Keoni Burns
CSCI 551

Final Project Proposal

1. Phase Vocoding (Pitch shifting)

Description

Phase vocoding is process used in DSP (digital signal processing) for time-stretching and pitch-shifting. These two terms are almost the same because in order to pitch shift we need to time-stretch.

Time stretching is the act of lowering the frequency of a given sine wave without changing the pitch of the sound, because frequency determines the pitch of the sound. This is done by partitioning the entire wave file into overlapping windows (samples) each overlap should be uniform. After we have our windows we can then use the fourier transform to give us the range of signals or values that each window has. Once this is done we can spread the windows out and use interpolation to fill in our missing values from the given sound samples. after which we can perform the inverse fourier transform on each of our new partitions and add them to our sound sample.

starter code

```
#include <cmath>
#include <complex>
#include <iostream>
#include <vector>

using namespace std;

const double PI = 3.14159265358979323846264338327950288 4;

void phaseVocoder(const vector<vector<complex<double>>>& anls_stft, int n_anls_freqs, int n_
                  int n_synth_frames, int hop_len, int win_len, int n_fft, double scaling, i
    vector<double> freqs(n_anls_freqs);
    iota(freqs.begin(), freqs.end(), 0);

    vector<double> anls_frames(n_anls_frames);
    iota(anls_frames.begin(), anls_frames.end(), 0);

    vector<int> og_idxs(n_synth_frames);
    for (int t = 0; t < n_synth_frames; ++t) {
```

```
        og_idxs[t] = min(static_cast<int>(t / scaling), n_anls_frames - 1);
    }

    vector<vector<double>> mags(channels, vector<double>(n_anls_freqs));
    vector<vector<double>> phases(channels, vector<double>(n_anls_freqs));

    for (int i = 0; i < channels; ++i) {
        for (int j = 0; j < n_anls_freqs; ++j) {
            mags[i][j] = abs(anls_stft[i][j]);
            phases[i][j] = arg(anls_stft[i][j]);
        }
    }

    vector<vector<vector<double>>> phase_diffs(channels,
                                               vector<vector<double>>(n_anls_freqs, vector<d
    for (int i = 0; i < channels; ++i) {
        for (int j = 0; j < n_anls_freqs; ++j) {
            for (int t = 0; t < n_anls_frames; ++t) {
                phase_diffs[i][j][t] = phases[i][j] - ((t > 0) ? phases[i][j] : 0);
                phase_diffs[i][j][t] = fmod(phase_diffs[i][j][t], 2 * PI);
            }
        }
    }

    // TODO: Implement interpolation functions for shifted_mags, shifted_phase_diffs, and u

    vector<vector<vector<double>>> shifted_phases(channels,
                                                  vector<vector<double>>(n_anls_freqs, vecto
    for (int i = 0; i < channels; ++i) {
        for (int j = 0; j < n_anls_freqs; ++j) {
            shifted_phases[i][j][0] = phase_diffs[i][j][0];
            for (int t = 1; t < n_synth_frames; ++t) {
                // TODO: Implement the loop logic
            }
        }
    }

    // TODO: Implement the remaining logic for synthesis and display
}

int main() {
    // TODO: Provide necessary parameters and call phaseVocoder function
    return 0;
}
```

above i have some code that was taken from this github and i used chat gpt to

convert what I could from python to c++.

chatgpt log

the code builds but I still need to work on it before it can be functional

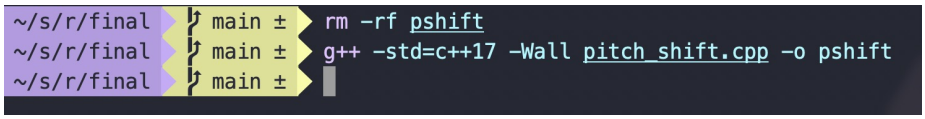

Figure 1: image

verification

I plan on verifying using Python libraries such as librosa to create a test wav file, which will be used when checking the values of the programs wav file. Additionally we can use python libraries like numpy to verify the correct output.

We can check the data using the forumla below

$$\boxed{\text{pitch change} = -12 \log_2 \left(\frac{\text{tempo}_1}{\text{tempo}_2} \right)}$$

here is the basic proof which can also be found on this link

$$\boxed{y = 2^{x/12}}$$

$$\boxed{\frac{y_2}{y_1} = \frac{2(x_1 + k)}{12^2 x_1/12}} \iff \boxed{\frac{y_2}{y_1} = 2^{\frac{k}{12}}} \iff \boxed{K = 12 \log_2 \left(\frac{y_2}{y_1} \right) = -12 \log \left(\frac{y_1}{y_2} \right)}$$

where k is the change in pitch, X is our time value, y is our value from the sample

I have not been able to test the outputs of the sequential program because it is not yet functional

2. Parallelization Method

I plan on using a hybrid using Mpi as a base for the majority of parallelization and sprinkling omp pragmas in where I can.

3. mathematics and numerical methods

- Harmonic analysis
- Interpolation
- numerical accuracy/precision

4. Machine

I will be using ECC-Linux for running this program. I plan on scaling first with 2-8 nodes using 1 cores and then scaling up the number of cores using openMp from 2-8 for each increment of nodes

5. Speed up goal

I hope to achieve atleast 50% speed with 2 or more cores and atleast 2x with 2 cores so hopefully we get about 4x speed up overall

6. interest

I chose this project simply because I wanted to become more familiar with sound programming and I believe that it would be awesome to build a realtime sound processor with cpp from scratch. Additionally i think that its applicable to the course becase the principles of DSP are numerical and the program itself is computationally expensive so there is room for speed up.

7. Biggest Challenge

the whole project is going to be pretty challenging. I don't have any experience with sound file i/o or the methods for manipulating wav file data. Additionally, I haven't done any serious math for about 3 years so I have to relearn transforms and such.