Chapter 7:

THE PRACTICALITIES OF NONSPEECH AUDIO

Introduction

To be written.

The Composition of the Team

eg., EuroPARC Resources

• Gaver - psychologist

• Buxton/Bristow - music

• physicist

• BBC - sound effects, etc

• Patterson - Human Factors / Psych

Planned this way, but didn't work out that way. In reality, were inspiration and sounding board, not collaborators. If all in a "team", management/logistic problems. i.e., mobility. Main sound work by individual (Gaver) with applications person. Real issue of overhead to get work done efficiently (the good people that you want to work with are typically busy.)

Everyone says you need a team, must have a musician, etc. But ... having a musician doesn't guarentee anything. A sound effects specialist might be far more useful. What is clear is that to avoid annoyance, etc., must be controled. Early systems willl, if the introduction of laserprinters is an example, be subject to the audio equivalent to fontitis.

In abscence of full ranginfg design team (and the above is just for the audio component - how unmanageable would be the whole project meetings?), a methodology with adequate user testing and evaluation, and just plain living with system can provide safeguards to overcome the imperfections/compromises in the team composition.

Trade-off between nobody can have all the skills, and be really good at all, which suggests a team, nbut you want the agility and mobility of small group, for budgetary for realisty.

Generating and Controlling Sounds

In order to do research in the area of nonspeech audio, one needs to be able to exercise control over a rich and potent sound repertoire. In the past few years, there has been a lot of progress in the development of technologies to support such control. From a practical point of view, the researcher's life is a lot easier than it was when Bly first started her experiments.

In this chapter, we discuss some of the "nuts-and-bolts" practicalities of working with nonspeech audio. We look at some important hardware and software issues that affect our ability to synthesize and process sounds.

While some of the information in this chapter will date quickly, the underlying issues will remain with us for some time. The objective is to help new researchers get started.

We want to create systems that will impose minimal constraints on possible research. To achieve this, we make a few assumptions:

• *Response time:* About 5 *ms* is the upper bound on response for audio cues.

• *Multi-Sound / Multi-Timbre:* Not only must we be able to control more than one sound at a time (polyphony), we must have independent timbral control over each (polytimbre).

- *Realtime continuous control:* Triggering sound events is not sufficient. We need realtime continuous control over various types of modulation, including loudness, pitch, timbre, etc.
- *Processing as well as synthesis:* Signal processing techniques such as filtering and reverberation are important, especially in placing sounds along the foreground/background continuum. Provision of these facilities is as important as sound synthesis.
- *Burden on host:* Maintaining adequate response time is hard enough without the additional burden of sound. Having the addition of sound degrade the performance of other functions is unacceptable.

Just as visual displays must echo characters as they are typed, so must auditory displays be synchronized with the processes with which they are associated.

But audio interfaces must do more than simply echo an event with a sound. In thinking about response, the continuous shaping of a window in a direct manipulation system is perhaps a better analogy than the echoing of a typed character. In most cases, it is not sufficient simply to trigger a sound at a particular moment. To provide a proper foundation for research, the sound control system must also support dynamic "shaping" of the sound's attributes. As the visual display of a window stretches when we drag its corner, so must a sound vary continuously as the parameter to which it is associated is manipulated.

We need control over sound which is rich, immediate, and continuous. However, even with the technology available today, this is not always easy. Much of the difficulty lies within our programming environments, which make it difficult to instrument processes. But even with the appropriate software tools, it can be very difficult to provide the desired control without adversely affecting other aspects of the system's performance.

One way to "save cycles" on the host machine is to delegate the audio processing and synthesis to peripheral hardware. This way, the computational bandwidth demanded from the host is reduced from audio levels down to control levels - a reduction of as much as two orders of magnitude.

While we briefly discuss some aspects of host-generated sounds, our working assumption is that the host is already busy, and that most audio work will employ external hardware. We briefly cover "hard" and "soft" synthesis, and synthesis versus playback (sampling). We pay particular attention to the control protocol standard in the music business, MIDI, and mention specific products as examples. However, please keep in mind the following points:

- We only discuss products with which we are familiar.
- We are not familiar with everything.
- New products come out every day.

This is just another way of saying that our examples are just that, examples. Use them when evaluating other products, then make your own choice.

Parameterized Icons

Auditory icons not only reflect categories of events and objects as visual icons do, but are *parameterized* to reflect their relevant dimensions as well. That is, if a file is large, it sounds large. If it is dragged over a new surface, we hear that new surface. And if an ongoing process starts running more quickly, it sounds quicker.

The possibility of parameterizing icons, whether auditory or visual, has largely been neglected in interface design [though see 2]. But parameterized icons can serve as more than mere labels for their referents, providing rich sources of information about relevant dimensions such as size, age, or speed as well. Parameterization allows single objects or events to be assessed along a number of dimensions. In addition, it creates families of icons that retain perceptual similarity while allowing comparison among members. In general, parameterized icons allow a great deal of information to be conveyed perceptually rather than symbolically.

Creating Parameterized Auditory Icons

Unfortunately, it is difficult to parameterize auditory icons because it is difficult to control a virtual source of a sound along relevant dimensions. Standard synthesis techniques have been developed for creating music, and thus afford changes of a sound's pitch, loudness, duration and so forth. But they do not make it easy to change a sound from indicating a large wooden object, for instance, to one specifying a small metal one. It is easy to create a wide variety of beeps and hums using standard synthesis techniques, but difficult to create and manipulate sounds along dimensions that specify events in the world.

Because of the limitations of standard synthesis techniques, interfaces using auditory icons have relied on digital sampling in their implementations. Desired sounds are captured by recording them on a computer, shaped by a designer, then played back and manipulated under the control of the interface. This enables the use of much more complex and realistic sounds than can be created by readily available synthesis algorithms. However, there are several drawbacks of sampling that limit its utility as a technique for creating and using auditory icons:

- It is difficult to capture an actual event that sounds like what is desired, because sounds are invariably coloured by the technologies used to record them.
- Shaping recorded sounds along dimensions relevant for auditory icons is difficult because available software is designed for making music.
- Real-time modification of sounds on playback is even more limited.
- The amount of memory needed for complex auditory interfaces is often prohibitive (on the order of 10K bytes per second of sound).

These limitations constrain the possibilities for designing auditory icons, and make their creation difficult and time-consuming.

In this paper, I suggest an alternative in the form of a new type of synthesis algorithm developed as a result of basic research on auditory event perception, and describe several examples in sufficient detail to allow readers to implement and explore them. These algorithms allow sounds to be specified in terms of their sources rather than their acoustic attributes. They promise to overcome both the limitations of traditional synthesis algorithms and of sampling by allowing parameterized auditory icons to be specified along dimensions of virtual source events.

Acoustic Information For Events

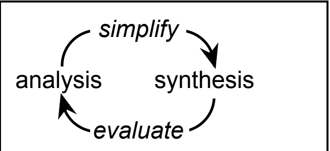


Figure 1: Traditional analysis and synthesis.

Creating algorithms that allow synthesis of virtual events implies an understanding of the acoustic information for event attributes – how sounds indicate the material or size of an object, for instance. Event attributes often have very complex effects on sounds, effects that must be described as functions of frequencies and amplitudes over time that specify the *partials*, or frequency components, that make up a sound. If these functions are understood, source attributes can be specified directly, instead of by using separate controls over partial frequencies, amplitudes, and durations. But how can we determine what these functions are?

Analysis and Synthesis of Events

One approach to this problem is suggested by the analysis and synthesis methods [9] used by computer musicians to capture the relevant properties of traditional instrument sounds (Figure 1). This approach involves recording sounds that vary along dimensions of interest and analyzing their acoustic structure using Fourier analysis or similar techniques. Hypotheses about acoustic information suggested by the analysis can be tested by synthesizing sounds based on simplified versions of the data. For instance, if one supposes that the temporal features of a sound indicates the event that caused it, but that its frequency makeup is irrelevant, one might use the amplitude contour from the original sound to modify a noise burst. The hypothesis can then be assessed simply by listening to the result.

In practice, however, it is often difficult to identify the acoustic information for events in the mass of data produced by acoustic analyses. Thus it is useful to supplement them with analyses of the mechanical physics of the event itself (see Figure 2). Studying the physics of sound-producing events is useful both in suggesting perceptible source attributes and in indicating the acoustic information for these attributes. Acoustic analyses help both in checking the adequacy of physical models and in evaluating particular parameters. Finally, the resulting models can provide the basis for synthesis algorithms that allow sounds to be specified in terms of sources attributes.

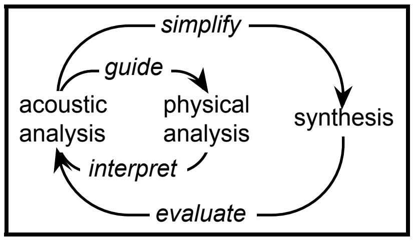


Figure 2. Analyzing and synthesizing events requires physical as well as acoustic analyses.

In the following sections, I discuss several case studies of events that have been studied in this manner and describe the synthesis algorithms that have resulted. The algorithms have been chosen for their utility in creating auditory icons, and are described in order of their complexity. I start with the sounds made by mechanical impacts, which involve a simple interaction of objects. Next I describe how more complex bouncing, breaking, and spilling sounds can be produced by specifying the temporal patterning of a series of impacts. A third algorithm allows the same virtual object to be hit and then scraped by distinguishing objects from the interactions that cause them to make sound. Finally, I describe an algorithm for producing machine-like sounds, showing that high-level attributes of complex events may be synthesized directly.

Impact Sounds

Many of the sounds we hear in the everyday world involve one solid impacting against another. Tapping on an object, placing it against another, letting it fall – all involve impact sounds. In the interface, impact sounds are useful in the design of a variety of auditory icons that indicate events such as selecting a file, moving it over another, or attaching one object to another.

Several studies have explored the perceptible attributes of impact events and the acoustic information about them. In this section, I briefly review these studies, then show how the information they provide can be used to create a synthesis algorithm that allows impact sounds to be specified along dimensions of the virtual source.

Mallet Hardness, Material, and Size

Freed and Martins [3] studied people's perception of the hardness of mallets used to strike objects. They recorded the sounds made by hitting cooking pans with mallets of various hardnesses, asked people to judge hardness from the sounds, and used a model of the peripheral auditory system to analyze the acoustic correlates of their judgements. They found that the ratio of high to low frequency energy in the sounds and its change over time served as the most powerful predictors of subjects' hardness judgements. To a good approximation, then, mallet hardness is conveyed by the relative presence of high and low frequency energy.

I studied the acoustic information available for the length and material of struck wood and metal bars and people's abilities to perceive these attributes [4]. I recorded and analyzed the sounds made by wood and metal bars of several different lengths, and developed a model of the physics of impacts that combined analytical solutions to the wave equation for transverse vibrations in a bar [e.g., 8] with empirical measurements of damping and resonance amplitudes. This model was used both to aid interpretation of the acoustic analyses and to synthesize new tokens.

