Introduction to Computer Music

Week 12

Instructor: Prof. Roger B. Dannenberg

Topics Discussed: Additive Synthesis, Table-Lookup Synthesis, Spectral Interpolation Synthesis, Algorithmic Control of Signal Processing, 3-D Sound, Head-Related Transfer Functions, Multi-Speaker Playback

Chapter 12

Spectral Modeling, Algorithmic Control, 3-D Sound

Topics Discussed: Additive Synthesis, Table-Lookup Synthesis, Spectral Interpolation Synthesis, Algorithmic Control of Signal Processing, 3-D Sound, Head-Related Transfer Functions, Multi-Speaker Playback

Previously, we considered *physical models* which simulate physical systems from which we get vibration and sound. In contrast, we can think about sound in more perceptual terms: What do we hear? And how can we create sounds that contain acoustic stimuli to create desired impressions on the listener? Your first reaction might be that our hearing is so good, the only way to create the impression of a particular sound is to actually make that exact sound. But we know, for example, that we are not very sensitive to phase. We can scramble the phase in a signal and barely notice the difference. This leads to the idea that we can recreate the *magnitude spectrum* of a desired sound, perhaps without even knowing how the sound was created or knowing much about the details of the sound in the time domain.

This approach is sometimes called *spectral modeling* and it is a powerful alternative to *physical modeling*. While *physical modeling* requires us to understand something about the sound *source*, *spectral modeling* requires us to understand the sound *content*. We can often just measure and manipulate the spectra of desired sounds, giving us an approach that is simple to implement and relatively easy to control. In this section, we consider variations on spectral modeling, from additive synthesis to spectral interpolation.

12.1 Additive Synthesis and Table-Lookup Synthesis

One of the earliest synthesis methods drew upon the intuition that if all sounds could be represented as sums of sine waves, why not simply add sine waves to create the desired sound? This *additive synthesis* approach is not the most efficient one, and another intuition, that musical tones are mostly periodic, led to the idea of constructing just one period and repeating it to create tones.

12.1.1 Additive Synthesis

Additive synthesis usually means the summation of sinusoids to create complex sounds. In this approach, every partial has independent frequency and amplitude, giving unlimited freedom (according to what we know about Fourier analysis and synthesis) to create any sound. Earlier, we studied analysis/synthesis systems such as McAuley-Quatieri (MQ) and Spectral Modeling Synthesis (SMS) showing that analysis and synthesis are possible. We also learned that parametric control is somewhat limited. For example, time-stretching is possible, but more musical control parameters such as articulation, formant frequencies and vibrato do not map directly to any additive synthesis parameters.

12.1.2 Table-Lookup Synthesis

If we are willing to limit sinusoidal (partial) frequencies to harmonics and work with a fixed waveform shape, then we can save a lot of computation by creating a table containing one period of the periodic waveform. Usually, the table is oversampled (the sample rate is much higher than twice that of the highest harmonic) so that inexpensive linear interpolation can be used to "read" the table at different rates. By stepping through the table by a variable increment and performing linear interpolation between neighboring samples, we can get frequency control. A simple multiplication provides amplitude control.

Here is a software implementation of a table-lookup oscillator, building on the algorithm introduced in Section **??**. This code computes `BLOCK_SIZE` samples every time `osc` is called. In a real implementation, we would keep the `table` and current `phase` variables in a structure or object so that we could avoid these as global variables and have more than one oscillator instance. This code does not implement amplitude control, so the output should be multiplied by an amplitude envelope separately.

```
float table[513] =  ... some waveform ... ;
double phase = 0.0;

void osc(double hz, float table[], float out[]) {
    double incr = hz * 512 / sample_rate;
    for (int i = 0; i < BLOCK_SIZE; i++) {
        int iphase = floor(phase);
        double x1 = table[iphase];
        out[i] = x1 + (phase - iphase) *
                        (table[iphase+1] - x1);
        phase += incr;
        if (phase >= 512) phase = phase - 512;
    }
}
```

One trick in this code is that `table` should be initialized to a 512-point waveform, with the first value duplicated as the 513^{th} entry. By duplicating the first sample at the end, we insure that the expression `table[iphase + 1]` (used to interpolate between two samples in the table) never accesses a value outside the bounds of the array, particularly when the `phase` exceeds 511 but has not "wrapped around" to zero.

Table-lookup oscillators have two basic parameters: frequency and amplitude, and otherwise they are limited to a fixed wave shape, but often table-lookup oscillators can be combined with filters or other effects for additional control and spectral variation.

12.2 Spectral Interpolation Synthesis

One simple idea to introduce spectral variation into table-lookup oscillators is to use *two* tables and interpolate between them, as shown in Figure 12.1. With no constraints, this could lead to phase cancellation and unpredictable results, but if the phases of harmonics are identical in both tables, and if the tables are accessed at the same offset (i.e. index or phase), then when we interpolate between tables, we are also interpolating between the magnitude spectra of the two tables. This is interesting! With interpolation between tables, we can create *any* smoothly varying harmonic spectrum.

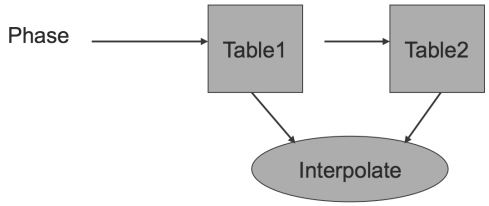

Figure 12.1: Basic Spectral Interpolation. Phase is incremented after each sample to scan through tables as in a simple table-lookup oscillator, but here, the phase is used for two table lookups, and the output is an interpolation between Table1 and Table2.

There are some commercial synthesizers that use a 2-D joystick to control real-time interpolation between 4 spectra. This can be an interesting control strategy and is related to using filters to modify spectra—in both cases, you get a low dimensional (1-D cutoff frequency or 2-D interpolation) control space with which to vary the spectrum.

