Physically Informed Sonic Modeling (PhISM): Synthesis of Percussive Sounds

Author(s): Perry R. Cook

Source: *Computer Music Journal*, Autumn, 1997, Vol. 21, No. 3 (Autumn, 1997), pp. 38-49

Published by: The MIT Press

Stable URL: https://www.jstor.org/stable/3681012

Perry R. Cook
Department of Computer Science and
Department of Music
Princeton University
35 Olden Street
Princeton, New Jersey 08544, USA
PRC@cs.princeton.edu

Physically Informed Sonic Modeling (PhISM): Synthesis of Percussive Sounds

Introduction: Percussion Instruments and Physical Modeling

Percussion instruments represent a broader spectrum of physical configurations than any other instrument family. Orchestration books, acoustics books, and taxonomies of instruments often lump some instruments into the percussion family not because of what they are, but just because they don't fit into the other instrument families. One loose characterization might state that percussion instruments exhibit exponentially decaying modes that are excited by striking. This results in the piano's being categorized as a percussion instrument, but this definition does nothing to address the many percussion instruments that can be shaken or rubbed continuously, nor those that don't exhibit any clear modal behaviors. The physical components of percussion instruments include bars, plates, membranes, cavity and tube resonators, and nonlinearities of countless types, all coupled to each other in varieties of ways. Because of this variety, there exist no common model features that allow the percussion family to be captured with simple changes in model topology, or with variations of the parameters of a meta-model. By grouping the percussion family into subfamilies, some similarities can be exploited, but there are still large varieties of configurations and excitation means. Another important aspect of percussion instruments is their relationship to everyday (nonmusical) sounds. Walking, preparing food, working with metal and wood tools, riding a bicycle, etc., all create sounds that are closely related to members of the percussion family.

The expressive nature of percussion lies in the ability to excite objects, with various objects, and in numerous ways. Sampling synthesis of percussion instruments based on digital recordings lacks the means for manipulating parameters in musical and physically meaningful ways. There has been some work on more-efficient physical modeling of membranes (Van Duyne and Smith 1993), and modeling of nonlinearities in percussion instruments (Van Duyne, Pierce, and Smith 1994). However, direct real-time synthesis of many percussion instruments by physical modeling is likely to remain beyond practical computing means for some time to come. Even if computing power were sufficient to make exhaustive physical modeling of percussion instruments economical, the physical mechanisms of many such systems are still not completely understood.

This article focuses on natural and expressive synthesis of many sounds in the percussion family, and describes a set of methods for systematically analyzing and parametrically synthesizing many percussion instruments. Some of this work was first presented at the 1996 International Computer Music Conference in Hong Kong (Cook 1996), but this article represents a significant expansion, elaboration, and refinement of that original project. It also includes detailed code examples and tables of parameters for synthesizing a wide variety of percussion sounds. The analysis/synthesis approach described here, called Physically Informed Sonic Modeling (PhISM), draws upon techniques found in research in physical modeling, Fourier analysis/synthesis, wavelet analysis/synthesis, and granular synthesis. Motivated by two distinct families of percussion instruments, two types of synthesis algorithm will be presented here. The first is Physically Informed Spectral Additive Modeling (PhISAM), which models systems with distinct modal oscillatory behaviors. The second is Physically Informed

Computer Music Journal, 21:3, pp. 38–49, Fall 1997

Stochastic Event Modeling (PhISEM), which models systems in which sound is produced by random collisions of many objects. Both families allow for scalable modeling of the instrument and excitation physics. One of many possible synthesis techniques might be employed for synthesizing the actual sounds, such as amplitude-modulated oscillator synthesis, frequency modulation, stochastically parameterized sampling (granular) synthesis, modal synthesis, or physical modeling. The principal motivation is to provide expressive control of synthesis, using whatever final synthesis algorithm is most suitable. The term "physically informed" implies that the instrument physics are taken into account in selecting the model, in deriving parameters for the selected model, and possibly in modifying certain synthesis parameter values in real time to allow expressive control of the sound.

PhISAM: Physically Informed Control of Modal Synthesis

The PhISAM technique is based on modal synthesis, but could be realized with sinusoidal oscillators as well. The modal-filter or oscillator-control parameters are driven and controlled by rules derived from predetermined Fourier boundary methods, and/or from analysis data extracted from recorded sounds. The main goal of PhISAM is that of spectral synthesis, but with controls and parameters that have physical meaning. Figure 1 shows the PhISAM analysis/resynthesis process in block-diagram form. This system is not a closed-form process that guarantees analysis/resynthesis identity; rather, it relies heavily on human analysis and decisions in all stages. The PhISAM approach is suitable for resonant percussion instruments, such as marimba, xylophone, vibraphone, cowbell, agogo bell, and other instruments characterized by impulsive excitation of a relatively few exponentially decaying, weakly coupled sinusoidal modes. In the domain of nonmusical percussive sounds, the PhISAM approach is capable of modeling exponentially decaying sounds exhibiting a few resonances, like many struck wood or metal objects.

Doutaut and Chaigne (1993) synthesized sound directly from finite-difference solutions of the physics of bar percussion instruments such as the xylophone; Serra (1986) proposed synthesis of the marimba and vibraphone using sinusoidal modeling with added residual noise; and Wawrzynek (1989) proposed a multiple-filter model of the marimba with impulsive and noise excitation. While PhISAM affords the same flexible control over the modal parameters described by Mr. Serra and Mr. Wawrzynek, it also provides expressive yet simple control over excitation parameters such as stick hardness, and important performance parameters such as strike position and strike vigor. Knowledge derived from physical modeling allows additional modeling of the performer to be coupled into a still-efficient synthesis algorithm, allowing for implementation of occasional accidental multiple strikes and missed strikes, as well as context-dependent re-excitation of the resonators.

