MUS 4711 –
Interactive Computer Music

Course Lecture Notes

MODULE 2: Audio Processing

PART 9: Smooth Transitions and Live/Stored Audio Manipulation

Objects – [line], [line~], [curve~], [snapshot~], [waveform~], [groove~] (advanced applications), [gate] and [gate~], [ggate] (a.k.a. [gswitch2]), [function], [munger~], [amxd~]

[line], [line~], and [curve~] – in the demo patch, I've created three different transition modes to randomly change the parameters of a flanger. None jumps between values (generated by [metro]-[random]) with no smoothing. Line uses [line] to generate a smooth linear ramp from the current value to the target value over a set number of MS. Curve uses the audio rate [curve~] object to do the same with a smoothed exponential curve instead of a linear ramp. To translate from audio to control rate, [snapshot~ 10] polls its input every 10MS and converts that current signal value to a float.

Note that since [random] cannot generate floats, there's a [* 0.01] before the [live.slider]'s to appropriately scale. Also note that an audio rate version of [line~] also exists and works the exact same way, just for audio signals.

Looping – This is just a basic looper. 1 starts the recording, 0 stops it. It records into a named [buffer~] via the signal going to the [record~] object. Here, it has been set to loop automatically, with the start and end points defined (although you can tweak them later in the patch). A [sig~] controls the playback of [groove~], which is a variable-rate looping sampler. As with [record~] it takes the same name as the [buffer~] it's accessing. In addition to defining loop pints and loop on/off, the [sig~] that drives it also controls speed and direction of the playback, which you can play with here. We'll look at a slightly more involved version of that in a bit with the Sampler patch.

Also, note the use of [line] to create a 20MS smooth ramp between values going into the [sig~] – VERY good idea to avoid pops, especially when crossing zero from positive to negative values!

Finally, check out the [waveform~] at the bottom – it displays the contents of a [buffer~] that's it!

Sampling – Here's a basic sampler. Once again a [sig~] is driving a [groove~], reading samples from a named [buffer~]. The difference is that the [expr pow(2., $f1/12)] is being used to transpose the sample from a root key of 0 up. [t i 0 b] is used to 1) restart the loop, 2) stop [sig~], and 3) pass the value of [kslider] to the [expr]. This [expr] equation is the classic sampling algorithm 2^(X/12), where X is the number of semi-tones away from the root key. Positive moves up, negative moves down. Note that there is no root key here. How would you fix that? What would you multiply by, and where would it go?

Note the graphical switch ([ggate] or [gswitch2] – same object, different names) that's used to send inverted values to [sig~] – 0 outputs values to the left, 1 to the right. Supremely useful.

For fun, change [expr pow(2., $f1/12)] to [expr pow(3., $f1/13)] for the Bohlen-Pierce tuning. VERY different effect with a thirteen-note equal tempered scale that has an octave "equivalent" (called a tritave) that occurs around the interval of a just-intoned 12th in 12EDO.

Granulation – Finally, let's look at granulation. Record some audio into the [buffer~], set a speed of 1, and turn on Metro/Random/Loop. Use the [kslider] to set the transposition level, and use the number boxes to set the time between grains, grain time, and delay (all in MS). These values go to the granulator subpatch.

In there, we see that our old friend [gizmo~] is controlling things once again! The metronome fires off a pulse that is used to set a random time for the [function] (just a basic envelope here, who's time is controlled by the Grain Time parameter going to the (setdomain $1) message), and is added to the time between grains to create an offset for the pulse. The [function] also is sent to multiply the audio going to [gizmo~] creating the actual grains of audio. One copy of this pulse is sent to the right outlet after a delay (max 1 sec).

If pitch is randomized, it overwrites the transposition from a [kslider] to randomly create a cloud of sound, that is used to control each grain going into [gizmo~]. This goes into a [gate~] controlled by a [random] synced to the [metro] to randomly output the transposed version out of both the left and right outlets.

