Sound Synthesis for Virtual Reality and Computer Games

by

Cornelis Pieter van den Doel

Ph.D., University of California at Santa Cruz, 1984

M.Sc., University of British Columbia, 1994

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

we accept this thesis as conforming
to the required standard

The University of British Columbia

November 1998

Abstract

The synthesis of audio in real-time computer simulations is investigated. A physics based parameterized vibration model for physical objects is constructed, and a real-time synthesis algorithm is developed which allows the synthesis of the sound made by such objects under any kind of interaction force.

Methods for obtaining the parameters of such models are investigated. We study mathematical models with simple geometries, parameter fitting to measured data, and empirical models.

Models for interaction forces occurring during contacts between rigid bodies such as impact and sliding interactions are developed, as well as models for the driving forces for combustion engines and avalanches.

Studies were conducted of several objects which were successfully modeled with these techniques. Several computer programs were written for the testing of models, for the construction of models, and for the demonstration of the level of realism that can be achieved with this type of synthesis.

It is concluded that this type of synthesis can generate realistic, interactive audio using only a small fraction of available CPU power on modern personal computers.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my thesis supervisor, Dinesh K. Pai, for his active participation in this research and his support. I appreciate feedback and comments from the other members of my thesis committee, Alain Fournier and Uri Ascher. Thanks to IRIS, ASI, and IVL Technologies Ltd., for financial support.

Cornelis Pieter van den Doel

The University of British Columbia

November 1998

Chapter 1

Audio in Virtual Reality

The research described in this thesis was conducted to create a methodology for the rendering of audio in virtual reality environments such as simulations and computer games. The focus is on a topic which has not been investigated very much at the time of writing, namely the computation (synthesis) and rendering of "natural" sounds produced by physical objects. Most research in sound synthesis has concentrated on "computer music," where the primary interest is to create interesting sounds, or to reproduce musical sounds, rather than to faithfully reproduce existing naturally occurring sounds.

1.1 Goals and Background

Audio techniques [37] used in games and virtual reality (VR) are more inspired by movie and television sound effects, than based on a physical simulation approach. Unfortunately, because of the interactivity of games and VR, these techniques are of limited use in such environments.

A common technique to generate audio in a game or simulation is to play

back a prerecorded sound (a "sample") when a certain event, such as one solid object colliding with another, or a player of a video game killing the final monster, takes place. For "effects" such as a triumphant fanfare after winning the game this is appropriate, but for more subtle interactive sounds caused by physical objects it is quite tedious to have the same sounds repeated over and over again. Other types of environmental sounds, such as those caused by objects sliding and rolling, are continuous and are driven by the momentary state of the physical objects that cause them. They can not be prerecorded and some type of synthesis is necessary. Synthesis is a form of modeling or simulation, so it is useful to compare audio synthesis with simulation in a more general context.

A lot of research in "reality simulation", i.e., the creation of a sensory illusion with the computer, has focused on computer graphics rather than on computer sound. As a result, many techniques and software packages exist to display and animate a scene using computer graphics at many levels of realism, but similar techniques to recreate the sounds of the scene are still underdeveloped.

The study of creating sound with a computer is analogous to computer graphics, just as the study of interpreting sound with a computer is analogous to computer vision. However, the analogy between sound and graphics is not straightforward, because vision and hearing are very different sensory modes. Nevertheless, it is useful to try to draw analogies between the two. In Table 1.1 we have indicated some possible connections one can make at various levels. The correspondences indicated in the table are intended to stimulate the mind, rather than to show actual connections between sound and graphics. For each of the correspondences one can find arguments against and for the analogy.

The modeling and simulation of rigid bodies with contacts has been inves-

Sound	**Graphics**
Sample	Pixel
Sampled sound	Digital picture
Stereo sound	Perspective
Spatial sound	Stereo display
FM synthesis	Color-map animation
Sound emission	Shading
Vibration analysis	Surface modeling
Distance attenuation	Haze or fog
Impact sounds	Flashes
Scraping, rolling	Animation
Material	Color
Room acoustics	Raytracing and radiosity

Table 1.1: Graphics-Audio analogies.

tigated widely [78, 79, 47, 42, 10, 9, 22] recently, and this type of simulation has a lot in common with the simulation of audio. In both cases the task is to update the state of the objects/audio-buffers in a manner consistent with the (simplified) physical laws governing the simulation. A rigid body simulation would also be an ideal environment to integrate with an audio simulation, as the information about body contact forces, collisions, and contacts, which is usually directly available from the simulation, can be used to drive the audio synthesis.

What are the physical factors determining the generation of audio? Sound is most commonly created by accelerated bodies and their vibrating surfaces. Heat and turbulence can also act as sources of sound. These bodies interact with the medium (air, usually), and generate pressure waves that propagate through the air, and scatter from other objects. The sound is scattered at the ear and passes through the ear canal and is detected at the eardrum.

It is often possible to regard the sound production as occurring in several

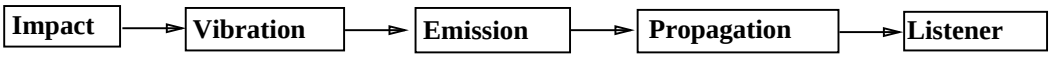

Figure 1.1: The stages of modeling needed to compute audio.

stages that can be modeled independently. A force such as an impact is applied to a material object, which vibrates. The object emits sound waves, which propagate through the environment and are detected by the human ears. We have depicted this audio "pipeline" in Figure 1.1.

To produce realistic simulated sounds with a computer we need to understand the physics behind the sound generating phenomena as well as the nature of human perception of sound. A simplified model of the complete physics of the sound generation should be created, so that the associated sounds can be computed and rendered in real-time. The computation should be based on physical laws, not on "hacked" algorithms. This provides the possibility of systematic refinement, based on knowledge of physical laws.

1.2 Example of an Audio Simulation

When we strike an object such as a metal bar, we hear a brief transient or click followed by a more or less sustained sound, which then decays. The click or onset has some role in identifying the sound. For example, try listening to a recording of a flute played backwards. The sound is no longer as clearly recognizable as a flute, even though the sustained part of the sound is unchanged. See also [15, 51].

Nevertheless, a lot of information about the nature of the object is present in the sustained part. To obtain this, we need to compute the vibrations of an object when it is struck, and compute the resulting sound emitted.

The sound made by a struck object depends on many factors, of which we

consider the following:

1. *The shape of the object.* The propagation of vibrations in the object depends on its geometry. This is why a struck gong sounds very different from a struck piano string, for example. Shape and material together determine a characteristic frequency spectrum.

2. *The location of the impact.* The timbre of the sound, i.e., the amplitudes of the frequency components, depends on where the object is struck. For example, a table struck at the edges makes a different sound than when struck at the center.

3. *The material of the struck object.* The harder the material, the brighter the sound. We also relate the material to the decay rate of the frequency components of the sound through the internal friction parameter, see below.

4. *The force of the impact.* Typically, the amplitude of the emitted sound is proportional to the square root of the energy of the impact.

All these factors give important cues about the nature of an object and about what is happening to an object in the simulation.

As an illustrative example, consider an environment where a user interacts with a metal bar by hitting it with a (virtual) hammer. When the user hits the metal bar, we want to synthesize the appropriate sound. Obviously, whatever processing we do after the impact will have to be done very fast, or unrealistic latency will appear. (Typically, for musicians, 20ms latency is considered acceptable, 3ms would be ideal. [1])

[1] Andy Schloss, private communication.

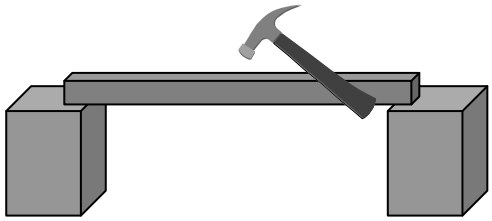

Figure 1.2: Interactive environment: a hammer hitting a bar.

A simple approach is to record a sample of an actual bar being struck, and load this sample in memory. When a strike event occurs the sample is then sent to the audio hardware.

But in this approach the sound will always be the same, no matter how hard or soft the user strikes the bar. One solution would be to record a set of samples for various levels of impact. At the impact event the sample that corresponds best with the actual force will be rendered. Another solution would be to change the amplitude of the sample in real-time before rendering it, to correspond to the volume of the sound for a particular strike force. This is generally such a low cost operation that it can be done in software even on very modest equipment.

This is a very simple example of manipulating a sound by processing it with a parameterized "effect", in this case the volume. A single "prototype" sound is changed in real-time to create many sounds. This is also the simplest example of parameterized synthesis. Synthesis by physical modeling is presently a very active topic of research in computer music [20, 67], but the name is somewhat misleading. The focus of most musical instruments research is not so much on the physical modeling, but in *parameterizing* sound with physical events. The parameters of interest to musicians for a simulated wind instrument are breath pressure, finger holes, speed of closure of finger holes, mouth position (for flute), and tongue position

(for ney). A "physical model" of a wind instrument in this context is a synthesizer that can create sounds that are parameterized by this set of physical characteristics.

In this case the parameter is the force of the impact, which is translated into audio volume. One may argue whether this deserves the name "synthesis", but it should be clear that there is a continuum between "samples with effects" and "synthesis". In this case we are synthesizing the volume, using a sample for the basic waveform. When using more complicated effects such as 3D spatialization and reverberation effects, we add more synthesis, but still use some recorded data.

However, by doing this we have sacrificed some level of realism, because we assume that the only effect of the force of the impact is a change in volume. This may be true in a first approximation, but non-linear effects cause the "timbre" of the sound to change also with the strike force. So we have sacrificed some level of detail by reusing a sample.

Let us continue with the example of the virtual bar. What if we allow the bar to be moved or allow the user to listen from different locations? This could be incorporated in the synthesis by filtering the output of our synthesizer through a set of HRTF filters (see 2.3). This can be done with off the shelf hardware or software. It can be thought of as an independent "multiplexer". We now have three parameters: volume, azimuth, and elevation. (We are ignoring room acoustics here.)

What if there were multiple bars in the scene? If the bars differ only in length, their sound spectra are approximately related through a simple scale factor in time, i.e., we can change the "pitch" of the stored sample depending on the length of the struck bar.

Many digital sampler synthesizers emulate musical instruments based on this principle. Typically, a group of tones will be represented by one sample, which is

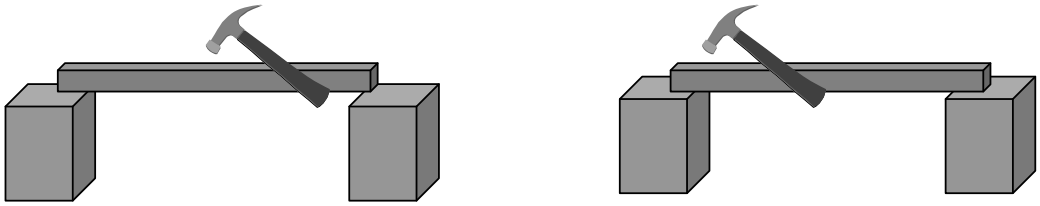

Figure 1.3: Interactive environment: a hammer hitting a bar at different locations.

played back at various rates. Because the timbral characteristics of instruments vary a lot over the entire range of the instruments, the recorded sample can only be transposed (scaled) to a certain extent, before it starts to sound "unnatural".

So far, we have been successful in synthesizing the sounds of different sizes of bars, at different spatial locations, which can be struck with any force. All these sounds can be obtained from a single recorded sample.

What if we strike the bar in the middle, and then near an end, as depicted in Figure 1.3? The timbre of the sound should change, depending on the strike point, because this will excite different resonance modes in different proportions. The same effect is well known in music; for example a violin will sound much more nasal when bowed near the bridge (which supports the strings).

A linear model (developed in detail in this thesis) predicts that the intensities of the partials (constituent sinusoidal waves) of the sound change as the strike point is changed. However, their frequencies remain the same, because the same vibration modes are excited.

To deal with this, we can record some samples of the sounds corresponding to various impact locations, and play back the closest one, or do a linear interpolation between two. [2]

[2] This could be called "sound morphing". It can be done in the time domain because the spectral content of the two sounds is the same.

But what if we now want to have different hammers, made out of metal, wood, and rubber? The sound will be different when hitting the bars with these different sticks. Should we now also pre-record the samples of the bar being struck by different sticks? If we want to have 10 different materials, 10 impact locations, and 10 different length bars, we would need 1000 samples! This clearly pushes straightforward sample playback to its limits.

The sample based approach does not require much modeling, and allows the use of recorded sounds, which will always (or at least for a very long time) be better than synthesized sounds. On the other hand, since recorded samples are not available to everyone, and are difficult to obtain, it restricts a designer of a virtual reality environment. A synthesis approach has the advantage that no external data is needed, but it also requires detailed physical modeling. Especially for complicated structures, not enough data may be available to model the vibrations accurately. However, in such a case one can use an experimental approach, and use recorded sounds to extract the relevant synthesis parameters empirically from real objects.

If we also want realistic continuous sounds when we scrape the bar with a stick, with velocity, direction, and speed under real-time user control, we can no longer use sample playback directly. One can imagine looping a sample at different speed/volume depending on the interaction, and this has been tried with limited success, but this does not take into account the resonances of the touched object. For these types of sound we need to synthesize the sounds in real-time.

1.3 Contributions of this Thesis

The work presented in this thesis is the first coherent effort to develop a physics based real-time audio synthesis methodology for environmental sounds caused by

interacting solid objects. We develop a real-time synthesis technique specifically tailored to the synthesis of sounds of solid objects under external forces. We address model rendering (audio synthesis), model authoring, interaction modeling, and software and system design issues. Using the techniques developed here, it is possible to generate realistic, interactive audio in simulation or games applications using only a small fraction of available CPU power on modern personal computers.

The remainder of this thesis is organized as follows. In Chapter 2 we discuss general background material relevant to audio synthesis and environmental sound simulation, and we describe related work in this field.

In Chapter 3 we summarize the physical laws which govern acoustics and which are the foundation upon which audio synthesis by physical modeling is built.

In Chapter 4 we derive a physics based parameterized vibration model for physical objects from the linearized vibration equations for solid bodies. A real-time synthesis algorithm is developed which allows the synthesis of the sound of such objects under any kind of interaction force.

In Chapter 5 we investigate methods for obtaining the parameters of the vibration models from mathematical models of simple geometries, by automatic parameter fitting to measured data, or by empirical means. Studies were conducted of several objects which were successfully modeled with these techniques.

In Chapter 6, models for interaction forces during contacts between rigid bodies (impulsive forces, and sliding interactions) are developed, as well as models for the driving forces for combustion engines and avalanches.

In Chapter 7 we describe the software tools that were built to implement the methodology developed. An object-oriented application programming interface (API) is developed to interface the audio synthesis to user code, and implementations

in C++ and in Java are described. Several computer programs were developed for the testing of models, for the construction of models, and for the demonstration of the level of realism that can be achieved with this type of synthesis.

In Chapter 8 we present our conclusions and outline directions for future research.

Some technical details such as the numerical results of parameter fitting, and file formats are relegated to the appendices.

It is hard to describe audio verbally or graphically, and many of our results should be heard in order to be appreciated. We have made several audio examples as well as some interactive applications available online on [4]. This material is also available on the accompanying CD.

Chapter 2

Background

In this Chapter we will discuss some general topics that are relevant to sound simulation, and describe related work in this field. The topics considered are

- Sound perception;
- Digital representation of sound;
- Environment modeling;
- Computer Music.

2.1 Sound Perception

To simulate sounds, we need to know some things about how humans perceive sounds, since we do not want to waste effort on aspects of sound that are not important perceptually. Knowledge of what is perceptually important may focus sonic modeling on relevant phenomena.

The human ear can detect frequencies in the range $20 - 20,000$ Hz and is most sensitive in the $500 - 6,000$ Hz region. The threshold of hearing is around $1dB$

(see Section 3.3 for a definition of dB), the threshold of pain is at about $140dB$. A normal conversation at a distance of 1 meter is about $50dB$. The sensitivity of the ear at various levels of loudness can be depicted in sensitivity curves, see for example [73]. When a sound is heard, various attributes are usually associated with it by the human brain. For example one may hear it as a liquid sound, or as a distant animal call, or as a metallic percussive sound coming from a specific direction. An attempt at classifying sounds into categories (like liquid, metallic, etc.) was made in [30].

The study of how sounds are processed by humans and grouped into auditory streams, is called auditory scene analysis, and there exists a standard work on the subject [15].

Considerable recent attention has focused on the localization of sound in three dimensional space, i.e., on how we perceive sounds as coming from a specific direction. A good review of the subject is [11]. In [94] psychophysical experiments are described which are used to investigate how inter-aural time delays provide cues for left-right localization. A good review of psychological aspects of sound localization by humans can be found in [50].

For wavelengths greater than the size of the human head, the horizontal position cues come from a phase difference in the waves at the ears. These waves don't "see" the head and are therefore unaffected by it. Smaller waves are scattered by the head and by the pinna. The primary cue for position at the onset of sound with small wavelengths comes from the inter-aural delay time, and the manner in which the sound is altered (filtered) by the head and the pinna. The brain has learned to correlate these scattering effects with the position of the source. The vertical position of a sound can also be perceived, though not as accurately. Cues

for this come entirely from the filtering of the sound through the head and pinna. Percussive sounds are often perceived as having a distinct "onset", and a sustained part. Localization cues obtained at the onset tend to be important.

The reverberations of the room also have a role in sound localization. Without reverberations it is harder to localize sounds. Different frequencies decay (attenuate) at different rates in the atmosphere. This is why sounds coming from a great distance can be heard as such, as they sound "muffled". Note that the propagation of sound through the atmosphere is strongly influenced by the weather, giving a distinct aural "feel" to various kinds of weather.

Blind people report that they can "feel" the presence of objects like a wall, as a pressure on the face. I can reproduce this sensation also to some extent by moving my head close to a wall in the dark. What actually happens is that small sounds like that of your breathing reflect off nearby surfaces, which is detected by your ears but somehow perceived as a kinesthetic sensation.

2.2 Representation and Rendering of Computer Sound

Various techniques exist to record and reproduce sound. For example, the sound can be stored as a physical image of the vibrations in a gramophone, or as a pattern of magnetization on a tape. These are analogue techniques, as they map pressure variations directly to some other physical characteristics. It has even been speculated that sound can leave imprints by accident, for example on drying paint, and might be reconstructed [93].

A different method of storing sound is by digital sampling. The signal is sampled at some rate and encoded as a set of numbers, which are typically stored on some electronic device. Digitally stored sounds can be manipulated algorithmically

and recordings can be duplicated without any loss in quality. A good introduction to digital audio techniques is [72].

The signal at each ear will be represented by a scalar function of time which, apart from a multiplicative factor, specifies completely the perception at the ear. Two such signals would give a complete description of a sound as perceived by a human. There are also phenomena which are usually classified as "sound" that are perceived not directly through the ears. For example, the deepest tones of a large church organ are "felt" in the chest cavity.

The signal at the ear could represent the pressure at the entrance to the ear canal, the deviation from equilibrium of the eardrum, or perhaps other relevant parameters. How this function is precisely to be translated into a physical rendering of the sound, with the aid of speaker, headphones, or other means, will depend on the setup of the speakers or on the headphone characteristics. But it should be clear that in principle any sound can be created in this manner.

With loudspeakers we have the complication of "leaks" between the speakers. A simple minded setup with a speaker for the left ear and a speaker for the right ear will not work, as the right ear will also pick up the signal from the left speaker. But one could for example cancel out the "leak" from the left speaker to the right ear with an additional signal from the right speaker that had the opposite phase. Unfortunately, this requires a knowledge of the position of the listeners ear. Various techniques, sometimes called "Surround Sound", "Dolby Surround", or "3D sound", exist to partially overcome this problem.

The signal function is represented by a digital *sample*. The values of the function are stored at time intervals of $1/S_R$, where S_R is the sampling rate. The values can be given as floats or as 16 or 8 bit signed integers. The method of

sampling is usually indicated by specifying the sample *width*, which is the number of bits used to represent the value, and the sampling *rate*.

To represent sounds with frequencies up to f Hz, the Shannon Sampling Theorem [61] tells us that we must have

$$S_R > 2f,$$

for an accurate representation of the function. If $S_R < 2f$, frequencies above f will "fold back" into low frequencies and produce distortions. The human ear can perceive frequencies of up to 20,000 Hz, [73]. A common sampling rate for sound (used also for commercial CD quality recordings) is 44,100 Hz. The lowest sampling rate in common use is 8,000 Hz, which is used often in Internet applications.

A wide variety of file formats is available for the storage of digital samples. Some widely used formats are

au Used by SUN and currently the only format supported by Java. (Java version 1.1.) This format stores the wave form logarithmically.

wave This format can use a variety of encoding techniques, but most commonly is used with linear encoding. It is used on PC's.

AIFC The format used by SGI. A linear encoding with a very complicated header structure with descriptions of looping etc. This is an elaboration of the older AIFF format.

IFF The format used by Amiga.

Most computers have some hardware to play a stereo sample in real time i.e., to output an electric signal that can be used with speakers or headphones to

generate sound. The sample will typically be located at some memory location of the computer.

Depending on the available hardware, it is possible to dynamically manipulate a sample and change certain aspects of it in real time. Many things can be done in software, but often one uses special DSP hardware to do the signal processing in real time. Examples of such processors are the readily available "reverb" units, which add simulated room acoustics and other effects to an input signal in real time. Also emerging are "spatialization" engines, which modify a signal to position it at a specified location in space. With increasing processor speeds there is a tendency to assign more and more DSP tasks to the main CPU.

2.3 Environment Modeling

Suppose we have a virtual reality environment in which we want to compute the sound perceived by an observer. That is, we want to generate a signal that drives speakers or headphones that will lead to a correct perception of the sound.

In principle one should model the sound generating objects, compute the resulting sound field at the eardrums of the simulated observer, and generate a signal at the headphones or speakers such that the eardrum of the human subject is exposed to the sound field computed.

To make this a somewhat manageable task, it is usually assumed that the sources of the sound are sufficiently far from the observation point that we can approximate the acoustic wave by a plane wave. We can then replace the sound source with a suitable point source. We can then divide the simulation in three parts: source modeling, sound propagation (reverberation), and sound reception. By "reception" we mean the scattering of the sound waves around the head and

ears, which needs to be simulated if we want the sound to appear to come from definite spatial locations. The vibrational modeling to compute the point source is discussed in Sections 3.5 and 3.6. However, we can also record a sample of a real sound. This is an approach often taken, as more is known about modeling the propagation (reverberation) and reception (the scattering around the head and ears) than about modeling the sound production.

Reverberation modeling involves computing reflections of the sound from surfaces along the path from the source to the observer. For this we need to have a good model of scattering of sound from materials. Some of the relevant physics will be discussed in Section 3. Off-line computation of acoustical properties of performance halls, in the context of graphical visualization techniques was investigated in [74].

When the sources and observers are not stationary, the Doppler shift changes the rate at which the wave arrives at the ear, which can be simulated by changing the effective sampling rate of the stored sample.

The filtering of the sound at the ears and the head, which is direction dependent, can be summarized by a set of direction dependent filters that simulate the ears. These are called Head Related Transfer Functions (HRTF) and have received much attention recently. The basic idea is to measure the sound inside the ear for a standard source (usually white noise) positioned at various locations with respect to the subject. The relation between the source and the measured sound is then extracted as a finite impulse response filter (FIR) for every direction. These are the HRTF filters. On playback, a given "dry" sound can then be spatialized by convolving it with the appropriate HRTF.

A review of the scientific and technological issues of auditory displays can

be found in [26]. Takala and Hahn introduced the concept of sound rendering for computer animation [80]. They associated a characteristic sound with each object which could then be rendered after filtering the sound to model the environmental effects. Recently [31] they proposed "Timbre Trees," which, like shade trees, provide a scene description language for sounds. In [17] complex sounds such as breaking events are analyzed and decomposed into individual components which can then be reconstructed with adjustable parameters to obtain a form of parameterized synthesis.

2.4 Computer Music

Computer music has been around for a long time, but has traditionally been more concerned with imitating the sounds of musical instruments, or with the creation of totally new sounds, than with the reproduction of everyday sounds. Surprisingly little can be learned from computer music that can be used in the synthesis of simple sounds such as emitted by an object of known material and geometry being struck or scraped.

Several synthesis techniques have been developed or are currently being investigated, which have some relevance to the synthesis of natural sounds: A comprehensive overview of musical synthesis techniques is given by Herbert Janßen, on the WWW site [7]. As this material is not published anywhere else and may be removed from this WWW site, we include a compilation of relevant material from that source with the author's permission in Appendix C.

In [36] audio synthesis is discussed from a more theoretical point of view and synthesis techniques are analysed using various criteria.

Many of the synthesis techniques described there are based on specific al-

gorithms or specific hardware configurations to create periodic oscillations. Some methods are general enough that they could be used to generate sound for some of our purposes.

One such technique is *additive synthesis*, which builds a complex sound from sinusoidal components with controllable amplitudes. Unfortunately, there is no obvious way to compute the amplitudes of the sinusoidal components of contact sounds, except for struck objects, where they decay exponentially.

Sample playback is currently the only technique used for environmental sounds in games and virtual reality. It has the capability of producing "photo-realistic" sounds, but with little or no interactivity. For discrete sounds such as impacts this is an excellent method.

Granular synthesis [82] superimposes many short fragments of recorded samples in a controllable stochastic manner. It has some applications in the generation of crowd sounds in computer games and seems to be well suited for other sounds which involve a large number of events, such as rain.

Waveguides [67] provide models of one dimensional structures such as vibrating strings and air columns. Because such structures have a harmonic spectrum of resonance frequencies, waveguides (a.k.a. comb filters) provide a very efficient means of synthesis and are used for musical instruments. The disadvantage of waveguides is that one has no control over the bandwidths and amplitudes of the resonances. This synthesis method is not suited for the sound of vibrating solid bodies, which usually have a non-harmonically spaced frequency spectrum.

Modal synthesis [52] has been used for percussive instruments with a non-harmonic frequency spectrum such as marimba and bells. Because each resonance requires additional computation, this synthesis technique is not well suited for har-

monic musical instruments which usually need many resonances. However it is very well suited for our purposes, and in the following chapter we shall show that this synthesis technique can be derived as an implementation of a real-time solution to the linearized vibration equations for solid bodies.

Several research groups are actively pursuing the physical modeling approach to music:

- The Center for Computer Music and Acoustics, CCRMA, at Stanford is developing physical models of musical instruments. They use mainly wave-guides for their modeling, but they also have done some work on modal synthesis. Musical instruments are modeled as filter banks to which an appropriate stimulus is applied. An overview of work at CCRMA can be found on their WWW site [1].

- The Center for New Music and Audio Technologies, CNMAT, at Berkeley is developing a synthesis technique based on additive synthesis. A sound is described in frequency space where a number of dominant partials are identified with a given time evolution. An efficient algorithm based on the FFT allows this description to be inverted in real time for sound synthesis [24]. An overview of the research at CNMAT can be found at the WWW site [3].

- The Institut de Recherche et Coordination Acoustique/Musique, IRCAM, in their Modalys program has investigated synthesis techniques using modal models. Their interest is primarily in the synthesis of musical sounds and in the specification of control algorithms, rather than in real-time techniques. Their web site is [6] and their scientific publications are available on-line at [2].

Chapter 3

Physics of Sound

In this chapter the laws governing the propagation of pressure waves in gases and the interaction of solids with these waves will be derived from fundamental physical laws. This allows a clear understanding of the nature of the approximations involved in the derivation, and thus shows under what conditions these simplified laws are valid and where they might break down.

We will not consider any conditions under which the simplest linear wave model of sound in air breaks down in this thesis, but this is a possible direction for future research. The material in this chapter indicates how the construction of more complicated equations for pressure wave propagation might proceed. It provides enough background material to allow the reader to follow such a derivation.

Material in this chapter can be found in many standard works on acoustics, see for example [13, 54, 45, 81]. We follow the notation of [81] here.

In the last section of this chapter we will give a brief introduction to the theory of vibrating solids.

3.1 Sound Propagation

For sound propagation through fluids and gases we shall use the continuum model for fluids. Such a model will be applicable when the *Knudsen number* is small enough. The Knudsen number is defined by

$$K_n = \Lambda / l, \tag{3.1}$$

where Λ is the mean free path, which is the average distance that a molecule travels between collisions with other molecules and l is the length scale of the phenomenon of interest, such as the wavelength of sound.

For example, a sound wave with a frequency of 20,000 Hz (at the limit of the audible frequency range) has a wavelength of 1.7 cm. The mean free path in air is $5.9510^{-8}m$. The Knudsen number is 3.5×10^{-6}, so the continuum model is applicable in this case.

3.2 Fundamental Equations for Gases

We shall present the fundamental equations that govern the propagation of sound. Derivations and more detailed discussions of the thermodynamic background can be found in [81].

The local state of a simple fluid (whose composition is uniform) can be specified by two quantities. For example, at low pressures all gases obey the ideal gas law,

$$p = R\rho T \tag{3.2}$$

where R is the gas constant, T the absolute temperature, ρ the mass density of the gas, and p the pressure. The local state of the gas can be described by ρ and

T, or by p and ρ, etc.

For gases in motion, we can formulate the law of conservation of mass in terms of the mass density field $\rho(\boldsymbol{x}, t)$ and the velocity field $\boldsymbol{u}(\boldsymbol{x}, t)$ as

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x^i}(\rho u^i) = 0. \tag{3.3}$$

To derive equations of motion for the gas we must specify a model for the forces and stresses acting on an infinitesimal volume of the gas. External forces on the gas are described by a vector field $\boldsymbol{F}$, which represents the force per unit volume acting on the gas.

Stresses are described by the symmetric *stress* tensor $\boldsymbol{\sigma}$ of rank two. The component σ_{ij} is interpreted as the i-th component of the force per unit area acting on an infinitesimal surface element perpendicular to a vector in the j direction.

For a Newtonian fluid, the fluid is governed by the Navier-Stokes equation plus thermodynamic equations. Most gases and simple fluids can be accurately described by the Newtonian fluid model. Fluids with complicated molecular structure can not, but are not considered here.

For a Newtonian fluid, the stress tensor is given by

$$\sigma_{ij} = -P\delta_{ij} + d_{ij}, \tag{3.4}$$

where δ_{ij} is the unit tensor. The *deviatoric stress tensor* d_{ij} is traceless (i.e., $d_{ii} = 0$). The quantity P is called the *mechanical pressure*. For small velocity gradients, the deviatoric stress tensor is given by

$$d_{ij} = 2\mu(e_{ij} - \frac{1}{3}\Delta\delta_{ij}) \tag{3.5}$$

where μ is the coefficient of viscosity (also called shear viscosity) of the gas and e_{ij}

is the rate-of-strain tensor, given by

$$e_{ij} = \frac{1}{2}(\frac{\partial u_i}{\partial x^j} + \frac{\partial u_j}{\partial x^i}). \tag{3.6}$$

The local rate of expansion Δ is defined as

$$\Delta = e_{ii}. \tag{3.7}$$

The difference between the mechanical pressure P and the thermodynamic pressure p is related to the local rate of expansion through

$$p - P = \mu_v \Delta \tag{3.8}$$

where μ_v is the *expansion coefficient of viscosity.*

For reference, we will now give the complete set of equations for a Newtonian gas assuming that temperature variations are small and that μ_v is constant.

1. Continuity:

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x^i}(\rho u^i) = 0. \tag{3.9}$$

2. Navier-Stokes:

$$\rho(\frac{\partial u^i}{\partial t} + u^j \frac{\partial u^i}{\partial x^j}) = \rho F^i - \frac{\partial p}{\partial x^i} + \mu(\frac{\partial^2 u^i}{\partial x^j \partial x^j} + (1/3 + \mu_v/\mu)\frac{\partial}{\partial x^i}(\frac{\partial u^j}{\partial x^j})). \tag{3.10}$$

3. Energy:

$$T\frac{\mathcal{D}S}{\mathcal{D}t} = c_p \frac{\mathcal{D}T}{\mathcal{D}t} - \frac{\beta T}{\rho}\frac{\mathcal{D}P}{\mathcal{D}t} = \Phi + \frac{1}{\rho}\frac{\partial}{\partial x^i}(k\frac{\partial T}{\partial x^i}) + \frac{\mu_v}{\rho}\Delta^2. \tag{3.11}$$

4. State:

$$p = p(\rho, S). \tag{3.12}$$

Here we have defined

$$\frac{\mathcal{D}}{\mathcal{D}t} = \frac{\partial}{\partial t} + u^i \frac{\partial}{\partial x^i}. \tag{3.13}$$

The viscous dissipation function Φ is defined by

$$\Phi = \frac{2\mu}{\rho}(e_{ij}e_{ij} - \frac{1}{3}\Delta^2). \tag{3.14}$$

The specific heat is denoted by c_p and k is the thermal conductivity of the gas. The coefficient of thermal expansion, β, is defined by

$$\beta = \frac{1}{v^*}\left(\frac{\partial v^*}{\partial T}\right)_p \tag{3.15}$$

with

$$v^* = 1/\rho. \tag{3.16}$$

For the important case of an ideal fluid, the stress force is assumed to be perpendicular to an infinitesimal surface element, and to have the same magnitude independent of the direction of the normal to the surface element. In this case the equations for the gas can be written as

$$\frac{\partial p}{\partial t} + u^i \frac{\partial p}{\partial x^i} + \rho c^2 \frac{\partial u^i}{\partial x^i} = 0 \tag{3.17}$$

and

$$\rho(\frac{\partial u^i}{\partial t} + u^j \frac{\partial u^i}{\partial x^j}) + \frac{\partial p}{\partial x^i} = \rho g^i, \tag{3.18}$$

where c is the local speed of sound. Thermodynamic considerations lead to $c^2 = (\frac{\partial p}{\partial \rho})_S$. The gravity vector is given by g^i.