Modes of vibration of percussion instruments can be derived from acoustical theory (Moore 1970), by modal analyses of physical systems (Rossing 1976), by peak-picking from spectra calculated by Fourier analysis (Smith and Serra 1987), adaptive filter analysis such as Linear Predictive Coding or LPC (Serra 1986), or other all-pole-filter modeling techniques (Larouche and Meillier 1994). Spectrum-analysis methods are in many ways most desirable, because new systems can be treated rapidly with signal-processing techniques applied to recordings. Theory can be developed as needed or convenient to add heuristics about the behavior of modes. By using a high-order filter analysis, the modes of significance can be detected, and weak modes can be either discarded or included as part of the excitation. If one uses an adaptive technique such as LPC or Autoregressive Moving-Average (ARMA) modeling, or a heuristically guided Fourier-peak-picking algorithm based on the continuity of sinusoidal "tracks," it is often advantageous to perform the analysis on the time-reversed sound, progressing in the analyzed signal from the more-pitched decay portion to the nonpitched attack.

A prototype excitation can be constructed by recording excitations in a nonresonant instrument

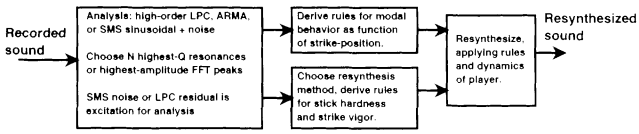


Figure 1. The PhISAM analysis/synthesis block diagram.

```
void Marimba::setStrikePos(float pos)
{                        /* Set position from 0.0 to 1.0 */
     float temp;
     temp = sin(1.0 * pos * PI);
     this->setFiltGain(0,0.12 * temp); /* 1st mode freq.*/
     temp = sin(0.03 + 3.9 * pos * PI);
     this->setFiltGain(1,0.02 * temp); /* 2nd mode freq.*/
     temp = sin(0.05 + 9.3 * pos * PI);
     this->setFiltGain(2,0.03 * temp); /* 3rd mode freq.*/
                              /* Leave 4th mode alone.  */
}
```

Figure 2. The C++ code to compute marimba strike position. Position is 0.0 and 1.0 at each end of the bar and 0.5 at the center, and it affects the excitation level of the first three modes. Leaving the fourth mode alone helps to avoid a "sampler phenomenon," where all modes in the spectrum track exactly in amplitude (and pitch). The sampler phenomenon is further avoided by applying some randomness on the frequencies of the modes, by implementing one or more fixed-frequency modes, or by augmenting the rule base of the modal behaviors as a function of pitch and strike position.

condition. This could be accomplished by damping the bar of a vibraphone and recording an actual stick strike, or recording a stick strike on some rigid, nonresonant surface. A prototype excitation can also be derived by extracting a residual resulting from adaptive filter analysis such as LPC, or Fourier-based deterministic/stochastic decomposition (Serra and Smith 1990). Once derived, the prototype excitation can then be manipulated in the time domain (filtering, stretching by interpolation, etc.) and/or frequency domain, to yield a flexible control space for excitation wave functions. A derived family of parametrically controlled excitation pulses can be related to (and realized by) wavelet analysis/synthesis, with a stochastic component representing the noisy character of percussive excitations. The process of identifying and segregating resonances from excitation can be a completely automatic process, but in the author's experience, some amount of guidance by the human hand and ear can be very helpful to the analysis software.

To simulate strike location, simple rules can be derived to control the relative levels of each mode in the modal synthesis, based on the one-dimensional spatial vibration patterns of the objects. Strike position data can also be inferred from modal analysis, or Fourier analysis of actual strikes in multiple positions. Some randomness should also be included in the resynthesis rule process. The C++ code example in Figure 2 demonstrates a simple but effective mapping of strike position to modal parameters in a marimba.

As suggested by work in modeling stringed-instrument players and performance styles (Garton 1992; Jánosy, Karjalainen, and Välimäki 1994), performance realism can be added to mallet percussion by including the physics of the percussion mallets and the performer. The resynthesis model can be built and implemented in a modular fashion, using an expert/player/instrument structure as I have described elsewhere (Cook 1995). A simple implementation of a marimba player object is shown in the C++ code example in Figure 3. This code automatically computes strike position based on the last notes played, the time since those notes were played, the skill level of the player, and simple physical models of the player's arms and hands. Figure 4 depicts a simple two-dimensional geometric layout that could be used to determine the strike positions for four mallets.

Figure 5 shows a PhISAM modal synthesis algorithm block diagram. Note that if the filters are implemented in parallel, and the individual mode gains are applied to the filter inputs, continuity is maintained in modes that may still be ringing, and new excitations interact naturally with current energy in the system. To match some resonant percussion instruments exhibiting weak nonlinearities (like some drums, such as bongos), resonant filters can exhibit time-varying parameters, but at increased computational cost. Figure 6 shows an actual marimba spectrogram, the analyzed residual after removing the modes, and the resynthesized result.

PhISEM: Particle Models

The PhISEM (Physically Informed Stochastic Event Modeling) algorithm is based on pseudorandom overlapping and adding of small grains of sound, or

Figure 3. The C++ code showing how player dynamics are taken into account in the MarmPlyr object. Successive notes that are far apart in space and close together in time are more likely to be missed. Skill level, a number from 0.0 to 1.0, affects accuracy of notes and velocity. In a complete player object, difficult note combinations take longer to reach as well.

```
void MarmPlyr::noteOn(int noteNum, int velocity)
{
     int hand, note, vel, rndVel;
     float temp;

     /* Which hand plays?  How difficult is it to play?*/
     hand = this->whichHand(noteNum);
     temp = this->howHard(noteNum, hand);
     if (skill > temp)     /*  If it's too difficult, */
                           /*  miss it by +/- 1 note. */
          note = noteNum + intRand(-1,1);

     /* Compute strike position and set it up. */
     temp = this->strikeWhere(hand,note);
     Marimba->setStrikePos(temp);

     /*   Miss target velocity by an amount depending */
     /*   on player skill, keeping velocity legal.    */
     temp = (1.0 - skill);
     rndVel = int (temp * velocity);
     rndVel = intRand(-rndVel,rndVel);
     vel = velocity + rndVel;
     if (vel > 127) vel = 127;
     if (vel < 0) vel = 0;

     /* Remember note played so we can use it later  */
     /* to calculate difficulty.  */
     this->stashNote(note);
     Marimba->noteOn(note,vel):  /* Play the note  */

}
```

