Audio and music production on Plan 9

Konstantinn Bonnet
qwx@nopenopenope.net

ABSTRACT

For well over a decade, the 9front fork of 4th edition Plan 9 has been expanding the audio capabilities of the system. It has brought support for modern hardware, has much improved audio codec coverage, and has significantly increased available tooling. 9front has been used successfully for gaming and for amateur music production in a variety of ways. This paper aims to provide an overview of the possibilities currently offered by the system, as well as its limitations.

1. The state of audio support

1.1. The audio device

Access to audio hardware on Plan 9 is provided via the audio device, #A, bound over `/dev` by default. Each configured audio card provides the same set of files: `audio` for reading and writing audio data, `volume` for setting volume levels, sampling rate and buffer size, `audioctl` for driver-specific configuration controls, and `audiostat` for driver-specific status information. To play audio, applications just need to open `/dev/audio` and write samples to it; to record audio, they only need to open and read from it. A common sample format and sampling rate have been chosen as default across the system: signed 16-bit integer (S16LE) stereo PCM audio, sampled at 44.1 kHz, which corresponds to CD Audio quality.

1.2. Device drivers

In 2011, a few months after the project began, 9front introduced driver support for Intel High Definition Audio. Written from scratch, the *hda* driver complemented *sb16* (Sound Blaster) and *ac97* (Realtek), which by then were already beginning to be relegated to legacy hardware as more and more consumer-grade hardware relied on integrated Intel chips. It has held up remarkably well since, requiring only minor changes in recent years.

USB audio is another important type of device, for which a new driver was added to 9front in the same year as *hda*, *nusb/audio*. It replaced the one from the 4th edition Plan 9 release, moving it to userspace along the rest of the USB device drivers. Over the years, it has gained support for USB Audio 2.0 and other capabilities. Both *hda* and *nusb/audio* implement the same interface, though the latter also provides some additional volume controls if available, such as bass boosting.

1.3. Play ought to be enough for everyone: codecs

Plan 9 included a decoder and encoder for MP3 files (resp. based on libmad and LAME) since at least its 4th edition release, as well as *games/juke*(7), a music jukebox. The codecs were supplemented in 9front by ones for WAV, FLAC, Ogg/Vorbis and even SUN audio files, largely ports using APE (the POSIX compatibility environment). Additional codecs exist out of tree, such as for Opus [1]. The majority of decoders now also

---

audio/μlawdec, later reworked into audio/sundec, was originally written by Erik Quanstro and imported into 9front.

support seeking to a given timestamp.

A simple script was added to leverage all available decoders, even supporting HTTP links and playlist files. *Play*(1) detects the input format of each of its arguments from their file extension, or by reading a small chunk of data and running *file*(1) on it. It then simply starts the corresponding decoder and pipes it (by default) to `/dev/audio`. Since *play*'s output can be redirected to a file including to standard out, it can be used as a general-purpose audio decoder in format conversion pipelines, or even in applications such as *games/doom*(1). With the addition of MIDI players and other tools, support for MUS and MIDI files has been added to *play*, something used to great effect in Doom as it now only needs to set up a pipe and launch the script to play music [2].

It is also trivial to implement additional features from common music players on top of *play*(1). Pause can be implemented by stopping and starting the decoder process launched by the script, which is easy to find since it is in the same process group. Similarly, the current track may be skipped by killing the process. Playlists and shuffling can be wrapped around the script as well.

Play(1)'s versatility demonstrates the simplicity yet power of this interface, where each codec simply reads data from standard in and transcodes it to standard out. More sophisticated players have been built around the same approach, such as *playlist*, a script which uses the plumber to control playback and implements playlists, shuffle, volume settings, seeking and others [3]. The playlist is manipulated by plumbing files and commands are sent by plumbing unicode runes. The script itself is below 250 lines of code.

1.4. Jukeboxes

While functional, *games/juke*(7) suffered from a number of issues, mostly in terms of usability. It was meant to be used on top of a music library, which in practice proved to be too cumbersome for many, who ended up preferring the simpler on-the-fly playback of tools like *play*. *Juke* itself was split into a playlist server, a browse server, and a graphical interface. Its graphical component was plagued by a number of bugs, and the overall complexity of the tool discouraged most from attempting a fix. Suffering from disuse, *juke* was eventually replaced by *audio/zuke*(1), a much more straightforward playlist-based graphical music player. A separate utility, *audio/mkplist* uses a newly written library *libtags* for reading metadata from supported audio formats, and can be piped into *zuke*. It dispenses with the file server interface, preferring a simpler human-readable playlist format instead. Under the hood, it still launches the audio decoders based on the metadata discovered by *audio/mkplist*. Supporting cover art, graphical controls and plumber messages, and integrating well with kbdtap-based shortcut systems such as *bar*(1), it has become the go-to music player for many.

1.5. Pcmconv: general-purpose sample format conversion

While the 44.1 kHz S16LE stereo PCM audio format chosen as default is common, it is not universal. For both playback and transcoding, a tool for converting to and from other formats is essential. *audio/pcmconv*(1) was added to 9front in 2012 to solve this, obsoleting a lot of duplicated resampling code from the codecs in the process. As an added benefit to simplifying the codecs, moving this code into a separate tool allowed them to benefit from improvements upon it at no extra cost. *pcmconv* has seen refinements in both quality, implementing band-limited resampling using a FIR (finite impulse response) filter [4], and capabilities, converting to and from various integer and floating-point formats.

Pcmconv has been used successfully within scripts and C programs for games (Wolfenstein 3D port [5]), audio hardware emulation (OPL2 and OPL3), and within NPE's implementation of the SDL2 audio API [6]. Converted audio data can either be piped back to the application, or written directly to `/dev/audio`. This pattern became so common that the conversion code was moved to a small library, *libpcm*, with *pcmconv* itself now just a shim around it. The library provides functions to compute conversion parameters and buffer sizes given an input and an output format, and to convert samples from an input buffer into an output one. This facilitates format conversions

particularly for code which does not simply output to /dev/audio.

