Developing a Versatile Audio Synthesizer
Victor Shepardson
Computer Systems Lab 2009-2010

Abstract

A software audio synthesizer is being implemented in C++, capable of taking musical and non musical input information and using additive and FM methods of synthesis to achieve a variety of timbres, vibrato, tremolo, and smooth pitch change effects.

Background

Electronic sound synthesis has been of interest to musicians, electrical engineers and computer scientists for as long as it has been practical. The goal of this project is to create an easy-to-use piece of software for exploring multiple methods of sound synthesis using digital oscillators.

Additive Synthesis

An audio signal can be represented by a collection of sine waves with different phase, frequency and amplitude called a spectrum (Moore). Additive synthesis creates audio signals by generating and superimposing waveforms. Each wave is cheap to compute in a digital system, but a separate oscillator is needed for every frequency; for sounds with rich spectra, additive synthesis becomes expensive and tedious to set up.

FM Synthesis

Frequency Modulation synthesis uses one signal to modulate the the frequency of another, producing an harmonically complex signal for the cost of just two oscillators. In FM synthesis, the carrier and modulating frequencies are both in the audio band; the result is an output signal which contains the carrier frequency as well as many audible sideband frequencies.

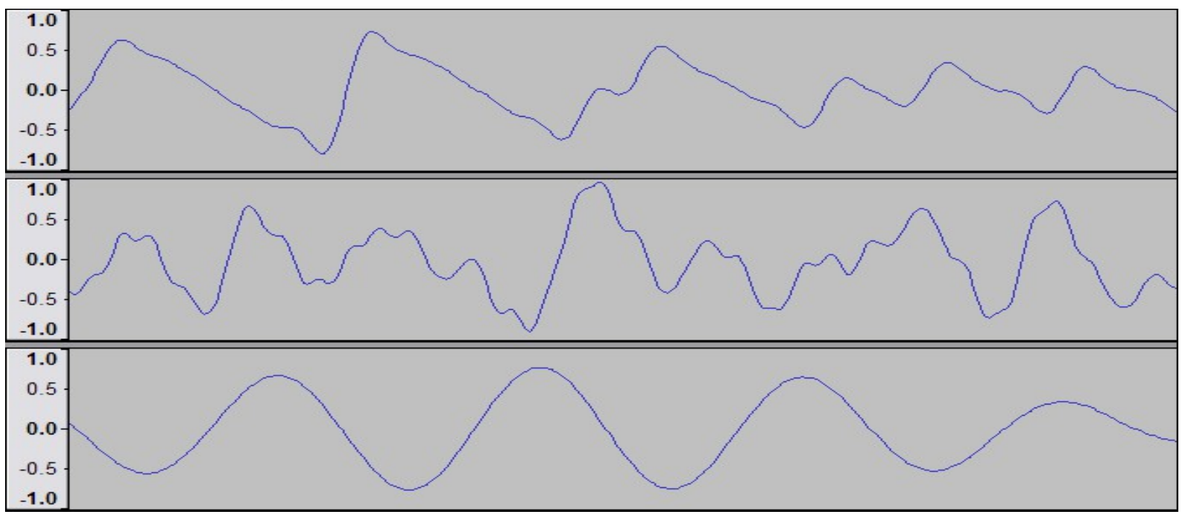
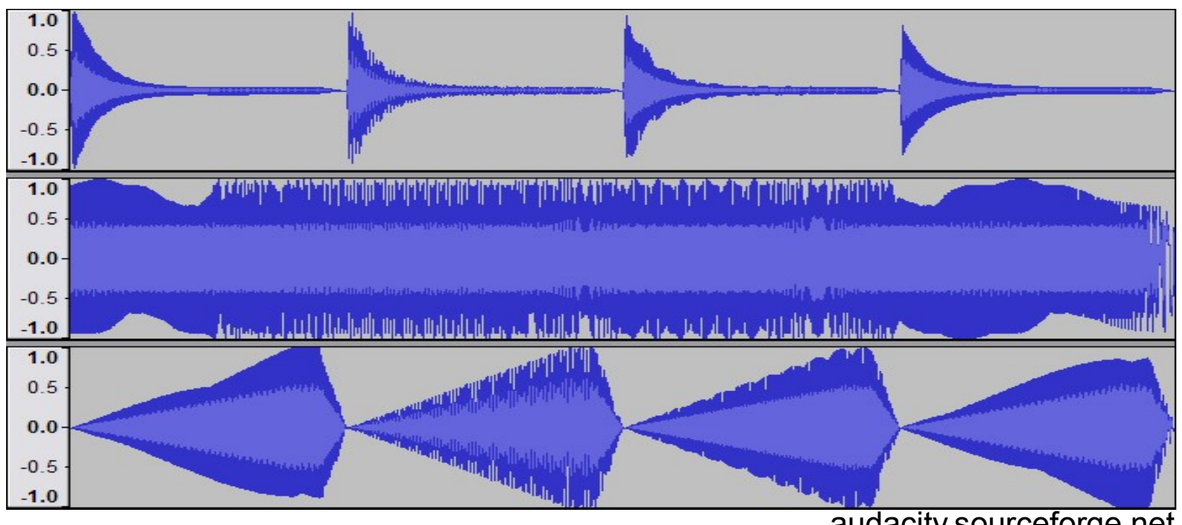


Fig 1: waveforms generated using additive synthesis



audacity.sourceforge.net

Fig 2: signals on a longer time scale

Testing

Testing has been primarily by ear. This has been sufficient to confirm that the correct audio is being produced. Speed testing was difficult to implement in Python versions without impacting performance; testing C++ versus Python versions has not been attempted rigorously, however, the newest version appears significantly faster and produces audio as expected.

