

# Real-time Physically-Based Sound

Marc O'Morain  
B.A. (Mod.) Computer Science  
Final Year Project 2005  
Supervisor: Dr. Carol O'Sullivan

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Acknowledgments . . . . .	6
<b>2</b>	<b>Previous Work</b>	<b>7</b>
<b>3</b>	<b>Abstractions</b>	<b>8</b>
3.1	Real-Time Computer Graphics . . . . .	8
3.2	Real-Time Physical Simulation . . . . .	9
3.3	Real-Time Audio Generation . . . . .	10
<b>4</b>	<b>Sound</b>	<b>11</b>
4.1	Digital Audio . . . . .	12
4.2	Sampling . . . . .	12
4.3	The Perception of Sound . . . . .	13
4.4	Sound Generation . . . . .	15
4.4.1	Collision Sounds . . . . .	17
4.4.2	Scraping Sounds . . . . .	18
4.4.3	Rolling Sounds . . . . .	19
<b>5</b>	<b>Audio Playback</b>	<b>20</b>
5.1	Hermite Curves . . . . .	20
5.2	Up-Sampling . . . . .	20
5.3	Sound Buffers . . . . .	22
5.4	Ensuring Seamless Playback of Audio . . . . .	23
<b>6</b>	<b>The Physical Models</b>	<b>25</b>
6.1	Tetrahedra . . . . .	25
6.2	Mass . . . . .	26
6.3	Springs . . . . .	27
6.3.1	Hooke's Law . . . . .	27
6.4	The Audio-Proxy . . . . .	28
6.5	Simulation . . . . .	29
6.6	The Sound Generation Process . . . . .	30

<b>7</b>	<b>Levels of Detail</b>	<b>32</b>
7.1	Levels of Detail in Other Fields . . . . .	32
7.2	Level of Detail in Graphics . . . . .	33
7.2.1	Multi Resolution Meshes . . . . .	33
7.2.2	Mip-Maps . . . . .	33
7.3	Level of Detail in Physics . . . . .	34
7.3.1	Sphere Trees . . . . .	34
7.4	Levels of Detail in Audio . . . . .	35
7.5	Using Multiple Proxies at Varying Levels of Detail . . . . .	36
7.5.1	Deferred Level of Detail Switching . . . . .	37
7.6	Using a Variable Sampling Rate . . . . .	37
7.7	Limiting the Number of Active Proxies . . . . .	38
7.8	Choosing A Level of Detail . . . . .	39
<b>8</b>	<b>Implementation</b>	<b>40</b>
8.1	The Application . . . . .	40
8.2	Tetgen . . . . .	40
8.3	The Simulation Code . . . . .	41
8.4	Screenshots . . . . .	42
<b>9</b>	<b>Conclusion</b>	<b>44</b>
9.1	Review . . . . .	44
9.2	Further Work . . . . .	45
9.3	Further Uses for this Technology . . . . .	45
9.3.1	Brittle Fracture . . . . .	45
9.3.2	Soft body physics . . . . .	46

# List of Figures

4.1	A diagram of a waveform showing amplitude and wavelength.	12
4.2	An example of the Nyquist-Shannon sampling theorem . . . .	14
4.3	A single face on the surface of a bell. . . . .	15
4.4	A face deforming during vibration . . . . .	16
4.5	Two bodies scraping against each other. . . . .	18
4.6	Two bodies rolling on a plane . . . . .	19
5.1	Two Hermite curves . . . . .	21
5.2	Using four samples to interpolate with a Hermite curve. . . .	22
5.3	A Ring Buffer of Audio Buffers . . . . .	24
6.1	A Tetrahedron . . . . .	26
6.2	A Force Impacting on a Surface Triangle . . . . .	31
7.1	A wooden crate texture with 4 levels of detail . . . . .	34
7.2	A Sphere Tree . . . . .	35
7.3	Three Levels of Detail . . . . .	36
8.1	A Screenshot of the Application. . . . .	43
8.2	A Second Screenshot of the Application. . . . .	43
9.1	A Triangle and a a Tetrahedron Being Split . . . . .	46

# Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

---

May 4, 2005

Marc O'Morain

# Chapter 1

## Introduction

The purpose of this project is to generate realistic sound. The sound is generated from interactions between different virtual objects in a computer simulation. Generating sounds in real-time is more desirable than using pre-recorded sounds. To use pre-recorded sounds in a simulation, the sounds need to be recorded, stored and played back. Recording the sounds requires trained Foley artists<sup>1</sup>. Storing sound data requires a large amount of space. When pre-recorded sounds are played back, they are not context sensitive. A light tap or a heavy knock on an object have very different sounds. The sound will also vary depending on where the object is struck. The force and location of a collision must be known in advance to record the corresponding sound. These factors cannot be pre-computed in a simulation that involves human interaction. The audio-generation system introduced in this project creates realistic, context sensitive sound in real-time. No Foley artists are required to generate the sounds, and no storage space is required to store the sounds, because they are produced automatically at run-time. The sounds are context-sensitive, and any solid material can be simulated.

Chapter 2 will provide a brief overview of state-of-the-art real-time collision-based audio generation. Chapter 4 will explain the concept of sound, and how computers represent sound internally. Chapter 5 deals with how the audio generated by this project was manipulated, buffered and played back smoothly. Chapter 3 provides an overview of the abstractions that are commonly used in the field of computer graphics and physics. Chapter 6 contains

---

<sup>1</sup>The *Foley artist* on a film crew is the person who creates and records many of the sound effects.

a detailed description of the techniques that were used to generate the audio. Chapter 7 shows how techniques can be used to improve performance by using multiple levels of detail. Chapter 8 provides a brief explanation of the implementation of the project, the code and libraries used. Further areas into which the technology created in this project could be extended are presented in chapter 9.

## 1.1 Acknowledgments

The author would like to thank Carol O'Sullivan for her supervision, help and support throughout the year and Edsko deVries for spending his time selflessly to teach others how to use L<sup>A</sup>T<sub>E</sub>X. Thank you.

## Chapter 2

# Previous Work

“If I have seen further than others, it is by standing upon the shoulders of giants” – Isaac Newton

In [14] van den Doel et al. first presented a system for generating real-time collision-based sounds. Their system, called `FoleyAutomatic`, consisted of a dynamics simulator, a graphics renderer and an audio modeler. The audio model is based on a system of modal analysis that they present in the paper.

O’Brien et al. [7, 9] have presented two systems for generating sound from physical simulations. Their work uses finite element analysis<sup>1</sup> to simulate the internal stresses and normal modes<sup>2</sup> of solid bodies, a technique which they first introduced to model brittle fracture in [8]. In [9] the technique of decoupling the rigid body simulator from the audio simulation is first presented. Decoupling the audio and rigid body simulations allows each system to be more specialised and optimized for the specific space that they model.

---

<sup>1</sup>In numerical analysis, *finite element analysis* is used for solving partial differential equations (PDE) approximately. Solutions are approximated by either eliminating the differential equation completely (steady state problems), or rendering the PDE into an equivalent ordinary differential equation, which is then solved using standard techniques such as finite differences, etc.[4]

<sup>2</sup>*Normal modes* in an oscillating system are special solutions where all the parts of the system are oscillating with the same frequency (called normal frequencies or allowed frequencies)[4].

## Chapter 3

# Abstractions

“Les bons outils font les bons ouvriers” – French Proverb<sup>1</sup>

From the very first cave paintings in 40,000 BC man has strived to create abstract representations of real-world objects. Drawing, painting, sculpture and photography all try to create a representation of real world objects.

### 3.1 Real-Time Computer Graphics

In computer graphics, an object is presented to the viewer, using an abstraction to represent the object inside the computer. 2D images and 3D models are used to represent real world objects. The majority of research in the field on computer graphics is now centred on 3D graphics, since 2D images can be easily generated from 3D models, and displayed on a 2D screen. These 3D models are purely graphical representations of objects, and store the attributes of the objects that are perceived by humans visually (location, orientation, size, shape, colour and texture). Graphical representation of objects will be referred to as *visual proxies*.

Visual proxies are most commonly stored by computers as shells of trian-

---

<sup>1</sup>Good tools make good workers.

gular polygons. A polygon is a triangle in 3D space, defined by 3 vertices. Associated with each of the three vertices in the polygon are some properties used to render the polygon on-screen<sup>2</sup>. This representation of an object is adequate to describe the visible properties of an object, but the models are merely shells. The models only represent the visible surface of the object, and can not be used to describe physical properties of the object, such as density, stiffness and internal forces. These properties that are lacking from a visual proxy are precisely the properties needed to generate physically-based audio.

### 3.2 Real-Time Physical Simulation

A *physics engine* is a computer program that simulates the movements of 3D objects. There are three main parts to a physics engine: dynamics simulation, collision detection and collision response. Dynamics simulation involves the simulation of the movement of objects in space. Collision detection is used to detect when the objects in the simulation collide as a result of the dynamics simulation. Calculating the correct action to take after a collision has occurred is known as collision response. There are two kinds of physics engines: rigid body engines and soft body engines. Rigid body simulations involve bodies that do not deform, buckle or break. Soft body simulations allow the objects to deform as a result of impacts between bodies. Physical simulation can be performed in real-time or with high-precision. In real-time simulations accuracy is sacrificed in favour of calculation speed, whereas in high-precision simulations, as much time as is needed to perform an exact simulation can be used. High-precision simulations are generally too slow to be performed in real-time.

To perform physical simulation, the physics engine needs to know the shape of the objects involved, as well as the mass, mass distribution, velocity, angular velocity, centre of mass, etc. Collision detection between bodies can be performed using the visual proxy, but usually the mesh resolution of a graphical proxy is excessively detailed for use in physical simulation. A physical proxy can be used to perform collision detection faster.