Figure 3

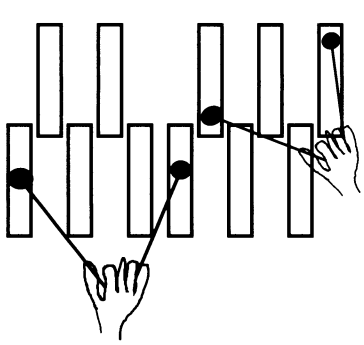


Figure 4

Figure 4. A depiction of the simple geometric setup for calculating strike position for four percussion mallets on a virtual marimba. The physics of the player constrain strike position and accuracy.

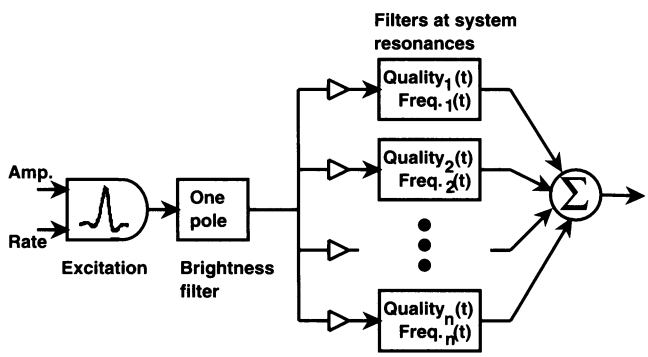


Figure 5. The basic PhISAM modal-synthesis block diagram. The one-pole filter allows a simple control for brightness, and the parallel topology with filter gains on inputs allows for strike-position-dependent re-excitation while the resonant filters are still ringing.

Figure 5

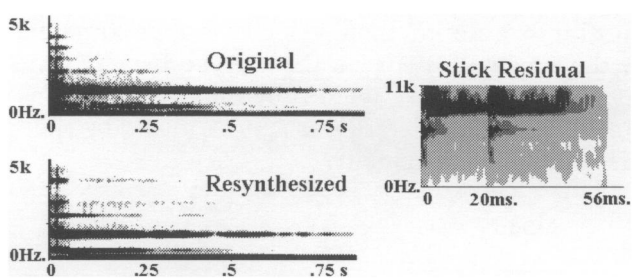


Figure 6. Spectrograms of the original (upper left), residual (right), and resynthesized (lower left) marimba tones. After removing resonances, it appears that the analyzed stick strike was a double hit. This allows the residual to be trimmed to the single-strike sound. Multiple strikes can then be generated as needed from a 20-msec stored sound.

Figure 6

pseudorandom modification of the parameters of a parametric synthesis model, according to rules and parameters derived from off-line physical simulations and heuristics. The PhISEM algorithm is suitable for instruments such as a police/referee whistle, a maraca, and a tambourine, as well as other systems that are characterized by random interactions of sound-producing component objects. In the nonmusical realm, PhISEM is suitable for many sounds, such as bamboo wind chimes, feet crunching on gravel, ice cubes in a shaken glass, and water drops.

The common element in instruments modeled by the PhISEM algorithm is that sound is generated

Figure 7. The C++ code showing the exchange of energy, with a slight loss, between two colliding particles in three dimensions.

```
for (i=0;i<NUM_BEANS;i++)  {
                    /* Loop on number of particles, */
     for ( j = i + 1; j < NUM_BEANS; j ++)  {
                    /* and over remaining particles.*/
          if (collide(beans[i], beans[j])  {
                    /* If collision, exchange       */
                    /* velocities in each direction */
                    /*  with some loss.             */
               temp1 = beans[i]->getXVelocity();
               temp2 = beans[j]->getXVelocity();
               beans[i]->setXVelocity(temp2 * B_LOSS);
               beans[j]->setXVelocity(temp1 * B_LOSS);
               temp1 = beans[i]->getYVelocity();
               temp2 = beans[j]->getYVelocity();
               beans[i]->setYVelocity(temp2 * B_LOSS);
               beans[j]->setYVelocity(temp1 * B_LOSS);
               temp1 = beans[i]->getZVelocity();
               temp2 = beans[j]->getZVelocity();
               beans[i]->setZVelocity(temp2 * B_LOSS);
               beans[j]->setZVelocity(temp1 * B_LOSS);
          }
     }
}
```

and/or modified by characteristic discrete events, such as the pea inside a police whistle crossing the airway entrance, or a bean striking the inside surface of a maraca gourd. Such systems can be simulated—and discrete events synthesized—directly in real time, or they can be simulated off-line for collecting statistics that change as a function of the excitation and system parameters. The information collected from the off-line simulation can be used to control a parametric synthesis model that is more efficient than the direct system simulation.

