COS426 Final Project: Sandstorm.mp3

Savannah Du (savannah), Jason Kim (jasonmk), Michael Zhang (mzz)

May 16, 2017

1 Introduction

Sandstorm.mp3 is an audio visualization application that creates a "sand art bottle" based on realtime audio playback. As lovers of music, we originally set out with the simple goal of incorporating music into our final project. We flirted with the idea of creating a sort of interactive virtual instrument where "beads" would fall on top of strings to generate sound, but we quickly decided that the user-controlled nature of such an application may result in difficulty in attaining "musical"-sounding patterns, and we were not particularly interested in merely triggering sample playback on collision events.

Instead, we decided that some form of audio visualization would allow us to explore music more deeply and be a natural fit for the specifications of the assignment. Audio visualizations have traditionally been a staple part of many popular music players (e.g. iTunes Visualizer, MilkDrop for Winamp) and serve to augment the overall experience of the listener by supplying a complementary visual media representing the music as it plays. In addition, visualization of music can be useful as a tool to show changes in frequency, which can aid in music production processes such as mixing and mastering. In fact, spectrograms, which are used to represent the available spectrum of frequencies in any given sample, are used to analyze audio files (e.g. comparing lossy vs. lossless compression), generate audio effects (e.g. phase vocoder), and help the study of speech and speech production.

For our project, we hoped to create a fun and interesting visualization that could change dynamically to accompany audio playback while also representing the audio in a way that might yield additional insight into the composition of the audio itself. In particular, we hoped to create something that may allow us to compare visually how music differs from genre to genre or explore different frequency distributions to help inform optimal equalization calibration for headphones or music players.

We searched online for previous work with JavaScript audio visualizations, finding some examples of animations using Three.js to resize cubes or generate random curves based on trigonomic functions to dynamically respond to sound. We also found many examples of simple moving bars or dots visualizations that changed with regard to the frequency spectrum of audio. However, we found these examples to be somewhat unoriginal and wanted to be a bit more creative with how we could represent audio. After brainstorming, we initially had the idea of using animation with a 3D particle system, similar to what we implemented in Assignment 4, in order to create a water/paint falling on "beads" animation that relied on frequency-filtered beat detection to determine the rate and velocity of particles descending from their source and colliding with sphere objects that we initialized in the scene. Upon colliding, we envisioned that the droplets would color the spheres temporarily, while also dispersing and pushing the spheres downward, leaving us with a spheres are different relative heights to indicate what frequency bands were strongest throughout the audio sample at the end of the audio sample.

Though we made a preliminary prototype of the scene (all of this can be found in our "old" folder), including adding new texture attributes and mappings to make more realistic water impacts (Figure 1), we had difficulty finding an accurate way to perform beat detection on frequency ranges other the lowest one, which is significantly simpler due to the beat-providing presence of the kick drum in modern music. We tried a number of beat detection algorithms, which included

Figure 1: Our first project idea.

calculating peaks above a certain threshold in audio that was low/band/high pass-filtered to get peaks for individual ranges, as well as another one that monitored whether the waveform went above a certain amplitude threshold in order to register a beat, with the threshold lowering in between beats to be sensitive to any shifting dynamics within the audio. Ultimately, we unsatisfied with the results because they did not provide a great variety of frequency bins and were quite poor at detecting beats outside of the bass frequency ranges. In addition, having to manage beat detection in realtime became a logistical issue, since it entailed controlled precise playback audio with either realtime beat detection (which performed poorly) or precalculated beat "tracks" playing alongside the audio that would require a high level of sychrony that we were not confident we could implement.

Using what we learned from our attempts to implement our previous idea, we came up with an idea that could produce similar results without relying on strict beat detection. Trying a different approach, we decided to create a visualization of an empty bottle being filled up with different colors of sand, randomly sampling from the realtime frequency spectrum in order to determine the color of the sand falling in. By treating this animation as a particle system, we could initialize sand particles that fall into the jar, colliding with the jar container and other sand particles to gradually fill up the jar. At the end, we would hopefully have a colorful piece of sand bottle art that also serves as a visual representation of how the frequencies change throughout the song. Since this approach relies heavily on frequency data, we expected that this would work especially well with music featuring a broad dynamic range and strong beats that would allow for differentiation betweeen different bins of frequencies, as this could cause clear striations in the sand, essentially encoding various markers into the art in realtime.

2 Methodology

In order to execute our approach, we had to implement a new particle system, as well as some way to feed in realtime audio data as it plays from the speakers. We opted to go with a 2D particle system instead of the 3D ones we implemented for the previous assignments because we wanted more artistic control in having the sand segment more clearly to be able to present a clearer and more informative audio visualization. Because of this, we ended up programming a new particle system engine from scratch.