1.6. Mixfs: audio multiplexing

Only one program can open /dev/audio for writing at a time. However, multiple programs playing audio simultaneously or even just holding the file open is an all too common case. For example, one may launch a game while a music file is playing. This also easily comes up in audio editing and production. *audio/mixfs*(1) is an answer to this challenge. It opts to preserve the same interface: rather than add more complexity driver-side, it serves a file tree which layers itself over the audio device's. By providing the same files and binding itself over /dev, applications can transparently use the same /dev/audio interface as before with no changes. Applications will only see the audio file provided by *mixfs*, which mixes samples from one or more writers and pushes the result to the audio device itself. The volume file provides access both to volume and delay settings of the audio device, and to controls specific to mixfs including its own soft mixing volume setting. Output devices can be set to a different sampling rate and format, in which case *libpcm* is used to convert the samples before pushing them.

Whether local or remote, there may be more than one audio interface available. *Mixfs* is also responsible for mapping the current output device to /dev/audio (and /dev/volume), by accepting output device switching commands on its own volume file. To avoid name conflicts with PCI audio drivers, the USB audio driver suffixes the files it serves with a unique device identifier.

Finally, *mixfs* handles reads from the audio file (ie. loopback) in the same way as writes, reading samples from the audio device then writing them out to one or more readers.

1.7. Screaming across the network

Exporting file trees is trivial in Plan 9. *Mixfs* makes sharing audio streams extremely simple, thanks to it handling multiple readers and writers. The following listing demonstrates how to implement a simple MP3 radio:

```
# hack: null output sink
; bind /dev/null /dev/audio
# start dedicated mixfs (also bound over /dev)
; audio/mixfs -s radio -m /mnt/radio
# make it available to user none
; chmod o+rw /srv/radio
# create listener script
; cat <<'!' >radio.rc
#!/bin/rc -e
rfork nefs
mount /srv/radio /mnt/radio
dd -bs 32k -if /mnt/radio/audio
!
; chmod +x radio.rc
# start listener
; aux/listen1 -p 50 'tcp!*!19000' ./radio.rc &
# transcode files to mp3
; { while() play -o /fd/1 /lib/music/mycoolalbum } \
    | audio/pcmconv -i s16r1c1 -o S16r1c1 \
    | audio/mp3enc >/dev/audio
```

Clients can simply use *aux/trampoline*(8) to connect and receive MP3 data directly:

```
; aux/trampoline tcp!$addr!19000 \
    | audio/mp3dec >/mnt/mix/audio
```

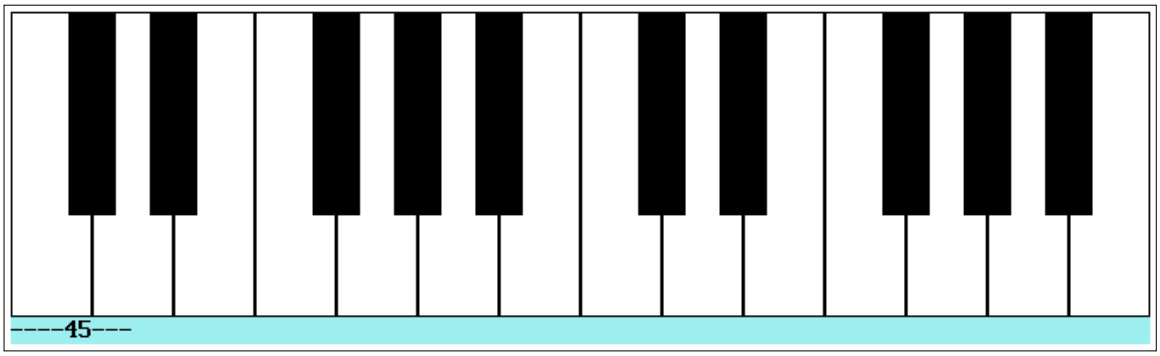
In addition to simple exports, 9front implements the Scream network protocol to transmit PCM audio over the wire to and from a virtual sound card on Windows machines via *screamsend*(1) and its *screanrecv*(1) counterpart.

2. Music production

Because there are no layers of abstraction on top of `/dev/audio`, creating and playing sounds on Plan 9 is very straightforward. Available methods can be broken down into several categories.

2.1. Software synthesis

There is more than one example out there of generating sounds and bytebeats directly in C. *Games/midi*(1) samples from square wave functions with frequencies corresponding to currently played notes. Several software pianos employ the same technique and even offer a selection of waveforms to choose from [7,8,9].



Software piano.

In a similar vein, *games/geigerstats*(1) outputs an amount of random noise proportional to the system load figure it reads from `/dev/sysstat`. There are many examples of bytebeats as well [9,10].