At the heart of PhISEM algorithms are particle models, characterized by basic Newtonian equations governing the motion and collisions of point masses, as found in any introductory physics textbook. Particle models are the basis of the CORDIS-ANIMA system from the Association pour la Création et Recherche dans les Outils d'Expression (ACROE), as described by Cadoz, Luciani, and Florens (1993). Particle systems can be solved numerically (Gould and Tobochnik 1996) using simple forward or backward Euler difference methods. In the simple Euler method, velocity is approximated by the difference between the current discrete time position and the last, divided by the time step Δt. Similarly, the acceleration is approximated by the first difference of velocity.

$$\text{velocity} = v(t) = \frac{\partial x(t)}{\partial t} \approx \frac{x(t) - x(t - \Delta t)}{\Delta t}$$

$$\text{acceleration} = a(t) = \frac{\partial v(t)}{\partial t} \approx \frac{x(t) - 2x(t - \Delta t) + x(t - 2\Delta t)}{\Delta t^2}$$

By selecting a time step that is suitably small, appropriate accuracy and stability can be achieved. In two and three Cartesian dimensions, these difference equations become separable multidimensional vectors in position, velocity, and acceleration. Runge-Kutta methods can be used for more accuracy at larger time steps. When collisions occur, momentum and center of mass are used for additional solution constraints. The C++ code example in Figure 7 shows a collision between two point masses (of equal mass) in three dimensions, and the code in Figure 8 shows a collision between a point mass and a rigid circular wall in two dimensions. For actual calculation, the masses are assigned radii, and if the distance between any two masses is less than the sum of their radii, a collision has occurred. Similarly, if the location of a mass is greater than the radius of the containing sphere, minus the radius of the mass, a collision with the spherical wall has occurred.

A Simple Model in 2-D: The Police Whistle

As a first example, a simple two-dimensional single particle model can be derived for a police/referee whistle. Figure 9 shows a side view of the whistle, with a single particle "pea" suspended in a breath-controlled vector field. The region around the fipple where oscillation takes place is characterized by a random vector field, reflecting turbulence and vortices. It would be entirely possible to connect the particle model parameters to a physical model of the air-jet fluid dynamics (Verge 1995; Chafe 1995), but the emphasis of Physically Informed Sonic Modeling is to use the simplest synthesis models possible.

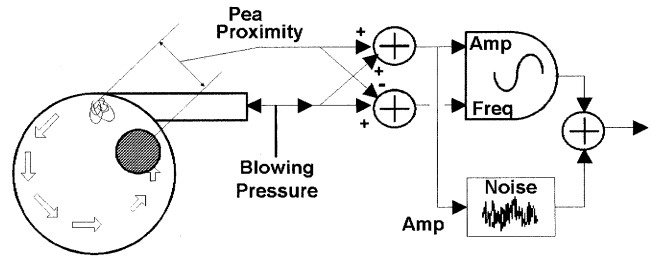
Experiments and spectrograms using a real police

Figure 8. The C++ code for implementing the collision of a single particle with the wall of a circular shell, in two dimensions. There are more direct ways to calculate this, but this form makes it obvious how a general case might be treated. Only the velocity component normal to the wall is reversed.

```
for (i=0; i<NUM_BEANS; i++)  {
          /*  Loop on number of particles. */
     if (collide(beans[i],shell))  {
          /*  If collision, get particle velocity.*/
          tempXVel = beans[i]->getXVelocity();
          tempYVel = beans[i]->->getYVelocity();
          /*  Get position on shell wall.          */
          tempX = beans[i]->getX();
          tempY = beans[i]->getY();
          /*  Compute angle to position.           */
          phi = -atan2(tempY,tempX);
          cosphi = cos(phi);
          sinphi = sin(phi);
               /*  "Rotate" shell and particle     */
               /*  so that all particle velocity   */
               /*  is directed in the positive     */
               /*  x direction.  Why?              */
          temp1 = (cosphi * tempXVel) -
                    (sinphi * tempYVel);
          temp2 = (sinphi * tempXVel) +
                    (cosphi * tempYVel);
               /*  Then we just reverse x vel,     */
               /*  then rotate things back again   */
               /*  and proceed with simulation.    */
           temp1 = -temp1 * SHELL_LOSS;
           tempXVel = (cosphi * temp1) +
                     (sinphi * temp2);
           tempYVel = (-sinphi * temp1) +
                     (cosphi * temp2);
           beans[i]->setVelocity(tempXVel,tempYVel);
     }
}
```

whistle show that when the pea is in the immediate region of the jet oscillator, there is a decrease in pitch (about 7 percent), an increase in amplitude (about 6 dB), and a small increase in the noise component (about 2 dB). The noise component is about 20 dB down from the first harmonic of the oscillator, and the oscillator exhibits three significant harmonics at f_0, $2f_0$, and $3f_0$, at 0 dB, −10 dB, and −25 dB, respectively. Center frequencies of the fundamental frequencies of six test whistles ranged from 1,800–3,200 Hz. These observed effects are easily coupled from the parameters of a physical particle simulation to the control inputs of a simple oscillator/noise synthesis model, as shown in Figure 9.

Figure 9. Physical control of the oscillator/noise synthesis model. The vector flow field is shown pushing the pea into turbulence at the entrance. The pea's proximity to the fipple modulates the oscillator's amplitude and frequency, as well as the noise component's amplitude.



This allows the entire model to run easily in real time under user-controlled breath pressure.

3-D Modeling: Characterizing Statistical Behavior

The particle model can be extended to multiple-particle systems in three dimensions, such as the maraca or other shaker instruments. Calculating these models in real time may not be practical with current computing equipment, but off-line simulations using particle systems can be employed for calculating time-varying statistical distributions, which can then be used in real time to control a parametric synthesis algorithm.

A spherical maraca model was constructed in C++, and it was virtually "shaken" with various numbers of beans. Statistics were collected of the frequency, waiting time, and sound intensity of collisions of beans with the wall of the spherical gourd. Waiting time was defined as the number of samples between sound-producing collisions. Only beans hitting the outside shell of the maraca were considered significant sound-producing collisions, since bean-bean collisions inside the gourd do not couple efficiently to the radiated sound. The square of the velocity component normal to the maraca surface was used to calculate the strength of the radiated sound, and corresponding energy was taken out of the colliding bean's velocity at the time of its collision with the maraca wall. Bean-bean collisions also caused a small decrease in the energy of both beans. As might be expected, once energy was put into the system, the average intensity of collisions decayed exponentially. The likelihood of a significant sound-producing collision occurring was found to be roughly constant, except when all beans were nearly at rest. This points to a Poisson

Figure 10. The smoothed amplitude responses from 10 shakes of simulated maracas containing 5, 10, 20, and 40 beans.

