MUS 4711 –
Interactive Computer Music

Course Lecture Notes

MODULE 2: Audio Processing

PART 6: Mics and Audio Files, Input, and Recording

Objects – [key], [keyup], [sel], [playlist~], [sfplay~], [ezadc~], [adc~], [buffer~], [record~], [play~], [polybuffer~], [dropfile], [combine], [prepend], [wave~], [groove~] (basic applications, only)

Key and Keyup – We can easily get MIDI into and out of MaxMSP, let's expand on that to get keystrokes from your QWERTY keyboard. There are two ways to do this: [key] and [keyup]. [key] outputs when a key is pressed, and [keyup] when a key is released. Note the differences… Also note the differences between ASCII and independent values for special keys like ESC and SPACEBAR. In general, I recommend ASCII for interoperability between platforms. Send those to a [sel], which outputs bangs based on matching the input to its arguments. The order of the arguments determines the outlet order, not alpha/numerical. You'll be using this object in particular a lot.

Playlist – The easiest way to get MaxMSP to play an existing audio file is to drag/drop it into an unlocked patch, which creates a [playlist~] object. You can also use built in audio clips from the menu on the left toolbar. For non-built-in clips, they must be in the same folder that the patch is saved in. Dropping multiple clips at once creates a multi-tiered playlist, where files can be changed by sending ints – 1 is the top, and it goes up for every file, until you hit the bottom. All [playlist~]'s include play/stop and loop controls. One issue though – the graphical display of [playlist~]'s can be rather CPU intensive, especially if you load it into a [poly~]. For efficiency, it's better to use [sfplay~].

Sfplay – [sfplay~] stores and plays a sound file using messages. As an optional argument, you can specify the number of output channels (it's stereo by default). To load a file, you send it the message (open file.type) – e.g. (open test.wav). It MUST include the file extension, and the file must be in the same folder as the patch. Controls are easy (1) plays it, (0) stops (a toggle also works well). (loop $1) toggles loop (1 = on, 0 = off), and you can also use the (pause) and (resume) messages to pause playback. As this is not a graphical object, I recommend using this in particular for structuring cues,

Analysis of *Hypnos* and the use of [sfplay~]

Record, Buffer, and Play – OK, that's all fine and dandy for audio files but how do we get live audio in? [ezadc~] and [adc~]! [ezadc~] defaults to inputs 1 and 2 of your interface, while [adc~]

lets you specify the channel as an argument. Pretty easy, but beware to set the interface volume properly…

To record this, we need two things: the [record~] object and a [buffer~] object to store the audio in, both of which have the same name as an argument. [buffer~] also requires a time length in MS that is the max amount of time to store in the memory buffer it creates. You don't have to hit the max, but it won't record beyond it. [record~] takes a (1) to start and a (0) to stop – I've rigged up a little delayed toggle action to auto stop one MS after the max time is hit.

To play from a [buffer~], we use [play~] – again, the argument we give it is the same as the [buffer~] we are going to access. Once again, (1) to start and a (0) to stop. Careful with the timing here!

To view the contents of a [buffer~], you can either send it the (open) message or double click on it.

Polybuffer – [polybuffer~] is a [buffer~] on steroids. It lets you load up a group of samples as a collection of [buffer~]'s. To easily load it, I'm using [dropfile], which creates a space to dump a file or folder in a patch, then outputs the system path to (readfolder $1) which loads it in proper syntax in [polybuffer~]. To select a sample for playback, I'm using [combine @triggers 1], which combines the name of my polybuffer~ (demoBoard) with a period, then the number coming in to the right inlet, which also triggers the output. This goes to (set $1) which tells [groove~] what sample to select from the [polybuffer~] (demoBoard.1, demoBoard.2, demoBoard.3, or demoBoard.4). Note that [groove~]'s argument is the same name as the [polybuffer~], minus the .1, .2, .3, etc.

We'll explore [groove~] in depth later, but for now, it's a variable rate sample player that requires a [sig~] with a value of 1 to play or 0 to stop. Perfect use of a toggle as a control with a little bit of delay logic to first send [groove~] the (1) message to trigger forward play, then sends (1) to [sig~] and a (0) after a second, short delay.

Buffers and Wave~ – All of this is going to combine to one little synth trick that I've been saving until we covered [buffers~], the [wave~] object. [wave~] is a wavetable oscillator, that has a named [buffer~] as it's argument. We can then load the buffer with a sound, and play it from [wave~] using a [phasor~] object to ramp through the points in the [buffer~]. Simple, cool, and effective! I'd strongly recommend googling Adventure Kid Wave Forms and Void Vertex – both are single cycle waveforms (which means 1 Hz), and so they're perfect for using in [wave~].

Max Puzzle 5 – Now that you know how to get audio files and a mic into MaxMSP, let's have some with audio files! Using the objects covered in class, you will create a soundboard (here's an example) that plays different sounds based on the number keys pressed. Number 1 should control a [polybuffer~] that randomly cycles through at least five different but related sounds (perc samples work great here). Number 0 should record a one second clip of audio from mic

channel 1, and number 9 should play that audio back. No audio should be looped. **Remember that your patch and audio files need to be in the same directory**…