12.2.1 Time-Varying Spectrum

Perhaps more interesting (or is it just because I invented this?) is the idea that the spectrum can evolve arbitrarily over time. For example, consider a trumpet tone, which typically follows an envelope of getting louder, then softer. As the tone gets softer, it gets less bright because higher harmonics are reduced disproportionately when the amplitude decreases. If we could record a sequence of spectra, say one spectrum every 100 ms or so, then we could capture the spectral variation of the trumpet sound. The storage is low (perhaps 20 harmonic amplitudes per table, and 10 tables per second, so that is only 200 samples per second). The computation is also low: Consider that full additive synthesis would require 20 sine oscillators at the sample rate for 20 harmonics. With spectral interpolation, the cost is only 2 table lookups per sample. We also need to compute tables from stored harmonic amplitudes, but tables are small, so the cost is not high. Overall, we can expect spectral interpolation to run 5 to 10 times faster than additive synthesis, or only a few times slower than a basic table-lookup oscillator.

Of course, if we just want to reproduce recorded instrument tones, maybe sampling is a better approach because it captures inharmonic attack transients very well. While the storage cost is higher, the computational cost of sampling (mostly due to sample-rate conversion to control frequency) is in the same range as spectral interpolation synthesis. But with sampling, we are stuck with a particular recording of every tone and we have little control over it. With spectral interpolation, we have the opportunity to *compute* the evolving spectrum.

12.2.2 Use Pitch and Amplitude to Compute Spectra

One successful approach that is especially good for wind instrument simulation is to model the spectrum as a function of amplitude and frequency. We carefully record instruments playing crescendos (increasing loudness) at different pitches. Then we analyze these tones to get a matrix of spectra at different pitch and loudness levels. As shown in Figure 12.2, we input desired frequency and amplitude, do a lookup in our 2-D matrix to find the appropriate next spectrum, then use spectral interpolation to smoothly change to that spectrum. As long as amplitude and frequency are slowly changing, as in vibrato and amplitude envelopes, the output will realistically change spectrum just like the real instrument. In fact, when amplitude is changing rapidly, we cannot hear the spectral changes very well. In our model (Figure 12.2), we normalize all the spectra in the 2D matrix so everything comes out of the spectral interpolation oscillator at about the same level, and we multiply that by the amplitude envelope. This captures the rapidly varying amplitude envelope faithfully, and at least approximately does the right thing with the spectrum, even though spectral changes may lag behind or be smoothed out a little.

This basic approach is used in the Dannenberg Brass software for Native Instruments Reaktor synthesizer. (www.native-instruments.com/en/reaktor-community/reaktor-user-library/entry/show/13607/). This implementation credits this author but was developed without any direct collaboration.

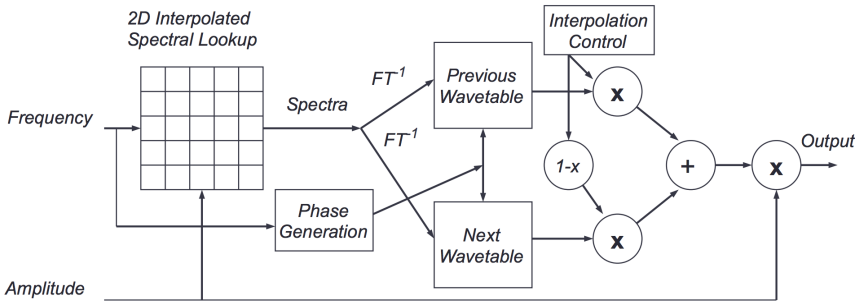

Figure 12.2: Computation for Spectral Interpolation Synthesis. Frequency and amplitude are input controls. The spectrum is varied by mapping the current frequency and amplitude to a spectrum stored in a matrix. This may be an interpolated lookup using the 4 nearest data points (spectra) to yield smooth spectral variations as a function of the continuous frequency and amplitude signals. There is also interpolation through time from one spectrum to the next, using a two-table-lookup oscillator. Since amplitude variations may need to be quite rapid, the amplitude scaling takes place at audio rates just before the output.

12.2.3 Dealing with Rapid and Inharmonic Attacks

The "pure" spectral interpolation synthesis model works well for some instruments, but others, particularly brass and saxophones,[1] have distinctive attack sounds that are too inharmonic and noisy to recreate from harmonic waveforms. The solution is to combine sampling with spectral interpolation, using short sampled attacks of about 30 ms duration. It is tricky to join the sampled attacks smoothly

[1] Saxophones are also made of brass, but are considered woodwinds.

to table-lookup signals because a rapid cross-fade causes phase cancellation if the sample and table are not matched in phase. The solution is to make the sample long enough (and 30 ms is usually enough) that it settles into a harmonic spectrum. Then, we perform analysis on the end of the attack to obtain the amplitude and phase of each partial. This becomes the first waveform, and every waveform after that has to keep the same phase relationship. With these tricks, we can achieve realistic, inharmonic or noisy attacks and then control the evolution of the harmonic spectrum using almost arbitrary amplitude and frequency controls.

12.2.4 Where Do We Get Control Information?

This raises the question of where to get amplitude and frequency controls. In traditional synthesis research, illustrated at the left of Figure 12.3, control is considered an input to the instrument, which is characterized by some synthesis algorithm (perhaps a physical model or a vocal model, etc.) To evaluate the synthesis technique, we make sounds with the synthesizer and (often) compare to acoustic sounds. The assumption is that if the comparison is not good, we should go back and fix the synthesis algorithm.