The material of the bars made several effects on the sounds they made. Perhaps most important, materials have characteristic frequency-dependent damping functions: the sounds made by vibrating wood decay quickly, with low-frequency partials lasting longer than high ones, while the sounds made by vibrating metal decay slowly, with high-frequency partials lasting longer than low ones. In addition, metal sounds have partials with well-defined frequency peaks, while wooden sound partials are smeared over frequency space. These results accord with Wildes and Richards' [12] physical analyses of the audible effects of the internal friction characterizing different materials, which show that internal friction determines both the damping and definition of frequency peaks.

Changing the length of a bar, on the other hand, simply changes the frequencies of the sound it produces when struck, so that short bars make high-pitched sounds and long bars make low ones. However, the effects of length may interact with the effects of material. For instance, frequencies change monotonically with length, but the frequency of the partial with the highest amplitude depends on material and thus may change nonlinearly with length [4]. These nonlinearities – and the perceptual confusion they cause – may be avoided by simplifying the model so that partial amplitudes do not depend on material.

A Synthesis Algorithm for Impact Sounds
These results may be captured in a synthesis model that uses frequency and amplitude functions to constrain a formula for describing logarithmically decaying sounds. This formula describes a complex wave created by adding together a number of sine waves with independent initial amplitudes and logarithmic decay rates:

$$G(t) = S_n F_n \; e^{-d_n t} \; \cos w_n t \qquad (1)$$

where G(t) describes the waveform over time, F_n is the initial amplitude, d_n the damping constant, and w_n the frequency of partial n.

This formula has two properties that make it a useful foundation for synthesizing auditory icons. First, its components map well to event attributes. Second, it can be made computationally efficient using trigonometric identities.

Mapping Synthesis Parameters to Source Attributes
By constraining the values used in this formula, useful parameters can be defined which correspond well to the attributes of impact sounds discussed above. The formula involves three basic components: the initial amplitudes of the partials F_n, their damping $e^{-d_n t}$, and their frequencies $\cos w_n t$. These can be set separately for each partial. However, these three components also correspond to information for mallet hardness and impact force, material, and size and shape respectively (see Table 1). Thus it is more useful to define patterns of behaviour over the partials for each component.

Table 1: Mapping Parameters to Events

Term	Effect	Event Attribute
F_n	initial amplitudes	mallet hardness; force or proximity
$e^{-d_n t}$	damping	material
$\cos w_n t$	partial frequencies	size; configuration

For example, the partial frequencies w_n can be constrained to patterns typical of various object configurations. The sounds made by struck or plucked strings, for example, are harmonic, so that $w_n = n w_1$. The sounds made by solid plates, in contrast, are inharmonic and can be approximated by random frequency shifts made to a harmonic pattern. The sounds made by solid bars can be approximated by the formula $w_n = (2n + 1)^2/9$. Finally, the sounds made by rectangular resonators are given by the formula $w_n = c/2 \sqrt{(p^2/l^2 + q^2/w^2 + r^2/h^2)}$, where *c* is the velocity of sound, *l*, *w*, and *h* are the length, width and height of the box respectively, and *p*, *q*, and *r* are indexed from 0 [12]. An algorithm based on Formula 1, then, can be constrained so that one of these patterns is used to control the partial frequencies w_n. In addition, w_1 can be specified such that w_1 µ 1/size to reflect the size of the object (this affects all the other partial frequencies).

The initial amplitude of the partials, F_n, can be controlled by a single parameter corresponding to mallet hardness. Recalling that Freed and Martin's [3] results identified the ratio of high to low-frequency energy as a predictor of perceived mallet hardness, we might maintain a linear relationship among the partial's initial amplitudes, and use the slope from F_1 to control perceived hardness. Thus $F_n = F_1 + h(w_n - w_1)$, where h is the slope – note that h may be negative, so that higher partials have less amplitude than low ones. F_1 (and thus all the amplitudes) may also be changed to indicate impact force or proximity.

Finally, the damping constants for each partial (d_n) can be controlled by a parameter corresponding to material. A useful heuristic is to set $d_n = w_n d_0$, so that high harmonics die out relatively quickly for highly damped materials and last longer for less damped materials (e.g., metal, which has low damping, tends to ring; wood, which is highly damped, tends to thunk). This strategy is suggested both by Wildes and Richards [12], and by my own research [4].

In sum, Formula 1 can be controlled by parameters that make effects corresponding to attributes of impact events. Controlling overall frequency corresponds to the object's size, while the pattern of partial frequencies corresponds to its configuration. The overall initial amplitude corresponds to the force or proximity of the impact, while the pattern of partial amplitudes corresponds to mallet hardness. Finally, the degree of damping corresponds well to the virtual object's material. By controlling these five parameters, then, a wide range of sounds can be created which vary over several useful dimensions.

An Efficient Algorithm for Synthesis

Formula 1 is useful in allowing parameters to be defined in terms of source events. It is also attractive because it can be implemented in a computationally efficient way.

An efficient implementation of this formula relies on Euler's relationship $e^{iwt} = \cos wt + i\sin wt$ to rewrite Formula 1 as:

$$\begin{aligned} S_n &= \mathrm{Re}[(a_n + ib_n)(p + iq)] \\ &= \mathrm{Re}[(a_n p - b_n q) + i(b_n p + a_n q)] \qquad (2) \end{aligned}$$

where S_n is the nth sample, a_0 is the initial amplitude, $b_0 = 0$, $i = \sqrt{-1}$, $p = e^{-dt}\cos wt$ and $q = e^{-dt}\sin wt$. (A full derivation is available upon request.)

Samples can thus be generated by calculating p and q, setting a and b to the initial amplitude and 0, and applying equation 2. The output sample is the real part of the result, and a and b are updated to the real and imaginary parts respectively (see pseudocode in Figure 3). Computationally expensive sine and cosines need only be calculated once, and only four multiplications, one addition and one subtraction are needed for each partial for a given sample. The efficiency of this implementation allows fairly complex impact sounds to be generated in realtime on many computers.

```
p = cos(freq * 1/samplerate) * power(e, -1 * damping.rate * 1/samplerate);
q = sin(freq * 1/samplerate) * power(e, -1 * damping.rate
        * 1/samplerate);

a = initial.amplitude;

b = 0;


repeat for duration.in.secs / samplerate:

        anew = a * p - b * q;

        bnew = b * p + a * q;

        a = anew;

        b = bnew;

        output = anew;

end repeat;
```

Figure 3. Pseudocode for efficient generation of a logarithmically-decaying cosine wave (equation 5).

Breaking, Bouncing, and Spilling

The impact algorithm can serve as a fundamental element in algorithms used to synthesize more complex sounds. For instance, an early example of analysis and synthesis of sound-producing events is Warren and Verbrugge's [11] study of breaking and bouncing sounds. In this study, they used acoustic analyses and a qualitative physical analysis to examine the auditory patterns that characterize these events, and verified their results by testing subjects on synthetic sounds.

Consider the mechanics of a bottle bouncing on a surface (Figure 4A). Each time the bottle hits the surface, it makes an impact sound that depends on its shape, size, and material (as discussed above). Energy is dissipated with each bounce so that, in general, the time between bounces and the force of each impact becomes less. Thus bouncing sounds should be characterized by a repetitive series of impact sounds with decreasing period and amplitude.

When a bottle breaks, on the other hand, it separates into several pieces of various sizes and shapes (Figure 4B). Thus a breaking sound should be characterized by an initial impact sound followed by several different, overlapping bouncing sounds, each with its own frequency makeup and period.

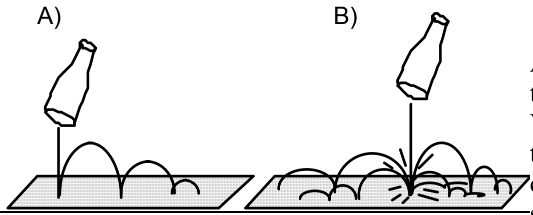


Figure 4. Bouncing (A) and breaking (B) sounds are characterized by the temporal patterning of a series of impacts [After Warren and Verbrugge, 11].

Acoustic analyses of bouncing and breaking sounds confirm this informal physical analysis. In addition, Warren and Verbrugge [11] found that people were able to distinguish tokens of bouncing and breaking sounds that were constructed by using these rules to splice tapes of impact sounds together.

Synthesized Breaking, Bouncing and Spilling

To create bouncing sounds, then, we need simply imbed the impact algorithm in another that calls it at logarithmically decaying intervals. To create breaking sounds, the bouncing algorithm is imbedded in another algorithm that calls it with parameters specifying sources of different sizes at times corresponding to several logarithmically decaying time series.

Several new event parameters become relevant for these algorithms: The initial height of the virtual object is indicated by the time between the first and second bounce, its elasticity by the percentage difference of delays between bounces, and the severity of breaking by the number of pieces produced. In addition, the asymmetry of the perceived object can be varied by adding randomness to the overall temporal pattern.

It becomes clear upon listening to sounds synthesized using this algorithm that although Warren and Verbrugge [11] claimed that information for breaking and bouncing depends only on temporal patterning, the perceived event depends on the virtual materials involved as well. For instance, if impacts specifying wooden objects are produced in a temporal pattern typical of breaking, we are liable to hear spilling rather than breaking. Similarly, if each of several virtual objects has different material properties, we again hear several spilling objects rather than breaking.

In sum, the impact algorithm described above can be used not only to generate the sounds made by mallets of different hardnesses striking virtual objects of a wide variety of shapes, sizes, and materials, but can also serve as the basis for more complex bouncing, breaking, and spilling sounds. As such, it serves as a research tool that allows the space of such sounds to be explored. Moreover, it provides an efficient method for generating families of related auditory icons. For instance, parameterized impact, bouncing, breaking and spilling sounds might be used to differentiate and provide details about the results of actions involving icons, windows, containers, and so forth.

From Impacts To Scraping

The sounds made by impacts and patterns of impacts are generally useful for creating auditory icons, but it is desirable to have access to sounds made by a wider range of events. In particular, it would be useful to generate the sounds made by the same object being interacted with in different ways. Using such algorithms, auditory interface designers might map a particular file to a particular object, and then hit, bounce, or scrape it depending on the relevant computer interaction.

In order to create such algorithms, it is necessary to separate the specification of a virtual object from that of the interaction that causes it to produce sound. This turns out to be possible because objects tend to vibrate only at certain invariant resonant frequencies. For example, the spectrogram in Figure 5 shows the sound made by a piece of glass being hit and then scraped across a rough surface. Note that despite the different temporal patterns of the sounds, the resonant modes of each are the same. These modes specify the object, then, while interactions determine the temporal pattern and amount of energy introduced to each.

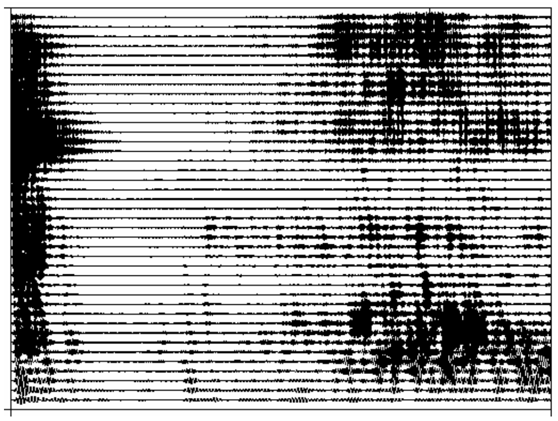


Figure 5. Spectrogram of a piece of glass being hit and then scraped: The resonant frequencies remain invariant over different interactions.

