The ITB Journal

Volume 3 | Issue 2 Article 11

2002

Extending Physical Simulation to the Audio Domain

Graham McCann

Hugh McCabe

Extending Physical Simulation To The Audio Domain

Graham Mccann & Hugh McCabe
School of Informatics and Engineering, Institute of Technology, Blanchardstown
Dublin 15, Ireland

Abstract

Using physical simulation to control movement and interactions between objects in a 3D environment, such as a computer game, has become increasingly common However; this idea has not been extended in a general sense to the audio domain. The audio component of these systems usually comprises of pre-recorded WAV files that are triggered in response to events. We argue that this approach has serious limitations and propose instead that the physical information made available by physics engines provides an opportunity to carry out synthesis of appropriate sounds from scratch. We outline a framework for adding a sound synthesis module to a physics engine and discuss some approaches to implementing this.

Introduction

Over the past few decades the games industry has made some startling advances in the various fields that constitute it, most notably in the areas of graphics and artificial intelligence (AI). With the ever increasing capabilities of modern computer hardware, particularly processors and graphical accelerators, it has become possible to do more complex calculations on the fly, hence the current booming popularity of physical realism in computer games. A number of companies have appeared in recent years developing real-time physical simulators that can be integrated with existing graphics engines to create a visually believable virtual world. However, in order to create a truly immersive experience you need to have both vision and sound (Preece, Rogers, et al., 1994). If the sounds you hear don't match what you see, then the illusion of realism will be ruined, no matter how impressive the visual aspect may be. With the inclusion of physics engines in many modern games, the foundation is already there for the addition of sound synthesis. The physics engine already calculates a considerable amount of data for its own purposes, and would usually discard any superfluous information once it has achieved the movements required. Instead of disposing of the intermediate physical calculations we can use the data to assist in the synthesis of an accompanying effects soundtrack.

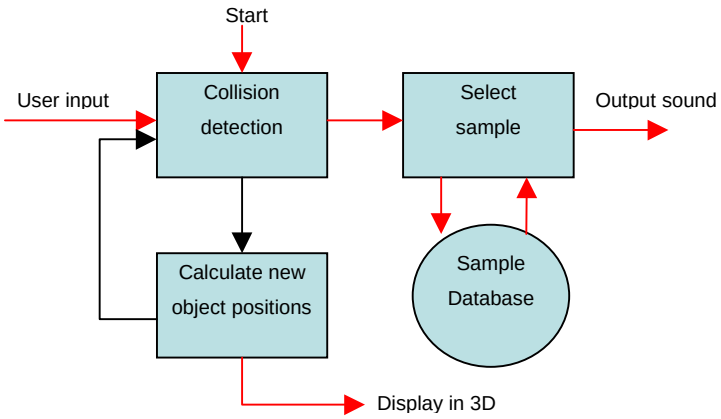

Figure 1: *A typical 3D engine for a game will use collisions between objects to trigger pre-stored samples selected from a database.*

The current trend for computer games and virtual environments is to use pre-recorded sounds, known as samples, for any of the audio components of the system. This has the advantage of requiring minimal processor time to play back the samples, since they usually can be played back without the aid of advanced filters. This method has two main disadvantages.

The first is the inconvenience involved for the developers in having to gather all the necessary sounds together before the game is completed. For large and complex games this can be quite an arduous task.

The second comes from the repetitive nature of the sounds. Due to the sounds being static recordings on the disk or in memory, each time they are played they sound exactly the same as the last time, or the last hundred times.

Computer simulated collisions, from an audio perspective at least, are a simple matter of what object collided with what, and they don't usually take into account the specifics of the collision. In reality the exact points of impact on the colliding objects are very important in determining the resultant positions after the collision. The same factors come into play when determining what sounds can be heard. Take for example, two cubes knocking together. If they were to collide face to face (i.e. flat surfaces together) you would get a particular sound. On the other hand, if one of the blocks was tilted at an angle so that one of its corners hit one of the faces of the other block, you would get a moderately different sound. The difference in sound isn't a huge amount, but enough that one would notice it. This is where sound synthesis appears to be advantageous. If the sounds were being created on the fly, then

factors such as point of impact and force of impact can be used to control the nature of the sounds being produced.

