Inside the Sweep Sound Editor

Conrad Parker `conrad@vergenet.net`

August 2001

Abstract

This paper outlines the development of new features in the sound editor Sweep. These new features, and similar requirements of other applications, motivated the development of a new audio sequencing and mixing library, Axel. This paper briefly describes Axel and the implementation of new Sweep features using this library.

1 Motivation

Sweep is a sound file editing application [?]. The core features include the basic editing operations of cut, copy, paste and splice with discontinuous selections, the ability to import standard LADSPA[?] effects plugins and a native plugin format for custom operations such as selecting sound regions by loudness threshold.

The original design goal of Sweep was to make a sound editor with similar interaction style to popular image editors such as the Gimp[?]. This would entail features such as layered editing, some notion of "transparency", and the ability to grab and move selected regions of sound. An initial attempt to implement these features directly in Sweep was made, however it soon became apparent that the code was growing with unnecessary complexity and that similar code was being rewritten to perform many different operations as outlined below.

1.1 Simple editing

Editing in Sweep operates on discontinuous selections, whereby a collection of small regions in a sound can be simultaneously cut out and then pasted elsewhere, or spliced into another sound. Thus the editing code in Sweep contained functions to merge, mix and splice lists with many small regions of sound.

1.2 Multiple layers with transparency

Layering provides the user with the ability to independently work on different aspects of a sound with some notion of occlusion or fading between them. For example, one could extract the high frequency components of a sound into a separate layer, which can then be enhanced and shifted in time before mixed back down into the original sound.

The introduction of layering required the coding of routines to mix layers, and to maintain synchronisation between layers during editing. These layers were designed to be able to each accommodate sparse collections of sound.

1.3 Floating selections

A floating selection, in the language of image editors, is essentially a special layer above all others in which the selected region exists. The floating selection can be interactively moved by the user. Thus, the implementation of floating selections required the ability to translate layers relative to each other.

1.4 Manipulation of large files

The initial releases of Sweep read an entire sound file into memory. This was clearly inefficient, especially as users were wanting to edit files of around 50MB (a 5 minute WAV file at CD quality).

Typically, a user might load up a 5 minute song file only to edit out a small section of silence at the beginning or end, to crop part of the file out or to apply an effect to a short region of it. For these operations it is not necessary to keep a copy of the entire file in memory, thus an obvious optimisation is to only store modified regions in memory and to read unchanged data off disk as needed.

1.5 Non-destructive editing

A similar request from users was the ability to perform non-destructive editing, in which many edits can be performed without touching the data on disk. However for large edits, and for persistence between editing sessions, this requires the paging and storage of modified regions on disk.

1.6 Editing of compressed files

Compressed file formats such as MP3[?] and Ogg Vorbis[?] are extremely popular and allow a 5 minute recording to occupy only a few megabytes. For small edits, it would be desirable to be able to work with compressed files without decompressing the entire file.

1.7 Parameter envelopes

Effects plugins such as those of LADSPA typically provide a number of parameters for the user to set, such as the desired depth of a reverberation or the cutoff frequency of a low-pass filter. Sweep already handled these by providing the user with a dialog to set parameter values when applying an effect, but it would be more interesting to allow the user to define a line or curve describing how each parameter should change over time. For example, rather than simply muffling a sound by applying a low pass filter with a particular cutoff frequency, the filter can be “opened up” over time to slowly reveal the higher frequency components like cymbals and melody.

2 Generalisation

There is quite a lot of overlap in the required functionality outlined above. Meanwhile, similar needs were encountered in other software such as Aube[?][?], a live music sequencer and effects program which needs the ability to record mix automation and song structure.

Unfortunately much of the existing data manipulation code in Sweep was tied to user interface callbacks or made assumptions about the user state so it could not be reused in other applications. It was apparent that an abstraction layer encompassing the features required of Sweep and Aube was required, and that it should be designed to allow generally useful manipulation of audio data.

2.1 Management of sparse, multichannel audio data

The most obvious abstraction was that of a container for multiple channels of audio data stored sparsely in a collection of memory regions. With a generic interface to access and manipulate such data, much of the merging and mixing code throughout Sweep could be centralised.

2.2 Generic caching

The handling of large files, non-destructive editing and editing of compressed files in Sweep all require the maintenance of memory and disk file caches. Similarly, the repetition of musical processing in Aube was a candidate for caching optimisations.

2.3 Parallel and serial processing

Aube implements an audio filter network, allowing the user to create arbitrary connections between many sound generation and processing elements. Typically such elements are chained together in series, but an element can also provide its output to a number of other elements in parallel. Mixers which read from multiple elements in parallel are also important in such an environment.

This kind of audio filter network is a popular mechanism for constructing complex arrangements of effects, however it can be cumbersome to deal with large networks.

