Sampled Waveforms And Musical Instruments

Josh Green
11321 Beauview Rd.
Grass Valley, CA, 95945 U.S.A.
josh@resonance.org

Abstract

Sampled Waveforms And Musical Instruments (SWAMI) is a cross platform collection of applications and libraries for creating, editing, managing, and compressing digital audio based instruments and sounds. These instruments can be used to compose MIDI music compositions or for other applications such as games and custom instrument applications. Discussed topics will include: common instrument file formats, Swami application architecture, Python scripted instrument editing, CRAM instrument compression, PatchesDB - a web based instrument database, and basic usage of Swami.

Keywords
Instruments, MIDI, Music

1 Introduction

It was decided early on in the development of Swami that the primary focus would be on providing a tool for editing wavetable based instruments. Wavetable instruments are composed of one or more "samples" of digital audio. While the original project goals have been expanded on, the focus remains the same. In the Linux audio and music paradigm of having many tools with well defined purposes interacting with each other, Swami provides a way to create and manage instrument sounds of specific formats. Useful types of applications to interface to the Swami application include: MIDI sequencers, audio sample editors, and JACK enabled applications. Wavetable synthesizers are also pluggable in the Swami architecture, although currently FluidSynth is the only supported soft synth. In addition, other applications may choose to interface to the underlying libraries for instrument file processing.

2 Instrument Formats

There are countless sample based instrument formats currently in use today. Some are commonly used, others obscure, some are based on open standards, some are proprietary. The goal of the Swami project is to support some of the more commonly used formats with a preference for open standards. Swami currently has varying levels of support for the following formats: SoundFont ®[1] version 2, DLS (DownLoadable Sounds), and GigaSampler ®[2]. Support for some of the popular Akai formats are planned for the future, and there is also some interest expressed in creating a new open instrument format. The rest of this section will be dedicated to describing the 3 currently supported formats mentioned (GigaSampler to a lesser extent).

2.1 Synthesis Model

All the currently supported instrument formats use digital audio as the basis of sound synthesis and have a fixed synthesis architecture. This means that there is a fixed number of synthesis components, each with a specific function. The synthesis components modulate various effects and parameters of the played sounds. Synthesis components and parameters include:

- Audio samples with loops
- Key splits and Velocity splits
- Envelopes
- LFOs (Low Frequency Oscillators)
- Filters (Low Pass Filter for example)
- Stereo panning
- Tuning (pitch of playback)
- Reverb and Chorus
- Effect Modulators (MIDI Controller, etc)

[1] SoundFont is a registered trademark of EMU Systems, Inc

[2] GigaSampler is a registered trademark of Nemesys Music Technology, Inc

2.2 Sample Loops

At the core of the synthesis model are the audio samples. The capability of having continuous sounds can be realized by looping a portion of a sample which also saves on storage space. An alternative is to have very long samples which satisfy the maximum expected note duration.

2.2.1 Key and Velocity Splits

When a MIDI note event is generated it contains the MIDI note # and velocity # (the speed at which the note was played, roughly translates to how hard the key was pressed). Key splits and velocity splits define a MIDI note range and velocity range respectively, for which a given audio sample will sound.

Due to the finite nature of digital sampled audio, a sample becomes more distorted the further from its original pitch it is played. While one could use a single sample for the entire MIDI note range, it is usually undesirable, since the sound will likely deviate far from the instrument or sound being reproduced. For this reason, an instrument or sound is often sampled at several different pitches and each one is given the note range (key split) which surrounds the notes closest to the original sampled pitch. Key splits are also used for percussion kits (usually one note per sound) as well as instruments with layered sounds.

Velocity splits are useful for playing back different sounds and/or different effect settings based on the speed a key is played.

2.2.2 Envelopes

Envelopes provide a simple and convenient way to modify an effect (such as volume) over the note playback cycle. The type of envelopes used in these formats are fixed 6 stage envelopes. Each stage is controlled by a value and are named: Delay, Attack, Hold, Decay, Sustain and Release. A brief description of a Volume Envelope: following the Delay period the Attack stage begins which controls the amount of time it takes for the sound to reach its maximum level, the volume is then held for the Hold time, followed by the Decay stage where the volume decreases to its Sustain level over a specified time and remains until the key is released which causes the volume to return to silence within the Release time interval.

