Music 320C: Audio Plugin Development in FAUST and C++

Center for Computer Research in Music and Acoustics (CCRMA)
Department of Music, Stanford University
Stanford, California 94305

Spring Quarter, 2020-2021

Contents

Music 320C: Audio Plugin Development in Faust and C++

1 Course Overview

Music 320C focuses on audio plugin development in the FAUST and C++ languages, organized and supported by the JUCE multiplatform development framework, or used within interactive programming environments such as ChucK, Pure Data, or SuperCollider. The principal activity is a software project due at the end of the quarter, with a progress-report demo mid-quarter. Lectures introduce the FAUST language for audio signal processing and the JUCE framework for audio plugin development. A progressive series of example plugins is covered in class. Familiarity with C++ and elementary UNIX-style software development is required. Signal processing theory on the level of Music 320A and/or 320B is assumed.

Goals of the Lectures

- Acquire facility in the FAUST language for audio signal processing
- Learn which interactive programming environments are most easily used with FAUST
- Become aware of some of the signal-processing tools in the FAUST Libraries[1]
- Be able to use FAUST-generated C++ modules in a professional quality audio plugin
- Be apprised of various options for creating an audio plugin GUI

Goals of the Project

- Hands-on experience making your own audio plugin using FAUST, C++, and signal processing tools

2 When, Where, Who

Term:	Spring Quarter
Location:	CCRMA Class Room (Knoll 217)
Time:	Mondays and Wednesdays, 4 PM
Instructor:	Julius Smith (jos@ccrma.stanford.edu)
TA:	Vidya Rangasayee (vidya@ccrma.stanford.edu)
Website:	`https://ccrma.stanford.edu/courses/320c/`

Prerequisite Software

While C++ development experience is assumed, no prior experience with the FAUST language or JUCE development framework is required. It is helpful but not required to have experience with an interactive programming environment supporting FAUST modules, such as ChucK, Pure Data, Max/MSP, or SuperCollider. Introductory materials appear below in the class schedule.

[1] `https://faustlibraries.grame.fr/`

3 Administrative Information

Announcements

Class announcements are often made via *email*. For this we are presently using Piazza:

`https://piazza.com/stanford/spring2021/mus320c`

If you have signed up for the class in axess, by the first day of the quarter, you should receive an invitation from Piazza to join the class (using the email address known to axess). Otherwise, please join by visiting the above URL and entering your preferred email address.

Units

You may sign up for 1 to 10 units. The first unit corresponds to weekly class attendance (about three hours per week). Each additional unit should represent approximately 3 hours of focused activity per week on your project work.

Prerequisite Knowledge

You are expected to know

- C++ (able to read and understand the basics, and able to look up anything new to you)
- Elementary signal processing on the level of Music 320A and/or 320B

Skills to Develop

This course should move you forward along the following dimensions:

- UNIX proficiency (working at the shell command line)
- git (and git submodules)
- C++ (at the level of JUCE, which is pretty modern)
- Faust (which compiles to C++ by default)
- JUCE (a large set of very useful C++ classes for audio plugins)
- Application and plugin development on the platform(s) of your choice
- 320-level audio signal processing

Your progress will naturally depend on how experienced you were coming in.
Just make a quarter's worth of progress!

When you are up to speed on these skills, extend your knowledge and expand your project!

Project Requirements

Project requirements apply to those enrolled for more than one unit. Your project must include

1. a JUCE audio plugin (which can be a standalone app as well),
2. at least one FAUST module (either by you or from the FAUST Libraries et al.), and
3. something you can show to the class.

Project Schedule

1. Project plan due by the end of 2nd week of classes
2. Progress demo due by the end of the fifth week of classes (about half way through the quarter)
3. Project presentation during the last week of classes
4. Final project turned in by email by the end of the quarter

Grading

Grades are based on class engagement, project updates/demos, and final project work. Those enrolled for one unit (class attendance only) should sign up for CR/NC.

4 Schedule

Below is the schedule of weekly in-class presentations, with pointers to all associated reading, lecture overheads, and so on. The default weekly topic is learning the FAUST language for signal processing and the JUCE development platform for C++ audio plugins. Additional discussion will be driven by the interests, projects, and any research activities of the participants. Results and associated pointers will be logged here.

- Week 1: Course Overview, Project Planning and Introduction to the Faust Language for Audio Signal Processing
  - Course Overview Videos:
    - FAUST Standalone App: Graphic Equalizer Example[2] [15:33]
    - Plugin Gui Magic Equalizer Example (JUCE Audio Plugin)[3] [10:23]
  - Read MUS320C main handout[4] (this document)
  - Plan your Audio Plugin Project
    - What novel signal-processing plugin would you like to have?

