MUS 4711 –
Interactive Computer Music

Course Lecture Notes

MODULE 1: MIDI and Instrument Design

PART 2: MIDI and Monosynths

Objects – [midiin], [midiparse], [notein], [ctlin], [mtof], [pack], [unpack], [pak], [join], [unjoin], [makenote], [adsr], [!-], [cycle~], [rect~] [saw~], [tri~], [phasor~], [triangle~], [sig~], [cos~], [noise~], [pink~], [scope~], [spectroscope~], [attrui], [ezdac~], [dac~]

MIDI Input – There are three objects to get MIDI into MaxMSP: [midiin], [notein], and [ctlin]. [midiin] takes in raw MIDI data and requires the use of [midiparse] to separate it into useable data like pitch/vel pairs, key pressure, CC values, aftertouch, and bend. It can also export direct MIDI Event Messages (which we'll discuss next week). [notein] outputs pitch, velocity, and channel – nothing else. It assumes that your MIDI device is not outputting CC values. Great if you're just using it for pitch/vel, not great for other thing. [ctlin] outputs CC values from a controller like a knob or fader. It sends out the value, CC number, and channel. If you know the CC number you're going for, you can use that as an argument and permanently look for values on that controller.

Pack, pak, and join – We've already briefly looked at [unpack], which splits a list of elements into any number of outputs. The arguments are simply the type of data you're expecting it to output from left to right. But how do we convert things *into* a list to unpack later? Well, there's three objects commonly used: [pack], [pak], and [join].

[pack is functionally identical to [unpack], just in reverse. One major thing to keep in mind is that it only outputs values when the left/hot inlet receives a value or bang. This is a slight problem if you're constantly changing one of the cold inlet values. Fortunately, [pak] is there to help. As with [pack] you have to specify the type of data, but it outputs when **any** inlet receives a value. Great for constantly changing data. Finally, there's [join] which joins together literally any data type (except audio!). The only argument it takes is the number of inlets, and using the @triggers attribute, you can specify which inlet or inlets cause the output (0 is left and then each right inlet is numbered up from there, -1 makes all inlets hot).

Oscillators and ADSR – There are a number of oscillators built into MaxMSP that can be used for audio or control rate signals. All of these are being summed at the [*~ 1.] object ([*] and [+] auto-sum!), which is controlled by [adsr~]. The initial values are in order, A, D, S, and R. All but Sustain are times in MS, while Sustain is a float from 0 – 1 (0 – 100%). These values are used to generate the classic ADSR envelope.

The MIDI data goes through [midiparse] to get the pitch/vel pair, then [unpack] sends the velocity to [adsr~] after it's divided by 127 to get it in the 0 – 1 range. The MIDI pitch goes to [mtof], converting pitch to frequency, and is then routed to the various oscillators.

[cycle~] is a classic sine. It's *technically* antialiased (meaning it will never exceed the sample rate and thus cause fold over distortion and alias frequencies), but can also be used as a control oscillator. Other antialiasing oscillators are [saw~], [rect~], and [tri~]. Guess what they produce. Turn up their respective [gain~]s to see them in the [scope~] and [spectroscope~]. Note that [spectroscope~] has an [attrui] object attached to it, which creates a menu of attributes and values you can use to tweak the behavior of an object during performance. Very useful…

Control oscillators are [cos~], which generates a cosine wave, and must be driven by a [sig~], which generates a DC offset signal. Note that it ONLY outputs in the 0 – 1 range if driven by [sig~], making it unsuitable for audio use. [Phasor~] generates a saw ramp, and while it *can* work in the audio range, take a look at the spectrum and compare it to [saw~] – helluva difference in the noise signal! Finally we have [triangle~], which requires a 0 – 1 float to set the slope of the wave. Like [cos~] it has to be driven by another oscillator, here, we're using [phasor~]. Try swapping in [sig~] to see what happens, then try replacing the [sig~] driving [cos~] with [phasor~] why is that happening?

Kslider and Makenote – [kslider] is a super useful UI control device that creates a keyboard you can click on with the mouse. It sends out MIDI pitch on the left and velocity on the right, but has no sustain. Since we don't want the pitches to drone on forever, we fix this by sending that to [makenote], which generates MIDI on/off pairs for a given pitch. It takes two arguments, a default velocity (doesn't usually matter as we'll override it pretty much immediately), but the second argument, duration in MS is the key. This lets us create a nice/plucky sound when sent to an [adsr~] envelope and kill those damn drones.

Noise – There are only two types of noise in MaxMSP: [noise~] generates white noise, and [pink~] generates pink noise. That's it!

Inversion Blending – This is one of the most useful tricks you will ever use. If we want to blend between two signals, values, etc., the easiest way to do it is to multiply them by floats from 0 – 1. As one goes up, the other must go down so that they sum to 1. The easiest way to do this is to use a [slider] or [dial] with a float range of 0 – 1 going to a [*~] for the signal on the right. Before going to the [*~] on the left, we're first going to run it through [!- 1.]. This is the same as the usual [-] object, except the inlets are reversed! So here, it's taking 1 and subtracting whatever the hot inlet is from it. So if the dial is at 0, [!- 1.] outputs 1. If it's 0.5, [!- 1.] outputs 0.5, and if it's 1, [!- 1.] outputs 0. These are perfectly inverted values of the unaltered values controlling the right signal. Instant mixer!

Summing – All [*~] and [+~] objects will sum any number of signal connections going into the left/right inlets. However, as you'll quickly see, this can result in clipping and distortion if they aren't scaled. The easy way to scale is to use a [*~] object with an argument that is the reciprocal of the number of signals going in. So if you have three, it's [*~ 0.33], two would be [*~ 0.5], and eight would be [*~ 0.125]. Why not just [/]? It's a convention from other programming languages, where it tends to be more CPU intensive and more importantly, it can result in rounding errors that aren't there when multiplying.

Max Puzzle 3 – Using any of the antialiasing oscillators, a noise source (white or pink), and [adsr], create a monosynth that uses three of the same oscillators tuned to root, 5th (+7), and 8va (+12), and can balance between the oscillators and noise. Be sure to properly scale the oscillators so they don't clip! Use the inverted subtract [!-] and a dial with a range of 0-1 to control the balance between the oscillators and the noise source. Everything should be controllable by **both** a [kslider] and a [midiin]. You may use any needed number of sliders or dials, and any other objects discussed in class as needed. Everything should go to a [gain~] and an [ezdac~] at the end. **Remember to use [makenote] with [adsr] to generate note off pairs! You might not need it for your [midiin] though…**