Because the effects of interactions and objects are distinct, each can be modelled separately. The resonant modes of a virtual object may be modelled as a bank of filters that allow energy to pass at particular frequencies. Interactions, then, can be specified by the pattern of energy passed through the filter bank.

Modelling Objects as Filter Banks

A simple formula for a bank of one-pole filters is:

$$y_n = \sum_m F_m(c1_m x_n + c2_m y_{n-1} - c3_m y_{n-2}) \qquad (3)$$

where F_m is an amplitude scalar for partial m, y_n is the nth output, x_n is the nth input, and:

$$c1_m = (1 - c3_m)\,[(1 - c2_m{}^2)/4c3_m]^{.5}$$

$$c2_m = (4c3_m \cos 2\pi f_m)/(c3_m + 1)$$

$$c3_m = e^{-2\pi b_m};$$

where f_m is the peak frequency and b_m the peak bandwidth of partial m.

The parameters used to control the impact algorithm can also be used to control this sort of filter bank. However, manipulating filter bandwidths to control damping actually provides more information for material than do simple manipulations of sine wave damping. Bandwidth *b* is proportional to damping: the narrower the resonance peak of the filter the longer the resonant response to excitation. This correlation between damping and the smearing of partials in frequency space corresponds well to the characteristics of sounds made by materials such as wood or metal [4, 12].

Simulating Interactions with Input Waveforms

A virtual object can be defined by the characteristics of the filter bank described above. The waveform passed through the filter bank, then, models the interaction that causes the object to sound. In this section, I describe two sorts of input waveforms that I have explored. The first models impact forces, the second scraping.

When objects are struck, the input forces are characterized by short impulses such as those shown in Figure 6. The energy of such impulses is spread out over many frequencies: the pulse width reflects low frequency energy, while its angularity reflects high frequency components. This corresponds to Freed and Martin's [3] characterization of mallet hardness. Hard mallets introduce force suddenly to an object, deforming it quickly, and thus introduce a relatively high proportion of high frequency energy to the resonant object. Soft mallets, in contrast, deform as they hit the object, introducing energy relatively slowly, and thus the corresponding impulses are characterized by a high proportion of low frequency energy. Shaping the impulses used to excite a filter bank, then, is a physically realistic way to control perceived mallet hardness.

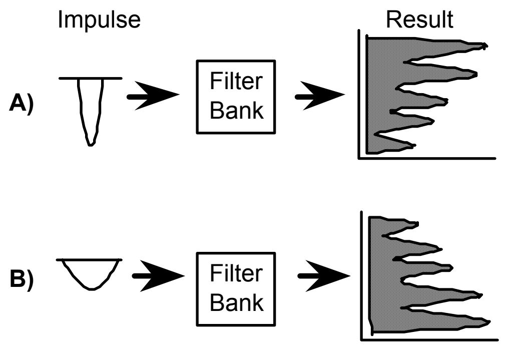


Figure 6. Sample impulse waveforms characterizing different impacts (see text).

When an object is scraped, force is applied more continuously. Scraping has been relatively unexplored in terms of its physical or perceptual attributes. However, an informal physical analysis suggests that the pattern of force on an object generated as it is scraped across a surface can be approximated by band-limited noise, where the center frequency of the noise corresponds to dragging speed, and the bandwidth to the roughness of the texture (see Figure 7). Although these parameters are only approximate, being less well motivated physically or psychologically than those used to model impacts, experience shows that a wide variety of realistic scraping noises can be produced using these heuristics.

In sum, the filter-based algorithm described in this section is based on a physically plausible model of sound producing events. By separating the parts of the model that specify the object from those specifying the interaction, a wide range of virtual sound-producing events can be simulated. The model can create any of the impact sounds that the algorithm described in the last section can. In addition, it can also be used to create a variety of scraping sounds (and, potentially, any other sound involving solid objects).

The ability to generate the sounds of the same object being caused to sound by different interactions offered by this algorithm has great potential for the creation of auditory interfaces. It allows the design of parameterized auditory icons in which the same interface object (e.g., a file) might make sounds indicating a variety of events (e.g., selecting, dragging, opening).

Machine Sounds

Just as complex interactions such as scraping can be modelled by a few summarizing parameters, so might still more complex events be captured succinctly by high-level descriptions. For instance, another class of sounds useful for auditory icons are those made by small machines. Sampled machine sounds were used effectively in the ARKola simulation [7], indicating ongoing processes that were not visible on the screen. More generally, they might be used to indicate background processes such as printing or compiling in more traditional multiprocessing systems.

A detailed account of the mechanical physics of machinery seems prohibitively difficult. But just as the scraping waveforms described above model the overall parameters of a complex force rather than each of the contributing details, so an approximate model of machines might capture some of the high-level characteristics of the sounds they produce. In particular, three aspects of machine sounds seem relevant for modelling: First, the overall size of the machine is likely to be reflected in the frequencies of sounds it produces; second, most machines involve a number of rotating parts that can be expected to produce repetitive contributions to the overall sound; and third, the work done by the machine can be expected to affect the complexity of the sound.

FM Synthesis of Machine Sounds

I have been exploring an efficient algorithm for creating a variety of machine-like sounds that capture these properties. The basic strategy is to synthesize a sound using complex tones that vary in a repetitive way, indicating cyclical motion. The rate at which the virtual machine is working, then, can be indicated by repetition speed, the size of the virtual machine by the base frequency, and the amount of work by the bandwidth of the sounds (see Figure 8).

This class of sound may be synthesized efficiently using Frequency Modulation (FM) synthesis [1]. FM synthesis involves modulating the frequency of a carrier wave with the output of a modulating wave. This produces a complex tone with a number of frequency components spaced equally around the carrier wave and

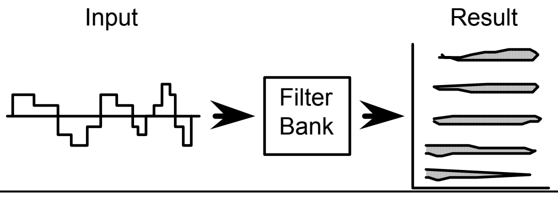


Figure 7. A sample force waveform characterizing a scrape with increasing speed.

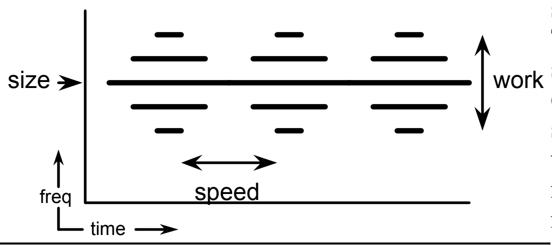


Figure 8. Machine sounds can be characterized by a compl
wave that varies repetitively over time.

separated from one another by the modulating frequency. The number of components (and thus the bandwidth of the sound) depends on the amplitude of the modulating wave (see Figure 9). Thus machine sounds can be created simply by associating the carrier frequency with the size of the virtual machine, setting the maximum amplitude of the modulator to the amount of work done by the virtual machine, and modulating the amplitude of the modulator according to the speed of the virtual machine (see pseudocode in Figure 10).

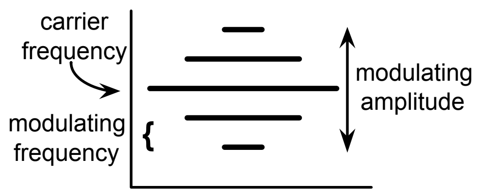


Figure 9. FM synthesis, which allows simple control over complex sounds, is well suited for generating machine-like sounds.

The resulting sounds are pitched humming noises that pulse at the speed of the virtual machine. When "work" is low, the throbbing is subtle; when it is high, it becomes quite pronounced. Moreover, the quality of the sounds can be varied by changing the ratio of modulating to carrier frequency: when the two are an integral multiple of one another, the resulting sound is harmonic, when they are not, the sound is inharmonic or noisy. Using this algorithm, then, a wide variety of machine-like sounds can be produced for use as indicators of ongoing processes in multiprocessing systems.

```
mod.wave.amp = work * sin(speed * time/samprate);

mod.wave.sample = mod.wave.amp * sin(mod.freq *        time/samprate);

output = amp * sin((size + mod.wave.sample) *     time/samprate);
```

Figure 10: Pseudocode for generating machine-like sounds characterized by speed, size, and work.

Conclusions

The algorithms described here allow the synthesis of a variety of everyday sounds specified in terms of attributes and dimensions of the events that cause them. Because they are based on a combination of acoustic and physical analyses and use relatively sophisticated synthesis techniques, they capture a great degree of the richness and complexity of their naturally produced counterparts. Because they are specialized for the classes of event they are to simulate, they are efficient, and can generate sounds in realtime on many computers. Finally, because they have been designed with potential applications in mind, the events they simulate are those useful for auditory icons.

These algorithms vary in their physical accuracy. Some are based on quantitative physical analyses, while others are based on more qualitative, informal descriptions of events. Moreover, even the quantitative analyses are only approximate. For instance, the physics of a struck bar of wood is much more complex than implied by the simple account given here. Insofar as these algorithms are approximate, the sounds they produce will differ from those made by real events.

Nonetheless, these algorithms do produce quite realistic sounds. Listeners comment that they have the impression of hearing an actual event rather than a synthesized sound. Insofar as the sounds do differ from those made by real events, they may be considered as "cartoon-sounds," sounds which capture the relevant features of their sources just as visual caricatures (or graphical computer icons) capture those of theirs. For the purposes of simulating sound-producing events, then, these algorithms are adequate. For the purpose of creating auditory icons, they show great potential, combining flexibility, intuitive controls, efficiency, and relevance.

I have specified these algorithms here in sufficient detail that readers may implement and explore them, in the hope that they will spur further research on parameterized auditory icons. These algorithms open many possibilities for the design of rich auditory interfaces. Impact and scraping sounds can be used to increase the tangibility of graphical objects in direct manipulation interfaces. Bouncing, breaking, and spilling sounds can be used to indicate events in virtual reality systems. Machine sounds might allow us to hear a remote printer as our job reaches the queue, and characteristics of the sound might tell us how fast the job is printing or how much time it will take. In sum, using these algorithms we can design interfaces that we can listen to the way we do to the everyday world.

The Host as Sound Generator and Processor

Realtime sound synthesis and processing are difficult and computationally demanding. Consequently, most computers are severely restricted in their ability to produce other than very basic sounds (beeps, bells, etc.), unless they have been explicitly designed with sound in mind.

Despite these limitations, it is clear that even the most primitive sounds can have a valuable effect on improving the user interfaces for some applications. Video games often demonstrate that sound can be used creatively on a small computer, even when executing a computationally demanding application. It can be done, but it can also be very difficult.

In some cases, the effort is worth it, despite the constraints. This is true, for example, for some applications that run on an installed base of machines which cannot be updated. Similarly, for the visually impaired, it is likely that some sound is better than none.