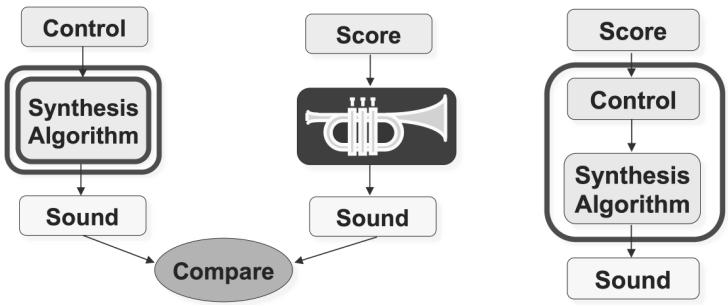

Figure 12.3: *Left:* Traditional approach to synthesis algorithm research and development. It is assumed that control functions are not critical and not part of the synthesis algorithm. *Right:* The SIS approach. Control and synthesis are considered to be related and must be developed and optimized together. Both control *and* synthesis are critical to good, realistic and musical output.

The SIS Research Approach

In contrast, the approach of Spectral Interpolation Synthesis is that *control is an integral part of the synthesizer*. If you do not control a synthesis algorithm properly, it will never sound natural. This idea is illustrated on the right of Figure 12.3. Research has shown that crude ADSR envelopes, and even envelopes derived from the analysis of individual acoustic instrument tones, do not give musical results *even if the synthesis algorithm is perfect*, so it follows that decades of synthesis research are based on faulty assumptions!

Divide-and-Conquer: Performance Model vs. Synthesis Model

To develop control for spectral interpolation, we take the divide-and-conquer approach shown in Figure 12.4 of separating the control problem from the synthesis problem (but keep in mind that we need both together to get good sounds in the end). In this model, the input is a *score*—the notes we want to play, along with

phrase marks, loudness indications, and perhaps other annotations. The *performance model* models how the performer interprets the score and produces controls for the instrument—in the spectral interpolation approach, this is mainly amplitude and frequency. Then, we use the synthesis model (Figure 12.2) to convert control signals into sound.

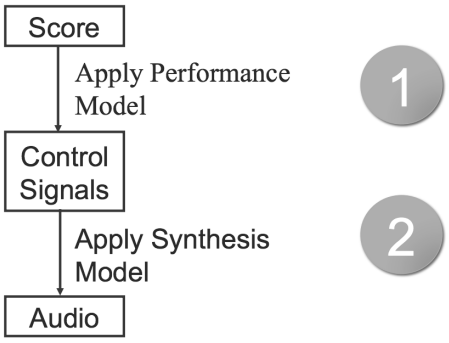

Figure 12.4: The divide-and-conquer approach. Even though control and synthesis must both be considered as part of the problem of musical sound synthesis, we can treat them as two sub-problems.

Research Model: Synthesis Refinement

Given this framework, we can develop and refine the synthesis model by extracting control signals from real audio produced by human performers and acoustic instruments. As shown in Figure 12.5, the human-produced control signals are used to drive the synthesis, or *instrument*, model, and we can listen to the results and compare them directly to the actual recording of the human. If the synthesis sounds different, we can try to refine the synthesis model. For example, dissatisfaction with the sound led us to introduce sampled attacks for brass instruments.

Note that the ability to drive the instrument model with amplitude and frequency, in other words parameters directly related to the *sound* itself, is a big advantage over physical models, where we cannot easily measure physical parameters, like bow-string friction or reed stiffness, that are critical to the behavior of the model.

Research Model: Control Refinement

Assuming we have produced a good-sounding synthesizer, we can then turn to the control problem. The idea here is to get humans to perform scores, extract control information, drive the synthesis with controls, and produce a sound. This output from *measured* control signals represents the best we could hope to achieve since a human produced the control with a real instrument. Now, we can also translate the score to controls with a *performance model*. If we do not like the results, we can improve the model and keep iterating until we have something we like. This is shown in Figure 12.6.

12.2.5 A Study Of Trumpet Envelopes

We will now discuss a particular study that followed the paradigm just described. My students and I constructed a good spectral-interpolation-based trumpet synthesizer and set out to obtain good control envelopes from scores. This discussion

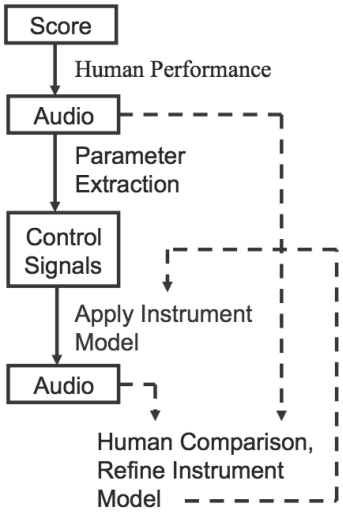

Figure 12.5: Synthesis refinement. To optimize and test the synthesis part of the SIS model, control signals can be derived from actual performances of musical phrases. This ensures that the control is "correct," so any problems with the sound must be due to the synthesis stage. The synthesis stage is refined until the output is satisfactory.

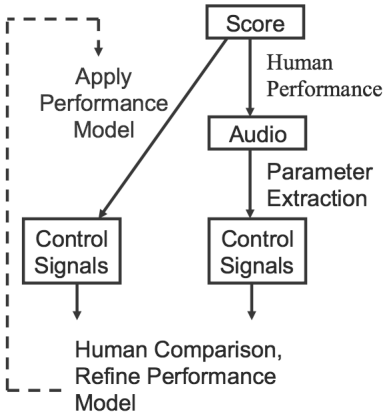

Figure 12.6: Control refinement. Having developed an adequate synthesis model (see Figure 12.5), we compare controls from a performance model to controls extracted from human performance. Both controls are synthesized by the known-to-be-good synthesis model, so any problems can be attributed to the performance model that computes controls. The model is refined until the output sounds good compared to the output based on human performance.

is not just relevant to spectral interpolation synthesis. The findings are relevant to any synthesis method and help explain some of the subtleties we find in acoustic instrument synthesis.

The main conclusion of this work is that

- envelopes are largely determined by context;
- envelope generation techniques can improve synthesis.