The particle system share many similarities with the engine used in Assignment 4. A buffer stores all of the particles arranged by time created. At each time step (as dictated by however long the window takes to return an animation frame) particles are created, updated, and re-drawn accordingly. All of the initial values are defined as constants at the top of sand.js. Particles are never killed because we want to have a finished piece of sand art at the completion of the song. We include forces of gravity, drag, and normal force for the box and other particles.

An interesting design challenge was the fact that calculating collsions between all sand particles often made particles vibrate back and forth. Normal collisions are calculated when particles overlap. They are simply given a small "bump" in the right direction. However, this ultimately led to situations in which all particles settled on the floor, at impossible distances, but stuck and

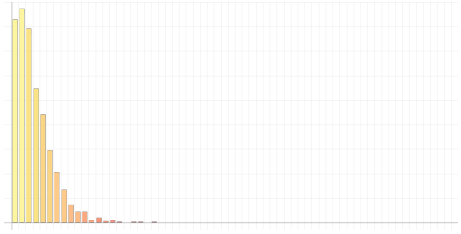

Figure 2: Chopin

unable to move. We fixed this problem by implementing a heuristic of the overall movement of the particles. At each time step, the distance travelled is added to this heuristic and the heuristic decays at a certain rate over time. At the point when this heuristic for overall "true movement" falls below a threshold, all movement stops so that the particles stop vibrating.

For the audio processing, we relied almost entirely on the Web Audio API, which is integrated into most modern browsers to enable audio manipulation. This API essentially treats audio as a stream of data that passes through a series of nodes, starting at a source (the file itself) and ending up at a "destination" (the output, often the available speakers for playback). The nodes provided by this API do much of the heavy lifting: for our implementation, we used their analyser and processor nodes in order to analyze the realtime stream of audio data and process it with JavaScript code. The API allows us to specify the size for the fast Fourier transform of the audio, which we chose to be 128 to yield 64 frequency bins. We then continuously collect the frequency data into an array and use this to select a weighted random bin, with selection probability corresponding to the amount of data in each bin. To perform this, we summed the values of the array to get a total value, then chose a random number within the range of 0 and the total sum and iterated through the array again, keeping a cumulative sum to compare and see if we have passed through the bin containing the random value. After attaining the random value, we use the bin index to select its corresponding color, which we have mapped to a pastel color wheel starting at yellow, with lower frequencies at yellow and higher frequencies at green. The random value determines the color of the sand particle falling at the current time point.

3 Results

In addition to coloring the sand, the random bin selection results were tracked in another array counting how many times each bin was selected. This allowed us to see a graphical representation of the prevalence of frequency bins at the conclusion of the song. While testing, we used R and Microsoft Excel to keep track of how our code was performing in accordance to changes we were making and make comparisons before we had a fully functioning sand visualization. This was mostly useful for confirming that our random weighted selection of frequency bins was working properly.

Since our visualization is meant to be a representation of the audio in realtime, we can observe the end product of our sand bottle to see if the distribution of colors makes sense, based on our knowledge that frequency is mapped to a color spectrum from yellow to blue. Though it may be difficult for us to accurately represent a song's frequency distribution in our heads after merely listening, we can compare music from different genres to see how the color distributions vary. For example, we might expect that a bass-heavy rap track should have a lot more yellows than a solo violin piece. However, between most genres, it was difficult to see a clear visual difference. Overall, we observed an general bias among modern music towards the lower end of the frequency spectrum. In the end, one of the best indicators for music we uncovered was the overall frequency range of the sample. For the Chopin, which was a piano solo, almost the entire piece dwells at the lower end of the spectrum (Figure 3), which makes sense since the piece we chose in particularly does not hit any particularly high notes. Meanwhile, we used some noise music (e.g. "data.convex") to test, getting some interesting skews of frequency toward other bins (Figure 4). Otherwise, the final bottles can be used to get a sense of how the frequency dynamics of an audio sample change over time (our demo PowerPoint contains more examples).

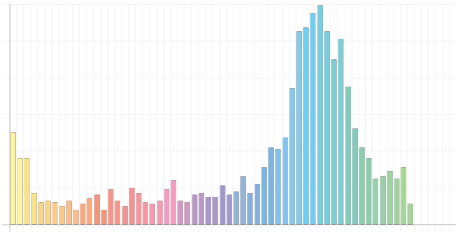

Figure 3: Ryoji Ikeda - "data.convex"

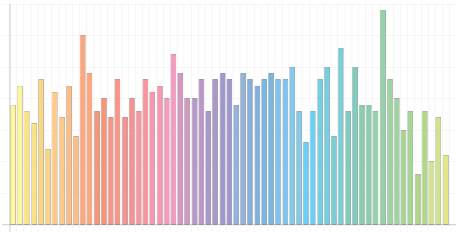

Figure 4: White noise

Delving deeper into signal processing, we can also use white noise (we include a preloaded 60-second sample generated from Audacity) to test whether our sampling and coloring is working properly. Since white noise represents a random signal with equal intensities across all frequencies, we should expect a level histogram and equal distribution of colors in our bottle, which we attained (Figure 4).