Some personal computers do have reasonable sound processing capabilities. Both the Commodore *Amiga* and the Apple *Macintosh* have reasonably good software support for sound by way of their programmer's tool-kits. The Amiga has special hardware that enables it to play up to four different complex sounds simultaneously. The Macintosh also has reasonable sound capabilities, as demonstrated by Gaver's *SonicFinder*. However, it is very difficult to play more than one sound at a time.

At the upper end of the scale, however, the only scientific workstation that has been designed with sound in mind is the *Next* machine. This may, in fact, be the ultimate workbench on which to study nonspeech audio, but too little is known about it at this stage to form a concrete opinion.

In the middle are a range of specially designed machine-specific boards that add audio capabilities to machines. These are discussed below.

What is clear is that no matter how good the hardware, the software must be carefully implemented for sound to be effectively integrated into a system. Making a call to a sound routine is not the same as calling some other system service, such as a disk access. In a Unix environment, for example, implementing sound routines as system calls would be far too slow to be effective. Sound service routines must be light-weight processes that have the calling overhead of applications routines, not system calls.

Synthesizing Everyday Sounds

Research on what people hear and how they hear it is useful in characterizing people's experience of sound in the way suggested by everyday listening. But everyday listening also suggests new ways to think about creating and manipulating sounds. In this section, we discuss several algorithms for the synthesis of sounds that can be controlled along parameters of their sources.

In a sense, these algorithms are instantiated theories about what and how we hear. They have not been subjected to empirical testing, although they could be. Instead, the identification of synthesis parameters with source attributes is based exploratory listening as well as on the research reported above.

Above all, however, these algorithms are designed to provide a practical means to create everyday sounds that can be controlled along useful dimensions. They are meant for eventual application, and thus they are shaped by their use of efficient synthesis techniques as much as their physical plausibility. Thus these algorithms fill a kind of middle ground between the relatively academic research reported in this chapter and the purely applied work described in the next.

Creating Impact Sounds With Constrained Additive Synthesis

A general formula for describing impact sounds is based on the simple addition of a number of sine waves with independent initial amplitudes and independent logarithmic decay rates. The formula for these sounds is:

$$G(t) = \Sigma_n \Phi_n \, e^{-\delta_n t} \cos\omega_n t \qquad (1)$$

where Φ_n is the initial amplitude of partial n, δ_n is the damping constant for partial n, and ω_n is partial n's frequency. This formula has two properties that make it an interesting foundation for synthesis. First, its components map well to properties of objects. Second, through the use of trigonometric identities, it can be made very computationally efficient.

Mapping Synthesis Parameters to Source Attributes

There are three basic components to this formula: Φ_n, or the initial amplitudes of the partials; $e^{-\delta_n t}$, or their damping, and $\cos\omega_n t$, the partial frequencies. In general, these can be set separately for each partial n. However, it is more useful to define patterns of behavious over the partials. By adding such constraints, useful parameters can be defined which correspond well to source attributes.

For example, the partial frequencies ω_n can be constrained to patterns typical of various object configurations (as can be heard in Sound Examples N - N). The sounds made by struck or plucked strings, for example, are harmonic, so that $\omega_n = n\omega_1$. The sounds made by solid plates, on the other hand, are inharmonic and can be approximated by random frequency shifts made to a harmonic pattern. The sounds made by solid bars can be approximated by the formula $\omega_n = (2n + 1)^2/9$. Finally, the sounds made by rectangular resonators are given by the formula $\omega_n = c/2 \sqrt{(p^2/l^2 + q^2/w^2 + r^2/h^2)}$, where c is the velocity of sound, l, w, and h are the length, width and height of the box respectively, and p, q, and r are indexed from 0. An algorithm based on Formula 1, then, can be constrained so that one of these patterns is used. In addition, ω_1 can be specified (affecting all the other partial frequencies) in order to reflect that size of the object ($\omega_1 \propto 1/\text{size}$).

Sound examples N - N: Synthesized sounds which are varied according to the pattern of partial frequencies (shape), and overall frequencies (size).

The initial amplitude of the partials, Φ_n, can also be controlled by a single parameter corresponding to a source attribute. Recalling that Freed's (1990) results identified the spectral centroid as a predictor of perceived mallet hardness, we might maintain a linear relationship omong the partial's initial amplitudes, and use the slope from Φ_1 to control perceived hardness (e.g., Sound Examples N - N). Thus $\Phi_n = \Phi_1 - (\omega_n - \omega_1)H$, where H is the slope (note that H may be negative, so higher partials are of greater amplitudes than lower ones). Of course, Φ_1 (and thus all the amplitudes) may be changed to indicate impact force or proximity.

Sound examples N - N: Synthesized sounds which are varied according to the pattern of initial amplitudes (mallet hardness) and overall amplitude (force or proximity).

Finally, the damping constants for each partial (δ_n) can also be controlled by a single parameter. A useful heuristic is to set $\delta_n = \omega_n\delta_0$, so that high harmonics die out relatively quickly for highly damped materials and last longer for less damped materials (e.g., metal, which has low damping, tends to ring; wood, which is highly damped, tends to thunk). This correspondence is suggested both by Wildes and Richards (xxx), and by Gaver (1988). Several of the other parameters suggested by Gaver (1988), such as the frequency band of peak vibrations for different materials, are not included in this account. But although material may affect these other attributes of sound, they seem also to be the cause of interactions in the effects of material and length found by Gaver (1988). Using damping to manipulate perceived material seems pragmatically advantageous in avoiding such interactions.

Sound examples N - N: Synthesized sounds which are varied according to the pattern of damping (material).

In sum, Formula 1 can be controlled by five parameters which make effects that correspond reasonably well to attributes of source events. Controlling overall frequency corresponds to the virtual object's size, while the pattern of partial frequencies correspond to its configuration. The overall initial amplitude corresponds to the force or proximity of the impact, while the pattern of partial amplitudes correspond to mallet hardness. Finally, the degree of damping corresponds well to the virtual object's material. By controlling these five parameters, then, we can create a wide range of sounds which vary over several useful dimensions.

An Efficient Algorithm for Synthesis

Formula 1 is useful in allowing parameters to be defined in terms of source events. It is also attractive because it can be implemented in a computationally efficient way.

An efficient implementation of this formula[1] relies on Euler's relationship:

$$e^{i\alpha t} = \cos\alpha t + i\sin\alpha t \qquad (2)$$

This implies that for a single logorithmically decaying cosine wave:

$$e^{-\delta t}\cos\alpha t = \mathrm{Re}[(e^{i\alpha-\delta})^{t}] \qquad (3a)$$

Since t = nT, where T is the sampling interval, equation b can be expressed as:

$$S_n = \mathrm{Re}[(e^{i\alpha-\delta})^{T}]^{n} \qquad (3b)$$

Where S_n is the value of the nth sample. Or:

$$S_n = \mathrm{Re}[S_{n-1} * \lambda], \text{ where } \lambda = e^{(i\alpha-\delta)T} \qquad (4)$$

Now, $S_n = a_n + ib_n$, where $S_0 = a_0$ is the initial amplitude, and $b_0 = 0$, and

$\lambda = p + iq$, where $p = e^{-\delta t}\cos\alpha t$ and $q = e^{-\delta t}\sin\alpha t$. Thus from 4:

$$S_n = \mathrm{Re}[(a_n + ib_n)(p + iq)]$$

$$= \mathrm{Re}[(a_n p - b_n q) + i(b_n p + a_n q)] \qquad (5)$$

This means that samples can be generated by first calculating p and q and setting a and b to the initial amplitude and 0, respectively, and then iteratively applying equation e. The output sample is then the real part of the result, and a and b are updated to the real and

[1]We are grateful to David Woodhouse (Cambridge University) for introducing us to this algorithm.

Pseudocode for the generation of a logarithmically-decaying cosine wave (equation 5):

```
p = cos(freq ∗ 1/samplerate) ∗ power(e, -1 ∗ damping.rate ∗ 1/samplerate);

q = sin(freq ∗ 1/samplerate) ∗ power(e, -1 ∗ damping.rate ∗ 1/samplerate);

a = initial.amplitude;

b = 0;


repeat for duration.in.seconds / samplerate:

        anew = a ∗ p - b ∗ q;

        bnew = b ∗ p + a ∗ q;

        a = anew;

        b = bnew;

        output = anew;

end repeat;
```

imaginary parts of the result, respectively (see pseudocode below). This means that computationally expensive sine and cosines need only be figured once for the calculation of p and q, and only four multipies, one add and one subtract are needed for each partial in a given sample. The efficiency of this implementation allows fairly complex impact sounds to be generated in realtime on many computers.

Synthesizing Patterns of Impacts

This algorithm allows us to synthesize a wide range of impact sounds varying over a number of source dimensions. Moreover, it can serve as a fundamental element in algorithms meant to synthesize more complex sounds.

For instance, we can return to Warren and Verbrugge's (1984) construction of bouncing and breaking sounds, and use the algorithm described above to synthesize them completely. To create bouncing sounds, we need simply imbed this algorithm in another which calls it at logarithmically decaying intervals. The shape of the perceived object can be varied by adding randomness to the overall temporal pattern (e.g., Sound Examples N - N). To create breaking sounds, this algorithm is imbedded in another which calls it with parameters specifying sources of different sizes at times corresponding to several logarithmically decaying time series (e.g., Sound Examples N - N).

Sound examples N - N: Synthesized bouncing (of regular and irregular objects) and breaking sounds.

Using the algorithm this way, it becomes clear that the sources we specify constrain the events we hear (as indicated above in the section on what we hear). For instance, if each impact in a temporal pattern typical of breaking specifies a wooden object, we are liable to hear a number of different wooden objects spilled on a surface rather than breaking. Similarly, if each of several objects have different material properties, we hear spilling rather than breaking (e.g., Sound Examples N - N).

Sound examples N - N: Synthesized breaking. When the material is inappropriate (e.g., wood), we hear spilling; as we do when several different materials are specified.

In sum, this algorithm allows us to synthesize impact sounds specifying a wide range of different materials, configurations, sizes, mallet hardnesses, and forces. Moreover, it can serve as the basis for more complex sounds such as those made by bouncing, breaking, and spilling. As such, it allows us to explore the space of such sounds to discover the constraints on perceived events placed by materials, and thus serves as a research tool. Perhaps most importantly, though, it is an efficient way to generate a family of sounds that might be applied in building auditory interfaces.

Filter-based Synthesis of Interacting Materials

The algorithm discussed above is specialized for impact sounds; its efficiency depends in part on the algorithm's integration of logarithmic decay with the generation of the partials. The algorithm discussed in this section separates the specification of the interaction from that of the object. It is thus less efficient than that above, but allows the synthesis of a much broader range of sounds.

The strategy used here is based on the observation that the resonant nodes of a given object are fixed despite the interaction that causes it to sound. Interactions serve to excite various of these resonant modes, and determine the temporal pattern of excitation and the amount of energy introduced to each mode. This can be seen from the two spectrograms shown in Figure X. The first shows a piece of wood first hit, and then scraped, the second a piece of glass hit and then scraped. Note that despite the very different temporal patterns of the sounds, in each case the resonant modes of each sound are the same,

Figure X. Two spectrograms of materials being hit and then scraped. The top shows the sounds made by a piece of wood; the second those of a piece of glass. Note that the resonant frequencies of the materials (indicated by long-lasting dark regions on the spectrogram) are the same despite different interactions.

