Project Proposal: Genesynth and Euterpea

Nathan Griffith

February 4, 2010

1 Abstract

The purpose of this project is to explore and manipulate sound in the frequency domain through the use of a genetic algorithm. This algorithm, given a target sound, will produce a model for synthesizing that sound, or as close a representation it can find, and will also allow for the exploration of all generations leading up to that model. The intention is first to reimplement, in Haskell, a technique presented through Michael Chinen and Naotoshi Osaka's *Genesynth*[1], and second to expand upon (or perhaps simply stray away from) this technique.

2 Introduction

Sound synthesis dates as far back as 1876, thought the first commercially available synthesizer was not available until almost a century later. Since then, the number of available synthesizers and synthesis techniques have dramatically grown, and the impact of sound synthesis on popular music has only increased the demand for newer and 'better' synthesizers.

Sound synthesizers generally rely on specific models or approaches to produce a certain range of sounds. This range can be extremely broad, and must often be explored through manually tweaking the settings until a desirable sound is produced. One goal in the application of a GA, therefore, is to automatically generate a model for producing a target sound. This algorithm will make use of additive synthesis, though a GA could certainly be applied to any number of synthesis techniques. Another goal in the use of a GA for sound synthesis is to explore the generations leading up to the final model, simply because they may also hold artistic merit. This can be done by providing not only a final model, but a 'family tree' of the model that may be traced back through the generations.

3 Implementation

Genes, the building blocks of life, are lumped into chromosomes. This project will use the word *chromosome* to describe the data type that will be used to represent sound. The original *Genesynth* project uses a number of data structures to store specific information within the chromosome, but the Haskell reimplementation of this chromosome will be concerned primarily with reimplementing its functionality, and less with its exact organization.

The function of the *Genesynth* chromosome is to store the following:

1. Unit generators, each with an amplitude a, frequency f, and bandwidth b. The unit generator is meant to represent a band of energy with a certain bandwidth, and as such should be thought of as more than a simple sinusoid. A bandwidth of 0 will produce a sine wave of frequency f, and a bandwidth of 1 will produce noise energy from 0 Hz to $2f$ Hz.

2. Definitions for how to change each unit generator over time, with the ability to adjust all three parameters of the unit generators separately. A single unit generator can have more than one of these defined for it, and would then be placed in multiple locations of the output sound.

3. A heiarchical defenition of unit generators, such that each may inherit the time definitions from its parent, if a certain flag is set.

3.1 Mutation and Crossover

Multiple types of mutations may be used. Firstly, single values inside of the Chromosome might be modified. This includes the values that define the unit generators, as well as the time data. A mutation may also move down the heirarchy and either remove children or move them up or down in the heirarchy. Finally, a unit generator may 'divide' into two or more unit generators, each representing a subset of the spectral space occupied by the original, by adding children and modiying itself to accomodate the children.

Crossing-over may also occur between two chromosomes. Crossing over can mainly be seen as an iterative combining of two chromosomes, though certain features are added to increase the productivity of crossing-over. The original *Genesynth* project uses 'allele markers' to track the history of unit generators and time defenitions. If two sets have the same marker, their subcomponents will always crossover. As such, allele markers become an

indicator of similarity. Sets without matching allele markers may crossover with a certain probability. When crossing-over, two individual parameters such as frequency or amplitude will combine to form a number linearally interpolated in-between the original two parameters.

3.2 Fitness Function

The goal of the fitness function is to compare the sound stored in the chromosome to the target sound. This might not be done very efficiently by simply synthesizing the sound from each chromosome and comparing its output to the target sound, as this would become time-intensive. Instead, the *Genesynth* project caches a Fourier transform analysis of the target sound, compresses it into bins, and compares that with an estimate of chromosome's spectrum. A score is computed that rewards 'good' overlap, and penalizs for excessively long trees (thus encouraging efficiency).

3.3 Synthesis

The set of unit generators and their properties are combined simply through summation, though applying properites (bandwidth, frequency, amplitude) to each unit generator for each sample/timestep is more complicated. The original *Genesynth* project defines a method for using random frequency modulations to expand the sinusoid out into an energy band the size of its bandwidth.

4 Possible Areas for Exploration

The *Genesynth* project starts in the first generation with randomly defined noise and uses the GA to build a sound model from that noise. It may also be possible to start from one pre-generated model and attempt to traverse the space between that model and a target sound. One might envision the practical use of the midpoint between models for two distincly different musical instruments. However, issues of getting 'trapped' in local optimum, or any other issues that might arise, might make this approach much more difficult than starting with noise.

Another possible alteration to this project is the use of Mel-frequency cepstral coefficients (MFCCs) in the fitness function. This would result in an added mapping of the Fourier transform of the target sound to line up with the 'mel scale,' a closer approximation of the human ear's response to

to the sound. MFCCs have proven useful in voice recognition and have only recently been applied in measuring musical similarity.[2]

Finally, the musical applications of this project deserve special consideration, so in the end an attempt should be made to demonstrate possible uses for the results.

5 Deliverables

The results of this project will be accompanied by a written report, a public release of the code, and an example music file demonstrating possible uses for the results.

References

[1] Michael Chinen and Naotoshi Osaka, *Noise Band-based Genetic Algorithm Analysis/Synthesis Framework*. Tokyo Denki University, 2007.

[2] Matthew Yee-King, Martin Roth, *Synthbot: An Unsupervised Software Synthesizer Programmer*. International Computer Music Conference, 2008.