In a real-time physical simulation, speed and a consistent frame-rate are

---

<sup>2</sup>The properties usually associated with a vertex are a position in space, a unit-vector normal for lighting calculations, and a 2D texture coordinate to map a bitmap texture over the polygon.

more important than numerical accuracy. To help speed up physical simulations, *physics proxies* have been used to store simplified representations of the shapes of objects. Previous work by O’Sullivan et al.[10, 2] has used hierarchical trees of spheres to represent a real world object during physical simulation. The simple and regular shape of spheres makes collision detection almost trivial in comparison to polygon-polygon collision detection.

### 3.3 Real-Time Audio Generation

This project has created the idea of an audio-proxy, which will be used to generate sound. The idea of an audio-proxy is a new one, and so this project uses techniques learned from visual and physics proxies, as well as the past research in the area<sup>3</sup> to create something new. A detailed description of the audio proxies created for this project follows in chapter 6.

---

<sup>3</sup>The past research in the area is detailed in chapter 2.

## Chapter 4

# Sound

“Take care of the sense and the sounds will take care of themselves” – Lewis Carroll, *Alice’s Adventures in Wonderland*

Sound is an abstract concept that describes a series of longitudinal waves<sup>1</sup> that are emitted from some source object that is vibrating. There are four properties of a sound wave that need to be considered: the frequency, wavelength, amplitude and the velocity<sup>2</sup>. The frequency range of sound audible to humans is approximately 20Hz to 20,000Hz.

When a force acts on a body, the sound produced is due to tiny oscillations in the body, at an audible frequency. These waves travel through the air (or any other intermediary medium) to the ear. When a pressure wave reaches the ear, a series of high and low pressure regions impinge upon the eardrum. The arrival of a compression or high pressure region pushes the eardrum inward; the arrival of a low pressure regions serves to pull the eardrum outward. The continuous arrival of high and low pressure regions sets the eardrum into vibrational motion[4]. The vibrations of the eardrum are detected by electrical nerve endings in the ear, and the sound is sent as an electrical signal to the brain.

---

<sup>1</sup>*Longitudinal waves* are waves that have vibrations along or parallel to their direction of travel. Any motion of the medium can be in the same direction to the motion of the wave.

<sup>2</sup>The speed of sound in air is approximately 340.29m/s, however, since this project deals with relatively short distances, we will treat the speed of sound as infinite.

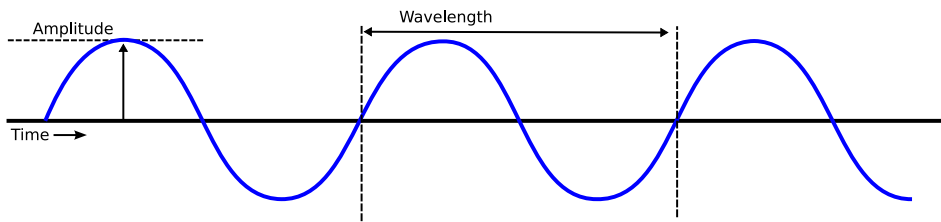


Figure 4.1: A diagram of a waveform showing amplitude and wavelength.

This project will create sound by simulating the minute vibrations in objects during physical collisions.

## 4.1 Digital Audio

Sound waves are analogue, and must be digitised before they can be manipulated by computers. To digitise a sound wave, the amplitude of the wave is measured at certain time intervals. The number of samples taken per second is known as the sampling frequency. The audio data on a CD has a sampling frequency of 44,100Hz. This means that the analogue sound wave that is stored on the disk was sampled 44,100 times per second when the sound was being digitised.

In CD-quality audio, each sample is stored as a 16-bit integer (two bytes). If the sound is stereo, two samples are taken at each measurement - the left channel and the right channel. Therefore 176,400 bytes are required to store one second of CD-Quality audio (44100 samples \* 2 channels \* 2 bytes).

## 4.2 Sampling

This project uses a computer, which is inherently digital, to generate analogue sound waves. The project does not produce the analogue sound wave - it produces the digital samples that would have been obtained, had the sound been recorded by the computer. Less information can be stored in a set of digital samples than was present in the original analogue wave. There will always be a loss of sound quality when using digital audio. In 1928 Harry Nyquist published a theory [6], which Claude Elwood Shannon

proved true in 1949 [13]. This theorem, now known as the Nyquist-Shannon sampling theorem, proves that the human ear is unable to distinguish between digital audio and analogue audio, once the sampling rate is twice as high as the maximum frequency audible by human ears<sup>3</sup> (about 20,000Hz). This project samples audio at 44,100Hz which is more than twice 20,000Hz.

The Nyquist-Shannon sampling theorem states that when sampling a signal, the sampling frequency must be greater than twice the bandwidth of the input signal in order to be able to reconstruct the original perfectly from the sampled version[6, 13].

Figure 4.2 shows a sine wave with a frequency  $X$ . If the wave were to be sampled at a frequency of  $X$ , samples would be taken as shown by the red markers in 4.2(b). This rate of sampling would only take samples at the crests of the wave. When the wave is reproduced, it is missing a significant amount of data. The sound wave in 4.2(c) was produced from the green sample markers, with a sampling frequency of one and a half times  $X$ . The wave is closer to the original wave than 4.2(b), but is still far from perfect. For a perfect reproduction of the wave, we need to sample at a frequency of  $2X$ , as shown by the blue markers in 4.2(d). This wave is an exact reproduction of the original input wave.

### 4.3 The Perception of Sound

Humans can hear sounds up to a frequency of about 20,000Hz. This project produces digital samples of audio as if it had been sampled at 44,100Hz, which is more than twice this maximum frequency audible by humans. Therefore, if the sound generation technique is perfect, a human will not be able to distinguish between the simulation and real sound. This will be discussed further in section 9.2.

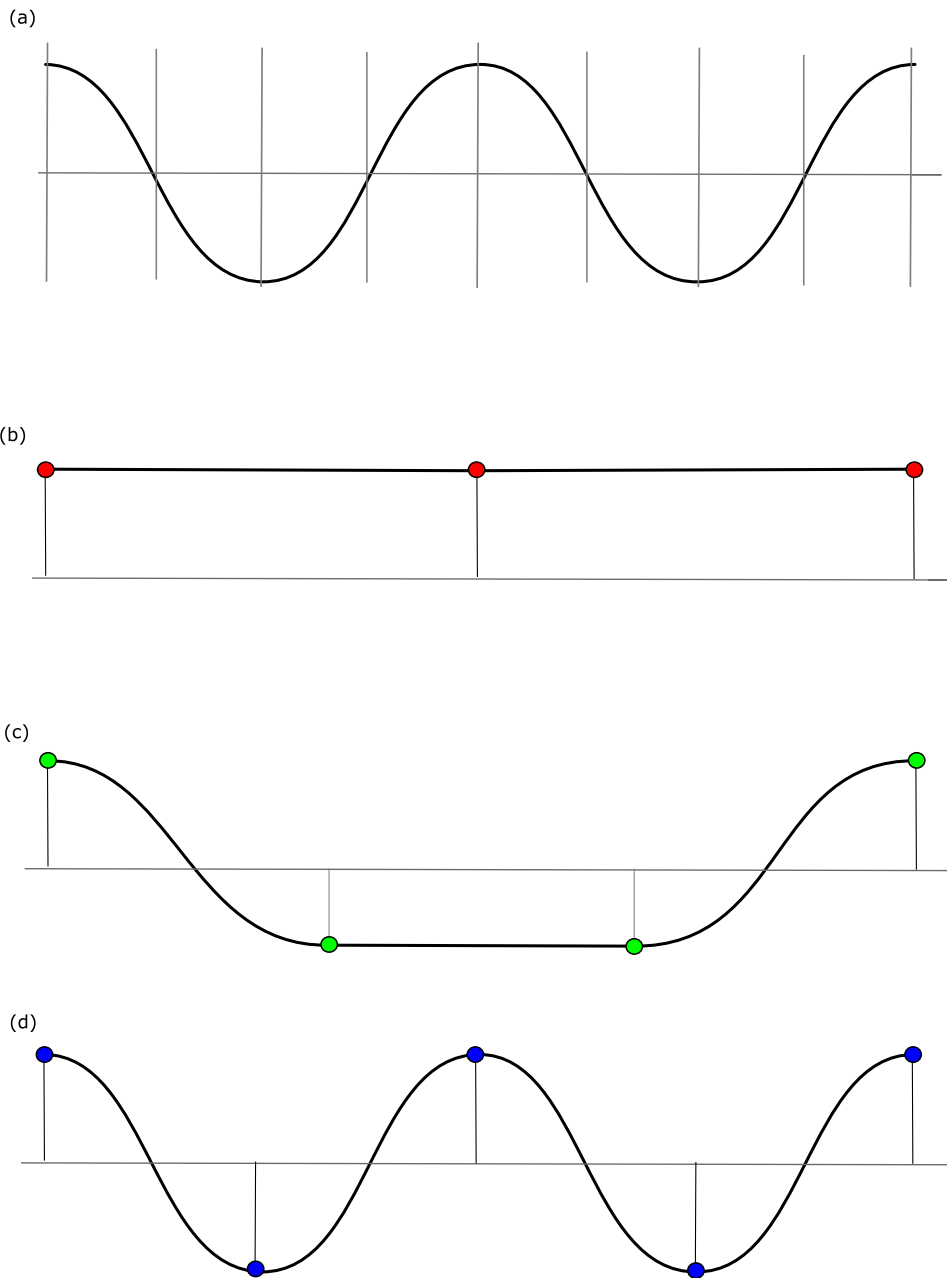


Figure 4.2: An example of the Nyquist-Shannon sampling theorem

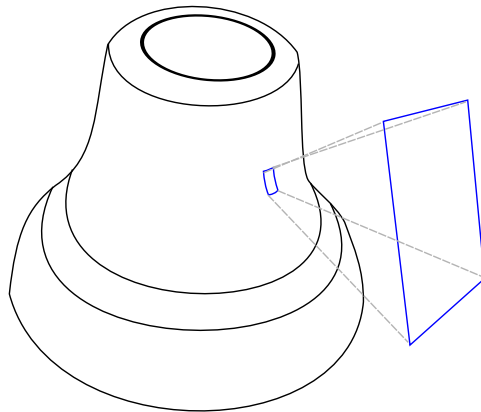


Figure 4.3: A single face on the surface of a bell.