Development

Modular Synthesizer

The current version was built from the ground up in C++. The basic method of synthesis is the same, but trades a relatively small amount of memory use for a drastic speed increase by storing discrete waveforms and reading from them during synthesis. Rather than using a collection of functions which must be stitched together in the body of the program, this version uses a collection of synthesizer element objects, instances of which can be created, altered and connected to produce audio.

These objects all inherit from an abstract class Element; Each Element has one output and some number of inputs as well as some number of constant parameters. Elements can be linked together by setting the inputs Elements to the outputs of other Elements. A few basic Elements are: Oscillator, Constant, and Mix_Sum.

Elements can be used to implement additive and FM synthesis; the creation of filter elements could enable subtractive synthesis. Though the sequential nature of signal flow through elements creates artifacts, oversampling can reduce them arbitrarily.

Oscillator
Takes amplitude and frequency inputs and wave parameter, outputs waveform.

Constant
Outputs a stored parameter value.

Mix_Sum
Takes a variable number of input signals, outputs their sum multiplied by a gain parameter.

Partial
Outputs an input value multiplied by a gain parameter.

Pitch
Takes a string containing notation and outputs frequency in Hz.

Envelope
Takes notation string and various shape parameters, outputs amplitude.

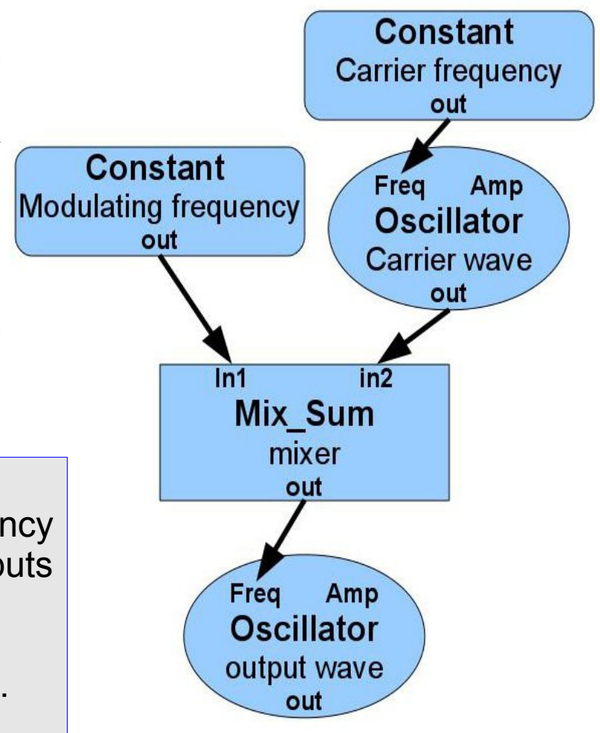


Fig 3: FM implementation

GUI

A graphical interface was created using Gtk and gtkmm. It enables creation and linking of Elements, setting of parameters, running the synthesizer, writing output to WAVE files, and saving and loading synthesizer configurations.

Elements appear in a list of collapsible blocks of text fields and buttons.

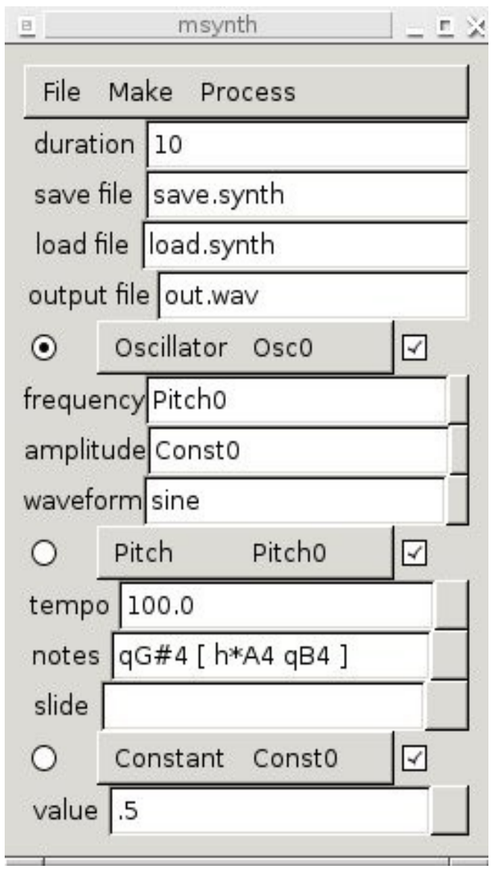


Fig 4: Interface

Results and Conclusions

The synthesizer produces sound as intended, in a fraction of real time. The interface is sufficient to implement FM patches and input simple musical notation. The modular synthesizer could be expanded to implement subtractive, spectral, even physical modeling synthesis by writing new Elements. As a creative tool, it is weak for composition but provides detailed control over small sounds which can be pitch shifted, time stretched and sequenced by other programs.