Advanced Csound

Michael Gogins
http://michaelgogins.tumblr.com

Irreducible Productions
New York

NYCEMF 2022
Workshop materials: https://github.com/gogins/michael.gogins.studio/blob/master/2022-NYCEMF/advanced_csound.zip

Agenda

Who is this for?

- This workshop is for anyone who uses Csound to *actually make music*.
- That includes beginners, expert users, and even programmers.
- I provide examples/exercises that are pre-written to speed things up. You should run them, but you should not have to debug them.
- The examples are written for macOS, Linux, and Windows, but the ideas also apply to Android and WebAssembly.

What is Csound?

- *Csound is a programmable software sound synthesis system with a runtime compiler.*
- Csound was written in 1985, so has *more unit generators* than later SWSS such as SuperCollider or Max.
- Csound runs on desktops, mobile devices, single-board computers, and Web browsers (WebAssembly).
- Csound has a straightforward "C" API (csound.h).
- The Csound API has interfaces in C++, Python, Lua, Lisp, and other languages. *You can run Csound inside other languages.*
- Csound has opcodes for hosting external plugins and even external languages (Python, C++, Lua). *You can run other languages inside Csound.*

A Moderately Sophisticated Piece

- I will demonstrate my piece ***Red Leaves v8.4.3.5.stereo***, which was performed here last Wednesday evening.
- I coded this piece for fixed media using iterative refinement.
- It is a single .csd file, that also embeds an HTML page with JavaScript, and some C++ source code.
- I committed every variant to my GitHub repository, so I could restore previous versions if needed.
- This piece has a number of dependencies, so I will just play it and talk about it.
- You should open the piece in your editor and follow along as I speak.
- Ask any question at any time!

Getting Csound

Install with brew on macOS, apt on Linux, vcpkg on Windows, or google for downloads.

- Install Csound for your desktop (https://csound.com/download.html). For the *current* release on Linux, build from sources (https://github.com/csound/csound/blob/develop/BUILD.md).
- Install the Audacity soundfile editor (https://www.audacityteam.org/download/).
- Consider building the Csound external plugins repository (https://github.com/csound/plugins) from sources and installing it.

Configuring

- The `csound` program should be in your environment's `PATH` variable.
- For the `vst4cs opcodes`, configure Csound to load them from their build or download directories.
- For the Csound plugins repository, build them locally and install using `sudo make install`.
- Shown for macOS, but the same variables should be set with appropriate values on Linux or Windows.

```
export OPCODE6DIR64="/Users/michaelgogins/Downloads"
export RAWWAVE_PATH="/opt/homebrew/Cellar/stk/4.6.2/share/stk/rawwaves"
```

M1 Macintosh

- Apple's arm64 M1 processor is fantastic, but can create problems of software incompatibility.
- Some software distributed for Csound is built for x86-amd or x86-64 architecture only, and won't work with software built for arm64 only.
- Currently, if you have an M1 Mac, CsoundQt and the plugins in the Risset package manager don't work.
- If you have an *Intel* Mac:
  - Install CsoundQt (`http://csoundqt.github.io/pages/install.html`).
  - Install the Risset package manager for Csound plugins (`https://github.com/csound-plugins/risset`).

Other useful software

Some of these have their own additional pre-requisites.

- Python 3.9 (https://www.python.org/downloads/).
- ABX comparator (https://github.com/gogins/python-abx).
- Faust DSP language (https://github.com/grame-cncm/faust/releases).

Running Csound

Csound does not have a built-in code editor or visual patcher like SuperCollider or Max.

- You can run Csound from the command line.
- You can run Csound from a general-purpose but customized code editor.
- You can run Csound from a purpose-built Csound code editor such as CsoundQt.
- In this workshop, we use any text editor and the command line as the lowest common denominator.

Running Csound from the command line

Example

Change to the `exercises/commandline` directory of your workshop directory.