Sweep's layering is a way of implicitly mixing audio data in series, and can be conceptually extended to the stacking of both data sources and sound filters in series. A complementary method for processing elements in parallel would be beneficial.

2.4 Mix automation

The term "mix automation" refers to the capability of an audio processing system to record changes in parameter values and the interconnection of processing elements over time. This is similar to the requirement for parameter envelopes in Sweep, and also to the ability to record the extents of regions over which effects are applied.

2.5 Low processing latency

In order to allow live manipulation in Aube, and to allow instantaneous previewing of large edits in Sweep, the audio subsystem must not introduce excessive processing latency; it must provide the ability to independently process tiny intervals of sound.

3 Axel

The generic audio processing requirements outlined above prompted the development of a library called Axel. Axel is an audio sequencing and mixing library that provides a multichannel, sparse audio data container (*streams*), a structured mixing abstraction (*decks*), and widely useful means of generating control data (via *envelopes*) and of caching audio data.

3.1 Streams, channels and chunks

Figure 1: Inside an Axel stream

The abstraction of multichannel audio data in Axel is known as a stream. The structure of a stream is shown in Figure 1. A stream may consist of multiple *channels*, each of which can consist of an arbitrary number of sparsely placed *chunks* of raw audio data. The channels are named with spatial names such as LEFT, RIGHT and CENTRE as required for common home, studio and theatre environments.

Generic routines are provided for mixing, multiplying and blending streams of data.

3.2 Decks, tracks, layers and sounds

The top level structured mixing abstraction in Axel is known as a deck. A deck contains a number of *tracks* which are mixed in parallel. Each track may contain a number of *layers* which are mixed from bottom to top in series. Finally, these layers each contain a sequence of *sounds* with transparency. This structure is illustrated in Figure 2.

Figure 2: Inside an Axel deck

The sequence of sounds in a layer can be indexed by samples, seconds or tempo. Sounds provide audio data from any instrument or effect source, and these sources can each be reused multiple times. A sound can even source its audio data from another entire deck, thus decks can be used to sequence other decks. In this manner effects can be applied to sequences of decks, and sequences of decks can be stored as higher level units such as verses and choruses in a music application.

3.3 Envelopes and mix automation

The information describing how a parameter changes over time appears as a generic data source. In order to create this mix automation information Axel provides linear and spline envelopes. However, parameters could alternatively be controlled by other means such as from a recording of physical slider values, from a sine wave generator, or from a deck constructed solely to generate interesting parameter values.

3.4 Processing latency and caching

All sound sources in Axel, including streams, decks and envelopes, implement a base set of functionality such as for seeking and for processing small regions of data. The requirement for low processing latency is met by these semantics, which are optimised for sequential processing but allow the evaluation of arbitrarily small temporal slices.

Axel also provides a generic caching abstraction which can be applied to any sound source.

4 Implementation of Sweep using Axel

By rewriting Sweep to use Axel for much of its functionality it was possible to implement the core features and to incorporate the newly desired features elegantly. Each sound file to be edited is assigned a deck with a single track. That track contains a number of layers, with the original sound data at the bottom, the user's floating selection and previews of effects on the top, and the user's layers in between.

4.1 Simple editing

The basic editing functions of cut, copy, paste and splice are handled by maintaining a separate deck to contain the cut buffer. The complexity of maintaining and merging a discontinuous selection is handled by Axel's stream structure.

4.2 Multiple layers with transparency

The desired layering features map directly to Axel's layering model, which of course comes as no surprise as Sweep's layering greatly influenced that portion of Axel.

4.3 Floating selections

A floating selection is maintained as the topmost layer. Moving this selection, even if discontinuous, is reduced to manipulating the start indices of the sounds in that layer. Mixing the floating selection back in is a layer mixing operation in Axel.

4.4 Large files and non-destructive editing

An Axel file source can provide its own sound data on demand, thus it is no longer necessary for Sweep to load an entire file into memory. If a small section of the file is edited, it is a implemented by adding the edited region to a layer above the file source with no transparency.

The data seen by the user during the editing session is directly sourced from the file outside of the edited region, and from memory or a separate disk cache within the edited region. Thus Axel's layering and caching can be used to implement non-destructive editing of large files.

4.5 Editing of compressed files

Compressed files appear to Axel only in their decompressed form. It is not possible to edit the compressed data fields via Axel, but it is possible to treat a compressed audio file as though it were uncompressed.

4.6 Parameter envelopes

Using Axel's linear and spline envelopes it is possible to vary effects parameters over time. The calculation of these envelopes is handled within Axel.

5 Conclusion

This paper outlined the problems faced by the new features of Sweep, a generalisation of these problems to a larger class of audio applications, the implementation of a library (Axel) to address these issues and its use in implementing new Sweep features.

Availability

Axel is available under the GNU Lesser General Public License at `http://www.vergenet.net/~conrad/axel/`