Easier Granulation with [munger~] – OK, that was a lot to take in with [gizmo~] based granulation, but it does explain the theory. Let's look at an easier way to do it. In the Package Manager, is a great set of objects called PeRColate from Dan Trueman and Luke DuBois for creating physical modeling instruments and audio fx. We're going to look at the latter, an object called [munger~], so install the package and get ready. [munger~] is a stereo granulator and pitch shifter, and it's got a bunch of inlets for different parameters, but only one argument, the max delay time in MS. The left inlet is your signal, and from there, we're just going to copy things from the help file – Grain Separation (rate), rate variation, size (in MS), size variation, pitch factor (which is what we're interested in!), pitch variation, and stereo spread. I'm just using the example values, 2 0.5 12 500 2 0.7 4 0.5 in a [loadmess] going to [unjoin 7], then into [munger~]. There's another thing we need to do though – we need to pass two messages to [munger~] before we get started – (voices 8) and (tempered), which tells it to behave polyphonically, and to use a tempered set of pitch sieves to shift and retune the pitch – the depth of these is controlled by the grain pitch variation inlet, so we'll set that to 1.

Next, we need to add an audio input or file to the left inlet of [munger~] – I'll just use a file, connect that to a [live.gain~] and output it. Now, playing with the values lets us hear what they do to shift the sound. But I want to make this playable. So I'll add a [notein], and send it to an [adsr~ 10 20 0.9 500] before sending that to multiply the signal before [live.gain]. But this doesn't do anything with pitch… Fortunately, I remember a way from last week to do that – [transratio]. We'll assume that Middle C (MIDI 60) is the root, so we'll subtract that from [notein], send it to [transratio], then to the grain pitch factor, and voila – easy, playable pitch shifted granular synthesis!

[amxd~] effects – We've reviewed a number of ways to create effects over the last few weeks, but there's an even easier one that I've saved until the end. If you already use Abelton Live Suite, you're probably familiar with Max4Live devices, and using [amxd~], we can load those into MaxMSP. Start by going to https://maxforlive.com/

Here we can find free devices to download. I'm going to use the search feature to find a tape glitch device called **College Dropout 2.2** and download it. I save it to the folder that has my patch, and now, I can just drag/drop it into an unlocked patch to add the device! You can add any number of Max4Live devices just like that, and they come with a nice GUI option in the Inspector if you enable "Show View in Patcher."

Here's where it gets cool. Send the left inlet of [amxd~] message (getparams) and it will dump out a list of parameters to the right outlet. So, I can add a message like (Depth $1) and hook that up to a dial or slider to control it. But what if there are parameters with multiple words, like "Depth Variation?"

Simple! Put them in quotes, so you get ("Depth Variation" $1) and Max will match the string.

Now you can easily find and add more effects to Max!

Work on Comp 2

Comp 2 Project – With your new understanding of how to process audio in realtime using some classic effects, AMXDs (from http://maxforlive.com), and on how to record/play audio from a microphone or soundfile, it's time to get creative. Construct a patch that takes audio from a microphone, and passes it to an effects chain of at least three effects (stuff you coded, or AMXDs). For the live portion, you will **not** need to double the dry portion through the output (it would effectively double your voice!) – only output the wet portions. Make good use of the panning law of your choice to place the sounds in the stereo field in real time.

Some parameters for changing both the recorded and live parts must be mapped to a MIDI knob or slider (HINT – Dry/Wet is best here), but others can be preset (use messages, [loadbang], or [loadmess], or the interpolation feature of [pattrstorage]). They can also be changed through the piece as long as they're smooth – refer to the [line] and [curve~] lessons).

You can also use the MIDI keyboard or the QWERTY keyboard to trigger changes, control sample/loop/granulation settings, etc.

Finally, at appropriate times in the performance, you will need to trigger the playback of additional audio files through [sfplay~] or [polybuffer~], or through recording your input in [record~] and [play~]. These can be **anything** musical or non-musical as long as it fits the piece. Mixing with [gain~] is required, but you can pre-pan these parts beforehand in your DAW of choice if applicable.

To accommodate the fact that you will be performing this while changing parameters, you do not have to sing or play an instrument. Instead, you will need to do a dramatic reading of a text from either http://www.gutenberg.org/ or https://archive.org/. Any text is fine, although if it's not in English, you will need to provide a translation. Total length should be 3-5 minutes long. Remember to set everything up in Presentation Mode so that only the necessary parts of the UI are visible to the user. [bpatcher] is encouraged but not required. You will also need to create a score that outlines when to trigger/change things in MaxMSP. You may do this in any way that you want, as long as it works. For inspiration, check out Peter Hulen's piece *Homage & Refuge* for voice and computer.

Remember that all audio files and AMXDs must be in the same directory as your patch.