Project Proposal

Group Members:
Vaishnav Janardhan, Rob Katz, Carlos Rene Perez, Albert Tsai

1. Describe the language you want to implement.

Analog Additive Synthesis Language

Sound is nothing more than variations in air pressure and, as such, can be predictably generated and modified using mathematical formulas. Synthesizers have been doing this for decades using analog circuitry and some simple digital effects (delays, etc.). In the past couple of decades, these analog circuits have been replaced by digital counterparts. However, the basic elements, such as oscillators and envelopes, and the methods for combining them, remain the same. The goal of this language is to encapsulate common data types and methods, corresponding to these classic elements, which will facilitate the creation of original sounds.

There is a similar project from the Fall 2006 PLT class (Music Mixing Language [MARS]). However, that language is concerned mostly with the arrangement of music tracks, and minor sound alterations like fading in/fading out. This idea is more fundamental - we want a language to help create sounds from scratch.

Our "Hello World" program will be a short piece of code that can output to a sine wave to the file, Sound.wav.

What is a sound?

Every program outputs a CD-quality sound(44.1k, 16-bit WAV file). This sound is the summation of the outputs of mixers and oscillators. The output of a mixer is, in turn, the summation of the outputs of oscillators and other mixers. Oscillators are associated with a waveform and a central frequency. Every oscillator has an input for frequency control and an input for amplitude control. Each input may be attached to a constant, an envelope, or another oscillator. The output of an oscillator is a modified version of the waveform associated with it. This output cannot exceed a frequency of 20 kHz or an amplitude of 1.
Envelopes are graphs that portray change over time. This graph can be described by a set a pairs. Each pair is of the form (time, amplitude). Both values must be between 0 and 1.

When an oscillator is attached to either the frequency or amplitude input of another oscillator, it's output acts as a control signal. For example, say osc1's output is a 1Hz sine wave at half maximum amplitude (control signal). This oscillator is attached to the frequency input of osc2, which is associated with a square wave and has a central frequency of 440 Hz. The output of osc2 would be a square wave who's frequency varies. The width of this variance from the central frequency would be half the maximum allowed amount (due to the amplitude of osc2). The central frequency occurs at each zero crossing of the control waveform, in this case every half second (due to the frequency of osc2). In other words, the sine wave output by osc1 can be viewed as a graph where the x-axis is time, the y-axis is the frequency of osc2, and a when y=0 osc2's output it's central frequency.
An envelope's value is tracked similarly, but they do not loop as oscillators do.

For the purposes of our language, we shall define a sound with the following syntax:
A,B,C are Signals.

"A : B + C" = A is the output of adding(super positioning) B and C together.
"A : B, C" = Used to separate a list of attributes.
"A : B[C]" = Signal1 is modified by Signal2.

And we shall define a sound as:
Sound : Mixers(s) + Oscillators(s);

Mixer : Oscillator(s) + Mixers(s);
Oscillator : Waveform, Amplitude[Envelope], Frequency[Envelope];
Envelope : Graph describing modification over specified time duration;
Constant : Modifier unrestricted by time;
Modifier : Integer value multiplied against a specific attribute;
Waveform : The shape of a signal;

2. What problems does this language solve and how should it be used?

The Analog Additive Synthesis language allows a user to create an enormous variety of sounds. No stand-alone synthesizer and few software synthesisers allow for such flexibility in terms of the number of components a sound can include or accessibility to each component's characteristics. The language is most suitable for those wishing to generate simple to complex analog-type sounds and noises, perhaps to be manipulated using other audio software. While it could be used to generate melodies or harmonic progressions, this would most likely prove terribly time-consuming and not worth the effort.

3. Data type for language

The basic data types of the language are:

1. Contants: It is an integer constant while defining the frequency of the oscillator and it is a floating point number when defining the amplitude of an oscillator.
2. Envelopes: Envelope defines the variation of either the frequency of the amplitude over time.
3. Oscillators: Oscillator will define the type of the wave, frequency and the amplitude range of the wave.
4. Mixers: Mixer takes two different wave forms as input and mixes them input a single output waveform.

4. Operators of the language

The operations permitted are "+" and "*". Both the "+" and "*" operators are overloaded.

