Chapter X

Sampler Instrument

A *sampler* is a synthesizer that generates sound by playing recorded sounds rather than calculated waveforms. The wave file player instrument described in an earlier chapter is a simple form of sampler. Unlike the wave player, a sampler has the ability to automatically select from a set of samples, shift the pitch up and down, sustain the sound indefinitely by looping, alter the envelope, and apply modulations, such as vibrato, to the recorded sample. The recorded sound is typically loaded into a table and scanned in a manner similar to a wavetable oscillator. This has led to the use of the term *wavetable synthesis* for this kind of instrument. Wavetable synthesis originally referred to a variation of additive synthesis that created a complex sound by cross fading between multiple wavetables. The confusion of terminology is unfortunate, but it is now common for a sampler to be called a wavetable synthesizer. To further add to the confusion, a digitally recorded sound used by a sampler is referred to as a *sample*. This is not the same as the quantized amplitude value that we normally call a sample, but rather refers to the entire set of samples that make up the recording. Descriptions of samplers will often make a distinction between *sample* and *sample data*, with the former referring to the whole recorded sound and the latter referring to the individual sample values that make up the recorded sound.

Although in principle a sampler can playback any waveform, it is most often used to emulate traditional musical instruments. Since the sample contains the complete spectrum of the sound, including any dynamic properties, we don't need to sum harmonics with separate envelopes or modulate to create a complex waveform, but simply copy the sample values to the output device at the sample rate. However, there are several concerns that complicate a sampler.

2 Sampler

The spectrum of a musical instrument sound will change depending on pitch, articulation, and loudness. To fully capture the sound of an instrument, we would need each pitch in the instrument's range recorded at dynamics from *pp* to *ff*, and with different articulation at each pitch and dynamic. For an instrument with a three octave range, six different dynamics, and four articulations, we would need over 3,000 samples. For an orchestra with only 10 different instruments we might need over 30,000 samples.

Fortunately, we can change the pitch of a sample in the same way we change the frequency of a wavetable oscillator, i.e., we alter the increment value used to index through the table. Unlike a typical oscillator wavetable, a recorded sound has a built-in envelope. When we change the increment to achieve a different frequency, we also vary the attack and decay of the sound envelope. For small pitch changes, the variation in the envelope may not be perceptible. But if we try and shift the pitch more than a small amount, the envelope will be significantly altered. Rather than rely solely on the recorded sound's envelope and volume level, we can apply an envelope generator and volume level to the sample. This will reduce the need for a large variety samples to represent volume and articulation variations.

Pitch shifting will also affect the spectrum. Skipping or repeating samples is a form of re-sampling and has several potential problems. If we shift the pitch too far, we can produce alias frequencies and audible quantization noise. Convolving the sample with a *sinc* filter can eliminate much of the problem, but is time consuming and will alter the timbre of the sound. Stretching samples over too far of a range, even with filtering, results in samples that sound artificial or like a low-quality recording, effectively defeating the purpose of a sampler. Fortunately, since nearby pitches sound about the same, we can reuse a sample for multiple pitches if we limit the pitch shift to a small range, usually 2-4 semitones.

Likewise, the changes to the spectrum at different volume levels and different articulations might be small enough that we can use a smaller number of samples without a noticeable effect. In some cases, we can apply a low-pass filter to attenuate higher frequencies, simulating a softer sound. As with pitch, we might need 2-4 samples recorded at different volume levels in order to reproduce the natural

variations in the spectrum, but can sometimes get by with only one sample recorded at a high volume level.

A sound recorded with tremolo, vibrato, or pitch bend, will also cause a problem during playback. Changing the wavetable increment will change the vibrato and pitch bend rate along with the pitch. If the sample is recorded without vibrato, we can apply vibrato with a LFO. Variable vibrato and pitch bend can be implemented with either an envelope generator, parameters from a score, or live performance controller values.

We must also consider varying the duration of the sound. For percussion instruments, including keyboards, we want the full duration of the recording, but with the ability to cut-off the sound before it completes. This can be accomplished easily by using an envelope generator with a short release segment to fade the sound out quickly when the note is stopped. However, for wind instruments, we want to be able to play a sound for any duration. This can be accomplished by looping over the wavetable, but if we loop over the whole recording we will reproduce the attack and decay portions of the sound. The result will sound like a repeated note instead of a sustained note. One solution is to cut off the attack and decay portions of the recording and then control the envelope independently. However, this eliminates the attack transients, often a key part of the instrument's timbre. For this reason, the sample is divided into two or more sections. The first section represents the attack, including it's transients, and is usually played straight through. The second section represents a typical sustained sound and is looped as long as needed. A third section can be used for the decay, and would be played straight through like the attack section, but usually the sampler will apply an envelope to the sustained portion to produce a variable decay.

Using these techniques, the sampler will be somewhat less than ideal, but still very close to the original sound and much more practical than using thousands of samples per instrument. The resulting instrument has the same basic structure as the Tone instrument described earlier, but with a specialized wavetable oscillator in place of the single-period oscillator.

Two standards for sample data collections are currently in use. The SoundFont® (SF2) format was developed by E-mu systems and then released for public use. SF2 is a proprietary format, but has

become a de-facto industry standard. The Downloadable Sounds format (DLS) was developed by the MIDI Manufacturers Association (MMA) and is also available for public use. DLS is typically used by manufacturers of keyboard synthesizers, but is also used in various mobile communications devices and software synthesizers.