- Execute `csound electric-priest.csd` to make sure that Csound runs. Also make sure that the maximum level at the end is negative so you won't blow out your speakers!
- Execute `csound --devices` to list your system's audio devices.
- Execute e.g. `csound electric-priest.csd -odac0` to render the piece with real-time audio.

Other code editors for Csound

- SciTE as customized in csound-extended (https://github.com/gogins/csound-extended/blob/master/playpen/.SciTEUser.properties).
- gedit as customized in csound-extended (https://github.com/gogins/csound-extended/blob/master/playpen).
- blue (https://blue.kunstmusik.com/).
- Cabbage (https://www.cabbageaudio.com/).
- Visual Studio Code (https://code.visualstudio.com/) with Csound extension (https://github.com/csound/csound-vscode-plugin).

The playpen concept

- A "playpen" is where you can play with things and not worry about breaking them.
- In computer music we have an iterative work cycle:
  - *Edit* a piece.
  - *Compile* the piece. If it doesn't compile, go back to *Edit*.
  - *Run* the piece to render audio. If it doesn't run, go back to *Edit*.
  - *Listen* to the piece. If you don't like it, go back to *Edit*.
  - When you don't need to go back to *Edit*, the piece is done.
- We will make a playpen that makes these steps as fast and automatic as possible, so that we can concentrate on making music.
- We can make our playpen by customizing a code editor, or by using a code editor specifically designed for Csound.

A command-line playpen

I present a command-line version because CsoundQt does not work with brew Csound on M1 Macs.

- Requires Python 3.
- Install `playpen.py` (https://github.com/gogins/csound-extended/blob/master/playpen/playpen.py) in your home directory, and make it executable.
- Install `playpen.ini` (https://github.com/gogins/csound-extended/blob/master/playpen/playpen.ini) in your home directory, and customize it for your installation.
- `~/playpen.py help` prints documentation.

Main playpen commands

Example

Change to the `exercises/playpen` directory of your workshop directory.

- Execute `~/playpen.py` to view help.
- Execute `~/playpen.py csd-audio oblivion.csd` to run `oblivion.csd` and hear it in real time.
- Execute `~/playpen.py csd-play oblivion.csd` to run `oblivion.csd` , render it to a soundfile, post-process it, tag it with your metadata, and open it in Audacity.

A Moderately Sophisticated Piece (workflow)

- All code is edited in one editor and in one file.
- Instantly hear the results of any edit.
- Instantly see a piano roll view of the generated score.
- See all Csound diagnostic messages.
- Tweak sensitive parameters of instruments during performance.
- Produce finished, tagged soundfiles in various formats with one click.

Best practices for core Csound

- Learn to hear electroacoustically.
- Optimize for audio quality.
- Learn the most useful builtin opcodes.

Hearing electroacoustically

- Before anything else, one should know what one is hearing *objectively*.
- One should establish the limits of one's hearing, because Csound will exceed them.
- One should understand typical artifacts of digital audio.