Addition: "+" operator can be used to,

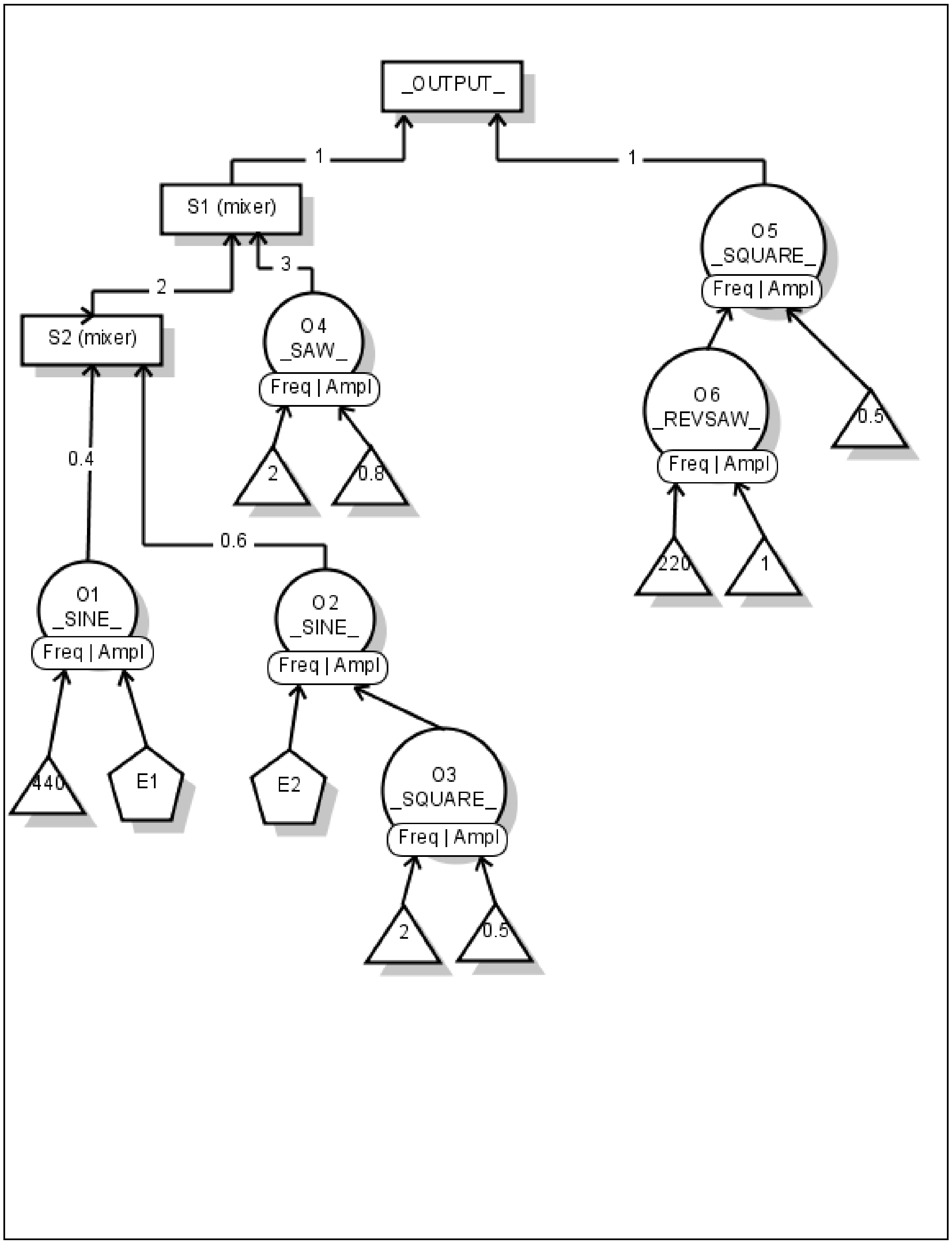
1. Add two constants values
2. Add a mixer to an oscillator (result is input to another mixer)
3. Add two oscillators. (result is input to another mixer)
4. Cannot add two envelopes.

Multiplication: "*" operator can be used to

1. Multiply a constant value and an oscillator. (result is a term in a mixer assignment)
2. Multiply a constant value with a mixer. (result is a term in a mixer assignment)

5.1 Keywords

mixer	start	end	header
func	connect	env	osc
def	_TONELENGTH_	_SINE_	_SQUARE_
SAW	_REVSAW_	_OUTPUT_	



Legend:
 Triangles => Constant Values
 Circles => Signal Generators
 Pentagon => Envelopes
 Rectangle => Mixer or OUTPUT

<u>6. Example program in your language.</u>

Every program would be broken up into two segments, a header and a main function. More segments can occur if the user chooses to implement functions, which are discussed below.
In the header, the name of the output file and the length in seconds of the output are given. Every function, including main, also contains two segments, named 'def' and 'connect'. In the the 'def'

section, all the elements (oscillators, envelopes, and mixers) are defined. In the 'connect' section, elements are attached to one another. Following the 'connect' section in every function, _OUTPUT_ is assigned.

The language will support the use of functions. Functions should be thought of as modules that encapsulate and generalize entire sections of the synthesizer. For example, should a user want multiple instances of an oscillator controlling the frequency of another oscillator, but wishes to vary the frequency of the controlling oscillator in each instance, such a structure could be defined in a function. The function would take the controlling oscillator's frequency as an argument. The main function will never receive any arguments. Functions will only be called in the 'connect' section; within 'connect', they can be used anywhere an oscillator would be, with their _OUTPUT_ acting as the output of an oscillator.

```
start header
   _OUTPUT_ = "testsound.wav"
   _TONELENGTH_ = 10            //10 second sound
end header

start func main()
   start def            //define all elements first
      mixer S1(2);            //defines a mixer for 2 oscillators to attach to
      mixer S2(2);

      osc o1(_SINE_);      //defines a sine wave oscillator, o1
      osc o2(_SINE_);
      osc o3(_SQUARE_);
      osc o4(_SAW_);
      osc o5(_SQUARE_);
      osc o6(_REVSAW_);

      env e1{ (0.0,0.3) (0.2,0.2) (0.4,0.2) (0.8,0.8) (1.0,0) };
          //defines envelope w/ (time, amp)
      env e2{ (0.0,0.0) (0.3,0.0) (1.0, 1.0) };
   end def                 //end definitions


   start connect            //now connect elements
      o1(40, e1);
      o2(e2(8000), o3);
      o3(2, 0.5);             //control LFO
      o4(2, 0.8);             //inaudible osc
      o5(o6(440), 0.5);
      o6(220, 1.0);             //very high freq LFO

      S1 = 0.4*o1 + 0.6*o2;
      S2 = 2*S1 + 3*o4;
   end connect
   _OUTPUT_ = S1 + o5;
end func main
```