Crossroads Audio Plug-In

Member #: 92711
Category: Undergraduate
I. INTRODUCTION

Crossroads is an audio software plug-in (seen above as an AU in Logic Pro 9) designed to provide a large and expandable array of effects that dynamically respond to the input signal.

Crossroads offers a diverse palette of sound design possibilities to anybody using AU, VST, RTAS, & AAX plug-ins on either a Mac or Windows based DAW. The dynamic responsive nature of the plug-in extends this palette: A certain preset may very well respond differently to a drum track than a mandolin!

Crossroads was built using C++ in Xcode 4.

1In the source code, “FX Modules” are called “FX Chains” or “FXChains”

The JUCE (juce.com) class libraries & IDE were used to create the “skeleton” of the plug-in & generate graphical components that could then be manipulated in Xcode.

The JUCE wrappers supports the four previously mentioned plug-in formats. At the time of writing this paper Crossroad has only been built & tested as an AU. Building & testing of the three other formats is expected after the AU beta testing.

II. SIGNAL FLOW

There are three main component sections that make up Crossroads: The Detector, Audio Router, & FX Modules¹ (Fig. 1).

A. Detector

The Detector consists of an envelope follower. The uni-polar output (range of 0-1) of the Detector

¹In the source code, “FX Modules” are called “FX Chains” or “FXChains”
section feeds into the control input of the Audio Router.

B. Audio Router

Being the “command bridge” of the plugin, the Audio Router was probably the trickiest of modules to implement. I spent an entire day with my notebook, jotting down algebraic ideas & flow diagrams.

There are two main functions in the AudioRouter class:

• calcCtlValue (Ctl = “Control”)
• processInput

calcCtlValue() takes the output of the envelope follower, the Center parameter value, the Width parameter value, & calculates a number between 0 & n-1, where n is the number of current FX Modules.

processInput() takes an audio sample as an argument and applies gain to its internal array of sample values according to the previously set control value. Then, in the main processBuffer function, these values are accessed and passed on to the FX Modules.

Using the Audio Router, the user can choose to either
1) Control which FX Modules the audio is routed into or (Width = 0)
2) Control the range to which the envelope follower controls the routing (Width > 0).

The entire signal flow (for a single channel) can be seen in Fig 2. The routing mechanism may be hard to visual because of the lack of level meters (a high priority on my updates list), but one can imagine a horizontal meter pointing to the right that displays the output level of the envelope follower. The meter would always be as big as the purple Width indicator bar & would follow it around as the Center was changed. Vertical meters at the top of each FX Module would point down. When the top

Fig. 1 Crossroad’s Sections
of the horizontal meter passes by an **FX Module**, audio would be routed into it and its meter would indicate so.

The **Center** and/or **Width** can be modulated with LFOs by pressing the **Center Mod & Width Mod** buttons respectively. As a future update, rhythmically syncing these LFOs to the host sequencer would offer a much more musical option. As of now, they add a nice cyclical pulse but are difficult to sync to music.

**C. FX Modules**

The number of **FX Modules** is dynamic. The current build is limited to 7 maximum & 2 minimum modules. The user can add & remove modules by pressing the +/- buttons in the top right corner of the plugin.

Using class inheritance, I made a pure virtual “FX Module” base class with all the crucial functions (prepareForPlay(), processAudio(), etc.). Then in each different type of FX Module, these virtual functions were overridden (for example, in the LPF’s prepareForPlay(), the bi-quad’s delay registers were reset). This allowed me to hold an array of the current FX Modules (each being whichever specified by the user via the drop down menu), iterate through them & call these functions generically.

Currently there are 5 different FX Modules the user can choose from (I have a massive list of updates to expand this list)

- 2nd Order LPF
- 2nd Order HPF
- “Octaviar” Distortion
- Ring Modulator
- Diffuser

**III. GRAPHICAL INTERFACE**

The various graphical components were built in the JUCE’s IDE “Introjucer” (Fig. 3) separately & then exported to the Xcode project.

Syncing the dynamic amount of FX Module parameter sliders to the FX Module processors themselves was difficult. In various online forums I was advised to not place any GUI updating functions directly in the audio thread (therefore the function processBlock()) as it is important to not impede on input/output buffering data flow. Therefore, the GUI sets and checks only the parameter objects within the processor. In the current build, the sliders & modulation indicators are updated every 30 mSec. Among other features this allows the user to see automation being read.

**IV. SMOOTHED PARAMETERS**

Early on in the development process I decided to create a class that would handle the smoothing of parameter changes. This helps to avoid pops & clicks as sliders update values too quickly (very
useful for automation). The smoothing equation is:

\[ A = A + (\tau \cdot (B - A)) \]

Where \( A \) is the current value, \( B \) is the target value, and \( \tau \) is the smoothing time constant (calculated using the sample rate).

An issue I recently ran into was that the function that checks to see if the smoothing is done (“current value” = “target value”) does not work with the current smoothing equation (comparing the equality of floats in general is a risky business).

I began implementing a “rate of change clamping” function that I created last year to solve this issue. Instead of using a timing constant & scaling the amount that is added on to the current value, the amount that the value can change every sample period is simply limited by a static value (calculated using the sample rate). This allows the comparator (“current value” >= “target value”) to be used. If the above condition returns true, a boolean flag is flipped and the smoothing turns off.

VI. OVERHEAD ISSUES

The current build the plug-in uses quite a lot of CPU. I have a large list of ways to reduce the overhead, the most two significant being:

• Implement the new smoothing algorithm
• Implement a non-channel specific processAudio() function in the FX Module hierarchy

Since the current smoothing algorithm cannot check to see if smoothing is finished, each FX Module has to recalculate (“cook”) its internal coefficients & values every sample period (as though the user was constantly moving every slider at once). I’m estimating that fixing this will reduce the CPU by more than 50%.

Some FX Modules can be made more efficient by moving some of the processing from a channel-specific processAudio() function to a non-channel specific function. The Ring Modulator, for example only needs one Oscillator (at the moment. A stereo ring modulator may eventually be implemented). Instead of running two oscillators & calculating their values during each sample period, only one can be used, which calculates the oscillation value before an channel specific occurs. Man of the “cooking” functions can be moved to non-channel specific processing as well. This will greatly reduce the CPU usage on stereo instances of the plug-in.

VI. FUTURE IMPLEMENTATIONS

This list is constantly expanding/changing!

• Meter Components
• More FX Modules
  • BPF
  • Moog Ladder LPF
  • Korg35 LPF/HPF
  • Diode Ladder Filter
  • Wave Shaper
  • Convolver (user loads custom impulse)
  • Reverb
  • Phaser
  • Tape Delay
  • Randomize Button (and randomizer sub-buttons for each FX Module)
  • Transient Designer
  • Glitch/Stutter Slowdown (envelope triggered)

(V I have coded many of these modules already but have not implemented them in Crossroads yet)

• Routing Modes (“Interpolate, Layer, Latch”. Current build only supports “Interpolate”).
• Detection Modes (“Frequency, Transient, etc…)
• Drag and drop FX Modules into place
• Delete select FX Modules.

VII. REFERENCES