## 4.4 Sound Generation

As stated above, sound is created when some physical body vibrates. This project will model the physical vibration of real world objects, and create the correct sound in response to a force that acts on the body. Figure 4.4 shows a small patch of the surface of the bell shown in figure 4.3. The surface of the bell is represented by the blue polygon. The vertical black lines represent the air pressure surrounding the object. Below each diagram is a sound wave, that represents the sound that is generated by the bell when it is struck.

In 4.4(a), the surface of the bell is at rest. The black lines that represent air pressure are all spaced evenly, showing that the air pressure is even. The sound wave below the diagram has zero amplitude.

In 4.4(b), the surface of the bell after it has been struck. The surface is being forced outwards by some internal force. The black lines show the increase in the surrounding air pressure. This increase is caused by the surface of the bell pushing the surrounding air into a smaller volume. There is an increase in the amplitude of the sound wave that corresponds to this increase in air pressure around the bell.

In 4.4(c) the surface of the bell is being forced inwards by some internal

---

<sup>3</sup>This is why the digital audio stored on a CD is sampled at 44,100Hz.

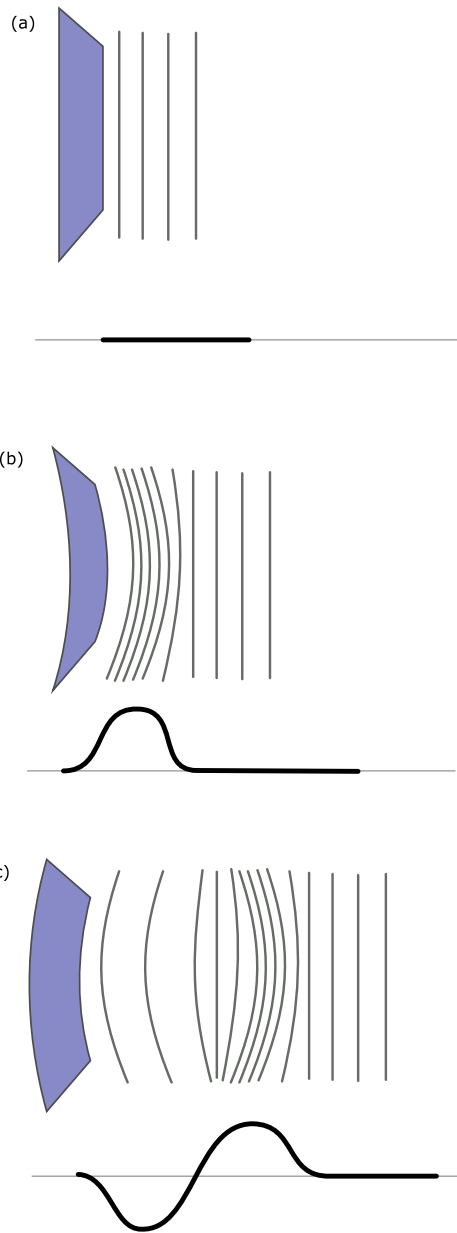


Figure 4.4: A face deforming during vibration

force. The air pressure around the bell lowers, because the surface is literally sucking air toward the bell. This is represented by the increased distance between the black lines. The corresponding sound wave has a crest of negative amplitude that corresponds to the decrease in air pressure.

This project models these deformations in air pressure, that we perceive as sound. Since the deformations are very small (the deflection of a tuning fork when vibrating is about 50 microns[5]), they can be ignored in any rigid-body physics simulation. This was a technique first introduced in [9].

Because the tiny vibrations that generate sound are ignored by the physical simulation, the physical simulation and the audio simulation can be decoupled. This decoupling allows for a substantial speed increase. Performing dynamics simulation is a very complex process. Dynamics simulation is usually performed at either 30 or 60 times a second<sup>4</sup>. Without this decoupling, the dynamics simulation steps would be tied to the simulation rate of the audio simulation, which is performed tens of thousands of times per second, making the simulation too slow to be performed in real-time.

The rigid bodies can be simulated at a more coarse time granularity compared to the audio engine. The audio engine does not have to calculate collision detection or collision response between different bodies - these collisions and responses will be taken care of by the physics engine. There are three main types of sound generated when bodies interact, which are all modeled by this project:

1. Collision Sounds
2. Scraping Sounds
3. Rolling Sounds

#### 4.4.1 Collision Sounds

Collision sounds are generated when two bodies impact on each other. This kind of interaction is the least difficult to model, and therefore collision

---

<sup>4</sup>American and Japanese (NTSC) televisions sets refresh at 60Hz. European (PAL) television sets refresh at 30Hz. Dynamics simulation is performed at the same rate.

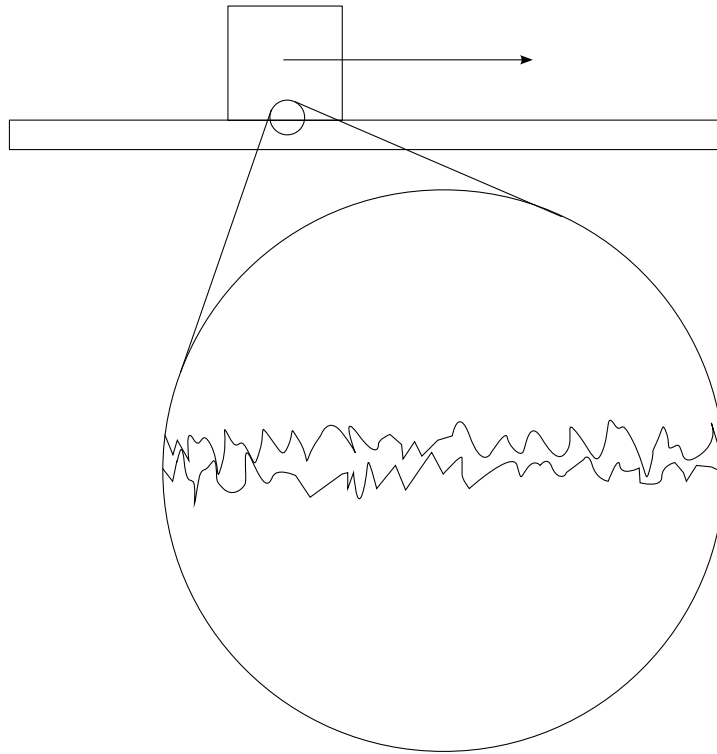


Figure 4.5: Two bodies scraping against each other.

sounds are the least difficult sounds to generate[12]. Two bodies collide and the force of the collision will be used to generate sound.

#### 4.4.2 Scraping Sounds

Scraping sounds are generated when the surfaces of two bodies scrape against each other. When two faces are scraping against each other, this will be represented as a large number of collisions between the two surfaces. Figure 4.5 shows two bodies scraping against each other. The inset shows the actual contact between the two surfaces at a very high level of magnification. The accuracy of the sound depends on the accuracy of the physics engine. A very accurate physics engine will produce a very large number of very low impact collisions from two bodies that are scraping against one another. Each of these collisions generates a small sound. When these sounds are all played, the observer will hear a very accurate scraping sound. It is worth noting here that this project will generate sound in constant time, which is independent

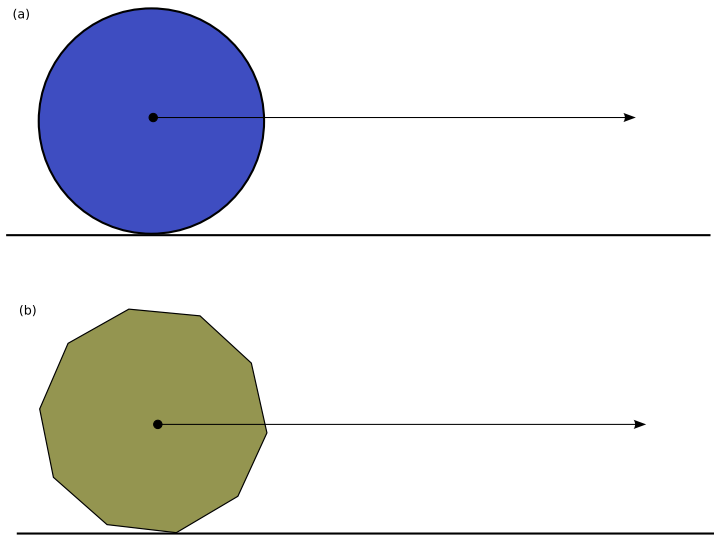


Figure 4.6: Two bodies rolling on a plane

of how many collisions occur in a time-step. The simulation time is only affected by the complexity of the model, as will be seen in chapter 7.

### 4.4.3 Rolling Sounds

Rolling sounds are generated when one body rolls on the surface of another body. Rolling sounds are the most difficult kind of sound to model correctly. Figure 4.6 shows two round bodies rolling across a plane. In 4.6(a), the body is a perfect circle. As it rolls, it will remain in smooth contact with the plane. However, in computer simulations, round surfaces are sub-divided into a number of discrete elements. This is the case in 4.6(b), which shows a round object that has been represented as 10 lines. As this body rolls one revolution across the plain, there will be 10 distinct collisions between the body and the plain – each time an edge rolls on to the plain a distinct collision will be heard, rather than the smooth rolling that would be produced by the perfectly round object[11].