2.2.3 LFOs

LFOs provide low frequency oscillation of effects. They are defined by a delay, frequency, and effect modulation level. They can be used to provide tremolo (volume), vibrato (pitch) and filter oscillations. The sine wave is the most commonly used oscillator waveform, but some formats allow for other types of waveforms as well.

2.2.4 Filter

The most common filter used is the Low Pass filter. This produces a “wah-wah” effect often used with guitars and electronic music. It consists of a filter cutoff frequency and Q (quality) value. As the filter cutoff decreases the frequencies above are attenuated. The Q parameter controls the intensity of the resonance at the cutoff frequency. The higher the Q the more pronounced the frequencies will be near the cutoff.

2.2.5 Modulators

Modulators provide a flexible mechanism for controlling effects in real time via MIDI controls or other parameters such as MIDI note or velocity value. SoundFont and DLS files both provide a similar model which allows one to map one or two controls to an effect using a math function (Linear, Concave, Convex, and Switch), direction, polarity and value amount to apply to the effect.

2.3 Instrument Model

Instrument files are organized into different components (lets call them objects). The most intuitive model of instrument files is a tree of objects which have properties (i.e., parameters). The instrument file is the root object and has parameters such as Name, Author, Date, Copyright, etc. This object in turn contains child objects such as Samples, Instruments and Presets. The Sample objects are rather similar between formats in that they represent a single sample of audio with properties such as SampleRate, RootNote, FineTune, and possibly loop information. The Instrument objects consist of a set of child Region objects (Zones in SoundFont terminology) which each reference a Sample. These regions contain all the synthesis parameters to be applied to the referenced sample (key/velocity split ranges, tuning, envelopes, LFOs, etc). In the case of SoundFont files there is an additional level called Presets which group

together Instrument objects and allow offsetting of effect values for all referenced instruments. Instruments (Presets in the case of SoundFont files) define the unique MIDI bank and program number which each instrument should be mapped to, which are used for selecting it for play.

2.3.1 Feature Comparison

The synthesis models of SoundFont and DLS files are very similar. Both have a Volume Envelope, Modulation Envelope (Filter and Pitch), Modulation LFO (Pitch, Filter and Volume), Vibrato LFO (Pitch only), Low Pass Filter, Reverb, Chorus, Panning and Tuning parameters.

The file format on the other hand differs quite a bit. DLS is much more flexible and allows for custom additions. Samples consist of embedded WAV files, which adds a lot of flexibility as far as the audio format (although only 8 bit and 16 bit is defined for compliance). SoundFont files on the other hand are 16 bit audio only. DLS files also have support for multi-channel instruments (surround sound), whereas SoundFont is limited to stereo. In addition DLS uses 32 bit parameter values and SoundFont uses 16 bit.

GigaSampler files on the other hand have a completely different synthesis model. While the file structure is based on DLS they contain many proprietary extensions. Instruments are composed of dimensions which multiplex samples to ranges of a parameter (such as note, velocity, a MIDI controller, etc). This model is a convenient way of triggering different samples or effect parameters based on different inputs. GigaSampler has support for other specific features also like sample cross fading, different filter types, different LFO waveforms, is designed for sample streaming and some other nice features. The down side of this format is that it is not an open standard and many of the parameter ranges are quite small compared to the other formats.

SoundFont version 2	
Creator	E-mu Systems, Inc.
Pros	Open standard, popular, simple fixed synthesis model, flexible effect modulators.
Cons	16 bit mono or stereo audio only, format not very expandable.

DLS (DownLoadable Sounds)	
Creator	MIDI Manufacturers Association
Pros	Open standard (although v2 spec must be purchased), simple fixed synthesis model, flexible file format, large parameter ranges, adopted by MPEG4.
Cons	Not yet in wide use.

GigaSampler	
Creator	Nemesys Music Technology, Inc
Pros	Designed for streaming, dimension model is nice, cross fading, additional filter and oscillator types.
Cons	Proprietary format, small parameter ranges compared to other formats, more complex synthesis.