This paper outlines a new project that aims to investigate the issue of employing physically based sound synthesis in the context of a physics engine for computer games, and hence break away from the traditional method of using sampled sounds. Section 2 details the elements behind sound synthesis, beginning with it's origins in music. In Section 3 we provide a detailed discussion of modal synthesis, which is the synthesis method of choice for this project. Section 4 outlines what is involved in integrating sound synthesis with a physics engine and the final section describes future work.

History of Sound Synthesis

The idea of sound synthesis is not new; in fact it has been used since about the 1950's (Roads, 1996). What is relatively new is the idea of generating sounds in real time and from physical data, rather than just abstract parameters as it would be on an electronic keyboard for instance. With recent advances in the algorithms behind digital audio synthesis, combined with the seemingly endless increases in computer processing power and speed, it has now become possible to achieve real-time digital audio synthesis from physical data using only a regular PC.

There is an important distinction that needs to be made when discussing the history of sound synthesis; that is the distinction between analogue and digital synthesis. The first electronic device capable of digital storage didn't appear until the 1940's, however the first modern synthesiser was created at the turn of the century in 1906 by Thaddeus Cahill. It was called the Telharmonium and consisted of shafts and inductors that produced alternating currents of different audio frequencies. Unfortunately the Telharmonium weighed over 200 tons, so it was shunned as being impractical. The basic principles underlying the structure of this device were later used to great effect in the 1950's and 1960's in the form of the Hammond Organ (Roads, 1996).

The first digital synthesis experiments were conducted in 1957 by Max Matthews, et al. in Bell Telephone Labs. These experiments were a far cry from what we know today as digital sounds, and the idea of real-time synthesis was still in the distant future. In fact the calculations for the sound were carried out using powerful computers at IBM in New York and then transferred back to the Bell Labs in New Jersey, where they were played on a much less powerful machine that had audio capabilities. In the years that followed Matthews was involved in creating the musical composition languages, imaginatively titled Music, from

version WE upwards. The original Music program, written solely by Matthews in 1957, could generate only a single waveform type, only allowing the user control over the pitch and duration of each tone. Music II was written a year later for a more powerful IBM computer. This version of the program increased the number of possible waveforms to 16 and also contained four independent voices of sound. This was a big step from the previous version, but in 1960 a new development appeared in the form of the unit generator (Roads, 1996).

Unit generators are basically single components like oscillators, filters, amplifiers, etc. which can be combined together in various ways to produce synthesis instruments, also known as patches. These patches can then generate complex sound signals. The unit generators come in two types, signal generators and signal modifiers. The signal generators are what create the generic sounds in the beginning of the synthesis process. The unit generator concept was used by Matthews and John Miller that same year to develop Music III. Unlike the two previous versions, this one allowed the user to generate a large variety of synthesis techniques, thanks to the user of the unit generators. Music IV and V were developed a few years later, with version V being ported to a multitude of platforms and serving as a good introduction to digital sound synthesis for a wide audience (Roads, 1996).

Since the advent of the unit generators, and in particular their use in Music III, IV, and V, a vast array of synthesis programs and compositional languages has appeared, one of the most popular of which these days is CSound (Roads, 1996). CSound encompasses a vast variety of synthesis features, the details of which are beyond the scope of this document. However one synthesis method that it does provide which is of great interest to us is that of modal synthesis.

Modal Synthesis

Modal synthesis represents objects as collections of many smaller vibrating objects, or substructures. The fundamental theory behind it is that each of these substructures has a set of natural modes of vibration, which are exhibited whenever the structure becomes excited; in our case this would be due to collisions between objects, but it can also occur due to sliding, scraping, pressures, etc. In modal synthesis these mini-objects are modelled mathematically as a set of modal data, consisting of the frequencies and damping coefficients. This data can either be obtained through already documented experiments in engineering, or experimentally through the use of existing equipment in the industry, such as the Active Measurement Facility (ACME) at the University of British Columbia, which has the capability to automatically acquire sound measurements by moving a sound effector around the surface of a test object by a robot arm (Doel, Pai, 2001).

A good physical system which can be used as a base for creating a modal synthesis system is one consisting of a mass and a spring attached to a static surface that can be used to represent the miniature objects that make up the complete object that is being modelled (Cook, 2002). Considering that in acoustics the sounds that are produced are caused by vibrating objects, the use of masses and springs to model them is a logical correlation. If, in one of these systems, the mass is pulled (or pushed) in a particular direction, then the spring will attempt to restore the system to its rest state, initially by applying a force in the opposite direction to the original movement. While the system is attempting to return to rest, it will behave as a simple harmonic oscillator.

