Authors: Diego Zamalloa-Chion, Aaron Skepasts
Professor: Felix Heide
Advisor: Joanna Kuo

VSKeys: 3D Piano Harmony Visualizer

Abstract

This project features a 3D playable piano and multiple ways for visualizing harmony using said keyboard. Using a codebase from Borja Morales, we turned a playable keyboard into a tool for giving colorful and geometric intuition for harmony to music students as well as a fun way to animate music for non-musicians. We will implement the Tonnetz grid in a unique three dimensional way and invented our own spirographic visualization of harmony as concatenated spinning vectors.

1. Introduction

The goal of this project was to find a way to visualize harmony as pitch relationships. There are many popular videos and games which visualize pitch, typically as streams of notes which either emanate from the piano (when playing freely) or arrive at the piano (when playing a rhythm game such as Guitar Hero). While pitch is interesting, harmony in music isn't simply about the collection of simultaneous pitches, rather it is the emergent relationships of consonance or dissonance between pitches which create the emotions that make music so powerful. Many musicians and mathematicians report synesthetic experience whereby notes or numbers evoke different colors. In this project we tried to leverage color and geometry to give an aesthetic and musically enlightening sheen to the musical experience of playing or listening to the piano. This project would benefit intermediate music students by showing the geometric relationships between chords, since most musicians don't receive a sufficiently graphical or intuitive music theory education. This project would also benefit those without a musical background who would like to play around with colorful displays of harmony for the aesthetic value.

We drew on previous work in both the computer graphics and musical domains. The musical inspiration for the first visualization we made came from the Tonnetz[1], which is an isometric grid layout for the twelve notes of the Western chromatic scale devised by Leonhard Euler. The Tonnetz representation works fairly well in two dimensions, but it inadequately captures the cyclical nature of the scale, which is why we plan to make a cylindrical version of the grid to both make the visual experience immersive and demonstrate the circular nature of the scale and harmony itself. Inspiration for the spirograph came from a visualization of the Fourier transform by 3Blue1Brown.[2] The idea of spinning concatenated vectors is tightly mathematically linked with deconstructing auditory signals in the first place. Given the limited time of the semester, we decided to focus more on the visualization aspects of the project and so borrowed much of the boiler plate code for rendering a playable piano from Borja Morales. This ended up almost being more trouble than it was worth because the initial code was written with an outdated version of three.js so we spent a significant amount of time refactoring it to be compatible with the most recent version of the library including replacing lots of helper functions.

---

[1] https://en.wikipedia.org/wiki/Tonnetz

[2] Sanderson, Grant. "Pure Fourier Series Animation Montage." YouTube, YouTube, 2 July 2019, www.youtube.com/watch?v=-qgreAUpPwM.

2. Methodology

2.1 Keyboard Layout

We used Three.js for the graphics portion of the project and we used Tone.js and MIDI.js for the musical playback aspects of the project. The first modification we made to Borja's source code was rearranging the keyboard layout. All of the online options for playing the piano with a computer keyboard are extremely limited in range. Many layouts don't even reach two octaves, which is very frustrating when trying to play large chords spanning multiple octaves. Additionally, most online keyboards begin at C. A common misconception is that the tonic of the scale must be the lowest note and since many pieces are written in C, then C must be the lowest note. On the contrary, many real-life pianos have A as the lowest note. By starting our layout with F on the left Shift key and following the pattern of black and white keys on the home and bottom rows of the keyboard, we reach note B on the slash key and then can cleanly transition to the Tab key for the next C to continue the keyboard. This leads to a 43 note keyboard, which is nearly half of a grand piano and in our view the biggest practical range playable on a computer keyboard. One complication was disabling the tab key on our site, since its default behavior is to toggle keyboard focus across all of the elements on the screen.

One other quite serious problem with this ambitiously large keyboard layout is that conventional computer keyboard hardware isn't designed for large numbers of keys to be pressed at the same time.[3] This makes playing large chords very difficult to play because frequently certain keys won't respond if you have too many keys pressed in their vicinity. This effect is called keyboard ghosting. If you have an external keyboard, you can test a workaround to these hardware limitations by playing on both your laptop keyboard and the external one so neither keyboard is over budget in the number of pressed keys. While we tried our best to make the piano keyboard work on the computer keyboard, this issue was out of our control and constitutes an unfortunate limit to creating piano keyboard simulators online with standard keyboard hardware which can only be circumvented by fancy non-ghosting gaming keyboards.