3.3 Linearized Equations for Acoustic Waves

For acoustic waves in an ideal gas we linearize Equations 3.17 and 3.18. We consider small fluctuations of pressure p and density ρ around the background values p_0 and

ρ_0. If the length scale for variations in velocity is much smaller than c_0^2/g, with c_0 the velocity of sound and g the absolute value of the gravity vector, the linearized equations are

$$\frac{1}{\rho_0(\boldsymbol{x})c_0^2(\boldsymbol{x})}\frac{\partial p}{\partial t} + \frac{\partial u^i}{\partial x^i} = 0, \tag{3.19}$$

$$\frac{\partial u^i}{\partial t} + \frac{1}{\rho_0}\frac{\partial p}{\partial x^i} = 0. \tag{3.20}$$

For uniform gases we can rewrite the linearized Equations 3.19 and 3.20 in a different form. First, observe that the *vorticity* $\boldsymbol{\omega}$,

$$\omega^i = \epsilon^{ijk}\frac{\partial u^j}{\partial x^k} \tag{3.21}$$

is conserved. ϵ^{ijk} is the antisymmetric rank-3 tensor (with $\epsilon^{123} = 1$). If we assume that the gas is at rest initially, we will have $\boldsymbol{\omega} = 0$. For velocity fields with zero vorticity, we can write

$$u^i = \frac{\partial \phi}{\partial x^i} \tag{3.22}$$

where ϕ is the *velocity potential*. The acoustics equations can now be formulated in terms of a single wave equation for ϕ,

$$-\frac{\partial^2 \phi}{\partial t^2} + c_0^2\nabla^2\phi = 0 \tag{3.23}$$

with $\nabla^2 = \frac{\partial^2}{\partial x^i \partial x^i}$. The pressure fluctuation p can also be obtained from the velocity potential by

$$p = -\rho_0\frac{\partial \phi}{\partial t}. \tag{3.24}$$

The energy density of the acoustic field is given by

$$E = E_k + E_p \tag{3.25}$$

where the kinetic energy is given by

$$E_k = \frac{1}{2}\rho_0 u^i u^i \tag{3.26}$$

and the potential energy is given by

$$E_p = \frac{1}{2\rho_0 c_0^2} p^2. \tag{3.27}$$

In terms of the velocity potential ϕ, this becomes

$$E_k = \frac{1}{2}\rho_0 \frac{\partial \phi}{\partial x^i}\frac{\partial \phi}{\partial x^i} \tag{3.28}$$

and

$$E_p = \frac{\rho_0}{2c_0^2}\frac{\partial \phi}{\partial t}^2 . \tag{3.29}$$

The *energy-flux vector* (the energy flow per unit area) is given by

$$\boldsymbol{q}_a = p\boldsymbol{u} \tag{3.30}$$

and its time average

$$\boldsymbol{I} = < \boldsymbol{q}_a > \tag{3.31}$$

is called the *acoustic intensity*. Its dimension is power per area, e.g. watt/m^2. It is customary to express the acoustic intensity in *decibels* (dB). For this we define a reference level

$$I_{ref} = 10^{-12} watt/m^2. \tag{3.32}$$

It corresponds to the lowest intensity at which a sine wave of 1000 Hz can be heard by the average human. The intensity level in decibels is given by

$$IL = 10 \log_{10}(I/I_{ref}). \tag{3.33}$$

We note that the linearized equations for acoustic waves can not deal with dissipation. For this we must use the Newtonian fluid model, described in Section 3.2. For sound waves in air, the decay rate is proportional to the frequency. The amplitude decay over 1000 wavelengths (340m at 1000 Hz) is only 1-2% in air, so it can be neglected in many applications.

3.4 Interaction of Acoustic Waves with Solids

Sound waves are reflected (scattered) by solids. Vibrating solids generate sound waves. Both phenomena require an understanding of the fluid dynamics at the boundary between the two media.

When a sound wave hits a surface, the surface will move in response. If the surface is part of the boundary of a solid there will be waves passing through the solid, and the solid will emit sound from its entire boundary. However, in many situations the propagation of sound inside the solid can be ignored and we have to deal only with reflection and absorption at the surface. How much of the wave is absorbed and how much is reflected depends on the material properties of the surface.

Since in general we do not know the dynamical equations of the surface, some sort of phenomenological model is necessary. The *specific acoustic impedance* of a boundary is used to characterize a surface as follows. We assume that the velocity of the boundary depends on the pressure variation of the air at the boundary with some time lag. For a monochromatic incident wave we assume the relation

$$u_n = p/z_a, \tag{3.34}$$

where u_n is the surface normal component of the velocity of the surface and p is the pressure fluctuation at the boundary. The specific impedance z_a depends on the frequency of the incoming wave and is complex in general (inducing a phase difference, i.e., a time lag, between the surface response and the pressure).

Some classical topics that can be solved analytically are the sound field in a piston driven tube and transmission of waves through tubes as a function of the tube diameter. An example of the latter is the human vocal tract. The vowel sounds

used in human speech are produced by changing the shape of the vocal tract with the throat and tongue, thereby changing the transmission of the sound radiated by the vocal chords, see [21].

The wave equation (Equation 3.23) has been studied thoroughly. The study of spherical waves leads to an exact solution for the sound field of a pulsating sphere. The sound field of a bursting balloon can be computed analytically. This is an important topic, because one may consider a small pulsating sphere as a "point-source" of sound leading to a field theory approach of sound. Such a point source plays a similar role as the point charge and point mass do in electromagnetism and gravity. Just as a finite body can be considered as an infinite collection of point masses, so a complex sound source can be considered as an infinite collection of point sound sources. The advantage of this approach is its similarity to the well studied fields of electromagnetism and gravity.

3.5 Sound Generation

Vibrating bodies generate sound waves through boundary conditions imposed on the wave equation (Equation 3.23) at the surface of a solid body. For an ideal fluid the boundary condition is

$$\boldsymbol{u}.\boldsymbol{n} = \boldsymbol{U}_b.\boldsymbol{n} \tag{3.35}$$

with $\boldsymbol{n}$ the surface normal and $\boldsymbol{U}_b$ the velocity of the surface. However, real gases "stick" to the surface, in which case we have

$$\boldsymbol{u} = \boldsymbol{U}_b. \tag{3.36}$$

For an ideal gas, the condition given in Equation 3.35 has to be used, as viscous effects are not taken into account in the ideal gas model.

A particularly useful formulation of the equations governing sound generation is the formulation in terms of Green's functions. An exact computation of the far field generated by a pulsating sphere leads to the following simple source potential for a point source (i.e., a source of volume) that is pulsating (with frequency ω)

$$\phi = -\frac{Q_0}{4\pi r} e^{i(kr-\omega t)} \tag{3.37}$$

with $k = \omega/c_0$ and r the distance from the source (which is located at the origin). The quantity Q_0 represents the volume flow out of a small sphere enclosing the source. The intensity of the acoustic wave described by Equation 3.37 is

$$I = \frac{\rho_0 \omega^2 Q_0^2}{32\pi^2 r^2 c_0} \tag{3.38}$$

and the total acoustic power is

$$\Pi = \frac{\rho_0 \omega^2 Q_0^2}{8\pi c_0}. \tag{3.39}$$

The solution 3.37 can be viewed as the solution of the inhomogeneous wave equation

$$-\frac{\partial^2 \phi}{\partial t^2} + c_0^2 \nabla^2 \phi = c_0^2 Q(\boldsymbol{x}, t) \tag{3.40}$$

which should be compared with Equation 3.23. The general solution to Equation 3.40 can be given as

$$\phi(\boldsymbol{x}, t) = -\frac{1}{4\pi} \int_V \frac{Q(\boldsymbol{x}', t - \frac{|\boldsymbol{x}-\boldsymbol{x}'|}{c_0})}{|\boldsymbol{x} - \boldsymbol{x}'|} dV(\boldsymbol{x}'). \tag{3.41}$$

An important case is a surface distribution of the source Q. For a plane, the result is

$$\phi(\boldsymbol{x}, t) = -\frac{1}{2\pi} \int_{Surface} \frac{\partial \phi}{\partial n} \frac{e^{ik|\boldsymbol{x}-\boldsymbol{x}'|-i\omega t}}{|\boldsymbol{x} - \boldsymbol{x}'|} dA(\boldsymbol{x}'), \tag{3.42}$$

where $\frac{\partial \phi}{\partial n}$ is just the normal velocity of the surface.

Unfortunately, Equation 3.42 can not be used directly to compute the sound emitted by a vibrating polyhedron. The problem is that the sound emitted by one facet of the polyhedron is dependent on the overall shape of the polyhedron.

Some special problems can be analyzed analytically. An example is the sound field of an oscillating piston in an infinite wall.

The problem of interest for simulating the sound field of vibrating solids leads to an integral equation for ϕ. This equation, known in slightly modified form as the *Kirchhoff-Helmholtz* equation [46] plays a fundamental role for an analysis of the emission of sound by vibrating bodies. It is derived in a particularly clear way in [81]. The end of [27] is devoted to a discussion of problems and numerical techniques associated with solving the Kirchoff-Helmholtz equation. One problem with the equation is that it breaks down if there is *resonance* in the system. At certain driving frequencies the amplitudes of the waves become infinite (as we have not incorporated dissipation in our model) and the solution of the integral equation diverges. At resonance, dissipation must be modeled. In this case the amplitudes often grow large enough that nonlinear effects become important. In such a case, adequate physical models are often not available. There is a section in [81] on nonlinear effects. Nonlinear acoustics is an important field of research, see for example [12]. The study of nonlinear waves (see for example [90]) is an extensive research field.

We will now state the Kirchoff-Helmholtz equation. The free-field Green's function (which is essentially the sound field of a harmonic point volume source with frequency ω) is defined by

$$G(\boldsymbol{x},\omega) = \frac{e^{ik|\boldsymbol{x}|}}{4\pi|\boldsymbol{x}|} \tag{3.43}$$

where $k = \omega/c_0$.

We consider a region $\mathcal{V}$ bounded by surface $\mathcal{S}$, which does not have to be

connected. We can think of $\mathcal{S}$ as consisting of an outer boundary like the walls of a room and some boundaries of objects in the room which produce sound.

Let $\boldsymbol{n}(\boldsymbol{x})$ denote the outward normal (i.e., pointing into the region $\mathcal{V}$) at point $\boldsymbol{x}$ on $\mathcal{S}$. We assume that every point $\boldsymbol{y}$ on $\mathcal{S}$ is oscillating harmonically with frequency ω and velocity $\boldsymbol{v}(\boldsymbol{y})$. The velocity potential is written as

$$\phi(\boldsymbol{x}, t) = \int_{-\infty}^{\infty} \phi_\omega(\boldsymbol{x}) e^{i\omega t} \frac{d\omega}{\sqrt{2\pi}}. \tag{3.44}$$

The time independent potential ϕ_ω satisfies

$$\phi_\omega(\boldsymbol{x}) = \int_{\mathcal{S}} \left[n^k(\boldsymbol{y}) \frac{\partial \phi_\omega(\boldsymbol{y})}{\partial y^k} G(\boldsymbol{x} - \boldsymbol{y}) - \phi_\omega(\boldsymbol{x}) n^k(\boldsymbol{y}) \frac{\partial G(\boldsymbol{x} - \boldsymbol{y})}{\partial y^k} \right] d^2S(\boldsymbol{y}) \tag{3.45}$$

where $\boldsymbol{n}(\boldsymbol{y})$ is the outward normal at point $\boldsymbol{y}$ on $\mathcal{S}$, and $d^2S(\boldsymbol{y})$ is an infinitesimal surface element on $\mathcal{S}$.

The boundary condition for an ideal gas, Equation 3.35, gives us the normal derivative of ϕ on $\mathcal{S}$, which appears in the right side of Equation 3.45,

$$n^k(\boldsymbol{y}) \frac{\partial \phi_\omega(\boldsymbol{y})}{\partial y^k} = \boldsymbol{n}(\boldsymbol{y}).\boldsymbol{v}(\boldsymbol{y}) \tag{3.46}$$

with $\boldsymbol{v}(\boldsymbol{y})$ the velocity of $\mathcal{S}$ at $\boldsymbol{y}$. We can consider this a given quantity (computed from a model of the physics of the vibrations of the surface $\mathcal{S}$). The value of $\phi_\omega(\boldsymbol{y})$ on $\mathcal{S}$ is related to the pressure fluctuation $p_\omega(\boldsymbol{x})$ (in the frequency domain) through

$$p_\omega(\boldsymbol{x}) = -i\rho_0 \omega \phi_\omega(\boldsymbol{x}) \tag{3.47}$$

which follows from Equations 3.24 and 3.44. If the pressure on the boundary was known, Equation 3.45 would give us the acoustic field on $\mathcal{V}$.

By restricting $\boldsymbol{x}$ on $\mathcal{S}$ in Equation 3.24, and dividing $\mathcal{S}$ in n elements, we formally obtain a system of n equations for the n values of p on $\mathcal{S}$. However, there are some problems with this approach.

Observe that if $\boldsymbol{x}$ lies on $\mathcal{S}$ in Equation 3.45, the Green's function defined in Equation 3.43 is not defined for $\boldsymbol{x} = 0$. In this case one should take the principal values of the integrals at the singular points. This entails a regularization of the Green's function, by defining $G_\epsilon(\boldsymbol{x}) = 0$ for $|\boldsymbol{x}| < \epsilon$ and letting $\epsilon \to 0$ at the end.

If the volume $\mathcal{V}$ is finite, there may be no unique solution for the pressure on the surface, for some values of the frequency. The physical reason is that at *resonance* the acoustic field grows without limit. In this case a separate analysis is required to identify the resonance frequencies. We refer to [27] for more details on numerical approaches to the Kirchoff-Helmholtz equation.

3.6 Vibration Theory

Vibrating bodies are sources of acoustic waves and produce sound through the mechanism described in Section 3.5. In this section, we describe the equations governing their behaviour.

Suppose the configuration of a vibrating system can be described by a vector $\boldsymbol{q}(t)$ of n real numbers. At equilibrium, without motion, we assume $\boldsymbol{q} = 0$. A typical example is a set of rigid bodies connected through (damped) springs. A solid body such as a bar can be thought of as an infinite collection of infinitesimal rigid bodies connected through generalized springs. They can be approximately described by a discrete set of n coordinates $\boldsymbol{q}$, provided n is large enough. Another route to this conclusion is to note that the solution of the partial differential equation for the vibration of a bar can be written as a discrete sum of basic functions (a generalization of a Fourier transform series), which can be interpreted as the $\boldsymbol{q}$ in the above.

The vibrating system, for sufficiently small amplitudes $\boldsymbol{q}$, is described by the

linear equation of motion

$$\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{C}\dot{\boldsymbol{q}} + \boldsymbol{K}\boldsymbol{q} = \boldsymbol{F} \tag{3.48}$$

where the dots denote differentiation with respect to time and $\boldsymbol{F}(t)$ is the external force on the system. The matrices $\boldsymbol{M}$, $\boldsymbol{C}$, $\boldsymbol{K}$, and the vector $\boldsymbol{F}$ can be found for a given system by various means, depending on the system.

A case of particular interest to us is a system of many *Finite Elements* [58] that approximates a given solid. The continuous system is divided into a number of elements, each with appropriate elasticity properties and degrees of freedom. The elements are then connected to approximate the continuous object. The degrees of freedom corresponding to the finite elements obey an equation of the same structure as Equation 3.48, in the linear approximation. The last Chapter in [66] discusses Finite Element methods in vibration analysis.

We solve Equation 3.48 by writing

$$\boldsymbol{y} = \begin{pmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{pmatrix}. \tag{3.49}$$

The equation of motion 3.48 becomes

$$\dot{\boldsymbol{y}} = \boldsymbol{B}\boldsymbol{y} + \boldsymbol{r} \tag{3.50}$$

with

$$\boldsymbol{B} = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{1} \\ -\boldsymbol{M}^{-1}\boldsymbol{K} & -\boldsymbol{M}^{-1}\boldsymbol{C} \end{pmatrix} \tag{3.51}$$

and

$$\boldsymbol{r} = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{M}^{-1}\boldsymbol{F} \end{pmatrix}. \tag{3.52}$$

The solutions to Equation 3.50 with $\boldsymbol{F} = 0$ are

$$\boldsymbol{y} = \boldsymbol{a}e^{\mu t} \tag{3.53}$$

where the eigenvectors $\boldsymbol{a}$ (of dimension $2n$) and the eigenvalues μ satisfy

$$(\boldsymbol{B} - \mu \mathbb{1})\boldsymbol{a} = 0. \tag{3.54}$$

There are $2n$ eigenvalues, each representing an oscillation with frequency $\mu/(2\pi)$. The reason we have two solutions for every degree of freedom is that there is also a phase associated with this degree of freedom.

In the presence of an external force $\boldsymbol{F}$ the solution to Equation 3.50 can be written as

$$\boldsymbol{y} = \boldsymbol{T}(t)\boldsymbol{y}_0 + \int_0^t \boldsymbol{T}(t-\tau)\boldsymbol{r}(\tau)d\tau \tag{3.55}$$

where the *state transition matrix* $\boldsymbol{T}$ is given by

$$\boldsymbol{T} = \boldsymbol{\Phi}\boldsymbol{T}_1\boldsymbol{\Phi}^{-1} \tag{3.56}$$

where

$$\boldsymbol{T}_1 = \begin{pmatrix} e^{\mu_1 t} & 0 & \cdots & 0 \\ 0 & e^{\mu_2 t} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e^{\mu_{2n} t} \end{pmatrix} \tag{3.57}$$

and the $2n \times 2n$ *modal matrix* $\boldsymbol{\Phi}$ is given by

$$\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{A}_1 & \boldsymbol{A}_2 & \cdots & \boldsymbol{A}_{2n} \end{pmatrix} \tag{3.58}$$

with $\boldsymbol{A}_i$ the i-th eigenvector $\boldsymbol{a}$, i.e., a solution of Equation 3.54. $\boldsymbol{y}_0$ in Equation 3.55 is given by the initial conditions on the system.

We conclude that a vibrating system can be characterized by a discrete set of eigenvalues which correspond to the natural frequencies and their associated decay rates. When the system is excited by an external force of finite duration some of

these frequencies will be excited. The relative amplitudes after the application of the force depend on the nature of the excitation.

When an object is struck, the applied force is of very short duration and contains many frequencies. Very shortly after the strike, many frequencies of the system will be excited, but most decay very rapidly, leaving only a few. The object will be perceived as having a definite "pitch" if one frequency decays much slower than the others, such as is the case for a tuning fork, for example, or if the object has a harmonic spectrum, as is the case for many melodic percussive musical instruments such as the piano. The sound emitted when an object is struck is perceived as containing a "click" of very short duration, which is a mixture of many frequencies, and a sustained part, which contains only a few frequencies.

Continuous systems such as bars and plates lead to partial differential equations. For some simple systems exact solutions can be found. The classical problems that can be tackled analytically are the vibrating string, the rectangular and the circular membrane and plate, and bars, under various boundary conditions.

Chapter 4

Contact Sounds

In this chapter we investigate the computation and rendering of sounds emitted by solid bodies in contact. We will construct a parameterized vibration model that lends itself to real-time synthesis. Some of the material in this chapter has been published previously by Dinesh Pai and the author in [84].

The computation of sound emitted by an object to which a time dependent driving force is applied can be done in principle as follows.

1. Formulate the equations of motion of the object under external forces. In general, this will be a partial differential equation.

2. Solve the resulting system of equations in the presence of the driving force.

3. Determine the surfaces that are exposed to air, where sound will be emitted. Using the theory described in Section 3.5, determine the acoustic field at relevant locations. Usually this will be at the ears of the (virtual) observer. Note that we also have to take bounding surfaces such as walls into account. We also have to model the scattering of sound at the pinna (the outer ear),

and at the head and shoulders.

To make this into a manageable task, we can make a number of simplifying assumptions:

- Often the observer will be "far" from the source, so we only have to compute the distant field.
- Often we can bypass the entire computation of the acoustic field and just replace the sound source with a point source. This point source oscillates in a manner which is obtained from some combination of the vibrations of the surfaces of the object which constitutes a reasonable approximation to the full emission theory.
- Reflections of sound at the walls adds important reality value. Commercially available digital reverb processors are available for this.
- Filters that model the scattering at the pinna (HRTF) have been measured widely and are available on the Internet. There are commercial "spatializers" available, that can model the scattering of sound at the pinna in real time. At the time of writing, low cost soundcards for PC's come equipped with hardware support for 3D sound and the free DirectSound3D API has built in spatialization support.
- The reverberation and HRTF contributions to the sound are, under certain conditions, independent of the emission computation and can therefore be dealt with separately and independently.

4.1 Overview

Based on material and shape properties, we do a pre-computation of the relevant characteristic frequencies of each object in Section 4.2. In Section 4.3 we then divide the boundary of the object into small regions and determine the amplitudes of the excitation modes if an impulsive force is applied to a point in this region. This is similar to the tesselation of a surface for graphics rendering. The whole procedure is analogous to assigning a color to a surface and rendering it with some shading model.

In Section 4.4, we normalize the energies of the vibrations associated with the different impact points to some constant value, and scale them proportional to the impact energy when rendered. In Section 4.5 we discuss the material properties and their effect on sound. The decay rate of each mode is assumed to be determined by the internal friction parameter, which is an approximate material property [91, 43]. In effect, the decay rate of a component is assumed to be proportional to the frequency, with the constant determined by the internal friction parameter.

After the preprocessing, a sound parameter map is attached to an object, which allows us to render sounds resulting from forces on the body. We discuss the structure of this map and a possible approach to reduce its storage requirements in Section 4.6. In Section 4.7 we will construct an algorithm to synthesize the sound under any type of interaction in real-time.

4.2 Vibration Modes from Shape

We now introduce the framework for modeling vibrating objects. We will illustrate it with a rectangular membrane, but the framework is quite general; we have used

it to generate sounds of strings, bars, plates and other objects. The framework is based on the well developed models in the literature on vibration or acoustics, for example [54] – for the calculus involved we refer to [75].

The vibration of the object is described by a function $\mu(\boldsymbol{x}, t)$, which represents the deviation from equilibrium of the surface, defined on some region $\mathcal{S}$, which defines the shape of the object. We assume that μ obeys a wave equation of the form

$$(A - \frac{1}{c^2}\frac{\partial^2}{\partial t^2})\mu(\boldsymbol{x}, t) = F(\boldsymbol{x}, t) \tag{4.1}$$

with c a constant (related to the speed of sound in the material), and A is a self-adjoint differential operator, under the boundary conditions on $\partial\mathcal{S}$.

Example: For the rectangular membrane we consider a rectangle $[0 - L_x, 0 - L_y]$ spanned by a membrane under uniform tension. For this case the operator A is given by

$$A = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

The boundary conditions are that $\mu(x, y, t)$ is fixed on the boundary of the membrane, i.e., the membrane is attached to the rectangular frame.

We will take the following initial value conditions.

$$\mu(\boldsymbol{x}, 0) = y_0(\boldsymbol{x}),$$

i.e., the surface is initially in configuration $y_0(\boldsymbol{x})$, and

$$\frac{\partial\mu(\boldsymbol{x}, 0)}{\partial t} = v_0(\boldsymbol{x}),$$

where $v_0(\boldsymbol{x})$ is the initial velocity of the surface.

The solution to Equation 4.1, in the absence of external forces, is written as

$$\mu(\boldsymbol{x},t)=\sum_{n=1}^{\infty}(a_n\sin(\omega_n ct)+b_n\cos(\omega_n ct))\Psi_n(\boldsymbol{x}), \tag{4.2}$$

where a_n and b_n are arbitrary real numbers, to be determined by the initial value conditions. The ω_n are related to the eigenvalues of the operator A under the appropriate boundary conditions (which we specify below), and the functions $\Psi_n(\boldsymbol{x})$ are the corresponding eigenfunctions. That is, we have

$$(A+\omega_n^2)\Psi_n(\boldsymbol{x})=0. \tag{4.3}$$

The spectrum of a self-adjoint operator A is discrete, and the eigenfunctions are orthogonal. Their norm is written as

$$\alpha_n=\int_{\mathcal{S}}\Psi_n^2(\boldsymbol{x})d^kx.$$

Example: For the rectangular membrane, the eigenfunctions and eigenvalues are most naturally labeled by two positive integers n_x and n_y and are given by

$$\Psi_{n_xn_y}(x,y)=\sin(\pi n_x x/L_x)\sin(\pi n_y y/L_y),$$

and

$$\omega_{n_xn_y}=\pi\sqrt{(\frac{n_x}{L_x})^2+(\frac{n_y}{L_y})^2},$$

In Figure 4.1, we show the first 9 eigenfunctions on a square membrane.

As Equation 4.3 is linear, we can normalize the eigenfunctions $\Psi_n(\boldsymbol{x})$ such that α_n is independent of n, which often simplifies some of the algebra. Using the

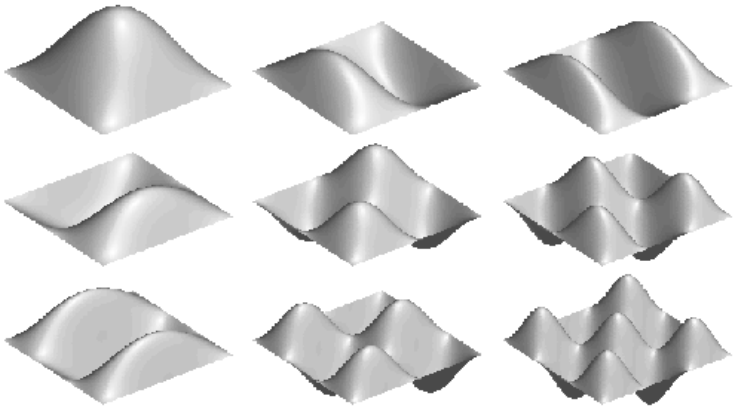

Figure 4.1: First nine eigenfunctions of a square membrane.

orthogonality of the eigenfunctions we can find the coefficients in the expansion given in Equation 4.2 as

$$a_n = \int_{\mathcal{S}} \frac{v_0(\boldsymbol{x})\Psi_n(\boldsymbol{x})}{c\alpha_n\omega_n} d^k x, \tag{4.4}$$

and

$$b_n = \int_{\mathcal{S}} \frac{y_0(\boldsymbol{x})\Psi_n(\boldsymbol{x})}{\alpha_n} d^k x. \tag{4.5}$$

The time averaged energy of the vibration is given by

$$E = \text{constant} \times < \int_{\mathcal{S}} \left(\frac{\partial \mu(\boldsymbol{x}, t)}{\partial t} \right)^2 \rho(\boldsymbol{x}) d^k x >, \tag{4.6}$$

where $\rho(\boldsymbol{x})$ is the mass density of the vibrating object. The $<>$ indicates an average over time. If the mass is distributed uniformly, we have

$$E = \text{constant} \times \sum_{n=1}^{\infty} \alpha_n \omega_n^2 (a_n^2 + b_n^2). \tag{4.7}$$

4.3 Mode Amplitudes from Impact Location

Next we compute the vibrations resulting from an impact at some point $\boldsymbol{p}$, when the body is initially at rest.

The initial value conditions are taken to be

$$y_0(\boldsymbol{x}) = 0, \tag{4.8}$$

and

$$v_0(\boldsymbol{x}) = \delta(\boldsymbol{x} - \boldsymbol{p}), \tag{4.9}$$

with $\delta(\boldsymbol{x})$ the k-dimensional Dirac delta function.

We note that Equation 4.9 is not strictly correct as an initial value condition. The reason is that the expression for the energy, given in Equation 4.6, involves the square of the time derivative of $\mu(\boldsymbol{x}, t)$. But the integral of the square of the Dirac

delta function is infinite. One symptom of this is that the infinite sum appearing in Equation 4.2 does not converge. A mathematically more correct method would replace the delta function in the initial value conditions by some strongly peaking external force function, representing the impact on a small region of the object over a finite region and over a small but finite extension in time. However, this would complicate things quite a bit, and we would gain little in terms of more realistic sounds. Therefore we shall just assume an appropriate frequency cutoff in the infinite sum appearing in Equations 4.7 and 4.2. Typically, we will only use the frequencies in the audible range. For more details and a more rigorous treatment of this problem for the special cases of the ideal string and the circular membrane, see [54].

Using Equations 4.8 and 4.9, and substituting them in Equations 4.4 and 4.5 we obtain the amplitudes of the vibration modes as a function of the impact location as

$$a_n = \frac{\Psi_n(\boldsymbol{p})}{c\alpha_n\omega_n}, \tag{4.10}$$

and

$$b_n = 0.$$

The energy of the vibration is determined by the impact strength. It will be used to scale the amplitudes of Equation 4.10. The energy is given by

$$E = \text{constant} \times \sum_{n=1}^{n_f} \frac{\Psi_n^2(\boldsymbol{p})}{\alpha_n},$$

where n_f is determined by the frequency cutoff mentioned above.

Example: In Figures 4.2 to 4.4 we show the amplitudes a_n, graphed against the frequency of the modes (i.e., ω_n) for a square membrane

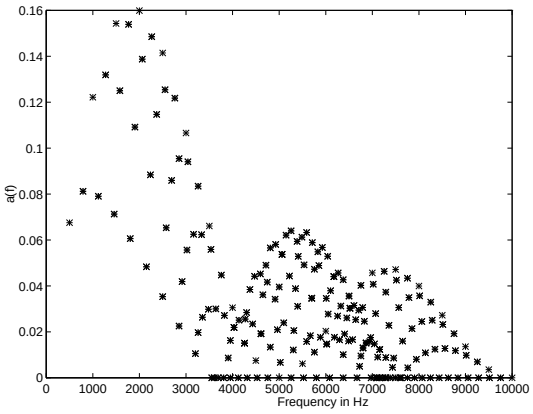

Figure 4.2: Excited frequencies of a square membrane struck near the corner.

struck at the points $(0.1, 0.1)$, $(0.1, 0.4)$, and $(0.5, 0.4)$, using a cartesian coordinate system with the origin $(0, 0)$ in the corner and the opposite corner at $(1, 1)$. We have taken the lowest frequency to be 500 Hz and taken the first 400 modes into account. We can see clearly that the higher frequencies become relatively more excited for strike points near the boundary of the membrane. In other words, the membrane sounds dull when struck near the center, and bright (or sharp) when struck near the rim.

The method outlined above is very general, and allows the computation of the vibrations under impact of any object governed by a differential equation of the form given in Equation 4.1.

The frequency spectrum ω_n and the eigenfunctions $\Psi_n(\boldsymbol{x})$ can be computed analytically in a number of cases. In general one has to resort to numerical methods. For membranes, the problem reduces to the solution of the Laplace equation on a given domain, which is a well studied problem. We mention the method of particular solutions [28], which we have adapted for the example of the L-shaped membrane,

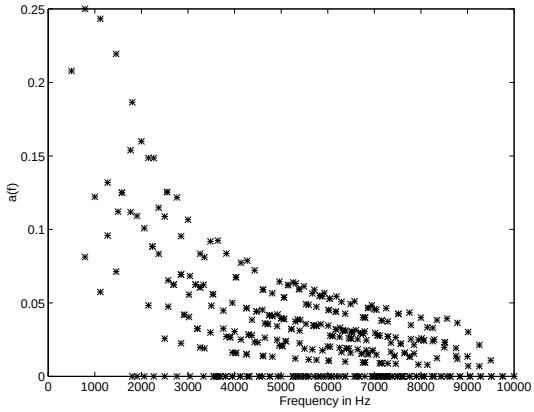

Figure 4.3: Excited frequencies of a square membrane struck near the middle of a side.

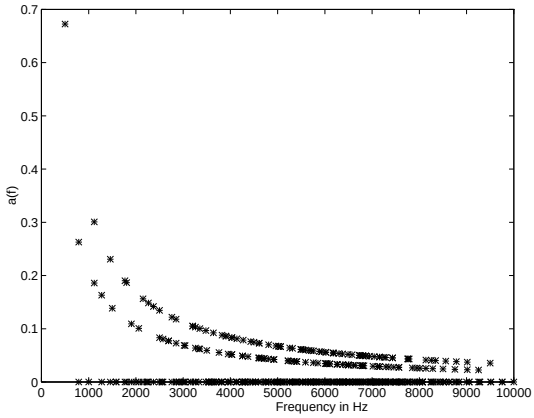

Figure 4.4: Excited frequencies of a square membrane struck near the center.

described below in Section 5.1. For plates, the operator A is fourth order, and a more general finite element method can be used. See for example [38].

4.4 Sound Sources from Vibrating Shapes

After obtaining the frequency spectrum and the eigenfunctions, using methods described in Section 4.3, we have to model the relation between the vibration of the object and the sound emitted. In general the sound-field around a vibration body is very complicated and non-uniform. However, it is clear that the sound emitted can be described as a sum of monochromatic components with frequencies $\omega_n c$, and amplitudes a_n^S, which will depend on the location of the observer with respect to the object, as well as on the environment.

As a first approximation, we will identify the coefficients a_n^S with the vibration amplitudes a_n, scaled with the inverse of the distance to the observer, as the amplitude decays inversely proportional to the distance.

This is not strictly correct, but we argue that it is reasonable as follows. Consider a vibrating plate. At some point above the plate, waves emerging from all locations on the plate arrive at this point. Some will be in phase, and some will be out of phase. This interference will depend very sensitively on the location of the observation point. However, in most real situations, the sound will not only arrive directly from the source, but also from reflections from the walls and other objects in the room. The total effect of this is to average out the phase differences, making the sound-field less sensitive to the locations of the listener.

As a heuristic, we assume that the intensity (i.e., the energy) of the sound

emitted in frequency ω_n , I_n, is given by

$$I_n = E_n \int_{\mathcal{S}} \Psi_n^2(\boldsymbol{x}) = \text{constant} \times \Psi_n^2(\boldsymbol{p}).$$

This seems reasonable, as it integrates the intensity of the vibration, but not the phase. This means, we can identify the a_n^S, which are the amplitudes of the heard sound, with the a_n given in Equation 4.10, omitting the factor α_n. Note that since we assumed that the eigenfunctions are normalized so that the α_n are independent of n, this does not matter.

Finally, we obtain the amplitudes a_n^S as

$$a_n^S = \frac{E_{\text{impact}} \Psi_n(\boldsymbol{p})}{\omega_n Q(\boldsymbol{p}) d}, \tag{4.11}$$

with d the distance from the sound source, E_{impact} the energy of the impact, and

$$Q(\boldsymbol{p}) = \sqrt{\sum_{i=1}^{n_f} \Psi_n^2(\boldsymbol{p})},$$

with n_f a suitable frequency cutoff. Of course, the a_n^S are only defined up to a multiplicative constant (corresponding to the volume setting of the audio hardware).

For a more detailed treatment of the radiation of vibrating plates, we refer to books on vibration analysis [65, 66, 27].