In early experiments, we studied the question of how the context of notes in the score affect the center of mass of amplitude envelopes. The *center of mass* tells us, if we were to cut out the envelope shape from a piece of cardboard, where would the shape balance? If the beginning of the note is loud, the center of mass will be earlier. If the sound grows steadily, the center of mass will be later. This is one of the simplest measures of shape and a good place to start.

Figure 12.7 shows some results of this study. The multiple curves show great consistency from one performance to the next when the same music is played, but there is a striking difference between the envelopes on the left, from articulated tones, to the ones on the right, from slurred tones. Some results from measuring center of mass include:

- When the previous pitch is lower and the next pitch is higher (we call this an "up-up" condition), notes showed a later center of mass than other combinations;
- large pitch intervals before and after the tone resulted in an earlier center of mass than small intervals;
- legato articulation gave a later center of mass than others (this is clearly seen in Figure 12.7.

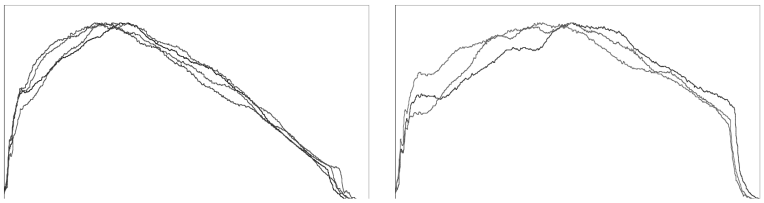

Figure 12.7: Context influences amplitude envelopes. The envelopes on the left are from the second of three articulated (tongued) notes. At right, the envelopes are from the second of three slurred notes. It is clear that articulation and context are significant factors in determining envelope shape. There is not a single "characteristic" trumpet envelope, and measuring the envelope of one tone out of context will not produce many of the features seen above.

Envelope Model

The center of mass does not offer enough detail to describe a complete trumpet envelope. A more refined model is illustrated in Figure 12.8. In this model, the envelope is basically smooth as shown by the dashed line, but this overall shape is modified at the beginning and ending, as shown inside the circles. It is believed that the smooth shape is mainly due to large muscles controlling pressure in the

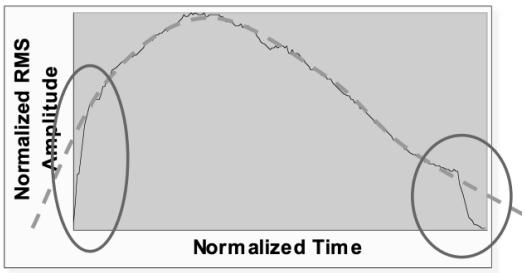

Figure 12.8: The "tongue and breath" envelope model is based on the idea that there is a smooth shape (dashed line) controlled by the diaphram, upper body and lungs, and this shape is modified at note transitions by the tongue and other factors. (Deviations from the smooth shape are circled.)

lungs, and the beginning and ending are modified by the tongue, which can rapidly block or release the flow of air through the lips and into the trumpet.

The next figure (Figure 12.9) shows a typical envelope from a slurred note where the tongue is not used. Here, there is less of a drop at the beginning and ending. The drop in amplitude (which in the data does not actually reach zero or silence), is probably due to the disruption of vibration when valves are pressed and the pitch is changed. Whatever is going on physically, the idea of a smooth breath envelope with some alterations at the beginning and ending seems reasonable.

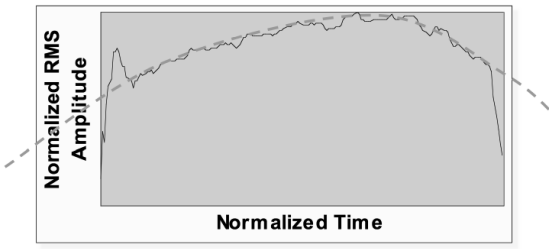

Figure 12.9: The envelope of a slurred note. This figure shows that the "tongue and breath" model also applies to the envelopes of slurred notes (shown here). The deviations at the beginning and ending of the note may be due to trumpet valves blocking the air or to the fact that oscillations are disrupted when the pitch changes.

Based on looking at many envelopes, we conclude that a "breath envelope" is useful to give the overall shape, and a specific attack and decay should be added to incorporate fine details of articulation. This envelope model is shown in detail in Figure 12.10, which shows 9 discrete parameters used to describe the continuous curve. The generic "breath envelope" shown in the upper part of the figure is an actual envelope from a long trumpet tone. By taking a range of the whole envelope, from *tf* to *tt*, we can get a more rounded shape (smaller *tf* larger *tt*) or a flatter shape (*tf* and *tt* close in time), and we can shift the center of mass earlier (later *tf* and *tt*) or later (earlier *tf* and *tt*). Additional parameters control the beginning and ending details. Note how this is decidedly not an ADSR envelope!

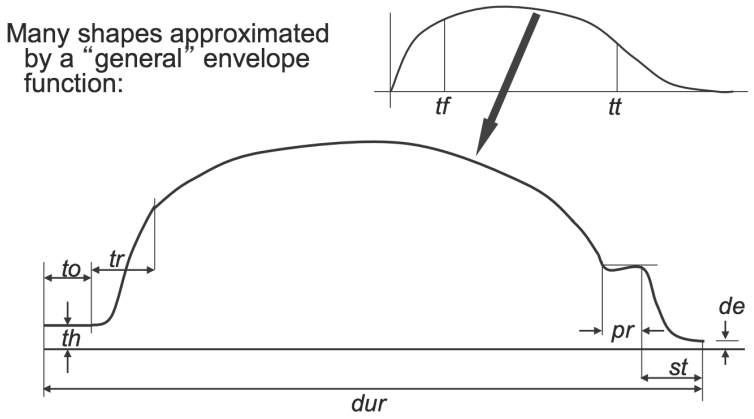

Figure 12.10: An amplitude envelope specification. In order to produce a wide variety of shapes based on the "tongue and breath" envelope model, we describe continuous envelopes with 9 parameters, as shown here.