2.2 Key Coloring

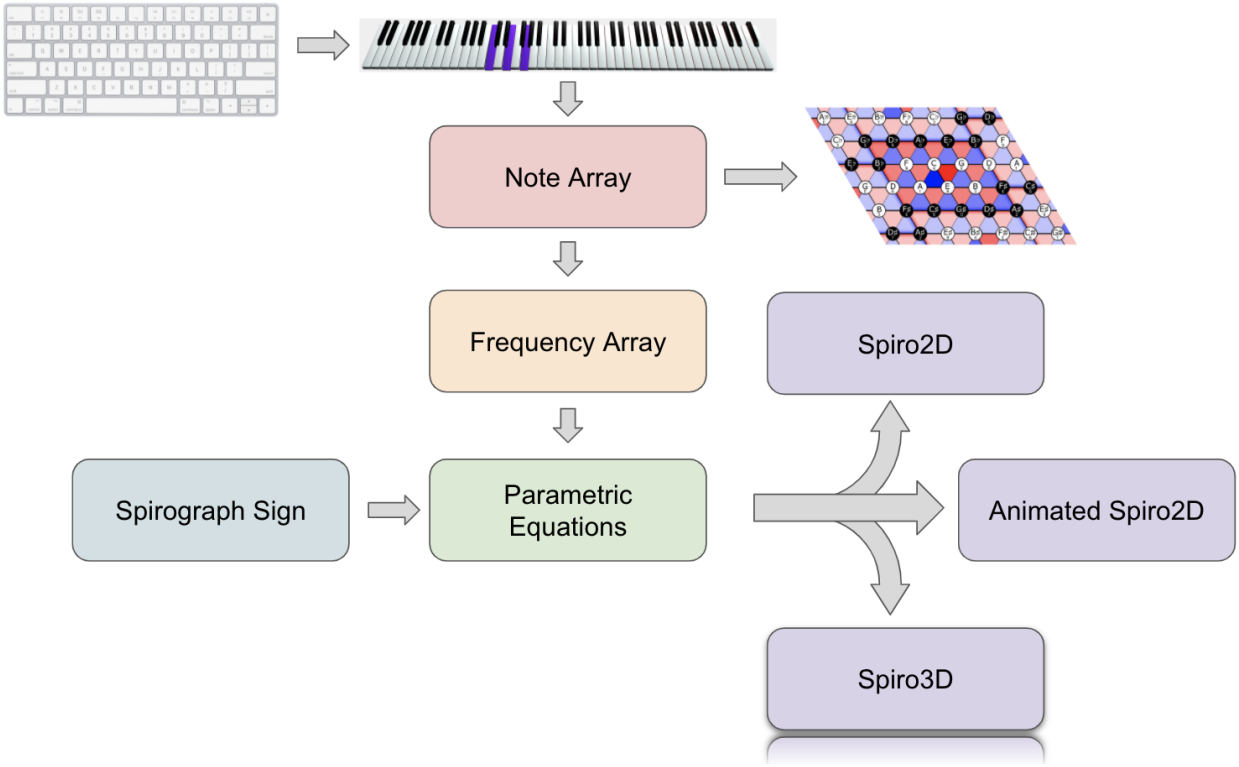
The next feature to implement was changing key colors. We wanted to set a different color for each of the twelve tones of the chromatic scale so our first pass was to use equally spaced colors in the HSL color space. However, the greens were much too similar since this color space isn't perceptually based so we had to create our own custom palette of colors that were as distinct as possible while remaining in a hue-wise gradient with uniform saturation and lightness where possible. One ordering of colors was simply to go around the color wheel by semitones going up the keyboard. Another interesting ordering derived from the music theory of the "Circle of Fifths", which Jacob Collier views as "increasing brightness", was to go around the color wheel every 7 semitones with a modulus operation so that adjacent hues are always spaced by 7 semitones, getting brighter every ascending fifth.

---

[3] "6. Three Simultaneous Key Presses and Ghosting." Three Simultaneous Key Presses and Ghosting, www.dribin.org/dave/keyboard/html/ghosting.html.

