[CTP431]Project 1
Soya Park soya@kaist.ac.kr
October 5, 2015

Code Documentation

Description for each mini-project

#1. Amplitude

This project is mainly about calculating amplitude of the music and display it as the size of circle e.g. bigger sound then bigger circle. All I need to do is calculating amplitude for the given processed data of sound by AnalyzerNode. AnalyzerNode is handy javascript API to provide user processed data either in time or frequency domain, hence, user doesn't need to care about the segmenting and windowing. User can get real-time processed data as following source code line 1-2.

```
1   var dataArray = new Float32Array(analyser.frequencyBinCount);
2   analyser.getFloatTimeDomainData(dataArray);
3
4    . . .
5
6   if (RMS) {
7         for(var i = 0; i < dataArray.length; i++) {
8               dataArray[i] = Math.pow(dataArray[i],2);
9         }
10
11        var mean = dataArray.reduce(function(a, b) { return a + b; }) /
    dataArray.length;
12
13        power = Math.sqrt(mean);
14  }
```

Figure 1. Peak of Amplitude http://physics.tutorvista.com/waves/amplitude-of-a-wave.html

Utilizing data, I compute RMS of the array in order to get RMS amplitude as in line 6 to 14. At last, I manage mainly two envelop phases - attack and decay. When it detects as attack phase, I enlarge the size of circle immediately, whereas decrease smoothly with decay coefficient.

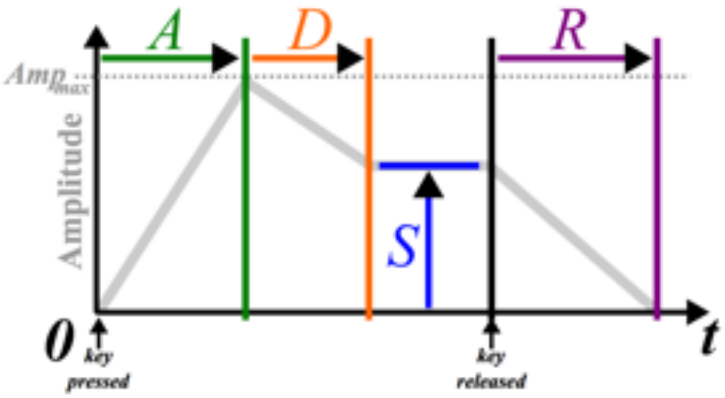

Figure 2. Envelop phase https://github.com/Tonejs/Tone.js/wiki/Envelope

#2. Pitch detection

It detects pitch of sound in means of maximum likelihood which is one of pitch detection method in frequency domain data. I generate comb and sum filter matrix which has a size of (interest frequency)*(pitch candidate range).

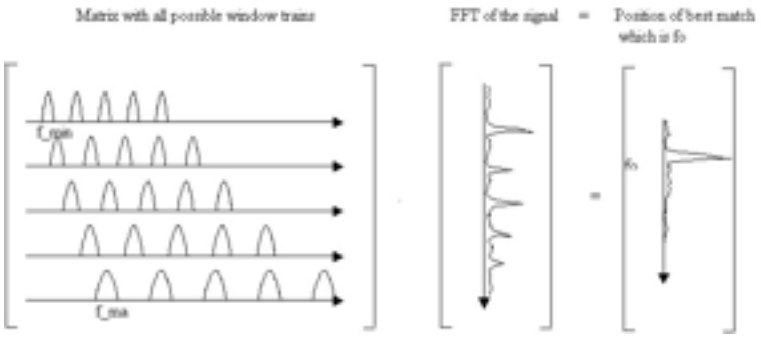

Figure 3. Maximum likelihood https://ccrma.stanford.edu/~pdelac/154/m154paper.htm

```
                interest frequency  frequency domain data

Pitch candidate ( filter ) *  (    ) =  (    ) Pitch gram
```

Figure 4. Comb filtering using frequency domain data

```
// Comb-filtering
for(var p = 0; p < pitch_range.length; p++) {
      pitch_gram[p] = 0;
      energy[p] = 0;

      for(var f = 0; f < dataArray.length; f++) {
            pitch_gram[p] += (comb_filter[f][p] * dataArray[f]);
            energy[p] += (sum_filter[f][p] * dataArray[f]);
      }
}
```

#3. MIDI scale

It requires to find pitch in MIDI scale. Pitch in midi could be converted in MIDI scale as following formula.

```
midi[x] = (a / 32) * (2 ^ ((x - 9) / 12));
```

```
// For each note
for(var v = Math.round(midi2hertz(m-1))+1; v < Math.round(midi2hertz(m + 1)); v++){
      var power = dataArray[Math.round(dataArray.length/context.sampleRate*v)];
      if (max < power) {
            max = power;
            max_index = v;
      }
}
```

#4. Audio Visualization

This visualization is consist of mainly two type - real-time and analysis part. Upper part is corresponding to real-time visualization which provides intuitive view to user with number of circles which is positioned at the amplitude and frequency of the music. And bottom is analysis which is represented as heat-map. Every second, the program capture the tendency of the music -this time pitch - and fill up the heat-map every second.

I utilize the graphic library d3. Additionally, for heat-map, I make good use of http://bl.ocks.org/tjdecke/5558084. My contribution at heat-map is as following.

* I enable to plug-in real-time data into heat-map so that it can fed data dynamically. Whereas the original heat-map is only able to use static data in *.tsv format which has to be built already before being used.

* User could analyze overall atmosphere of music while playing music.

* For this project, I only plug pitch but it could be potentially used for any other factors of music.