2.2. Hardware and emulation

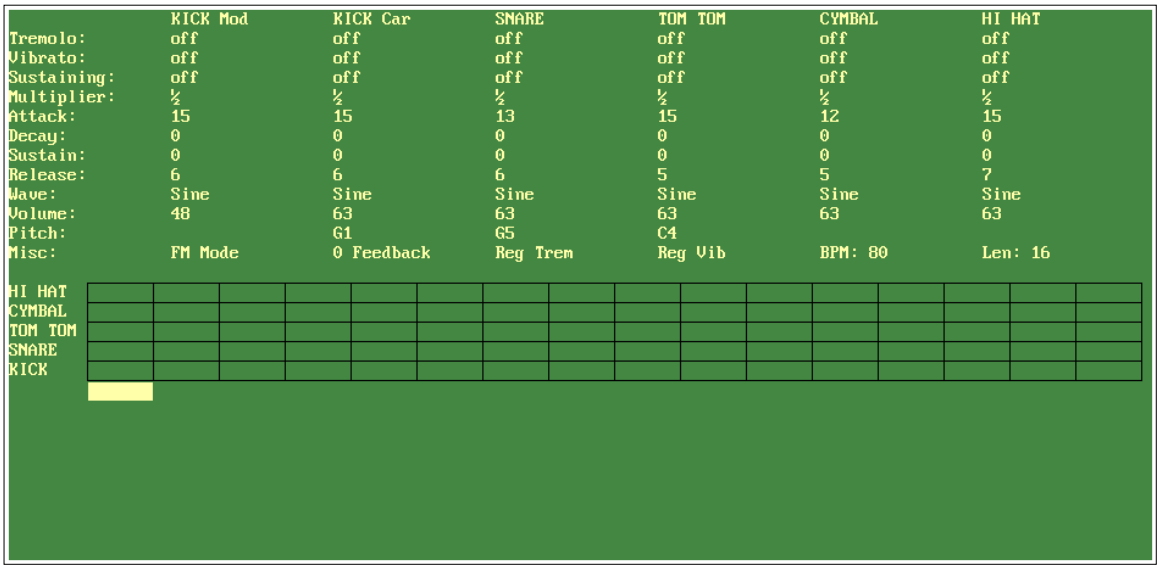
One way of producing music is to interface directly with hardware synthesizers, or sound chips providing some kind of synthesis. Outside of USB interfaces, there currently are no drivers specific to such hardware. However, many popular sound chips have been emulated.

Two such chips are the YM3812 (aka OPL2) and YMF262 (aka OPL3), relatively affordable chips which provide FM synthesis. Sound cards using those chips such as Adlib or some Sound Blaster models have seen wide usage in game music during the late 80's and early 90's. *Games/opl3*(1) is an OPL3 chip emulator, which reads register-value-delay tuples on its standard input (the brackets indicate the size of the field in bytes):

```
register[2] value[1] delay[2]
```

When the time delay value is above 0, *opl3* samples the emulated hardware and produces a corresponding amount of PCM audio samples on standard out. It's used primarily by *games/dmid*(1), a MIDI player originally written for Doom, and which configures OPL3 instrument banks from game data files. Since 9front now ships with the shareware version of Doom, *dmid* and *opl3* are now used in the *play*(1) script for MIDI files by default.

Several tools are available for editing OPL3 instrument banks [12].



opldrums: drum bank editing

There also are several OPL3-based trackers, both graphical and text-based, which have been used with good results [13].

An earlier OPL2 emulator, *opl2,* exists as well [14]. It has been used successfully to play music files from Wolfenstein 3D, its sequel Spear of Destiny, and Commander Keen, all of which store music and sound effects in a format which *opl2* can read directly. It has also been used for Ultima 6 music, for which a decompressor and decoder were written from scratch [15].

2.3. MIDI tools

MIDI is roughly speaking a standard for a physical and digital communications protocol, in wide use in the music industry for connecting together and driving electronic music hardware, as well as other studio and stage-related equipment. MIDI events can be streamed in real time at a certain rate, or read from a file where events may be spaced far apart, slightly changing how timing is encoded and handled. 4th edition Plan 9 did not ship with much MIDI-related code, but 9front has slowly been growing an entire ecosystem.

There are several MIDI players available, all using different synthesis techniques. As mentioned above, *games/midi*(1) simply produces square waves, while *games/dmid*(1) leverages OPL3 emulation via *games/opl3*(1) and preconfigured instrument banks. Of note is also *games/sf2mid*, which uses SF2 soundfonts. Soundfonts can be used in lieu of a sound chip emulator to reproduce instrument sounds with variable degrees of accuracy. *sf2mid* can be hooked up to Doom with a Roland SC55 soundfont, recreating the most well-known rendition of its soundtrack [2].

There currently is a lack of MIDI editing software, but there have been multiple attempts at both text-based and graphical MIDI file editing. Text-based approaches use either a horizontal or vertical tracker-like format, specifying note events and a delay. One such attempt is *mst,* interpreting syntax such as the following, which sets a tempo, the instruments for the first two channels (channel 9 is hardcoded for percussions), and several measures in 4/3 time [16]:

```
t 124
i 0 34
i 1 19
4/3 0e3 9d3 9a#3
4/3 +
4/3 0e3 1a#5 9d3 9a#3
4/3 0b3 1b5 9a#3
4/3 0a#3 9a#3
4/3 0b3 1e6 9a#3
```

Notes are held only for one measure unless the next event is + (hold). Additional symbols were used to specify effects such as slides or bends. Ultimately, the tool was never completed, as microdelays and uneven timings proved difficult to encode nicely. Other approaches have been attempted as well. *abcmidi* together with *abcm2ps* have been ported and mostly fulfill this need [17], but no native tools have been completed yet. A guitar tab editor in similar spirit is rumored to be in progress.

There are no USB MIDI drivers, but data can be read from and written to specific endpoints directly. *dmid* and a few other tools support streaming MIDI as input. This allows them to be hooked up to MIDI instruments, for example via USB:

```
; games/dmid -s /dev/usb/ep10.1/data \
    | games/opl3 -s >/dev/audio
```

Here, *dmid* reads a MIDI stream from an exposed endpoint and writes them to the OPL3 emulator, turning them into audio. The inverse operation is also possible. *games/mid2s*(1) streams a MIDI file to standard out and can be used to play a file on a hardware synthesizer that reads MIDI via USB:

```
; games/mid2s sshock01.mid >/dev/usb/ep11.2/data
```

Mkey is a software piano which can stream MIDI events on standard out (and nothing else) [7].

Multiple USB MIDI dongles, which provide a USB interface to hardware which does not have one, have also been tested on 9front and work in an identical manner.