3 Swami Architecture

The Swami application consists of several components, including:

- libInstPatch (Lib Instrument Patch) – Object oriented instrument editing library.
- libSwami – All useful non GUI specific functionality can be found in this library.
- SwamiGUI – The GUI front end. Also implemented as a library to allow plugins to link against it.
- Plugins – FluidSynth for sound synthesis and FFTune for computer aided sample tuning.

The underlying libInstPatch and libSwami libraries are written in C and utilize the GObject library. This popular architecture was chosen so as to benefit from object oriented programming while still providing the most flexibility in language bindings. These two libraries are also not dependent on any graphical toolkit, simplifying their usage for non GUI applications.
The GUI is also written in C, in an object oriented fashion, and uses the GTK+ version 2 toolkit and GnomeCanvas. Of note is that GnomeCanvas is not dependent on Gnome and likewise neither is Swami.

3.1 libInstPatch

This library is responsible for loading, saving and editing instrument files. Its features include:

- Instrument files are stored in memory as trees of objects with type specific properties.
- Object conversion system which handles converting between file formats and objects. Also handles loading and saving instrument files and importing samples.
- Paste system for an intuitive method of adding objects to an instrument tree.
- Flexible sample data storage (in RAM, in a file, etc).
- Sample format conversion (bit width, mono/stereo, integer/float).
- Voice caches for fast lock free synthesis (only SoundFont synthesis model currently).
- CRAM instrument compression for supported formats and a generic decompresser implementation.
- Attempts to be multi-thread safe.

3.2 libSwami

Functionality which is not specific to instrument file editing or the GUI finds its way here. Features include:

- Plugin system for extending Swami.
- Wavetable driver object.
- MIDI device object for MIDI drivers.
- GValue based control networks.

Of particular interest is the GValue control networks. GValue is a structure used in the GObject library which acts as a wild card container. It can store an integer, float, enum, flags, string, GObject and more. Control networks are formed by connecting one or more SwamiControl objects together, values or events generated by a control object are sent to all connected objects. Examples of usage include, connecting GUI controls to an instrument parameter. When the parameter changes, all controls are notified with the new value. Another example usage is in routing MIDI events between the virtual keyboard and FluidSynth.

Also of note is the SwamiWavetbl object which forms the basis of adding support for a synthesizer such as the FluidSynth plugin.

3.3 SwamiGUI

The Swami Graphical User Interface is what many users will interface with when editing or managing their instruments. The GUI interface is also object oriented and makes heavy use of Model View Controller functionality (change a parameter all views update, as the user expects). The GUI layout is also flexible so that it may be subdivided at will and interface elements can be changed simply by drag and drop.

Many of the interfaces (virtual keyboard, sample loop editor, splits, etc) are implemented using the GnomeCanvas. Although not perfect, it has helped to create flicker free zoomable widgets without having to pay attention to a lot of details.

3.4 Existing Plugins

Current plugins include the FluidSynth software synthesizer and the FFTune plugin which uses the FFTW library (Fastest Fourier Transform in the West) to assist users in tuning their samples.

4 Python Binding

The 3 library components of Swami each have their own Python binding. The most useful of these is libInstPatch. Using this binding it is possible to create or modify instruments in a scripted fashion. When dealing with files containing many instruments it can become rather tedious to perform an editing operation across all instruments. With a bit of Python knowledge the tediousness of these tasks can be greatly reduced. This also adds a level of ease for users who wish to extend the functionality of Swami at runtime (versus having to write in C). Example uses include: writing custom instrument export or import routines, auto defining key splits based on the optimal note ranges, batch parameter setting or instrument renaming, etc.

5 CRAM Instrument Compression

CRAM (Compress hybRid Audio Media) is a format that was created specifically for compressing instruments. There are a few instrument compression formats in use today, but I found them to be either proprietary, lack cross

platform support or restricted to a specific instrument format. CRAM is not constrained to only instruments though and would likely be useful for other file formats containing binary and audio data as well. The file format uses EBML (created for the Matroska multimedia container format) which was chosen for its compact design and extendability.