4.5 Sounds and Material Properties

When the object is struck, each frequency mode is excited with an initial amplitude a_i, which depends on where the object is struck. The relative magnitudes of the amplitudes a_i determines the "timbre" of the sound. Each mode is assumed to decay exponentially, with decay time

$$\tau_i = \frac{1}{\pi f_i \tan \phi}, \tag{4.12}$$

where ϕ is the internal friction parameter. The internal friction parameter is roughly invariant over object shape, and depends on the material only. In [91] a method was proposed to identify the material type from the sound emitted by a struck object, by extracting the internal friction parameter of the material via Equation 4.12. Such a model is also used in [80] to simulate object sounds. Some experiments were reported in [43], where it was concluded that a rough characterization of material was indeed possible. However, the internal friction parameter is only approximately invariant over object shape. See also [87].

To emulate external damping of the object, we add an overall decay factor of e^{-t/τ_0}. This also allows us to adjust the length of the emitted sound, while maintaining its "material character", which is determined by ϕ.

So we assume the sound-wave $p_S(t)$ to be given for $t \geq 0$ (for $t < 0$ it is zero) by

$$p_S(t) = e^{-t/\tau_0} \sum_{i=1}^{n_f} a_i^S e^{-t f_i \pi \tan\phi} \sin(2\pi f_i t), \tag{4.13}$$

with the amplitudes a_i^S given in Equation 4.11, and

$$f_i = \frac{\omega_n c}{2\pi},$$

with the ω_n determined by Equation 4.1.

Although this simple one-parameter characterization of material works perceptually reasonably well, there is no advantage to restricting the damping coefficients in any way. When dealing with model parameters acquired from measurements we will allow the damping coefficients to take on values independent of the frequencies. In this case we will write the impulse response as

$$p_S(t) = Im(\sum_{i=1}^{n_f} a_i e^{i\Omega_i t}), \tag{4.14}$$

with $\Omega_i = \omega_i + i d_i$, where d_i are the dampings.

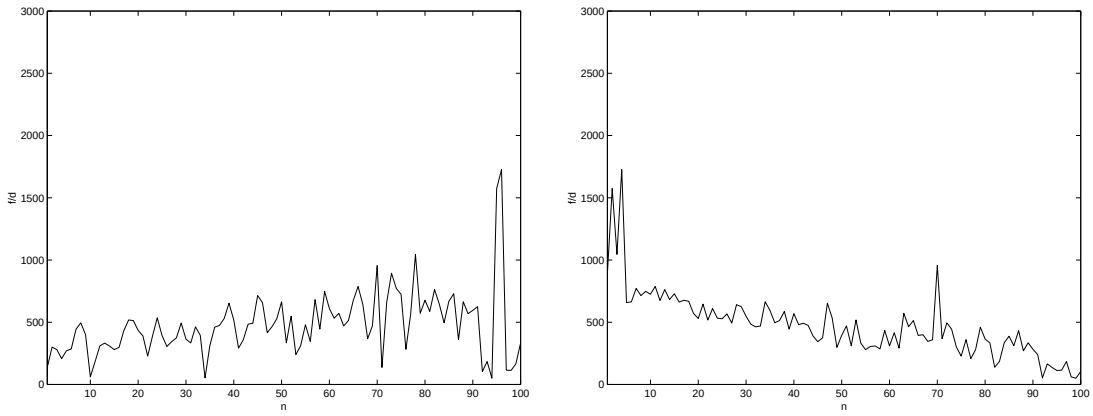

Figure 4.5: Ratio of frequency/damping for the first 100 modes sorted by amplitude (left) and frequency (right) of a struck metal vase. Reconstructed with a Fourier window of 4096.

We have investigated the extent to which the ratio of frequency and damping is constant in real objects, and found that this relation is only very approximately valid. In Figure 4.5 we plot the ratio f/d for the first 100 modes of a metal vase reconstructed with the techniques developed in Chapter 5. The average ratio f/d is 477, characterizing metal, but still varies considerably, with a standard deviation of 262.

In Figure 4.6 we plot the ratio f/d for the first 100 modes of a wooden hockey stick. The average ratio f/d is about 35 which, as an order of magnitude, seems a good characterization of wood. The standard deviation is 22.

In Figure 4.7 we plot the ratio f/d for the first 100 modes of a metallic computer tower box. This is an extremely complex object with rattling parts inside, and this seems to be reflected in the great variance in the ratio for this object.

In Figures 4.8 we plot the ratio f/d for the first 100 modes of a metal sword. The average ratio f/d is 1005. The standard deviation 949 is very high.

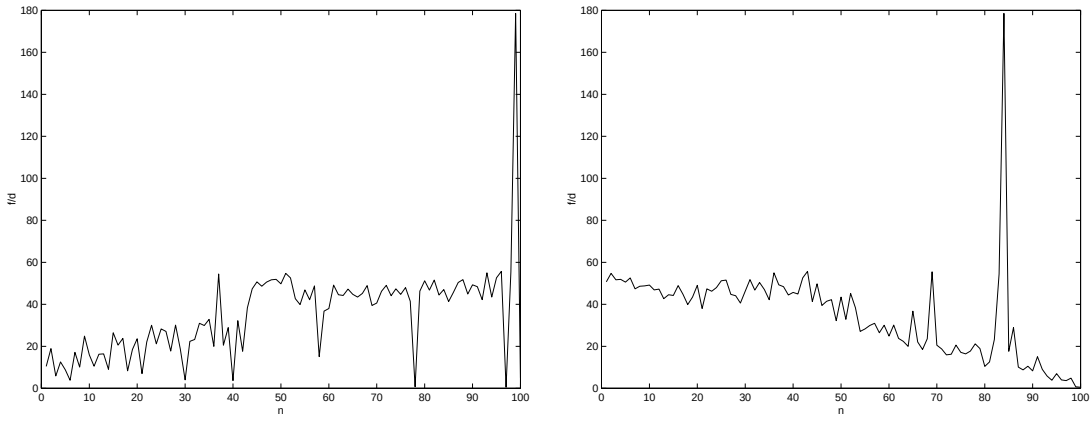

Figure 4.6: Ratio of frequency/damping for the first 100 modes sorted by amplitude (left) and frequency (right) of a struck hockey stick. Reconstructed with a Fourier window of 1024.

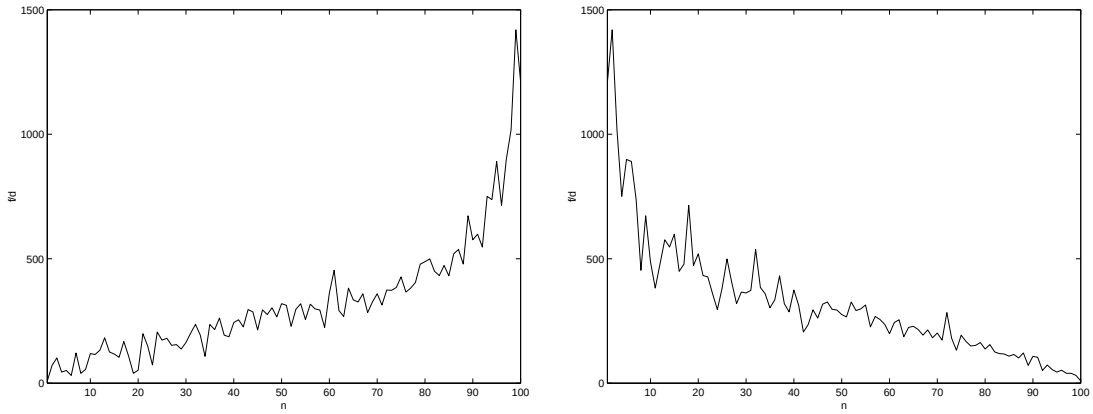

Figure 4.7: Ratio of frequency/damping for the first 100 modes sorted by amplitude (left) and frequency (right) of a struck computer tower box. Reconstructed with a Fourier window of 1024.

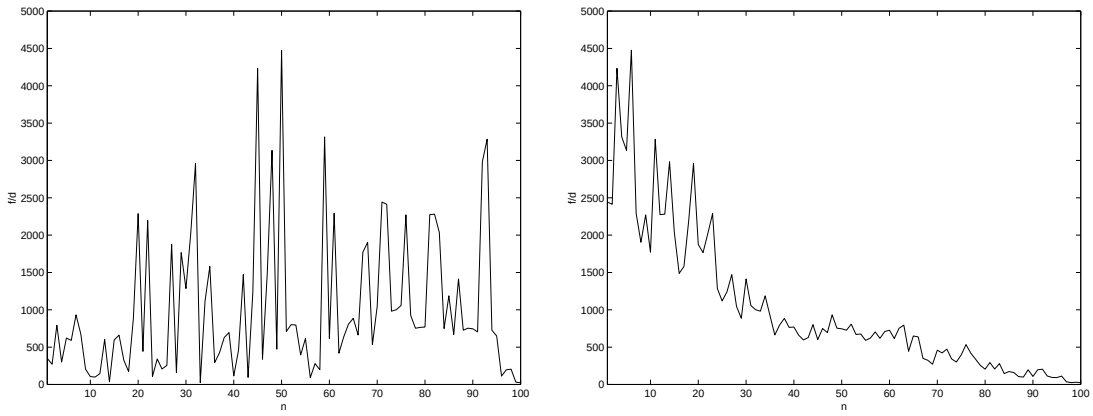

Figure 4.8: Ratio of frequency/damping for the first 100 modes sorted by amplitude (left) and frequency (right) of a struck sword (sword I). Reconstructed with a Fourier window of 2048.

4.6 The Sound Map

In the preprocessing stage we first compute the frequency and damping spectrum, and then the excitation spectrum a_i under a suitably normalized impact (i.e., with fixed energy) for a number of locations on the surface. This gives us all the information needed to compute the sound under any kind of external force during the real time simulation. This is somewhat analogous to texture mapping in computer graphics. An interpolation of the timbre spectrum a_i^S between pre-computed locations is also obvious to implement.

One may ask how many points on the surface need to be computed. In general, the timbre of the sound changes non-uniformly over the surface. For example, a string sounds "dull" when plucked near the center, and becomes dramatically brighter when excited near the endpoints. In this case, one would need a denser set of points near the ends.

Given two sounds S_1 and S_2, a measure $d(S_1, S_2)$ is needed, that tells us

how different sound S_1 "sounds" from sound S_2, such that if $d(S_1, S_2) < d_0$, with d_0 a threshold (depending on the individual), S_1 and S_2 can not be distinguished. Perception of timbre is a complex subject, see for example [15, 51] for a discussion, so we can not expect to be able to formulate such a sound-distance measure easily and accurately. A complication is that a very important distinguishing factor between pitched sounds is the perceived pitch, which is the same in our case, so our timbre measure depends on subtle and poorly understood aspects of audio psychology.

As an initial proposal we take the sonic distance $d(S_1, S_2)$ between two sounds to be

$$d^2(S_1, S_2) = \sum_{i=1}^{n_f} S(f_i)(H(\log(E_i^1/E_0)) - H(\log(E_i^2/E_0)))^2,$$

where E_i^r denotes the energy contribution of the i-th mode of sound r $(= 1, 2)$, i.e., $E_i^r = (a^{Sr}{}_i)^2 f_i^2$. We take the logarithm of the energy, as the human ear is sensitive to the logarithm of intensity (measured in decibels). The function $H(x)$ is zero for $x < 0$, and x otherwise. The constant E_0 represents the lowest sound-level that can be heard, so the term $H(\log(E_i^r/E_0)$ vanishes if $E_i^r < E_0$. The function $S(f)$ models the sensitivity of the ear to frequency. Without loss of generality we take $0 \leq S(f) \leq 1$. The function $S(f)$ has to be determined psychoacoustic experiments.

We have ignored the "masking" effect, which changes the sensitivity curve of the ear in the presence of other stimuli. One could also argue that the threshold energy E_0 depends on frequency. We leave a refinement of the measure d as a topic for future research.

In addition to encoding the location dependency of the interaction on the surface, one can also extend this by taking into account the direction of the interaction at a given point. It is also possible to have different coupling coefficients at different locations and orientation around the object, thereby encoding the spatial

field of sound around the object.

4.7 Real-time Synthesis

The vibration modeling developed thus far in this Chapter generalizes to more general interactions besides impulsive forces. Because the model is linear, any interaction force can be represented as the sum of an infinite number of "impulses", and the resulting equations lead to an efficient real time synthesis algorithm for the synthesis of the sound of an object under any kind of external force.

We shall now derive the algorithm, show how it can be implemented most efficiently, and we then show that it can be viewed as a discretization of a continuous time spring-damper system.

According to our linear model, the sound produced by an impulsive force of magnitude F at time s can be described by the imaginary part of the complex wave form

$$y(t) = \sum_n a_n e^{i\Omega_n(t-s)} H(t-s) F, \tag{4.15}$$

where the sum is over the complex eigenfrequencies Ω_n (the imaginary part determines the damping of a mode). $H(t) = 0$ for $t < 0$ and $H(t) = 1$ for $t \geq 0$. A continuous stimulus force $F(t)$ can be represented formally as an infinite sum of infinitesimal impulses

$$F(t) = \int_0^\infty \delta(t-s) F(s) ds,$$

where $\delta(t)$ is the Dirac delta distribution, assuming the force is zero for negative times. Using the principle of linearity, the output of the model driven by this force can be written as a sum of infinitesimal contributions from each of these impulses:

$$y(t) = \int_0^\infty ds \sum_n a_n e^{i\Omega_n(t-s)} H(t-s) F(s).$$

Discretizing this equation in time, with sampling rate S_R, gives

$$y(k) = \sum_{l=0}^{k} \sum_{n} a_n e^{i\frac{\Omega_n}{S_R}(k-l)} F(l),$$

with $y(k) = y(t_k)$ and $t_k = k/S_R$. This can be rewritten as a recursion by defining the functions $y_n(k)$, one for each partial. The complex signal is written as a sum of modal contributions

$$y(k) = \sum_{n} y_n(k).$$

For the partials $y_n(k)$ we have

$$y_n(0) = a_n F(0)$$

and the recursion relation

$$y_n(k) = e^{i\frac{\Omega_n}{S_R}} y_n(k-1) + a_n F(k) \tag{4.16}$$

determines the audio signal $Im(y)$. As $|e^{i\frac{\Omega_n}{S_R}}| < 1$, the recursion relation is always stable. Equation 4.16 requires 5 multiplications per sample point, which can be reduced to 3 as we will now derive.

To simplify the notation, let us drop the subscripts n, labeling the mode, and define a two-component vector

$$\boldsymbol{y} = \begin{pmatrix} Re(y) \\ \\ Im(y) \end{pmatrix}.$$

The recursion 4.16 can now be written as

$$\boldsymbol{y}(k) = \boldsymbol{A}\boldsymbol{y}(k-1) + \begin{pmatrix} aF(k) \\ \\ 0 \end{pmatrix}, \tag{4.17}$$

where

$$\boldsymbol{A} = \begin{pmatrix} c_r & -c_i \\ \\ c_i & c_r \end{pmatrix},$$

with

$$c_r = e^{-d/S_R}\cos(\omega/S_R),$$

$$c_i = e^{-d/S_R}\sin(\omega/S_R),$$

$$d = Im(\Omega),$$

and

$$\omega = Re(\Omega).$$

Let us change variables to $\boldsymbol{z}$ by writing

$$\boldsymbol{y} = \boldsymbol{Q}\boldsymbol{z}.$$

In terms of $\boldsymbol{z}$, the recursion 4.17 reads

$$\boldsymbol{z}(k) = \boldsymbol{B}\boldsymbol{z}(k-1) + \boldsymbol{Q}^{-1}\begin{pmatrix} aF(k) \\ 0 \end{pmatrix},$$

where

$$\boldsymbol{B} = \boldsymbol{Q}^{-1}\boldsymbol{A}\boldsymbol{Q}.$$

Since $\boldsymbol{B}$ has the same eigenvalues as $\boldsymbol{A}$, this does not affect the stability of the recursion. There are several (equivalent) choices for $\boldsymbol{Q}$ that reduce the number of multiplications to 3, and we choose

$$\boldsymbol{Q} = \begin{pmatrix} 1 & \sqrt{1-c_i^2} \\ 0 & c_i \end{pmatrix}.$$

Defining

$$c_+ = c_r + \sqrt{1-c_i^2}$$

and

$$c_- = c_r - \sqrt{1-c_i^2},$$

we arrive at the following equation for $\boldsymbol{B}$,

$$\boldsymbol{B} = \begin{pmatrix} c_- & -1 \\ 1 & c_+ \end{pmatrix}.$$

In terms of the real variables u and v,

$$\boldsymbol{z} = \begin{pmatrix} u \\ v \end{pmatrix},$$

the recursion becomes

$$\begin{aligned} u(t) &= c_- u(t-1) - v(t-1) + ac_i F(t) \\ v(t) &= u(t-1) + c_+ v(t-1). \end{aligned}$$

We have now only 3 multiplications per sample point (as ac_i can be pre-computed), but because the physical quantity of interest, $Im(y)$, is related to v by

$$Im(y(t)) = c_i v(t),$$

it appears that we need another multiply per sample point to obtain this. We can avoid this by multiplying a with c_i in the preprocessing phase. Assuming that we have initially silence, i.e., $u(t) = v(t) = 0$ for $t < 0$, the linearity of the system guarantees that multiplying the input signal with a factor c_i will multiply the output signal with this factor also. We therefore arrive at the following synthesis recursion for the individual modal contribution $v(t)$:

$$\begin{aligned} u(k) &= c_- u(k-1) - v(k-1) + \hat{a} F(k) \\ v(k) &= u(k-1) + c_+ v(k-1), \end{aligned} \tag{4.18}$$

with

$$\hat{a} = ac_i^2.$$

The intuitive picture of a set of spring-damper systems driven by an external force is verified by considering the behaviour of u and v for large S_R. In that case, a continuous time differential equation should emerge, which describes a spring-damper system under an external force. By substitution of

$$u(k-1) = v(k) - c_+ v(k-1)$$

in Equation 4.18 we obtain a second order recursion for $v(k)$:

$$v(k+1) - (c_+ + c_-)v(k) + (1 + c_- c_+)v(k-1) = \hat{a}F(k).$$

We now make the connection to the continuous time system by expanding v to second order in $1/S_R$. If we denote the continuous time limit of v by V, with $V(t) = v(tS_R)$ wherever v is defined, we obtain

$$v(k+1) = V(t) + \dot{V}(t)/S_R + \ddot{V}(t)/2S_R^2,$$

(dropping the arguments t)

$$v(k-1) = V - \dot{V}/S_R + \ddot{V}/2S_R^2,$$

$$c_+ = 2 - d/S_R - \omega^2/S_R^2 - d^2/2S_R^2,$$

and

$$c_- = -d/S_R - d^2/2S_R^2.$$

With some algebra we obtain

$$\ddot{V} + 2d\dot{V} + (\omega^2 + d^2)V = \hat{a}S_R^2 F,$$

which is precisely the equation for a spring-damper system driven by an external force $\hat{a}S_R^2 F$. In some applications we have found that the quantity $u + v$ sometimes

produces a better sound, especially for car engine sounds. The quantity $w = u + v$ satisfies in the continuum limit

$$\ddot{W} + 2d\dot{W} + (\omega^2 + d^2)W = \hat{a}S_R(dF - \dot{F}).$$

This means that by using w instead of v for the audio signal, we can obtain the same sound (up to a volume factor) as obtained by using v with input signal $(dF - \dot{F})$. Because of the derivative of F, this generally gives a brighter driving signal.

To synthesize the sound in real-time, we repeatedly compute an audio buffer of length T. The synthesis algorithm fetches the values of the coefficient arrays c_+, c_-, and a, as well as the external force F for the time interval T. Equation 4.18 is then used to sequentially add contributions of the modes v_n until all modes have been added or until a certain deadline has been passed. If the modes are sorted in a decreasing order of importance, this allows for a graceful degradation in the quality of the synthesized sound, when the time available for audio synthesis is not constant. This algorithm is explained in more detail in Section 7.1.

Chapter 5

Creating Vibration Models

The linear vibration model presented in Chapter 4 parameterizes an object's acoustic response by a set of frequencies, dampings, and a set of amplitude functions on the surface that determine the coupling between an external force and the modes at that location. The amplitudes will be sampled on a discrete set of locations and when we are not interested in the location dependency of the sound, the coefficients a will be a set of numbers. The number of parameters needed depends on the nature of the model. Dense sounds like that of drums, or complex musical instruments will require a large amount of modes, whereas sounds with a less dense spectra, such as bars and plates, require much less.

The model parameters are stored in an ASCII file of a format that we denote by *sy*, which is used by all our implementations. The file format is designed to be human readable, and also contains some extra information which allows the user to change overall characteristics of the model (such as the frequency scale of the object) easily. This file format is explained in Appendix A.

The model parameters can be acquired from mathematical modeling of the

material and shape, or by parameter identification methods, or by "manual" means.

5.1 Computing Model Parameters Analytically

For simple shapes and simple material properties it is possible to write down an explicit partial differential equation (PDE) for the vibration of the object, and for *very* simple shapes it is even possible to solve the resulting PDE analytically for some boundary conditions. For more complicated shapes and materials, a finite element model [38, 58] could be used to compute the vibration modes and frequencies. However, in many cases not enough information about the object's geometry and material characteristics is available to do a sensible computation from first principles. We therefore have not pursued the finite element modeling approach in this work.

We have explicitly computed the model parameters for a number of simple geometries, which allow an analytic solution of the vibration equations. These shapes already provide a fairly rich set of objects for use in simulations and are very useful if the goal is to provide generic interactive audio rather than the sound of specific physical objects (which are usually too complicated to consider modeling from first principles). We have obtained analytic model parameters for the following shapes, which can be generated using the software described in Chapter 7.

1. **The taut string.** This is the simplest example of a vibrating system. The eigenfunctions are simple sine functions. The sound becomes brighter for impacts near the ends of the string. The frequency spectrum is harmonic, i.e., all frequencies are integer multiples of the lowest (fundamental) frequency. The amplitudes a_n are inversely proportional to n, for large n, in contrast to a

plucked string, considered in [80], where they decay as $1/n^2$. This is one factor accounting for the difference between a piano and a guitar sound, for example. A derivation of the eigenfunctions and eigenfrequencies can be found in [54], Chapter III. The nonlinear behaviour of the string (making it less suitable for this type of modeling) was investigated in [16].

2. **The rigid bar.** For the rigid bar the operator A appearing in Equation 4.1 is given by

$$A = -\frac{\partial^4}{\partial x^4}.$$

As A is a fourth order operator, we need to specify 4 boundary conditions. We have computed a clamped-clamped bar, i.e., the bar is rigidly attached at both ends, and a clamped-free bar. The boundary conditions are

$$\Psi_n(0) = \Psi_n(1) = \left(\frac{d\Psi_n}{dx}\right)_{x=0} = \left(\frac{d\Psi_n}{dx}\right)_{x=1} = 0 \tag{5.1}$$

for the clamped-clamped case and

$$\Psi_n(0) = \left(\frac{d\Psi_n}{dx}\right)_{x=0} = \left(\frac{d^2\Psi_n}{dx^2}\right)_{x=1} = \left(\frac{d^3\Psi_n}{dx^3}\right)_{x=1} = 0. \tag{5.2}$$

for the clamped-free bar, which is assumed to be clamped at $x = 0$ and free at $x = 1$. The frequency spectrum is less dense than for the string, and not harmonic, due to the different nature of the restoring forces on a bar. The sparse spectrum makes this shape very suitable for efficient modeling with the synthesis methods considered here. For details, see [54], Chapter IV.

3. **The rectangular membrane.** This is the simplest two-dimensional geometry. This shape has been used as an illustrative example in Chapter 4. The sound spectrum is extremely dense, giving a rich complex sound. Because of this density, it does not lend itself well to the synthesis algorithm described in

this thesis, as many modes are needed to obtain a good sound. For details on the rectangular membrane, see [54], Chapter V, Section 18.

4. **The circular membrane.** This corresponds to the vibrations of a drum, ignoring the effects of the surrounding air on the drum membrane. The eigenfunctions are Bessel functions, and the eigenfrequencies can be computed as the zeros of Bessel functions. The sound is also very dense, and is therefore not well suited for our type of synthesis for real-time applications. The rectangular and the circular membrane can be thought of as idealized models for drums, which fall outside the scope of this work. For details on the circular membrane, see [54], Chapter V, Section 19.

5. **The circular plate.** This is one of the few cases where the two-dimensional plate equations can be separated, which allows an analytic solution. The eigenfunctions are a combination of Bessel functions and modified Bessel functions. We have considered a plate clamped rigidly at the boundary. The spectrum is much less dense than for the circular membrane. This is due, as for the bar, to the larger restoring forces in a plate, compared to a membrane. Therefore this model leads itself very well to our synthesis method. For details on the circular plate, see [54], Chapter V, Section 21.

6. **The simply supported rectangular plate.** This is another of the few cases where the two-dimensional plate equations can be separated, which allows an analytic solution. The eigenfunctions are products of sine functions. The spectrum is much less dense than for the rectangular membrane. This is due, as for the bar, to the larger restoring forces in a plate, compared to a membrane. Therefore this model leads itself very well to our synthesis method.

For details on the rectangular plate, see [55], page 389.

7. **The L shaped membrane.** A membrane supported by a domain consisting of three unit squares in the shape of an L does not allow an analytic solution of the wave equation. This problem has received some attention in the literature, as the resulting boundary value problem requires some refined numerical methods. We have computed the eigenfunctions and the spectrum with an adaptation of the method of partial solutions, see [28], which is available for the L shaped membrane from within MATLAB. As an aside, we note that the first eigenfunction features prominently on the cover of the MATLAB reference guide [8].

5.2 Fitting Model Parameters to Empirical Data

An experimental approach to obtaining sound model parameters is to record sounds of real objects and fit the model parameters to the recorded actual sounds. Because of the linearity of the model, all we need is the response of the object to an impulsive force. By striking an object and recording the sound, we can fit the recorded waveform with a function of the form given in Equation 4.15.

We can think of this as designing a digital filter of a specific type with a given impulse response (the recording). Because recordings have a lot of noise in them (we want a method that is applicable for "home users"), and objects can't be excited with a true impulse, we need a robust parameter estimation method. The extraction of sinusoidal signals from time-series data has attracted a lot of attention in the statistics and signal processing literature. For a general introduction see [61]. For more recent work, see [44, 56, 14, 70, 69].

Experimentation in MATLAB with the Prony method [23], and other filter design methods such as the maximum entropy method [71] used in Linear Predictive Coding gave very unsatisfactory results. We tried constructing IIR filters from recorded sounds of several objects and then compared the reconstructed impulse response to the originals. In most cases the reconstructed sounds were very distorted, and often the damping coefficients were completely wrong (too large). We believe this is because the actual sounds are only partially described by the linear model, and non-linearities and external noise are known to cause problems with these methods. Therefore we have not pursued this approach any further.

Similar difficulties were reported in [41] when trying to construct an IIR filter for the impulse response of the body of a guitar. It was found that autoregressive (AR) modeling using the autocorrelation method of linear prediction (LP) [48], as well as the pole-zero (ARMA) model using Prony's method [57], require extremely high order filters in order to accurately predict the decay rates reasonably well. The general consensus seems to be that for complex noisy data and rough models (as we are considering here), methods based on spectral analysis are more appropriate.

Better results were obtained with a more robust approach using windowed Fourier transforms. We construct a spectrogram from the recorded audio signal, and use this to determine the dominant frequencies, their decay rates, and their amplitudes.

We will now describe the algorithm. The input consists of a recorded impulse response of a real object. The recordings were made at a sampling rate of 44,100 Hz and encoded as a 16 bit `wav` file. A brief fragment of silence before the strike is (optionally) used to determine the noise level. The analysis consists of the following steps:

To be computed: Arrays f, d, and a corresponding to the frequencies, dampings, and coupling amplitudes of a modal model.

Perform the windowed DFT:

1. Set the following constants:

   f_S = sampling rate (44100)

   N = size of DFT window (1024 – 8192)

   M = desired number of modes (100)

   X = signal to noise ratio (10)

   Q = window overlap factor (4)

2. Load the sample $y(i)$ from disk. Store as a floating point array of length L. Total duration of the sample is L/f_S.

3. Compute the overlapping windowed discrete Fourier transforms $W_k(i)$, $i = 0, \ldots, N/2-1$. W_k is obtained by windowing the vector $y(i)$ over the interval $[kN/Q, kN/Q + N - 1]$ with a Hanning window [25, 72, 61], and taking the DFT. The index k which labels the DFT's lies in the range $k = 0, \ldots, \lfloor (L-N)Q/N - 1 \rfloor$.

Identify the part of the signal used for analysis

4. Compute the intensities

$$A_k = \sum_{i=0}^{N/2-1} |W_k(i)|.$$

5. Find the k_{max} for which A_k is maximal, and define $k_0 = k_{max} + 1$. This is the start of the impulse response.

6. Analyze the amplitudes A_k on the "silence" fragment $k = 0, \ldots, k_{max} - Q$ and compute the average $\bar{A}$, and the standard deviation $\sigma(A)$. Find the smallest k, $k > k_0$, such that $A_k < \bar{A} + X\sigma(A)$, and call this value k_{end}. This is the end of the "signal". The constant X, together with the background noise level determines this.

7. Define

$$V_k(i) = log|W_{k_0+k}(i)|,$$

where $k = 0, \ldots, K-1$, with $K = k_{end} - k_0$. This is the part of the signal we will use to extract the model parameters.

Estimate the frequency modes

8. Define an array of $N/2 - 1$ "bins", $B(i)$, $i = 0, \ldots, N/2 - 1$, each corresponding to a frequency of $f_S(i+1)/N$. Initialize them to zero.

9. For $k = 0, \ldots, K-1$, find the set of indices i, call them $\{i^{max}\}$, for which $V_k(i)$ is a local maximum, i.e., $V_k(i) > V_k(i+1)$ and $V_k(i) > V_k(i-1)$. (There is a different set $\{i^{max}\}$ for each value of k.) Select the M indices $i_1^{max}, \ldots, i_M^{max}$ with maximal values of $V_k(i)$, and add 1 to $B(i_j^{max})$ for each of these indices. We say these bins have been voted for as candidates for a resonance mode. There are K voting rounds.

10. Select the M bins $B(i)$ with most votes, call them $i = l_0, \ldots, l_{M-1}$ and obtain the estimated frequencies

$$f_i = f_S(l_i + 1)/N,$$

for $i = 0, \ldots, M-1$.

Estimate the damping coefficients of the modes

11. For each l_i, fit $V_k(l_i)$ as a function of k, $k = 0, \ldots, K-1$, with a linear function $-\alpha_i k + \beta_i$, using a least squares algorithm. The damping coefficients d_i are identified as

$$d_i = Q f_S \alpha_i / N,$$

for $i = 0, \ldots, M-1$.

Estimate the coupling amplitudes

12. The amplitudes a_i are obtained as

$$a_i = \frac{e^{\beta_i} \rho_i}{1 - e^{-\rho_i}},$$

with $\rho_i = d_i N / f_S$.

13. Normalize the amplitudes a_i so that $\max(a_i) = 1$, and sort f_i, d_i, and a_i by the value of a_i.

The sample $y(i)$, $i = 0, \ldots, L-1$ is divided in a number of (overlapping) windows of size N. The window size N is typically $N = 1024, 2048, 4096, 8192$, or about $25 - 200ms$. The windows overlap, and the overlap of about 75%, corresponding to $Q = 4$, was found to work well by trial and error. For each window, a discrete Fourier transform $W(j)$, $j = 0, \ldots, N-1$ is computed, using a Hanning window [25, 72, 61], in step 3. The value of N determines the spectral resolution as $\Delta f = f_s/N$, where f_s is the sampling rate. For $N = 2048$ this gives a resolution of about $20Hz$. For a typical pitch of about $400Hz$ this corresponds to a perceived pitch error of $(12/\log(2)) \log((400+20)/400) = .13$ semitones, or 13 cents. This is clearly audible, but in the types of sound we are concerned with, absolute pitch is not an important factor. If so desired, an overall fine-tuning of the pitch can be made

later, either manually or by using a specific pitch detection algorithm [68, 62, 60]. See Figure 5.1 for an example of the recording of the sound of a metal vase struck at the top, and its reconstruction using the parameter fitting described here. Displayed is the sonogram based on $N = 4096$ of the actual recording on the left, and its reconstruction using 40 partials on the right. The best window size for a given sound was determined experimentally, by reconstructing the impulse response and comparing it "by ear" with the original. There seems to be no obvious way to automate this process. Often reconstructions with different window sizes would differ audibly, but it was not possible to say which one was "closer" to the original sound. Until we have a better understanding of timbre perception, the best approach seems to be to provide interactive tools with humans making perceptual judgements.

For each window the norm of the Fourier transform is summed over all frequencies, giving the average intensity of the signal as $A_k = \sum_{i=0}^{N/2-1} |W_k(i)|$ in step 4. The window W_{k_0} *after* the window with maximum intensity A_k is chosen as the start of the impulse response in step 5. Note that if the signal starts somewhere *inside* an FFT window, this window may or may not register as the maximum intensity window. In order to avoid artifacts from this, we throw away the first window and start the analysis from the next one, which is guaranteed to contain only signal.

