Final Project Report

for

Sever Tipei's
Music 499B Proseminar
PROJECT-ORIENTED COMPUTER MUSIC

David Tcheng
May 1st, 2014

Goal

The goal of my class project was to develop software for smart phones to listen to ambient sound, visualize it, and classify musical attributes of the sound it in real time to aid musicians in improving their sound.

Feature Extraction

For music analysis, I took the approach of simulating the behavior of the human ear. The human ear has roughly 3,500 "hair cells" which are devoted to detecting acoustic energy at different frequencies. For the computer simulation, each hair cell is represented as virtual "tuning fork" that vibrates when exposed to acoustic energy in its frequency range. To prevent tuning forks from vibrating indefinitely, a damping factor is applied to bleed off the energy of the tuning fork in controlled manner. Mathematically, the original signal is passed through a bank of "band pass filters" which implement the tuning forks. In addition to tuning fork simulation, I calculate the instantaneous energy of each tuning fork by summing kinetic and potential energy after each sound sample is processed (e.g., 44,100 times a second). Changes in the energy of the frequency detectors (tuning forks) form a "spectrogram".

Initially, the tuning fork simulations (spectrogram computations) were coded in java. Running on a modern Android smart phone, only about 128 tuning forks could be simulated in real time. To improve speed, I used a new Android OS feature called "filterscript" (a limited subset of "renderscript") which is a way for developers to access all the multiple processors, GPUs, and DSPs, on the smart phone simultaneously. After spending a lot of time learning how to use "filterscript" on Android, which is similar to doing kernel based programming on GPUs with CUDA, I was able to speed up the computation by a factor of about 30x. Now I am able to simulate all 3,500 "hair cells" using a 16KHz sampling rate.

Visualization

Visualization of sound is achieved by displaying a scrolling spectrogram in real time. Intensity is maped to 11 discrete colors using the "heat" color scale. The delay time between the sound and visualization depends on android device and OS, but based on testing with my devices, the latency was about 1/10th of a second.

Using a Samsung Note Pro 10.1 android device, I was able to display a 2500x1600 spectra in real time. To aid visualization, the spectral energy of each is normalized so that even the softest sounds in the room can be clearly visualized.

Using the real time visualization tool, I was able to "see" sound and recognize things I was unable to perceive before. For example, while at a jazz club I was watching the room sound as I moved around. Towards window I noticed a high frequency sound (that could not be heard due to live music being played and due to its high frequency of around 10Khz). It turned out to be the neon light. In another case I found a very high frequency (e.g. 24Hz) source in my bathroom. It turn out to be the charge of my electric razor.

The real time spectral visualization gives incredible feedback to musicians as they can see intricate spectral elements of their timbre. Inaccuracies in rhythm or pitch are blatantly obvious. I think musicians in training would benefit greatly by the ability to "see" their sounds.

Finally, I think the deaf community could benefit from the real time sound visualization. If started at a young age, I think a totally deaf person could learn how to decode human speech in real time using the visualization. Because microphones are capable of recording frequencies both above and below the threshold of human hearing, deaf people using state of the art microphones could learn to recognize some sounds better than people with perfect hearing.

Real Time Music Analysis

There are two types of music analysis I worked on: (1) fundamental pitch detection for tuning instruments, and (2) chord detection to help some people lacking real time chord detection skills better jam with people and learn new music where only audio recordings are available.

For pitch detection I simulate a large number of tuning forks (e.g., 1024 frequencies ranging from 40Hz to 10KHz) which covers the fundamental range of most instruments. The resulting spectra is analyzed. A simplistic approach would be to find, for every point in time, the frequency with the highest energy and report that as "the pitch". However, since we are looking for the "fundamental pitch" which assumes a harmonic instruments, we are looking not for the frequency with the strongest pitch, but rather a fundamental frequency that explains as many of the harmonic energies as possible. The fundamental pitch need not be sounded to be detected. The algorithm assumes a template for an idealized instrument which has all harmonics but with each successive harmonic weighted to be 0.97 the weight of the previous harmonic. The algorithm attempts to match this harmonic template computed for a wide range of frequencies to the current spectra and uses the best matching frequency as the pitch prediction.

The resulting system did extremely well at detecting fundamental pitch of all musical instruments except for percussive instruments with no obvious harmonic structure. I took my Android smart phone to the local guitar shop to compare it with existing tuners. My algorithm seemed superior, especially when trying to detect the pitch of "distorted" guitars from across the room.