Because the effects of interactions and objects are seperable, each can be synthesized separately. The resonant modes of a virtual object may be modelled as a bank of one-pole filters (which allow energy to pass at a single frequency). Interactions, then, can be specified in terms of the energy passed through the filter bank.

Modelling Objects as Filter Banks
A simple formula for a bank of one-pole filters is:

$$y_n = \sum_m \Phi_m (c1_m x_n + c2_m y_{n-1} - c3_m y_{n-2}) \qquad (6)$$

where Φ_m is an amplitude scalar for partial m, y_n is the nth output, x_n is the nth input, and:

$$c1_m = (1 - c3_m)\,[(1 - c2_m{}^2)/\,4c3_m\,]^{.5}$$

$$c2_m = (4c3_m \cos 2\pi f_m /S)/(c3_m + 1)$$

$$c3_m = e^{-2\pi b_m/S};$$

with f_m the peak frequency and b_m the peak bandwidth of partial m.

The same sorts of parameters can be used to control the peak frequencies and amplitudes of this sort of filter bank as are used to control the impact algorithm described in the last section. The pattern of peak frequencies reflects the object's configuration, while overall frequency indicates its size. The pattern of amplitudes Φ_m can indicate mallet or surface hardness (though see below).

Damping can be controlled via manipulations of the peak bandwidths of the filters. Bandwidth b is proportional to damping: the narrower the resonance peak of the filter the longer the resonant response to excitation. This corresponds well to the characteristics of resonating materials. As Gaver's (1988) empirical results and Wildes and Richards (19xx) analytic calculations show, highly damped materials such as wood have partials that are less well defined than those produced by less damped materials such as metal. Thus manipulating damping naturally -- and serendipitously -- involves changing the bandwidths of the partials as well.

Synthesizing sounds based on a filter-based model of the objects which produce them allows the same sort of control as the more specialized algorithm described in the last section. The size, material, configuration and surface hardness of the object may be controlled, and thus a wide range of virtual objects can be specified. But this method allows an even wider variety of sounds to be synthesized, because the simulation of the vibrating object can be combined with a number of virtual interactions.

Simulating Interactions with Input Waveforms

While the filter bank described in Equation 6 specifies the resonant modes of a virtual object, the waveform passed through the filters specifies the interaction which causes the object to vibrate. Thus this waveform describes the temporal pattern of forces applied to the object.

For instance, the pattern of force generated by an impact is characterized by a short impulse of noise such as one of those shown in Figure X. Note that these impulses are not point impulses, but rather have varying degrees of width and sharpness. Thus the energy they contain is spread out over many frequencies: low frequency energy contributes to a given pulse's width, and high frequency energy to its angularity. This corresponds to Freed's (xxx) characterization of mallet hardness. Hard mallets introduce force suddenly to an object, deforming it quickly, and thus introduce relatively large amounts of high frequency energy to the resonant object. Soft mallets, on the other hand, introduce energy relatively slowly (because they deform relatively quickly), and thus the corresponding impulses are characterized by a relatively high degree of low frequency energy.

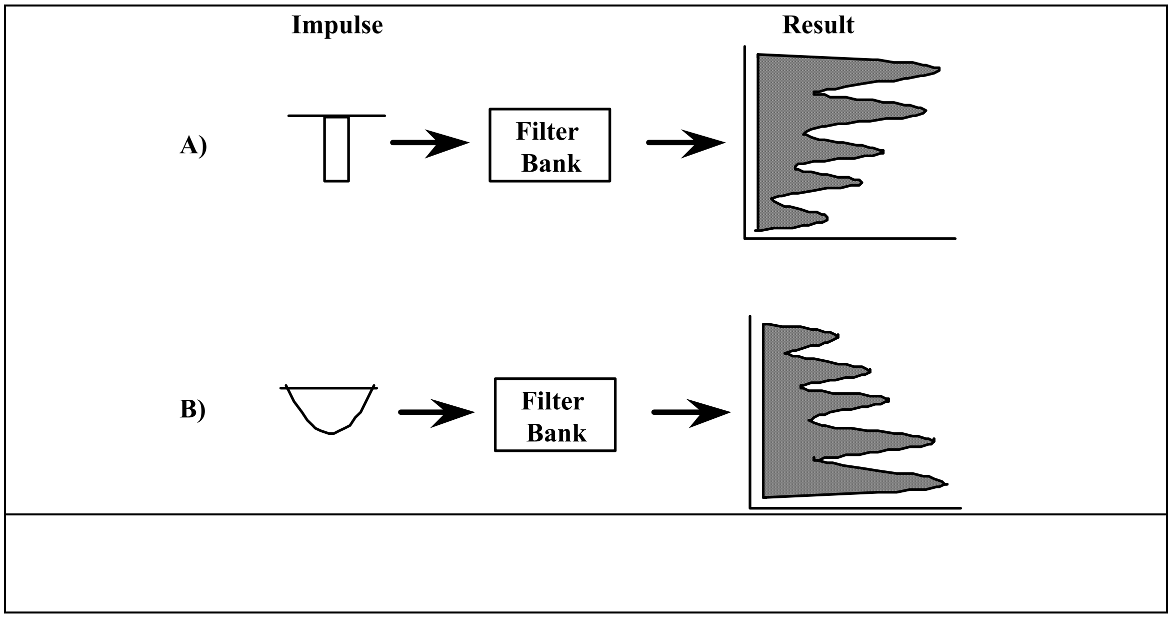


Figure X. Sample impulse waveforms characterizing different impacts.

The pattern of force on an object generated as it is scraped across a surface can be considered to a first approximation as a series of impacts. The density of the suface's texture and the speed of scraping is indicated by the rate of impacts; the speed is also likely to be indicated by the degree of high-frequency energy in the changes of force. This can be approximated by the sort of waveform shown in Figure X. This waveform is generated by sampled noise held for varying degrees of time, in this example new samples are held for increasingly shorter times. By speeding up the input of new energy in this way, the perceived speed of scraping is increased. The perceived texture and scraping speed might also be approximated by low-pass filtering the input waveform, thus rounding the transitions between new cycles.

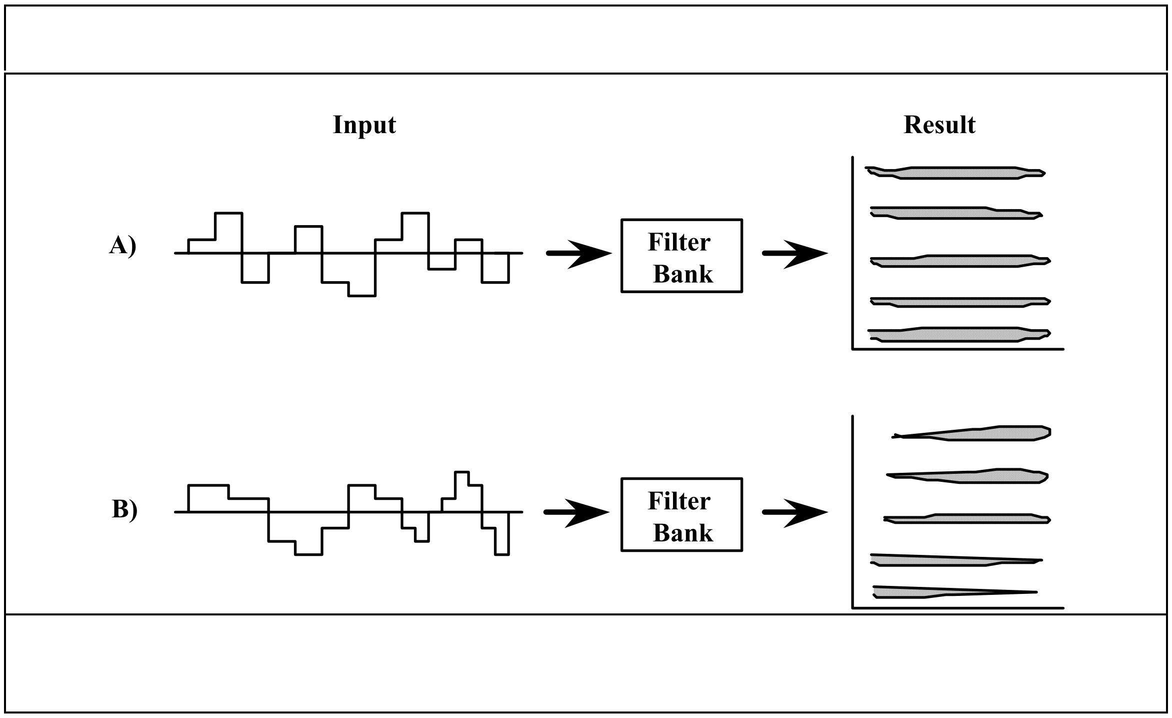


Figure X. Sample noise waveforms characterizing different scrapes.

There has been little research on the perception of scraping, and we know relatively little about the physical attributes that are important for or might be perceived on the basis of scraping sounds. Nonetheless, by exploring the range of input waveforms used in this synthesis model, a wide variety of scraping sounds can be produced (e.g., Sound Examples N - N).

Sound examples N - N: Synthesized sounds of a variety of different virtual objects being hit and then scraped. Note the effects on scraping noises made by various changes in the input waveform.

In sum, this algorithm is based on a physically plausible model of sound producing events. By separating the parts of the model that specify the object from those specifying the interaction, a wide range of virtual sound-producing events can be simulated. The model can create any of the impact sounds that the algorithm described in the last section can. In addition, it can also be used to create a variety of scraping sounds (and, potentially, rolling sounds). Perhaps most importantly, the model allows us to generate the sounds of the same object being caused to sound in different ways. This allows an integration of the events we simulate that promises to be quite useful in applications.

Clearly, the sort of synthesis strategies we present here are relatively primitive. Only the sounds made by simple events may be simulated, far fewer than might be captured by sampling. Nonetheless, there are a number of reasons, both theoretical and applied, that these algorithms are important. In serving as instantiated theories about everyday listening, they allow us to explore the characteristics of everyday sounds that we hear. More practically, these algorithms trade the burden of storage implied by using sampled sounds for the problem of generation. Insofar as they can be implemented efficiently, this tradeoff is a beneficial one. Perhaps most importantly both theoretically and practically, they allow us to explore *spaces* of sounds characterized by dimensions corresponding to those of sources. In conclusion, while we have only begun to develop useful synthesis algorithms of this sort, this sort of work is clearly important both for understanding everyday listening and for its application.

Soft to Hard to Soft: History Comes Full Circle

In the "good old days" of the 60's and 70's, nearly all sound synthesis involving computers was done using software synthesis. Programs were written that simulated the functions of signal processing modules. They were executed on general purpose computers. These modules generated digital samples that were stored on disk or digital tape. When all of the samples were computed, they were then fed to a digital-to-analogue (D/A) converter at a constant rate, usually about 32kHz.

The fluctuation of the magnitude of the samples was designed to correspond to the fluctuation of the acoustic pressure function associated with the sound desired. Consequently, when the output of the D/A converter was fed into an amplifier, the desired sound was audible. The "classic" programs of this genre were *Music IV* and *Music V* (Mathews, 1969).