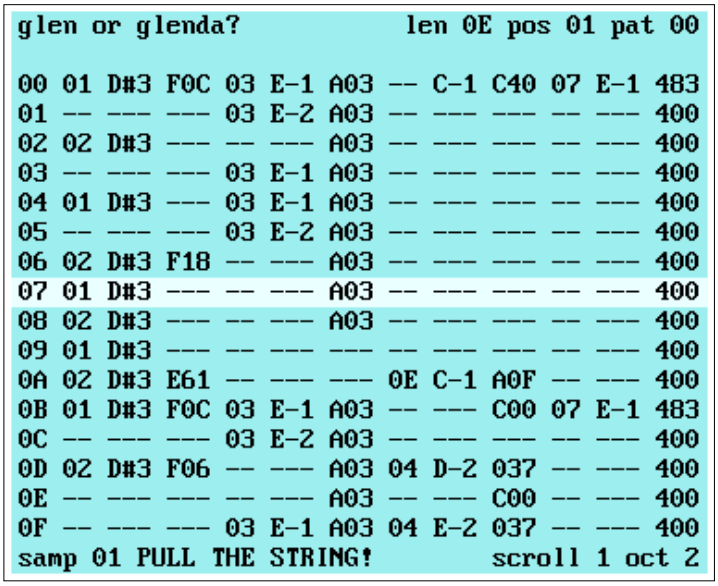
There is ongoing work to add an easier to deal with abstraction for USB MIDI hardware, as well as to add tools for filtering events, remapping channels, recording to file and so on.

2.4. Sample-based trackers

Trackers provide a vertical, tabular view of a musical partition, where each row is a time step (for example a quarter note) and each column a different instrument. Each cell can turn on or off a note at a specific pitch, as well as set one or more effects, such as stereo panning, fade in/out, slides, bends, instrument changes, and so on. A page constitutes a pattern, often 4 bars long. Patterns are chained together to form entire tracks. Instruments may use digital samples, represent an instrument patch of a hardware chip, or even just forward events to external hardware via MIDI. Trackers have been a popular way of producing music especially on low-powered hardware.

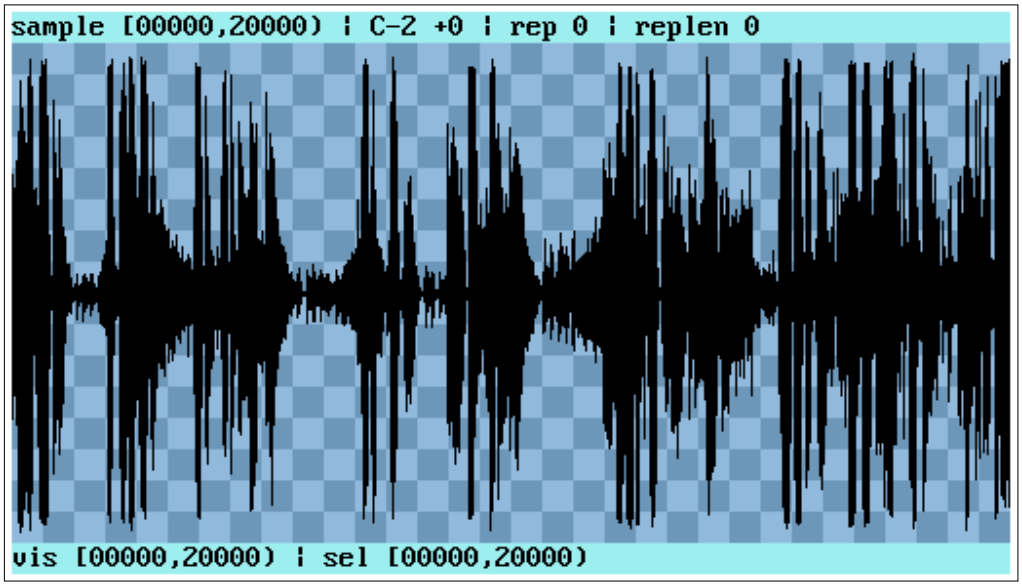
Mod/track, was the first sample-based tracker to appear for Plan 9, written from

scratch and based on the Amiga ProTracker format [18].



mod/track playing an example track, glen.mod.

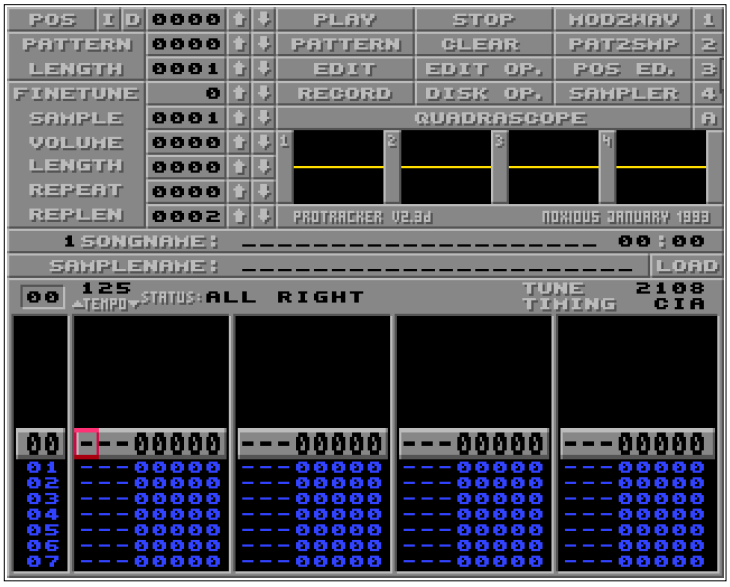
It was used to staggering effect for several tracks, and also provided a sampler and graphical sample editor.



mod/edsamp displaying a waveform.

