COSC 4P98 Lecture notes: **VST programming**
March 9, 2012
B. Ross

- **VST: Virtual Studio Technology**
  - Created by Steinberg, makers of Cubase.
  - A standard that permits plug-in modules to be implemented for host applications.
  - Has become the most accepted standard, although others exist (RTAS for Powertools, Audio Units in Mac OS X,...

- Three types of modules:
  1. VST instrument: sound generator
     - Normally a sampler or synthesizer.
     - Many emulations of famous hardware synths exist.
     - example: Arturia Moog Modular
  2. VST effect: an audio effect processor
     - takes sample data from host, alters it, and gives it back.
     - example: echo
  3. VST midi effect: processes midi data
     - eg. arpeggiator: creates sequences of midi notes, perhaps from chord input
- Versions of VST
  - VST 2 (eg. 2.4)
    - older, stable version
    - recommended to use this for 4P98
  - VST 3 (eg. 3.51)
    - newer
    - many optimizations and enhancements
- SDK:
  - software developer's kit
  - class definitions for accessing data from host application
  - documentation as well
  - Many commercial applications publish SDK's for their software
    - Adobe, SoftImage, ...

- We will concentrate on audio effects here. Please look at the documentation for information about VST instruments, and VST midi effects.
  - Note that an audio effect assumes that a sample stream is available to process. If there is no audio stream, then the effect can't do anything.
  - You may need to buffer the stream, depending on your effect. Circular buffers (aka ring buffer, cyclic buffer) are very useful for this (see Wikipedia reference).

- Audio processing in VST uses 2 methods:
  - process( ) :
    - adds results to the output stream
    - more efficient when lots of effects work on stream

- processReplacing( ): optional
  - results replace output stream
  - more efficient with chains or sequences of effects

- Some terms:
  - parameters: user-defined values, obtained from user interface (eg. dial)
  - program: this is a set of parameters. Some VST methods permit them to be saved in files, for easy retrieval later.
  - editor: this is a user-supplied GUI for their plug-in. If you don't do this, then you must use the default GUI of the host application.

- Data types:
  - audio samples are 32-bit float, in range -1.0 to 1.0.
  - parameters are 32-bit float, range 0.0 to 1.0.
  - You will need to convert these to integers or whatever values make sense for your plug-in.

- Structure of audio plug-in:
  - AudioEffectX: base class
    - you extend this
  - constructor of your class: audioMasterCallback
    - host passes this, which you pass to base class constructor
  - some std flags and identifiers are set, and I/O requirements are declared
  - You define some callbacks which the host will repeatedly call. These do the work!

- Event-driven programming:
  - Idea is to let your plug-in fit seamlessly into the host machine, and work in parallel with it and other plug-ins.
  - Host will call your plug-in methods when particular events need servicing.
    - eg. User changes a parameter dial. New value must be transferred to your plug-in code.
  - Done via "callbacks" you define the code for special methods that are called by host.
  - This lets host execute your plug-in along with normal host processing, other plug-in execution.
  - Your callback methods should "do their thing" and release control back to host.
    - If you write an infinite loop, then entire system may stall!
  - This is the same kind of programming involved when developing Windows applications, graphics/game programming, etc.

COSC 4P98 Lecture notes: **VST programming**
March 9, 2012
B. Ross

- Potentially 2 different user interfaces can be used to control your plug-in:
    - 1. Host has a "default interface". Controls will be mapped to your plug-in parameters.
        - Ableton has a "bare bones" interface: simple and lean, but functional.
    - 2. Optionally, you can define a GUI for your plugin.
        - Those controls can be used to set parameters. They can also be operated in parallel with the host controls.
        - You supply "skins".
        - Will pay a penalty in CPU execution: graphics must be redrawn.
    - Some bookkeeping required...
        - get/set parameter values: User's parameter changes must be sent back and forth between plug-in and host. Host interface may be used to change one of your parameters. It is then immediately sent to your plug-in, which will save the value, and use it from then on.
        - Likewise, your plug-in interface may set (default) parameter values, which should be sent to host interface.
        - Send names of parameters (for display in host interface), means for displaying parameter values.
        - Set/get program name: this gives the host the name of your plugin. Needs to be labelled in the host environment.

- Following examples are discussed in detail here:

http://ygrabit.steinberg.de/~ygrabit/public_html/vstgui/V2.2/doc/2.0/examples.html

NOTE: slight change in data types between VST 2.2 and 2.4; so 2.2 version examples may need slight tweaking to compile in 2.4.

- Example 1: **aGain** (Simple gain, or volume control)
    - (p.8-13, vst20sped.pdf; also sample code with VST SDK 2.4 zip file)
    - File 1: aGain.cpp (with aGain.hpp)
        - declaration section: indicate main features of plug-in.
        - setProgramName and getProgramName: set and get the plug-in name
        - setParameter and getParameter: set/get parameter values
            - if more than one, they are indexed (see delay example)
            - note: you may have to convert from/to float/integer, depending on nature of parameter
        - getParameterDisplay: converts param value to string (for host GUI display)
        - getParameterLabel: again, for describing value type in GUI
    - File 2: aGainMain.cpp
        - contains "main"
        - controls interaction between host and plug-in
    - Central processing is in the process/processReplacing

        - take inputs (L and R), and multiply by a gain value to increase amplitude
        - process: adds value to output
        - processReplacing: sets output to value
    - Notice how fgain value is automatically updated via setParameter. The fgain variable should be defined in “aGain.hpp”, visible to all methods.