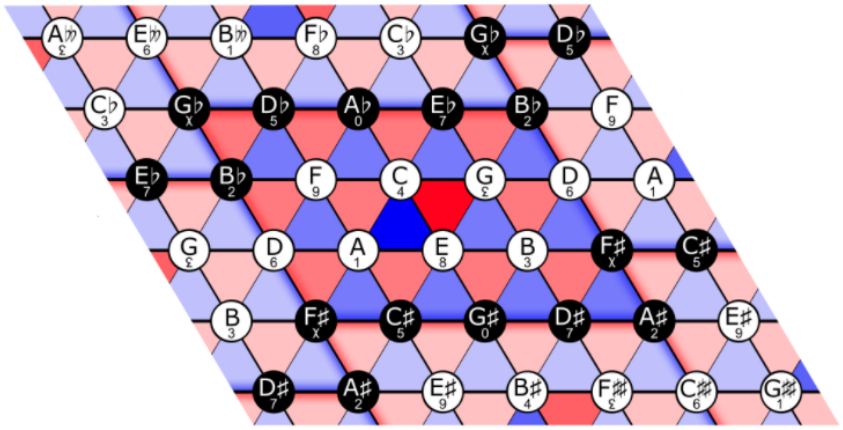
2.3 Architecture

The figure below shows the data flow in the application. At every time step, the application takes as input the set of keys pressed on the computer keyboard. It then translates these according to the layout described above into piano key presses. These piano keys can be thought of as a boolean array (indexing from low to high notes) where each key is either being pressed or not (dynamics aren't possible with a computer keyboard as the controller). The indices of the keys that are being pressed, we'll call the "note array". Using a mod 12 operation to identify pitch classes, this array goes to the Tonnetz to light up and animate the corresponding spheres. This note array is also converted into frequencies for use in the spirograph visualizations.



2.4 Tonnetz Grid

We implemented Tonnetz as a grid of spheres with a colorful scaling animation for pressed keys according to the scheme outlined above. The Tonnetz arranges the 12 tones of the Western scale into an isometric grid, so we labelled each sphere with the note that triggers its animation. In order to make the animations as realistic as possible, for midi playback, we simulated piano key presses according to the required note and based all of our visualizations on events triggered by pressing keys on the piano. We also used the same smoothing function as the keypress animation to ensure that the sphere scaling didn't appear abrupt. We added lines between adjacent spheres in the grid to show the shapes that result from different chords connecting notes and their corresponding spheres.



2.5 Spirograph

To understand the translation from notes to frequencies, we have to briefly discuss tuning paradigms. Since sound perception is logarithmic (2x frequency sounds 1 octave higher) and the 12 tone Western scale divides the octave into 12 equal parts, the mathematically exact way to divide the octave is using the function $2^{x/12}$ where x is the number of semitones above the initial frequency. This leads to many irrational numbers whose harmony sounds more or less in tune in all 12 keys, but is slightly out of tune by the same amount in all of them. This is called equal temperament and pianos must be tuned this way because they are expected to be playable in all 12 keys. Just intonation is a tuning system based on the harmonic series, which leads to more consonant, harmonically pleasing, and simple rational ratios between frequencies. Bells are tuned in this way because they are only ever going to play as a set in one key. For example, the perfect fifth interval in equal temperament gives a ratio of 1:1.49830707688, while just intonation yields the interval 1:1.5. Clearly just intonation will yield more visually pleasing results because of its nice ratios, however a just intonation piano can't play in all keys. Therefore, we decided to make playback equal temperament but translate the note relationships into cleaner just intonation ratios for the purpose of visualization in a way that was root invariant. By root invariant, we mean that we subtracted the minimum of the note array for each of its elements so that the lowest note would always be 0. We then translated the root invariant array into the frequency array according to the harmonic series. This is slightly deceiving in that the frequencies being heard during playback aren't exactly those input to the visualizations, rather they are rounded to nicer ratios but we view this as taking the best of both the equal temperament and just intonation worlds for a pleasing audiovisual experience.

Using this frequency array, we implemented the spirograph visualizations. While the Fourier transform decomposes a signal into its corresponding frequencies and sinusoidal functions, where the signal is air pressure over time in the case of music, we wanted to increase the number of spatial dimensions in the visualization. Drawing inspiration from 3Blue1Brown, we switched from air pressure being a function of time to position on a 2D plane being a function of time and from up/down sinusoidal oscillations in one pressure dimension to circular motions in two dimensions.