The beginning of the sample (before the impulse response) is analyzed in step 6 for the average level (the background noise) and the standard deviation. This section ends with the window indexed by $k_{max} - Q$ because the window indexed by k_{max} contains the beginning of the impulse response, and we therefore have to go back Q windows to obtain the previous non-overlapping window. This information is used to determine the end of the impulse response, which is set at the point where the signal amplitude falls below the background level plus some reasonable number

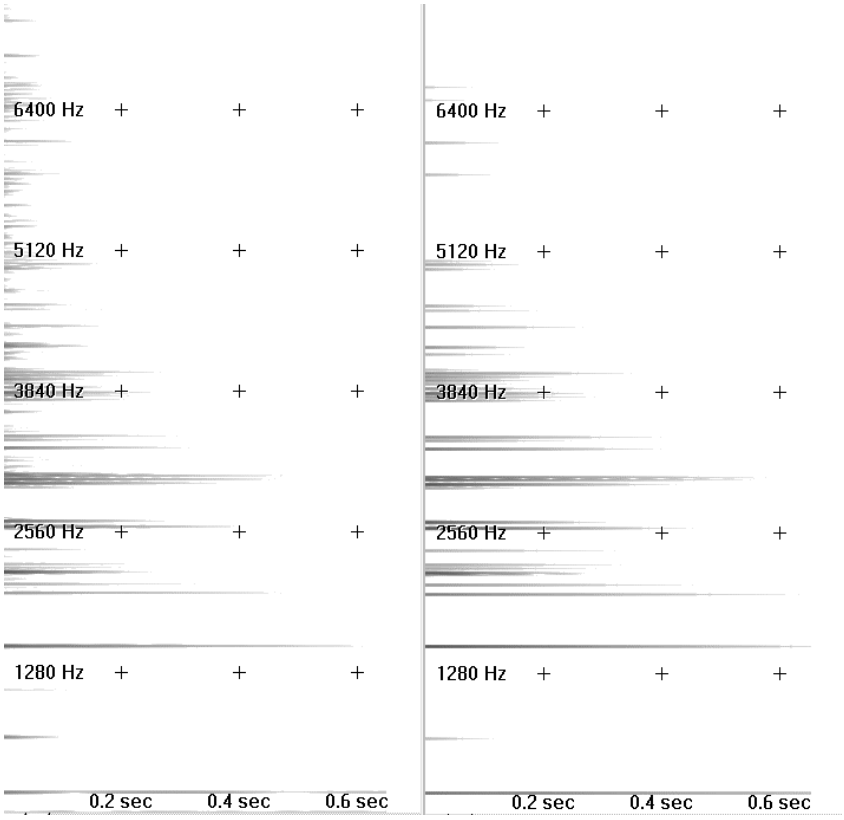

Figure 5.1: Sonogram of recorded (left) and reconstructed (right) impulse response of vase. The window size of the Fourier transform was 4096 and the sampling rate was 44100 Hz.

(empirically set to 10) times the background standard deviation. This corresponds roughly to what would be obtained by a visual inspection and cutting of the signal when it "looks like noise".

The logarithms of the absolute values of the windows containing a signal are identified in step 7 to extract frequencies, amplitudes, and dampings. For each time-slice we find the frequency bins which are local maxima in step 9, and for the M largest peaks we increase the counts of those bins by one (they are initialized to zero, of course). After doing this for all time-slices, we keep the M bins with the most votes and these are the frequencies identified in step 10. For each of those bins i we fit $V_k(i)$ as a function of k with a linear function, and we extract the damping parameters in step 10 and the amplitudes in step 11.

In Figure 5.2 we show the first 100 frequency peak functions and their fits. The plots are sorted left to right and top to bottom by the number of votes. We have set the vertical scale for each subplot separately. Note that the strongest peaks in amplitude don't necessarily get the most votes. We do it like this in order not to miss weak frequencies, because perceptually the strongest are not always the most audible ones. The corresponding frequencies are depicted in Figure 5.3. The resulting vibration model can be visualized by plotting its frequency response in Figure 5.4. Note that the highest peaks do not necessarily correspond to the largest coupling coefficients, as the damping also plays a role in the spectral response.

Note that some modes seem to behave very linearly whereas others are almost completely random. In a refinement of the parameter fitting one could reject modes that did not produce an acceptable fit with a linear function. However, it is by no means clear that this would improve the model.

Some of the linear fits to the modes (such as the mode in the lower left corner,

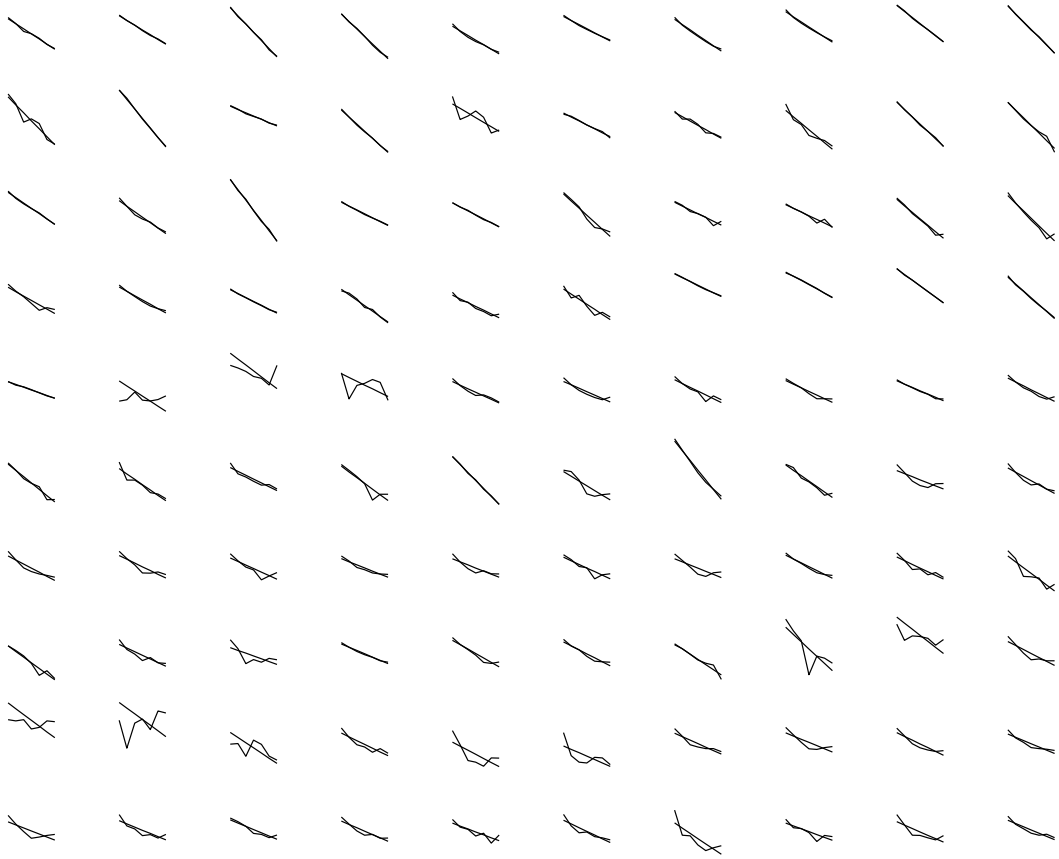

Figure 5.2: Identified modes for vase. Shown are the logarithmic amplitudes of the frequency peaks and their linear fits.

4425	4005	3822	3758	3467	3424	3391	3316	2993	2649
2390	2261	2078	1992	183	4996	4576	4177	3973	3865
3790	3208	2229	2121	2046	1109	6105	5846	5028	4619
4242	3941	3908	3714	3499	3165	3058	3036	2595	1518
1270	301	97	65	8656	7558	6611	6298	5922	5814
5060	4113	3650	3230	2961	2778	2186	484	10444	9109
8829	8742	8355	6826	6686	6643	6578	6492	6385	6072
5394	5243	5179	5136	4953	4037	3112	2175	1324	678
624	420	388	10993	10476	9615	8775	8269	8236	8053
7203	6891	6460	6223	5986	5717	5663	5599	5523	5426

Figure 5.3: Identified modes for vase. Shown are the frequencies corresponding to Figure 5.2.

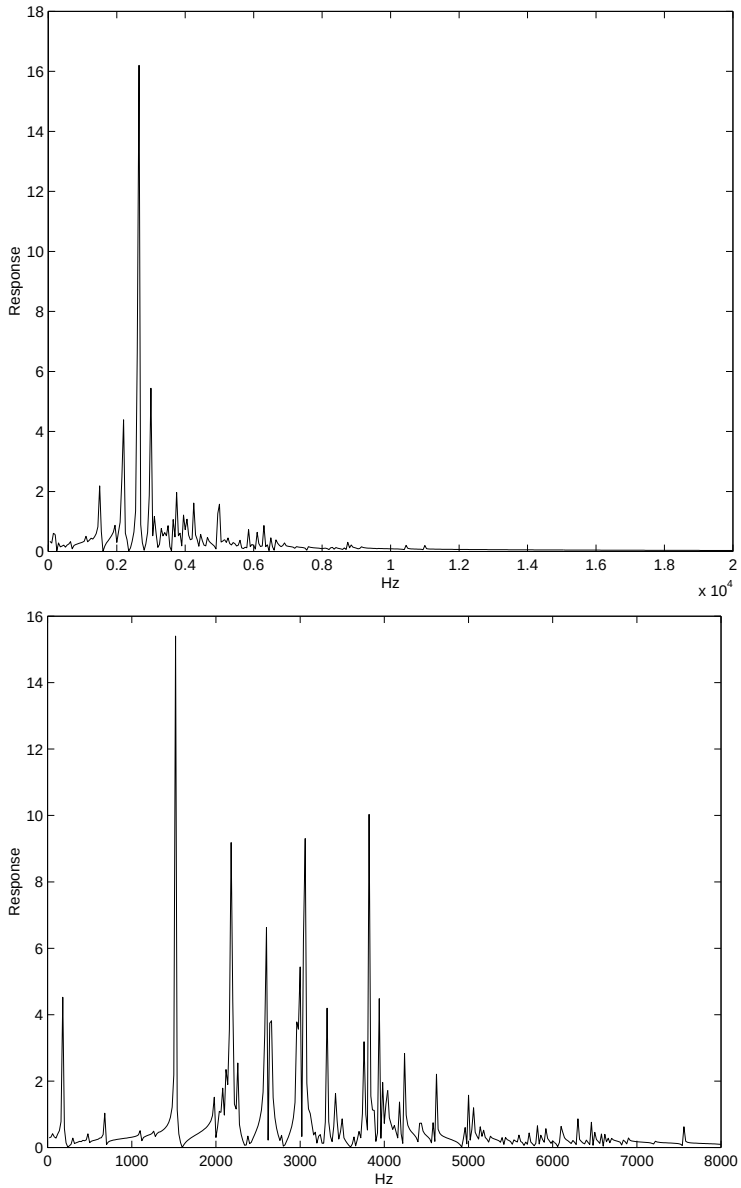

Figure 5.4: Identified modes for vase. The frequency response is shown. The bottom figure shows a subset of the frequency range of the upper figure.

one from the bottom) seem to be wrong. This is an artifact of the least squares algorithm we used, which only works correctly if the best fit has negative slope, which corresponds to a positive damping coefficient. Negative damping modes are artifacts caused by trying to fit noise. In all cases their amplitudes were so small that they do not contribute to the resulting sound. We preferred to have the algorithm produce positive damping always, even when a least-squares fit would result in a negative damping, as an undetected negative damping coefficient causes instability in the synthesis algorithm, and must be removed before using the data for synthesis. This is straightforward but the result of forgetting this can be rather unpleasant. [1] A very weak but positively damped spurious mode is harmless, on the other hand.

Some modes may appear to be oscillating, which can occur easily if two frequencies are approximately degenerate. Degeneracy in the spectrum is closely related to geometrical symmetries, with the exact degeneracy being broken by higher order effects, and occurs frequently. Beating between the frequency doublet causes apparent oscillations in a frequency. Such behaviour has been observed in bells [35], kettledrums [53], and in timpani [77]. Such oscillating behaviour could be detected and one could attempt to fit such spectral lines with two frequencies. For the purpose of this research we have not found a compelling reason to pursue this further.

Various other refinements of the fitting method are possible, but have not been pursued in the context of this research. For example, one could work with a spline interpolation of the windowed discrete Fourier transforms, to obtain a more precise estimation of the frequencies. This would necessitate some form of frequency tracking, as we don't have discrete bins. For this an adaptation of the McAulay-Quatieri algorithm [49] for frequency tracking could be used. For rough models of

[1] It results in a horribly loud squeak which may damage the ear.

"common" objects little seems to be gained by this, as an accurate determination of the frequencies is not so important (unlike in musical instruments). Polynomial interpolated Hanning windowed Fourier transforms were used successfully in [77] to identify the spectral lines for timpani sounds.

After acquiring the model parameters we usually have to make some manual adjustments. Often background noise such as the hum of a refrigerator will show up as a spurious mode with zero decay constant. Often these modes are easily recognized by a visual inspection (or by some pruning code) and can be removed easily. They are also very audible, so when importing the constructed model to the model tester described in Chapter 7, these spurious modes will be discovered.

In Appendix B we show results for the vase at different values of the window size, as well as results for three other objects: a santur, a piano, and a metal computer tower. The piano and the santur parameters did not produce a good vibration model. This was expected as musical instruments are much more complicated than "common" objects. In particular the santur is a very complicated instrument. It consists of a set of strings on a wooden frame, three strings per note. The strings are struck with wooden hammers. Because of the coupling between the frame and the strings if a single string is struck, the other strings resonate. The sound is therefore extremely complex and thousands of modes would be needed in order to approximate even the linear response. We have included the analysis of this instrument to indicate where our method breaks down. The piano string, whose data was obtained by playing a note on a piano, produces a reasonable model, though hardly musically satisfying. The vase and the computer tower produced very realistic sounding vibration models for a modest number of modes, around $5 - 10$. The original sounds and the reconstructed impulse responses can be accessed on-line on [4], and on the

accompanying CD.

Our reconstruction assumed the recorded sound is the impulse response of the system. However in reality the impulsive force will have some finite duration. One could account for this if the exact force profile of the strike was known, by correcting the amplitudes a obtained by the algorithm with a factor which depends on the exact force profile of the interaction, as follows. The impulse response $y_0(t)$ and the response $y_F(t)$ to a finite duration force $F(t)$, (non-zero only for $0 \leq t \leq \tau$) are related by

$$y_F(t) = \int_0^\tau y_0(t-s)F(s)ds.$$

If the impulse response $y_0(t)$ is assumed to be of the form

$$y_0(t) = \sum_{k=1}^{n} a_k^0 e^{i\Omega_k t},$$

and the response $y_F(t)$ is assumed to be of the same form with different coefficients a_k^F, we can relate a_k^F and a_k^0 by

$$a_k^F = \gamma_k^F a_k^0,$$

where the correction coefficients γ_k^F are given by

$$\gamma_k^F = \sqrt{(\int_0^\tau e^{d_k t} sin(\omega_k t)F(t)dt)^2 + (\int_0^\tau e^{d_k t} cos(\omega_k t)F(t)dt)^2}.$$

We can now use the algorithm described previously, and divide the coupling coefficients by the correction coefficients γ_k^F to obtain the impulse response. For this the objects would need to be hit with a hammer equipped with a force sensor. We attempted to use the initial part of the recorded impulse response as a profile for the contact force, however this did not improve the results and we conclude that this is not an acceptable approximation. The resulting reconstructed sounds can be heard for the vase and the santur on the web page and on the CD.

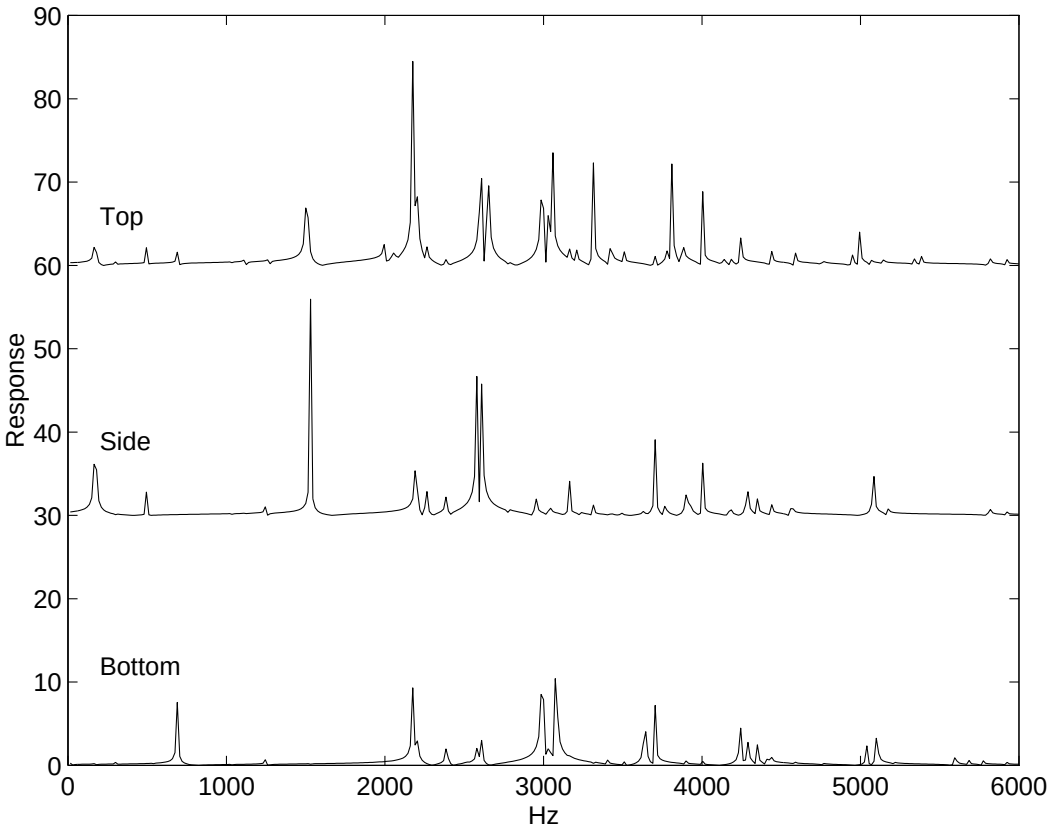

Figure 5.5: Frequency response of vase for three regions.

We also have tested the models by applying various input forces to the models taking into account different number of vibration modes and a selection of these sounds can also be accessed on the web page and CD.

In Figure 5.5 we show the frequency response, as reconstructed by our method, for three different locations on the vase at the top, middle, and bottom. As can be seen, many frequencies and dampings (which show up as the widths of the resonances) are shared but the amplitudes differ, as expected.

5.3 Designing Model Parameters by Hand

In addition to computing or measuring the parameters, there are also circumstances where a less automated approach is desirable. For example, we have been able to synthesize the sound of avalanches by a set of random resonances with frequencies and dampings within a specified range. The avalanche models can be generated on-line and heard on the Java applet on [4], and on the accompanying CD.

Also, the sound of "generic" objects such as unspecified structures of a certain type of material can be generated by choosing a set of resonances and controlling their distribution by stochastic methods. For example on the web page referred to above there is an object denoted by "pseudo string" which consists of a set of resonances which are spaced almost harmonically as in the ideal string model, but with some random perturbation.

To generate the sounds of combustion engines, described in more detail in Chapter 6, a stochastic model for the resonances is also found to be very effective.

In an application described in Chapter 7 we constructed a xylophone object, which was obtained by adding resonances with frequencies corresponding to the seven white piano keys (using just intonation) and adding a bar-like spectrum for each on top with coupling and damping coefficients adjusted by hand until it "sounded right". The xylophone was examined in great detail in [18], with particular attention to the mallet-bar interaction.

We have also tried to extract the model parameters from a recorded sound of a church bell "by hand". For this we used a software package to analyze the sound, do Fourier transforms, plot spectrograms etc., in order to identify important frequencies. After a time consuming analysis, it was found that the first 20 or so modes identified were also found by our parameter fitting software, and the

weaker modes were not audible. This provides a nice validation of the parameter identification method.

Although the "manual" approach did not give us a better model, in general one can take more extensive measurements of vibrating structures under a variety of interactions to arrive at a modal model. For example, steel drums [63], harpsichords [64], and guitar bodies [41] have been investigated using a variety of measurements.

Chapter 6

Interaction Force Models

To create sounds we need an interaction force model as well as a vibration model. In this chapter we consider four types of interaction models:

- Impact forces
- Continuous contact forces (sliding and rolling)
- Engine forces
- Live data streams

6.1 Impact Forces

When two solid bodies collide, large forces are applied for a short period of time. The precise details of the contact force will depend on the shape of the contact areas as well as on the elastic properties of the involved materials. For example, a rubber ball colliding with a concrete floor will experience a contact force which will increase faster than linearly with the compression of the ball, because the contact area also

increases during the collision. A generic model of contact forces based on the Hertz model and the radii of curvatures of the surfaces in contact was considered in [39]. A Hertzian model was also used to create a detailed model of the interaction forces between the mallet and the bars of a xylophone [18].

To create the simplest model of a collision force to drive the audio synthesis, we assume that the two most important distinguishing characteristics of an impact on an object is the energy transfer in the strike and the "hardness" of the contact. A psychophysical study of perceived mallet hardness [29] of xylophones showed that this is indeed a very perceptible parameter of an acoustic event. The hardness translates directly in the duration of the force, and the energy transfer translates directly in the magnitude of the force profile.

We have experimented with a number of force profiles and found that the exact details of the shape is relatively unimportant. A phenomenological model of a finite duration impact has been constructed and implemented. The model approximates the contact force by a function of the form

$$F(t) = F_{max}(1 - \cos(\frac{2\pi t}{T})) \tag{6.1}$$

for $0 \leq t \leq T$, with T the total duration of the contact. This function has the qualitative correct form for a contact force. The force increases slowly in the beginning, representing a gradual increase in contact area, and then rises rapidly, representing the elastic compression of the materials. An implementation of this contact profile can be found on the interactive demo's on the accompanying CD and on [4]and is found to be quite effective. The sounds of soft contacts (with large T) are recognizable as such, which shows that this model can produce this perception. We have also adopted this contact force model in a real time synthesis program described in Chapter 7, as well in other testing modules.

The spatial extension of the contact area will have an influence on the resulting sound. This effect can be incorporated easily in the linear model by averaging the amplitudes a over a region, which will have only a small effect on the sound for small contact areas. This is not an important effect and it will be ignored.

More complex interactions during contact such as damping effects may have an important effect in some cases. For example, the hammers in a piano are covered with felt, so during the contact between the hammer and the string they damp the higher modes of vibration. How this actually occurs is quite complicated [32, 33, 34, 76] and potentially important, especially for high quality musical instrument modeling. Another example is a ringing sword. When the sword is sliding over a surface, the continuous contact will provide extra damping, compared to a free ringing sword. This can be taken into account by adjusting the damping parameters appropriately during continuous contact.

6.2 Continuous Contact Forces

An important ingredient in synthesizing realistic scraping and rolling sounds is a surface interaction model. A lot of research has been conducted on models of contact interactions between solids [78, 79, 47, 42, 10, 9], but they usually focus on predicting forces at a coarser time scale than needed for our purposes. An recent exception is [83]. Nevertheless a rigid body simulator would be able to provide information about the contact force magnitudes and the friction forces at the contact areas which may be used as inputs for a high sampling rate contact force model.

This model should be able to generate contact forces for a specific type of contact depending on the contact force and the sliding speed. The roughness profile of both surfaces will determine the effective force stimulus to the object

and therefore have an important effect on the sound. Initial experiments indicated that bandwidth limited white noise and Gaussian noise (in the time domain) are reasonably satisfactory, but they lack a surface property parameter.

We have also experimented with scaling (fractal) noise. This is noise with a spectral content which behaves as f^{α} (at least over a sizable region), for some constant α. Such noise (of which white noise is a simple example with $\alpha = 0$) sounds the same when played back at any speed, and is extremely common in nature [89]. Especially $1/f$ noise [85, 88] occurs frequently, and is a field of study by itself. The exponent α can be used as a roughness parameter, $\alpha = 0$ being a very rough surface, and $\alpha = 1$ representing a smoother surface.

The most satisfactory solution we found is to use a looping digital sample with pitch shifting and volume control to adjust the speed and force of the contact. With a small set of short samples representing a variety of textures a great variety of contact surface profiles can be imposed upon the vibration models. Surface profiles were created by simply scraping a real object with a contact microphone. See the accompanying CD or [4]for a palette of scraping sounds and their use.

The physical picture behind this contact model is that the sample encodes the shape of the surface and the object scraping it is following this surface profile exactly. In reality microscopic deformations and breaking of contacts will complicate the mechanism whereby the interaction force is generated. These interactions are so complex that it seems hopeless to try to model the physics behind this in real time. We therefore believe the pitch-shifting sample playback approach is the best way to model these type of contacts. If it is necessary to take into account that the contact force does not simply scale in time at different speeds, one can have a numbers of samples for different contact speeds and cross-fade between them. In essence, this is

the technique used presently for wave table synthesis of musical instrument sounds, but used here for the contact force. Viewed in this way, our synthesis can be thought of as an "effect" added to a recorded sample, in this case representing an interaction force, instead of a sound.

6.3 Live Data Streams

An interesting interface to this type of synthesizer is a sensor that measures real interaction forces. We have implemented this very simple (and cheap) with a contact microphone (designed to be attached to a guitar body) attached to a real object. When touching and scraping real objects the audio signal is sent to our real-time simulator and the audio signal is interpreted as a force to whatever vibration model is currently loaded. We can then scrape some interface object and transfer the measured signal to the audio synthesis to create the impression of touching a virtual object. Another type of application is to use the output of an electrical guitar as the driving force for some virtual guitar body. In Chapter 7 we will describe some applications written around this idea.

6.4 Engine Forces

Engine sounds are very difficult to achieve in computer games, as they are continuous sounds. Racing games are very popular and a properly modeled car sound that responds in a realistic way to input parameters would greatly enhance the audio in such games.

It is not obvious that combustion engines can be modeled with our techniques, as the sources of sound are explosions and very complicated gaseous phe-

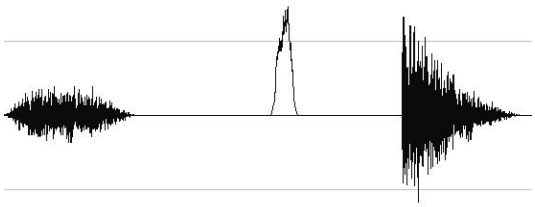

Figure 6.1: Sample driving four stroke one cylinder engine.

nomena. Rather surprisingly, we found that very good sounding interactive models can be made by driving some resonance object (a lumped model of everything that is vibrating) with a rather simple-minded model of a combustion engine.

The first model we created is a 4-stroke engine. The driving force is obtained by constructing a looping audio sample divided into four regions which represent the 4 stages of the engine. The sample depicted in Figure 6.1 contains an intake stroke, a compression stroke (silence), a combustion stroke, and an exhaust stroke. The intake stroke was modeled as white noise enveloped with a bell curve. The exhaust stroke is modeled as white noise, rapidly decaying in time, inspired by a high pressure gas mixture being released when the valve opens. The combustion stroke consists of an enveloped burst of $1/f$ noise. It was found, after trying various $1/f^{\alpha}$ noises, that this gives the most realistic sound. The reason is probably that the combustion takes place inside the cylinder, so the shock wave is transmitted to the mass of metal of the engine block, which acts as a low pass filter.

The sample is looped at adjustable rate, corresponding to the running speed of the engine, but we keep track of the four stages in the sample and allow the volume of the intake, combustion, and exhaust stages to be set in real-time independently of each other. This gives extra dimensions of control to map for example to driving

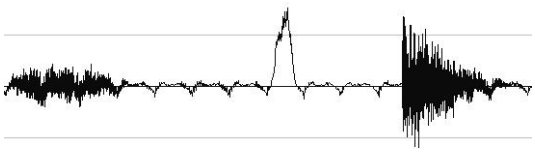

Figure 6.2: Sample driving four stroke one cylinder engine with fan.

uphill, with a large load, etc.

This simple driving force model can generate a variety of engine sounds by coupling it to various vibration models. Models with relatively high frequency resonances with high damping give a "lawn mower" effect, whereas a low frequency object gives a very convincing motorcycle sound. The best motorcycle sound was obtained by using a mathematical model of a rectangular plate with dimensions 7 by 1 and lowest resonance at $50Hz$. About 7 modes gives an optimal result.

A slightly different sound is produced by adding a background pitched sound to the sample at all four stages. This simulates the sound of the fan, and perhaps other rotating parts. For the pitched sound, a short noisy note played on a $70cm$ long pipe with holes (a.k.a. a ney) was used, which is very satisfactory. The driving sample is depicted in Figure 6.2.

A race car engine sound was obtained by creating a four cylinder version. We assume the four cylinders fire $\pi/2$ out of phase and simply add the samples from the one cylinder engine four times, with a relative phase shift of $\pi/2$. The resulting sample is depicted in Figure 6.3. If we just use this driving force as is the result is not very good. The reason is that in a real engine the four cylinders are attached to the intake manifold at different locations and therefore sound different. To incorporate this we adjust the volumes of the four one cylinder samples individually and when

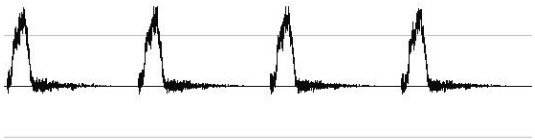

Figure 6.3: Sample driving four cylinder engine.

they are set all different the resulting sound is very convincing.

In a real game application one would probably spend more time on the actual modeling of the car engine than we have done here. But our simple models show that a convincing result can already be obtained using extremely simple models. To create richer engine sounds, different cylinders may be coupled to different exhaust manifolds or muffler pipes. It is interesting that the Harley Davidson motorcycles are designed specifically to generate their characteristic sound. Sub optimal engine performance is tolerated, just to get their characteristic sounds. An interesting article on the audio aspects of the Harley Davidson motorcycle by Joel C. Moser appeared as the 1996 winner of the Streuben essay competition of the College of Engineering at the University of Wisconsin-Madison. The article is available on the WWW at [5].

Chapter 7

Implementations

The theory presented in the previous chapters was tested in practice in order to evaluate the quality of interactive audio created with the methodology in several applications.

In this chapter we will first describe the general architecture of applications with real-time audio synthesis and describe the software components for audio synthesis that have been implemented in C++ and in Java. Following this, we will discuss authoring tools that we created, and finally we will describe several demonstration programs that have been constructed to illustrate what can be done with the audio synthesis.

7.1 General System Architecture

In order to apply the theory developed here to the creation of interactive audio in software applications (such as simulations) we need to implement software components which will have to be integrated with the application. The low level audio synthesis and the control algorithms should be designed as orthogonal as possible.

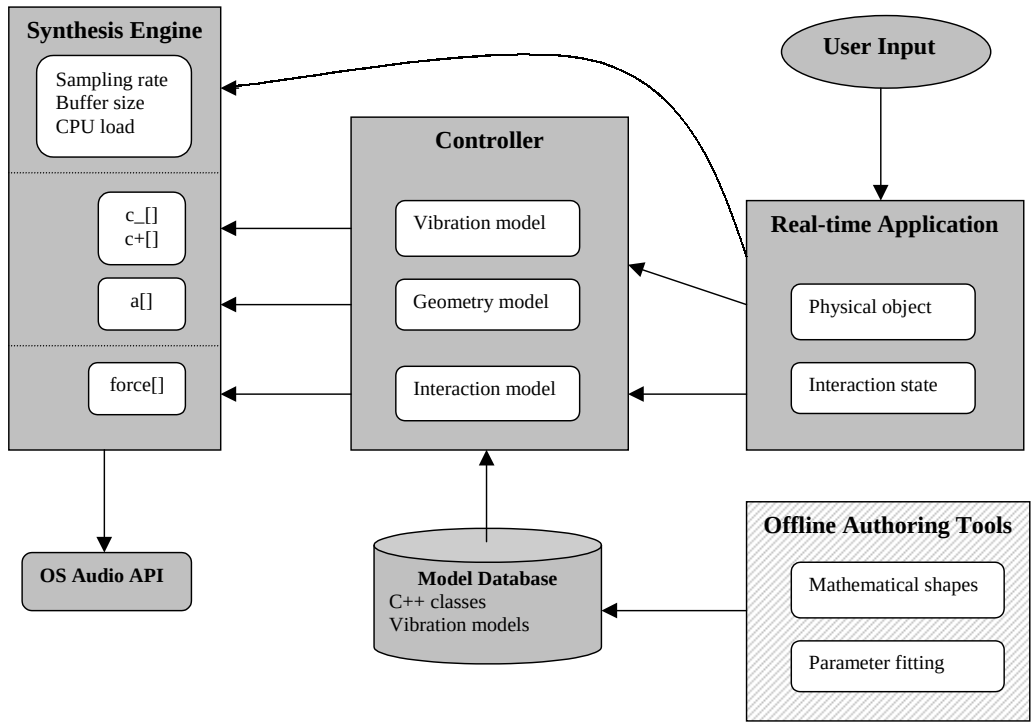

Figure 7.1: System architecture for applications with real-time audio synthesis.

We therefore have divided the runtime system into two main components:

- **SynthesisEngine.** This component represents a real-time thread or process that continuously writes computed audio buffers to the audio rendering hardware. This object implements the core synthesis algorithm at the lowest level.
- **Controller.** This controls the SynthesisEngine by feeding it the necessary parameters. This object encapsulates composite models of objects and interactions and computes the filter coefficients and input buffers from higher level parameters such as sliding speed, throttle settings, etc.

The SynthesisEngine communicates with the audio hardware by repeatedly submitting a computed audio buffer of duration T using the audio API of the operat-

ing system. The operating system will provide an abstraction of the audio hardware in the form of a queue on which audio buffers can be placed. The SynthesisEngine must submit buffers at a rate of $1/T$, or the queue will underflow, resulting in spurious noises. The buffer size T will determine the latency of the synthesis algorithm (the delay between an interaction and when it is heard) and the control overhead. We discuss this in more detail in Section 7.1.1. Before computing the next buffer, the SynthesisEngine will obtain information from the application about the object whose sound it should render and about the force applied to it. This information is obtained through the Controller class, which is responsible for making the coefficients c_+, c_-, a, and the force F (see Equation 4.18) available for the duration T of the buffer.

The SynthesisEngine will then compute the modal contributions of each mode as defined by the parameters obtained from the Controller one by one. After each mode is added, it checks if it has time to compute another mode. If its allotted time has run out or if all modes have been computed, it will place the computed audio buffer on the output queue and start computing the next buffer. The time allowed for the computation of a single buffer should be less than T, in order to insure an uninterrupted audio stream. By allowing significantly less time than T for the completion of the buffer, one can effectively force the synthesis algorithm to use only a small fraction of the available computational resources of the system. If more resources become available (for example when a computationally intensive task in a game is finished), more modes can be added and the quality of the synthesized sounds will adjust dynamically to the load on the system.

For a buffer length of T, the latency of the synthesis will be between T and $2T$. The force $F(t)$ will become available after a time interval T, and the synthesis

algorithm will have to finish its computation of the audio buffer after another time interval T, because after that the next buffer F will have to be processed. In practice, the audio synthesis will often be required to use only a small fraction of the computational resources of the host machine, and will be required to finish in a time much shorter than T. To achieve minimum latency, we should choose T to be as small as possible. However, there is additional overhead for each buffer, as the model parameters may change. For large values of T this overhead can be made as small as desired, at the expense of more latency.