- Example 2: **ADelay**
    - see sample code in VST SDK 2.4 zip file
    - This has 3 user parameters: Delay, Feedback, and Volume
    - Delay: number of seconds to pause before mixing delay back in
        - multiply by sampling rate → number of samples to wait (and to buffer): “delay”
        - Buffer [0] to [delay] used, with wrap-around (circular buffer).
        - max 44,100 samples (1 second @ 44.1K sampling rate)
    - Feedback: strength of old (buffered) sound, when mixed in.
        - simply a weight applied to old buffer values.
        - Weight < 1.0: they always weaken.
        - If weight = 0.0, no effect. If weight = 1.0, maximal mix (to point of increasing distortion!)
    - Volume (isn’t used (?))
    - Main code for the delay effect:

```
void ADelay::processReplacing (float** inputs, float** outputs, VstInt32
sampleFrames)
{
   float* in = inputs[0];
   float* out1 = outputs[0];
   float* out2 = outputs[1];

   while (--sampleFrames >= 0)
   {
         float x = *in++;
         float y = buffer[cursor];
         buffer[cursor++] = x + y * fFeedBack;  // delay calculation
         if (cursor >= delay)
               cursor = 0; // wrap-around the circular buffer
         *out1++ = y;
         if (out2)  // stereo?
               *out2++ = y;
   }
}
```

- Firstly, the host is giving a chunk (block) of 1-channel input data to process, of size “sampleFrames”.
    - Idea is for plug-in to process a block of samples, and then return control to host.
    - This is preferrable than to (say) call the plug-in for each separate sample: much too slow, would bring system to a grinding halt!

- Your plug-in processes the block, and sends it on output (L&R).
  - more sophisticated would have L&R input, and L&R output
- buffer contains input signal mixed with earlier buffer data
- that buffer may have signals mixed from earlier calls, etc.
- feedback occurs when delayed buffered sound overwhelms current input
- Once all the block is processed, control will resume with host.
- The output is put on pipelines to other plug-ins, and eventually to hardware (sound card).

Some words O'wisdom...

- The SDK documentation will list the different callbacks and facilities available via the SDK.
  - For example, timing (tempo) information from host can be accessed. This can allow a plug-in to synchronize its sound and effects to the beats of the main tune in the host!
  - Electronic dance musicians LOVE tempo-synchronized plug-ins!
- As far as I know, VST does NOT give the plug-in access to the hosts sample data directly. In other words, you can't access sample tables. You can only access audio passed to your Audio effect plug-in.
- If you want to read entire samples into your plug-in (sample-based granular synthesis?), you will need to find a suitable file I/O dialogue utility library (VS.net?). You will also have to convert samples to audio if you want to save them in your plug-in.
- VST 3.51:
  - New facilities, better organization and documentation.
  - LOTS of example plug-ins: see "Plug-ins examples" in installed SDK.
  - Includes access to open-source MDA plugins, which include filters and instruments (soft synths, etc.). You might try them out, to see how things might be done.

Advantages of developing VST plugins

- Many commercial systems can be hosts to your plugin: Ableton Live, Cubase, FL Studio, Adobe products, ...
- Host can do much of audio file I/O, so long as you are implementing an effect.
- GUI's easy to implement using VST "editor" concept. Just provide skin bitmaps for buttons, components.
- Big advantage: Host can do all the difficult timing and tempo stuff!
  - Musicians like plug-ins that synchronize to the host clock.
  - Imagine making a delay or granular engine synchronize grains or effects with a tempo.
- Your plug-in parameters can be recorded, edited, animated by host.
- Host can also integrate with external hardware: sound cards, MIDI interfaces.
  - Your plugin doesn't need to implement these low-level details.
  - Easy way to have external hardware control your plug-in!
- Large VST developer community.
- Commercial possibilities!

COSC 4P98 Lecture notes: **VST programming**
March 9, 2012
B. Ross

References

- http://www.cosc.brocku.ca/Offerings/4P98/software.html
    - latest VST references.
- http://www.cosc.brocku.ca/Offerings/4P98/local/VstSDK/
    - main documentation, with examples, for VST 2.4
    - Some description is here (but for earlier VST 2.0):
        - http://www.cosc.brocku.ca/Offerings/4P98/assignments/vst20spec.pdf
- http://ygrabit.steinberg.de/~ygrabit/public_html/index.html
    - Main site for SDK from Steinberg. The next link has above program examples…
    - http://ygrabit.steinberg.de/~ygrabit/public_html/vstgui/V2.2/doc/2.0/examples.html
- http://www.asktoby.com/#vsttutorial
- Stromcode tutorial (please see me)
- www.kvraudio.com
    - portal for VST technology
- http://en.wikipedia.org/wiki/Virtual_Studio_Technology
- http://en.wikipedia.org/wiki/Circular_buffer
- http://synthmaker.co.uk/
    - a graphical VST editor!