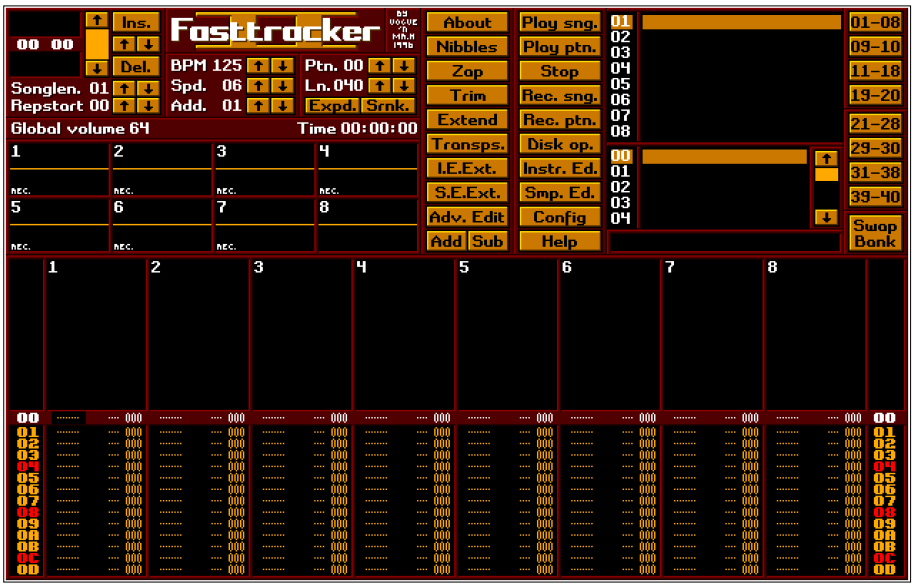
The advent of NPE allowed for easy ports of many cross-platform projects, including two fully-featured trackers, *ft2-clone* and *pt2-clone*, reimplementations of resp.

Fast Tracker 2 and ProTracker 2 [19,20].



pt2-clone: ProTracker 2 reimplementation.

The initial port of *ft2-clone* was later forked again and updated against upstream, while fixing several issues related to slow file loading and adding MIDI input support [26]. This fork has been used quite extensively for a number of tracks.



ft²: Fast Tracker 2 reimplementation port fork.

In addition, a port of *m8c*, a cross-platform client for the Dirtywave M8 hardware tracker is now also in progress [27]. While relying on an Teensy 4.1 flashed with firmware from Dirtywave, it offers a far more powerful environment than other trackers, complete with multiple sound synthesis engines, sampling, and driving external MIDI

equipment.

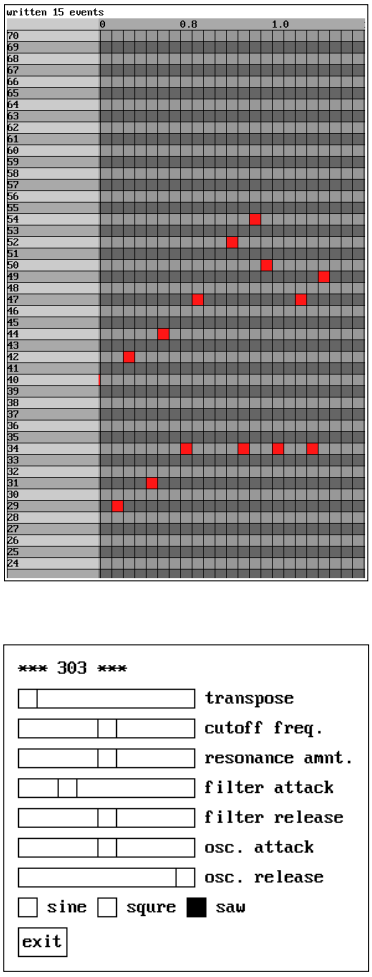
m8c: instrument tweaking.

Finally, several decoders for MOD files and most other similar formats have been ported, including *dumb* (dynamic universal music bibliotheque), *ft2play*, and *itdec*, complementing the array of MIDI players and tracking tools.

2.5. Environments, libraries, frameworks

One of the earliest examples of a suite of audio utilities is ment's *gui*, which provides tiny utilities to create waveforms, oscillators, sequence a track, and so on. The graphical interfaces themselves were built from a textual description:

```
; cat 303.gui
; cat 303.gui
text '*** 303 ***'
nl
slider trans
text 'transpose'
nl
slider cutoff
text 'cutoff freq.'
nl
[...]
; 303 <{roll '{seq 70 -1 24}} <{gui <303.gui} >/dev/audio
```



Gui: sequencer and instrument parameters.

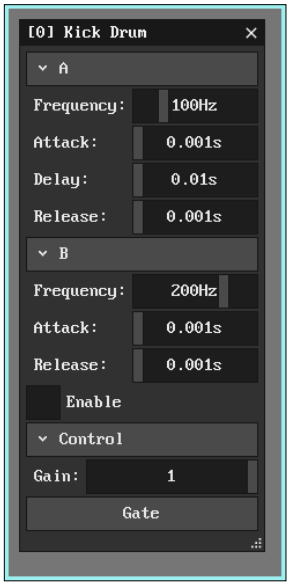
While quite neat, it is very much unfinished.

Neindaw is similar in spirit [27]. Each instrument is a filesystem, on top of which graphical interfaces may be built but are optional. The instruments themselves are implemented in Faust, a functional programming language for sound processing and synthesis. Graphical interfaces mirror the shell commands used to interact with each instrument and unlike *gui* are intrinsically linked to the instrument itself.

```
// Simple kick drum
declare name "Kick Drum";
declare group "Synthesis";
import("stdfaust.lib");

aFreq = hslider("v:A/[0]Frequency[unit:Hz]", 100, 10, 400, 5);
aA = hslider("v:A/[1]Attack[unit:s]", 0.001, 0.00001, 0.2, 0.001);
aD = hslider("v:A/[2]Delay[unit:s]", 0.01, 0.00001, 1.0, 0.001);
aR = hslider("v:A/[3]Release[unit:s]", 0.001, 0.00001, 1.0, 0.001);
bA = hslider("v:B/[1]Attack[unit:s]", 0.001, 0.00001, 0.2, 0.001);
bR = hslider("v:B/[2]Release[unit:s]", 0.001, 0.00001, 1, 0.001);
bFreq = checkbox("v:B/[3]Enable") * hslider("v:B/[0]Frequency[unit:Hz]", 200, -400, 400, 5);

process = os.hs_oscsin(aFreq + bFreq*en.ar(bA, bR, 1)) * en.adsr(aA, aD, 0.000001, aR, 1.0);
```



Neindaw: defining instruments and configuration interfaces together.

be*Neindaw*can *ORC*Λ, an esoteric programming language and livecoding environment [28], which in turn can be coupled with OPL3 emulation via *orcopl*[11]. While impenetrable at first, this environment is quite capable, if not tied to a specific setting and approach to music making.