Let us denote the computational cost of a multiplication by $C(\text{mult})$ and the computational cost of an addition by $C(\text{add})$, and let S_R be the sampling rate and n the number of modes. The computational load per second of the synthesis will be

$$L_{\text{synth}} = (3C(\text{mult}) + 4C(\text{add}))nS_R. \tag{7.1}$$

The control overhead is assumed to have a computational cost of $C(\text{control})$ per mode, and the control parameters are recomputed every T seconds. The control overhead will be

$$L_{\text{control}} = nC(\text{control})/T \tag{7.2}$$

per second. As an illustrative example let us consider an object whose overall frequency scale will be modified in real time. In this case the frequencies will have to be multiplied by a scale factor which will cost $nC(\text{mult})$. Then the coefficients c_+ and c_- appearing in Equation 4.18 have to be recomputed giving a total cost per mode of

$$C(\text{control}) = C(\text{exp}) + 2C(\text{trig}) + 2C(\text{div}) + 4C(\text{mult}) + 2C(\text{add}) + C(\text{sqrt}),$$

where $C(\text{exp})$ is the cost of an exponentiation, $C(\text{trig})$ is the cost of computing a trigonometric function, $C(\text{div})$ is the cost of a division, and $C(\text{sqrt})$ is the cost of

taking a square root.

The condition under which the control cost can be neglected, $L_{\text{control}} \ll L_{\text{synth}}$, reads

$$C(\text{control})/T \ll (3C(\text{mult}) + 4C(\text{add}))S_R \tag{7.3}$$

In practice a buffer length of about 50 ms, corresponding to about 1000 samples at $22,050$ Hz was used. The interface to the audio hardware on the computer introduces additional latency and requires a certain minimum buffer length for the audio queues. In practice this also determines the buffer length used in the synthesis implementation. In this case the condition given by Equation 7.3 becomes

$$C(\text{control}) \ll (3308C(\text{mult}) + 4410C(\text{add})),$$

which is easily satisfied.

The runtime components are implemented as C++ and/or Java classes. The application writer will create a properly configured SynthesisEngine object, which will start the low level synthesis thread, but not interact with it directly. The SynthesisEngine is given access to a Controller object, which encapsulates the task of translating physical parameters such as resonance frequencies and excitation forces into appropriate control parameters for the low level synthesis algorithm. The relation between the components and the application is depicted in Figure 7.1.

We will now describe the run-time components in more detail.

7.1.1 SynthesisEngine

The SynthesisEngine implements the audio synthesis algorithm defined by Equation 4.18. The class definition in simplified form is as follows:

```
class SynthesisEngine {
```

```
private:
  int samplingRate;
  int bufferSize; // =T*samplingRate
  double cpuFraction;
  SynthesisEngineController *sec;
public:
  // Define controller object
  int setSynthesisEngineController(SynthesisEngineController *sec);
  // Set output buffer size
  int setBufferSize(int bufsz);
  // Set fraction of CPU load to use
  int setCPUFraction(double fraction);
  // sampling rate in Hertz
  int setSamplingRate(int samplingRate);
  // Create synthesis thread
  int run();
  // Destroy synthesis thread
  int stop();
};
```

When a SynthesisEngine object is created and its `run()` method is called, a thread is started which implements the following pseudo code:

```
buf[bufsz]; // Audio buffer to be computed
uPrev[] = vPrev[] = 0; // Zero'd arrays
while(not_stopped) {
  Obtain arrays a[], c_[], and c+[] from Controller;
```

```
// See Equation 4.18 for these coefficients
Obtain array F[bufsz] from Controller;
N = length of c[] array, ie., number of modes;
buf[] = 0;
time_allowed = fraction * bufsz / srate;
start_time = getTime();
for(i=0;i<N && (getTime()-start_time)<time_allowed;i++) {
  // Add contribution of mode i to buf[] and
  // recover the state variables from the previous time slice
  u_prev = uPrev[i];
  v_new = vPrev[i];
  // Load parameters for this mode in register
  ccplus = c_[i];
  ccminus = c+[i];
  aa = a[i];
  // Add this mode to output buffer
  for(k=0;k<bufsz;k++) {
    u_new = ccminus * u_prev - v_new + aa * F[k];
    v_new = ccplus * v_new + u_prev;
    u_prev = u_new;
    buf[k] += u_new;
  }
  // Save state variables for next buffer
  uPrev[i] = u_new;
  vPrev[i] = v_new;
```

```
  }
  // Computed as many modes as time allows, now send to audio hardware.
  submitBufferToAudioHardware(buf);
}
```

The inner loop adds the contributions from the modes one by one, until either all modes are computed or the allotted time has run out. If other processes are running at high priority on the machine, less modes will be added in the time allotted than if the synthesis thread has more resources available. In this case there will be a graceful degradation of the audio, provided that there is enough time to computed the "essential" modes of the object. [1] When the computed buffer is submitted to the audio hardware, it will be put on a playback queue and the function `submitBufferToAudioHardware()` is assumed to block if the queue is full, thereby taking care of the timing of the synthesis loop. This function needs to be called at a rate of `samplingRate/bufferSize` to keep the audio stream moving at the correct rate.

How `submitBufferToAudioHardware()` is implemented depends on the operating system on which the application is built. The operating system's audio API will also impose restrictions on latency, in addition to those associated with the value of the buffer size used in the actual synthesis algorithm.

We have implemented the SynthesisEngine on top of the SGI IRIX AL audio API and on top of the Microsoft DirectSound API. On AL, we were able to achieve a total latency of about 50ms at a sampling rate of 22,050 Hz. Using DirectSound, the lowest latency we could achieve was about 100 ms. It is to be expected that the latter figures will come down when the DirectSound implementation matures.

[1] We assume that the granularity of the thread scheduling on the processor is fine enough.

	C	Java (Compiled)	Java (JIT)
Max. modes	450	250	150
CPU load for 10 modes	2.2%	4%	6.7%

Table 7.1: Performance of synthesis algorithm at a sampling rate of 22,050 Hz on a 233 Mhz Pentium PC with MMX using a C implementation and a Java implementation.

The latency of the IRIX implementation is caused by the minimum length of the input queue buffer. If necessary, this buffer can be bypassed and the latency can be made as small as desired. As the achieved latency is quite satisfactory, we have not pursued this possibility. Bypassing this buffer also requires the process to run with root privileges, which severely limits its deployability.

In order to test the speed of the algorithm, we timed the inner synthesis loop on a Pentium 233 MMX PC. The results are summarized in Table 7.1. At the sampling rate of 22,050 Hz, using a C implementation of the synthesis loop, we found that we can synthesize a maximum of 450 modes. This means we can compute about 5 modes utilizing 1% of the computational resources on average. [2] We also benchmarked the algorithm in Java. With the Symantec JIT compiler version 3.00.029(i) we were able to synthesize a maximum of 150 modes, about three times slower. Compiling the Java code into an executable allowed us to increase the performance to 250 modes, which is slightly better than twice as slow as the C implementation. We have not implemented the algorithm in assembly language, which could provide a significant performance boost.

[2] This estimate assumes that there are no exceptional events such as page faults, context switches, etc.

7.1.2 Controller

The Controller class is an abstract base class and is subclassed by the application writer to encapsulate concrete audio models of objects. The Controller will usually load data from a database of vibration models created with various authoring tools described in more detail below. It translates parameters such as throttle setting or contact sliding speed into the low level parameters that the SynthesisEngine needs. In bare-bones form the class is given by:

```
class SynthesisEngineController {
public:
  // Callback function to get current filter parameters
  virtual RTSFilter *callBack(
    int *location_index,const double *force,int srate,int buflen) = 0;
};
```

The function `callBack()` is called by the SynthesisEngine thread and returns a pointer to a `struct`, `RTSFilter` (real Time Synthesis Filter) which has the following form:

```
#define MAX_SPECTRUM 100 // Max. of 100 modes
#define MAX_LOC 10       // Max. of 10 locations
typedef struct {
  int nf; /* Number of components */
  int np; /* Number of points on object*/
  double c_plus[MAX_SPECTRUM];
  double c_minus[MAX_SPECTRUM];
  double a[MAX_LOC][MAX_SPECTRUM];
```

```
} RTSFilter;
```

This `struct` contains the parameters necessary to set the coefficients in Equation 4.18, for a set of "locations" on the object. A "location" is just an integer $0 \leq i < np$, which indexes the arrays $a[\,][\,]$. The translation to this linear array of "locations" from geometrical concepts used in the application will be done at a higher level in the code layers.

The floating point array `force[ ]` represents the force applied to the object. The parameter `buflen` indicates the number of samples to be returned in the force buffer and is set by the SynthesisEngine thread. Similarly the parameter `srate` is used to inform the application of the current sampling rate used by the SynthesisEngine.

The member function `SynthesisEngineController::callBack()` is overridden by the user (application programmer) of the SynthesisEngine class and also returns the location index of the vibration model in the pointer variable `int *location_index`.

Example: MouseTouchableObject

The first example (used in the demonstration program described in Section 7.3.7) uses this base class to define a controller allowing the user to scrape virtual objects with the mouse:

```
class MouseTouchableObject : public SynthesisEngineController {
private:
  double m_scrapeSpeed;
  double m_scrapeForce;
  double *m_heightProfile;
```

```
  SonicObject *sob;
public:
  int setScrapeSpeed(double);
  int setScrapeForce(double);
  int setSonicObject(SonicObject *sob);
  int setSurfaceProfile(double *heightProfile,int buflen);
};
// Implementation of callBack()
RTSFilter *MouseTouchableObject::
callBack(int *location_index,const double *force,int srate,int buflen)
{
  *location_index = 0; // Only one location
  // Maintain a circular pointer m_p in the buffer m_heightProfile. Copy
  // a section of length buflen*m_scrapeSpeed into tmpbuf[], and advance
  // m_p  by this amount. Resample tmpbuf[]  to  contain buflen samples,
  // scale with  a factor m_scrapeForce and  copy in force[].  This is a
  // pitch-shifted segment of the height profile buffer.
  return sob->getRTSFilter(srate); // return the filter parameters
}
```

The interface metaphor used in this class is a surface profile, whose shape is stored in the array `m_heightProfile`. This surface is touched by an object with force `m_scrapeForce` and speed `m_scrapeSpeed`, and the resulting excitation force can be obtained by looping through the circular array `m_heightProfile` at speed `m_scrapeSpeed`, resulting in a pitch shift of the buffer.

The `SonicObject` class is a wrapper around a parameter set for a vibration model of an object. It contains data members for frequencies, dampings, and amplitudes, and can be loaded and saved from file. It stores its data in the `sy` file format explained in Appendix A. The member function `getRTSFilter(int srate)` computes the actual coefficient arrays c_-, c_+, and a (see Chapter 4) from the higher level model parameters. [3] It also provides member functions to change the material properties by rescaling frequencies, changing the material constant, etc.

Example: AudioSignalProcessor

The second example (used in the demonstration programs described in Section 7.3.7, and in Section 7.3.3) uses the base class to define a controller that will obtain the input force from the input channel of the audio hardware. In this manner, the SynthesisEngine can be viewed as an audio effects processor.

```
class AudioSignalProcessor : public SynthesisEngineController {
private:
  SonicObject *sob;
public:
  int setSonicObject(SonicObject *sob);
};
// Implementation of callBack()
RTSFilter *AudioSignalProcessor::
callBack(int *location_index,const double *force,int srate,int buflen)
{
  *location_index = 0; // Only one location
```

[3] It caches the result and computes this only once, unless model parameters change.

```
  getInputBufferFromAudioHardware(force,buflen); // Get input from audio
  return sob->getRTSFilter(srate); // return the filter parameters
}
```

The function `getInputBufferFromAudioHardware(double *,int)` uses the operating system's audio API to obtain a buffer representing the input audio signal. This can originate for example from a microphone.

7.2 Authoring Tools

In this section we will discuss some authoring tools that were created to facilitate the creation of vibration models.

7.2.1 Off-line Model Tester

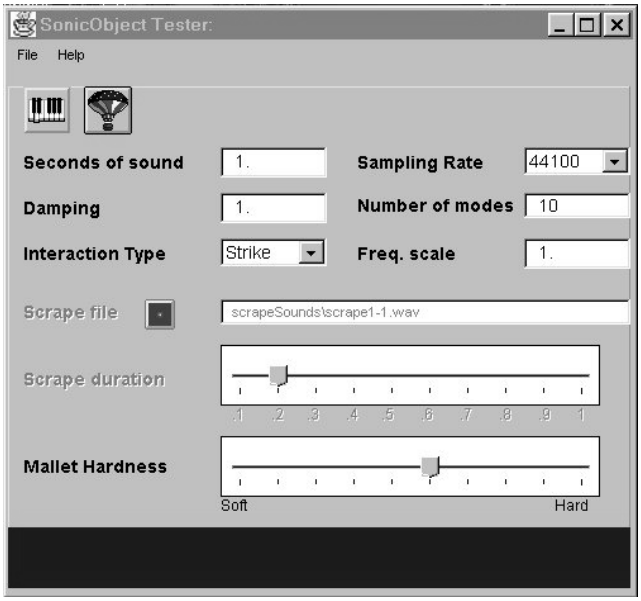

Figure 7.2: Off-line model tester.

This utility allows the testing of a vibration model by applying various forces

to it and writing the resulting audio to disk as a `wav` file. This allows a user to fine-tune the model parameters and to analyze in detail the sounds made. The interface (shown in Figure 7.2) requires the user to select a vibration model from disk by loading an `sy` file. The vibration model can then be excited in three ways. The object can be struck with a virtual mallet with adjustable "hardness". This parameter corresponds to the width of the peak of the excitation force, which has the form given in Equation 6.1. The object can be scraped with white noise and the interaction force can also be loaded from disk as a `wav` file.

The parameters of the vibration model can be modified in a number of ways. The dampings and frequencies can be scaled by constants, the sampling rate can be set, and the number of modes can be set.

The sounds on the web page [4], and on the accompanying CD referred to in Section 5.2 were generated with this tool. The tool is written in portable C++ with a command-line interface. A GUI written in Java was wrapped around this.

7.2.2 Vibration Model Generators

Two command-line tools were used to construct vibration models. A parameter fitting module implemented in MATLAB was discussed in detail in Section 5.2. We have used it to create models of vases, pots and pans, stringed instruments, hockey sticks, bells, boxes, plates, glasses, cups, basketballs, swords, and more. At a sampling rate of 44,100 Hz we would typically generate models with 100 modes at window sizes of $N = 1024, 2048, 4096, 8192$, or about $25 - 200ms$. We would then try the resulting models with the off-line tester described in Section 7.2.1. Sometimes spurious modes need to be eliminated manually. These modes arise from constant background noise and lead to frequencies with zero decay rate. They are

immediately audible, and can also be recognized by inspecting the `sy` file. Usually one or two window sizes give results that are considerably better than the others and those would be selected. As expected, the results were best for objects with a rather simple spectrum. None of the sounds were "photo-realistic", except perhaps the hockey stick. But in an interactive setting the sounds were convincing.

A tool to create vibration models of mathematical shapes as described in Section 5.1 was implemented in C. The shape, material, size, and other relevant parameters are given as command line arguments and the module produces a sequence of `sy` files.

7.3 Demonstration Programs

7.3.1 Sonic Explorer 1.0

We have constructed a test bed application in C++ for SGI, called the "Sonic Explorer", which demonstrates the enhancement interactive audio can bring to a three dimensional graphics environment.

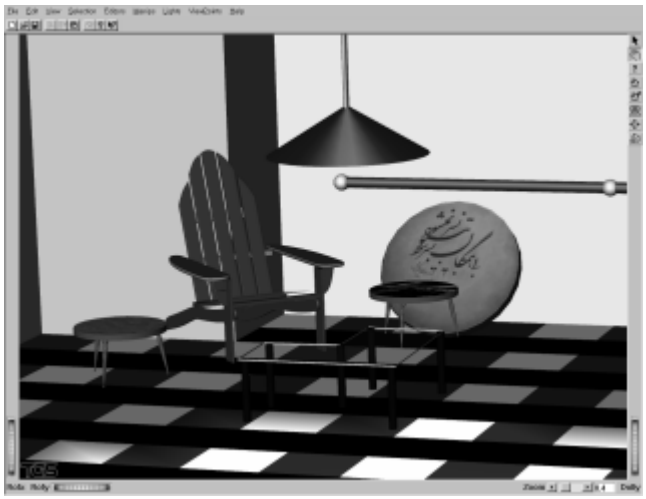

Figure 7.3: A room modeled with the Sonic Explorer

The first version of the Sonic Explorer presents to the user a three dimensional model of a room with various objects as depicted in Figure 7.3. The user can view the room from any point by moving a virtual camera. The purpose of this program is to evaluate the quality and usefulness of simple impact sounds in a real environment. This demo does not use real-time synthesis, it is intended to test impact sounds only.

Several objects in the room were modeled by vibration models of mathematical shapes. The impulse responses of these objects were computed on a grid of locations using 10 points in each dimension and the resulting audio samples were stored on disk. The user can move a virtual drumstick around and hit objects at various locations and the corresponding audio sample is identified and played back. The material of the objects was modeled by the method described in Section 4.5.

Even though the interactivity is very limited in the demo, the effect of changing sound as objects are struck at different locations and the material of the objects are conveyed quite well with these simple models.

7.3.2 Sonic Explorer 2.0

The second version of the Sonic Explorer implements a real-time synthesis engine as outlined in Section 7.1 using the IRIX audio library. Geometrical models of a vase (see Figure 7.5) and of a xylophone (see Figure 7.4) were created using the Open Inventor toolkit. A vibration model of the vase was constructed using the parameter fitting module discussed in Section 7.2.2. Data from three regions on the vase the top, middle, and bottom, was used and integrated manually into an `sy` file with three locations. The vibration models for the eight xylophone bars were constructed manually from the solutions of the free bar equations and the

Figure 7.4: A xylophone modeled with the Sonic Explorer

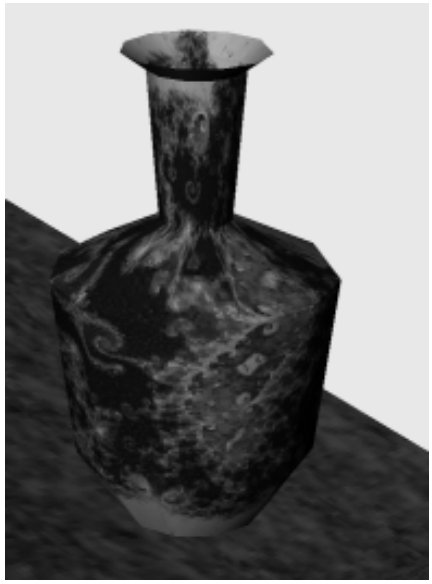

Figure 7.5: A vase modeled with the Sonic Explorer

base frequencies were tuned to form a "white key" scale of pitches tuned in just temperament.

A special interaction device, called the Sonic Probe was constructed to allow the user to touch these objects by tapping and scraping. The probe utilizes a graphics tablet to move the mouse pointer. When the mouse pointer is on a the vase or on a xylophone bar the appropriate action is taken by the controller, as described in Section 7.1.2, and the SynthesisEngine will model the corresponding vibration model. The stimulus force on the virtual object is obtained from a contact microphone embedded in the pen. When the user touches the graphics tablet the local interaction, caused by tapping or scraping, is picked up by the contact mike and used as the input signal for the audio synthesis as described in Section 7.1.2. A small cover of rough wood was optionally placed over the graphics tablet. The surface of the wood is rougher than the smooth plastic of the tablet and in this way we obtain a more interesting signal. In this manner we effectively map the surface structure of the wood to the object being modeled.

An additional feature is that the vase can be resized in the vertical direction, stretching or shrinking it, and the frequencies of the modes are scaled according to the length of the vase. Even though this is not the actual way the modes will change if the geometry of a vase is changed in this manner, it illustrates the concept quite well.

7.3.3 Sonified Objects

This application uses the idea of picking up real interaction forces with contact microphones and mapping them on virtual sonic objects in a non-graphical context. The SynthesisEngine was extended to process two independent input streams and

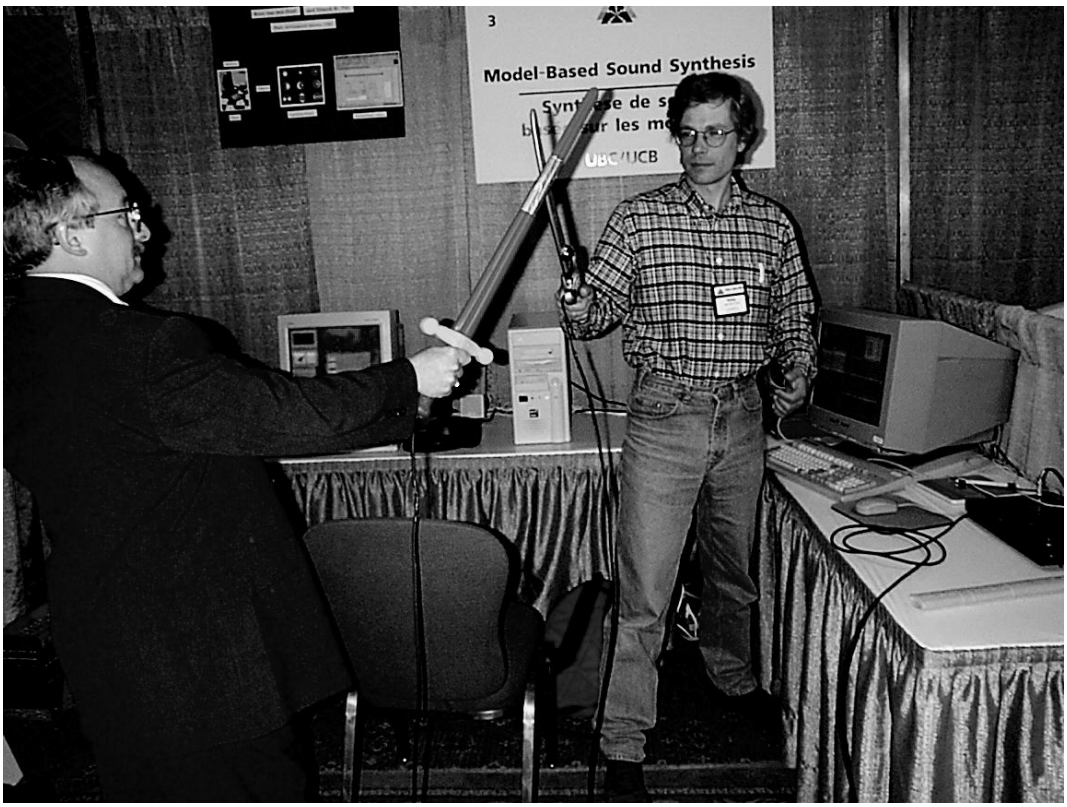

Figure 7.6: Plastic swords with embedded contact mikes used to create metal sword sounds.

apply the virtual forces to two independent objects and to render the audio to different speakers. We attached contact microphones to various objects and obtained interesting effects by feeding these data streams to various vibration models.

Two plastic toy swords were equipped with contact mikes and two different metal sword audio models were used to render the sounds. See Figure 7.6. Amusing effects could be obtained by changing the model to church bells or other inappropriate sounds.

We have also used this setup (with one data stream) to create a digital effect box for an electric guitar. Very interesting effects were obtained by hooking the guitar up to vibration models of plates, objects with randomized modal structures,

bells, etc.

7.3.4 Avalanche Sounds

An implementation of the synthesis algorithm was made in Java. Unfortunately Java 1.1 does not provide any method to render computed sound in real time. It even doesn't provide a method to render computed sounds by any means whatsoever. Fortunately an "unofficial" audio API (the "sun.audio" library) does provide a minimal API to allow the playback of computed arrays of 8-bit audio data at the sampling rate 8,000 Hz.

We have created a small package of classes to allow the creation of sonic objects and forces. The force can then be "applied" to the sonic object and the resulting audio can be saved in a buffer and rendered later. An advantage of this is that the sounds are created locally and need not be transmitted over the network. Viewed in this way the synthesis is the ultimate compression method.

The class hierarchy and the complete documentation for the Java synthesis classes are available on the accompanying CD and on [4]. The classes were used to create parameterized avalanche sounds in the applet depicted in Figure 7.7. An avalanche sound is generated by applying white noise to a vibration model containing an adjustable number of modes with frequencies randomly generated within a selectable frequency band. These are called the "rumbling frequencies". The stimulus force consists of modulated white noise whose time envelope can be controlled by four set-points. The avalanche sound is computed when the button "Compute" is pressed and played by pressing "Trigger". By repeatedly pressing the trigger button overlayed sections of the pre-computed sounds can generate a certain amount of interactivity. It is possible to re-compute the sounds while they are playing and

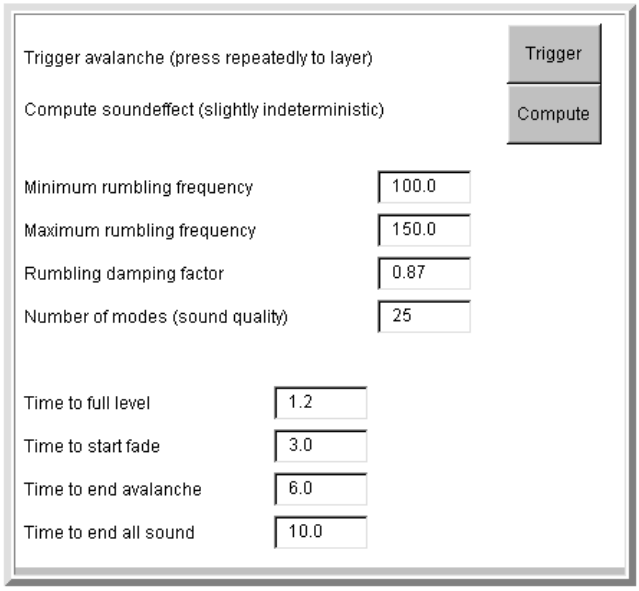

Figure 7.7: Applet to generate avalanche sounds.

every time a new set of random modes will be created, providing never repeating sounds. By setting the maximum frequency to higher values other interesting sounds reminiscent of wind or even eerie musical effects can be created.

7.3.5 Location Dependent Sounds of Mathematical Shapes

If the ideas presented here are to be used in a virtual reality environment, better tools will need to be provided to the user to specify the sonic attributes of objects. A graphical design tool that allows a user to define the types of sounds made by an object interactively would be very useful. For example, a user will describe an object as a circular metallic plate with a base frequency of 415 Hz. The user should then be allowed to poke and scratch the virtual object and adjust the model parameters until the object behaves as desired. A prototype has been implemented as a Java Applet, which allows a small class of objects to be created. See Figure 7.8. The object selected is depicted graphically and the user can hit it and scrape it

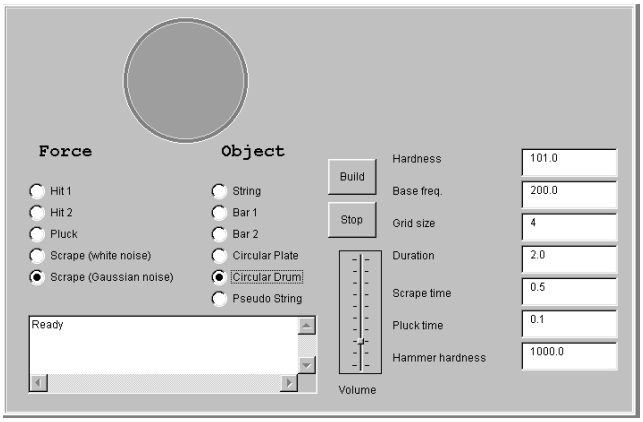

Figure 7.8: Applet to create location dependent sounds for a variety of objects.

and adjust the model parameters. Besides as a prototype of a sonic design tool, the program is also useful to test contact models. For example, the finite duration contact model described in Section 6.1 is currently implemented and this allows an immediate evaluation of the model.

By repeatedly hitting an object, possibly at different locations, some interactive control can be exercised. Every mouse click generates an audio thread and many can be overlayed, through which dynamic effects can be created.

7.3.6 Virtual Bell Tower

A third Java applet was created to show an application which is both entertaining, small, and musically interesting. The applet maintains models of eight bells (a bell tower) and allows the user to select among different bell models, retune the bells using various presets or custom, play them interactively with the mouse, change the hardness of the mallet used to strike the bells, and sequence "change ringing" patterns.

English bell ringing [40, 92, 19] is a complex mathematical art-form based

on ringing tuned bells in a steady rhythm in such a way that the bells ring all, or a subset of, the possible permutations of the bells. Typically six, eight, ten or twelve bells are used, but as few as three or as many as sixteen tower bells can also be rung to "changes".

A mathematical notation system to describe a "method" exists. This "place notation" system can be viewed as an algorithm for generating permutations. The applet implements a place notation parser to allow the interested virtual change ringer to try out methods. A collection of ancient traditional methods can also be selected from the pull-down menu.

Apart from the synthesized bell sounds, it is also possible to use wave table playback. For this purpose, eight recordings of actual church bells have to be loaded. Though the quality of the sound is somewhat better, all interactivity is lost, as the bells can not be tuned, or struck with different mallets etc. Under Netscape 4, the digital samples are not loaded from the server until the samples are actually used. The very noticeable delay while the audio samples are loaded illustrates the advantage of creating the sounds locally rather than downloading them over the network.

7.3.7 Real-time Model Tester

This demonstration program runs on Windows 95/98 using the DirectSound audio API. It is intended as a test-bed for object models, interaction models, and API design, as well as a demonstration program of the versatility of the audio synthesis methodology. The main control window is depicted in Figure 7.11. The check boxes allow the selection of various operation modes.

In the basic mode, when all boxes are unchecked, except possibly the "3D"

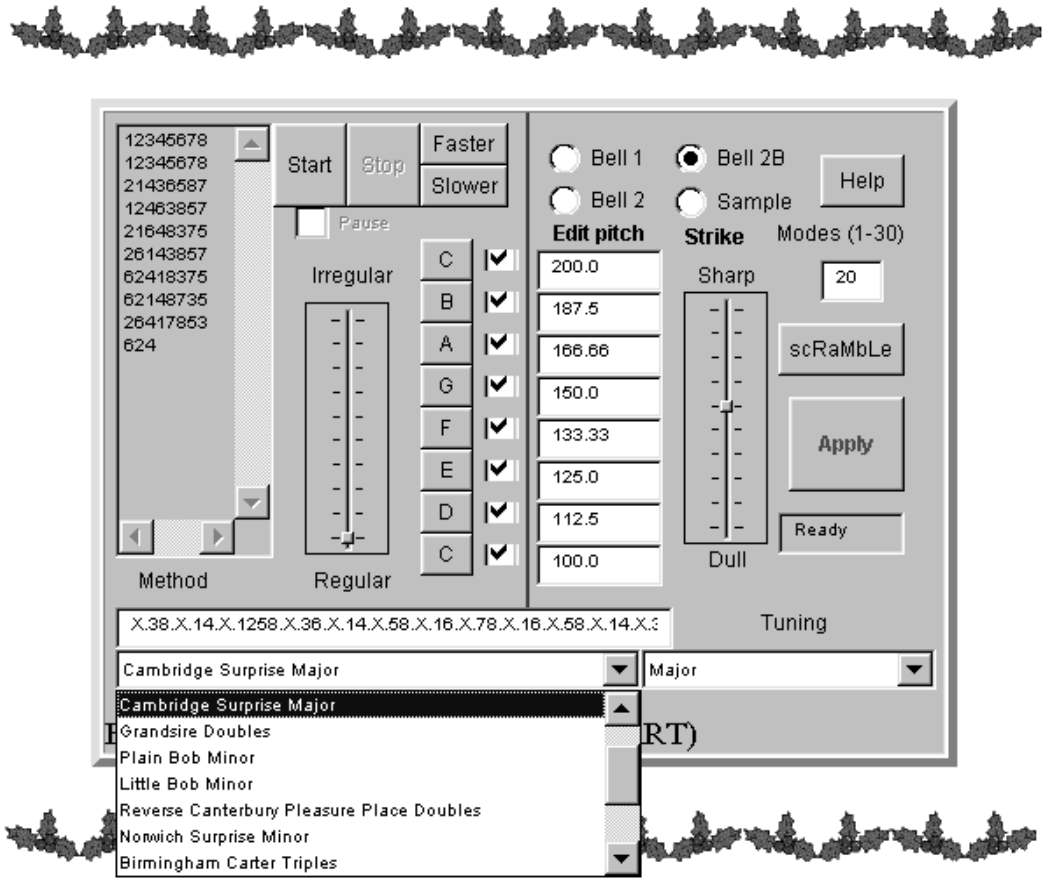

Figure 7.9: Applet for ringing a bell tower.

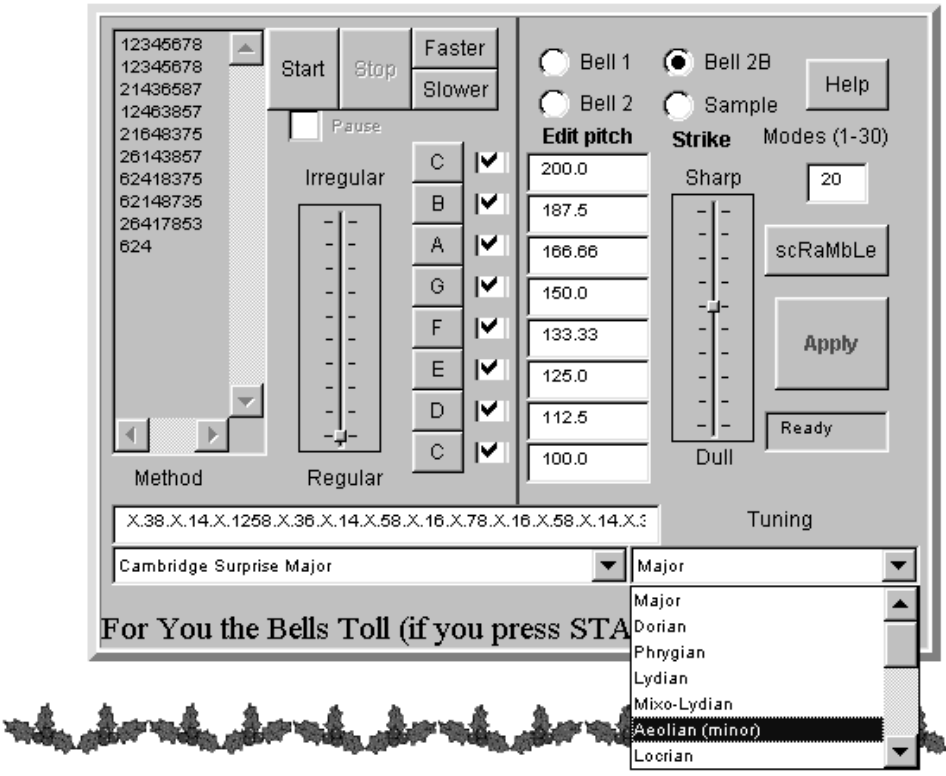

Figure 7.10: Applet for ringing a bell tower.

