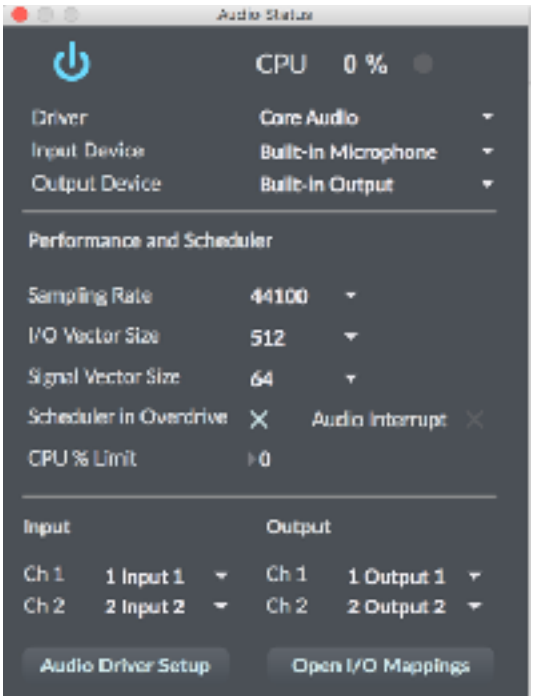
INPUT, EFFECTS, ANALYSIS & SAMPLING

Max/MSP provides us with the ability to customize our own music system. While we've looked at methods of internal synthesis, and means of controlling this synthesis, there's also the potential of using Max/MSP as an **extension** of your instrument. This can be accomplished by using a mic to process, resample and analyze your instrument's input.

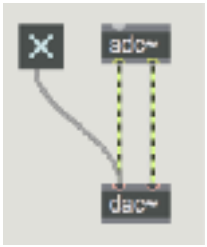
INPUT

The same way our output needs a digital to analog converter (which we've been using as our **ezdac~** object, though there is also a pure **dac~** object), our input needs an **analog to digital converter**. What do you think this object is called? That's correct, it is the **adc~** object! The adc~ object accepts audio input into our patch from external mic source. Most computers come with a built-in mic; if you want a mic of better quality, one needs an **audio interface**, which allows professional audio equipment (guitar, synthesizers, mics, etc) to be recorded or processed by the computer. If you do use an audio interface, you must go to **Options->Audio Status** and change your input source. The arguments for your adc~ object determine which channel from the input the object is connected to.



For the purposes of today's lesson, we will stick with the built-in mic in our computers. Let's connect our adc~ object to a dac~ object, with a toggle to turn on/off our dac~. Before we turn anything on, *plug in your headphones*. What would happen if we didn't plug in our headphones? Feedback is something to always be conscientious of when working with live input. Though, as we saw last lesson, and as musical greats such as Jimi Hendrix and Merzbow have demonstrated, feedback can also be used for convincing musical effect.

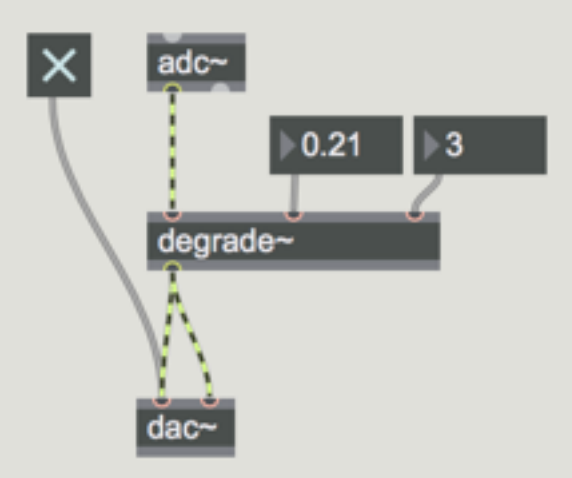
PATCH: adc_1.maxpat



You should hear the sounds of the room streaming into your headphones. Let's begin processing these a little bit.