Binary compressors such as gzip, bzip2, etc are generally poor at compressing audio. FLAC (Free Lossless Audio Codec) provides lossless audio compression and often achieves much better compression ratios than a standard binary compressor. CRAM utilizes FLAC for audio portions of an instrument file and gzip for the rest. Each audio sample is compressed individually, taking advantage of the knowledge of the bit width and number of channels. The binary is concatenated and compressed as a single stream of data. Instrument compressors must be aware of the structure of the file they are compressing but the decompresser on the other hand is generic, and sees the resulting output as simply interleaved binary and audio segments. CRAM supports multiple files in a single archive and can preserve file timestamps.

In the future support for lossy audio compression may be added to the CRAM format. While lossy audio is generally undesirable for music composition, it might be nice for previewing an instrument online or used as the instruments embedded with a MIDI file on some future web site.

One of the technical difficulties of lossy compression of samples is the phase relationship of the decompressed signal to the original. If the sample contains any loops, they may end up with audible clicks due to differences in the data at the loop end points. It has not been determined yet how pronounced this effect is with a codec such as Vorbis or what is the most effective way to deal with it.

6 PatchesDB

A web based instrument database written in Python. This project was started for the purpose of creating a free, concise, online instrument database.

The current implementation is called the Resonance Instrument Database and can be found at:
http://sounds.resonance.org

Features include:

- User accounts
- Upload and maintain your own instruments
- Support for all libInstPatch supported formats (Python binding used for import)
- CRAM used for compression
- Comment and rating system
- Browse by category, author or perform searches

Future plans include the ability to browse, download and synchronize instruments and information between a local Swami client and an online instrument database; and preview instruments by selecting an instrument, choosing notes or a predefined sequence in a web form and server synthesizes the sound and streams it via an Ogg Vorbis stream.

7 Future Plans

Here are some future ideas for Swami. Many of these are only in the idea stage and may not become reality, but I like to dream about them anyways :)

- SwamiSH – The Swami Shell. A command line interface which could be used instead of the GUI. Imagine editing large instruments remotely or providing a visually impared individual the chance to do some serious instrument editing.
- TCP/IP Jam Sessions – Multiple Swami clients connect together, each user chooses the instruments they would like to play, clients download instruments as necessary, streamed MIDI data is then exchanged (sequencers, MIDI equipment, etc). Users hear the same synthesized instruments with minimal bandwidth overhead. Might be better suited to LAN or dedicated low latency networks, but might be fun and expiremental otherwise too.
- Vector Audio Waveforms – The concept of waveforms described by splines is

appealing to me. Think Structure Vector Graphics, but for audio. Design a waveform from scratch or trace an existing sampled waveform. Resulting synthesis would not suffer from sampling errors or looping continuity issues. Automation of spline control points in a periodic fashion may lead to a new method of audio synthesis. By no means all encompasing, but perhaps useful in its own way.

- GStreamer plugin – Add wavetable instrument support to the GStreamer multimedia framework.
- DSSI interface – A Swami DSSI plugin would likely make integration with other software much smoother. DSSI is an API for audio processing plugins, particularly useful for software synthesis plugins with user interfaces.
- C++ Bindings – For those who prefer C++ but would like to use or interface to Swami or its underlying libraries.
- Open Instrument Format – It may be desirable to design a new flexible and open instrument format. Something along the lines of an XML file which references external audio sample files and has support for vector audio (for envelopes, LFO waveforms, etc). A flexible yet simple synthesis model might be desirable.

8 Resources

- Swami – http://swami.sourceforge.net
- FluidSynth – http://www.fluidsynth.org
- CRAM http://swami.sourceforge.net/cram.php
- Resonance Instrument Database http://sounds.resonance.org
- DSSI - http://dssi.sourceforge.net
- EBML - http://ebml.sourceforge.net

9 Thanks

- Peter Hanappe and Markus Nentwig for creating FluidSynth. I'm happy to be a contributor to that project and will try and be a better maintainer ;)
- Ebrahim Mayat for his continued efforts in testing Swami and FluidSynth on OSX.
- Keishi Suenaga for his recent work on building Swami on win32, it was a dirty job, but somebody had to do it.
- Takashi Iwai for his work on the AWE wavetable driver which sparked my interest in creating an instrument editor.
- Countless others who have provided a place to stay during my travels in Europe.