2001. Foleyautomatic: physically-based sound effects for interactive simulation and animation. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 537–544.

VAN DEN DOEL, K., KNOTT, D., AND PAI, D. K. 2004. Interactive simulation of complex audiovisual scenes. *Presence: Teleoper. Virtual Environ. 13*, 1, 99–111.

ZWICKER, E., AND FASTL, H. 1990. In *Psychoacoustics*. Springer-Verlag, Berlin.