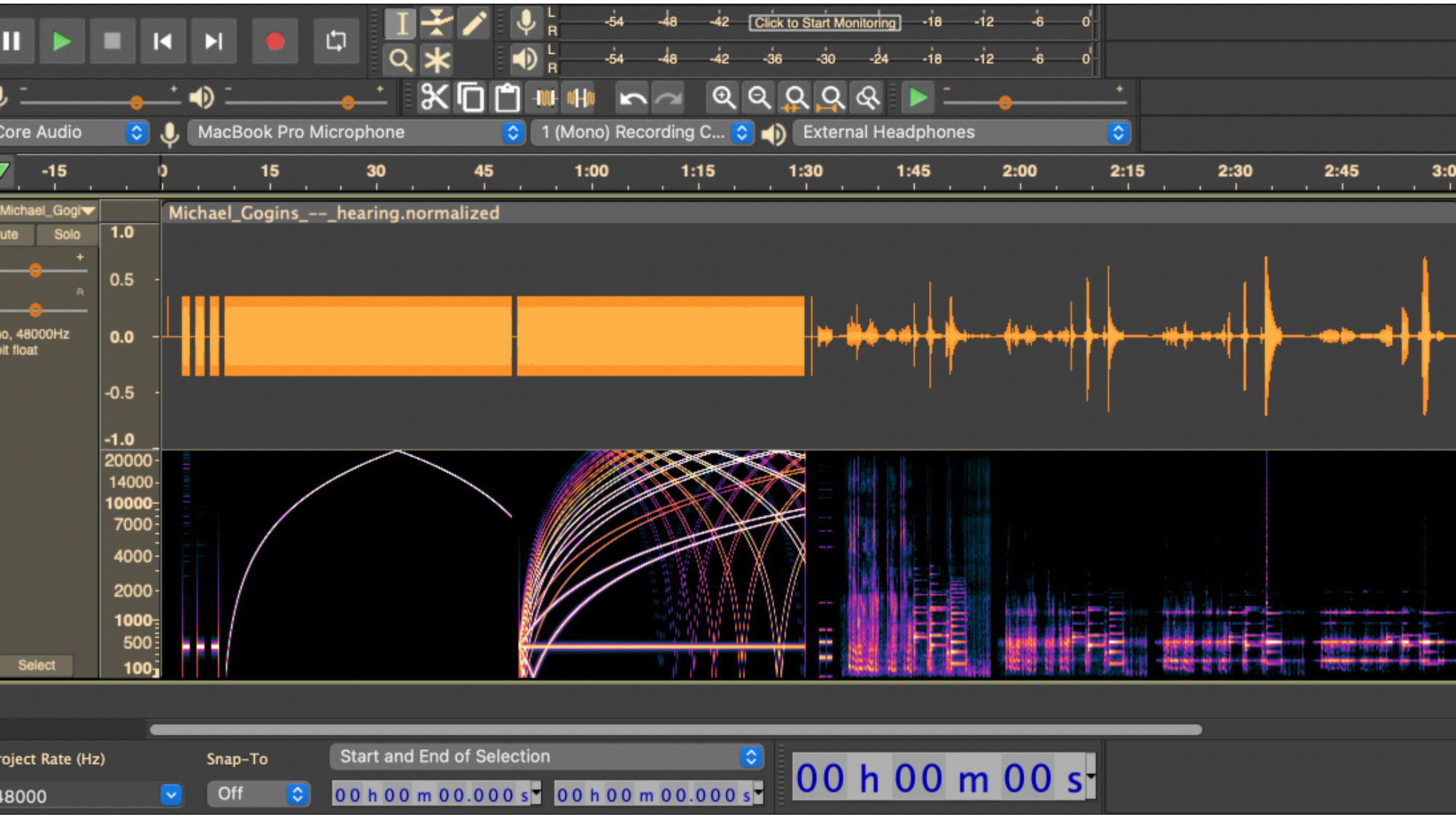
Hearing electroacoustically

Example

Ideally, you should get a hearing test from an audiologist.

- Change to the `exercises/hearing` directory.
- Execute `csound hearing.csd` to render a soundfile.
- Open `hearing.wav` in Audacity
- Configure Audacity's preferences for Tracks to auto-fit track height, default view mode multi-view.
- Configure Audacity's preferences for Spectrograms to use the frequencies algorithm, window size 4096, min frequency 0, max frequency 40,000 (to view aliasing), Mel scale, gain 20 dB, range 80 dB.

Hearing electroacoustically



Hearing electroacoustically

- The initial spike shows that a 1 sample click contains all frequencies up to the Nyquist frequency.
- The next three columns show the effects of increasing wave table size on noise.
- The first arc shows a frequency sweep from 0 up to 40 KHz, though the sampling rate is 48 KHz.
- What can you really hear? Where does your ability to hear high frequencies end? Use good headphones.
- As for me, I can't hear anything past about 12 KHz.