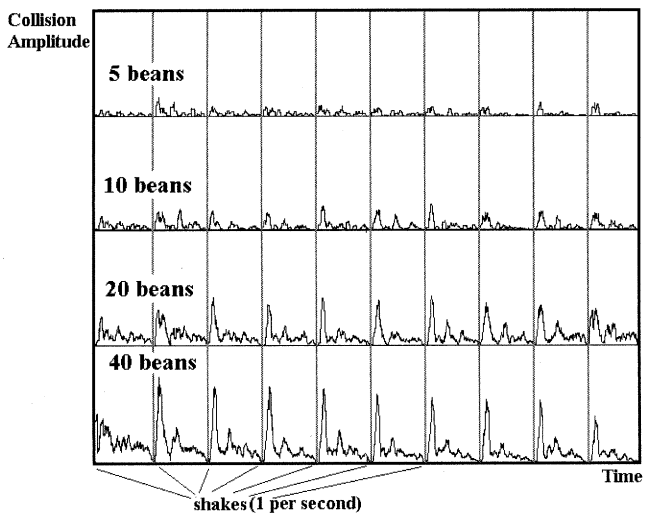
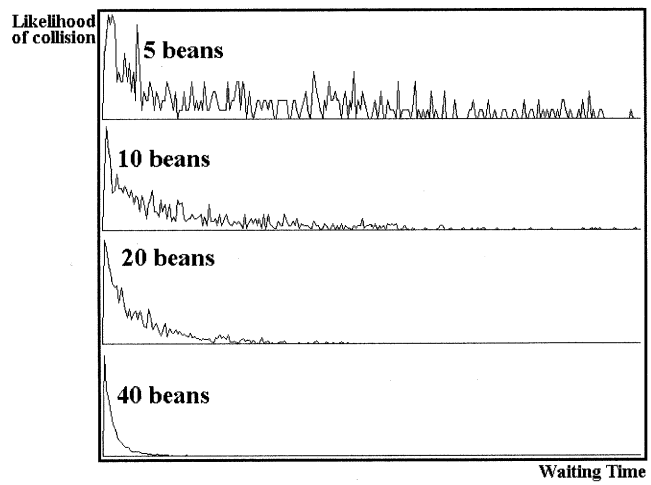


Figure 11. The averaged histograms of waiting times between sound-producing events for simulated maracas containing 5, 10, 20, and 40 beans.

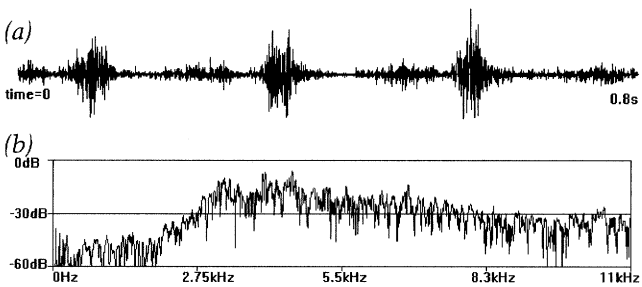


process, in which a constant probability of an event at any given time gives rise to an exponential probability of waiting times between events. Figure 10 shows smoothed amplitude responses of simulations that were calculated using 5, 10, 20, and 40 beans. The oscillation superimposed on the exponential decay is due to the initial bouncing of the beans together in an ensemble, which eventually breaks up into more random behavior. Figure 11 shows histograms of waiting times averaged over 10 shakes, for simulations calculated using 5, 10, 20, and 40 beans. It is clear that the Poisson process becomes more like the ideal for larger numbers of beans.

Once statistics are collected, parametric random number distributions that model the simulated distributions can be derived, and used to control the overlapping and adding of a number of single-collision sounds. These sounds could be acquired by recording single beans dropping onto an actual maraca, or synthesized using a parametric algorithm. Note that a single bean hitting a gourd that exhibits a single dominant resonance will generate a decaying sinusoidal oscillation much like a formant wave function or FOF (Rodet 1984), and one might be tempted to investigate this technique for synthesis of the maraca. Further, the notion of randomly added short sounds is similar to the asynchronous "clouds" of the granular synthesis algorithms described by Roads (1991). The potential number of randomly timed overlapped collisions to be calculated and managed, however, suggests investigation of a more economical synthesis algorithm. Figure 12 shows a recorded maraca wave and spectrum. Note the resonant nature of the gourd filter, and the noise-like nature of the sound. This noise-like characteristic motivates an efficient synthesis model.

For this project, an efficient maraca synthesis model was developed using rapidly decaying noise for the individual collision events, and a single biquad filter for modeling the resonance of the maraca gourd. If the envelope of a single collision sound is assumed to be an exponential decay, the model is greatly simplified. Since the sum of exponentially decaying random noises is equal to a single noise source multiplied by a decaying value, only a single exponential decay and a single noise source are required to compute the total sound. The sound envelope is increased additively by each new collision, and collisions are determined by the Poisson process calculation. A slower exponential decay models net system energy, and determines how much energy is added with each collision. One could use a linear function, or some other monotonically decreasing function, for the sound and system-energy decay, but the exponential decay requires only one multiply and no comparisons to calculate, and lends itself to efficient DSP and modern host-processor implementations. The code ex-

Figure 12. Three shakes of an actual maraca; time domain (a), and average spectrum (b).



ample in Figure 13 shows an entire C program that calculates a simple maraca synthesis.