These programs worked in what is known as "deferred" time, as opposed to "real" time. They often took on the order of 90 seconds to compute one second's worth of sound samples. Through the '70's and early

'80's, efforts in the area focussed on two main areas: the formulation of more computationally efficient algorithms, and the design of special purpose hardware that could do the sound processing in real time.

The combination of these two trends resulted in a number of commercially available synthesizers that incorporated special hardware executing sonically rich digital algorithms. Perhaps the best known of these is the Yamaha *DX7*, which incorporated a frequency modulation (FM) algorithm developed by John Chowning at Stanford University (Chowning, 1973; Chowning & Bristow, 1986). This was followed by a number of inexpensive synthesis and signal processing devices, including digital delay lines and reverberators. The Yamaha *SPX-1000* is a good example of the latter.

Software synthesis never died out, however. The reason involves the age-old tradeoff between strength and generality. While the specialized algorithms and hardware were powerful, they were also limited. What they did, they did well, but that was all that they did. On the other hand, if the software techniques were slow, they were also completely general. It might take a long time, and be hard to do, but any sound processing algorithm or technique could be implemented.

This is where the Cruise Missile comes in. Through the late '70's and early '80's, significant progress was made in developing specialized digital signal processing (DSP) chips. Developed largely for missile radar systems, these chips could execute more-or-less general signal processing algorithms in realtime.

This brings us up to the present. We are now at a stage where we have a choice of devices for hardware synthesis. We can opt for either the relatively inexpensive and easy to use specialized hardware, or the more expensive, harder to use, but more general programmable DSP chips.

At this stage, it is likely that the former is the better for starting off, especially for those new to signal processing. Polyphony, for example, is simply easier with the specialized hardware. However, this is an area changing rapidly, and the line between "hard" and "soft" signal processing is becoming blurred, as manufacturers are incorporating DSPs and increasing amounts of programming capability into their black boxes.

For those interested in exploring the software approach, perhaps one of the best and least expensive approaches is to use a board called *Turbosynth*[1] for the Apple Macintosh.

[1] By Digidesign,1360 Willow Road, Suite 101, Menlo Park, CA 94025. tel: (415)-327-8811.

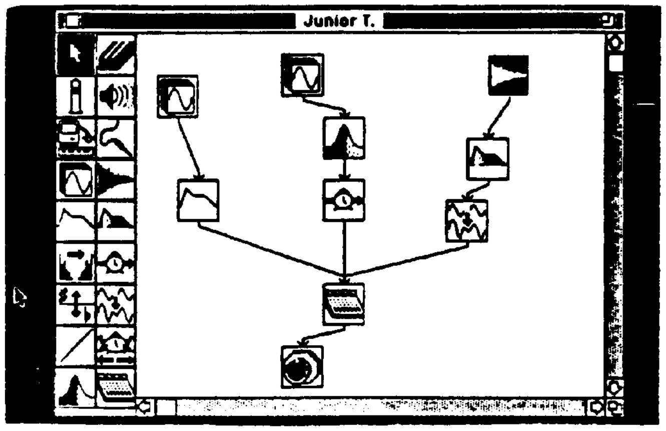


Figure 1: Software for the Digidesign Turbosynth Board for the Apple Macintosh

This board comes in versions for both the Macintosh SE and Mac II. It is one of the most usable of all DSPs. Its software, shown in Figure 1, is based on a variation of the original Music IV graphic notation.

Another company making a programmable DSP for audio is the AudioFrame[1]. Their unit is rather more "up-market" than the Turbosynth, but is representative of a new class of emerging hardware.

Outside the domain of music products, some labs have been putting together systems based on separate components. For instance, the McGill Speech and Hearing Lab[2] has put together a general purpose system for the analysis and synthesis of sounds, based on a general 386 microcomputer, a data acquisition board, a match co-processor, and some software.

6.4 Synthesis vs. Record/Playback (a.k.a. Sampling)

In the previous section, we dealt with one of the "big" tradeoffs in computer science, strength *vs*. generality. In this section, we address another: memory *vs*. cycles.

Nearly all of our discussion thus far has been concerned with how to cope with the computational overhead required to work with digital audio. However, we need go no further than our CD player to find an illustration that we can obtain professional quality digital audio without carrying a Cray computer around in our back pocket. In short, there is an alternative to computationally expensive synthesis and processing.

The alternative, as illustrated by the CD player, is to record the desired sounds digitally, save them in the system, and play them back on demand. This is exactly the technique used by Gaver's *SonicFinder.*

[1] By WaveFrame, 4725 Walnut St., Boulder, Colerado, USA 80301. tel: (303)-447-1572.

[2] Contact A. Bregman, Dept. of Psychology, Stewart Biological Sciences Building, 1205 Dr. Penfield Ave., Montreal, Quebec, Canada, H3A 1B1. tel: (514)-398-6103.

The assumption made in this approach is that memory is a more available resource than CPU cycles. The *SonicFinder* notwithstanding, this assumption may break down fairly quickly. Simply stated, despite decreasing memory costs, sound takes a lot of memory. Furthermore, the computational overhead in playing back samples is non-trivial, since doing so makes heavy demands on accuracy of timing. Consequently, despite the apparent simplicity of merely playing back samples, we are still likely to come up against a wall before we are able to achieve the sonic richness that we desire.

As with synthesis and processing, some of the problems with sampling can be addressed through the use of peripheral hardware. Commercially available samplers are used extensively in the music business, and are available from a number of manufacturers, including Casio, Roland, and Akai. Perhaps the most commonly used sampler is the S-900 by Akai. Its successor, the S-1000, is shown in Figure 2.

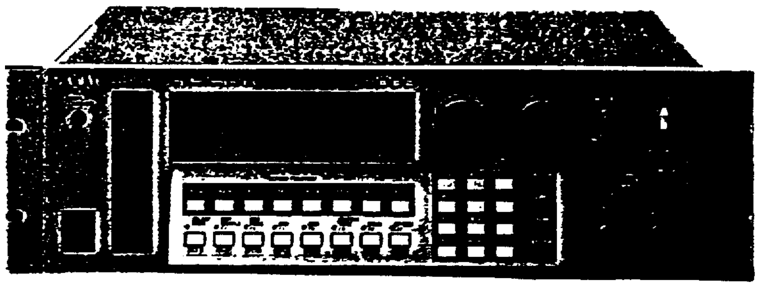


Figure 2: The Akai S-1000 Sampler

One problem with sampling, however, is getting the sounds that you want. In principle, you can sample anything that you can record with a microphone. But this is easier said than done. Getting good recordings takes immense care and technique. Fortunately, help comes from two directions: software and commercially available prerecorded samples.

If you are using samples and a commercially available sampler, then Digidesign's *Sound Designer* software is highly recommended. This helps you to "massage" your samples, as well as to organize them. It is the industry standard, available for virtually all commercial samplers, and runs on both Macintosh and Atari computers. Another program from the same company, *Softsynth*, computes samples from the envelopes of the desired sound's partials, as specified by the user. Figure 3 shows a Softsynth specification for a clarinet sound.

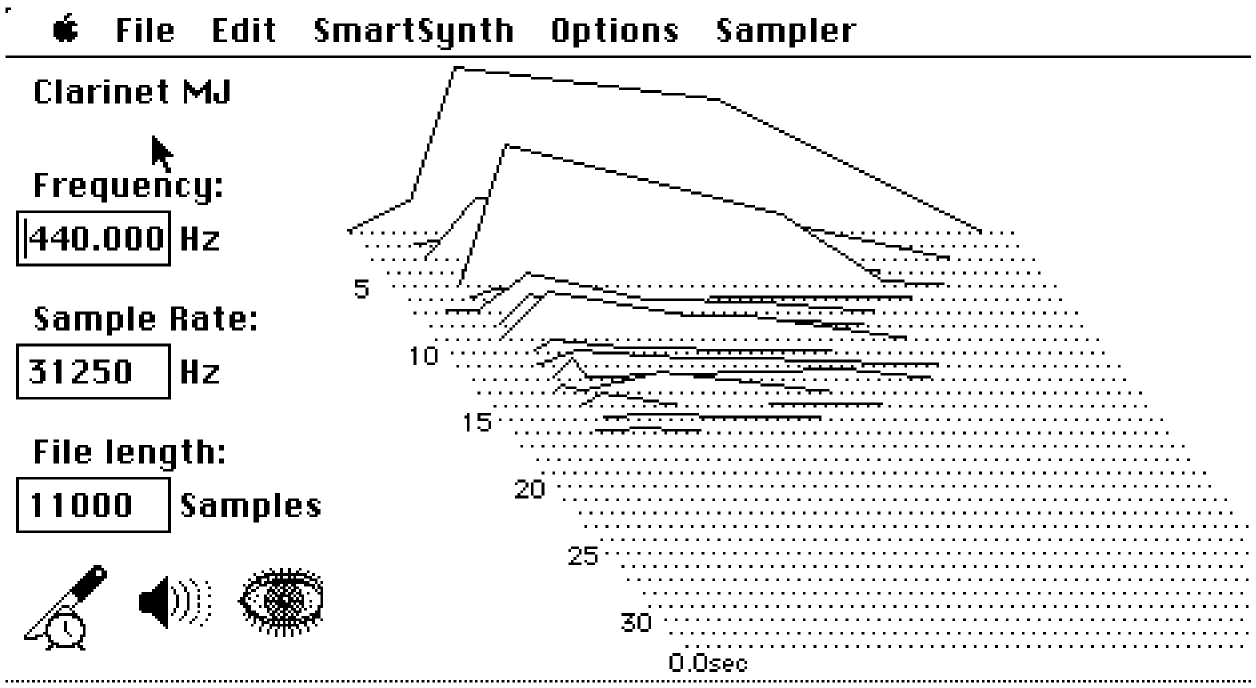


Figure 3: Digidesign Softsynth Specification for a Clarinet Sound.

Prerecorded samples are available from a number of sources. Magazines like *Keyboard* and *Sound Technology* are excellent sources of information. Sounds are available on floppy disks and on specially prepared CD-based libraries[1]. These include instrumental sounds, as well as extensive collections of sound effects.

But how useful are samples, and how well do they meet our original criteria for nonspeech audio sources? On the one hand, the *SonicFinder* and many video games demonstrate that they can be used to good effect. However, in our opinion, their use - while valuable - has distinct limitations. The use of samples is largely tantamount to playing back sound effects on demand. Since the sounds are prerecorded, we have few, if any, hooks into their internal structure. Consequently, the realtime control that we can exercise over them is limited to transposition in pitch, filtering, loudness, envelope and stretching in time (looping). Whereas synthetic sounds may be harder to generate, they lend themselves to far more subtle dynamic control.

On the positive side, sampling provides access to sounds derived from nature. Therefore, it is very valuable in exploring the use of everyday listening and sounds.

[1] Sound Ideas, 105 West beaver Creek Road, Suite 4, Richmond Hill, Ontario, Canada L4B 1C6. Tel: Canada: (416)-977-0512 / US: (800)-387-3030. Fax: (416)-886-6800.

Consequently, what do we do in our lab? We use a combination of sampling and synthesis techniques, both using external hardware. However, it should be kept in mind that the decisions we have made in a research environment are likely to be different than those we would have made if we had been developing a product.