## Chapter 5

# Audio Playback

### 5.1 Hermite Curves

This project uses hermite curves to interpolate between between sound samples. Hermite curves provide a fast and powerful way to interpolate between key points. Figure 5.1 shows two hermite curves. The interpolated points are very fast and easy to calculate. Given a starting point  $p_0$  and an ending point  $p_1$  with starting tangent  $m_0$  and ending tangent  $m_1$ , the polynomial can be defined by:

$$\mathbf{p}(t) = (2t^3 - 3t^2 + 1)\mathbf{p}_0 + (t^3 - 2t^2 + t)\mathbf{m}_0 + (-2t^3 + 3t^2)\mathbf{p}_1 + (t^3 - t^2)\mathbf{m}_1 \quad (5.1)$$

Where  $t \in [0, 1]$ . We can use this formula to create a set of points that lie on the curve between  $p_0$  and  $p_1$ .

### 5.2 Up-Sampling

This project can simulate sound with a sampling frequency of less than 44,100Hz <sup>1</sup>. If the sound is sampled at a rate of 11,050Hz, the simulation will be performed four times faster than sampling at 44,100Hz. However only one quarter of the audio data will be produced, than when sampling

---

<sup>1</sup>The reason that sound would be sampled at a lower frequency is explained in section 7.6.

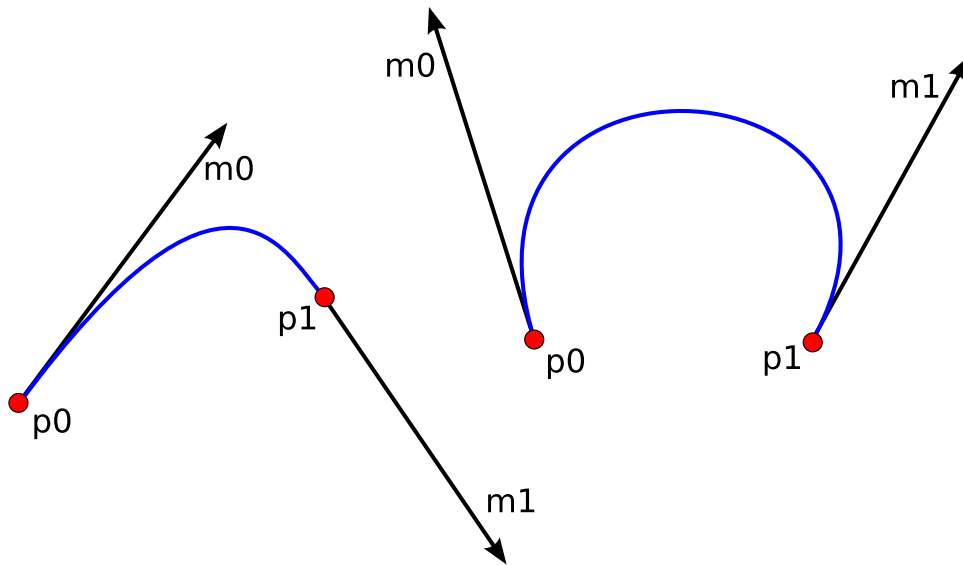


Figure 5.1: Two Hermite curves

at 44,100Hz. The sound-card is expecting audio data at a rate of 44,100Hz, and the sound sampled at a lower frequency would be played four times too fast. A naïve solution to this would be to just repeat each sample 4 times to give data at the required frequency, but this would produce poor audio. This project uses a technique called up-sampling to interpolate between the audio samples taken at the lower frequency, and to extrapolate from the generated curve what the missing audio samples would have been. Taking four samples at a time in a piecemeal fashion, a hermite curve is fitted to the sample data.

Figure 5.2 shows the up-sampling process. 5.2(a) shows a sound wave consisting of 4 samples. The two central samples can be used as  $p_0$  and  $p_1$ . The tangents  $m_0$  and  $m_1$  can be generated from the the slope from sample 1 to the sample 2, and from sample 3 to sample 4 respectively. A hermite curve can then be fitted over the space between the second and third sample. By using this curve to interpolate between sample 2 and sample 3, seven new samples have been created between sample 2 and sample 3, as shown in 5.2(b). By applying this process to a whole set of sample data, taking four samples at a time and filling in the intermediate samples, an audio sample at any lower frequency can be up-sampled to 44,100Hz.

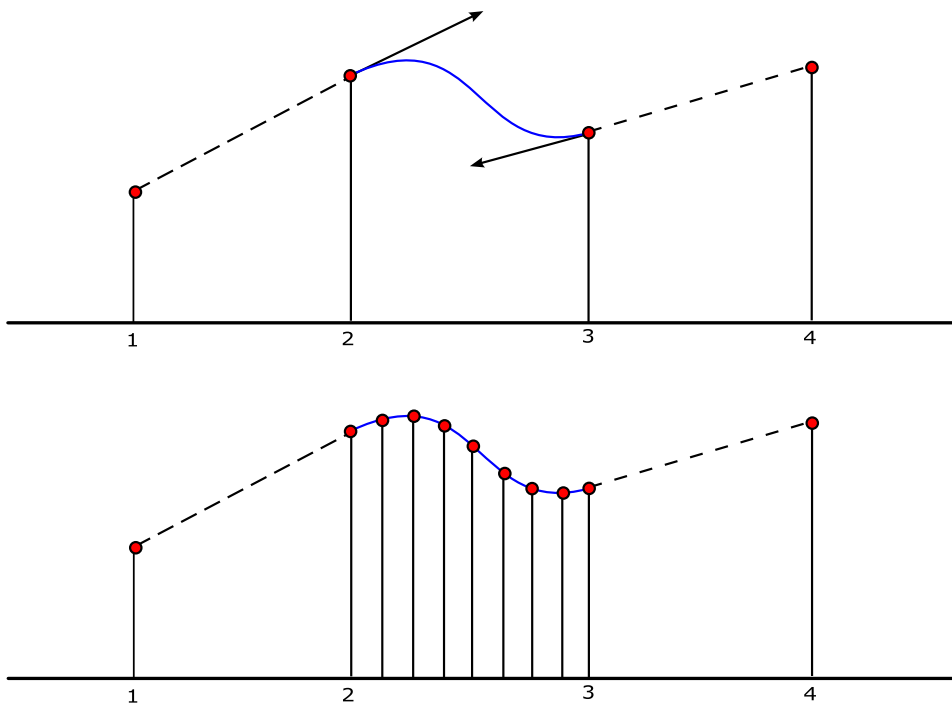


Figure 5.2: Using four samples to interpolate with a Hermite curve.

### 5.3 Sound Buffers

Microsoft Windows plays audio from buffers. The client application provides the Windows WaveOut API with a pointer to a sound buffer and the length of the buffer. Windows then plays this buffer of audio. Since the system in this project produces sound in real time, the sound output must be stored in buffers before it is played. These buffers must be small, so that the delay between the sound being produced, added to a buffer and finally played is not imperceptibly small. The graphics produced by a typical computer graphics simulation are refreshed at a rate of 60 frames per second. This adds a delay of at least  $\frac{1}{60}^{th}$  of a second between the user giving input to a system, and the system responding with visual feedback. This delay is imperceptible to humans. Informal testing was performed during implementation of this project, which showed that this delay ( $\frac{1}{60}^{th}$  of a second) between input and feedback was imperceptible to humans, whether the feedback was visual, aural or both. The system produces  $\frac{1}{60}^{th}$  of a second's worth of audio, and begins to play this sound when the screen is refreshed. This also has the effect of guaranteeing that the audio and video are perfectly synchronised.

## 5.4 Ensuring Seamless Playback of Audio

During simulation a buffer is filled with the audio to be played. If just one buffer were to be used to store the audio for playback, the audio would have to finish playing before the buffer could be filled again. This would create very audible clicks in the audio playback. To overcome this, the same double buffering technique that is used in computer graphics to prevent screen flicker has been employed.

To double buffer the audio, the system creates two sound buffers, and fills one with audio, while the other buffer is playing. When one buffer is finished playing the system swaps the two buffers, so the newly filled buffer begins playing, and the system starts to fill the most recently played buffer. However, if just two audio buffers are used, there still may be an audible gap when the buffers are swapped. To overcome this problem the project will use many audio buffers. These buffers are queued as they are filled, and thus the audio device will always have a sufficient supply of audio to play.

The audio buffers are stored as a ring buffer. The audio buffers can be either ready (to be filled), or busy (in use by the sound card). This ring buffer serves as a ready queue and a busy queue. When an audio buffer is full it is sent to the ready queue of audio buffers, and the next free audio buffer in the ring is next to be filled. Two threads run in the program - one filling the ready audio buffers with sound, the other dispatching the full buffers to the sound card, and returning them to the ready queue after they have been played. The ring buffer is shown in figure 5.3.