Hearing electroacoustically

- The second set of arcs shows aliasing of an FM tone as the frequency of the modulating signal increases. Note the *negative* aliasing, which reflects off frequency 0.
- After the arcs, there is a single cosine grain .01 seconds long.
- Then there is a note with a non-releasing envelope, and the same note with a releasing envelope.
- Finally, there are bands where a source sound is convolved with an increasingly longer impulse sound.

Hearing electroacoustically

- In Audacity, zoom in on the single short grain.
- Change the spectrogram window to the smallest size.
- Time is well-resolved, but *all* frequencies are shown in the spectrogram, i.e. there is *no* frequency resolution.
- Change the spectrogram window to the largest size.
- Frequency is now well-resolved, but the energy is smeared out *well* before and after the grain.
- This shows the time/frequency uncertainty that exists in all convolution and spectral processing.
- The minimal area of resolved energy is called the Gabor logon. It can get taller or wider, but its area is constant.

Hearing electroacoustically

- In Audacity, zoom in on the two notes after the grain.
- The first note uses a long exponential decay with no release, so it just cuts off at the end of the note.
- This causes a discontinuity in the waveform, which is the same as noise in the frequency domain. The spectrogram shows this as energy spread out in the vertical dimension.
- The second note is exactly the same, but using the releasing form of the envelope.
- Clicks are a bedeviling artifact of digital audio, because avoiding discontinuities in digital signals is hard.
- Natural sounds don't tend to have such discontinuities, as physical energy takes time to build up or to release.
- Always use a releasing envelope or a de-clicking envelope.

Hearing electroacoustically

- Finally, there are four bands at the end of the soundfile.
  1. An original recording of junk noises from my apartment.
  2. Junk convolved with a .01 second bell sound. It's subtle, a bit of ringing on the final note.
  3. Junk convolved with a .1 second bell sound. The convolution can be heard on all sounds. There is a kind of ringing from the impulse.
  4. Junk convolved with a 1 second bell sound. The original sounds are obscured by the ringing of the long impulse.
- The ringing is an artifact *in some contexts*, I call it "convolution smear."
- What is true of convolution is true also of filters and time/frequency operations. A positive frequency in the impulse produces ringing, a negative frequency causes notching. Maybe you *want* this, maybe not, but *notice* it.

Coding for high-resolution audio

- Many of Csound's opcodes, man pages, and examples are old, and prioritize memory and run time over audio quality.
- *Today,* ***prioritize audio quality in all cases***.
- Use a sample rate of 48,000 or even 96,000 frames per second with `ksmps = 100` or even `ksmps = 1`.
- Use the double-precision version of Csound (now default).
- Render to floating-point soundfiles for high dynamic range.
- Use table sizes of at least 65,537 for lower noise.
- Use *audio-rate* envelopes to prevent zipper noise.
- Use releasing or de-clicking envelopes on all final outputs.
- Use the *latest versions* of opcodes.
- Used like this, Csound has unsurpassed audio quality.

