Name: Tal Blumstein
ID: 304858137
Guiding Professor: Mira Balaban
Date: March 2019
Ben Gurion University of the Negev

GRADUATION PROJECT

“Analysing and Synthesizing Live & Recorded Audio Input with the ChucK Audio Programming Language”
INTRODUCTION
THE CHUCK AUDIO PROGRAMMING LANGUAGE
ChucK is a programming language for real-time sound analysis, synthesis and music creation based on code. ChucK presents a unique time-based, concurrent programming model that's precise and expressive. In addition, ChucK supports MIDI and other common useful audio controllers with an easy integration.

ChucK offers composers, researchers, and performers a powerful programming tool for building and experimenting with complex audio synthesis/analysis programs, and real-time interactive music with (almost) standard syntax and features that correspond with other common programming languages.
ChucK Unit Generators

Unit Generators are function generators that output signals that can be used as audio or control signals. Unit generators are specially integrated into the virtual machine such that audio is computed for every sample as time is advanced. Via the timing mechanism, we have the ability to assert control over the audio generate process at any point in time and at any desired control rate.
ChucK Unit Analyzers

Unit Analyzers are analysis building blocks, similar in concept to unit generators. They perform analysis functions on audio signals and/or metadata input, and produce metadata analysis results as output. Unit analyzers can be linked together and with unit generators to form analysis/synthesis networks. Like unit generators, several unit analyzers may run concurrently, each dynamically controlled at different rates.
PROJECT’S DESCRIPTION

WHAT IS IT ALL ABOUT?
General Overview

My main goal was to combine two things I am most passionate about into one project: **Music and Coding**.

To achieve this goal, I chose the ChucK programming language that was designed to program and manipulate sound to create and compose music through code while using key components of Computer Science and very basic sound fundamentals. Basically, making music through mathematical basics and the very primitive definitions of what sound actually is - starting with sound waves and resulting in pure music as we know it - all made by the computer.
Project’s Components

- Recording the input
- Pitch Detection (FFT)
- Harmonizer
- Melody Pattern Detection
Recording the Input

- For this project I used 3 different recordings of an electric piano with different melodies.
- Each recording is in the C major scale and consists of 1 or 2 melodies to be detected by the program later on.
Pitch Detector (FFT)

Using FFT is a great way for extracting a pitch estimate out of an audio input signal. When a microphone/any other kind of recording picks up signal, it picks up many different pitches that also include the sound of the room for example, in addition to the main frequency coming into it. The FFT (Fast Fourier transform) allows us to convert this input into a graph with “spikes” that correspond to pitches of the input with the highest velocity.
Therefore, for pitch detection, I used the built-in ChucK FFT Unit Generator that applies the Fourier transform on the live/recorded input. Then, in an endless loop (where time is advanced each iteration with a division of the sample rate of the input) I iterate over each value of the current FFT graph and find the maximum of it. This maximum is therefore the main (loudest) pitch of the input. I’ve also learned through reading, researching and playing with the FFT parameters, that the best results for pitch detection are received with an FFT window size of 2048 and a Hanning window function (which I didn’t find necessary to expand on here).

This extraction doesn’t result in a perfectly clear sound, therefore I am also filtering low-gain frequencies from the original input to clean the sound as much as possible. This kind of filtering is called gate filtering.

The gate filtering in my project is implemented with the help of a ChucK Unit Generator called OnePole:

The ability of ChucK to run processes and functions concurrently, allowed me to run the filtering process at the same time as analysing the input with the FFT. I created another endless loop (in which again, time is advanced each iteration with a division of the sample rate of the input) that uses the filtering object to check the gain of the input at each iteration and determines if it is high enough to be a part of the analysis or not.
Harmonizer (C Major Scale)

A Harmonizer is a musical effect that its purpose can be understood straight from its name: It takes a single note melody input and harmonizes it according to a specific scale or to a specific chord type or given musical intervals. I chose to implement my harmonizer according to the C Major scale since it involved more computations and interesting coding (identifying if the note is in the scale or not and harmonizing it accordingly). This means that every note from the original input that will match one of the 7 notes in the scale, will be harmonized according to its scale degree. For example: If there is a D in the input, the harmonizer will harmonize it with a Dm chord (root position). If the current note is not one of the scale notes, I chose to harmonize it with a chord that is an approach chord to the upper closest scale degree. For example: if the note is D#, it will be harmonized with a B chord which is the dominant chord of the scale degree E.
Implementation:
The harmonizer is also implemented as a concurrent process that runs concurrently with the analysis and the filtering processes. It gets the last analyzed note, converts it to a MIDI frequency value and applies the pitch correction function which performs a simple computation and determines the closest note within the western 12 half-tones musical system of the current processed pitch. After getting the return value of the pitch correction function, the harmonizer applies the right harmony to the corrected pitch according to the C Major scale.
Melody Pattern Detector

This unit is a fun addition to the project. It takes the input and tries to find a specific given melody within it. If it finds it, it will cause the computer to respond in a different way than if it didn’t find the specific melody.

In this project, we can insert 2 arrays of melodies consisting of 3 notes each and the program will know to recognise them and activate the corresponding behaviour - for example: if the melody in the first array is detected within the recording, the program activates a reverb effect on the recording.
CONCLUSION
Although this was a long and very challenging process with many ups and downs, this was definitely one the most interesting projects I’ve worked on during my Computer Science studies at Ben Gurion University. I learned a lot about the fundamentals of music, their relations to math and computers and about algorithms that allow sound to be processed into music as we know it.

And for the very end of this presentation, I want to thank Gil Dori for advising me along the way and making this a very interesting process with a lot of added value.