We also used some other comparisons as tests, such as comparing Pink Floyd's "Any Colour You Like" in CD and vinyl issues to see if we could visually glean any differences (Figure 5), but we could not discern any. Because we used mp3 files for most of our testing (with 128-256 bitrate, potentially even transcoded), it is very possible that our audio data is not rich enough to capture the fine distinctions between formats such as CD and vinyl. Frequency data is one of the primary aspects of the data that is lost during audio compression, so this is perhaps to be expected. In addition, digital representations and playback of audio differ greatly from their analog counterparts.

4 Discussion

Conceptually, we believe that our approach is a promising method for attaining an aesthetically and artistically interesting and informative visualization of audio that both represents the audio in realtime and as an overall, cumulative distribution of frequency. The inherent randomness incorporated into the project creates pieces of art that are unique (Figure 6).

For our visualization, we had to balance our desire for creating a realistic way for the sand particles to interact while still visually representing different frequency distributions throughout an audio sample in a clear manner. Though we succeeded in achieving most of our goals with this

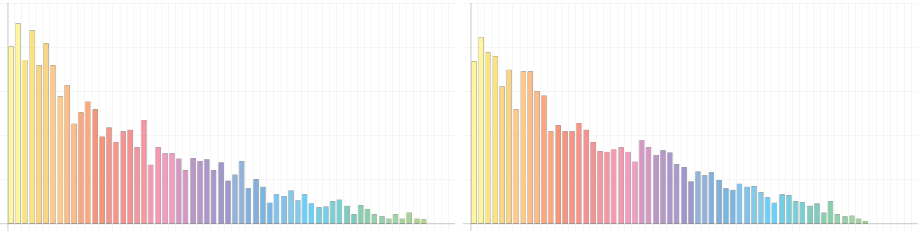

Figure 5: Left: CD. Right: Vinyl.

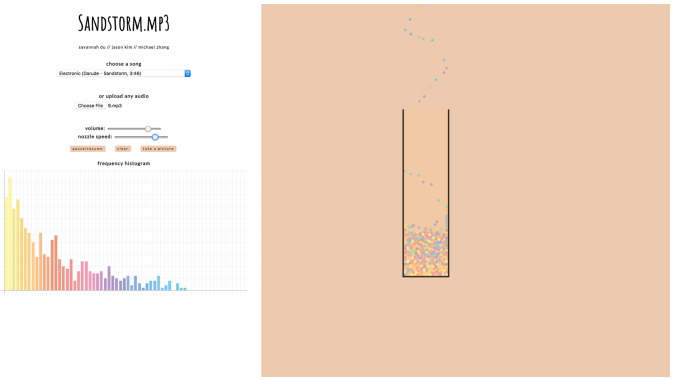

Figure 6: Final project

implementation, we found it very difficult to create nice and clean segmentations of constrasting sections within an audio sample. In some of our examples (e.g. "Party in the U.S.A."), we can see minor changes as it plays through different sections, but these sections tend to be neither long nor different enough to produce very noticeable visual differences. To help remedy this issue, we added an user option for controlling the "nozzle speed," which allows the user to release sand particles from a moving point source instead of a stationary one, with the rate of side-to-side movement controlled by the user. This way, the sand can fall in different ways instead of simply forming a mound, causing segments to fall in a caret (^) shape, as it does when the point source is fixed at the center.

An implementation of our application in 3D would be a natural extension of our project. This would add another dimension (no pun intended) to our visualization though it would require a different aesthetic on our page. Furthermore, computationally intensive extensions could involve treating the sand particles as polygonal meshes so interactions would be based off of geometry rather than estimation with randomness. Otherwise, there are certain more UI functionalities we would have loved to implement with more time, such as a better track playback seeker and file browser, in addition to having more user interaction (e.g. letting the user decide where the sand falls from).

5 Conclusion

Though our project was partially inspired by the particle systems from Assignment 4, we learned a great deal from having to implement our own particle system engine from scratch. In addition, manipulating the sand particles to behave the way we wanted them to was difficult, since we needed to figure out a way for the sand to distribute cleanly to the sides and settle down into a fixed position. Our solution to implement a heuristic to settle particles that should not be moving answered one of the most difficult visual design questions we encountered with this type of particle engine. Aside from the animation, we also learned how to manipulate and harness audio data, which none of us had prior experience with. Thus, we had to learn about some of the finer details of how audio is represented digitally, as well as how to use the Web Audio API.

After doing this project, we can certainly apply our newly-gained knowledge of audio processing and particle systems to create a greater variety of audio visualizations, as well as games and other audio analysis tools.

Our final project can be run locally with a simple Python, or you can find it hosted here.