To compute each audio sample, the PhISEM maraca synthesis algorithm requires only two random-number calculations (for collision-event likelihood and noise sound source), two exponential decays, and one biquad filter. The Poisson process and system exponential decay calculations could be carried out on a multisample basis (although not necessarily at the same granularity), yielding even less computation per sample. This extremely simple model is less complex than synthesis using interpolated pulse-code modulation (PCM) wavetables and filtering, and requires no memory for sample storage. The flexibility of real-time manipulation of control statistics, number of beans, and gourd resonance makes the model even more playable than a real maraca in an expressive sense, and makes the model reconfigurable to a large variety of shaker-type instruments. The assumptions of exponential decay of system energy and Poisson probability of sound-producing collisions are more valid with larger numbers of beans and increasing gourd size, but the model motivated by the simplifying observations was found to yield plausible synthesis even for small numbers of virtual objects, and for many other related instruments. The next section will motivate some simple additions to the basic model, and will list parameters for synthesizing a variety of instruments.

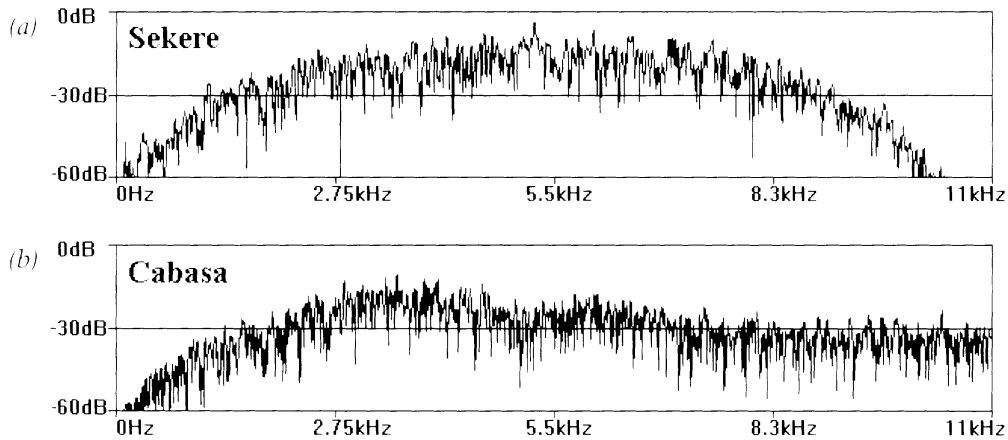
Extending PhISEM: Some Example Musical Instruments

A variety of maracas can be found in any music store, ranging from inexpensive wooden models to "professional" models manufactured entirely of

Figure 13. The C code for simple maraca synthesis.

```
#define SOUND_DECAY 0.95
#define SYSTEM_DECAY 0.999
#define SHELL_FREQ 3200.0
#define SHELL_RESO 0.96
#define TWO_PI 6.28318530718
#define SRATE 22050.0

void main(int argc, char *argv[]) {
     FILE *file_out;
     double temp = 0;
     double shakeEnergy = 0.0, sndLevel = 0.0, gain;
     double input = 0.0, output[2] = {0.0,0.0}, coeffs[2];
     long i, num_beans = 64;
     short data;
     extern double noise_tick();    /* -1.0 < this < 1.0*/

     gain = log(num_beans) / log(4.0) * 40.0 / num_beans;
     if (file_out = fopen(argv[1],"wb")"wb")) {
          /* Initialize gourd resonance filter.  */
          coeffs[0] = - SHELL_RESO * 2.0 *
                    cos(SHELL_FREQ * TWO_PI / SRATE);
          coeffs[1] = SHELL_RESO * SHELL_RESO;
          /*  The main loop begins here.  */
          for (i=0; i<100000; i++)       {
               if (temp<TWO_PI) {
                    /* Shake over 50 msec and   */
                    /* add shake energy.        */
                    temp += (TWO_PI / SRATE / 0.05);
                    shakeEnergy += (1.0 - cos(temp));
               }
               /* Shake 4 times/second.  */
               if (i % 5050 == 0) temp = 0;

               /* Compute exponential system decay.  */
               shakeEnergy *= SYSTEM_DECAY;

               if (random(1024) < num_beans)
                    sndLevel += gain * shakeEnergy;
                    /* If collision add energy.        */

               /* Actual sound is random.              */
               input = sndLevel * noise_tick();

               /* Compute exponential sound decay.*/
               sndLevel *= SOUND_DECAY;

               /* Do gourd resonance filter calc.  */
               input -= output[0]*coeffs[0];
               input -= output[1]*coeffs[1];
               output[1] = output[0];
               output[0] = input;

               /* Extra zero for spectral shape.  */
               data = output[0] - output[1];

               /* That's all! Write it out.   */
               fwrite(&data, 2, 1, file_out);
          }
     }
     fclose(file_out);
}
```

Figure 14. The spectra of two shaker-type instruments: the sekere (a), and the cabasa (b).



plastic parts. Three different instruments were investigated in this study; synthesis parameters for a typical instrument are given in Table 1. Other instruments similar to the maraca include the sekere and the cabasa. The sekere consists of a gourd with beans knitted into a netting and suspended over the outside of the gourd. This allows the grains to be manipulated more directly with the hands, and typical playing style includes rubbing while throwing the sekere from one hand to the other. The cabasa is a highly controllable, commercially manufactured shaker, consisting of a corrugated metal drum mounted on a handle, with rows of metal beads wrapped loosely around the drum. The combination of the handle and the symmetric drum allows the user to directly manipulate the beads. Both the sekere and the cabasa exhibit resonances that are less pronounced than that of the maraca. Figure 14 shows spectra of these shaker-type instruments. These instruments are easily modeled by the single-resonance basic PhISEM algorithm. Parameters are given in Table 1. Other shaker instruments in this family include various Native American and African "rattle drums."