Note Detection

Since the system can accurately determine pitch in Hz, it is easy to determine what "note" on a 12 tone

tempered scale is being played. Simply find the closest note to the measured pitch. The system displays the name of the note is recognizing (e.g., C#) to the user as well has how sharp or flat the note is in cents. In addition, the color of the note indicates tuning. When "green" the note is close to being perfectly in tune. When "yellow", the note is sharp, and when "blue" the note is flat. This proved to be a useful approach to quickly indicating to the performer the state of the current notes tuning.

Chord Detection

For chord detection, I was able to detect major an minor chords accurate and quickly using the smart phone. The algorithm used to detect chords is as follows:

(1) compute real time spectrogram at high resolution (e.g., 1024 bands)
(2) for each "note" (e.g., every note on the piano) compute the energy of the note in the spectrogram
(3) apply templates for each chord, major, minor, 7th, etc. to find best match

For step (2), lower notes are given more weight than higher notes through the use of a weighting factor. When computing the energy of each note, not all frequency bands are used, only ones that are close (+/- 20 cents) to the correct tuning.

The following java code shows how the algorithm distiguishes between major and minor chords.

```
.
. Intialization
.

for (int bandIndex = 0; bandIndex < numBands; bandIndex++) {

        double octaveFraction = (Math.log(freq) - Math.log(13.75)) / Math.log(2.0);

        int octaveIndex = (int) octaveFraction;

        octaveFraction = octaveFraction - octaveIndex;

        double noteFraction = octaveFraction * 12.0;
        int noteIndex = (int) (noteFraction + 0.5) % 12;

        double delta = noteFraction - (int) noteFraction;

        int cents = -1;

        if (delta <= 0.5) {
                cents = (int) (delta * 100);
        } else {
                cents = (int) ((delta - 1.0) * 100);
        }

        int minCents = 20;
        if (Math.abs(cents) < minCents) {
                bandNoteIndices[bandIndex] = noteIndex;
        } else {
                bandNoteIndices[bandIndex] = -1;
        }

        freq *= bandToBandFactor;
}

.
. Each Time Frame
.
```

```
                for (int i = 0; i < numNotes; i++) {
                        noteEnergies[i] = 0.0;
                }

                double weight = 1.0;
                double endWeight = 1.0 / 128.0;
                double weightReductionFactor = Math.exp(Math.log(endWeight) / (numBands - 1));
                for (int i = 0; i < numBands; i++) {
                        if (bandNoteIndices[i] != -1) {
                                noteEnergies[bandNoteIndices[i]] += weight * spectra[i];
                        }
                        weight *= weightReductionFactor;
                }

                double maxMajor3rdChordStrength = Double.NEGATIVE_INFINITY;
                double maxMinor3rdChordStrength = Double.NEGATIVE_INFINITY;
                int maxMajor3rdChordStrengthNoteIndex = -1;
                int maxMinor3rdChordStrengthNoteIndex = -1;
                for (int noteIndex = 0; noteIndex < numNotes; noteIndex++) {

                        double majorChordStrength = 1000 * noteEnergies[(noteIndex + 0) % 12] + 1000 * noteEnergies[(noteIndex + 4) % 12]
+ 1000 * noteEnergies[(noteIndex + 7) % 12];
                        double minorChordStrength = 1000 * noteEnergies[(noteIndex + 0) % 12] + 1000 * noteEnergies[(noteIndex + 3) % 12]
+ 1000 * noteEnergies[(noteIndex + 7) % 12];

                        if (majorChordStrength > maxMajor3rdChordStrength) {
                                maxMajor3rdChordStrength = majorChordStrength;
                                maxMajor3rdChordStrengthNoteIndex = noteIndex;
                        }

                        if (minorChordStrength > maxMinor3rdChordStrength) {
                                maxMinor3rdChordStrength = minorChordStrength;
                                maxMinor3rdChordStrengthNoteIndex = noteIndex;
                        }

                }

        }
```

Summary

I have demonstrated that smart phones are powerful enough for to provide real-time feedback to musicians. Using low level native coding via Renderscript was effective at speeding up computation. Audio visualization at high resolution can sometimes shows patterns that can not be heard. Harmonic template matching works well for pitch detection. Using chord templates and lower note weighting work well for chord detection.