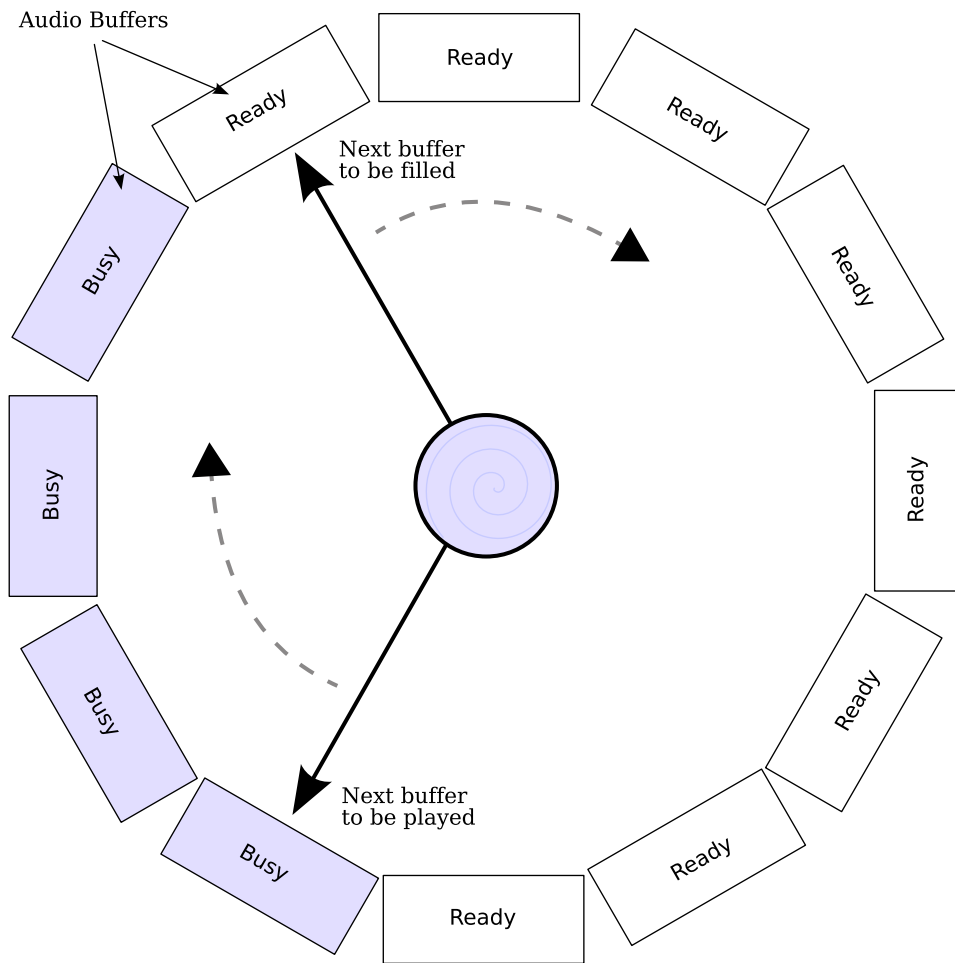


Figure 5.3: A Ring Buffer of Audio Buffers

## Chapter 6

# The Physical Models

“I don’t pretend to understand the universe - it’s much bigger than I am.” – Albert Einstein

### 6.1 Tetrahedra

Commonly in computer graphics, a three-dimensional object is represented as a shell of triangular polygons. Such a representation is adequate for creating a perceptually convincing 3D representation of an object, but it is a poor representation for modeling the internal physical state of an object. This internal state of an object is what the project will use to generate sound. To generate sound we therefore need an abstraction that can represent the internal state of an object.

A better representation is one using tetrahedra. A tetrahedron is a polyhedron<sup>1</sup> composed of four triangular faces, three of which meet at each vertex. A regular tetrahedron is one in which the four triangles are regular, or equilateral, and is one of the Platonic solids<sup>2</sup>. For this project the focus will

---

<sup>1</sup>In mathematics, a *polyhedron* is a three-dimensional shape that is made up of a finite number of polygonal faces which are parts of planes, the faces meet in edges which are straight-line segments, and the edges meet in points called vertices. Cubes, prisms and pyramids are examples of polyhedra[4].

<sup>2</sup>In solid geometry a *Platonic solid* is a convex polyhedron where all sides are equal, all

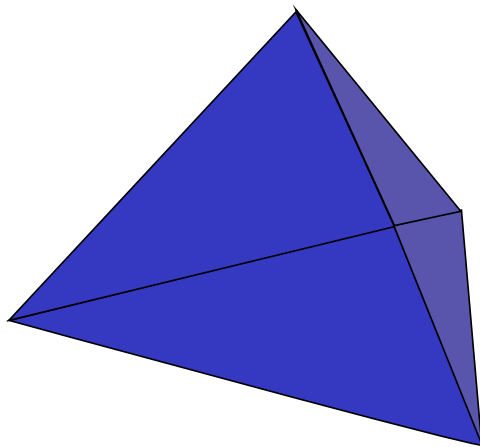


Figure 6.1: A Tetrahedron

be on irregular tetrahedra (a tetrahedon composed of 4 irregular triangles), because they have a very important property: just as any 3D surface can be represented by a mesh of polygons, any complex volume can be broken down into a mesh of irregular tetrahedra.

There is a very important distinction between modeling an object as a mesh of polygons, and modeling a model as a collection of polyhedra. A model composed of polygons is just an empty shell. A model composed of polyhedra can describe density and volume as well as shape. To create sound, this project will model the internal stresses, strains and deformations of solid objects, using polyhedra.

## 6.2 Mass

In the real world, a solid object has its mass distributed continuously across its volume. To model the mass of an object, this project represents the continuous mass as a set of discrete point-mass pairs distributed throughout the volume of the object. Each point-mass pair contains a position inside the object, and a mass associated with that point. This way of modeling the mass distribution of an object is an approximation of how mass is distributed

---

angles are equal, and all faces are identical. The same number of sides must touch at each vertex. Only five Platonic solids exist: the tetrahedron, the hexahedron, the octahedron, the dodecahedron and the icosahedron.[4].

inside a real-world object. The validity of this approximation increases as the number of masses used to model the system increases. As the number of masses approaches infinity, we have returned to a system with continuous distribution of mass. A feature which will be exploited by this project, as explained in section 7.5.

Point masses have no orientation, which makes them ideal for real time simulations, because the simulation does not have to calculate or keep track of their orientation. Calculating the orientation of a body in 3D space is a non-trivial calculation, and the orientation of a body in space is usually stored as a matrix of 16 floating-point numbers. This provides a large saving in storage space and time. The point masses can be evenly or unevenly distributed through out the model, and they can all have a uniform mass, or they can all have different masses. We can thus model objects that do not have uniform mass distribution, at no extra cost to the simulation.

## 6.3 Springs

To bind the masses in the system together, simple springs are used. The springs are computer simulations of real life springs, such as the springs in the suspension of a car. When the springs are compressed they apply a force outwards, and when the springs are extended they will apply a squeezing inwards. Each spring binds two masses together. A spring will endeavor to remain at its natural length by following Hooke's Law.

### 6.3.1 Hooke's Law

Hooke's law of elasticity states that the extension of an elastic rod (its distended length minus its natural length) is linearly proportional to its tension, the force used to stretch it. Similarly, the contraction (negative extension) is proportional to the compression (negative tension).

$$F_s = k_s(L - r) \tag{6.1}$$

Where  $F_s$  is the spring force,  $L$  is the stretched length of the spring,  $r$  is the natural length of the spring and  $k_s$  is the spring constant, a quantity that relates the force exerted on a spring to its deflection. Hooke's law describes a perfect spring, where there are no damping forces. A simulation using just Hooke's law, without damping forces, would oscillate forever. In the real world a friction force,  $F_d$  is also exerted on a spring. This force is proportional to the relative velocity of the two objects connected by the spring, and a damping constant  $k_d$ .

$$F_d = k_d(v_1 - v_2) \tag{6.2}$$

Where  $v_1$  and  $v_2$  are the velocities of the two masses that are connected to either end of the spring. Hooke's law actually holds only approximately, when the deformation (extension or contraction) is small compared to the springs's overall length. For large enough deformations, the atom bonds get broken or rearranged, and the spring may snap, buckle, or permanently deform. Even if that limit is not reached, the force may deviate noticeably from Hooke's law[4]. For this project only very small deformations are considered<sup>3</sup>, and thus Hooke's law is quite adequate.

## 6.4 The Audio-Proxy

In this project, four masses have joined by six springs to create a tetrahedron. Each audio-proxy is made up of many tetrahedra. By combining many of these tetrahedra, any volume can be approximated. The quality of the approximation is directly related to the amount of tetrahedra used to construct it. Increasing the number of tetrahedra used to create the model results in a better approximation of the real world object that the model represents.

When an object is represented as a tetrahedral mesh, the surface of the object is composed of a mesh of triangles. When the body is vibration, the vibrations of this surface mesh will be measured in order to generate sound. The sound produced by each model is determined by the number of springs in the system, the spring and damping constants of each spring,

---

<sup>3</sup>Typical deformations will be in the order of 50 microns.

and the masses at each point. Each spring in the audio proxies can have a different natural length, spring constant and damping constant. Each point-mass in the system can have a different mass. These factors give the system unlimited flexibility in the range of sounds that can be created with it.

The friction forces caused by the dampers reduce the deformation of the spring at each time-step. The frictional forces are proportional to the velocity of the masses that the spring connect. Therefore, at each time-step the deformation of the spring is reduced by some fraction, but may never reach zero, as a result of floating point inaccuracy. In the simulation, a very small number has been defined as the smallest movement by a mass that is significant. Each mass has a *recent memory* of five time-steps associated with it. This memory is used to keep track of how many significant movements were made by that mass in the last 5 time-steps. If a mass has made no significant movements in *recent memory*, the internal state of the mass is set to *stable*. Newton's first law of motion states:

Every object in a state of uniform motion tends to remain in that state of motion unless an external force is applied to it.

When any force acts on a mass, the internal state of the mass is set to *unstable*. While no forces are acting on a mass, the mass will be in *stable* state. Thus we do not need to perform any simulation calculations on a stable mass, until an external force is applied to that mass. When all the masses in an audio-proxy are stable, we do not need to perform any calculations on that proxy until an external force acts upon it. This means that each audio-proxy in the system has no computational cost, unless it is generating sound.

## 6.5 Simulation

This project will be used in conjunction with a physics engine. Each body in the physics simulation will have a corresponding body in the audio simulation. When an interaction occurs in the physics simulation, the audio simulation will be notified of the bodies involved, the point of collision and the force involved. These forces will be applied to the corresponding bodies in the audio simulation.

The audio-proxy and the physics proxy may have subtle differences in shape. If a sphere-tree physics engine is used, such as the kind described in [10], the surface of the physics proxy will be curved, and the surface of the audio-proxies will be composed of many flat polygons. To convert the point of impact on the surface of the physics proxy to a point on the surface of the audio proxy, a ray-tracing technique has been employed. A ray is cast from the centre of mass in the audio proxy to the point of collision. The point of intersection between this ray and the surface of the audio proxy that is closest to the original point of collision is found. This point of collision will be located on a triangle on the surface of the proxy.