Some libraries have been ported as well, but have seen little use, including *sox*, *soundpipe*, *sndkit*, *libsamplerate*, and *sndio9*.

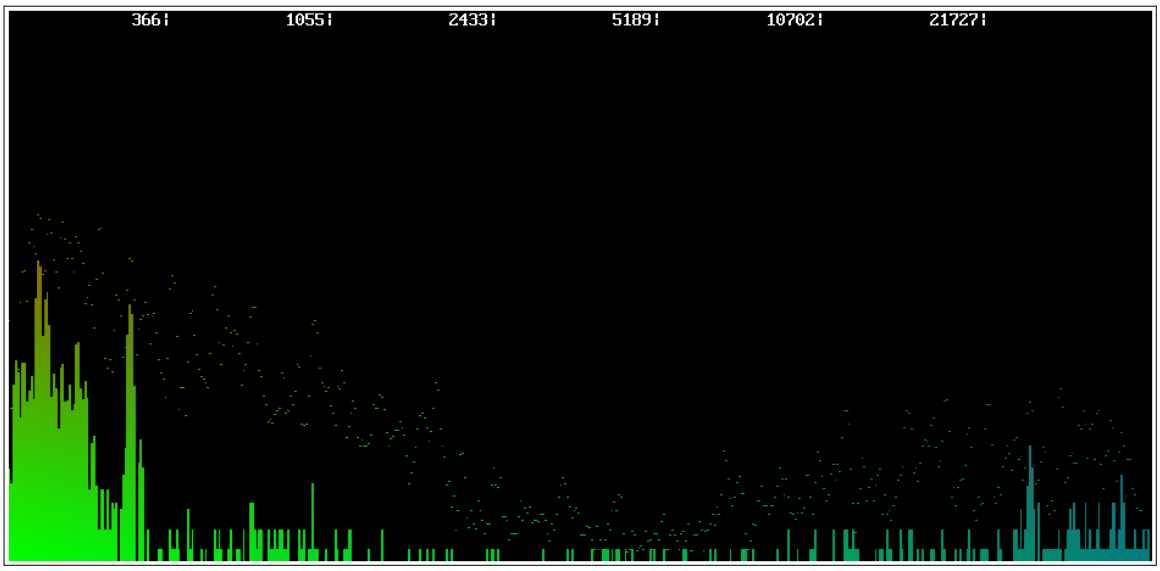
3. Analysis and editing

The area where Plan 9 audio tooling is the most lacking is analysis and post-processing.

3.1. Visualization and transformation

Existing visualization programs mostly revolve around displaying a time or frequency domain representation of the audio signal, although some programs for generating colorful visuals for livecoding also exist. *freqgraph* draws a spectrogram in real time of audio it passes through from standard in to */dev/audio*[21]:

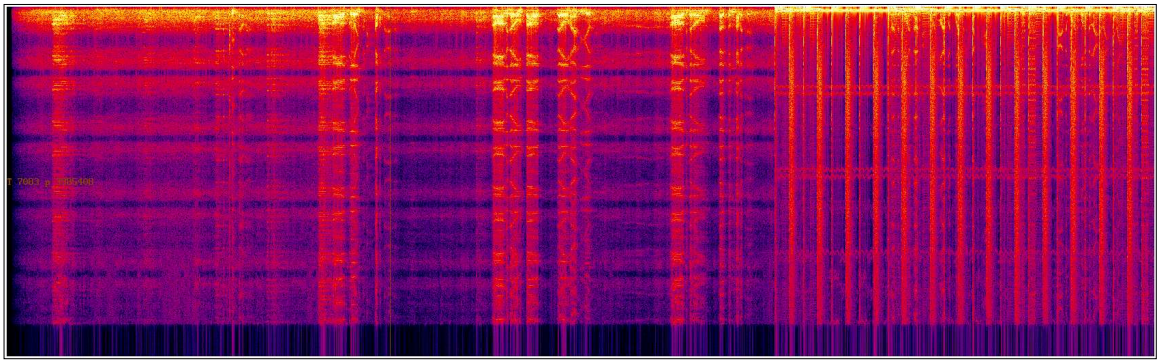
```
; play -o /fd/1 glen.mp3 | freqgraph
```



Fregraph (modified) spectrogram showing peaks as dots.

freqgraph's FFT size can be changed at runtime. It has shown to be invaluable for audio recording at least.

Fplay is another spectrogram program forked off of *audio/pplay*(1) (described below) [22]. It computes a static 2D spectrogram over its entire input.



fplay spectrogram of glen.mod.