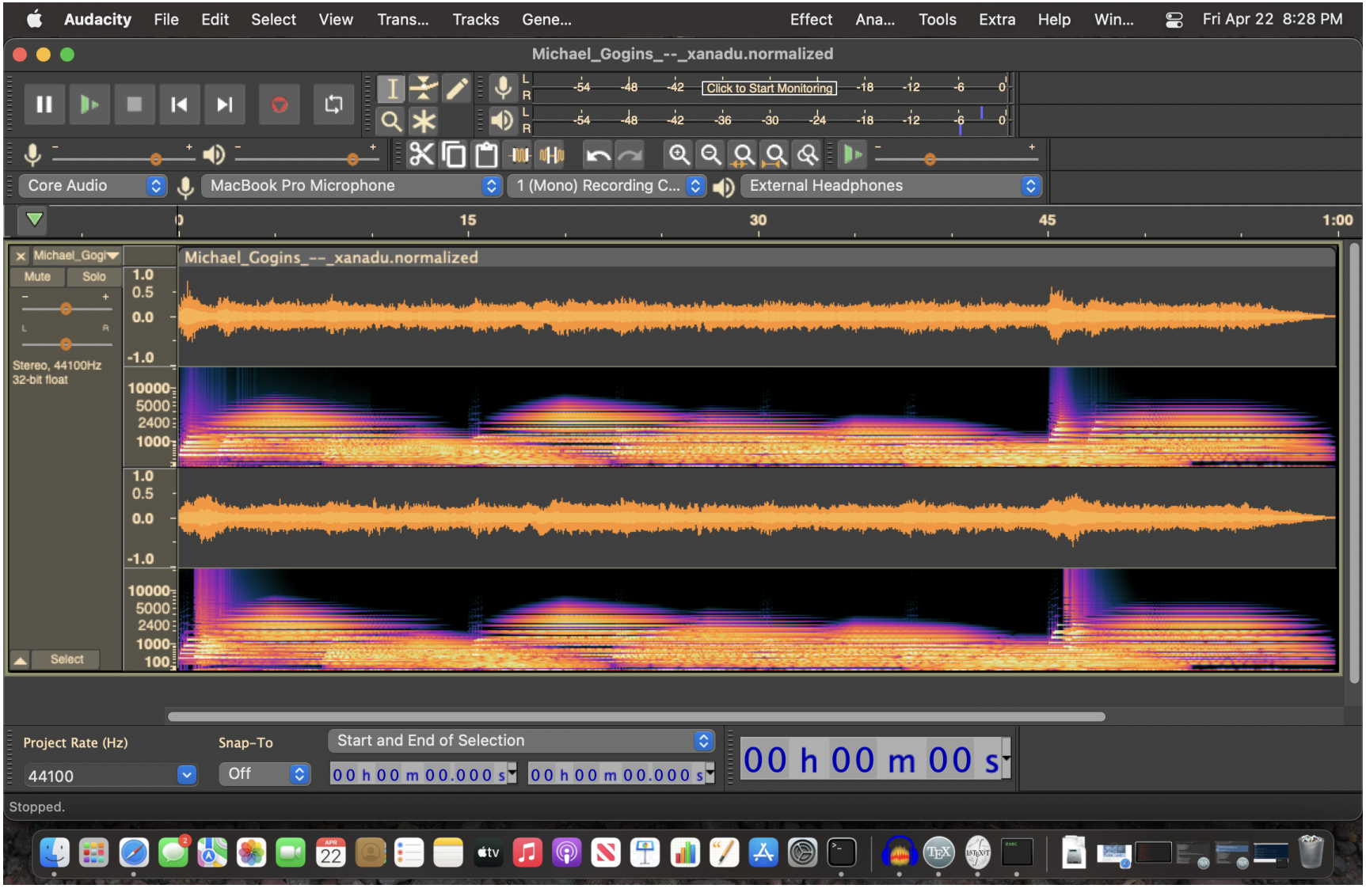
Exercise: Coding for high-resolution audio