Both SF2 and DLS are closely tied to the MIDI event protocol. In fact, the primary purpose of both specifications is to enhance the MIDI protocol with better note-by-note control of synthesis parameters. Thus a SF2/DLS sound collection is generally designed to be used in a MIDI-based synthesizer. The widespread adoption of MIDI has produced a standard combination of MIDI and SF2/DLS that has become ubiquitous in both hardware and software synthesizers. This widespread adoption has resulted in the production of a large number of both SF2/DLS sample data files and MIDI files along with software tools for creating both. Incorporating a synthesizer with these capabilities into a software application allows immediate use of a large set of readily available sounds and music.

Sound Bank Instrument Overview

Although the two formats are significantly different in file structure, both are designed around a similar synthesis model and parameter set, and can be converted back and forth with little to no loss in information. Thus both formats can be combined into one representation called a Sound Bank (SB). The top-level structure of a sound bank instrument is shown below.

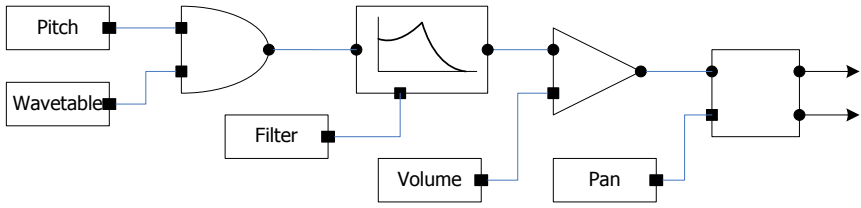


Figure 1 - Sound Bank Instrument

A wavetable oscillator is used to generate the audio signal, which is then passed through a low-pass filter, amplifier and pan control. This is a very generic synthesis structure and can potentially support

multiple synthesis algorithms, including additive, subtractive, and sample playback. Typically, only recorded sound playback is used, implemented using a multi-period oscillator with separate transient and steady-state (loop) sections.

Each synthesis parameter is a combination of initialization values, internal unit generators, and MIDI channel voice and controller values. Four unit generators are available as inputs to the synthesis parameters.

1. Volume EG
2. Modulation EG
3. Vibrato LFO
4. Modulation LFO

Note that DLS1 does not include the modulation LFO. Instead, the Vibrato LFO can be used as either a frequency control or an amplitude control, with the appropriate scaling unit for either frequency or amplitude.

Envelope generators have the form shown below.

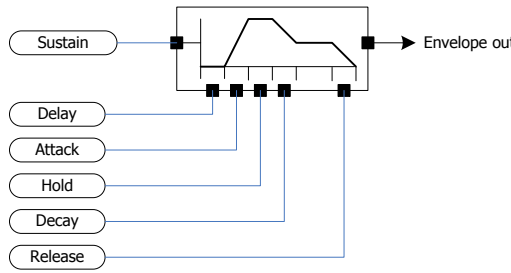


Figure 2 - Envelope Generator

The EG is a constant-rate type with the peak output normalized to [0,1]. Sustain is specified as a percentage of peak output. Rates are specified in time cents (explained below). DLS1 does not include the delay or hold segments, using a typical ADSR instead, and the two values are set to 0.

LFO generators are sine wave oscillators, normalized to [-1,+1], with the output passed through a gate. The LFO delay prevents applying vibrato to the attack portion of the sound.

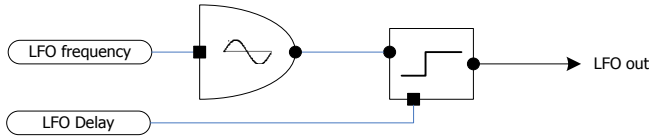
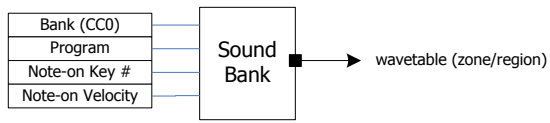


Figure 3 - LFO

Wavetable Selection

Wavetable selection is made based on a combination of MIDI values.



A sample may span multiple key and/or velocity values, indicated by low-key, high-key, low-velocity, and high-velocity settings in the sound bank. The range of notes is called a *zone* or *region*. When a sound bank instrument contains multiple zones that match a key and velocity combination, the synthesis structure shown above must be replicated for each zone and the outputs summed to produce the final amplitude value.

The wavetable may be looped or non-looped. For a non-looped table, the table is scanned once from beginning to end. For a looped table, the portion of the table prior to the loop start point is played through once, then the oscillator cycles over the samples between loop start and loop end.

The same sample data may be used for different zones, and may have different start and end loop points for each zone.

Stereo, or other multi-channel recordings, are produced by playing multiple zones with each zone panned appropriately. This allows for simulated stereo as well as a stereo recording. Note that this makes all wavetables single channel inherently, and is different from the interleaved samples found in WAV files. This is to be expected since single channel wavetables are required if the wavetable phase increment is to work properly.

Example Code

The *Example09a* program implements an example MIDI sequencer using SF2 or DLS sound banks.

BasicSynth Library

The *GMPlayer* and *SFPlayerInstr* classes implement instruments that play sound banks, either SoundFont or DLS files.

The *SFPlayerInstr* class uses a separate envelope generator and ignores the articulation and modulation information from the sound bank. This instrument will respond to real-time frequency changes through the *Param*() function, changing zones as needed. In order to avoid discontinuities, it performs a cross-fade between samples.This instrument also includes an FM capability, local LFO, and pitch bend.

The *GMPlayer* class uses the MIDI bank and program messages to select the sound. Only the bank and program numbers are settable through the *Params* function. It assumes the sound bank implements the GM specification for instrument definition.

Files:

```
Src/Instruments/SFPlayer.h
Src/Instruments/SFPlayer.cpp
Src/Instruments/GMPlayer.h
Src/Instruments/GMPlayer.cpp
```