Sound in Space

There are a few comercial systems available that control sound in space. One that is intended for use with loudspeakers is the SP-1 Spatial Sound Processor from Spatial Sound, Inc.[1] Another recent product is the Roland *Sound Space* processing system (RSS).[2]

There are at least two systems specifically designed for controlling sound in space using headphones equipped with position and orientation sensors. One is the *convolvatron* (Wenzel, Wightman & Foster, 1988a&b). This is a stand-alone unit. A comparable device designed to run on a build-in signal-processing board on the Macintosh II family is also available.[3]

A Word About Control: MIDI

We have discussed how we can off-load computationally expensive processes to external hardware. However, most of our experience with computers tells us that this tends to be expensive, idiosyncratic "one-off" type of work, and difficult to do from both the hardware and software perspectives.

Luckily, there has been a most important development. In 1982, the music industry agreed upon a guideline for a means of interconnecting computers and audio equipment called the *Musical Instrument Digital Interface*, or *MIDI* (IMA, 1983; Loy, 1985). MIDI is a medium speed serial interface for communicating *control* information (such as keyboard events, button pushes, and fader motion) among digital devices.

MIDI specifies the mechanical, electrical, and logical interface for interconnecting equipment. It has been incorporated into literally millions of units. It works, it is widely available, and it is inexpensive and supported.

Researchers can easily acquire samplers, synthesizers, signal processors, and mixers which are MIDI controllable. MIDI interfaces are available for virtually all microprocessors. For other computers, such as Suns, adaptors are available to convert between MIDI and RS-232 ports.[4]

---

[1] Spatial Sound, Inc., 743 Center Boulevard, Fairfax, CA 94930 (USA). Tel: (415)-457-8114. Fax: (415)-457-6250.

[2] Roland Corp, US., 7200 Dominion Circle, Los Angeles, CA, USA 90040-3647. Tel: (213)-685-5141.

[3] Gehring Research Corp., 189 Madison Ave., Toronto, Ontario, Canada M5R 2S6. Tel/Fax: 416-963-9188.

[4] Hinton Instruments, 168 Abingdon Road, Oxford 0X1 4RA, United Kingdom.

MIDI is by no means perfect. However, it is usable and useful. One problem is that virtually all commercially available MIDI software is "closed;" that is, it is almost impossible to tailor applications. Consequently, it is often necessary to start from scratch in building software tools to support research. One exception, however, is a package distributed on a cost-recovery basis by Roger Dannenberg of Carnegie Mellon University.[1] This is a set of application-callable library routines for the Macintosh and IBM-PC computers.

For anyone planning to work with MIDI, we highly recommend joining the International MIDI Association (IMA)[2] which will, among other things, provide you with a copy of the MIDI specification.

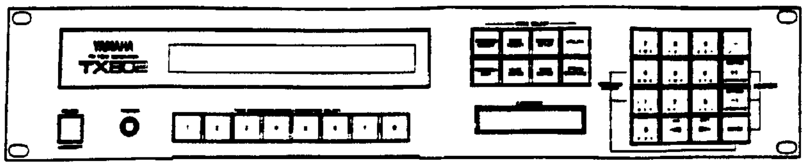
In the remainder of this chapter, we will discuss various MIDI modules, including synthesizers, and conclude with a brief overview of the MIDI protocol and its use.

Synthesis Modules

Generally, when one thinks of a synthesizer, one thinks of something that looks much like a high-tech organ. It has a piano-like keyboard, and a bunch of knobs, dials, and buttons. Whereas that image is still partially accurate, MIDI has changed things somewhat.

One thing that MIDI has done is to separate the *control* part of musical instruments from the *synthesis* part. Before MIDI, an instrument's controller (typically a keyboard) had to be tightly coupled with the synthesis stage. Now, the synthesis unit needs only to understand MIDI commands. It need not have any idea whether they come from a keyboard, computer, or some other control device.

Consequently, manufacturers now produce a wide range of synthesizer modules that have no keyboard, or any other controller. Typically, these units come in standard 19 1/2" rack-mounting units.



Yamaha's TX-802. The DX7 Mk II is essentially the same unit, but with an integrated keyboard.

[1] For details write: Prof. Roger B. Dannenberg, Center for Art and Technology, Carnegie Mellon University, Pittsburgh, PA 15213-3890, USA.
[2] IMA, 11857 Hartsook St., North Hollywood, CA, 91607

In looking for an appropriate synthesizer, there are a number of features to keep in mind:

- multi-timbre
- timbral range
- ease of control
- ease of programming
- size
- cost

While there are a number of synthesizers on the market, few meet the combined criteria of being poly-timbral, flexible, rich in sounds, small, and inexpensive.

One can question whether any of these devices is actually easy or intuitive to use. *Voicing programs* and *librarians*, discussed later under the heading of *Auxiliary MIDI Software*, provide some help. However, as with samplers, one way to circumvent the problem of "rolling your own" sounds, is to acquire them ready made. It is almost always easier and more reasonable to modify someone else's sound to your needs than build your own from scratch.

Additional sounds are available for most synthesizers. Music stores, or magazines like *Keyboard* and *Sound Technology* are good sources. The thing to keep in mind, however, is that the quantity and quality of sounds available depends largely on the popularity of the synthesizer in question. In fact, this holds true for most forms of secondary support, such as software tools.

Consequently, the number of candidate modules is more limited. As a start, we recommend that you look at two families of devices, one from Yamaha, the other from Roland.

The Yamaha devices are the TX-802, already mentioned, and its smaller companion, the FB01.

The FB01 is interesting due to its rich sound repertoire, small size, and very low price. Board-level versions of it are available that fit into the backplane of IBM-PC machines[1]. Both devices use Chowning's FM synthesis algorithm . The TX-802 has better fidelity and a more complex implementation.

The Roland units we recommend are the D-110 and its smaller companion, the MT-32. Like the FB01, the latter is a smaller, less expensive version of its companion, whose only real shortcoming is somewhat

[1] The IBM Music Feature Card. EM Bookshelf, 6400 Hollis St. #12, Emeryville, CA 94608. tel: (415)-653-3307.

lower sound quality. Both Roland units use another form of algorithm, known as LA synthesis, which is extremely rich and useful.

Signal Processing Modules

Regardless of whether sounds are synthesized or sampled, derive from the host or a peripheral, or are software or hardware generated, good sounds alone are not enough.

In many cases, the *ambience* of a sound is as important, if not more so, than the identity of the sound itself. This brings us into the world of MIDI controllable audio signal processors and effects. Perhaps the most important of these is the digital reverberator.

The past few years have seen a dramatic reduction in the price of mid to high quality digital reverberators. There is a wide choice from a number of manufacturers, including *Alesis, Roland,* and *Yamaha.*

Many of the newer effects modules are not limited to one function, such as reverberation. The new generation of units typically incorporate an internal DSP that can perform filtering, delay, echo, tremolo, transposition, as well as reverberation. In fact, some units, such as from *Digitech* and *Roland*, can perform more than one function simultaneously.

One class of peripheral that warrants investigation is known as a "pychoacoustic enhancer." This is a "mystery" black box that is claimed to give sounds more "presence." Be that what it may, it is clear that some degree of timbral processing is useful, along with reverb and loudness control, in establishing figure-ground relationships among sounds. What we mean by this is that we want to be able move sounds along a continuum from background to foreground *without necessarily using loudness as the primary cue*. Enhancers may help here, but are as yet unproven. However, dynamic filters, especially parametric equalizers (such as the Yamaha *DEQ7*) are very useful in this regard.

No matter how sounds are synthesized or processed, they have to be heard and so their levels have to be adjusted. The *audio mixer* is the device normally used to do this. Two particular devices, the Yamaha *DMP7* and *DMP11*, are especially attractive. They are both fully MIDI controllable and they have a significant amount of signal processing effects built in, including delays, reverb and parametric equalization Consequently, they provide an extremely convenient package to support research.

Auxiliary Audio Devices

An audio system needs speakers, amplifiers, cables, etc. to function. While most of this is obvious, there are a few comments worth making. Perhaps the most important is, *keep it simple.*

While it is likely that the sound synthesis and processing elements can be hidden from the end user, the final stages of the audio chain are normally far more evident. For speakers, we strongly recommend using self-powered models, such as those designed to be used with personal stereos. These are available from a number of manufacturers. We particularly like the price and performance of the Warfdale *Active Diamond* self-powered speaker. Other speakers are available from Tandy/Radio Shack.

In many environments, headphones are required. In such cases, remember that headphones are often as useful for keeping external sounds out, as for keeping sounds in. That is, besides keeping sounds from disturbing neighbours, headphones can keep neighbours' sounds from disturbing us. Consequently, seriously consider headphones which fully cover the ear.

Auxiliary MIDI Devices

There are a few MIDI peripherals that one should be aware of and perhaps consider acquiring when equipping a lab. The first might be a MIDI piano-like keyboard. While a keyboard sis not likely to be a part of the end-user's system, it can be very useful for triggering sounds during the design and debugging phases.

Another device which is almost a must if you have more than two or three MIDI devices is a *MIDI Patching Matrix.* This allows the routing of MIDI data to be controlled. To understand the need for such a device, one has to understand a basic shortcoming of MIDI: it is not a bus nor is it a LAN.

A MIDI chain can only have one "master," or source. Everything else is a slave, or sink. For example, if I want the computer to record the state of a signal processor, then the signal processor must be the master, and the computer the slave. If I then want to test the recording of the data by having the computer reset the signal processor, I must first change the MIDI control chain to make the computer the master and the signal processor the slave.

Very quickly, we get in a total muddle, buried in a tangle of MIDI cables, not knowing what is connected to what. The simple solution is to have the input and output of each MIDI device connected to a programmable switching matrix. That way, any input can be routed to any single output, or combination of outputs. Such MIDI patching matrices are available from a number of companies. Inexpensive but useful units are available from J.L Cooper, Roland, Akai, 360 Systems, and Digital Music Corp. Beware, however, of some products, such as from Yamaha, which (unbelievably) don't permit the MIDI patch matrix to be controlled by MIDI! Check this, since you want to be able to control the entire system from your computer.

The final device that you may end up needing is what is known as a *MIDI Merger.* As its name suggests, this merges MIDI streams from two different sources. This is another way around the "single master" problem discussed above. Several MIDI patch matrices have merge capability built in, and this is definitely worth looking for. Alternatively, *Pocket Products* has available an extremely inexpensive stand-alone merger.

Auxiliary MIDI Software

There are two classes of software that are worth investigating if one is setting up a MIDI-based sound facility. The first, *Voicing Software,* provides a more civilized user interface to many of MIDI modules. The second, *Librarian Software,* helps keep track of all of one's sounds.

One of the first things that one notices when beginning to work with MIDI devices is how incredibly moded and difficult the user interfaces are. This is especially true with rack-mount equipment.

Manufacturing prices can be cut by reducing the number of controls on devices. Hence, almost every box, regardless of purpose or number of controllable parameters, seems to have only 2 buttons, one knob, and a 16 character display. Everything must be controlled with these devices, usually by stepping through a seemingly infinite number of modes until the controls are "attached" to the desired parameter.

