C Programming (MPATE-GE 2618) Final Project

Kiran Kumar & Ryan Edwards – Fall 2014

Title

A Pretty Nifty Audio Synthesizer

Authors

Kiran Kumar & Ryan Edwards

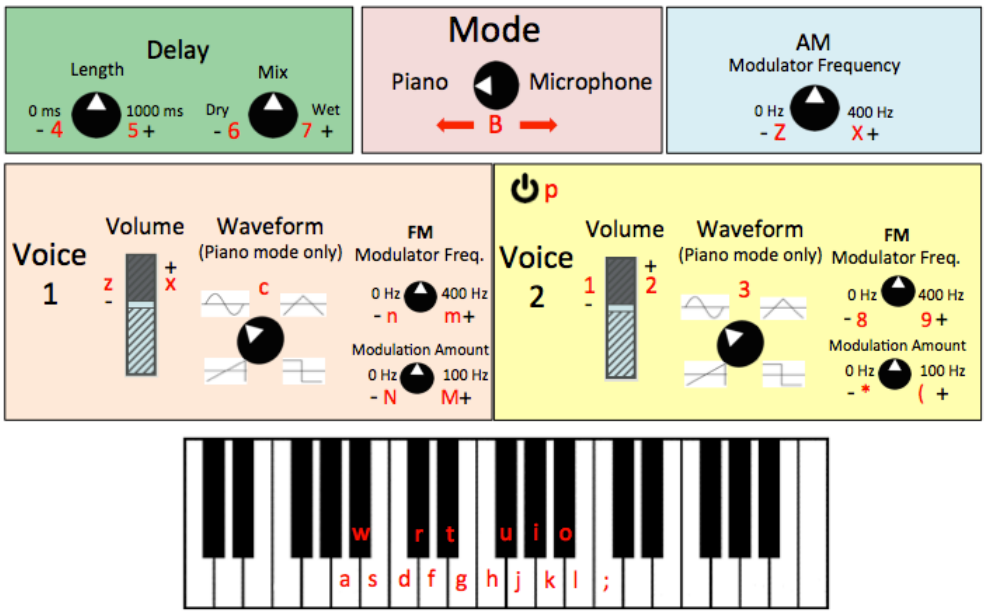
Abstract

We built an audio synthesizer based on a keyboard interface. The synthesizer can generate two different voices and combine them via additive synthesis. Controllable parameters include wave type, delay amount and mix, and AM & FM modulator frequencies.

What

The program uses the PortAudio library to synthesize audio and the OpenGL library to display a visual of the synthesizer along with keyboard commands.

The synthesizer image (representative of its features) displayed when running the program is as such:



The structs to encapsulate the necessary data are as follows:

```
27 //Data for a single keyboard note
28 typedef struct {
29     float frequency;
30     char *noteName;
31 } noteInfo;
32
33 //Audio signal source
34 typedef enum signalSource {
35     OSCILLATOR,
36     MICROPHONE
37 } sigSource;
38
39 //Oscillator wave type
40 // (applies only to oscillator mode)·
41 typedef enum wave {
42     SINE,
43     SAWTOOTH,
44     SQUARE,
45     TRIANGLE
46 } waveType;
47
48 //PortAudio Data
49 typedef struct {
50     float frequency;
51     float volume;
52     float volume2;
53     float amModFreq;
54     float fmModFreq;
55     float fmModFreq2;
56     float fmModAmt;
57     float fmModAmt2;
58     int delayLen; //delay length in samples
59     int delayLenMs; //delay length in milliseconds
60     int prevDelayLen;
61     int delayReader;
62     int delayWriter;
63     float *delayBuffer;
64     float delayPctDry;
65     float delayPctWet;
66     sigSource sigSrc;
67     waveType sigWaveType;
68     waveType sigWaveType2;
69 } paData;
```

Mode

The user can select either “piano (oscillator) mode” or “microphone mode.”

In piano/oscillator mode, the program will generate a simple waveform of a certain type (discussed later), the pitch depending on the keyboard note pressed. The synthesizer has two separate voices, and each voice has its own waveform control. The second voice can be turned on and off at will.

In microphone mode, the synthesizer will play back audio input through the computer’s microphone. Additionally, the system will continue implementing its delay and amplitude modulation effects on the input, and the second voice on the piano mode is still available to play to combine with the input signal.