A force equal to the force of impact in the collision is divided and applied to the three masses that define the triangle that the collision occurred on. The ratio of the forces applied to each of the masses is inversely proportional to the distance from the point of impact to the mass, as shown in figure 6.2. The top mass is closest to the point of impact, and will receive 60% of the impact force. The other two masses are further from the point of impact, and will receive less of the impact force.

At each time-step, each spring will be examined. If a spring is extended or contracted, it will exert a force on the two masses that it is connected to, following Hooke's law. Once all the springs have been examined, and all these forces have been exerted, the simulator will examine each mass in the system. The total resultant force on each mass is calculated. If the resultant force on a mass is non-zero, the position of the mass is updated, using an improved Euler integration technique[1]. When a spring exerts a force on a mass, and the mass is moved, the movement will usually cause one or more other springs in the system to contract, which will produce vibration. This technique allows the impact forces to propagate through the audio-proxy in a realistic manner. This is how the vibrations that produce sound are simulated.

## 6.6 The Sound Generation Process

The sound will be generated by measuring the amplitude of the tiny vibrations that will take place in the surface of the object. At each time-step in the simulation, each triangle on the surface of the object is examined. The deviation of each triangle from the resting position will be measured. This deviation will be multiplied by the area of the triangle, to give the

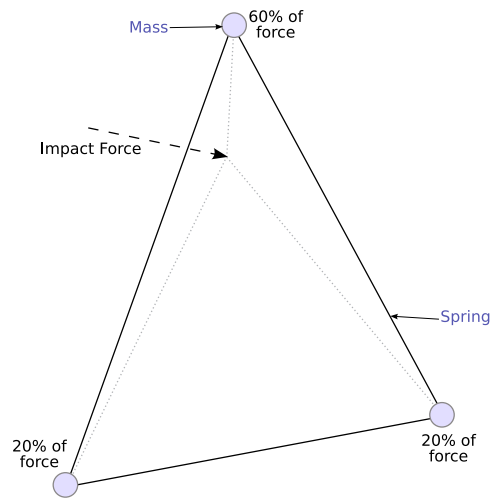


Figure 6.2: A Force Impacting on a Surface Triangle

final amplitude of the sound wave produced by that triangle. Multiplying the amplitude of the sound wave by the area of the face that produced it is very important. It ensures that the volume of a sound produced by a fine tetrahedral mesh will be the same as the volume of a coarse mesh that has been used to represent the same object<sup>4</sup>. The sound waves produced by each triangle on the surface of an object are all added together to produce one sound wave for the object.

---

<sup>4</sup>The reason why an object would be represented by more than one mesh will be explained in section 7.5.

## Chapter 7

# Levels of Detail

“We think in generalities, but we live in detail.” – Alfred North Whitehead

### 7.1 Levels of Detail in Other Fields

To the best of the author’s knowledge, this is the seminal paper on using multiple levels of detail when generating real-time collision based audio. In conducting research into the area of using multiple levels of detail in a real-time simulation, it has been noted that the most closely related fields are that of real time computer graphics and real time physical simulation. A discussion of common techniques employed in using multiple levels of detail in these fields follows. This project introduces some techniques for generating audio using multiple levels of detail.

The aim of using multiple levels of detail in interactive simulations is to use the finite amount of processing time available in two ways – to make as complex a scene as possible, and to perform as rich a simulation as is possible. Certain features of the system, and of human perception, can be taken advantage of to gain speed increased in certain areas, so that the total time available to the system can be used more efficiently. An emphasis can be placed on areas where the observer will notice the added quality, and

certain liberties can be taken in areas where the observer will not notice, or where the observer does not have an interest in. Generally in computer graphics, if an object is big, or close to the observer, it should be rendered using a high level of detail. If an object is small or is far away from the observer, a lower level of detail can be used to render the object, in the hope that the viewer will neither notice nor care about the lack of visual accuracy.

## 7.2 Level of Detail in Graphics

### 7.2.1 Multi Resolution Meshes

In computer graphics, meshes may be stored at different resolutions. When an object is close to the observer, it is drawn with a high resolution mesh, and as the object moves further from the observer, lower and lower resolution meshes can be used. The farther an object is from the observer, the fewer the number pixels that will be used to draw the object on screen. In an ideal scenario, when an object is far from the observer, the observer will not notice that a lower level of detail has been used to render the object. This will be true if the limiting factor for visual accuracy is the screen resolution rather than the accuracy of the mesh used to represent the object.

### 7.2.2 Mip-Maps

A *Mip-Map* is a set of bitmaps that are used as textures for a polygon. The set contains the original bitmap texture, as well as a copy of the image at  $\frac{1}{2}$  size,  $\frac{1}{4}$  size,  $\frac{1}{8}$  size etc. As the polygon that is going to be textured using the mip-map moves farther from the observer, a lower resolution image will be used to texture it. This has two positive effects: The amount of texels that need to be processed are reduced, and lower resolution textures are effectively already anti-aliased. At each mip-map level  $n$ , the storage space requirement to store the mip-map is one quarter of the space needed to store the mip-map at the level  $n - 1$ . This is because the textures are two-dimensional, and therefore the space saving is  $2^2$ .



Figure 7.1: A wooden crate texture with 4 levels of detail

Figure 7.1 shows a wooden crate texture with 4 different mip-maps. Each level of detail takes one quarter of the storage space than is needed for the preceding level of detail. The largest image measures 256 pixels by 256 pixels, the next measures 128 pixels by 128 pixels, and the remain two images measure 64 pixels by 64 pixels and 32 pixels by 32 pixels respectively. Consider a wooden crate that is rendered on the screen using this texture. If the wooden crate is far away from the observer, it takes up less screen area than if the same wooden crate was closer to the observer. When fewer pixels are used to render the texture on-screen, a lower resolution bitmap can be used to texture the crate. If the crate is positioned far enough from the viewer, there will be no loss of visual quality.

## 7.3 Level of Detail in Physics

### 7.3.1 Sphere Trees

In a physical simulation, detecting collisions between polygonal meshes is a very time consuming process. In [10], O'Sullivan et al. present a system for using collections of spheres to approximate the shape of a real world object at multiple levels of detail. Detecting collisions between two spheres is a trivial computational operation compared to detecting a collision between two polygonal meshes. The system increases the number of spheres used to

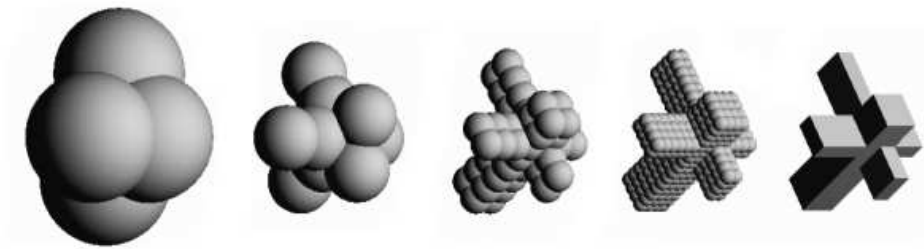


Figure 7.2: A Sphere Tree

represent the the object at each level of detail. Figure 7.2 shows a sphere tree at four levels of refinement, and the actual polygonal mesh of the object being modeled.

## 7.4 Levels of Detail in Audio

In generating audio, this project proposes three ways of varying the level of detail in a simulation.

1. Using Multiple Proxies at Varying Levels of Detail
2. Using a Variable Sampling Rate
3. Limiting the Number of Active Proxies

Using multiple proxies at varying levels of detail, and using a variable sampling rate both work in the same way. They offer the client application a trade off between simulation time and the accuracy of the sound produced. The client application is free to choose a level of detail for each model in the system, and change the level of detail in real-time as the simulation progresses. In other fields, the level of detail is usually chosen based on the amount of time available to the simulation. If there are multiple bodies in the simulation that have variable levels of detail, the client application must prioitise which body to spend most time on. Usually this decision is based on some error metric.

This project proposes storing the bodies in the simulation in a priority tree. The priority associated with each body would be generated by some heuristic

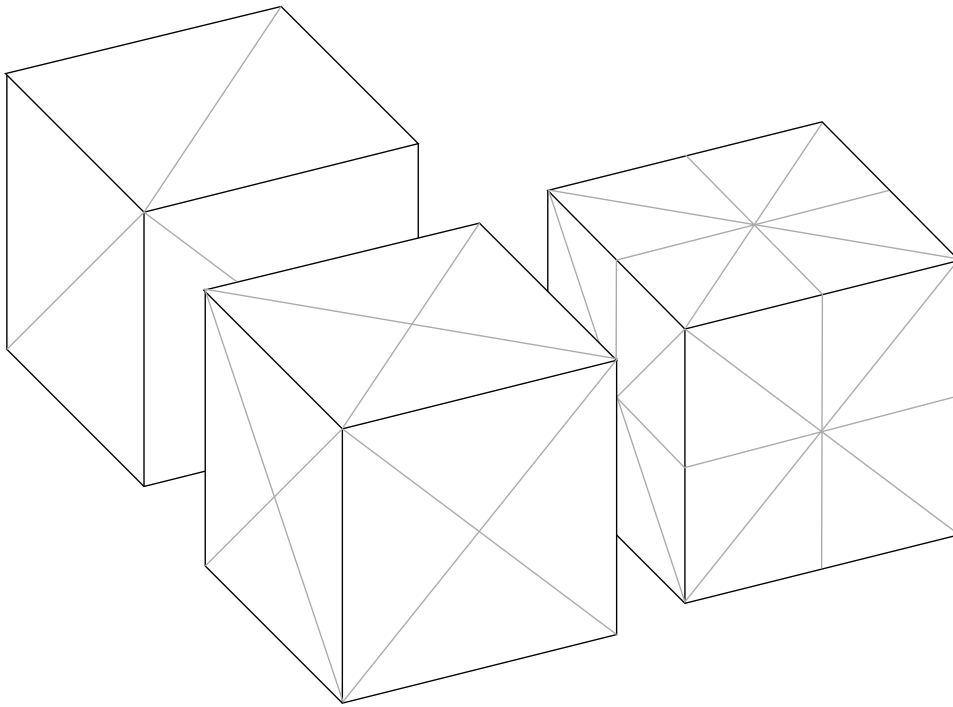


Figure 7.3: Three Levels of Detail

defined by the user. Possible heuristics are discussed in section 7.8. The bodies with the highest priority are simulated using the highest levels of detail, and the bodies with the lowest priority are simulated with the lowest level of detail. This priority tree approach has been adapted by the priority queue used for graphical level of detail in [3].