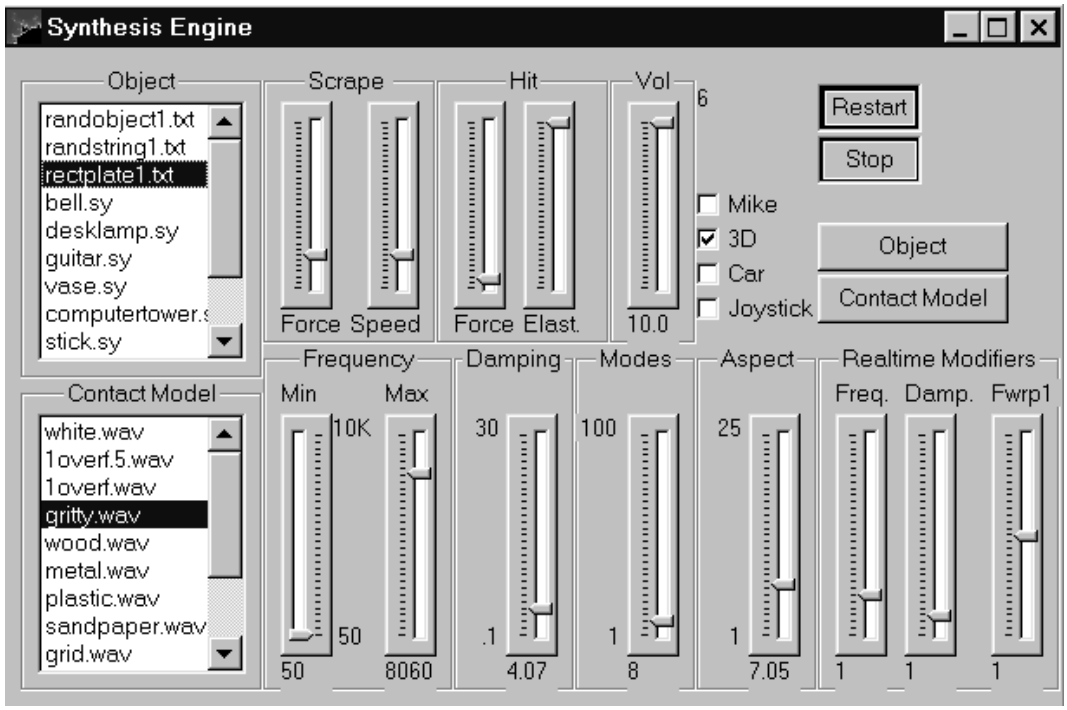

Figure 7.11: Real-time model tester. Main control panel.

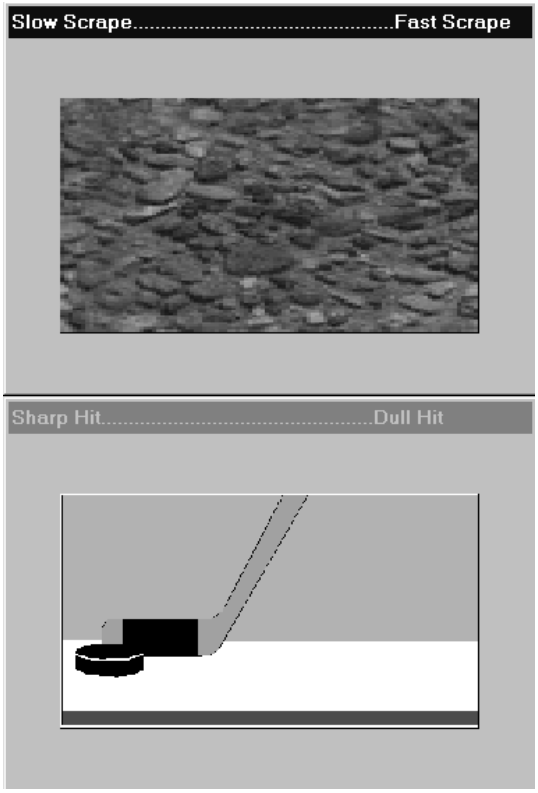

Figure 7.12: Real-time model tester. Interaction windows to scrape and hit objects.

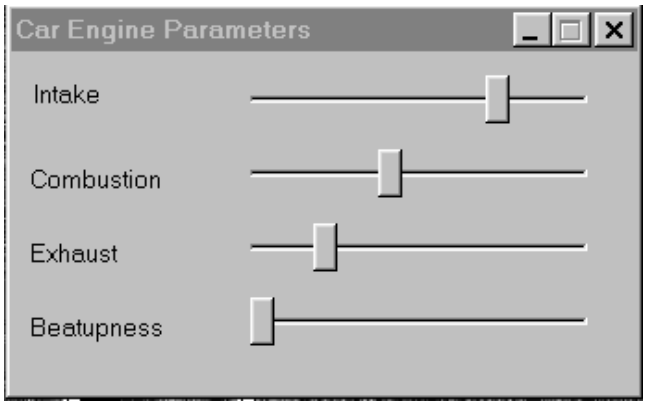

Figure 7.13: Real-time model tester. Engine model editor.

box, the program allows the user to interact with several virtual objects. The object is selected from the "object" list, or can be loaded from disk as an "sy" file (see Appendix A) or as a "txt" file. The sy file is expected to contain a vibration model of an object with one location. If a "txt" file is selected, the object is specified by its shape, base frequency, damping constant, maximum frequency, number of modes, and its aspect ratio (if applicable). We currently have implemented the ideal string, the rectangular plate and a "random" object, which creates randomized resonances and dampings.

An example of a "txt" file specifying a rectangular plate is

```
rectplate
base_frequency:
50.000000
damping:
5.559740
maximum_frequency:
8060.745000
number_of_modes:
13
npoints_x((must_be_1_for_now)or_npoints_in_1_dimension):
1
npoints_y((must_be_1_for_now)unused_in_1_dimension):
1
aspect_ratio_rectangle(unused_for_others):
7.052800
```

After a model is loaded from a "txt" or "sy" file, the user can alter its

properties with the sliders labeled "Frequency", "Damping", "Modes", "Aspect". This requires a restart of the synthesis by pressing the button in the upper right. The three sliders labeled "Realtime Modifiers" can be adjusted during synthesis. They allow the frequencies and the dampings to be scaled uniformly, and the slider labeled "Fwrp1" applies a more complicated transformation to the vibration model. This transformation is intended to demonstrate the capability of real-time "sound morphing" with this synthesis method.

After the vibration model is selected, the user can interact with it in various manners. If the "Mike" option is selected, the audio input channel is interpreted as the applied force, as in the demo described in Section 7.3.3. A problem with DirectSound causes very high latency of up to half a second, making this mode very awkward. Nevertheless we have used it to test vibration models stimulated by real-time interactions.

If the "Mike" option is not checked, the user can stimulate the object in various other manners. If the "Car" and "Joystick" modes are not selected, the user interacts with the object by selecting a contact model from the second list box or from disk. The contact model is stored as a `wav` file and represents the surface texture of the virtual object, according to the model explained in Chapter 6. Mathematical models using scaling noise, as well as recorded dry scraping sounds have been used. Once the object and the contact model are selected, the user can scrape the object by moving the mouse over the upper interaction window depicted in Figure 7.12. The position of the mouse on the interaction rectangle determines the scraping normal force (top to bottom) and the scraping speed (left to right). The range of the force and speed can be adjusted with the "Scrape" sliders. If the "3D" option is selected, the horizontal position of the mouse is mapped to left-right

spatialization of the resulting sound.

The second interaction window depicted in Figure 7.12 allows the user to hit the object by clicking the mouse at various points on the interaction window. The vertical position is mapped to force, while the horizontal position is mapped to the hardness of the virtual hammer, using the contact model for hitting described in Chapter 6.

Sounds of a four stroke combustion engine are obtained by selecting the "Car" option. The interaction with the engine models is similar to the interaction with scraping models, except that one now controls the engine speed through the upper window. For this purpose, several wav files representing models for combustion engines as discussed in Chapter 6 can be selected.

If the mouse pointer leaves the window, the engine sound continues at this rpm. If the "Joystick" option is selected, the user can control the rpm and the spatial position of the sound with the joystick. In this case the idling speed, and the maximum rpm can also be adjusted in real time with the joystick buttons.

In "Car" mode, a third interaction window is present, depicted in Figure 7.13, which allows the user to adjust the volumes of the four stages of the four stroke engine model independently, to create different sounds interactively.

The "Realtime Modifiers" allow the user to change the resonance properties of the engine and exhaust system while running the engine interactively with the joystick. Very convincing sounds of motorcycles, race cars, lawn mowers, and chain saws, can be created using relatively simple vibration and interaction models.

Chapter 8

Conclusions and Extensions

8.1 Summary of Results

In this thesis we have constructed a methodology to synthesize audio in real-time in simulation or game environments. We have shown how the physics of vibrating objects (using linear approximations) leads to a parameterized vibration model suitable for real-time synthesis of interactive audio. The synthesis algorithm maps the eigenvalues of the associated PDE to resonances and maps the mode shapes of physical objects to coupling coefficients. We believe the connection between the physics and the synthesis method is important as it allows a refinement of the method if so desired. In contrast to synthesis methods such as FM, our method provides a clear path to extension and improvement: include more physics.

We have investigated mathematical models of simple geometries and computed the vibration models directly. A one parameter model for the material was found to convey the perception of wood versus metal versus glass quite well.

A parameter fitting algorithm was implemented, which reconstructs a vibra-

tion model from a recording of a struck object. A large number of vibration models of objects have been reconstructed and the method performs satisfactorily.

We have investigated interaction models for collisions, continuous contact, engines, and more abstract forces such as avalanches. It was concluded that a simple one parameter model of impact suffices to convey the perception of the "hardness" of a virtual mallet. For continuous contacts, we conclude that looped wave-table playback of a small sample (representing knowledge about the surface contact) is the best solution. It conveys the perception of "roughness" and of contact speed quite well and it has a solid physical justification to be used as the first order approximation for contact force.

The synthesis algorithm was implemented on several platforms and its performance in terms of speed, latency, and quality was analyzed in some detail, both experimentally and theoretically. It was found that the synthesis is efficient enough to allow a real-time synthesis of reasonably complex objects using just a few percent of CPU cycles on modern personal computers.

We have designed and implemented an API for synthesis around the idea of a SynthesisEngine (which implements the low level synthesis) and a SynthesisController (which implements models of interaction). The API was implemented in C++ and Java and was used to create several demonstration programs, some of which can be found on the accompanying CD or WWW site [4].

From the performance and quality of the sound obtained in the demonstration programs we conclude that this type of audio synthesis can be used to significantly enhance the feeling of immersion in interactive environments. Current wave-table technologies are not able to provide the level of interactivity required in audio for continuous sounds. The synthesis method presented here is the most

efficient and realistic method to obtain contact sounds of interacting objects. The method also produces very good engine and rumbling sounds.

8.2 Extensions and Future Work

Several directions for future work present themselves naturally. Some are obvious and straightforward, some are more challenging.

8.2.1 Faster Synthesis

The performance of the synthesis algorithm can be improved in several ways. A significant performance boost can be expected by implementing the inner loop of the synthesis algorithm in assembly language instead of C. Another possible improvement is to implement the algorithm at multiple resolutions. For a resonance mode of 100 Hz, a sampling rate of 200 Hz should be high enough to accurately compute the contribution of this mode to the total vibration. It is perhaps more efficient to compute each resonance at the minimum sampling rate for that specific resonance and "up-sample" the result when adding all the modes together. It remains to be seen if the overhead of down-sampling the interaction force once for every mode and up-sampling the resulting modal contribution is computationally more efficient than a straightforward computation at a fixed sampling rate.

8.2.2 Integrate Waveguide Models

For linear structures such as strings and air-tubes (exhaust pipes!), waveguide models [67] provide a more efficient method to model resonance properties. The reason is that the resonances in those structures are linearly spaced and are much denser than in solid structures. These systems can be modeled with the modal techniques

pursued in this thesis, but this will be computationally expensive because each resonance takes a fixed percentage of the CPU. Waveguides provide a large set of harmonic resonances for the price of a single mode. Though the amplitudes and dampings can not be individually controlled as in modal synthesis, the performance gain is considerable.

Extending the entire methodology presented here to incorporate waveguide models of vibrating structures is completely straightforward.

8.2.3 Improve Parameter Fitting

How can the linear models be improved? The parameter fitting module produces a set of vibration models which depend on how several parameters such as the window size for the windowed Fourier transform, but there is no systematic or automatic way to chose the "best". As discussed in Section 4.6, this requires a measure on perceptual audio space that would allow us to determine if sound A is a better approximation to sound C than sound B. This appears to be a very difficult problem, which probably requires more understanding of audio perception.

A less ambitious project would be to formally evaluate the quality of the models by psycho-acoustic experiments. It would be interesting to see if a more objective classification of physical objects with respect to their suitability to be modeled with modal vibration models can be obtained.

On the technical level, as discussed in Section 5.2, several improvements and refinements of the parameter fitting algorithm could be attempted.

8.2.4 Improve Interaction Models

The interaction models presented in Chapter 6 are all "first-cut" models and could potentially be improved. The one-parameter impulse model could be refined to include properties relating to the detailed shape of the force profile.

The continuous contact model essentially assumes that the surface profile of the object is followed exactly during contact. In reality there are deformation, hysteresis effects, and contact is frequently broken when one object slides across another. Unfortunately the actual physics of contact is prohibitively complex, and it is not clear how to improve the model in an obvious and simple manner. We suggest investigation of a model where one object frequently "goes ballistic", i.e., at higher contact speeds the objects break contact frequently. Perhaps a simple inelastic collision model would be computationally tractable enough to refine the contact models driving scraping and sliding sounds.

The engine excitation models were constructed in an ad-hoc manner, using simple-minded ideas about explosions and hissing sounds in a four stroke cylinder. Many different types of engines such as diesel engines, two stroke engines, multi-cylinder configurations with accurately modeled firing sequences, could be modeled and investigated.

8.2.5 Non-linear Models

The resonance models are strictly linear. Though non-linear effects can be obtained by coupling vibration models with feedback loops, objects in the real world do not behave in a linear fashion to various degrees. If you hit an object harder and harder, the resulting sounds are not related by just a volume factor. This is partly due to the interaction, which changes non-linearly, and partly due to the actual vibration

equation of the object, which is linear only in first order approximation.

A general theory of non-linear vibration does not exist. In general, non-linear phenomena are extremely complicated and no universal unified method for analysis exists [90]. Some work has been done on non-linear extensions to harmonic oscillators [59], but a physical motivation seems to be lacking.

An interesting observation is that many objects, such as glasses, behave approximately linearly when struck, but then suddenly change their behaviour completely: they break. The interesting thing to note is that after the breaking event we again have an approximately linear system of glass fragments that fall within the scope of the present method. After the breaking event we will have to model many objects and their collisions in order to correctly synthesize the sound of the glass fragments falling. Presumably, some stochastic method of generating the impulsive forces and the distribution of the sizes of the fragments could drive the synthesis. If we consider the breaking characteristic of the glass as part of the glass model, we can view this as a piecewise linear simulation. Before the glass breaks we use a linear model and after the glass break we have a (different) linear model. See also [30, 17, 86] for other work on this topic.

This type of piecewise linear behaviour occurs very frequently in reality. For example, an old rattly car consists of many parts which collide against each other as a result of the oscillations of the different parts. This line of reasoning would lead to a generalization of the modal audio synthesis model where different components are allowed to collide and impulsive forces are generated internally by collisions.

A simple example of a non-linear model of this kind is a constrained pendulum, depicted in Figure 8.1. The pendulum behaves approximately linearly as long as the amplitude of the oscillation is small, but as soon as the pendulum starts

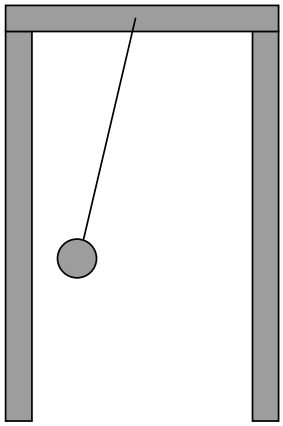

Figure 8.1: Pendulum is non-linear when colliding with the walls.

colliding with the walls, it is only piecewise linear. Such a constrained pendulum could perhaps be used as a generalization of the spring-damper systems on which the audio synthesis described in this thesis is built.

Bibliography

[1] http://ccrma-www.stanford.edu.

[2] http://mediatheque.ircam.fr/articles/index-e.html.

[3] http://www.cnmat.berkeley.edu.

[4] http://www.cs.ubc.ca/nest/lci/thesis/kvdoel.

[5] http://www.engr.wisc.edu/epd/steuber/motorcycle.html.

[6] http://www.ircam.fr.

[7] http://www.neuroinformatik.ruhr-uni-bochum.de/ini/people/heja/sy-prog/node1.html.

[8] *MATLAB Reference Guide*. The MathWorks, Inc., 1992.

[9] D. Baraff. Dynamic simulation of non-penetrating flexible bodies. *Computer Graphics*, 26(2):303–308, 1992.

[10] D. Baraff. *Dynamic Simulation of Non-Penetrating Rigid Bodies*. PhD thesis, Cornell University, 1992.

[11] Durand R. Begault. *3-D Sound for Virtual Reality and Multimedia*. Academic Press, London, 1994.

[12] R. T. Beyer. *Nonlinear Acoustics*. Department of the Navy, Sea Systems Command, Washington, D. C., 1974.

[13] W. G. Bickley and A. Talbot. *An Introduction to the Theory of Vibrating Systems*. Oxford University Press, London, 1961.

[14] S. Braun and Y. M. Ram. Determination of structural modes via the prony model: System order and noise induced poles. *J. Acoust. Soc. Am.*, 81(5):1447–1459, 1987.

[15] Albert S. Bregman. *Auditory Scene Analysis.* MIT Press, London, 1990.

[16] G. F. Carrier. On the non-linear vibration problem of the elastic string. *Q. Appl. Math*, 3:157–165, 1945.

[17] Michael Anthony Casey. *Auditory Group Theory with Applications to Statistical Basis Methods for Structured Audio.* PhD thesis, Massachusetts Institute of Technology, 1998.

[18] Antoine Chaigne and Vincent Doutaut. Numerical simulations of xylophones. i. time domain modeling of the vibrating bars. *J. Acoust. Soc. Am.*, 101(1):539–557, 1997.

[19] A. W. T. Cleaver. *The theory of change ringing: an introduction.* J. Hannon and Co., Oxford, 1976.

[20] Perry R. Cook. Physically informed sonic modeling (phism): Percussive synthesis. In *Proceedings of the International Computer Music Conference*, pages 228–231, Hong Kong, 1996.

[21] Perry Raymond Cook. *Identification of Control Parameters in an Articulatory Vocal Tract Model, with Applications to the Synthesis of Singing.* PhD thesis, Stanford University, 1991.

[22] J. Cremer and A. J. Stewart. The architecture of newton, a general-purpose dynamics simulator. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1806 – 1811, 1989.

[23] Baron Gaspard Riche de Prony. Essai éxperimental et analytique: sur les lois de la dilatabilité de fluides élastique et sur celles de la force expansive de la vapeur de l'alkool, à différentes températures. *Journal de l'École Polytechnique*, 1(22):24–76, 1795.

[24] P. Depalle and X. Rodet. Synthèse additive par FFT inverse. Technical report, IRCAM, Paris, 1990.

[25] James F. Doyle. *An FFT-Based Spectral Analysis Methodology.* Springer-Verlag, New York, 1989.

[26] N. I. Durlach and A. S. Mavor, editors. *Virtual Reality, Scientific and technological challenges.* National Academy Press, Washington, D. C., 1995.

[27] Frank Fahy. *Sound and Structural Vibration. Radiation, Transmission and Response.* Academic Press, London, 1985.

[28] L. Fox, P. Henrici, and C. Moler. Approximations and bounds for eigenvalues of elliptical operators. *SIAM J. Num. Analy.*, 4:89–102, 1967.

[29] Daniel J. Fried. Auditory correates of perceived mallet hardness for a set of recorded percussive sound events. *J. Acoust. Soc. Am.*, 87(1):311–321, 1990.

[30] W. W. Gaver. What in the world do we hear?: An ecological approach to auditory event perception. *Ecological Psychology*, 5(1):1–29, 1993.

[31] James K. Hahn, Joe Geigel, Jong Won Lee, Larry Gritz, Tapio Takala, and Suneil Mishra. An integrated approach to motion and sound. *Journal of Visualization and Computer Animation*, 6(2):109–123, 1992.

[32] Donald E. Hall. Piano string excitation in the case of small hammer mass. *J. Acoust. Soc. Am.*, 9(1):141–147, 1986.

[33] Donald E. Hall. Piano string excitation II: General solution for a hard narrow hammer. *J. Acoust. Soc. Am.*, 81(2):535–545, 1987.

[34] Donald E. Hall. Piano string excitation III: General solution for a soft narrow hammer. *J. Acoust. Soc. Am.*, 81(2):547–555, 1987.

[35] H. L. F. Helmholtz. *On the Sensations of Tone.* Dover, New York, 1954.

[36] David Jaffe. Ten criteria for evaluating synthesis and processing techniques. *Computer Music Journal*, 19(1):76–87, 1995.

[37] David Javelosa. *Sound and Music for Multimedia.* Henry Holt & Company Inc., New York, 1997.

[38] Claes Johnson. *Numerical solutions of partial differential equations by the finite element method.* Cambridge University Press, Cambridge, 1987.

[39] K. L. Johnson. *Contact Mechanics.* Cambridge University Press, Cambridge, 1985.

[40] Ron Johnston and Allsopp Graham. *An atlas of Bells.* Blackwell Reference, Cambridge, Mass., 1990.

[41] Matti Karjalainen and Julius Smith. Body modeling techniques for string instrument synthesis. In *Proceedings of the International Computer Music Conference*, pages 232–239, Hong Kong, 1996.

[42] J. B. Keller. Impact with friction. *Journal of Applied Mechanics*, 53(1):1–4, 1986.

[43] Eric Krotkov and Roberta Klatzky. Robotic perception of material: Experiments with shape-invariant acoustic measures of material type. In *Preprints of the Fourth International Symposium on Experimental Robotics, ISER '95*, Stanford, California, 1995.

[44] Debassis Kundu. Estimating the parameters of undamped exponential signals. *Technometrics*, 35(2):215–218, 1993.

[45] Horace Lamb. *The Dynamical Theory of Sound.* Edward Arnol, London, 1910.

[46] Strutt Lord Rayleigh. *The Theory of Sound.* Dover, New York, 1896.

[47] Per Løtstedt. Numerical simulation of time-dependent contact and friction problems in rigid body mechanics. *SIAM J. Sci. Stat. Comput.*, 5(2):370–393, 1984.

[48] J. D. Markel and A. H. Gray. *Linear Preciction of Speech.* Springer Verlag, New York, 1976.

[49] Robert J. McAulay and Thomas F. Quatieri. Speech analysis/synthesis based on a sinusoidal representation. *IEEE Trans. Acous., Speech, Signal Processing*, ASSP-34(4):744–754, 1986.

[50] John C. Middlebrooks and David M. Green. Sound localization by human listeners. *Annu. Rev. Psychol.*, 42:135–159, 1991.

[51] Brian C. J. Moore. *An Introduction to the Psychology of Hearing.* Academic Press, London, 1986.

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

[53] P. M. Morse and K. U. Ingard. *Theorectical Acoustics.* Princeton University Press, Princeton, NJ, 1986.

[54] Philip Morse. *Vibration and Sound.* American Institute of Physics for the Acoustical Society of America, fourth edition, 1976.

[55] D. E. Newland. *Mechanical Vibration Analysis and Computation.* Longman Scientific and Technical, New York, 1989.

[56] M. R. Osborne and G. K. Smyth. A modified prony algorithm for fitting functions defined by difference equations. *SIAM J. Sci. Stat. Comput.*, 12(2):362–382, 1991.

[57] T. W. Parks and C. S. Burrus. *Digital Filter Design*. John Wiley and Sons, Inc, New York, 1987.

[58] Maurice Petyt. *Introduction to Finite Element Vibration Analysis*. Cambridge University Press, Cambridge, 1990.

[59] John R. Pierce and Scott A. van Duyne. A passive nonlinear digital filter design which facilitates physics-based sound synthesis of highly nonlinear musical instruments. *J. Acoust. Soc. Am.*, 101(2):1120–1122, 1997.

[60] L. R. Rabiner, M. J. Cheng, A. E. Rosenberg, and C. A. McGonegal. A comparative performance study of several pitch detection algorithms. *IEEE Trans. on Acoustics, Speech and Signal Processing*, 24(5):399–418, 1976.

[61] Clifford T. Mullis Richard A. Roberts. *Digital Signal Processing*. Addison-Wesley, Reading, Massachusetts, 1987.

[62] M. J. Ross, H. L. Schaffer, A. Cohen, R. Freudberg, and H. J. Manley. Average maginitude difference function pitch extractor. *IEEE Trans. on Acoustics, Speech and Signal Processing*, 22(5):353–362, 1974.

[63] Thomas D. Rossing, D. Scott Hampton, and Uwe J. Hansen. Music from oil drums: the acoustics of the steel pan. *Physics Today*, March:24–29, 1996.

[64] William R. Savage, Edward L. Kottick, Thomas J. Hendrickson, and Kenneth D. Marshall. Air and structural modes of the harpsichord. *J. Acoust. Soc. Am.*, 91(4):2180–2189, 1992.

[65] A. A. Shabana. *Theory of Vibration, Volume I: An Introduction*. Springer-Verlag, London, 1991.

[66] A. A. Shabana. *Theory of Vibration, Volume II: Discrete and Continuous Systems*. Springer-Verlag, London, 1991.

[67] J. O. Smith. Physical modeling using digital waveguides. *Computer Music Journal*, 16(4):75–87, 1992.

[68] M. M. Sondhi. New methods of pitch extraction. *IEEE Trans. on Audio and Electro Acoustics*, 16(2):262–266, 1968.

[69] William M. Steedly, Chinghui J. Ying, and Randolph L. Moses. Statistical analysis of tls-based prony techniques. *Automatica, Special Issue on Statistical Processing and Control*, 1994.

[70] William M. Steedly, Chinghui J. Ying, and Randolph L. Moses. A modified tls-prony method using data decomation. *IEEE Transactions on Signal Processing*, 42(9):2292–2303, 1995.

[71] Ken Steiglitz. *Classic maximum entropy. In: Maximum Entropy and Bayesian Methods*. Kluwer Academic, Norwell, MA, 1989.

[72] Ken Steiglitz. *A Digital Signal Processing Primer with Applications to Digital Audio and Computer Music*. Addison-Wesley, New York, 1996.

[73] R. W. B. Stephens. *Acoustics and Vibrational Physics*. Edward Arnold (Publishers) Ltd., London, 1966.

[74] Adam Stettner and Donald P. Greenberg. Computer graphics visualization for acoustic simulation. *Proc. SIGGRAPH'89, Computer Graphics*, 23(3):195–206, 1989.

[75] Gilbert Strang. *Introduction to Applied mathematics*. Wellesley-Cambridge Press, 1986.

[76] Anatoli Stulov. Hysteretic model of the grand piano hammer felt. *J. Acoust. Soc. Am.*, 97(4):2577–2585, 1995.

[77] Donald L. Sullivan. accurate frequency tracking of timpani spectral lines. *J. Acoust. Soc. Am.*, 101(1):530–538, 1997.

[78] Frank W. Sinden Suresh Goyal, Elliot N. Pinson. Simulation of Dynamics of Interacting Rigid Bodies Including Friction I: General Problem and Contact Model. *Engineering with Computers*, 10:162–174, 1994.

[79] Frank W. Sinden Suresh Goyal, Elliot N. Pinson. Simulation of Dynamics of Interacting Rigid Bodies Including Friction II: Software System Design and Implementation. *Engineering with Computers*, 10:175–195, 1994.

[80] Tapio Takala and James Hahn. Sound rendering. *Proc. SIGGRAPH'92, ACM Computer Graphics*, 26(2):211–220, 1992.

[81] Samuel Temkin. *Elements of Acoustics*. John Wiley and Sons, Inc., New York, 1981.

[82] Barry Truax. Real-time granular synthesis with a digital signal processor. *Computer Music Journal*, 12(2):14–26, 1988.

[83] C. Ullrich and Dinesh K. Pai. Contact response maps for real time dynamic simulation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1019 – 1025, Leuven, May 1998.

[84] Kees van den Doel and Dinesh K. Pai. The sounds of physical shapes. *Presence*, 7(4):382–395, 1998.

[85] Richard F. Voss and John Clarke. 1/f noise in music: Music from 1/f noise. *J. Acoust. Soc. Am.*, 63(1):258–263, 1978.

[86] R. M. Warren and R. R. Verbrugge. Auditory perception of breaking and bouncing events: A case study in ecological acoustics. *Journal of Experimental Psychology: Human Perception and Performance*, 10:704–712, 1984.

[87] C. A. Wert. Internal friction in solids. *Journal of Applied Physics*, 60(6):1888–1895, 1986.

[88] Bruce J. West and Michael Shlesinger. On the ubiquity of 1/f noise. *Inernational Journal of Modern Physics B*, 3(6):795–819, 1989.

[89] Bruce J. West and Michael Shlesinger. The noise in natural phenomena. *American Scientist*, 78:40–45, 1990.

[90] G. B. Whitham. *Linear and Nonlinear Waves*. Wiley, New York, 1974.

[91] Richard P. Wildes and Whitman A. Richards. Recovering material properties from sound. In Whitman Richards, editor, *Natural Computation*, Cambridge, Massachusetts, 1988. The MIT Press.

[92] Wilfred G. Wilson. *Change ringing : the art and science of change ringing on church and hand bells*. Faber and Faber, London, 1965.

[93] Richard Woodbridge. Acoustic Recordings from Antiquity. In *Proceedings of the I.E.E.E.*, pages 1465–1466, 1965.

[94] P. M. Zurek. The precedence effect and its possible role in the avoidance of interaural ambiguities. *J. Acoust. Soc. Am.*, 67(3):952–964, 1980.

Appendix A

File Format for Vibration Models

We give the specification of the "sy" file format used by our applications to define a vibration model of an object.

The lines with text are comments, indicating the meaning of the following data. The **nactive_freq:** field describes how many modes should be used by the synthesis software to render the sound. It should be less than or equal to **n_freq:** which is the number of modes stored. The **n_points:** field indicates how many discrete "locations" are available for the amplitudes a. These "locations" may correspond to actual geometric locations on the contact surface of the objects, or to directional parameters. The fields **frequency_scale:**, **damping_scale:**, and **amplitude_scale:** multiply the following frequency, damping, and amplitude parameters by a constant scale factor. These fields allow a quick manual edit by shifting all frequencies, increasing the damping, or boosting the coupling strength of the model. After the **frequencies:** header field, the frequencies are listed in Hertz, after the

damping: header the damping coefficients (the factors d in the impulse response $ae^{(-dt+2\pi ift)}$ of an individual mode), and after the **amplitudes[point][freq]:** the amplitudes; first the amplitudes of the first "location", then the second location, etc. The end of file is indicated by the string **END**. Below we give a short example of an sy-file for an ideal string with coupling data for three locations.

```
nactive_freq:
10
n_freq:
10
n_points:
3
frequency_scale:
1.000000
damping_scale:
1.000000
amplitude_scale:
1.000000
frequencies:
440.000000
880.000000
1320.000000
1760.000000
2200.000000
2640.000000
3080.000000
3520.000000
3960.000000
```

```
4400.000000
dampings:
1.000000
2.000000
3.000000
4.000000
5.000000
6.000000
7.000000
8.000000
9.000000
10.000000
amplitudes[point][freq]:
0.258819
0.250000
0.235702
0.216506
0.193185
0.166667
0.137989
0.108253
0.078567
0.050000
0.707107
0.500000
0.235702
0.000000
0.141421
```

```
0.166667
0.101015
0.000000
0.078567
0.100000
0.965926
0.250000
0.235702
0.216506
0.051764
0.166667
0.036974
0.108253
0.078567
0.050000
END
```

Appendix B

Parameter Fitting Data

In this section we present the results of the models obtained by parameter fitting for several objects. The corresponding sounds can be heard on the accompanying CD or on [4].

Each figure shows the linear fits overtime to the identified frequency peaks (in dB), the estimated frequencies in Hz, and the frequency response of the resulting model on two different scales. The sampling rate was 44100 Hz and we show the results of various size Fourier windows. The material constant plots show the ratio of frequency and damping for the first 100 modes (sorted by amplitude), which should be constant according to the simplest material model.

