Audio Extension Report

# Summary

For my project I chose to implement a visualizer for a song. To do this I used the FFT object in minim to create my own beat detection objects to detect beats at certain frequencies. I used this data to change various visual aspects of my program, such as the height of a 3d plane made up of multiple quads.

# Development

I used the FFT object [1] in the Minim library [2] to extract data about certain frequencies in the song I used. I initially loaded my program into a sketch that displayed all the frequencies in the song and noted the frequencies at which certain sounds in the song were triggered, putting them into variables.

The variables storing these frequencies are used in objects of the Beat class.

Below is the main function in the beat class that checks if a beat has been detected:

void check() {

if ((fft.getFreq(freq)>thresh)&&(timeSinceLast>timeThresh)) { //checks if the amplitude of the frequency to be checked is above the threshold and it has been enough time since the last beat

beatOn = true;

timeOfLast = millis();

} else { //at all time beatOn is false and the time since the last beat is recorded

beatOn = false;

timeSinceLast = millis()-timeOfLast;

}

if (beatOn == true) { //if beatOn is true, sets the inc to the maxInc

inc = maxInc;

}

if (inc>1) { // inc is constantly decreased by speed until it is equal to 1

inc \*=speed;

}else{inc = 1;}

}

This code is run in the draw function. Firstly, the function gets the amplitude of the frequency being detected [3] . If it is above the set threshold, then that part of the if statement is true. It then checks if the time since the last beat at this frequency is above the time threshold to ensure that multiple beats aren’t detected at frequencies that stay above the threshold for long periods of time. If both of these are true, then the beatOn Boolean is set to true, and the time that this beat was detected is recorded. If this if statement evaluates to false, then the beatOn is set to false, and the time since the last beat is recorded. After this is done, a variable called inc that is used to control other elements of the program is set to its maximum value when beatOn is true. Then, the inc value is decreased by a multiplier that is below 1 and above 0, until it reaches a value of 1. This inc value is the main thing that my visualizer uses from each different beat.

//important frequencies of various instruments in song

float beepF = 1162.793;

float clapF = 1874.9219;

float kickF = 129.19922;

float sawF = 2024.1211;

float snareF = 9087.012;

float breathF = 2885.4492;

float saw1F = 516.797;

//initalizing all the objects to detect various instruments in song

beepB = new Beat(beepF, 40, 500);

clapB = new Beat(clapF, 5, 10);

saw1B = new Beat(saw1F, 40, 1000);

kickB = new Beat(kickF, 100, 200);

sawB = new Beat(sawF,10, 100);

The above variables contain the frequencies of instruments in the song to be used to create the following objects:

//creates the grid

for (int i=0; i<dim; i++) {

for (int j=0; j<dim; j++) {

//calculates the next vertices

int iN = (i+1);

int jN = (j+1);

//sets the b component of the stroke to its height relative to the z axis

float c = map(noise(i\*0.02, (j+frameCount)\*0.02), 0, 1, 0, 255);

//sets the r and b values of the stroke to 255 if a synth is heard

stroke(saw1B.inc, saw1B.inc, c);

//increases the stroke width if a different synth is heard

strokeWeight(map(sawB.inc,1,255,1,10));

noFill();

beginShape();

//creates a grid with y values corresponding to perlin noise. the y axis of the noise is constantly increased

vertex(i\*s, j\*s, (h)\*(noise(i\*0.02, (j+frameCount)\*0.02)-0.5));

vertex(iN\*s, j\*s, (h)\*(noise(iN\*0.02, (j+frameCount)\*0.02)-0.5));

vertex(iN\*s, jN\*s, (h)\*(noise(iN\*0.02, (jN+frameCount)\*0.02)-0.5));

vertex(i\*s, jN\*s, (h)\*(noise(i\*0.02, (jN+frameCount)\*0.02)-0.5));

endShape();

}

}

Each of these beat objects contain the the previous frequencies as well as individual amplitude and time thresholds that seem to work best for the individual instruments.

The above code snippet is of the function that creates the 3d grid in my program. This is rotated scaled and translated beforehand to be in the view before it is drawn. The number of quads as well as the size of each quad is taken from variables that are pre determined. the z values of the points on the grid are created using the noise function [4] using the I and j values multiplied by 0.02 to create softer variations in height. The stroke colour is based on the noise as well as the inc value from a beat object used to detect a certain synth sound, and the stroke weight is mapped between the values of 1 and 10 from the inc value of a beat object used to detect another synth. The h variable that is multiplied by the noise generated is as follows:

h =100+(255 -kickB.inc)\*10;

background(0, beepB.inc/2, clapB.inc/2); //the g component of the color is set to 255 if a beep is detected and the b is set to 255 if a clap is detected

This value is calculated by taking the inc value of the kick detector away from 255(the maximum inc value), multiplying this by 10, and adding it to 100. This means that when a beat is triggered, the noise height of the grid is reduced to 100 before gradually raising, creating an effective visual metaphor for the kick drum.

Finally, the background colours of the program are set to half the inc values of a beat object that detects a percussive sound, and one that detects the clap used as different components of the colour.

# Evaluation

Overall, I found the implementation of the beat objects fairly straightforward, as the FFT object allows you to take the amplitudes of certain frequencies in a sound very easily. This made the process of creating beat objects a matter of trial an error to see which values worked best. However, getting accurate values for frequencies of instruments proved difficult as sometimes their frequencies would change. Also, other instruments that are at similar frequencies tend to trigger beats for other instruments which proved to be an issue that was difficult to solve, and subsequently events are occasionally triggered when they are not supposed to. This visualizer may have been more suited to a more minimal song with instruments and sounds at very particular frequencies to avoid this problem. To extend this program, I would add more visual elements such as some sort of basic particle system to visualise more elements of the song, as currently, the grid is the main visual aspect of the program, and the visuals can become repetitive because of this.

# References

[1] FFT class <http://code.compartmental.net/minim/fft_class_fft.html>

[2] Minim library documentation <http://code.compartmental.net/minim/>

[3] getFreq function <http://code.compartmental.net/minim/fft_method_getfreq.html>

[4] noise function <https://processing.org/reference/noise_.html>