Fortunately, MIDI can provide a partial solution. Using *system exclusive* messages, one can use an external computer to gain access to most of a modules functions. Consequently, a side industry has grown up which provides computer-based software interfaces to these modules. Voicing programs run on most personal computers and are available for most popular synthesizers and some signal processing units. In the absence of

commercially available software, one can build one's own. This is especially easy on the Apple Macintosh using the MIDI extensions to *Hypercard*, available on most public bulletin boards.

A User's Guide to the MIDI Commands

Using MIDI commands is much simpler than it appears at first glance. Basically, a MIDI command, just like any other computer command, is a series of symbols that have specific meanings and which depend on a specific syntax. The only difference between sending a MIDI command and a command like "list files" is that MIDI commands are usually expressed in hexadecimal format. So, for example, to tell a synthesizer that is receiving on channel 1 to start a note at middle C with a medium key down velocity, one sends **90 3C 40**. This may look a bit obscure at this point, but the purpose of this guide is to make MIDI make sense.

Message Syntax

The syntax for any MIDI message is simply: status byte, data byte(s). The status byte is like the command name, and the data bytes are like arguments. In the *note on* command above, "90" is the status byte, and **3C** and **40** are data bytes.

There are two basic kinds of commands. The first are called *channel voice messages*, the second are *system messages*. I discuss each in turn below.

Channel Voice Messages

Channel voice messages are control messages used to play and make changes to sounds. For instance, a *note on* message is a channel voice message, as are *note off*, *pitch bend*, and *control change*.

Why the "channel" in "channel voice messages"? Because these commands are sent to specific MIDI channels. A channel can be thought of as an address for the message. To see why this is necessary, consider a system with 16 synthesizers and effects units all hooked up to a computer's MIDI output. If a *note on* message was received by all the synthesizers every time it was sent, the system would be next to useless. But by assigning each unit to a different channel, and sending *note on* messages to specific channels, each synthesizer can indeed be controlled independently.

The channel number that is being addressed is sent on the low four bits of the status byte of channel voice messages. So the **0** in **90** means that this message is directed at channel 1 (note that though we refer to channels 1 - 16, channels are addressed from 0 - 15.) To send a *note on* message to the synthesizer on channel 5, the status byte would be **94** instead of **90**.

This example can help illustrate why MIDI messages are often expressed in hexadecimal format. The units of MIDI messages are bytes, but often two kinds of information are sent in one byte. The high four bits encode one message, the low four another. Four bits can range in value between 0 and 15, and one hexadecimal digit also ranges from 0 (0 in hex) to 15 (F in hex). Thus it is often clearest to use a two digit hexadecimal number to stand for the 8 bits; the first hex digit standing for the high four bits, and the second the low four. For instance, consider how to send a *note on* message to the first synthesizer in our hypothetical system, and then one to the fifteenth. In hex the status bytes of these messages would be **90** and **9F**. In decimal, they would be **144** and **159**. In the hex version, the note on command **9-** and the channel number **-F** are both obvious. In the decimal form, both are obscured.

Following the status byte are one or more data bytes, which act as arguments to the status byte's command. For instance, in the *note on* command **90 3C 40**, **3C** and **40** are data bytes. The first one, 3C, says which note to turn on (in this case, a middle C), and the second, 40, says how hard to press it. The meaning and number of data bytes varies with the command being sent; see the MIDI Summary Sheet following for the syntax of the commands.

One of the more flexible (and perhaps confusing) channel voice messages is the *control change* message. This message allows you to pass information to any one of 32 continuous controllers, 32 on/off switches, or to set the mode of a particular synthesizer (see below). These messages consist of a status byte, **Bn**, where n is the channel number; then the first data byte, the control number, which specifies the controller; and finally a data byte which specifies the new controller value. Officially, the assignments of controllers to control numbers is somewhat vague; unofficially a number of devices have been assigned to certain control numbers by common use. So, for instance, the message used to turn the sustain pedal on is: **Bn 40 7F** (40 hex = 64 decimal; 7F hex = 127 decimal).

One of the control changes that can be sent is a *channel mode message*. This message controls how the receiving device will respond to following MIDI messages. There are four MIDI modes:

- **Mode 1: Omni on/Poly**. *Omni on* means that the device will respond to messages sent on *any channel*. This is useful in some situations, particularly those in which one synthesizer is driving another. But it is probably not a useful mode for using a computer to exert independent control on a number of different devices. *Poly* means that the synthesizer will perform polyphonically using all its voices.

- **Mode 2: Omni on/Mono**. Same as above, but only one voice will be used. This is probably not the mode to use if you have a polyphonic synthesizer.

- **Mode 3: Omni off/Poly**. *Omni off* means that the device will only respond to messages sent on the appropriate channel(s); Poly means all its voices can be used, but which voice will play when is a controlled by the synthesizer's internal assignment algorithm, not you.

- **Mode 4: Omni off/Mono**. This is probably the most useful mode, in which each voice on each device can be controlled by a separate channel (up to the 16 possible). *Mono* in this context doesn't mean that you only can use one voice per synthesizer, but one voice per channel. Thus in this mode you can assign a different channel to each voice, or each device, and control each separately.

System Messages

System messages are unlike channel voice messages in that they are not sent to a specific channel and thus do not include the channel number in the low four bits of the status byte. Instead, their high four bits are always set, so that all system messages have the status byte **Fn**, where n codes the particular message.

System messages are divided into three categories: *System Common*, *System Real Time*, and *System Exclusive*. System common messages are used to select songs and positions for sequencers, and to command analog synthesizers to start tuning to their automatic oscillator if they have one. System real time messages transmit codes that are important for sequencers.

The most important system messages are *system exclusives*. These allow manufacturers to define their own commands to which their equipment will respond. The first data byte is the ID of the manufacturer, then a variable number of other data bytes follow which depend on the command. System exclusive messages are *always* ended with the system common message **F7**.

For instance, let's say a company makes a MIDI controlled effects unit with reverberation. Their manufacturer number might be 1B (hex), and they might define the data byte A2 (hex) as the command to change reverberation time, and the following byte as the new level. Then to set the reverberation time to be very short, the user could send the message: **F0 1B A2 03 F7**.

There is little more to say about system exclusive messages, because they are left to the manufacturer to define. There is no doubt, however, that these are the most powerful features of MIDI.

Summary

MIDI is a fairly powerful control protocol for instruments, and its command language is thus somewhat complex. As we hope to have demonstrated in this section, however, it is not terribly difficult to learn, once a few of the basic principles have been indicated. Following this section is a MIDI summary sheet which briefly describes the basic MIDI commands. With the exception of system exclusive commands, this sheet should be all that is needed to start using MIDI commands.

MIDI Code Summary Sheet

I. Channel Voice Messages -- "n" in status byte = channel number (0 - F)

Command	Status Byte	Data Bytes	Comments
Note Off	8n	1) Key Number	0 - FF (3C = middle C)
	2) Release Velocity		0 - FF
Note On 9n	1) Key Number	if velocity = 0 then same as	2) Key Ve
Polyphonic Key	An	1) Key Number	usually channel pressure is used
Pressure	2) Key Pressure	(see Dn, below)	
Control Change	Bn	1) Control Number	see below for official and 2) C

<u>Official Assignments:</u>

0 - 31	continuous controller 0 - 31 (MSB's)		
32 - 63	" " " (LSB's)		
64 - 95	on/off switches (0 = off, 127 = on)		
96 - 121	undefined		
122 - 127	channel mode message:		
	122, 0	local control off	
	122, 127	local control on	
	123, 0	all notes off	
	124, 0	omni off	
	125, 0	omni on	
	126, no. channels		mono mode on
	127, 0	poly mode on	

<u>Common assignments:</u>

1	modulation wheel

	2	breath controller	
	4	foot controller	
	5	portamento time	
	6	data entry knob	
	7	main volume	
	64	sustain pedal	
	65	portamento	
	66	sostenuto	
	67	soft pedal	
	96	data entry increment	
	97	data entry decrement	
Program Change	Cn	1) Program number	
Channel Pressure	Dn	1) Channel pressure	Compare with An, above
Pitch Bender	En	1) Pitch Bend (LSB)	
	2) Pitch Bend (MSB)		

II. System Messages

Command	**Status Byte**	**Data Bytes**	**Comments**
A) System Exclusive			
Enter System Exclusive	F0	Manufacturer ID	System exclusive commands control data
B) System Common			
undefined	F1	–	reserved for later assignment
Song Position Pointer	F2	1) Song Position (LSB)	
	2) Song Postion (MSB)		
Song Select	F3	1) Song Number	
undefined	F4, F5	–	
Tune Request	F6	none	for tuning analog synths
End Of Exclusive	F7	none	signals end of exclusive command
C) Real Time			
Timing Clock	F8	none	
undefined	F9	none	
Start FA	none	starts sequencer from beginning	

Continue	FB	none	starts from last stop or position
Stop	FC	none	stops sequencer
undefined	FD	none	
Active Sensing	FE	none	produced so receiver knows
System Reset	FF	none	resets system to the condition of just having been switched on (caution!)

References

1. Chowning, J. (1973). The synthesis of complex audio spectra by means of frequency modulation. Journal of the Audio Engineering Society. 21, 526-534.

2. Draper, S., Waite, K., & Gray, P. (1991). Alternative bases for comprehensibility and competition for expression in an icon generation tool. Proceedings of Interact'90 (Cambridge, UK. 27 - 31 August, 1990). Amsterdam: North Holland..

3. Freed, D. J., & Martens, W. L. (1986). Deriving psychophysical relations for timbre. Proceedings of the International Computer Music Conference (Oct. 20 - 24, 1986, The Hague, The Netherlands).

4. Gaver, W. (1988). Everyday listening and auditory icons. Doctoral Dissertation, University of California, San Diego.

5. Gaver, W. (1989). The SonicFinder: An interface that uses auditory icons. *Human-Computer Interaction.* 4 (1).

6. Gaver, W. (1993). What in the world do we hear? An ecological approach to auditory source perception. *Ecological Psychology* (5) 1.

7. Gaver, W., Smith, R. B., & O'Shea, T. (1991). Effective sounds in complex systems: The ARKola simulation. *Proceedings of CHI 1991,* New Orleans, April 28 - May 2, 1991, ACM, New York.

8. Lamb, H. (1960). *The dynamical theory of sound*. 2nd ed. New York, Dover.

9. Risset, J. C., & Wessel, D. L. (1982). Exploration of timbre by analysis and synthesis. In D. Deutsch (Ed.), The psychology of music. New York: Academic Press.

10. Smith, R. B. (1989). A prototype futuristic technology for distance education. *Proceedings of the NATO Advanced Workshop on New Directions in Educational Technology.* (Nov. 10 - 13, 1988, Cranfield, UK.)

11. Warren, W. H., & Verbrugge, R. R. (1984). Auditory perception of breaking and bouncing events: A case study in ecological acoustics. *Journal of Experimental Psychology: Human Perception and Performance*. 10, 704 - 712.

12. Wildes, R., & Richards, W. (1988). Recovering material properties from sound. Richards, W. (ed.), *Natural computation*. Cambridge, MA: MIT Press.