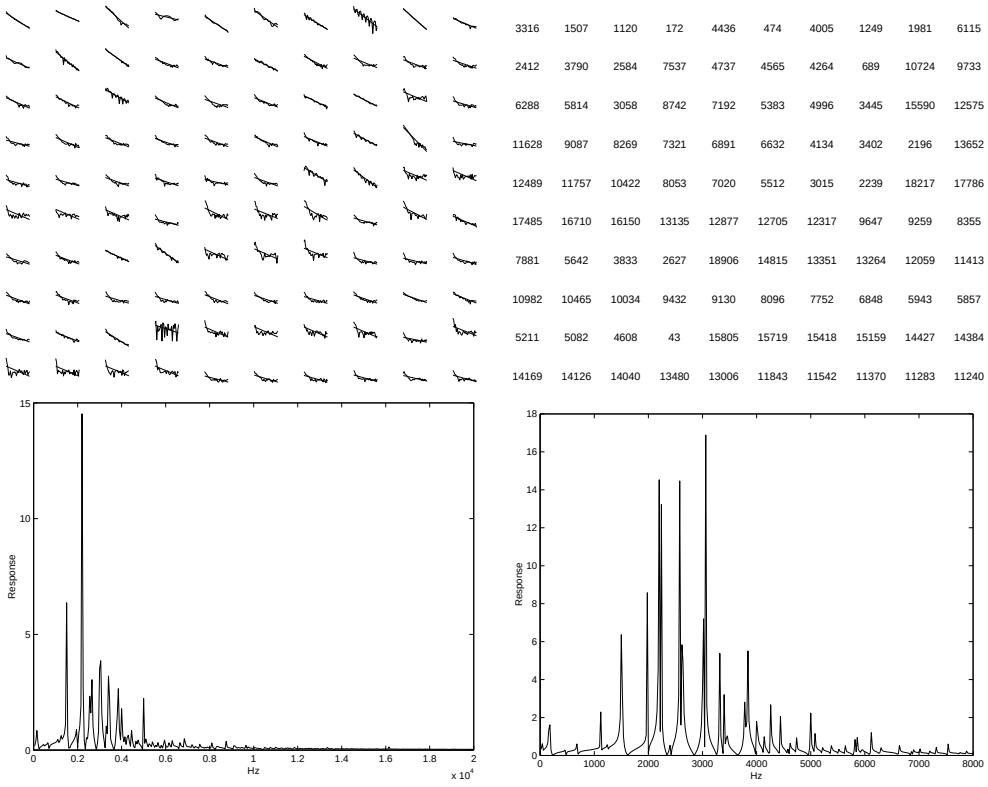

Figure B.1: Top of vase with a window of 1024

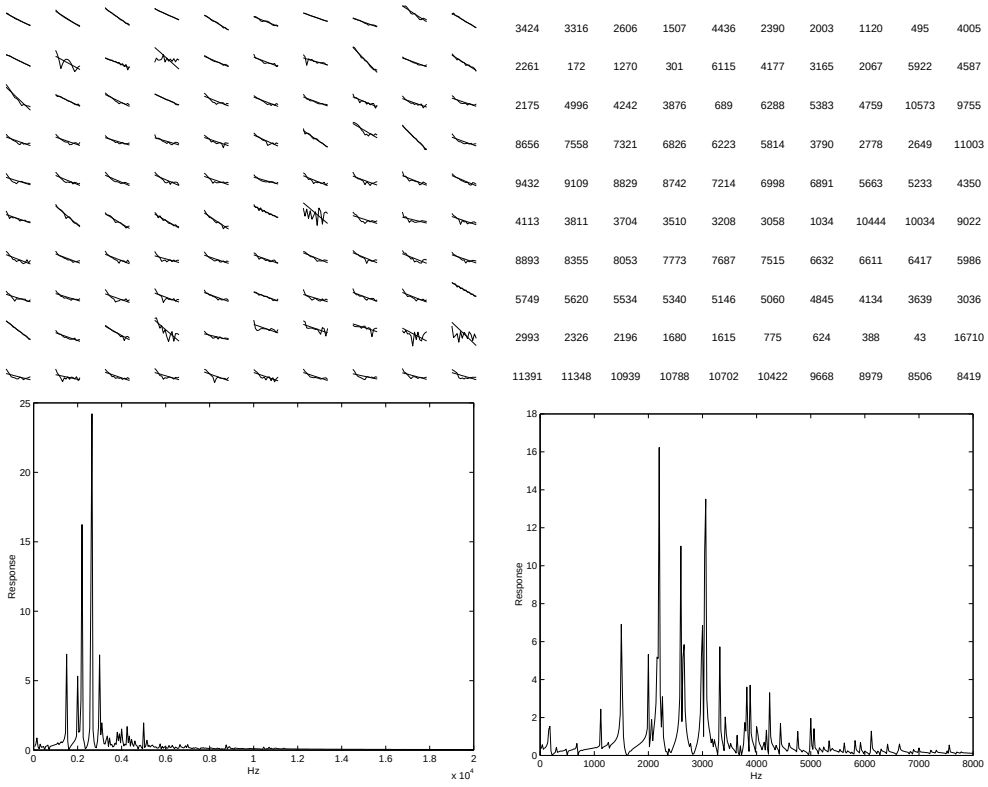

Figure B.2: Top of vase with a window of 2048

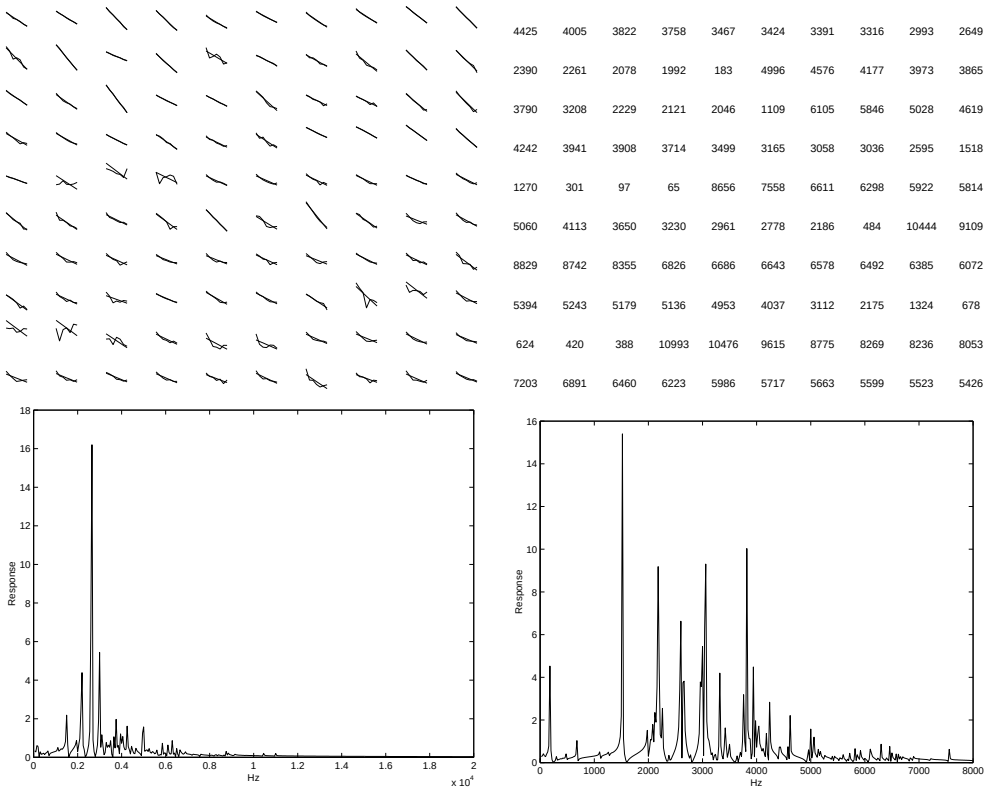

Figure B.3: Top of vase with a window of 4096

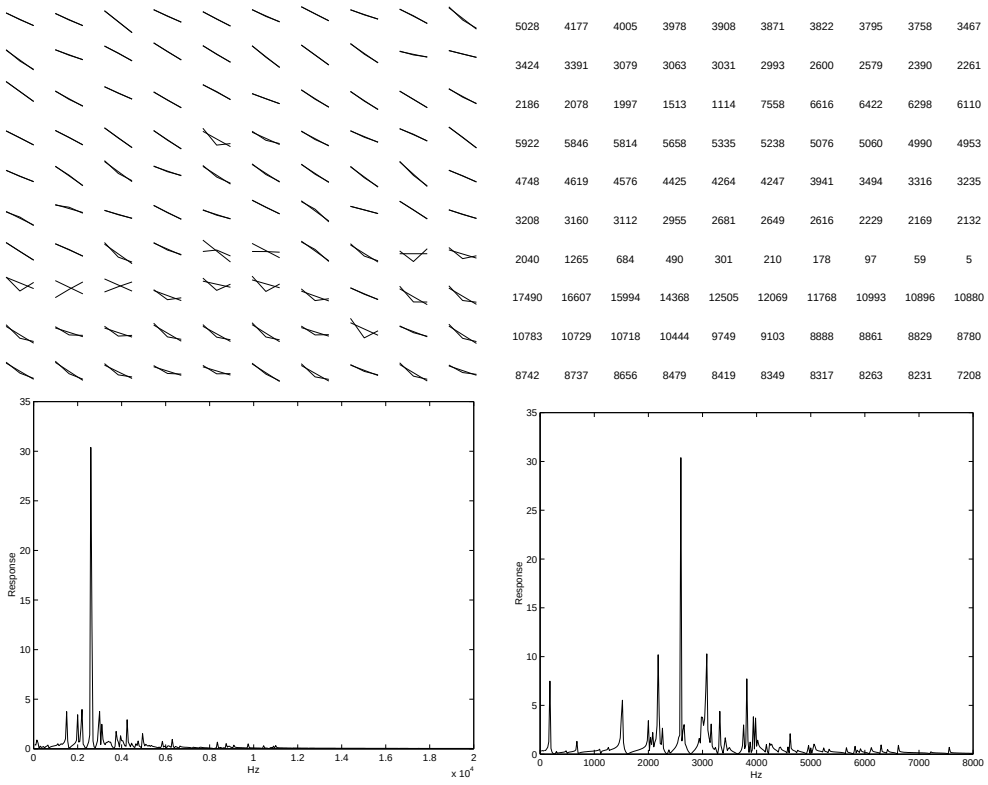

Figure B.4: Top of vase with a window of 8192

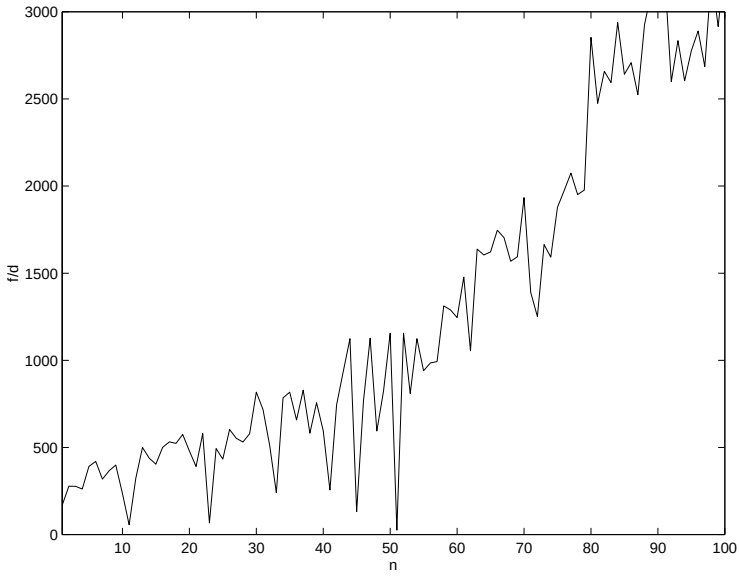

Figure B.5: Top of vase with a window of 1024: Material constant

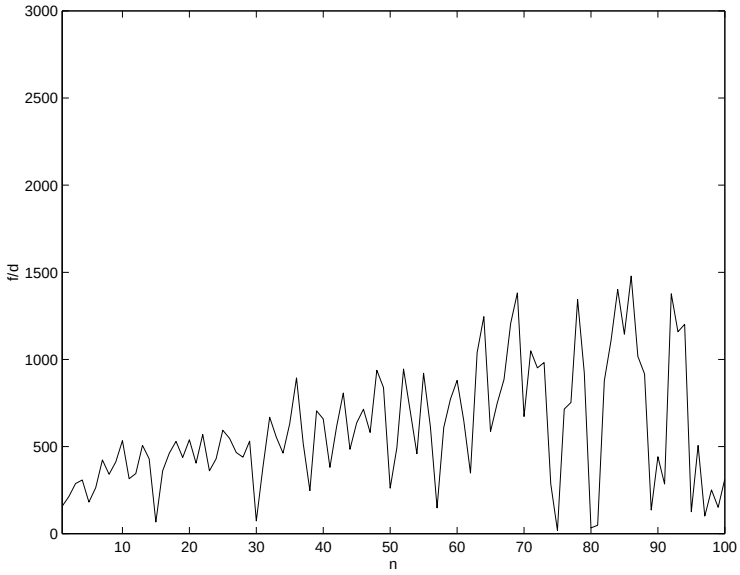

Figure B.6: Top of vase with a window of 2048: Material constant

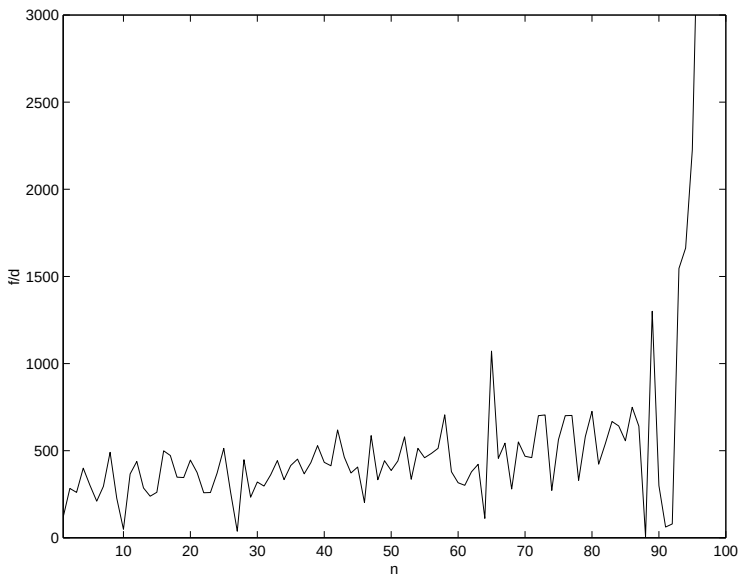

Figure B.7: Top of vase with a window of 8192: Material constant

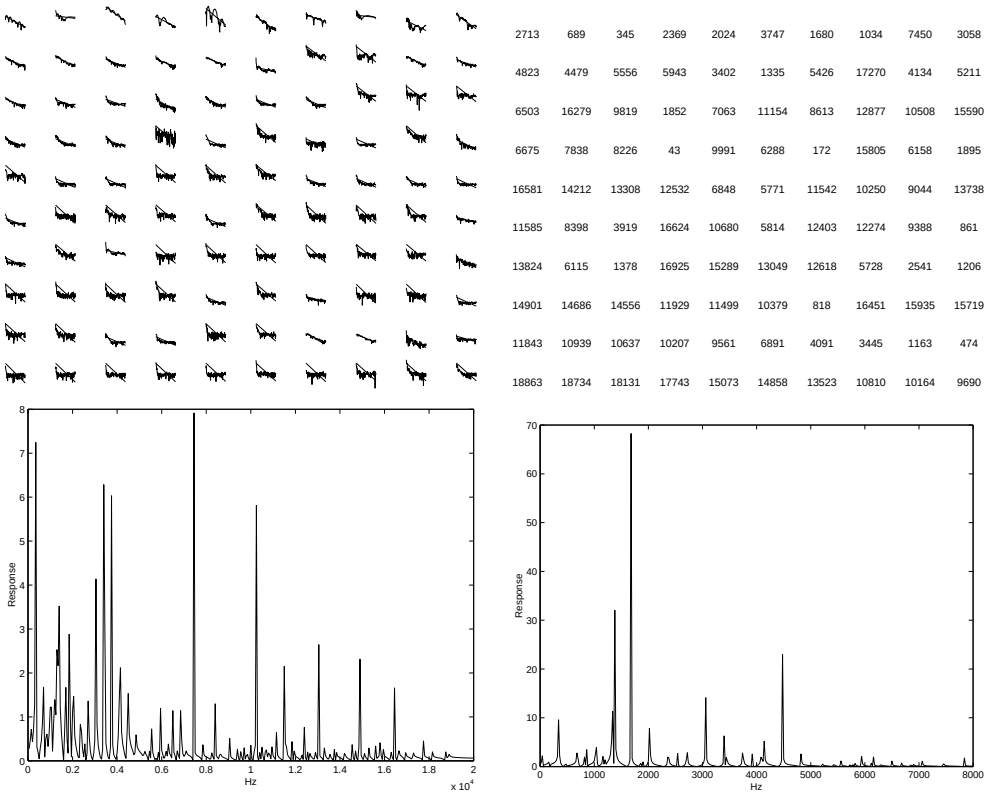

Figure B.8: Santur with a window of 1024

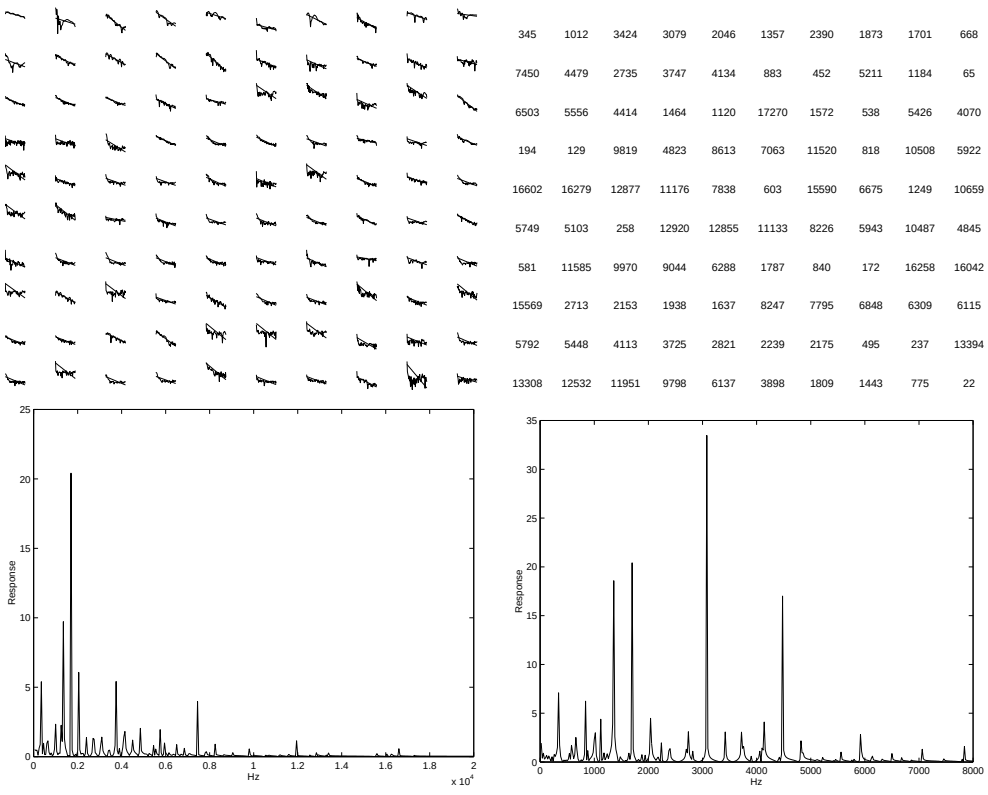

Figure B.9: Santur with a window of 2048

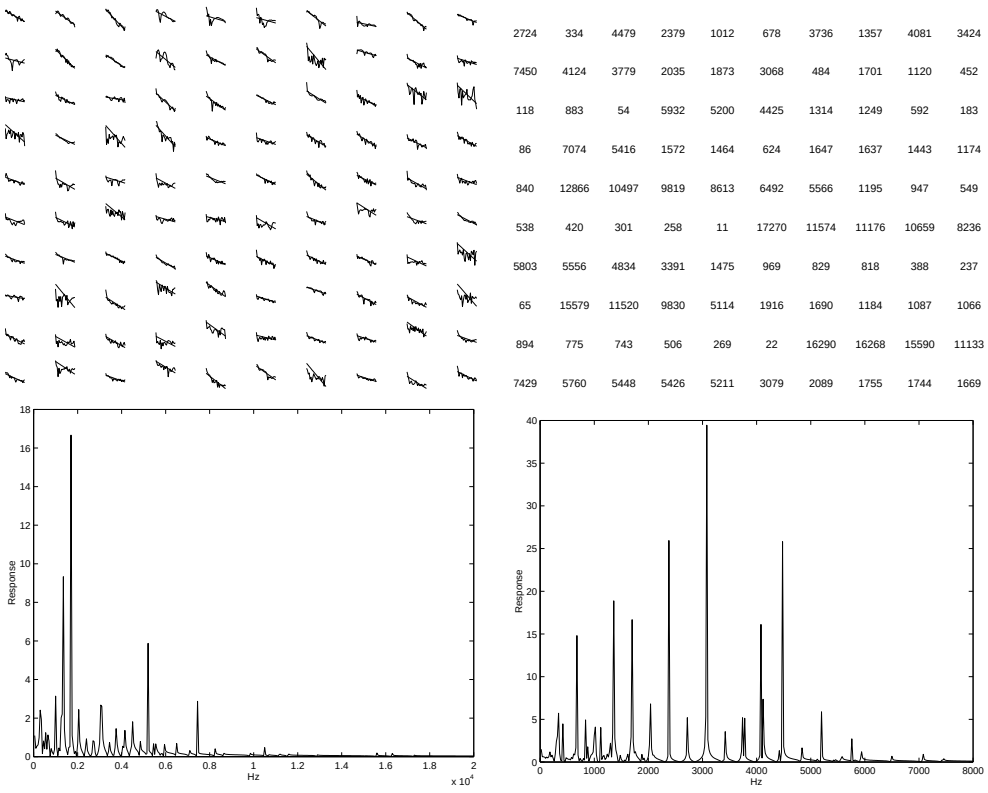

Figure B.10: Santur with a window of 4096

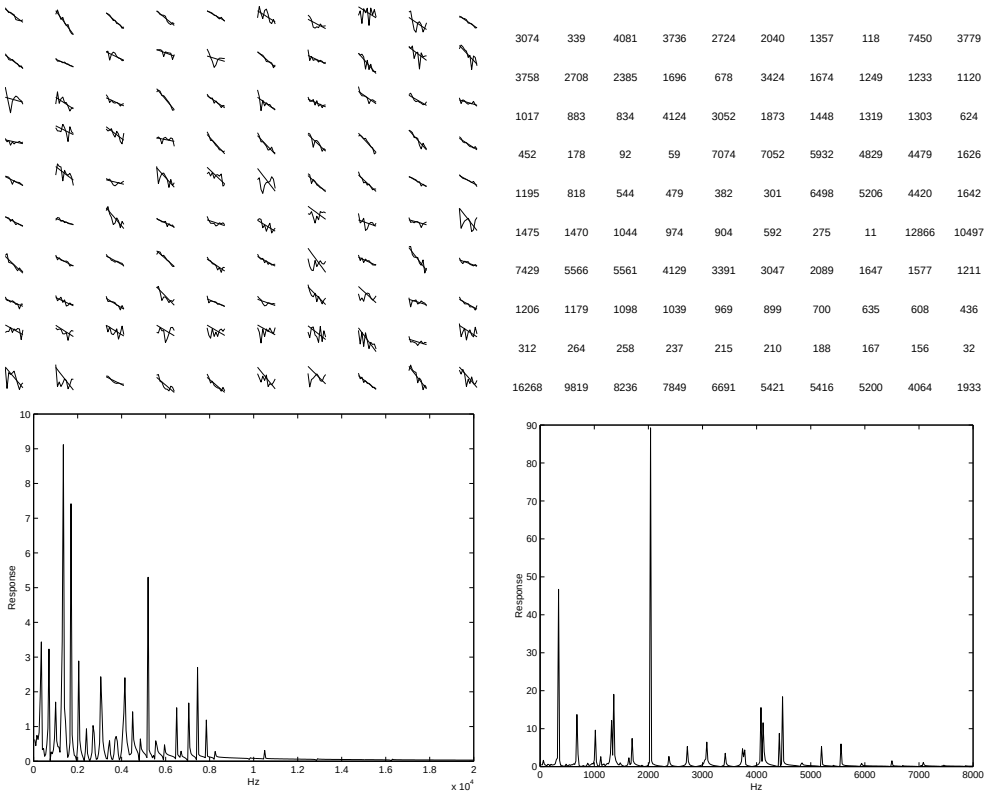

Figure B.11: Santur with a window of 8192

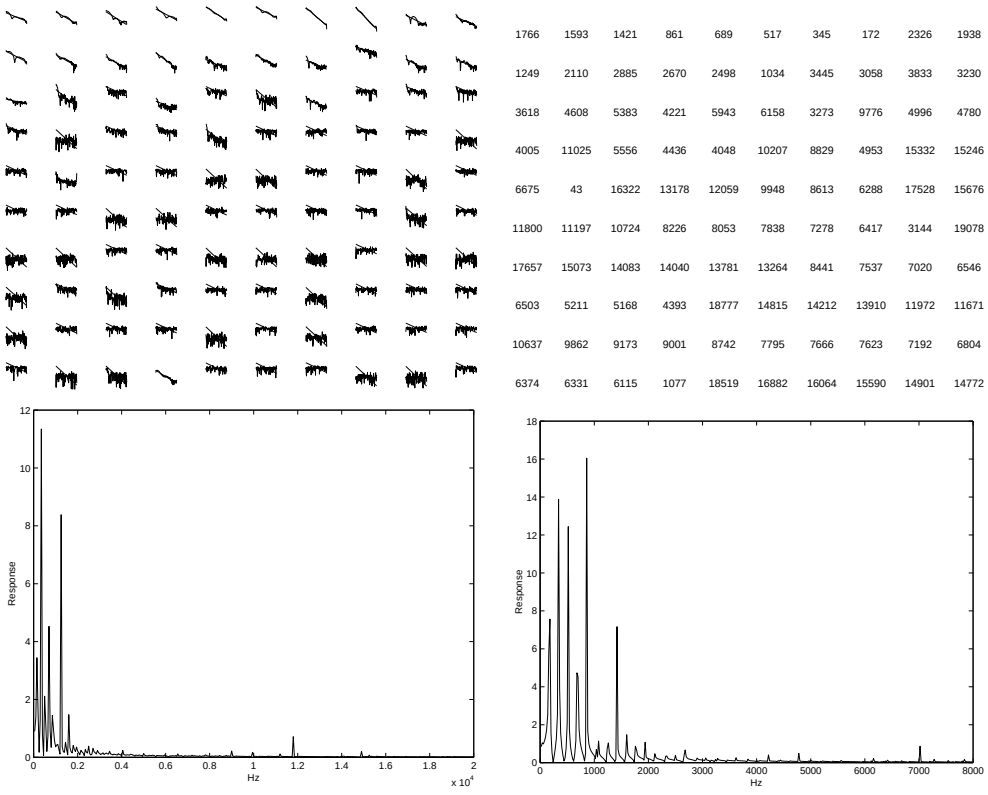

Figure B.12: Piano with a window of 1024

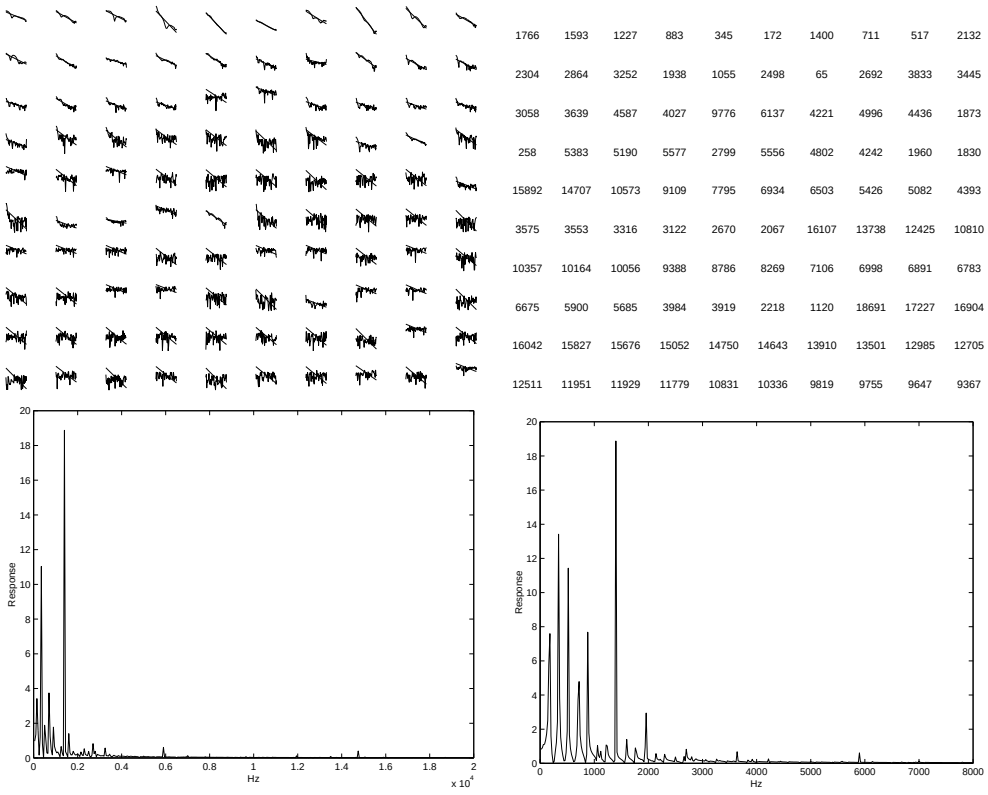

Figure B.13: Piano with a window of 2048

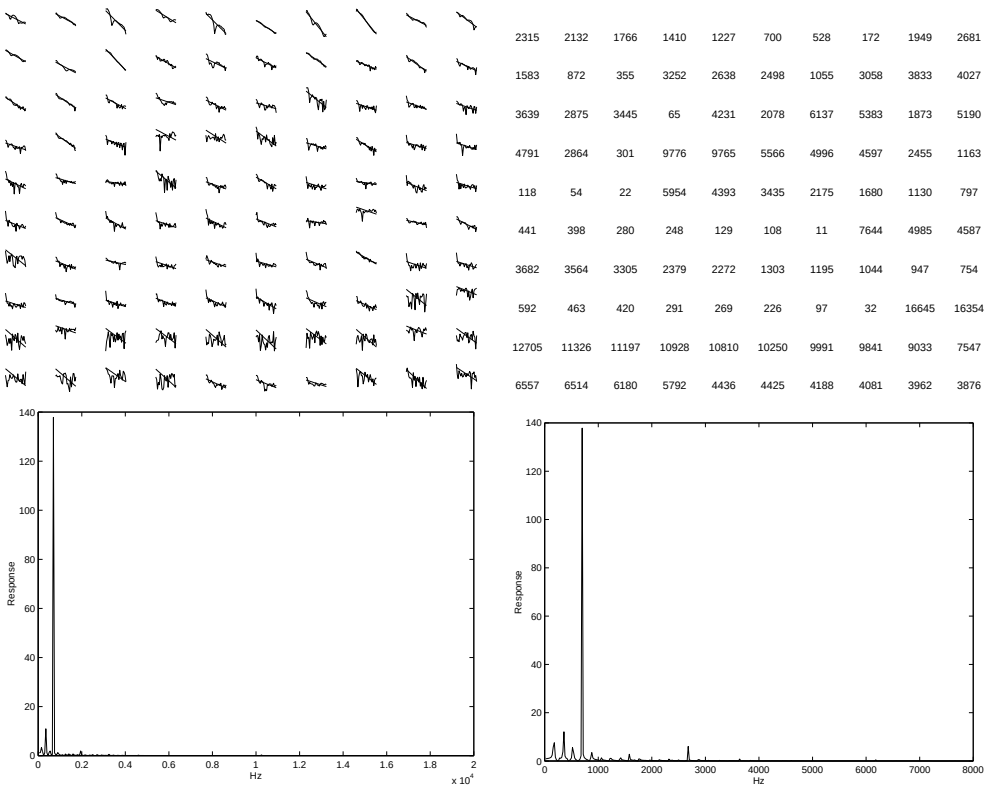

Figure B.14: Piano with a window of 4096

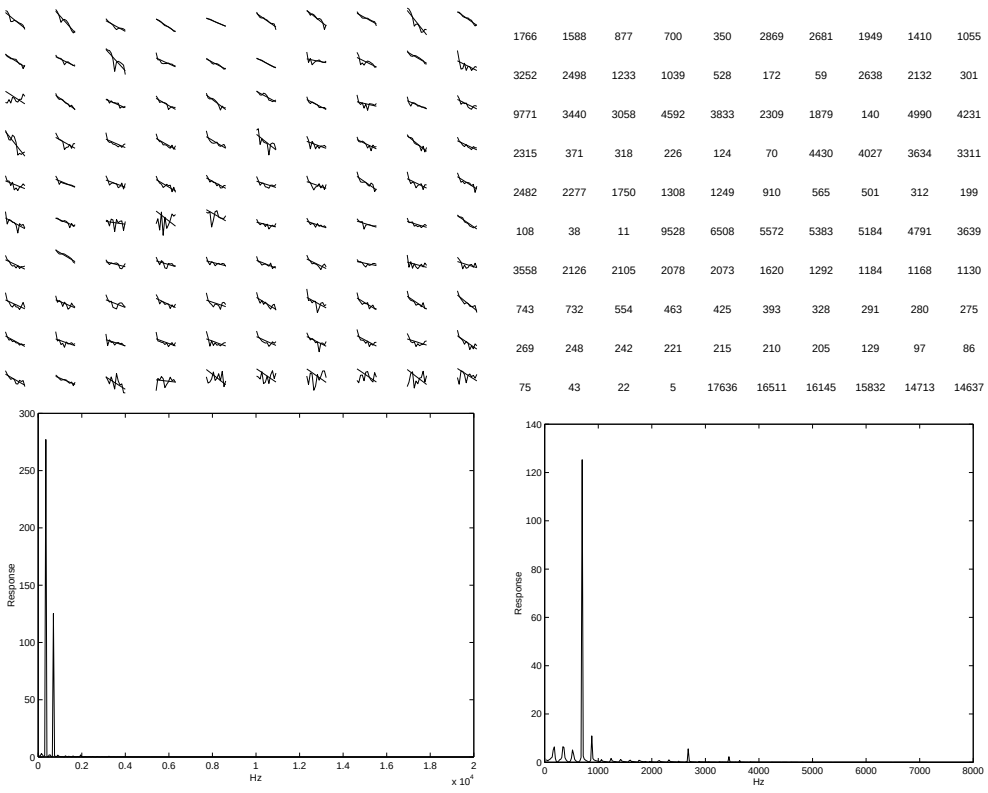

Figure B.15: Piano with a window of 8192

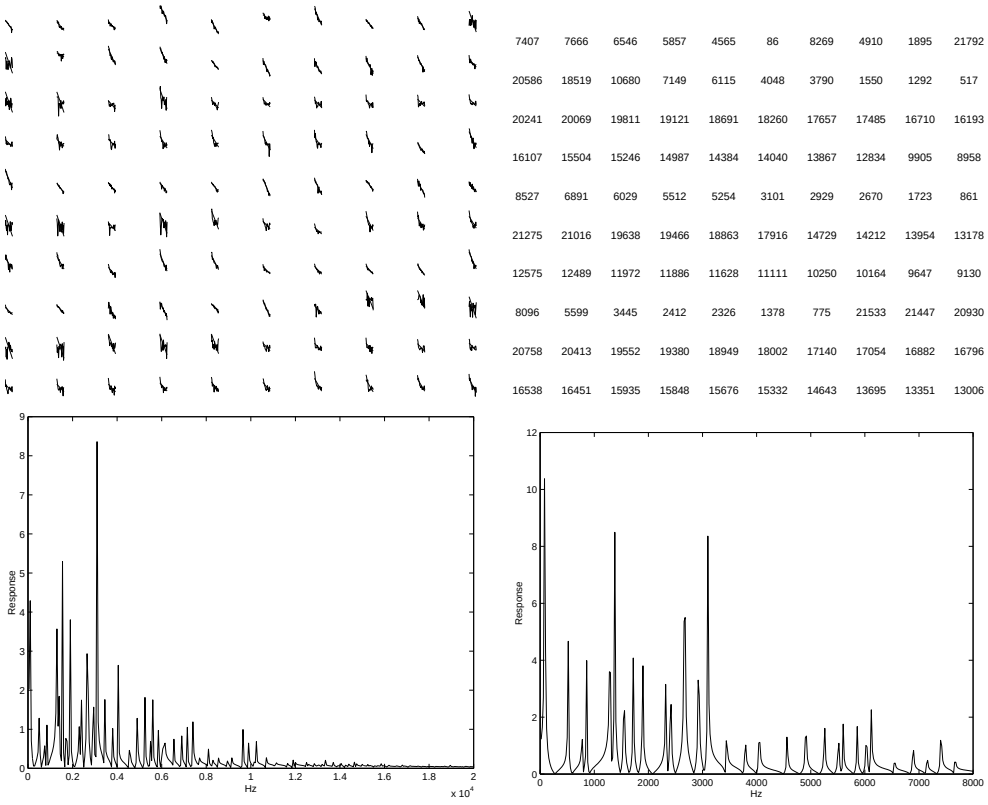

Figure B.16: Computer tower with a window of 512

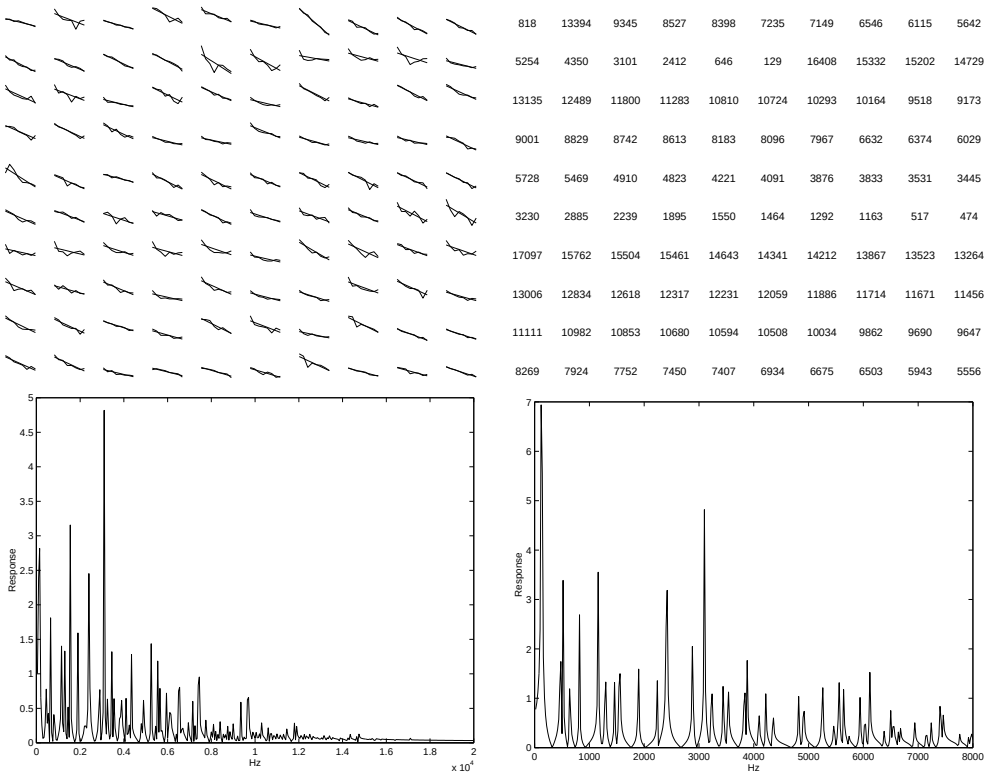

Figure B.17: Computer tower with a window of 1024

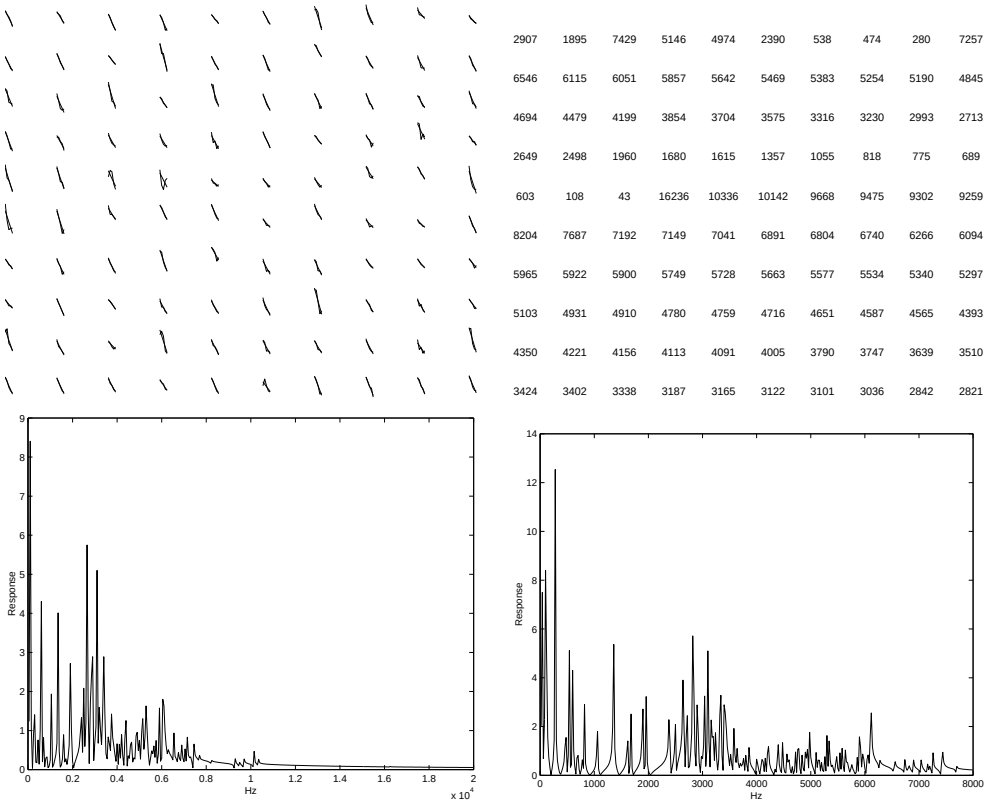

Figure B.18: Computer tower with a window of 2048

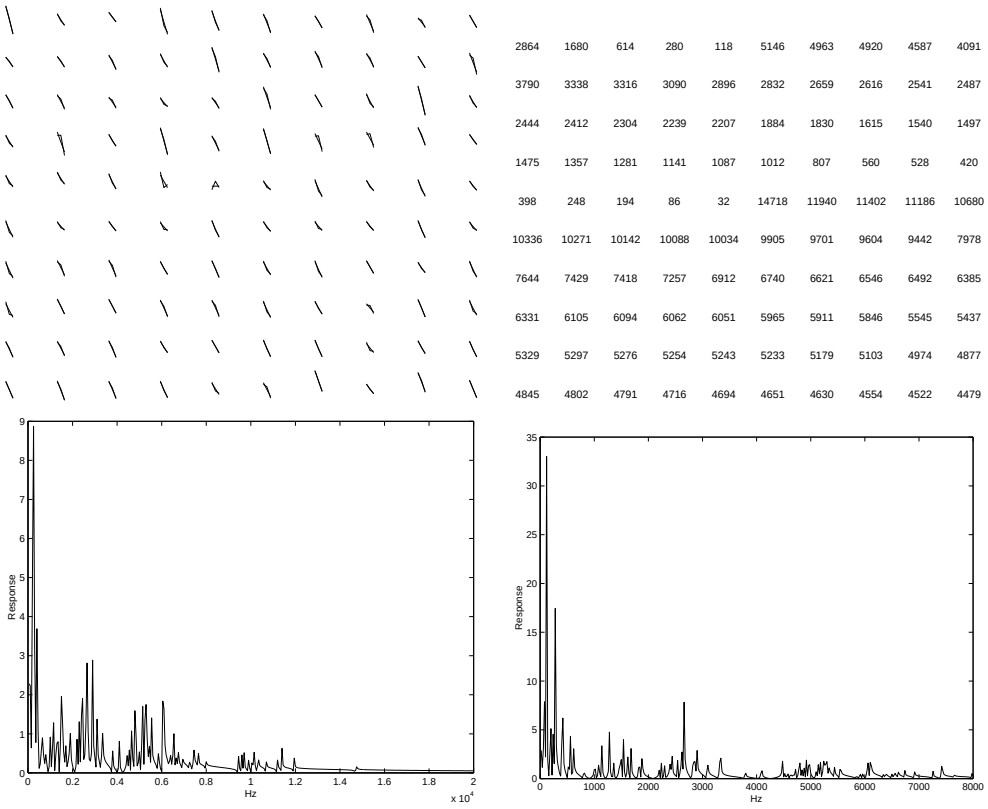

Figure B.19: Computer tower with a window of 4096

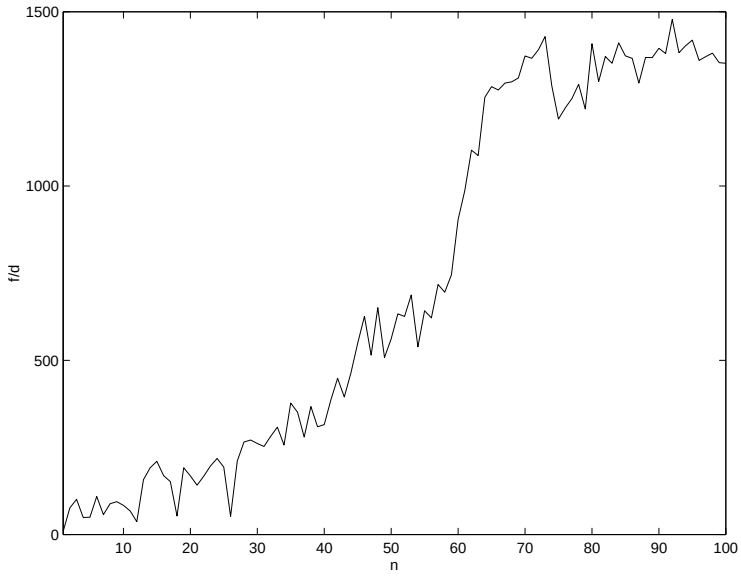

Figure B.20: Computer tower with a window of 512: Material constant

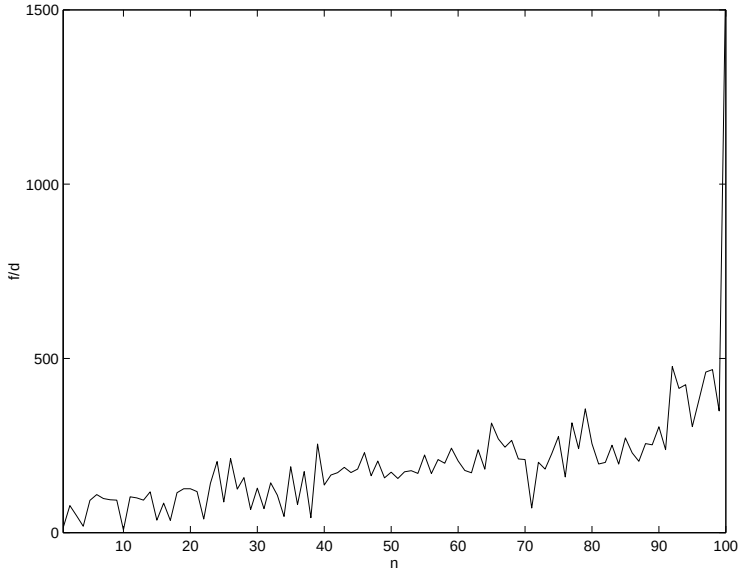

Figure B.21: Computer tower with a window of 2048: Material constant

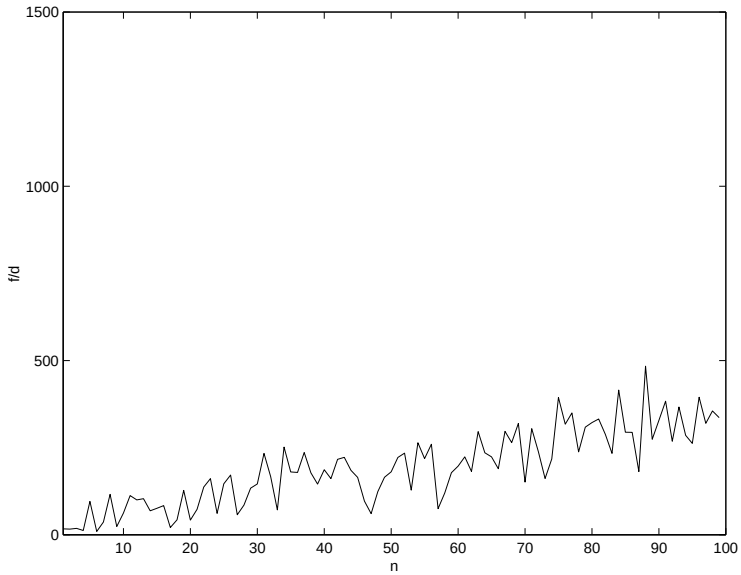

Figure B.22: Computer tower with a window of 4096: Material constant

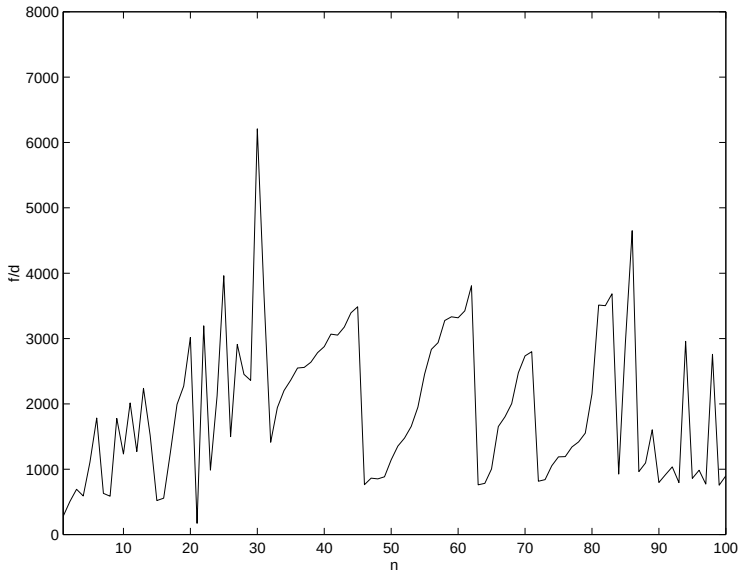

Figure B.23: Bell with a window of 1024: Material constant

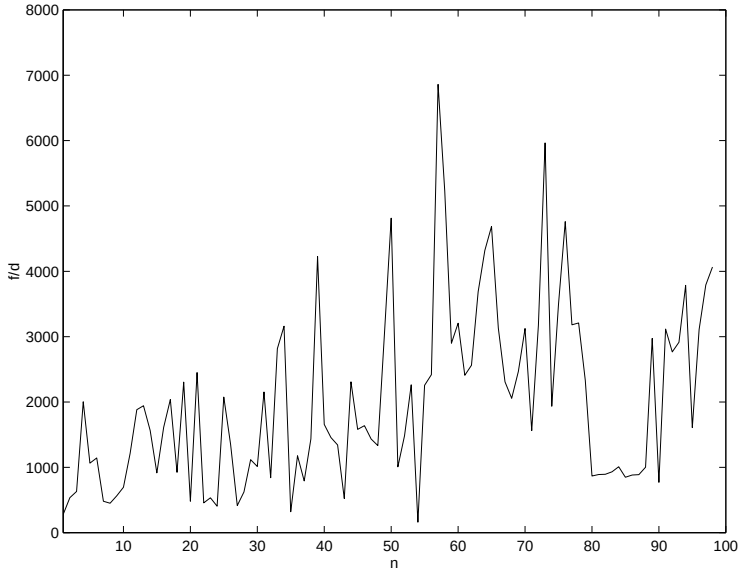

Figure B.24: Bell with a window of 2048: Material constant

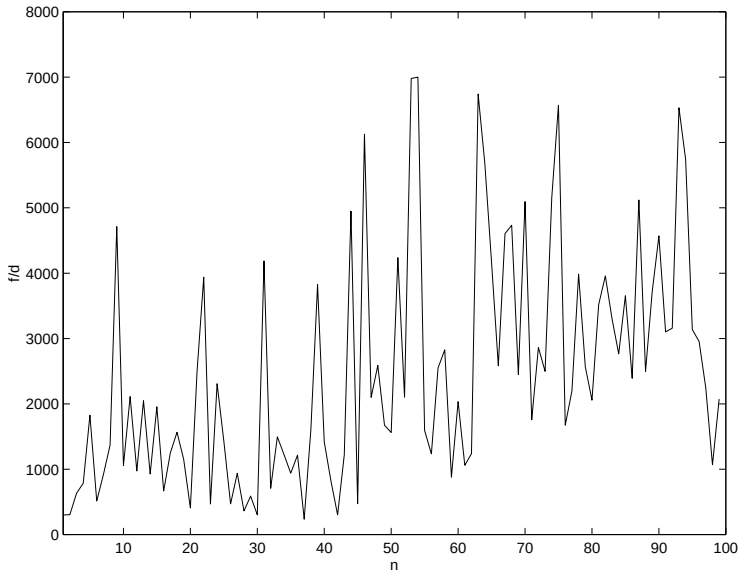

Figure B.25: Bell with a window of 4096: Material constant

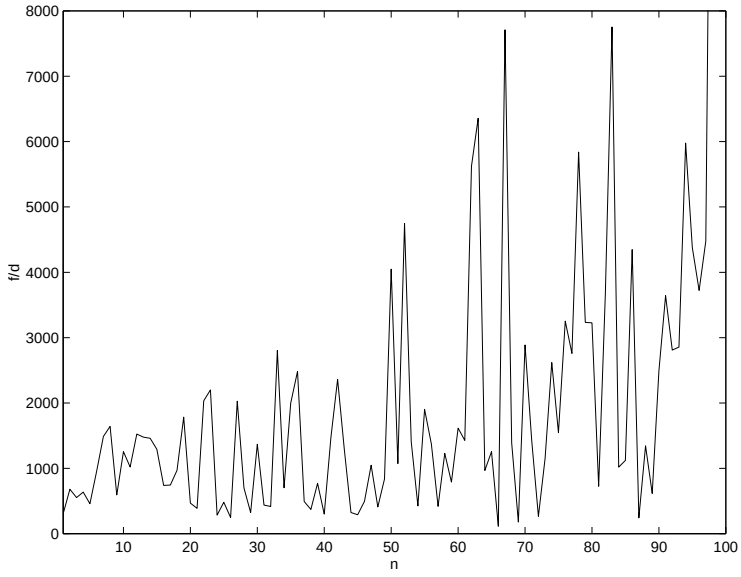

Figure B.26: Bell with a window of 8192: Material constant

Appendix C

Audio Synthesis Techniques for Music

A comprehensive overview of musical audio synthesis techniques is given by Herbert Janßen, on the WWW site [7], from which the following material is compiled.

- **Additive Synthesis, Fourier Synthesis**

  Any sound, however complex it may be, can be described as a mixture of a number of sine wave components with different phases and amplitudes. These are the partials of a sound, which are also called harmonics if their frequencies are an integer multiple of the fundamental frequency.

  The method to generate a complex sound spectrum as the sum of (many) simple sine waves is called Fourier synthesis, after Joseph Fourier who found its mathematical basis. The more general term additive synthesis can also be used if the waveforms added are not sine waves.

  Ideally, a lot of sine oscillators are needed for Fourier synthesis. How many

depends on the required range and brightness: a bright bass note, think of a slap bass, may need more than hundred, while a high pitched harmonic sound will probably need only a dozen.

For dynamical sounds and expressive play of an additive synth very many parameters are needed: ideally, each oscillator should have its own amplitude envelope, pitch envelope, velocity sensitivity and modulation routing.

Although this may sound like the hardware is the limiting factor, the usability is even more so. Most of the many parameters have only little influence on the sound and generally it is very hard to estimate how the spectrum of a desired sound looks like. Thus the simulation of acoustic instruments seems to be impossible without appropriate analysis hardware and software.

- **Subtractive Synthesis**

  This is just the classical method of synthesis used in most analog synths and in most sample playback synths and samplers.

  Subtractive synthesis means that you take a sound (preferably a spectral rich one like a sawtooth or a square/pulse wave, or a sample of a grand piano) and route it trough a modulatable filter and amplifier to change its timbre. This way you reduce the level of some partials of the original spectrum and hence the term. The terminology is a bit fuzzy for the real world, since almost any synth uses filters. The general usage of the term tends to refer to the classical "oscillator/filter/amplifier" trinity though.

  What is nice about subtractive synthesis is that by selecting the oscillator waveform or sample, the basic timbre is rather well determined and the usual filter and amplifier parameters allow to effectively tweak it to make it brighter,

duller, more percussive etc.

The main problem with subtractive synthesis is that its tools are rather bold: the oscillator waveforms or samples have a distinct character that is hard to overcome and the usual filters do not allow for very subtle changes.

- **Analog Synthesis**

  Analog Synthesis is not really a synthesis method, but rather a hardware issue: analog synths use analog instead of digital electronics to create their sounds. What most of them can do in terms of synthesis methods is quite simple subtractive synthesis. However some more advanced machines and especially modular synths may use a great variety of synthesis methods including FM, wave shaping, vector synthesis and others.

  Analog synths are considered "cool" nowadays, because of their supposedly "warm" and "fat" sound. So what is so special about analog synthesis then?

  First, most analog synths have an extensive user interface with dedicated knobs and switches for every function. This gives a very intuitive access and results in instant gratification for sound tweaking. This is possible because typical analog synths offer a limited number of control parameters, but those parameters are highly effective.

  Second, rather simple analog circuitry can perform the functions needed for subtractive synthesis rather well: the resulting sound will be artificial but with slight variations and instability. Thus, the sound quality is lively in a way similar to acoustic instruments. On the other hand most digital synths compromise in sound quality to achieve the high number of voices many buyers seem to be fond of today.

Modular analog synths have the additional advantage that there is no distinction between audio and control signals. Everything is just a control voltage which results in a vast number of patching possibilities. These beasts are very rare and pricy nowadays, but are probably still the best way to learn about synthesizing sounds, and can be used as a musical instrument of exceptional power.

- **Sample Playback (PCM, AWM, AWM2, AI, ...)**

  This is a form of subtractive synthesis that is also called PCM (Pulse Code Modulation), AWM (Advanced Wave Memory), AWM2 (Advanced Wave Memory Version 2) AI (?) by manufacturers. Usually all those terms refer to basically the same thing: An audio signal e.g. a miked acoustic instrument or an electrical or electronical instrument is sampled (digitized) and the recording is stored in RAM or ROM. If a device is able to sample and store the result in RAM or to disk, it is called a sampler. A device that can playback samples (from RAM, ROM or disk) at different pitches is called a sample playback synth. Most samplers and sample-based synths use subtractive synthesis although there are some samplers and synths that offer only very limited processing.

  The term PCM refers to the coding technique which is used in virtually all digital instruments. The terms AWM and AWM2 are Yamaha marketing slang for 16-bit sample playback and 16-bit sample playback with filters. Korg uses the term AI synthesis for their M1, which is just another sample playback synth, synthesis wise.

  Sample playback is what has made synths realistic sounding. On the other

hand sampling per se offers less options for expressive play than almost any other synthesis scheme. Samples that have a lot of inherent character or are easily recognized as an acoustic instrument are hard to shape, so filters and other processing options will merely adjust the timbre of the sample.

There are however unique possibilities in sample synthesis, but these are often not implemented in commercial synths. I'm talking about modulation of the sample playback parameters itself: extreme transposition of multisamples, modulation of sample start and sample loop length, multiple sample loops and more. Among others, various Ensoniq and Emu synths and samplers are capable of some of these. On many synths (including the SYs) even transposition (the changing of sample playback rate) can only be achieved with the trick of a constantly biased pitch envelope.

- **Frequency Modulation (FM, AFM)**

  Frequency Modulation is usually abbreviated FM or AFM (for Advanced Frequency Modulation). This is the family of synthesis methods that brought a breakthrough for commercial digital instruments in the eighties. Basically it means that you control the frequency of an audio oscillator by the frequency of another audio oscillator. The interesting aspect sound-wise is that you can generate a very wide variety of spectra plus many transient sound characteristics with FM (and not only the never ending variations of electric pianos and bells).

  FM was "invented" by John M. Chowning at Stanford and used in the academic computer music scene long before Yamaha marketed it. The commercial Yamaha implementation introduced some restrictions, but also some useful ex-

tensions like feedback.

FM exists in many different flavours: some analog synths resp. digital/analog hybrids are able to do a very basic FM. But FM relies mainly on the frequency ratios of the oscillators involved, and therefore requires very high tuning stability. Also FM becomes a versatile synthesis technique only if you have multiple oscillators with multiple envelopes to control their amplitude which results in a big number of components/modules needed in the analog realm. Maybe this is why it was and is not as popular with analog synths.