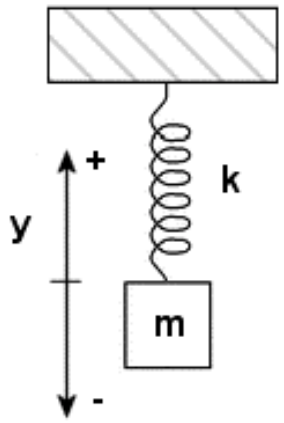

Figure 2: Mass-spring system

In each of these mass/spring systems, the mass is denoted by the symbol, *m*. The spring is defined as being the force required to displace the spring from its rest position, with the force per unit distance being represented as *k*. As with any physical system, energy losses are a factor. Here they are referred to collectively as damping, and denoted by the symbol, *r*. In most systems the damping takes the form of forces such as air resistance, gravity, friction, etc. To use a musical analogy, on a piano damping is imposed through the use of felt pads which deaden the sound after a key is released by the player. The displacement of the mass from rest position is denoted by *y*. Using the same variables, we can now apply Newton's second law of motion to the system to get the following equation:

$$-ky - mg - rv = ma$$ (Force = mass x acceleration)

Here we also introduce the new variables for gravity, *g*, velocity, *v*, and acceleration, *a*. Compared to the magnitude of the force exerted by the deformed spring, the force exerted by gravity is negligible, so we can simplify the equation by removing this operand. This leaves us with:

$$-ky - rv = ma$$

In order to get a solvable equation we need to make some substitutions. Knowing that velocity is the rate of change of position *y* with respect to time, *t*, and acceleration is the rate of change of velocity with time we can get the following:

$$-ky - r\frac{dy}{dt} = m\frac{d^2y}{dt^2} \quad \text{or}$$

$$\frac{d^2y}{dt^2} + \left(\frac{r}{m}\right)\frac{dy}{dt} + \left(\frac{k}{m}\right)y = 0$$

This equation has a solution of the form:

$$y(t) = y_0 e^{(-rt/2m)} \cos\left(t\sqrt{k/m - \left(r/2m\right)^2}\right)$$

This basically informs us that the mass will oscillate up and down in simple harmonic motion, i.e. with a constant frequency. Having to model one of these systems for each mode of each component of an object would be quite an excessive load on the processor for a real-time system. One way of decreasing this load is to use digital filters. We can quantise the previous equation using the following approximations:

$$dy/dt = (y(n) - y(n-1))/T$$
$$d^2y/dt^2 = dv/dt$$
$$= (y(n) - 2y(n-1) + y(n-2))/T$$

where *n* is the sampling number and *T* is the sampling interval. This would yield an equation of the form:

$$y(n) = \frac{y(n-1)(2m+Tr)}{(m+Tt+T^2k)} - \frac{y(n-2)m}{(m+Tr+T^2k)}$$

Although this looks drastically different to the original continuous equation it can be shown experimentally that the two solutions yield results that are quite similar (Cook, 2002). Having the equation in this digital form allows it to be implemented through use of digital electronics like resonant filters set to the previously mentioned resonance frequencies, or in our case simplified algorithms for a computer program that simulates these same filters.

However knowing how to solve a single mass/spring system or having a single modal filter is only a small part in being able to physically model an object. We need to form a type of mesh to represent the object in all dimensions, and to do this it will be necessary to have multiple masses connected together by multiple springs, or the equivalent using digital electronics (Cook, 2002). This not only makes the mathematical calculations more complicated, but it also increases the processing power and time that will be required for a real-time system to able to keep up with the graphical simulation. Once the waveforms have been calculated for each sub-component, they are combined using basic sinusoidal waveform addition.

Integrating Sound Synthesis with a Physical Simulation

Previous work has been done in the area of real-time sound synthesis along with the use of a custom built graphics application (Doel, Kry, Pai, 2001). This project aims to take a more generic approach to allow integration with virtually any existing physical simulation engine available. So before discussing how the sound synthesis is integrated with an existing engine, it is prudent to discuss briefly what a physics engine is, and does. The primary purpose of using a physics engine in an application is to include as much physical realism as possible. Instead of using the traditional methods of animation, like predetermined paths for objects, we can now simply model the virtual world just as one would the real thing, and then let the physics take over. Natural forces like gravity and friction exist in the system as standard, but it is also possible to introduce user-created forces to influence and control the system. The most common feature used in these engines is that of rigid body dynamics.