Example

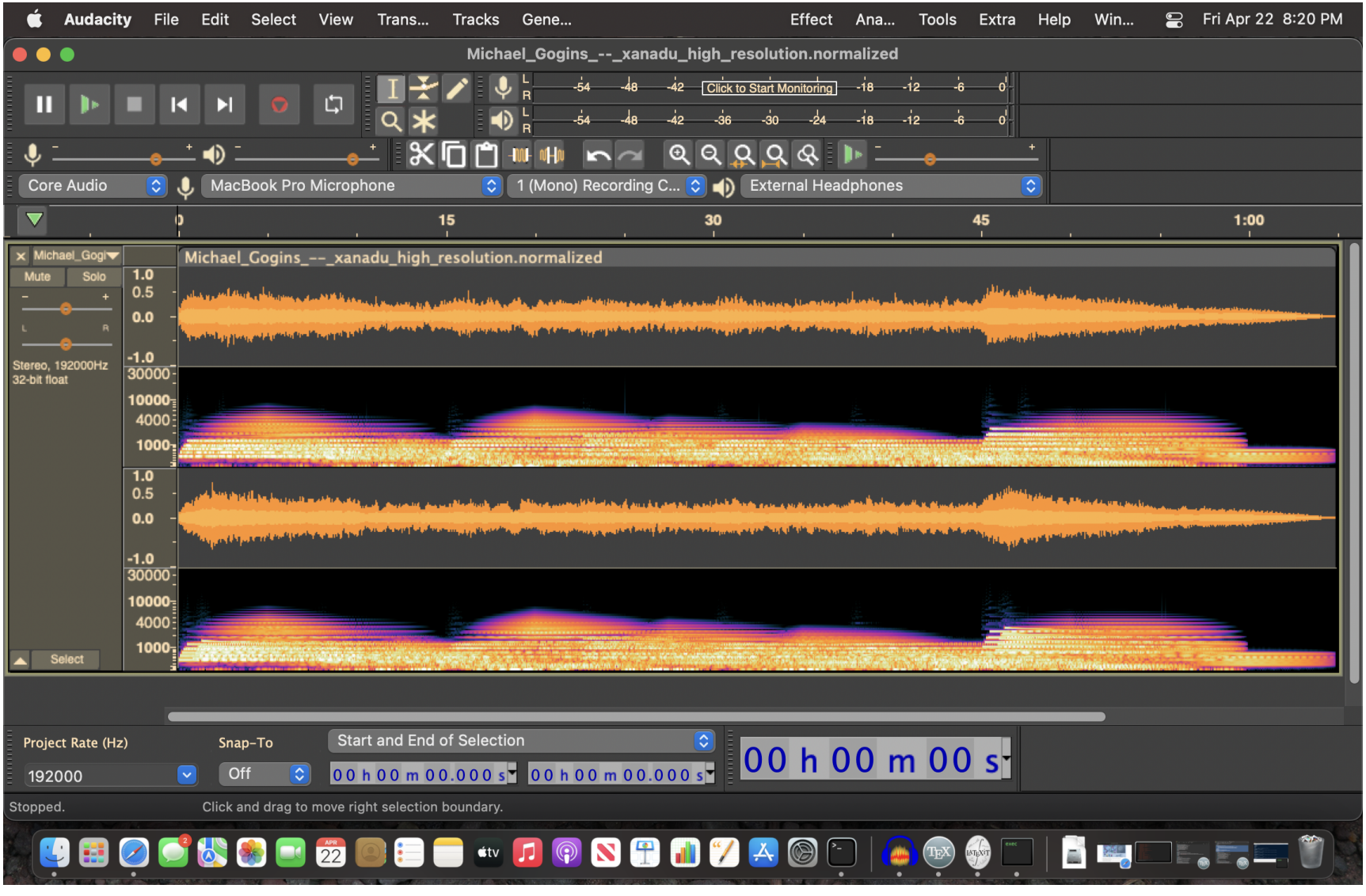
Change to the `exercises/resolution` directory in your workshop directory.

- Execute `./playpen.py csd-play xanadu.csd`, listen on headphones, and examine the spectrogram in Audacity.
- Execute `./playpen.py csd-play xanadu-high-resolution.csd` the high-resolution version, listen on headphones, and examine the spectrogram in Audacity.

Original Xanadu



High-Resolution Xanadu



Useful opcodes

Exercises for some of these follow.

- `transegr`: most flexible envelope generator.
- `ftgen`: move function tables from score to orchestra.
- `poscil`, `poscil3`: most precise oscillators.
- `crosspm`, `crosspmi`: phase modulation/frequency modulation.
- Fluidsynth opcodes for SoundFonts (plugin).
- Opcodes for Faust DSP language (plugin).
- `diskin2`: flexible soundfile input.
- `deltapi`: building block for physical models.
- `PVS` opcodes: toolkit for time/frequency operations.

transegr

Example

Change to the `exercises/opcodes` directory of your workshop directory.