Computing Parameters

All 9 envelope parameters must be automatically computed from the score. Initially, we used a set of rules designed by hand. Parameters depend on:

- pitch (in semitones, according to score);
- duration (in seconds, according to score);
- begin-phrase (is this the first note in a phrase?);
- end-phrase (is this the last note in a phrase?)
- from-slur (is there a slur from preceding note?);
- to-slur (is there a slur to the next note?);
- direction-up (is this note higher than preceding note?).

The rules were developed by fitting generated envelopes to actual measured envelopes and generalizing from observed trends. Here is one example, illustrating the computation of the parameter *tf*. As shown in Figure 12.11, there are three cases. If coming from a slur and the direction is up, $tf = 0.1$. If the direction is not up, $tf = 0.4$. If *not* coming from a slur, $tf = 0.03 - 0.01 \times \log_2(dur)$.

Similar rules are used to compute all 9 parameters of every envelope. The score-to-envelope mapping is hand-crafted and far from complete and perfect, but the results, at least for simple scores, is quite good and sounds very natural. Constructing mappings between multi-dimensional spaces (in this case, from multiple features of scores to multiple envelope parameters) is a natural problem for machine learning. In fact, my student Ning Hu created an automated system to analyze audio recordings of acoustic instruments and create both a spectral interpolation synthesis model and a performance model that includes amplitude envelopes and vibrato. No manual rule construction or design was required [**?**].

12.2.6 Summary and Discussion of Spectral Interpolation

Spectral Interpolation Synthesis is based on modeling "real" performances, using measurements of spectra and other audio features to model what the listener hears

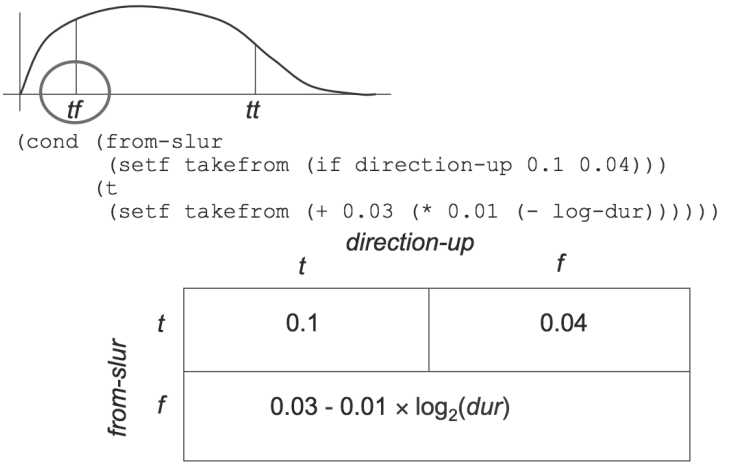

Figure 12.11: Computing envelope parameters. This example shows how the parameter *tf* is derived from score features *from-slur*, *direction-up* and *dur*.

without any modeling of the physical instrument. The performance model results in phrase-long control functions rather than individual notes. Since control functions are based on note-to-note transitions and context, the method requires some look-ahead. (The shape of the envelope depends on the pitch and articulation of the following note.)

12.2.7 Conclusions

What have we learned? First, envelopes and control are critical to music synthesis. It is amazing that so much research was done on synthesis algorithms without much regard for control! Research on the spectral centroid showed statistically valid relationships between score parameters and envelope shape, which should immediately raise concerns about sampling synthesis: If samples have envelopes "baked in," how can sampling *ever* create natural-sounding musical phrases? Certainly not without a variety of alternative samples with different envelopes and possibly some careful additional control through amplitude envelopes, but for brass, that also implies spectral control. Large sample libraries have moved in this direction.

The idea that envelopes have an overall "breath" shape and some fine details in the beginning and ending of every note seems to fit real data better than ADSR and other commonly used models. Even though spectral interpolation or even phrase-based analysis/synthesis have not become commonplace or standard in the world of synthesis, it seems that the study of musical phrases and notes in context is critical to future synthesis research and systems.

12.3 Algorithmic Control of Signal Processing

In the previous section, we looked carefully at some ideas on the synthesis of acoustic instrument sounds. In this section, we consider something that has no basis in the acoustical world: the direct generation and control of audio through algorithms.

There are relatively few works that push decision making, selection, wave shape, and parametric control down below the note level and into the audio signal itself. Ianis Xenakis developed a system called GENDYN that creates waveforms from line segments using algorithmic processes [**?**]. Herbert Brun's SAWDUST is another example, described as a program for composing waveforms (see Section **??**). Curtis Roads wrote the book *Microsound* [**?**], which concerns granular synthesis and more generally the realm between samples and notes. We have explored granular synthesis with Nyquist in some detail. In this section, we will look at signals controlled by (Nyquist) patterns and patterns controlled by signals.

12.3.1 Sounds controlled by Patterns

In Figure 12.12, we see some code and a generated control function (a Nyquist signal). The signal is created by the SAL function `pat-ctrl`, which takes two patterns as parameters. The first pattern (`durpat`) returns durations and the second (`valpat`) returns amplitude values. As seen in the generated signal, `durpat` alternates between 0.1 and 0.2, while `valpat` cycles through the stair-step values of 0, 1, 2.

Pat-ctrl

```
pat-ctrl(make-cycle({0.1 0.2}), ;duration
         make-cycle({0 1 2}))   ;amplitude
```

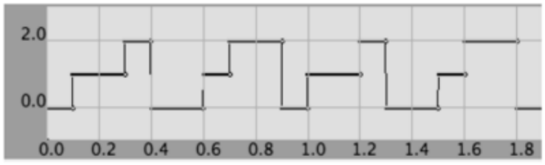

Figure 12.12: The `pat-ctrl` function creates a piecewise-constant function where each step has a duration and amplitude determined by the two pattern generator object parameters. Each pair of numbers retrieved from the two patterns produces one step in the stair-step-like function.