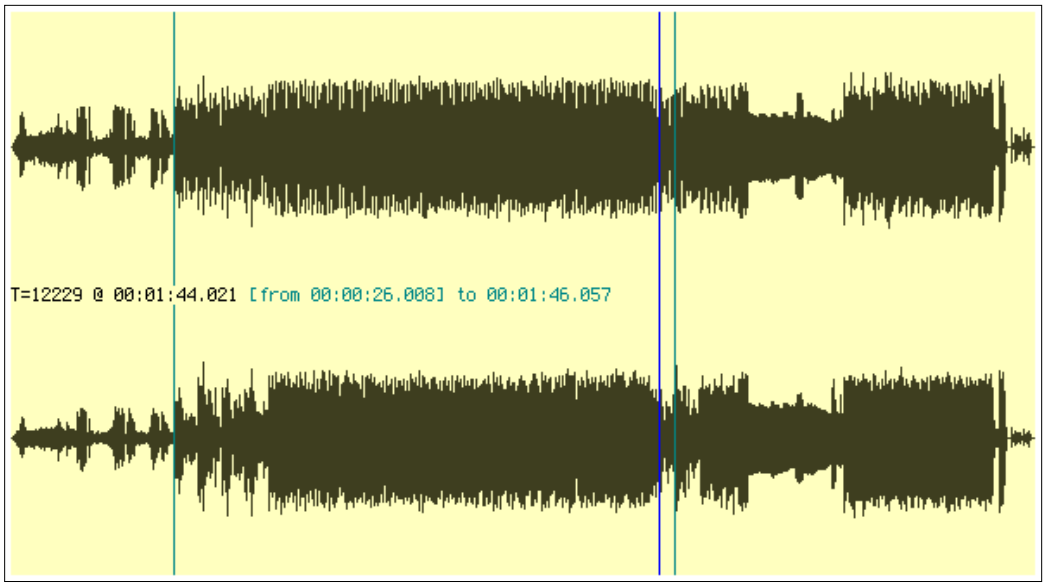
Several bugs exist, including the lack of any filtering.

While there are a few visualization tools availble, not much can be done with the information gained, and the tools themselves are relatively simplistic, unsuitable for more precise analysis.

There are several general-purpose transformation tools available, which can be used in pipelines. *Pcmenv* computes a fade in or fade out envelope [23]. *Pcmrev* reverses the order of its input samples. *Pcmmix* mixes one or more sample buffers together, and can multiply all signals by a constant as a simplistic volume normalization method [24]. *Audio/stretch* implements time-domain harmonic scaling, for changing pitch without affecting duration and vice versa [25].

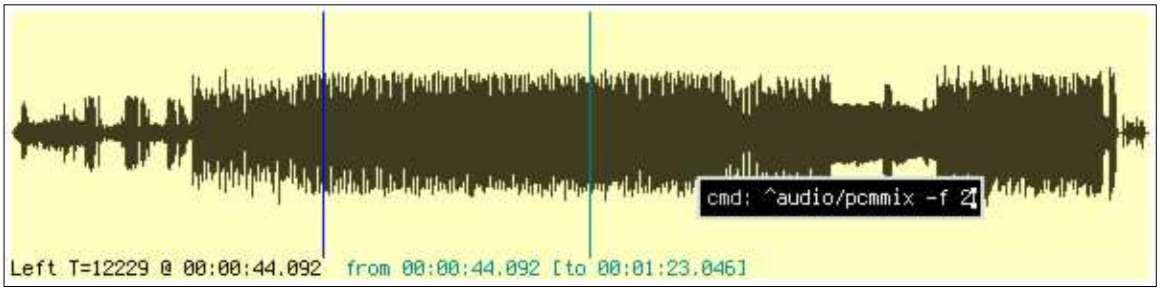
3.2. Pplay: graphical audio editing

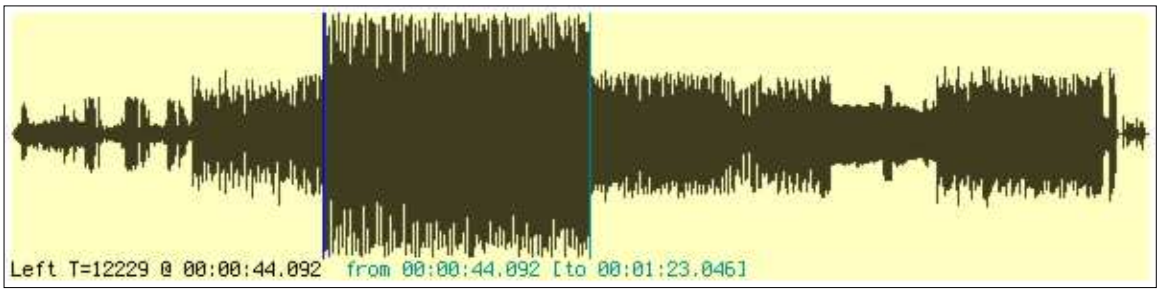
Audacity has been one of the go-to Open Source software for simple audio editing. However, its user interface leaves room for improvement, and the size and scope of the project have grown considerably. *Pplay* is a graphical PCM audio editor inspired in part by *Audacity*[24]. It reads audio from its standard in, then displays the corresponding waveform. The view can be zoomed in and panned with the keyboard or mouse at will. *Pplay* accepts simple commands for controlling its behavior and changing the contents of the buffer. Most commands act upon a selected region of audio which can be set with the mouse or via commands.



pplay stereo waveform of glen.mod. A loop section has been selected (green cursors).

The current selection is termed the **dot**. Commands include cut, copy and paste, but also I/O commands similar to *sam*(1): the | command pipes the dot to an rc command, while ^ pipes and replaces the dot by the result of a command.





Before and after screenshots of a ^ command.

Instead of a plugins system, *pplay* relies on small external programs such as *pcmenv* to modify parts of a buffer and read back the result. It also sports an infinite undo/redo stack. A dot can be "forked" into a new instance of *pplay* as well. Finally, audio can be read into *pplay* until interrupted, with the < command.

4. Conclusion: The Plan 9 DAW

Modern music production heavily relies on DAWs (digital audio workstation) software, which are typically massive and proprietary applications. Through elaborate plugin systems such as VST, commercial software offers high-fidelity hardware and software synth emulation, and any kind of pre- and post-processing imaginable. While offering virtually limitless capabilities and fantastically powerful facilities, it often also requires a high amount of processing power, exacerbating latency issues. Needless to say, even Open Source DAWs are unlikely to ever appear on Plan 9 with full support, simply due to the often colossal number of dependencies of both the application and the plugins. Since 9front now has a virtual machine hypervisor, *vmx*(3), it is possible to instead use a DAW in a limited capacity from within a virtual machine, piping audio separately. Connecting via VNC or Remote Desktop to another machine, combined with Scream or any other way of streaming audio over the network, are another easy alternative.