- Execute `~/playpen.py csd-play transegr.csd` and view the soundfile in Audacity.

- Observe that an exponent of 0 creates linear segments, a negative exponent creates concave segments.
- See how the `r` (release) suffix handles termination of notes without clicking.
- There is seldom a need for any other envelope opcode.

deltapi

Example

Change to the `exercises/opcodes` directory of your workshop directory.

- Execute `~/playpen.py csd-play livingston-guitar.csd` and view the soundfile in Audacity.
- Read the code to see how double delay lines with filters model reflecting waves in strings.

FluidSynth opcodes I

Ways to use sampled sounds in Csound:

- Load samples into function tables using GEN01 and play them with oscillators.
- Low-level `sf` opcodes to play SoundFonts, built in to Csound.
- High-level `fluid` opcodes to play SoundFonts, available from `https://github.com/csound/plugins/tree/develop/fluidOpcodes`.
- Various open source and commercial plugins, some *far* better than the above.

Here we discuss the FluidSynth opcodes as they are free, simple to use, and work pretty well.

FluidSynth opcodes II

Example

- Execute `~/playpen.py csd-play fluidsynth.csd` and view the soundfile in Audacity.
- You can load another SoundFont (just change the filename) or select a different preset (just change the preset parameter for the `fluidLoad` opcode) and see what happens.

Faust opcodes

The Faust opcodes are available from https://github.com/csound/plugins/tree/develop/faustcsound.

- Oops! I didn't have time to get the Faust opcodes working on my Mac!
- But, check out the online Faust IDE at https://faustide.grame.fr/.
- You can actually write Faust code online, compile it online, and download the result as a plugin Csound opcode.
- The Faust opcodes are documented in the standard Csound Reference Manual.

PVS opcodes

The Phase Vocoder Streaming opcodes are built in to standard Csound. Kudos to Victor Lazzarini for developing this clean and flexible toolkit.

Example

Change to the `exercises/opcodes` directory of your workshop directory.

- There are a number of `pvs*.csd` examples in this directory, as well as required input soundfiles.
- Just run each example and listen.
- *Just keep in mind – everything you hear here, can also easily be done with real-time audio input.*

Software design

- **Encapsulation** Hide implementations in *modules* that expose only inputs and outputs. Define modules with `instr` (classes) and `opcode` (functions).
- **Abstraction** Define abstract interfaces for instruments or UDOs that perform similar functions; e.g. different synths use the same pfields.
- **Normalization** Use *musical* units: MIDI key not Hz, MIDI velocity not amplitude.
- **Signal Flow** Signals flow from input devices through opcodes to instruments; from opcode to opcode through variables; from instruments through opcodes to output devices. Processing is first by instrument number, then by order of definition. Signals also flow through control channels and global variables.

Modular design

- Using naming conventions to simulate namespaces.
- Normalize pfields and variables.
- Associate global function tables with modules.
- Associate global control variables with modules, and use `chnexport` to export these variables as control channels.
- Define signal flow in the orchestra header, *not* inside modules.
- Use the MIDI interop command line options so the same instruments work for both MIDI and scores.

Modules in Csound

Example

Change to the `exercises/coding` directory of your workshop directory.

- Execute `~/playpen.py csd-play modules.csd` and view the soundfile in Audacity.

Read the code...

- Function tables are created as global variables associated with the instrument definition.
- To that end, namespaces are implemented using prefixes to names.
- Note "natural" units (MIDI key, MIDI velocity) in the score.
- Note instruments can be re-ordered, added, removed; all connections are made in the orc header.

