**CSC 545/645 Computer Speech, Music and Images**

**Exercise No. 11b, Week 14, due April 25, 2021**

**Visualizing the Structure of Music**

**Goal**

Display a self-similarity matrix of music

**Background**

Music information retrieval (MIR) is a relatively new area of research, having started with query by humming systems in the 1990’s and developing into systems, such as Shazam, that can identify music played over the radio. As you are undoubtedly aware, most music has a structure defined by repeating elements. A similarity matrix, first used in biology to show repeating structures in genetic sequences, can show the repetitive structure of music.

**Procedure**

We will use Minim (<http://code.compartmental.net/minim/>), a third-party audio library, for this exercise. Minim has many more features than the Processing Sound library, but the main reason for using it here is that it has the ability to stream audio from disk – the entire sound file does not have to reside in memory. Furthermore, when the entire file *is* read into memory, it loads the file much faster than the Processing Sound library. You’ll have to add the Minim library through the sketch/Add

The initial program, downloaded from Blackboard, performs a sequence of short-time Fourier transforms over a song. The resulting spectra, averaged into three frequency bands per octave, are stored in a two dimensional array *(spectra[][]).* The first dimension is the spectrum for a given time period; the second dimension is the 30 frequency bands for that spectrum.

A *similarity matrix* shows the difference of two sequences at each point of comparison. In this case, the matrix is more properly called a *self-similarity matrix* because the song is being compared against itself. The song is split into analysis frames for spectral analysis—the matrix should show the difference between each frame and every other frame. The (0, 0) pixel, for example, will show the similarity between the first frame and itself. Of course, a frame is exactly equal to itself, so it will show maximum similarity.

Your task is to write the function that displays the frame similarities. When these matrices are displayed for audio, they usually display the difference (or distance) between frames—a difference of 0 will show up as a black pixel, while pixels that are very different will show up as bright. Because each frame is equal to itself, the diagonal will be black. Pixel (1, 0) will show the difference between the first frame and the second frame; pixel (2, 0) will show the difference between the first frame and the third, etc, so the top row shows the difference between the first frame and every other. Dsistance measure are usually symmetric (we’ll use the absolute value of the difference between frames), so the matrix is symmetric about the diagonal. In other words, for each pixel, (x, y) is equal to (y, x); map the distance to the range 0 to 255—you may want to print the max distance to find out what range you’re scaling from).

Remember that each frame is represented by a spectrum. Because of the octave averaging, there are 30 frequency bands in each spectrum, so your distance measure for frames i and j will be:

dist = 0

for (k = 0; k < number of frequency bands; k++)

dist += abs(frame[i][k] – frame[j][k]);

You will have nested loops that compare each frame with all others. The image will probably be too big to fit on the screen—so create the matrix in a PImage then resize it to the size of the window (the resize is done for you in setup()). To maximize the window size, and keep it square, setup() sets the size to fullScreen() then resizes it to (height, height). If you run out of memory, you can increase the number of samples in the FFT (you may also have to increase the buffer size) and/or increase the memory size in the File/Preferences menu.