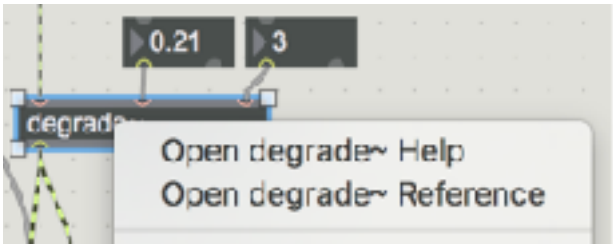
EFFECTS

Max/MSP allows for a wide range of very strange effects. Now that we've learned about sample rate, let's look at sample rate reduction with the **degrade~** object. Let's make our effects **mono** for now (meaning only one channel, as opposed to **stereo**, two channels). Hovering over degrade~'s inlets, we see that one inlet controls 'sample rate ratio' and the other controls bit depth. Bit depth refers to the vertical quantization of the waveform - bits indicate the number of possible values for any sample. For example, with 8-bit resolution, there are 256 possible values for any given sample (2^8). **Bit-crushing** has a very distinct aesthetic. Try playing with these values.



Note - if you need any further information about any object, you can right click and go to the 'Help' or 'Reference' file. Feel free to copy/paste from the examples given in those files, as well as to explore the 'See Also' section in the inspector for help files.

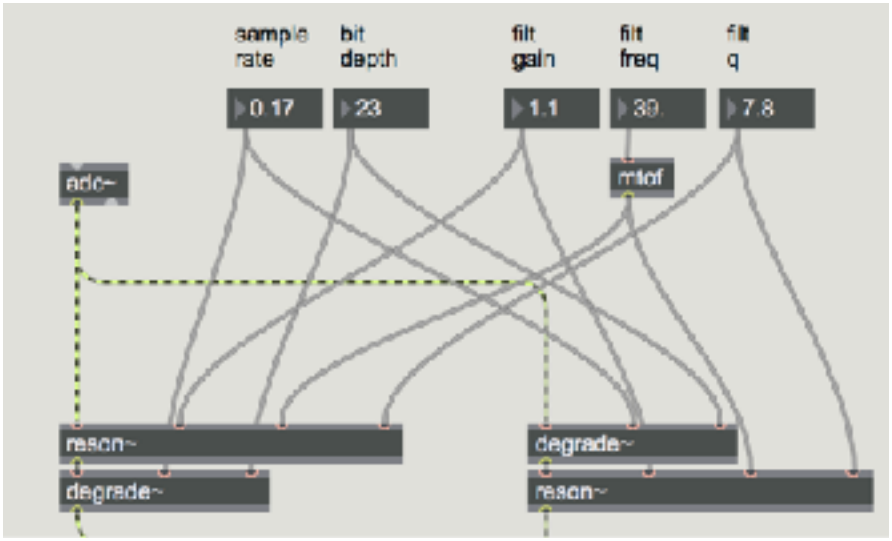
PATCH: adc_2.maxpat



Wow! That sounds exquisitely nasty. How can we expand on this? As mentioned when studying Ableton effects, whether the effects are **parallel** or in **series** will shape the behavior of the sound. What if we put a reson~ object (bandpass filter) before our degrade object? How is this is different than putting it after our degrade object? Try both out and compare the resulting sound.

PATCH: adc_3.maxpat

We can use a selector~ object to toggle between these settings. We can use the same inputs for both sides, maybe even labeling them using comments (hit C and you can write a comment on the patch).

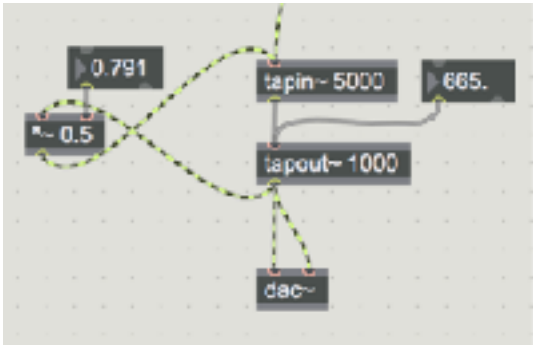


PATCH: adc_4.maxpat

This demonstrates how you can use the same control data to manipulate completely different signal flows within Max/MSP, resulting in dramatically different, yet interchangeable, aesthetics.

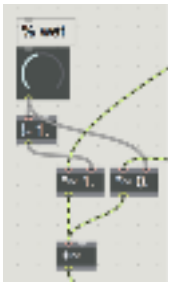
Delays in Max/MSP are implemented using the **tapin~** and **tapout~** object. The tapin~ object records the incoming sound for a period of time equal to its first argument in milliseconds. The tappet! object reads from that buffer according to the time of its first argument. On its own, this sounds identical to what's happening, just delayed for the time equal the tapout~ argument. Let's create some feedback and see how that sounds.