## 7.5 Using Multiple Proxies at Varying Levels of Detail

The time taken to simulate one audio time-step is linearly proportional to the amount of tetrahedra in the proxy. If the number of tetrahedra in a model is doubled, the time taken to computer one time-step will also double. The proxies in this project are built with tetrahedra. If more tetrahedra are used to build a tetrahedra, the audio produced will be of a higher quality. Figure 7.3 shows a cube with three different levels of detail. Multiple audio proxies can be used to represent a body in the simulation,

at varying levels of tetrahedral refinement. This project allows the client application to dynamically change the number of tetrahedra in a model in real-time.

The time taken to perform a simulation step with an audio-proxy at each given level of detail is constant. Whereas mip-maps are two dimensional, the tetrahedra in this project are three-dimensional. The space saving of  $2^2$  per level of detail for mip-maps becomes a space saving of  $2^3$  when the domain is three-dimensional. Each level of detail will be eight times faster to simulate than the level of detail above it. An application that uses this system will know ahead of time how much time is needed to simulate, and how much time is available. The client application can choose how it wants to spend the time available to it. An audio proxy with a higher level of detail will simulate better quality sound, but the time taken to perform the simulation will be higher.

### 7.5.1 Deferred Level of Detail Switching

The simulation will only switch to a proxy from a higher or lower level of detail if the current proxy is in a stable state. If the simulation requests a change in level of detail while a proxy is vibrating it will defer the switch until the current body has stopped vibrating. When the level of detail switches, the new proxy will be in a stable state - and thus not vibrating. If a level of detail transition took place with a proxy is vibrating, the audio would stop abruptly, producing audible artifact<sup>1</sup>. Deferring the change of level of detail has virtually no adverse effect on the simulation, because the proxies vibrate for a very short period of time, but it does have a notable positive effect, in terms of the fidelity of the audio simulation.

## 7.6 Using a Variable Sampling Rate

Since all sound generated will be up-sampled<sup>2</sup> to a frequency of 44,100Hz before it is played by the sound card, the sampling frequency for each model

---

<sup>1</sup>In this context, an *artifact* is an inaccurate effect or result resulting from the technology used or from experimental error. (American Heritage Dictionary of the English Language)

<sup>2</sup>The *up-sampling* process is explained in section 5.2.

is irrelevant for the sound hardware. All bodies in the system are stored and simulated independently from one another. This means that the simulation is free to use a different time-step for each model in the system. If the time-step is doubled, the simulation can be calculated in half the time, but there will be a loss of audio quality. A simulation that contains many different bodies, is free to chose a different sampling rate for each body in the simulation. A body in the simulation that has been assigned a high priority can be simulated with a very fine sampling rate, and very high quality audio will be produced. A low priority body can be simulated with a very coarse sampling rate. The audio generated will be of a lower quality than that produced at a fine sampling rates, but will be sufficient for a low priority body.

## 7.7 Limiting the Number of Active Proxies

When rendering and lighting a 3D scene, the OpenGL graphics library imposes a limit of 8 lights on the client application. This is an adequate restriction to place on a rendering system, which can be seen in the quality of graphics in any of the latest computer games. This project proposes applying the same kind of limit on the audio generation system. This limit can be any arbitrary integer, but for the sake of explanation, 8 will be used. By limiting the number of active proxies at any one time, the worst-case time requirement for a simulation step can be very easily calculated. This is a very desirable property for a real time system.

This project performs the simulations in order of priority. This implies that at any given time in the simulation, the 8 highest-priority bodies in an unstable state will be simulated, and sound generated by the lower priority bodies will be ignored. If there are 8 bodies vibrating, and a collision occurs in the physics engine with a body that is not vibrating, the action taken will depend on the priority of the new body.

If the new body has a lower priority than all of the 8 bodies that are currently vibrating, the new collision is ignored by the audio engine. The user will not notice that the collision did not make a sound, because there are eight other bodies in the system with a higher priority generating audio at the current time.

If the new body has a priority higher than any of the eight bodies that are currently vibrating, the vibrating body with the lowest priority is set to a stable state, and its vibration is stopped. Stopping the vibration of a body from vibrating when it is the only body generating audio in the system would produce an audible artifact, but there are now 8 other bodies with higher priority that are generating audio at the current time, so the audible artifact will not be discernible to the user.

## 7.8 Choosing A Level of Detail

In the field of computer graphics, the distance from the observer is sometimes used for choosing a level of detail. This method would work well if all objects in the system were at the same size and had the same complexity. However if for example a very large object was situated far away from the observer, different levels of detail would be quite noticeable to the observer. Other papers[3] have suggested using an error metric based on the difference between the actual object and the current level of detail, measured in pixels in screen-space. The distance from the observer to the object can also be used as a metric for an audio simulation. However, a parallel can be drawn between visual size in computer graphics and sound intensity in simulated audio. If a sound was emitted from a source that was far away, but the sound was very intense, an observer would notice if an inadequate level of detail was used in the simulation. There is no parallel that can be made with the screen-space error metric, since there is no original sound wave to compare the simulated sound wave to.

Humans can only see that which is in the view-frustum in front of the eyes. In computer graphics it is common to cull (not render) anything that is not within the view frustum. This culling produces a huge time saving, because texturing, lighting and rendering geometry takes some time. Human ears, however, can hear in all directions, not just what is in front of them, and thus it is not possible to cull any sound in a similar fashion.

To implement this project, distance from the observer has been used to choose a level of detail.

## Chapter 8

# Implementation

### 8.1 The Application

To demonstrate the sound generation and level of detail techniques developed for this project, a small test application was written. The application was written in an object oriented fashion, using C++. The user interface was written using the Fox Toolkit<sup>1</sup>. The Fox Toolkit was chosen for a number of reasons. The toolkit is free software released under the GNU Lesser Public License. It works on both Microsoft Windows and X11 based systems. It is written in C++, the same language as was used for writing the project. It provides a rich widget set, including an OpenGL canvas widget. The OpenGL graphics library was used for rendering all graphics in the project.

### 8.2 Tetgen

A program called *tetgen*<sup>2</sup> was used to generate the tetrahedral models for this project. Tetgen was used to convert standard 3D models into tetrahedral meshes. Each model was converted into three tetrahedral meshes, at

---

<sup>1</sup>The Fox Toolkit is available from <http://fox-toolkit.org/>

<sup>2</sup>Tetgen is available from <http://tetgen.berlios.de/>

different mesh resolutions. These three tetrahedral meshes were used as the three levels of detail in the program

### 8.3 The Simulation Code

The main loop of the sound generation system can be described by the following pseudo-code:

```
Do 60 times a second:
{
  Do (44,100/60) times:
  {
    For each Body B in the simulation:
    {
      For each Spring S in B:
        Resolve Forces(S);
      For each Mass M in B:
        Simulate(M);
      For each Face F on the surface of B:
        Generate Sound(F);
    }
  }
  If current sound buffer is full:
  {
    Play current sound buffer,
    Prepare the next free sound buffer.
  }

  Advance the physics simulation.
  Re-draw the screen.
}
```

#### **Resolve Forces(Spring S)**

This function calculates the forces to applied to the two masses that are connected to the spring S. These forces are calculated from Hooke's law, as explained in 6.3.1. It applies the forces to the masses involved.

### **Simulate (Mass M)**

This function calculates the resultant force on the Mass M, from all the forces that have been applied to M. Forces may have been applied to M by any Springs that it is attached to, and by any collisions that have occurred if M is on the surface of the object. It then advances the simulation by one time-step. To perform the simulation, an improved Euler integration technique is used, as described in [1].

### **Generate Sound (Face F)**

This function calculates how far the face F has deviated from its resting position. This deviation is multiplied by the area of the face, to produce the amplitude of the sound wave. This new sound sample is added to the current audio buffer.

## **8.4 Screenshots**

Figure 8.1 shows the application in use. The green and red bar at the top of the application shows any sound that has been output by the application in the last 2 seconds. Below this bar are four views of an object – a view from the top, front and side, as well as a 3D view in the top-right corner. The user can click on the object in this view to simulate striking the object at the position of the mouse cursor. The six slide controls control the tension and friction of the springs in the object. Figure 8.2 shows a different model which has been loaded into the application. A sound wave is visible in the top bar.

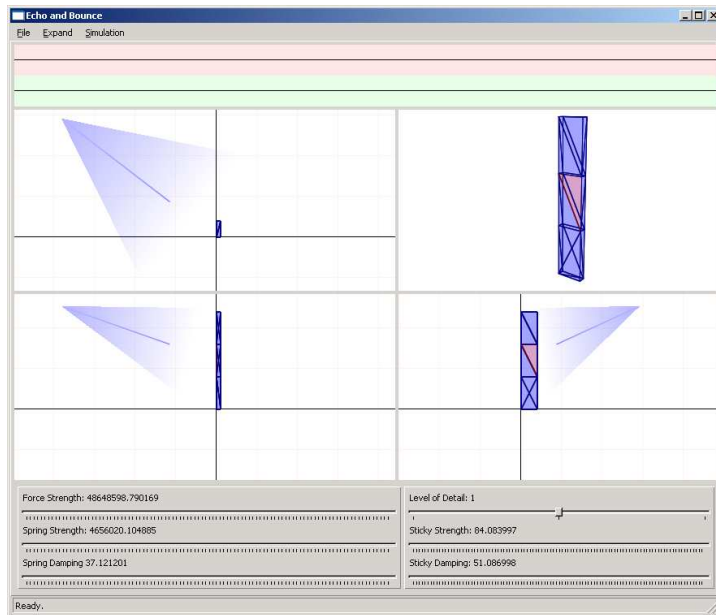


Figure 8.1: A Screenshot of the Application.