A class of related gourd instruments are the scrapers, such as the guiro. In these instruments, the sound is caused not by random particles rattling around the gourd, but rather by the scraping of a stick or brush on serrations cut into the outside of the resonator. As shown by Figure 15, the time-domain nature of the excitation is quasi-periodic and pulsatile in nature, corresponding to the stick's brushing across the regular, serrated notches. The spectrum shows two principal weak resonances, which are easily modeled in the PhISEM algorithm with the addition of one more filter section. The system-energy component is managed more directly in the guiro simulation, by controlling the rate and amplitude of a sawtooth "scraping" function. Resonance parameters for the guiro are given in Table 1.

In contrast to the maraca and guiro-like gourd resonator instruments, which exhibit one or two weak resonances, instruments such as the tambourine (timbrel) and sleigh bells use metal cymbals or bells suspended on a frame or stick. Other related instruments are the African sistrum and the Turkish/Greek zilli masa. The interactions of the metal objects produce resonances that are much more pronounced than those of the maraca-type instruments, but the Poisson event and exponential system-energy statistics are similar enough to warrant use of the PhISEM algorithm for synthesis. The tambourine, as a relative of the hand drum, often has a head suspended over a round frame, as well as pairs of cymbals suspended in spaces in the frame. For this study, only tambourines without heads were investigated. The individual cymbals are small enough that only one or two partials are present in the spectrum. There is a random distribution of these partials (due to the difference in manufacture of the individual cymbals) around a center resonance, as shown in the tambourine spectrum of Figure 16. The sleighbells, as also shown in Fig-

Figure 15. The time-domain waveform (a) and the spectrum (b) of the guiro, a scraper instrument.

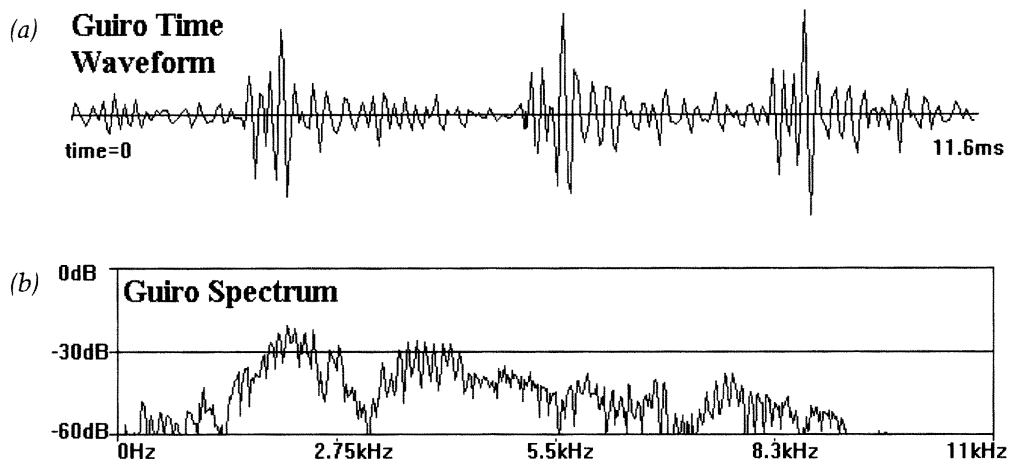
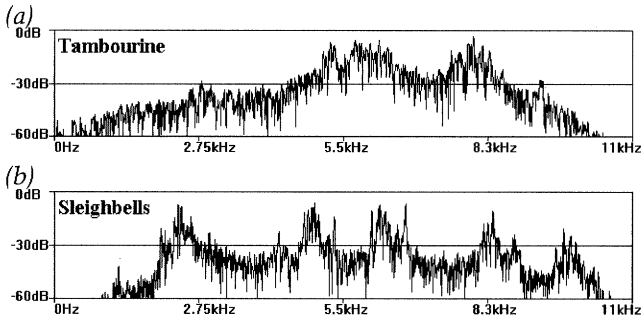


Table 1. Parameters for PhISEM percussion synthesis algorithms for various percussive sounds

	Maraca	*Sekere*	*Cabasa*	*Guiro*	*Tambourine*	*Sleighbells*	*Water Drops*	*Bamboo Wind Chimes*
Probability	1/(4–32)	1/16	1/2	1/8	1/32	1/32	1/8192	1/1024
System decay	0.999	0.999	0.997	—	0.9985	0.9994	0.999*	0.95
Sound decay	0.95	0.96	0.95	0.95	0.95	0.97	0.95	0.99995
Zeros?	D.C.	Both	D.C.	Both	Both	Both	Both	None
Number of resonances	1	1	1	2	3	5	3	3
Resonance frequencies	2–4.5k	5.5k	3k	2.5k 4k	2.3k% 5.6k% 8.1k%	2.5k% 5.3k% 6.5k% 8.3k% 9.8k%	450%* 600%* 750%*	2.2k% 2.8k% 3.4k%
Resonance poles	0.96	0.6	0.7	0.97 0.97	0.96 0.995 0.995	0.999 0.999 0.999 0.999 0.999	0.9985 0.9985 0.9985	0.995 0.995 0.995

Notes: All parameters are for a 22-kHz sampling rate. The zeros field indicated a zero at 0 Hz, at both 0 Hz and 1/2 of the sampling rate, or none. A percent symbol (%) indicates that frequencies are randomly selected at each sample, centered around the resonance frequencies. An asterisk (*) in the water-drop algorithm indicates that there is only one frequency reallocated per collision, and a separate exponential gain is initialized that corresponds to the allocated resonance.

Figure 16. The spectra of two shaker-type instruments: the tambourine (a) and sleighbells (b). Note the broad flat peaks at 5.7 kHz and 8.1 kHz in the tambourine, caused by a distribution of sharply resonant partials in the cymbals. Also note the weak wood-frame resonance at 2.75 kHz in the tambourine spectrum. The sleighbells exhibit more peaks, at 2.5, 5.0, 6.0, 8.3, and 9.8 kHz.



ure 16, exhibit many more partials, but still show a random distribution around a few main resonant peaks. To more closely match these types of spectra in PhISEM, more filters are used to model the individual partials, and at each collision, the resonant frequency of the filters are randomly set to frequencies around the main resonances. Efficient yet flexible tambourine synthesis is accomplished using two resonances for the cymbals, and one weak resonance for the wooden frame. The sleighbell requires five resonant filters. Parameters for the synthesis models are given in Table 1. It might also be possible to use a simple FM pair for modeling the inharmonic partials of the bells and cymbals.