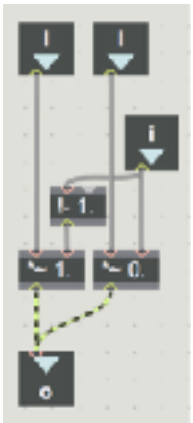
PATCH: adc_5.maxpat



Now this is sounding a bit juicier. What if we wanted to cross fade back to our original sound? We can use the same signal flow from our subtractive synthesis patch, where we use one dial to crossfade between sounds.

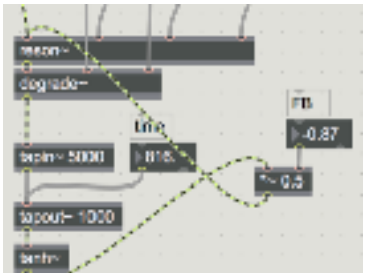
PATCH: adc_6.maxpat





This signal flow seems quite useful, and when work with effects we'll often want the ability to move between a **wet** and **dry** signal. We can copy/paste this into its own patch, save that patch, and then re-use it in all of our patches. Within this new patch, use the **inlet** and **outlet** objects to create input and outputs for the patch. Where would we put these? What kind of inputs/ outputs do we want? Let's save our patch as xfade~, and then use it in our patch.

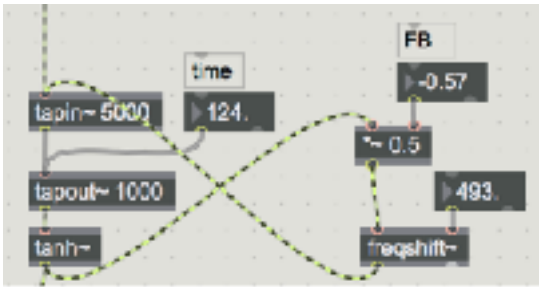
PATCH: xfade~.maxpat
PATCH: adc_7.maxpat



Another exciting part of delays in Max/MSP is the ability to place effects *inside* the feedback chain. Let's first place a **tanh~** object at the output of our tapout~ object. This object will clip our output at -1 and 1, while creating a smooth roll-off as the signal values

approach these limits, thereby preventing harsh clipping and feedback explosions. Connect the feedback to the reson~ object so that our feedback chain includes the filtering and distortion. Now our delay is colored by two completely different effects.

This technique of adding effects is particularly convincing when used with effects that have cumulative changes in the sound. For example, the **freqshift~** object changes the frequency of our incoming sound by a set amount. However, when inserted into a delay's feedback line, the frequency keeps shifting. Try changing the delay line so that it just connects to the freqshift~ object to see how it sounds.

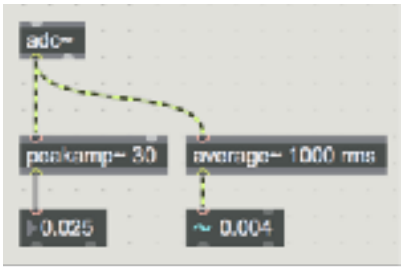


PATCH: adc_8.maxpat

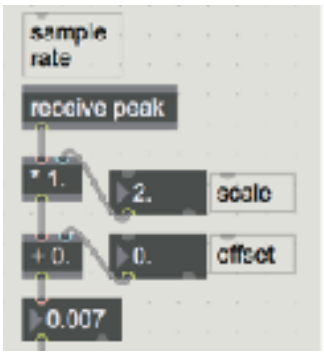
ANALYSIS

Like the sidehcain effects in Ableton, we can analyze incoming signals in Max/MSP and use that to control our effects. What sets Max/MSP apart from Ableton's effects is its complete modularity, so that the envelope we used to control our filter, or the incoming signal we used to control our compressor, could instead be used to control distortion, reverb, or any synthesis parameter.

There are several useful objects that can be used for amplitude analysis in Max/MSP. The **peakamp~** object takes the peak amplitude value every bang, or every millisecond time determined by the object's argument. This object outputs a floating point number.The **average~** object takes the average amplitude every X number of samples as determined by the first argument, which it outputs as a signal. The second argument sets the different averaging mode. Since signals fluctuate between -1. and 1., it is best to use the absolute or rms mode if analyzing an audio signal. Recall that rms (root mean square) takes the root of the value squared. This is the most computationally heavy, but most accurate means of taking the average.