Yamahas digital FM implementations use custom chips to reduce cost. In case of digital FM, there are also many variants: depending on the number of oscillators (minimum is 2, most synths use 4 or 6, I recall to have heard of 10 in some Yamaha organs), whether there is a real envelope per oscillator (some very simple Yamaha sound chips like the one used in the old Atari STs miss them) and of course how variable the routing between the oscillators is (number of "algorithms, modulation and feedback paths).

- **RCM (Real-Time Convolution and Modulation) Synthesis**

  This is another marketing term by Yamaha and is mainly an extension to FM. The background is that you use a whole AWM2-element as modulator input for an AFM-operator, which also means that you can apply the filter on the sample before you put it through the FM-process. The former fact may be used to motivate the term modulation, the latter the term convolution (one possible algorithm for a filter is the convolution of the signal with a kernel). In my opinion the term RCM is ill defined and misleading. In lack of a better term, I will use RCM to denote the capability to use feed the AWM section

into the AFM section and vice versa.

- **Phase Distortion (PD) and Interactive Phase Distortion (iPD)**

  The terms phase distortion and interactive phase distortion were used by Casio for their synths (CZ and VZ series).

  Actually the two methods seem to be very different. The phase distortion synths (the CZ series) offer eight basic waveforms (saw, pulse, resonance, etc). Each of them can be morphed continuously from and to a sine wave via an eight-stage envelope, thus emulating the use of a low-pass filter on a basic analog synthesizer. Another two envelopes are used for pitch and amplitude and for two-oscillator patches a ring modulator can be used.

- **Wave Shaping**

  Wave shaping refers to a sound manipulation (not generation) technique which applies a (nonlinear) function on the original signal (i.e. the output of an oscillator). This scheme is similar in principle to analog distortion in a guitar amp or fuzz unit, but offers much more sound variation possibilities including resonance-like effects. Wave shaping can be used as an advanced synthesis method in a way similar to FM.

- **LA (Linear Arithmetic) Synthesis** This buzzword was used by Roland to describe their approach to digital sound synthesis in the eighties. It is based on the observation that the attack transient of a sound is its most important part with respect to human perception. Therefore the LA-synths (D-50, MT-32 and others, but most notably not the D-70) used a combination of sampled attack transients and simple digital oscillators with only sawtooth and pulse waveforms to generate the sustained part of the sound.

- **Ring Modulation, Amplitude Modulation**

  Ring modulation and amplitude modulation are not complete synthesis methods, but rather processing techniques that are quite common on advanced analog and digital synths. Sometimes these features are wrongly named or used when the actual implementation is quite different. Manufacturer specific terminology for similar schemes includes the terms cross modulation and FXM (frequency cross modulation).

  Ring modulation is the multiplication of two signals. The output of a ring modulator will contain the sum and the difference of all available input frequency pairs.

  Amplitude modulation is the multiplication of two signals, where one signal is always positive.

- **Vector Synthesis**

  The first synth to implement this paradigm was the SCI Prophet VS. The VS can mix four oscillators with different waveforms in real-time via a joystick controller and a multistage envelope. While this is a really simple concept, it is effective for expressive play and nice evolving sounds.

  The Korg Wavestations and the Yamaha SY22, TG33 and SY35 are other "vectorized" synths. The Yamahas can mix up to two FM and two sample elements, while the Wavestations mix up to four sample based wave sequencing oscillators.

  In principle most synths can do real-time vector synthesis, when fed with MIDI joystick data to cross-fade oscillators. If you like to try that you can rewire a PC game joystick to fit your synths pedal jacks.

- **Wave Table Synthesis**

  This term is used for two completely different things: Many sound card companies call their RAM based sample playback capabilities like this (because the samples are stored in a table in RAM).

  For the PPG Wave and the Waldorf Microwave and Wave synths this term is used to describe the ability to produce a sound by sequencing through a table of different waveforms during the duration of a single note. For the wave tables and waves there is a preset ROM area as well as a user loadable RAM area provided. Which entry of the wave table is selected may be controlled by an envelope, LFO or any other modulation source in real-time. Also these synths can interpolate between subsequent waveforms in the wave table thus smoothing the timbral change. The waveforms are single cycle ones, so realistic acoustic emulations are out of reach for this technique, but the vastly improved modulation capabilities, compared to sample playback, more than make up for this.

- **Wave Sequencing**

  This term means that a sequence of different sample segments can be used to generate a sound. Korg implemented this on their famous Wavestation synths. The Wavestations oscillators can sequence through programmable patterns of samples. Each of the patterns consists of a number of individually tunable sample snippets and each sample in the sequence is assigned its own level and duration. Typical for the Wavestation (and rather easy to program) are "rhythmic" wave sequences in which an oscillator steps through a number of samples in a predefined periodic rhythm. The Wavestations also combine this

with vector synthesis capabilities.

- **Granular Synthesis**

  Granular Synthesis means sequencing through very many very short sound (sample) snippets. The difference to wave sequencing is that the single samples are played for such a short time, that the sequencing is heard more as a timbre than as a rhythm. Granular synthesis has been developed in the academic computer music scene and has not found its way into commercial products so far.

- **Physical Modeling Synthesis**

  Physical modeling (PM) is a whole class of synthesis methods that do not synthesize sound based on an abstract mathematical description, like the Fourier transform for additive synthesis or by classical signal processing means like filtering for subtractive synthesis, but rather tries to model the diverse instruments themselves: e.g. the bow, string and resonance corpus of a cello or the plucking finger, string and body for an acoustical guitar.

  There are many different physical modeling algorithms including relatively simple ones like Karplus-Strong synthesis and rather complex ones like the waveguide [67] approach which uses multiple delay lines (comb filters) to model strings or air columns.

  The biggest advantage of physical modeling is the real-time control it offers. While other synthesis methods offer some algorithm specific and rather arbitrary control parameters like filter cutoff or modulation index, physical modeling enables the use of control parameters that are more musical and have a more complex influence on the timbre and phrasing. Examples for such

parameters are embouchure or tonguing.

Another advantage of PM is that the sound generation is context sensitive: a note on a clarinet model will sound different if it is played with legato binding to its predecessor or with a little pause in between. The dependency is much more complex than with the traditional synthesizer portamento or glide function. Another example: the pitch bend of a clarinet patch will not just linearly shift the frequency of the note, but the synth will respond in a similar way to a real clarinet, i.e., it will shift the frequency and timbre for a while but then jump to the octave.

PM has its disadvantages though: Instrument models have to be designed with great care and a lot of knowledge of both instrument acoustics and the necessary math. On the available PM instruments by Korg and Yamaha, editing is only possible via macro parameters of the otherwise hard coded instrument models, probably the only practicable way to provide sound programming to the "normal" user.

- **Karplus-Strong Synthesis**

  This synthesis method uses a percussive sound, like a noise burst or a single pulse, which excites a delay unit with feedback. If the feedback is high enough (90-99%) an exponentially decaying sound with definite pitch will result. It is the delay time that determines the pitch in this case. To be exact the delay time equals the period of the resulting periodical wave. This synthesis method is particularly well suited for emulating plucked strings and other percussive harmonic sounds. To make the decay more realistic, one can include a low-pass filter in the feedback path, so that higher harmonics are damped faster.

Karplus-Strong synthesis is actually a very simple form of physical modeling and shares the most important physical modeling advantage: since the percussive sound acts as an exciter for the delay loop that produces the harmonic sound, one can change the plucking/fingering of the model-led string by changing the percussive sound, and change the string by controlling the delay line parameters.

- **Modal Synthesis**

  This synthesis techniques uses a bank of resonators with individually adjustable frequencies and dampings, which is driven by some input signal. Non-harmonic percussive instruments such as bells and marimba have been successfully modeled with this technique [20]. For general musical instrument synthesis this technique has the disadvantage that a resonator is needed for every partial, and the partials are spaced linearly for most musical instruments, with the exception of non-harmonic percussive instruments like bells. The many resonators needed leads to a large computational cost, which can be avoided by using waveguides or comb filters, which by themselves already have a complete harmonic spectrum and are therefore better building blocks.