The implementation of `pat-ctrl` is shown below. This is a recursive sequence that produces one segment of output followed by a recursive call to produce the rest. The duration is infinite:

```
define function pat-ctrl(durpat, valpat)
      return seq(const(next(valpat), next(durpat)),
                 pat-ctrl(durpat, valpat))
```

We can use `pat-ctrl` to construct a frequency control function and synthesize that frequency contour. The function `pat-fm`, shown below, adds the result of `pat-ctrl` to a `pitch` parameter, converts the pitch from steps to Hz, and then synthesizes a sinusoid. If `durpat` returns very small values, the resulting sound will not have discernible pitch sequences, at least not at the level of segment durations returned by `durpat`. However, higher-level structures in `durpat` and `valpat` will create structured timbral variations that catch our attention. In other words, the intent here is to modulate the sine tone at audio rates using complex patterns and resulting in novel sounds.

```
define function pat-fm(durpat, valpat, pitch, dur)
  begin
```

```
  with hz =
    step-to-hz(pitch + pat-ctrl(durpat, valpat))
  return pwl(0.01, 1, dur - 0.1, 1, dur) *
         hzosc(hz + 4.0 * hz * hzosc(hz))
end
```

Using Scores

One long sine tone may not be so interesting, even if it is modulated rapidly by patterns. The following example shows how we can write a score that "launches" a number of `pat-fm` sounds (here, the durations are 30, 20, 18, and 13 seconds) with different parameters. The point here is that scores are not limited to conventional note-based music. Here we have a score organizing long, overlapping, and very abstract sounds.

```
exec score-play(
 {{ 0 30 {pat-fm-note :grain-dur 8 :spread 1
          :pitch c3 :fixed-dur t :vel 50}}
  {10 20 {pat-fm-note :grain-dur 3 :spread 10
          :pitch c4 :vel 75}}
  {15 18 {pat-fm-note :grain-dur 1 :spread 20
          :pitch c5}}
  {20 13 {pat-fm-note :grain-dur 1 :spread 10
          :pitch c1}}})
```

12.3.2 Pattern Controlled by Sounds

If we consider sounds controlled by patterns, we should think about the opposite. Here, we will look at control envelopes (which are also of type `SOUND` in Nyquist) and how they can be used in patterns. (Maybe you can also think of interesting ways for *audio* sounds to control patterns.)

When constructing sequences of events, scores and `score-gen` may be all you need. But if you want to influence the evolution of fine-grain decisions, such as pattern output over time, then perhaps envelopes and other controls can be useful. This relates to a concept we saw before: *tendency masks*, which specify long-term trends. A standard example is: randomly choose the next pitch between an upper and lower bound, where the bounds are given by functions that change over time.

To implement something like tendency masks in patterns, we will use Nyquist `SOUND`s to specify the global continuous evolution of parameter values. To access the sound at a particular time, we use `sref(`*sound*`,` *time*`)`. Note that *sound* can be any `SOUND`, but it might be most convenient to use a piece-wise linear envelope.

In `sref`, *time* is relative to the environment, so *time* = 0 means "now." And remember that while behaviors start at "now," existing sounds have a definite start time. So when we write `sref(`*sound*`, 0)`, it means access *sound* at the current time, not access the beginning (time 0) of *sound*.

A Template for Global Control using Sounds

Here is an example you can build on for controlling patterns with sounds. There are three definitions:

- `pitch-contour` defines a time-varying contour that we want to use with some pattern computation;
- `get-pitch` is a function that simply accesses the pitch contour at the current time (indicated by time 0);

- `pwl-pat-fm` is a function that will create and use a pattern. Within the pattern, we see `make-eval({get-pitch})`. The `make-eval` pattern constructor takes an *expression*, which is expressed in Lisp syntax. Each time the pattern is called to return a value, the expression is evaluated. Recall that in Lisp, the function call syntax is (*function-name arg1 arg2 ...*), so with no arguments, we get (*function-name*), and in SAL, we can write a quoted list as {*function-name*}. Thus, `make-eval({get-pitch})` is the pattern that calls `get-pitch` to produce each value.

```
variable pitch-contour =
           pwl(10, 25, 15, 10, 20, 10, 22, 25, 22)

function get-pitch()
  return sref(pitch-contour, 0)

function pwl-pat-fm()
  begin
    ...
    make-eval(get-pitch),
    ...
  end

play pwl-pat-fm()
```

We can do something similar with `score-gen`, using a `SOUND` to provide a time-varying value that guides the progress of some parameter. In the following example, the variable `pitch-contour` is accessed as part of the `pitch:` calculation.

```
begin
  with pitch-contour =
           pwl(10, 25, 15, 10, 20, 10, 22, 25, 22),
       ioi-pattern = make-heap(0.2 0.3 0.4)
  exec score-gen(
      save: quote(pwl-score),
      score-dur: 22,
      pitch: truncate(
          c4 + sref(pitch-contour, sg:start) +
          #if(oddp(sg:count), 0, -5)),
      ioi: next(ioi-pattern),
      dur: sg:ioi - 0.1,
      vel: 100)
end
```

You can even use the envelope editor in the NyquistIDE to graphically edit `pitch-contour`. To evaluate `pitch-contour` at a specific time, `sref` is used as before, but the *time* parameter to `sref` is `sg:start`, not 0. We need to use `sg:start` here because the entire `score-gen` computation is performed at time 0. We are not in a behavior, and there is no environment that changes the current time with each new note. Since everything is (normally) relative to time 0, we use `score-gen`'s special `sg:start` variable to refer to the "current" time, that is, the start time of each note.

12.4 3-D Sound

We now turn to the topic of sound as originating in 3-D space. We know from previous discussions of perception that there are various cues that give the impression of the origin of sounds we hear. In computer music work, composers are interested in simulating these cues to simulate a 3-D sonic environment. Earlier,

we introduced the idea of head-related transfer functions and the simulation of spatial cues over headphones. In this section, we expand on this discussion and also consider the use of multiple speakers in rooms and particularly in concert halls.