[2] https://www.youtube.com/watch?v=o0WQQ2ABEHc
[3] https://www.youtube.com/watch?v=8bZVHQk4L7k
[4] https://ccrma.stanford.edu/~jos/intro320c/

    * Default Project: Modify a JUCE plugin example (such as one covered in class) to use a new Faust module in its signal-processing function(s)
  - Try out the FAUST Web IDE[5]
    (Firefox browser most recommended right now)
    Video: FAUST IDE Intro[6] [23:20]
    This appears to be the FAUST IDE of choice for live-coding presentations.
  - Optional (we won't be using these in this class):
    * FAUST Editor[7]
    * FaustLive Tutorial[8]
    * FAUST Playground[9]
  - Read "Audio Signal Processing in Faust" (ASPF)[10] up to "Pattern Matching in FAUST"
    * Following ASPF, clone (*i.e.*, download using `git`), `make` (compile and link), and `make install` FAUST on your personal computer:
      Video: Installing the Latest FAUST on a Mac[11] [32:54]
    * FAUST Intro Videos:
      - Bottom-Up Intro to the FAUST Language[12] [1:12:18]
      - FAUST Syntax with Examples[13] [1:05:45]
  - Briefly look over "Faust Quick Reference"[14]
    This has long been the *most comprehensive and up to date reference manual* for FAUST.
  - Briefly read about FAUST "Architecture Files"[15]
  - Briefly see what other sections exist under "Faust Manual" at `https://faustdoc.grame.fr`
    For example, you'll want to come back to this to learn about support for *MIDI, OSC, sound files*, and the various `faust2*` tools.
  - Try making some standalone-apps from the FAUST examples using `faust2caqt`, `faust2jaqt`, `faust2jack`, or the like.
    This is a good way to prototype FAUST plugin modules.

- Week 2: Introduction to the FAUST Libraries
  - Continue project discussions in class and beyond
  - Continue reading "Audio Signal Processing in Faust"[16]
  - Continue with videos from last week
  - Continue with reading/perusings from last week

[5] `https://faustide.grame.fr`
[6] `https://www.youtube.com/watch?v=n6F5AT_huPA`
[7] `https://fausteditor.grame.fr`
[8] `https://www.youtube.com/watch?v=8ZUD2c5D-PU`
[9] `https://github.com/grame-cncm/faustplayground#readme`
[10] `http://ccrma.stanford.edu/~jos/aspf/`
[11] `https://www.youtube.com/watch?v=x1z50q6U1GQ`
[12] `https://www.youtube.com/watch?v=b8dn2R3hAe0`
[13] `https://www.youtube.com/watch?v=1f7m3W5fofI`
[14] `https://github.com/grame-cncm/faust/blob/master-dev/documentation/faust-quick-reference.pdf`
[15] `https://faustdoc.grame.fr/manual/architectures/`
[16] `http://ccrma.stanford.edu/~jos/aspf/`

- Week 3: Integrating FAUST modules within interactive programming environments
  - Continue interactive video playback with discussion
  - FaucK[17] (FAUST modules in ChucK[18]) by guest lecturers Ge Wang and Romain Michon
  - Optional (if for you): Try out `faust2puredata` and/or `faust2max6` and/or `faust2supercollider`
- Week 4: More FAUST Examples and Introduction to the JUCE Development Framework
  - Using `faust2octave` to test signal-processing modules[19] [9:12]
  - FAUST Examples and Demos (all from `faust/examples/` or `faustlibraries/demos.lib`):
    * The FAUST IDE[20] is a good way to peruse them all (find the Examples pull-down menu near the top)
    * `examples/filtering/parametricEqLab.dsp`[21] [10:54]
      Assumes you have watched our first "FAUST Standalone App" video covering `examples/filtering/graphicEqLab.dsp`[22] [15:33]
    * `examples/filtering/vcfWahLab.dsp`[23] [13:56]
  - Web-search for *JUCE Tutorial* to find `https://juce.com/learn/tutorials`
    (first search-result as of this writing)
    If new to you:
    * Read "Tutorial: Projucer Part 1: Getting started with the Projucer"[24]
    * Read "Tutorial: Projucer Part 2: Manage your Projucer projects"[25]
    * Optional: Learn how to replace Projucer with CMake:
      · JUCE CMake Repo Prototype[26]
      · CMake Tutorial[27] (look it over and read the first few examples to get a feel)
      · Announcement of CMake support in JUCE[28]
      · JUCE Forum discussion on CMake support in JUCE[29]
      · The JUCE CMake API[30]
      · CMake Examples in JUCE[31]
    * Optional: Explore making JUCE plugins for **Wwise** middleware (videogames): ADC-18 Talk Video[32], Example on GitHub[33]