PhISEM: Nonmusical Sounds

One exciting possibility for the PhISEM algorithm is the synthesis of nonmusical percussive sounds. Two such sounds will be described here: the sound of dropping water, and bamboo wind chimes. (Actually, the latter are also related to a musical instrument, the Javanese angklung.) Since these two sounds exhibit a distribution of weak resonances of similar frequency, they are easily modeled by relatively few resonant filters in the PhISEM algorithm. Parameters for these two instruments are listed in Table 1. The water-dropping algorithm requires three resonant filters, whose parameters vary in time to model the "boip" sound made by water drops. Only one filter is reallocated in frequency for each detected collision, and a separate exponential amplitude envelope is initialized for each drop. To extend the wooden wind chimes to metal, more resonant filters may be used, or the filters can be replaced with simple FM pairs in inharmonic frequency ratios. Other nonmusical sounds for which the PhISEM algorithm shows promise are footsteps on gravel, a pepper grinder, the sounds of eating, etc.

PhISM: MIDI and Real-Time Control

All of the instruments described in this article are rewarding and effective when manipulated in real time. Standard MIDI messages such as note-on and aftertouch can be used to add energy to the systems. The use of custom MIDI control numbers to control strike position and stick hardness makes the PhISAM marimba model very expressive to play. The MIDI breath pressure feeds directly into the PhISEM whistle model. Custom controllers with knobs, buttons, and accelerometers have been constructed and attached to the PhISEM shaker instruments, with excellent results. Latency and preemption problems in current operating systems make most high-level language implementations unconvincing for real-time percussion performance, but since the PhISM algorithms are so simple, dedicated microcontroller solutions are being investigated for creating a truly expressive family of instruments.

Conclusions

By exploiting knowledge about the physics of some percussion instrument families, but utilizing the simplest synthesis models that are suitable, sounds can be synthesized that are at once efficient and expressive. Parametric models that cover a large variety of instruments and sounds can be played in real time, and these models show much promise for performance, composition, and the generation and manipulation of sound effects.

References

Cadoz, C., A. Luciani, and J. L. Florens. 1993. "CORDIS-ANIMA: A Modeling and Simulation System for

Sound Image Synthesis—The General Formalization." *Computer Music Journal* 17(1):19–29.
Chafe, C. 1995. "Adding Vortex Noise to Wind Instrument Physical Models." *Proceedings of the 1995 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 57–60.
Cook, P. 1995. "A Hierarchical System for Controlling Synthesis by Physical Modeling." *Proceedings of the 1995 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 108–109.
Cook, P. 1996. "Physically Informed Sonic Modeling (PhISM): Percussion Instruments." *Proceedings of the 1996 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 228–231.
Doutaut, V., and A. Chaigne. 1993. "Time Domain Simulations of Xylophone Bars." *Proceedings of the Stockholm Music Acoustics Conference.* Stockholm: Royal Swedish Academy of Music, pp. 574–579.
Garton, B. 1992. "Virtual Performance Modeling." *Proceedings of the 1992 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 219–222.
Gould, H., and J. Tobochnik. 1996. *An Introduction to Computer Simulation Methods.* New York: Addison-Wesley.
Jánosy, Z., M. Karjalainen, and V. Välimäki. 1994. "Intelligent Synthesis Control with Applications to a Physical Model of the Acoustic Guitar." *Proceedings of the 1994 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 402–406.
Larouche, J., and J. Meillier. 1994. "Multichannel Excitation/Filter Modeling of Percussive Sounds with Application to the Piano." *IEEE Transactions on Speech and Audio Processing.* New York: 1EEE, pp. 329–344.
Moore, J. 1970. "Acoustics of Bar Percussion Instruments." PhD Dissertation, Ohio State University.
Roads, C. 1991. "Asynchronous Granular Synthesis." In G. De Poli, A. Piccialli, and C. Roads, eds. *Representations of Musical Signals.* Cambridge, Massachusetts: MIT Press, pp. 143–185.
Rodet, X. 1984. "Time-Domain Formant-Wave-Function Synthesis." *Computer Music Journal* 8(3):9–14.
Rossing, T. 1976. "Acoustics of Percussion Instruments—Part 1." *The Physics Teacher* 14:546–556.
Serra, X. 1986. "A Computer Model for Bar Percussion Instruments." *Proceedings of the 1986 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 257–262.
Serra, X., and J. Smith. 1990. "Spectral Modeling Synthesis: A Sound Analysis/Synthesis System Based on a Deterministic Plus Stochastic Decomposition." *Computer Music Journal* 14(4):12–24.
Smith, J., and X. Serra. 1987. "PARSHL: Analysis/Synthesis Program for Non-Harmonic Sounds Based on a Sinusoidal Representation." *Proceedings of the 1987 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 290–297.
Van Duyne, S., J. Pierce, and J. Smith. 1994. "Traveling Wave Implementation of a Lossless Mode-Coupling Filter and the Wave Digital Hammer." *Proceedings of the 1994 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 411–418.
Van Duyne, S., and J. Smith. 1993. "Physical Modeling with the 2-D Digital Waveguide Mesh." *Proceedings of the 1993 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 40–47.
Verge, M. 1995. "Aeroacoustics of Confined Jets, with Applications to the Physics of Recorder-Like Instruments." Thesis, Technical University of Eindhoven.
Wawrzynek, J. 1989. "VLSI Models for Sound Synthesis." In M. Mathews and J. Pierce, eds. *Current Directions in Computer Music Research.* Cambridge, Massachusetts: MIT Press.