12.4.1 Introduction

To review from our discussion of sound localization, we use a number of cues to sense direction and distance. Inter-aural time delay and amplitude differences give us information about direction in the horizontal plane, but suffers from symmetry between sounds in front and sounds behind us. Spectral cues help to disambiguate front-to-back directions and also give us a sense of height or elevation. Reverberation, and especially the direct-sound-to-reverberant-sound ratio gives us the impression of distance, as do spectral cues.

12.4.2 Duplex Theory

In the *duplex theory* of Lord Rayleigh, sound localization is achieved through a combination of interaural time difference (ITD) and interaural level difference (ILD). ITD is caused by the difference in distance between our ears and the sound source when the source is to the left or right. The time difference is related to the speed of sound, and remembering that sound travels about 1 foot per millisecond, we can estimate that ITD is a fraction of a millisecond. (No wonder our ears can perceive time so precisely–at least precise relative timing between left and right ears is useful for localization.)

The ILD is caused by the masking effect of our heads when sound comes from one side or the other. Since different frequencies refract around our heads differently, the ILD is frequency dependent and more pronounced at higher frequencies where the wavelengths are short and our head does a better job of shielding one ear from the sound.

In duplex theory, there is ambiguity caused by symmetry. Any direction from the head falls on a cone-shaped surface, where the apex of the cone is at the head and center line of the cone runs through the two ears. This set of ambiguous directions is called the *cone of confusion* (Figure 12.13) because every point on the cone produces the same ITD and ILD.

In fact, the duplex theory ignores the pinnae, our outer ears, which are not symmetric from front-to-back or top-to-bottom. Reflections in the pinnae create interference that is frequency-dependent and can be used by our auditory system to disambiguate sound directions. The cues from our pinnae are not as strong as ITD and ILD, so it is not unusual to be confused about sound source locations, especially along the cone of confusion.

12.4.3 HRTF: Head-Related Transfer Functions

When sound reaches our ears from a particular direction, there is a combination of ITD, ILD, and spectral changes due to the pinnae. Taken together, these cues can be expressed as a filter called the head-related transfer function, or HRTF. We can measure the HRTF, for example by placing tiny microphones in the ears of a subject in an anechoic chamber, holding their head steady, and recording sound from a speaker placed at a carefully controlled angle and elevation. There are some clever ways to estimate the HRTF, but we will skip the signal processing details. To fully characterize the HRTF, the loudspeaker and/or listener must be

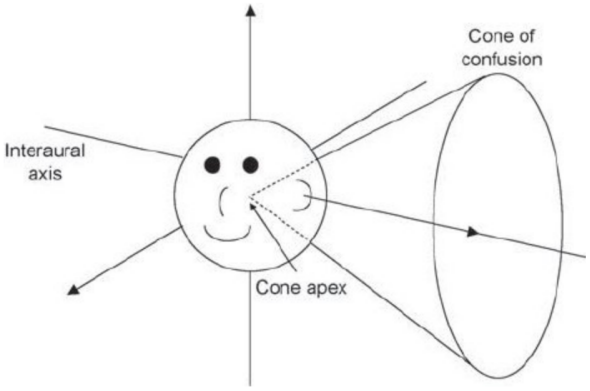

Figure 12.13: The "cone of confusion," where different sound directions give the same ITD and ILD.
(Credit: Bill Kapralos, Michael Jenkin and Evangelos Milios, "Virtual Audio Systems," *Presence Teleoperators & Virtual Environments*, 17, 2008, pp. 527-549.)

moved to many different angles and elevations. The number of different angles and elevations measured can range from one hundred to thousands.

To simulate a sound source at some angle and elevation, we retrieve the nearest HRTF measurement to that direction and apply the left and right HRTFs as filters to the sound before playing the sound to the left and right ears through headphones. (See Figure 12.14.)

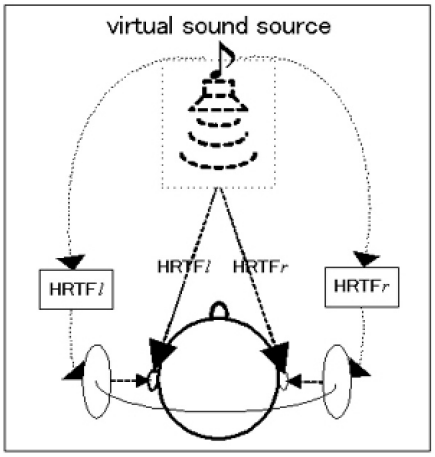

Figure 12.14: Sound is localized with HRTFs by filtering a sound to simulate the effect of delays, head, and ears on the sound. A different filter is used for left and right channels, and the filters are different for different directions. (From http://www.ais.riec.tohoku.ac.jp/Lab3/sp-array/index.html).

One way to implement the HRTF filters is to compute the HRIR, or head-related impulse response, which is the response of the HRTF to an impulse input. To filter a sound by the HRTF, we can convolve the sound with the HRIR. If the sound is moving, it is common to interpolate between the nearest HRTFs or HRIRs so that there is no sudden switch from one filter to another, which might cause pops or clicks in the filtered sound.

12.4.4 HRTF, Headphones, and Head Tracking

When HRTFs are used, the listener normally wears headphones, which isolate sounds to the left and right ears and eliminate most of the effects of actual room acoustics in the listening space. Headphones have the problem that if the listener moves their head, the simulated space will rotate with the head and the headphones. To solve this problem, headphones are often tracked. Then, if the head turns 20° to the left, the virtual sound source is rotated to the right to compensate. Headphones with head tracking are sometimes combined with virtual reality (VR) goggles that do a similar thing with computer graphics: when the listener turns 20° to the left, the virtual world is rotated to the right, giving the impression that the world is real and we are moving the head within that world to see different views.