Plugins

- Csound has its own plugin format, not only for plugin opcodes, but also for plugin function table generators.
- Csound also has opcodes that can load non-Csound plugins:
  - VST2 plugins, now deprecated by Steinberg (freeware from me).
  - VST3 plugins, now free software (on GitHub from me).
  - The Jack Audio Connection Kit (JACK) can be considered a kind of plugin protocol. Csound supports JACK both as an input/output driver, and with the Jacko plugins from me.
- Plugins are available from *many* sources. The largest number are VST plugins.

Sources of Csound plugins

- Csound external plugins repository.
- The Risset package manager for Csound plugin opcodes.
- `vst4cs` opcodes. Not discussed here as VST2 is now deprecated by Steinberg.
- `csound-vst3-opcodes`.

External Csound plugins repository

- This contains opcodes former in Csound's GitHub repository that have external dependencies, and so were moved to their own repository.
- These currently need to be built from source code, which is not difficult if you can program.
- Go to `https://github.com/csound/plugins` to see what's currently available.

csound-vst3-opcodes

This assumes you have installed `csound-vst3-opcodes`, or built them from sources.

Example

Change to the `exercises/plugins` directory of your workshop directory.

- Execute `~/playpen.py csd-play vst3-opcodes-minimal.csd` and view the soundfile in Audacity.

Opcodes in Risset

Example

Change to the `exercises/plugins` directory of your workshop directory.

- Execute `python3 -m risset list` to view all plugins in the repository.
- Execute `python3 -m risset show beosc` to view details on the band-limited+noise oscillator opcodes.

Csound and other languages

- The following exercises all implement the same Csound piece, for purposes of comparison.
- Embed Csound in Python with `ctcsound`.
- Embed C++ in Csound with `csound-cxx-opcodes`.
- Embed Csound in C++ with `csound_threaded.hpp`.
- There are many other language bindings for Csound....

Csound in Python

Example

Change to the `exercises/languages` directory of your workshop directory.

- Execute `python3 csound-in-python.py`
- Open `csound-in-python.py` in Audacity to see and hear the soundfile.

C++ in Csound

Example

Change to the `exercises/languages` directory of your workshop directory.

- Execute `csound cplusplus-in-csound.csd` to render the piece.
- Open `cplusplus-in-csound.wav` in Audacity to see and hear the soundfile.

Csound in C++

Example

Change to the `exercises/languages` directory of your workshop directory.

- Execute `~/playpen.py cpp-app csound-in-cplusplus.cpp` to compile the piece.
- Execute `./csound-in-cplusplus` to run the piece.
- Execute `open csound-in-cplusplus.wav` to view and hear the soundfile.

A Moderately Sophisticated Piece (design)

- The piece uses C++ for score generation, HTML for the user interface, and Csound for synthesis.
- All C++ and HTML5 code is embedded in the Csound code.
- The orchestra uses modular instruments with signal flow graph connections between instruments and effects.
- With the HTML user interface I intensively fine-tuned levels and other parameters during trial renderings.
- Instruments include Pianoteq's VST3 modeled piano, and Csound instruments by other musicians I have adapted.
- The piece uses the `cxx_os` opcode to load resources from specific locations depending on operating system (macOS/Linux). Thus I can run my piece on both macOS and Linux without editing it.

A Moderately Sophisticated Piece (composition)

- The score is generated in C++ using CsoundAC's multiple copy reducing machine algorithm, which makes many fractals.
- This MCRM shrinks the score down onto each of its four quarters, repeating this 7 times. Each "lens" is a bit different to create melodies, canons, and variations.
- The "lenses" in the MCRM algorithm are C++ lambdas, so I can do more than just apply transformation matrices.
- E.g. at specific levels of iteration, i.e. specific time scales, I transform the major/minor 9th chord that controls the harmony. This enables the generation of root progressions, mutations, and quasi-tonal modulations.
- The geometric transformations in the "lenses" make it easy to change the structure of the entire piece by editing a few numbers.

Questions?

Please ask some questions! They can be about Csound, about programming, or about computer music in general.