[17] `https://ccrma.stanford.edu/~rmichon/fauck/`
[18] `http://chuck.stanford.edu`
[19] `https://www.youtube.com/watch?v=1f7m3W5fofI&t=42m`
[20] `https://faustide.grame.fr`
[21] `https://www.youtube.com/watch?v=HCQXlOb77Uk`
[22] `https://www.youtube.com/watch?v=o0WQQ2ABEHc`
[23] `https://www.youtube.com/watch?v=ZYkJb07l52Q`
[24] `https://docs.juce.com/master/tutorial_new_projucer_project.html`
[25] `https://docs.juce.com/master/tutorial_manage_projucer_project.html`
[26] `https://github.com/eyalamirmusic/JUCECmakeRepoPrototype.git`
[27] `https://cmake.org/cmake/help/latest/guide/tutorial/index.html`
[28] `https://juce.com/discover/stories/announcing-juce-6#new-features-in-juce-6`
[29] `https://forum.juce.com/t/native-built-in-cmake-support-in-juce/38700`
[30] `https://github.com/juce-framework/JUCE/blob/master/docs/CMake%20API.md`
[31] `https://github.com/juce-framework/JUCE/tree/master/examples/CMake`
[32] `https://youtu.be/uBWt6_0f7uw?t=12910`
[33] `https://github.com/joelrobichaud/Voluminous`

    - Read "Tutorial: Create a basic Audio/MIDI plugin, Part 1: Setting up"[34]
    - Read "Tutorial: Create a basic Audio/MIDI plugin, Part 2: Coding your plug-in"[35]
    - Read "Tutorial: Configuring the right bus layouts for your plugins"[36]
    - Read "Tutorial: Plugin examples"[37]
    - If you need 3D graphics or GPU-based animations, read "Tutorial: Build an OpenGL application"[38]
  - Build and run the JUCE `DemoRunner` to see examples of nice available starter code
    Video: Building and running the JUCE `DemoRunner` on a Mac[39] [2:35]
  - Familiarize with the `juce_dsp` classes[40]
  - Carefully read Adding Faust DSP Support to Your JUCE Plug-ins[41]
    (our method will be essentially the same)

- Week 5: Project progress reports, Integrating FAUST in JUCE

- Week 6: JUCE Audio Plugin GUI Options
  - Host AutoGUI
  - Using `faust -xml some.dsp` to create XML readable into a `juce::ValueTree`
    (*I.e.*, start with a FAUST standalone app and port its GUI to JUCE via XML. Recall our first in-class demo: FAUST Standalone App: Graphic Equalizer Example[42] [15:33])
  - FAUST-specified GUI via `faust2juce`
    (Start with a FAUST standalone app and convert it to a JUCE project)
  - Run-Time Editable AutoGUI using "Plugin Gui Magic" (PGM)[43]
    - PGM EqualizerExample demonstrated in Week 1
    - GUI Rapid Prototyping in PGM (before any parameters exist)[44]
  - Custom JUCE GUI Hand-Coded in C++
  - JUCE Tutorial: "Advanced GUI Layout Techniques"[45]
  - External controllers (MIDI, OSC) driving the GUI however it's done.
    JUCE supports MIDI timestamps down to a resolution of `minimumSubBlockSize`, which is 32 samples by default, which is under 1ms for normal audio sample rates, and matches standard MIDI pretty well. This is independent of your audio block size.

- Week 7: Example JUCE plugins made using FAUST

[34] `https://docs.juce.com/master/tutorial_create_projucer_basic_plugin.html`
[35] `https://docs.juce.com/master/tutorial_code_basic_plugin.html`
[36] `https://docs.juce.com/master/tutorial_audio_bus_layouts.html`
[37] `https://docs.juce.com/master/tutorial_plugin_examples.html`
[38] `https://docs.juce.com/master/tutorial_open_gl_application.html`
[39] `https://www.youtube.com/watch?v=Pxk5F9RRuPM`
[40] `https://docs.juce.com/master/group__juce__dsp.html`
[41] `https://faustdoc.grame.fr/workshops/2020-04-10-faust-juce/`
[42] `https://www.youtube.com/watch?v=o0WQQ2ABEHc`
[43] `https://foleysfinest.com/developer/pluginguimagic/`
[44] `https://youtu.be/tsm1hp-X01w`
[45] `https://docs.juce.com/master/tutorial_rectangle_advanced.html`

- Week 8:
  - JUCE + PGM GUI wrap-up
    Nick Porcaro guest lecture
  - Measuring Acoustic Bodies to use as Impulse Responses (IRs)
    Mark Rau guest lecture
- Week 9: Example JUCE+PGM Plugins, GitHub and CMake Workflows
  Jatin Chowdhury guest lecture
- Week 10 - Project Presentations (Five 10-Minute Presentations in 15-Minute Slots)