A Plan 9-style DAW would probably most resemble *gui* or *neindaw* as described above, a somewhat loose collection of tools and scripts, possibly with some dedicated language for audio processing. This would be the best way to leverage already existing work, and to reduce redundancy between tools as much as possible. NPE also makes it possible to more easily port more elaborate tools which cannot easily be reimplemented, though they likely wouldn't integrate as well with the rest of the system.

5. References

[1] K. Bonnet, ''Adding Opus support to Plan 9'', *http://nopenopenope.net/posts/opus*, retrieved March 2026.

[2] K. Bonnet, ''Plan 9 Doomed: a review'', 11th International Workshop on Plan 9, Paris, 2025.

[3] umbraticus, ''playlist, a plumber-based audio player'', *http://runjimmyrunrunyoufuckerrun.com/rc/playlist*, retrieved March 2026.

[4] Julius Orion Smith III, ''Digital Audio Resampling Home Page'', *https://ccrma.stanford.edu/~jos/resample/*, retrieved March 2026.

[5] K. Bonnet, ‘‘Wolfenstein 3D port’’, *http://shithub.us/qwx/wl3d/HEAD/info.html*, retrieved March 2026.

[6] J. Moody and S. Haflinudottir, ‘‘Portability has outgrown POSIX’’, 10th International Workshop on Plan 9, Philadelphia, 2024.

[7] K. Bonnet, ‘‘mkey, software piano’’, *http://shithub.us/qwx/mkey/HEAD/info.html*, retrieved March 2026.

[8] umbraticus, ‘‘piano, software piano with selectable waveforms’’, *http://runjimmyrunrunyoufuckerrun.com/src/piano.c*, retrieved March 2026.

[9] Amavect, ‘‘piano32/pianoJI, and byte beats’’, *https://git.sr.ht/~amavect/music*, retrieved March 2026.

[10] umbraticus, ‘‘Byte beats’’, *http://runjimmyrunrunyoufuckerrun.com/audio/bytebeat/*, retrieved March 2026.

[11] umbraticus, ‘‘orcopl, ORCΛ to OPL3’’, *http://runjimmyrunrunyoufuckerrun.com/src/orcopl.c*, retrieved March 2026.

[12] umbraticus, ‘‘opldrums, op2ed: OPL2/3 instrument bank editors’’, *http://runjimmyrunrunyoufuckerrun.com/src/op2ed.c*, *http://runjimmyrunrunyoufuckerrun.com/src/opldrums.c*, retrieved March 2026.

[13] umbraticus, ‘‘opltracker: text-based OPL3 tracker’’, *http://runjimmyrunrunyoufuckerrun.com/src/opltracker.c*, retrieved March 2026.

[14] K. Bonnet, ‘‘opl2, OPL2 emulator’’, *http://shithub.us/qwx/opl2/HEAD/info.html*, retrieved March 2026.

[15] K. Bonnet, ‘‘u6m, Ultima 6 m format decompression and decoding’’, *http://shithub.us/qwx/u6m/HEAD/info.html*, retrieved March 2026.

[16] K. Bonnet, ‘‘mst, plaintext-based MIDI synthesis’’, *http://shithub.us/qwx/mst/HEAD/info.html*, retrieved March 2026.

[17] umbraticus, ‘‘ABC tools port’’, *http://runjimmyrunrunyoufuckerrun.com/src/foreign/abcmidi/*, retrieved March 2026.

[18] umbraticus, ‘‘mod, Amiga-style MOD tracker’’, *http://runjimmyrunrunyoufuckerrun.com/src/mod.tgz*, retrieved March 2026.

[19] S. Haflinudottir, ‘‘ft2-clone port’’, *http://shithub.us/sigrid/ft2-clone/HEAD/info.html*, retrieved March 2026.

[20] S. Haflinudottir, ‘‘pt2-clone port’’, *http://shithub.us/sigrid/pt2-clone/HEAD/info.html*, retrieved March 2026.

[21] umbraticus, ‘‘freqgraph, real-time spectrogram’’, *http://runjimmyrunrunyoufuckerrun.com/src/freqgraph.c*, retrieved March 2026.

[22] K. Bonnet, ‘‘fplay, static spectral visualization’’, *http://shithub.us/qwx/fplay/HEAD/info.html*, retrieved March 2026.

[23] umbraticus, ‘‘pcmenv and pcmrev, fade envelope and reversal’’, *http://runjimmyrunrunyoufuckerrun.com/src/pcmenv.c*, *http://runjimmyrunrunyoufuckerrun.com/src/pcmrev.c*, retrieved March 2026.

[24] K. Bonnet, ‘‘pplay, visual PCM audio editor’’, *http://shithub.us/qwx/pplay/HEAD/info.html*, retrieved March 2026.

[25] K. Bonnet, ‘‘audio/stretch, TDHS port’’, originally by David Bryant, *http://shithub.us/qwx/audio-stretch/HEAD/info.html*, retrieved March 2026. [26] K. Bonnet, ‘‘ft², fork of 9front ft2-clone with MIDI support and fixes’’, *http://shithub.us/qwx/ft%C2%B2/HEAD/info.html*, retrieved March 2026.

[26] K. Bonnet, ‘‘m8c port’’, *http://shithub.us/qwx/m8c/HEAD/info.html*, retrieved March 2026.

[27] S. Haflinudottir, ‘‘neindaw, a wip DAW for plan9 using faust’’, *http://shithub.us/sigrid/neindaw/HEAD/info.html*, retrieved March 2026. [28] S. Haflinudottir, ‘‘ORCΛ port’’, *http://shithub.us/sigrid/orca/HEAD/info.html*, retrieved March 2026.