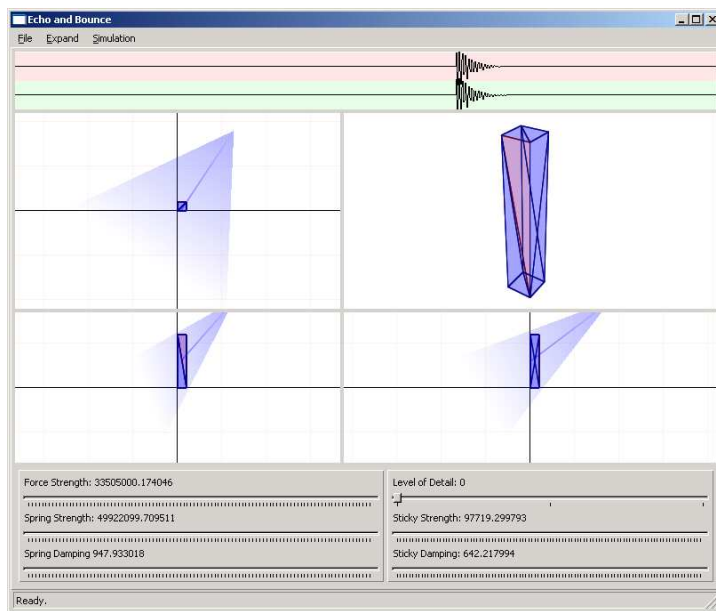


Figure 8.2: A Second Screenshot of the Application.

# Chapter 9

## Conclusion

### 9.1 Review

The level of detail system created for this project was innovative, and based on new concepts. These concepts drew on level of detail work in other fields, and applied their techniques to the field of sound generation. A working system was produced which demonstrated the effectiveness of these new concepts.

The audio quality in this project was not as high as was anticipated. The reason for this is that the springs in the audio-proxies were not of a high enough tension, and thus the vibrations were not of high enough a frequency to model realistic sound. Although floating point accuracy was a problem here, the real limitation was a speed issue. To improve the quality of the sound, more simulation time was needed, but this was not possible in a real-time system<sup>1</sup>.

The modularity of this project makes it useful in a wide variety of applications.

---

<sup>1</sup>During testing more time was used to simulate, in an effort to increase audio quality. The audio buffers could not be filled at a fast enough rate. The thread playing the audio would run out of data to play, and have to wait for more audio to be produced. This lead to a loud audible artifact at a frequency of 60Hz. This sound was most unpleasant.

## 9.2 Further Work

One area of work which is left outstanding is a study into the perception of collision based sound. Vision is a very strong human sense, and the eyes can only see what is in front of them. If a collision occurs behind the observer, it may be possible to assign a very low priority to this collision. If the sound played is not accurate, the observer will not be able to see the source of the sound to verify that it is indeed not an accurate sound. Using a low priority for sounds like this would free up processing time for collisions that occur in front of the observer. There may be areas other where a simulation can take liberties with audio fidelity where the human perception of sound will not notice the imprecision.

Another interesting area of research would be to create a “Turing Test” for computer-audio simulations. Such a test would involve a blindfolded person listening to audio generated by a computer simulation, and real audio. If the human cannot distinguish between computer generated sounds and real sounds, then the audio generation technique can be defined as “perfect”.

## 9.3 Further Uses for this Technology

### 9.3.1 Brittle Fracture

The physical simulation used in this project could very easily be modified to be used to model brittle fracture of objects. The tetrahedral mesh used in this project can be made to shatter by following two very simple rules:

If a mass is being acted on by two or more forces in opposite directions, and the difference between the forces is higher than some maximum force, split the mass into two masses, and add one to each tetrahedra involved. The corresponding tetrahedra are no longer attached to one another.

If a spring is stretched tighter than some maximum tension, split the spring, and create a mass half-way along its length. Any tetrahedra that are composed of this spring would also be split – if a tetrahedra is split in this way, two tetrahedra are generated. The splitting process is shown in Figure 9.1.

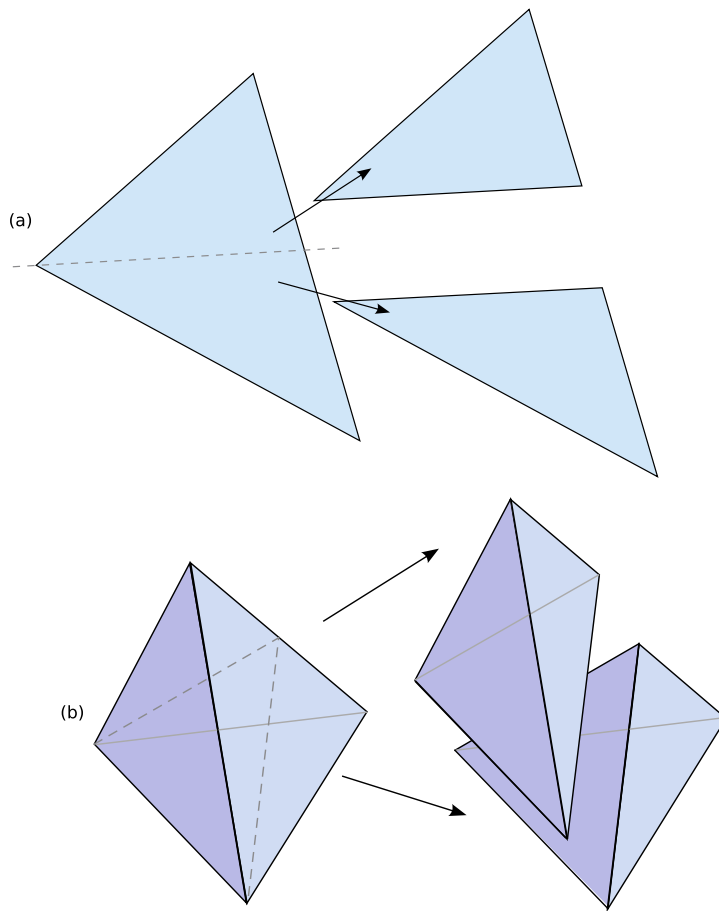


Figure 9.1: A Triangle and a Tetrahedron Being Split

Figure 9.1(a) shows a triangle begin split in two, from an edge to a vertex. Figure 9.1(b) shows a tetrahedron begin split in two, also from an edge to a vertex. This property of tetrahedra makes them perfect for a brittle fracture simulation.

### 9.3.2 Soft body physics

The springs in this project are have a high tension, in order to produce vibrations at an audible frequency. If the springs are given a much lower tension, the vibrations are visible. Videos have been rendered showing this slow vibration, have been included on the project presentation CD enclosed

with this report. This slow vibration is perfect for modeling soft bodies, such as trees, plants or jelly.

# Bibliography

- [1] D. M. Bourg. *Physics for Game Developers*, chapter 11, pages 180–183. O’Reilly, 2002.
- [2] G. Bradshaw and C. O’Sullivan. Sphere-tree construction using dynamic medial axis approximation. In *SCA ’02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 33–40, New York, NY, USA, 2002. ACM Press.
- [3] M. A. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. Roaming terrain: real-time optimally adapting meshes. In *IEEE Visualization*, pages 81–88, 1997.
- [4] W. Foundation, editor. *Wikipedia, The Free Encyclopedia*. Wikimedia Foundation, 2005.
- [5] A. D. L. Humphris, J. K. Hobbs, and M. J. Miles. Ultrahigh-speed scanning near-field optical microscopy capable of over 100 frames per second. In *Applied Physics Letters*, volume 83, pages 6–8, July 2003.
- [6] H. Nyquist. Certain topics in telegraph transmission theory. *AIEE Trans.*, 47:617–644, 1928.
- [7] J. F. O’Brien, P. R. Cook, and G. Essl. Synthesizing sounds from physically based motion. In *Proceedings of ACM SIGGRAPH 2001*, pages 529–536. ACM Press, Aug. 2001.
- [8] J. F. O’Brien and J. K. Hodgins. Animating fracture. *Communications of the ACM*, 43(7):68–75, July 2000.
- [9] J. F. O’Brien, C. Shen, and C. M. Gatchalian. Synthesizing sounds from rigid-body simulations. In *The ACM SIGGRAPH 2002 Symposium on Computer Animation*, pages 175–181. ACM Press, July 2002.
- [10] C. O’Sullivan and J. Dingliana. Real-time collision detection and response using sphere-trees. In *15th Spring Conference on Computer Graphics*, pages 83–92, Apr. 1999.

- [11] M. Rath. An expressive real-time sound model of rolling. In *DAFx-03 - The 6th International Conference on Digital Audio Effects*, pages 165–168, Queen Mary, University of London, Sept. 2003.
- [12] M. Rath, D. Rocchesso, and F. Avanzini. Physically-based real-time modeling of contact sounds. In *Proc. Int. Computer Music Conf. (ICMC'02)*, pages 16–20, Gteborg, Sweden, Sept. 2002.
- [13] C. E. Shannon. Communication in the presence of noise. *Proc. IRE*, 37:10–21, 1949.
- [14] K. van den Doel, P. G. Kry, and D. K. Pai. Foleyautomatic: Physically-based sound effects for interactive simulation and animation. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 537–544, 2001.