12.4.5 Room Models

An alternative to HRTFs and headphones is to use speakers in a room to create the effect of localized sounds. One approach is to use ray-tracing ideas from computer graphics to estimate a sound field in an imaginary space. One difficulty is that sound is slow and has long wavelengths relative to objects and rooms, which means that refraction is significant—sound does not always travel in straight lines! The refraction is frequency dependent. Nevertheless, ray tracing can be used to develop early reflection models for artificial reverberation that convey the geometry of rooms, even if the simulation is not perfect.

12.4.6 Doppler Shift

Another interesting process that works well with loudspeakers is the simulation of motion using Doppler shift. Sound at a distance is delayed. When sound sources move, the distance changes, so does the delay, and the change creates Doppler shift, or a change in frequency. Doppler shift can be modeled with actual delay consisting of buffers of samples, perhaps with some interpolation to "tap" into the delay at fractional positions. Sometimes, we can just use frequency modulation to give the effect of Doppler shift without any real simulation of delay. If there are multiple reflections, each reflection can have a different Doppler shift. This could add additional realism, but I have never seen a system that models reflections with individual Doppler shifts.

12.4.7 Reverberation

Doppler shift is enhanced through the use of reverberation. If a sound is receding rapidly, we should hear several effects: the pitch drops, the sound decreases in amplitude and the ratio of reverberation to direct sound increases. Especially with synthesized sounds, we do not have any pre-conceptions about the absolute loudness of the (virtual) sound source. Simply making the source quieter does not necessarily give the impression of distance, but through the careful use of reverberation, we can give the listener clues about distance *and* loudness.

12.4.8 Panning

Stereo loudspeaker systems are common. A standard technique to simulate the location of a sound source between two speakers is to "pan" the source, sending some of the source to each speaker. Panning techniques and the "hole-in-the-middle" problem were presented in detail in Section **??**. Panning between two

speakers can create ILD, but not ITD or spectral effects consistent with the desired sound placement, so our ears are rarely fooled by stereo into believing there is a real 3-D or even 1-D space of sound source locations.[2] Finally, room reflections defeat some of the stereo effect or at least lead to unpredictable listening conditions.

12.4.9 Multi-Speaker Playback

If two speakers are better than one, why not many speakers? This is not so practical in the home or apartment, but common in movie theaters and concert halls, especially for concerts of computer music. "Surround Sound" systems are available as consumer audio systems as well. In general, the simulation of direction is achieved by choosing the nearest speaker in the desired direction. Often, sound is panned between the nearest 2 speakers when speakers are in a plane around the audience, or between the nearest 3 speakers when speakers are arranged in a dome.

At the risk of over-generalization, there are two general schools of thought on sound reproduction with loudspeakers. One school views loudspeakers as an approximation to some ideal. In this view, loudspeakers should have high fidelity to recreate any desired source sound with minimal alteration. When many speakers are used, they are almost invariably identical and uniformly distributed to allow panning to any angle. The focus is on sound content, and any speaker limitations are just a necessary evil.

The other school views electro-acoustic music more holistically. Loudspeakers are a part of the process. Rather than bemoan speaker imperfections, we can create an "orchestra" of loudspeakers of different types and placement. Composers can utilize directionality and frequency responses of different loudspeakers with musical intent. We embrace the fact that sound comes from loudspeakers rather than trying to hide it.

Recently, a new approach to sound localization with loudspeakers has come under investigation. The technique, called *wavefield synthesis*, uses a large array of speakers to reproduce the wave front of an imaginary 3-D or 2-D scene. Imagine a cubical room with one wall open and sound sources outside the room. Imagine an array of microphones on a grid stretched across the open wall, capturing the incoming sound wave at 100 or more locations. Now, imagine closing the wall, replacing each microphone with a small loudspeaker, and playing back the recorded sound wave through 100 or more loudspeakers. In principle, this should reproduce the entire incoming sound wave at the wall, creating a strong 3-D effect. Of course there are questions of directionality, how many speakers are needed, whether to use a 2-D or 1-D array, room reflections, etc. An example of wavefield synthesis and playback is an auditorium at the Technical University of Berlin, which has 832 audio channels. Small speakers are placed in a continuous line on the left, front, and right walls. The speakers are separated by only 10 cm, but because of the small size, there are larger speakers just below them, every 40 cm. A small cluster of computers is used to compute each channel based on the precise delay from virtual sound sources. The results are quite impressive and

[2] Thus, the whole idea of stereo is highly suspect. Could it be that stereo was promoted by manufacturers who wanted to sell more gear? There were multi-speaker systems in use before consumer stereo, and experiments have shown that three-channel systems (adding a center channel) are significantly better than stereo. My guess is that stereo is ubiquitous today because it was both interesting enough to sell to consumers yet simple enough to implement in consumer media including the phonograph record and FM radio. Even though we have only two ears, stereo with two channels falls far short of delivering a full two-dimensional sound experience, much less three dimensions.

unlike any typical speaker array, giving the impression that sounds are emerging from beyond or even within the line of speakers.

12.4.10 Summary

In this section, we have reviewed multiple perceptual cues for sound location and distance. HRTFs offer a powerful model for simulating and reproducing these cues, but HRTFs require headphones to be most effective, and since headphones move with the head, head tracking is needed for the best effect. For audiences, it is more practical to use loudspeakers. Multiple loudspeakers provide multiple point sources, but usually are restricted to a plane. Panning is often used as a crude approximation of ideal perceptual cues. Panning can be scaled up to multiple loudspeaker systems. Rather than treat speakers as a means of reproducing a virtual soundscape, loudspeaker "orchestras" can be embraced as a part of the electroacoustic music presentation and exploited for musical purposes. Finally, wavefield synthesis offers the possibility of reproducing actual 3-D sound waves as if coming from virtual sound sources, but many speakers are required, so this is an expensive and still mostly experimental approach.