While it doesn't do all the work for us, it does relieve much of the workload by doing all of the complex physical calculations. In general the physics engine does not provide any additional features other than the physical data, so it is left up to the user to create the graphical engine and, in particular for this project, the sound engine. With the existence of modern hardware-independent graphics APIs, like OpenGL and DirectX, the creation of a graphical display system for an application is not the mammoth task it once was.

As such, the Havok physics engine is one of the key elements of this project, providing real-time physical simulations where required. In recent times the popularity of Havok technology has gone from strength to strength, with major graphics companies incorporating it into their projects, for instance Discreet. Macromedia and Intel have also chosen the Havok system as the backbone to their Shockwave 3D platform (Havok, 2002).

For the purposes of demonstrating the methods researched throughout this project a synthesis engine is being developed. This engine is being developed in three stages: the first being a basic traditional sound engine, i.e. one which uses sampled sounds, rather than synthesising them in real-time. The purpose of this part is to become familiarised with the operation of such an engine when used in conjunction with the Havok physics engine, or any other physical simulator for that matter. At this point in the project, this stage in the development has already been completed. The second stage, which is currently underway, involves the actual synthesis side of the project.

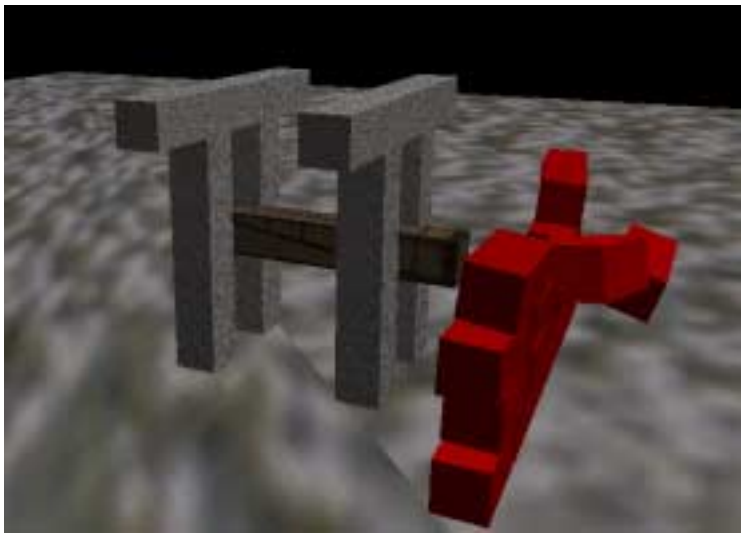

Figure 3: An early demo using the Havok physics engine

In the current state of development of the project a synthesis API developed by Perry Cook, known as the Synthesis Toolkit in C++ (STK) [REF], is being used to demonstrate the possibilities of having a full sound synthesis engine. The toolkit is a set of digital audio signal processing and synthesis classes which can be integrated into an existing program to allow the generating of synthesised sounds. Initially, the STK is being triggered by any collision and plays a synthesised sound according to the material specified in the demo. As the development progresses this will be upgraded to incorporate the detailed physical simulation that already exists in the graphical components. The STK classes contain methods by which we can use the algorithms necessary to create sounds through modal synthesis. These methods take a number of parameters, such as magnitude of impact force, points of impact on the colliding bodies and shape and composition of the objects, and then calculates the appropriate sounds.

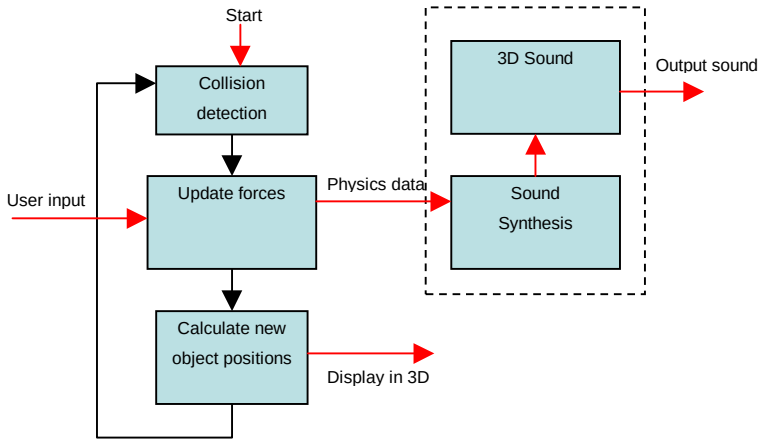

Figure 4: Synthesising sounds based on data from physics engine

Future Work

The next major step in the development will be to use the data retrieved from the Havok engine to more accurately control the output from the synthesis engine. Whether or not modal synthesis is kept as the method of choice for this project remains to be seen, but either way a module will need to be developed that will interpret the physical collision information outputted from the Havok engine and translate it into a format that can be used as input for the synthesis engine.