PATCH: analysis.maxpat

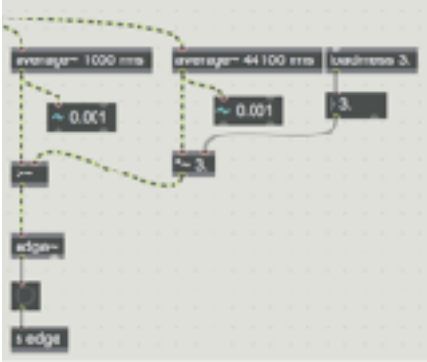


Let's bring this analysis into our effect patch and send the values to parameters. We also want to add a multiplication to our analysis control to scale the input, as well as addition to create a base value for our effect. Let's first map it to very obvious effects, such as our

filter frequency and our sample rate reduction, although ultimately we could connect these analysis values to anything. Arbitrarily, we'll send our average to our filter frequency, and our peak amplitude value to our sample rate ratio.

PATCH: adc+analysis.maxpat

We can expand upon our analysis so that any sort of percussive hit creates a bang. How do we know when there has been a percussive attack? It's loud! How do we know if something is loud? How can we represent that within our analysis objects?



One approach would be to compare a longer average value with a more immediate average value. If the immediate average value is significantly higher than the longer average value, it most likely indicates that there's a percussive hit. We can use the **>~** object, which outputs a 0 if the left input is less than the right, and a 1 if it's greater than the right, to see which average is greater. We can then use the **edge~** object, which outputs a bang when the signal changes from a 0 to 1, to give us our output. What could we drive with this bang? Why not try our sequencer from before? Let's also open our FM synth so we can hear our sequencer being read. We now use snapping to play our sequencer!

PATCH: edge_analysis.maxpat
PATCH: edge_sequencer.maxpat
PATCH: FM_tutorial_synth.maxpat

What if we wanted to route the audio from our FM patch to our effects processing? How might we do this? That's right! Through the send~ object. You are learning so quickly, my Max minions.

PATCH: fm_synth_send.maxpat
PATCH: fm_effects.maxpat

There are numerous other parameters which can analyzed. For instance, we could use the **zerox~** object, which counts the number of zero crossings per vector (a chunk of samples that's sent to the DAC; can be changed via the audio preferences, which can save computational power if necessary). We could use **retune~** with the pitchdetection parameter turned on (using the **pitchdetection** message set to 1), where the second outlet puts out our pitch. Let's route these parameters to a few more effects.

PATCH: fm_effects2.maxpat
PATCH: analysis_pitch&zerox.maxpat

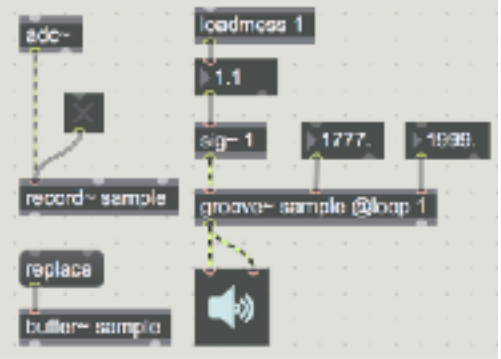
How could we expand upon this? For instance, how could we send out a bang each time a certain pitch is detected? Or how could we send out a bang if we hear a rhythm of a certain length? How can we detect just the energy within the low end of the spectrum? The high end?

SAMPLING

We've covered a lot so far, but one last component to using Max/MSP as accompaniment is live resampling. We can play back samples in Max/MSP by using a combination of the **buffer~** and **groove~**. Buffer~ loads a sample, and groove~ plays it back (there are numerous objects for sample playback, but groove is the most intuitive and user friendly). Groove~ accepts a signal in its first inlet, loop minimum in its second, and loop maximum in its third. It needs a loop attribute to turn on looping. Replace the buffer~ sample with the replace message (must be a wav or aiff file).

PATCH: sample1.maxpat

As you can hear, even with just manipulation of the playback rate, we can get some very interesting sounds. Add some number boxes to the loop min and max to control the loop points (in milliseconds). Bare in mind that all of these parameters could be controlled via sequencers or audio analysis.



We can use the **record~** object to record new audio into our buffer~ object. Record~ is controlled via a toggle (1 to record, 0 to turn off). Let's play with this a bit.

PATCH: sample2.maxpat

Everything we've done here can be combined with everything we've done previously. We could use recorded sample values as input to audio rate sequencer. We can use bangs from our analysis patch to control our record~ object's toggle. We can mix our audio together and send it through layers of processing. Next class, we'll dive into bringing these elements together into a workable system, focusing on principles of GUI design, encapsulation, and routing in Max/MSP.