The signalSource enum defines what mode the synthesizer should perform in, and a variable of this enum type is part of the main paData struct

```
//Audio signal source
typedef enum signalSource {
    OSCILLATOR,
    MICROPHONE
} sigSource;
```

```
//PortAudio Data
typedef struct {
    float frequency;
    float volume;
    float volume2;
    float amModFreq;
    float fmModFreq;
    float fmModFreq2;
    float fmModAmt;
    float fmModAmt2;
    int delayLen; //delay
    int delayLenMs; //del
    int prevDelayLen;
    int delayReader;
    int delayWriter;
    float *delayBuffer;
    float delayPctDry;
    float delayPctWet;
    sigSource sigSrc;
```

If “oscillator” mode is chosen, the frequency in paData’s frequency is used for both voices (only for the second if “microphone” mode is selected). The frequency will be determined by an array of note names and their frequency values.

```
typedef struct {
    float frequency;
    char *noteName;
} noteInfo;
```

```
//on the computer keyboa
noteInfo keyMap[123];
```

```
keyMap['a'] = (noteInfo*)malloc(sizeof(noteInfo));
keyMap['a']->noteName = "A3";
keyMap['a']->frequency = 220.00;

keyMap['w'] = (noteInfo*)malloc(sizeof(noteInfo));
keyMap['w']->noteName = "A#/Bb3";
keyMap['w']->frequency = 233.08;

keyMap['s'] = (noteInfo*)malloc(sizeof(noteInfo));
keyMap['s']->noteName = "B3";
keyMap['s']->frequency = 246.94;

keyMap['d'] = (noteInfo*)malloc(sizeof(noteInfo));
```

Volume

The user has the option to increase and decrease volume on each of the synthesized voices (and the input signal in mic mode). We set minimum and maximum values to minimize distortion and prevent hearing loss ☺

The values at any point in time for the two voices are stored in the paData struct's volume members

```
//PortAudio Data
typedef struct {
    float frequency;
    float volume;
    float volume2;
```

Waveform Type

When the synthesizer mode is set to “Piano,” the user can choose one of four waveform types to play for each voice (the user can always use this for the second voice):

- Sine
- Sawtooth
- Square
- Triangle

Like with “Mode,” the signal waveform type is represented as an enum (waveType) stored in the paData struct:

```
typedef enum wave {
    SINE,
    SAWTOOTH,
    SQUARE,
    TRIANGLE
} waveType;
```

```
  waveType sigWaveType;
  waveType sigWaveType2;
```

Delay

The user can control two delay-related parameters: the length of the signal delay in milliseconds (between 0 and 1000), and the percentage of dry and wet signals to play (the mix).

The delay length members, the percent of wet signal, and the percent of dry signal are all stored as floats and integers in the paData struct

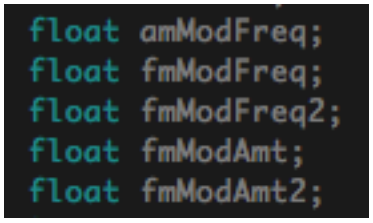
```
int delayLen; //delay length in samples
int delayLenMs; //delay length in milliseconds
int prevDelayLen;
int delayReader;
int delayWriter;
float *delayBuffer;
float delayPctDry;
float delayPctWet;
```

AM & FM Modulation

The signals from the synthesizer act as a carrier wave one AM modulator (for the whole output) and two FM modulators (one for each voice). The user can specify a

frequency for each of these modulating waves as well as modulation amounts for the two FM waves.

The values are all floats in the paData struct



API

Below are the signal processing related functions we implemented in this project:

```
float dbToAmplitude(float decibel);

void setDelayLen (float delayLen, paData *data, float sampleRate);

void addDelayLen (float delayLen, paData *data, float sampleRate);

void createDelayBuffer (float *buffer, int bufferLen);

void freeDelayBuffer (float *buffer);

void createFMBuffer (float carrFreq, float modFreq, float modIndex, float
*buffer, int numSamples, float sampleRate, float *phase, float *prevPhase);

void createSineWave (float freq, float *buffer, int numSamples,
        float sampleRate, float *phase, float *prevPhase, float *FMbuffer);

void createTriangleWave (float freq, float *buffer, int numSamples,
        float sampleRate, float *phase, float *prevPhase, int *direction,
float *FMbuffer);

void createSawWave (float freq, float *buffer, int numSamples,
        float sampleRate, float *phase, float *prevPhase, float *FMbuffer);

void createSquareWave (float freq, float *buffer, int numSamples,
        float sampleRate, float *phase, float *prevPhase, float *FMbuffer);
```