As it stands, all the research done up until now suggests that modal synthesis is probably going to be the most effective synthesis method. However, as more detailed research is carried out into the inner workings of the various methods available, it may become evident that some of the other synthesis techniques might be more appropriate, or even that it may be more appropriate to use different techniques for different types of objects.

In graphical modelling there is a concept known as level of detail, whereby the model only displays enough detail that the user will notice. We can use a similar technique to reduce the level of detail of the sonic model, by decreasing the number of masses (and hence the number of springs, and thus calculations) in the body. Incorporating this feature will allow us to find a good balance between performance and audio adequacy. Depending on the computers capabilities, or simply the number of sounds that need to be played at the same time, the amount of processing can be drastically reduced for each sound by reducing the number of modes calculated. Also, by giving priority to the higher amplitude modes, we can make sure that the sounds still bear a close resemblance to the real thing, even with only a few basic modes, rather than the full range. This is possible because when all the modes are resonated together the lesser ones are somewhat drowned out by the others. Although having all of them play together gives it a much fuller sound, it isn't necessary to be convincing.

Conclusion and Comments

This paper has described an on-going research project that aims to incorporate a form of physically generated sound synthesis into an existing game engine. We have outlined a framework for incorporating this into a physics engine and pointed out the advantages of implementing this approach. We concluded by detailing the workings of modal synthesis, which has been determined as the most appropriate method for these purposes, and by examining some of the existing work done within the graphics community in this area.

At present a number of demonstration applications have been developed as well as some preliminary research into various synthesis techniques. In the immediate future the intention is to continue more detailed research into modal synthesis and how best to implement a generic sound engine that can be integrated with the existing or future physics engines, and extract any necessary physical data from them.

References

Cook (2000). Perry R. Cook, *Physically-based Parametric Sound Synthesis and Control*, In Proceedings of SIGGRAPH 2000 Conference.

Cook (2002). Perry R. Cook, *Real Sound Synthesis for Interactive Applications*, A K Peters, Ltd., Natick, MA 01760.

Doel (2001). Kees van den Doel, *Modal Synthesis for Resonating Objects*, To appear in a book. Available from: `http://www.cs.ubc.ca/~kvdoel/`

Doel, Kry, Pai (2001). Kees van den Doel, Paul G. Kry, and Dinesh K. Pai, *FOLEYAUTOMATIC: Physically-based Sound Effects for Interactive Simulation and Animation*, In Proceedings of SIGGRAPH 2001 Conference.

Doel, Pai (1996). Kees van den Doel and Dinesh K. Pai, *The Sounds of Physical Shapes*,

Doel, Pai (2001). Kees van den Doel and Dinesh K. Pai, *Modal Synthesis for Resonating Objects*,

Havok (2002). Havok, Retrieved on October 12 2002 from: http://www.havok.com.

O'Brien, Cook, Essl (2001). James F. O'Brien, Perry R. Cook, and Georg Essl, *Synnthesizing Sounds from Physically Based Motion*, In Proceedings of SIGGRAPH 2001 Conference.

Preece, Rogers, et al. (1994). J. Preece, Y. Rogers, et al., *Human-Computer Interaction*, Addison Wesley Publishing Company.

Roads (1996). Curtis Roads, *The Computer Music Tutorial*, The MIT Press, Cambridge.

Young, Freedman (1996). Hugh D. Young and Roger A. Freedman, *University Physics*, Addison Wesley Publishing Company, Inc.

Takala, Hahn (1992). Tapio Takala and James Hahn, *Sound Rendering*, In Proceedings of SIGGRAPH 1992 Conference.

Smith (1992). Julius O. Smith III, *Viewpoints on the History of Digital Synthesis*, In Proceedings of the International Computer Music Conference, Montreal.