Karol Wadolowski

ECE-413 Music and Engineering

Homework #3 – MIDI

Overview

In this assignment the following files were added on top of the real time synthesis files that were supplied on GitHub:

- main.m: Runs the 3 supplied MIDI files
- playMIDI.m : Takes a MIDI file along with other parameters and plays the MIDI file
- objMusic.m : An object which stores an array of notes that will be synthesized and played

Of the supplied real time files several were altered to have more functionality and to make them compatible with the three new added files (listed above).

How it works

The playMIDI function works by taking the file name of the MIDI file that is to be played. The file is broken up by bytes and stored in array of integers with values from 0-255. After obtaining the array of bytes preprocessing can begin. The initial header is decoded as to find the number of tracks, the MIDI type, and the pulses per quarter note amount (ppqn). After getting this initial information the index is set to the location of the first track header. Once a track is found its length is used to locate the next track header so that the next track can be appropriately decoded. After finding the next track start position the current track is decoded. The track is decoded by alternating of decoding delta times and decoding commands. Delta times always precede a command. After decoding the delta time the current time is updated to reflect the change. The next two bytes store a command and channel unless you are in running status in which case it is the command values (ex, note and velocity for Note On). Once a command is found the appropriate switch case can handle the command. After a command has been completed the index is incremented appropriately to the location after which the command ends. The loop starts from the top to check if the new byte is at a header location. If yes, the header is handled as before. Otherwise the byte is a delta time command and is handled appropriately. If the command received is a non-MIDI event then the appropriate switch case is used to handle it. This is used primarily for the end of track command ‘FF 2F 00’ and the tempo info command ‘FF 51 03 tt tt tt’.

The Note On command creates a new entry in a cell array that stores the starting time of the key press. In the case that the Note On command is used as a Note Off command the key end time is updated in the cell array entry. The Note Off command is the same as the Note On with velocity equal 0. The patch change command updates the patch for its corresponding track. A patch

change corresponds to an instrument change/ synthesizer change. The other 5 commands are treated as do nothings in the code written.

After going through the entire bytes array the key presses with their corresponding start and end times, note, and instrument (patch/synthesizer) are stored in a matrix (musicNotes) that is then sorted in increasing order of start times. The next step is for the patches to be mapped to synthesizers. This is done through the oscillTypes input of the playAudio function. oscillTypes has all the synthesizers that will be used for a patch. The mapping from patches to oscillTypes is modular explained below:

Given oscillTypes = {'synth0','synth1',…'synth(N-1)'} and M different patches (0-(M-1)) in the MIDI file. Each synth does not need to be unique.

If $N \geq M$ then match patch0 to synth0, patch1 to synth1, …, patch(M-1) to synth(M-1).

If $N < M$ then match patch(k) to synth(kmodN). ie some patches may share the same synthesizer.

After matching the patches to the synthesizers, the synthesizers for each note are stored in oscParams.oscTypes. This is then used in objSynth file to assign the proper synthesizer to each note. Before synthesizing, the input xSpeed into playMIDI changes the tempo and note times to speed up or slow down the file. This allows the user to play 'furelise.mid' 4 times as fast for example. The musicNotes are then used to create a objMusic object which is a scaled down version of the objScale object type. The objMusic object allows the notes to be prepared in arrayNotes much more easily. After making the objMusic object the playAudio function is used to play the music.

The following oscillTypes are available: 'sine', 'SUB' (subtractive synthesis), 'FM' (frequency modulation), 'ADD' (additive synthesis), and 'WS' (wave shaping). These different synthesizers were implemented within objOsc and were copied from the homework 2 with minor tweaking.

Results and Discussion

For the files MIDI files that where provided I found that certain synthesizer combinations worked better than others for a given file. I have recorded my preferences for oscillator types and times speed (xSpeed) settings in the main.m file.

From testing I found that additive synthesizer produces the worst results. The additive synthesizer I used was supposed to make bell sounds based on a figure in the Jerse text. The duration of the note from the MIDI file is too small for the bell to sound good and thus becomes a horrible mess. I found wave shaping to only work kind of well for the 'furelise.mid'.

Among the three files, the ‘furelise.mid’ with the settings in the main.m file produces the best music in my opinion.