For the instantaneous 2D spirograph visualization, we used the frequency array to create a parametric equation for the chord which we then plotted for $t = [0, 96\pi]$. The parametric equation was constructed as follows from the frequency array F of length L.

$$x(t) = \sum_{i=0}^{L} \frac{1}{F[i]}(sin(F[i] * t)),\ y(t) = \sum_{i=0}^{L} \frac{1}{F[i]}(cos(F[i] * t))$$

In other words, the parametric equation for each chord simulates a chain of concatenated vectors. Each vector corresponds to a note in the chord and its speed of rotation is proportional to its frequency and its length is inversely proportional to its frequency. Note that unlike a standard 2D Fourier transform, the frequency is doubly encoded in the vector (in length and speed) so there can't be slow turning short vectors and there can't be fast turning long vectors. Additionally note, that the above equations describe vectors that all turn in the same clockwise direction. This synchrony results in most of the shapes being quite similar and cardio-like without sharp points. One of our more innovative ideas was to allow the user to choose different directions of rotation. One good alternative to all clockwise rotation which we found was an alternating pattern of signs. This meant reversing the direction of rotation for every other vector (having sorted the vectors by frequency). We also built a function to manually input rotation directions as (+) and (-) signs since given our format, every chord has 2^{L-1} unique shapes for an L length chord.

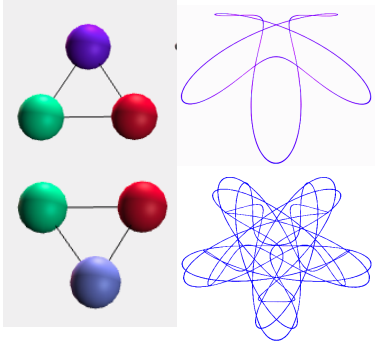
The 2D animated spirograph shows the behind the scenes computation of the parametric function explicitly and in real time, allowing users to see how concatenated spinning vectors can create such complex drawings. Of course, in reality all of the vectors live in two dimensions, but we stretched them into the third dimension to illustrate the drawing process in a more realistic way that simulates how a drawing automaton might create these drawings.

The 3D spirograph uses the same equations as the 2D spirograph, but with the addition of $z(t) = t$. Since t has finite resolution, we created a particle for each timestep at the position described by the parametric equations. By gradually shifting the phase of the graph very slightly $t = [0+\partial, 96\pi+\partial]$ created a pleasing animation effect where as the chord is held, the particles appear to move along the 3D curve.

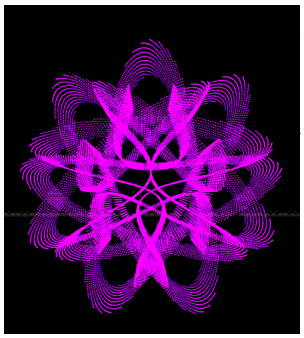
We created a piano and synth using a sampler and AMSynth respectively using Tone.js and we kept Borja Morales' MIDI playback code and added a few of our favorite songs to his repertoire. We also made a brief Shepherd tour to guide a new user through the GUI and controls.

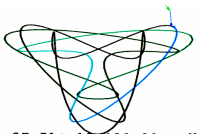
3. Results

Even though our spirograph was constructed from very simple equations derived from the frequencies of the notes in the chord, the emergent patterns we were able to find were surprisingly complex. We experimented with various animations for the Tonnetz Grid, since we didn't want our implementation to too closely resemble the 2D version. We tried to make the spheres jump up in the positive-y direction, but this looked too messy because the offset in vertical position looked like it broke away from the grid despite the grid being the grounding element in the visualization. We were successful in implementing the majority of our big picture ideas and in innovating harmony visualization using mathematical techniques.



Major chord above
Minor chord below
On the Tonnetz
& 2D spirograph





^ (Shift + M + ,) 2D
< (Shift + N + ,) 3D

4. Discussion

The approach we took broke ground in the domain of harmony visualization. This is promising both in the direction of improving musical education by providing colorful and geometric intuition to music students who don't like reading verbose textbooks AND in the direction of attracting more non-musicians and visual learners to take interest in the mathematical beauty of music. Most music visualizers are either pitch based or algorithmic (e.g. screensavers) in a way that doesn't provide any musical insight or depth, so we believe that the realm of mathematical music visualizations can be much further explored. A variant of this approach might be to visualize music with a physics simulation where notes are objects which interact with each other. While creating this project, we learned lots about javascript libraries for music creation and playback. We also learned how to use the Three.js library in much more depth building on top of the practice from course assignments. We also improve many programming skills: modularity, memory management, algorithmic efficiency, and effective prototyping before committing to a design.

5. Conclusion

Overall, we are satisfied by the result of our project. Our goal was to use the ThreeJs graphics library to analyze music harmony and we successfully did that. I think our approach was quite effective. The Tonnetz grid shows the basic relationships of harmony and the Spirograph illustrates deeper relationships about harmony that are interesting to musicians, while also providing nice visuals for everyone to enjoy. All that remains for this assignment is to reformat the Tonnetz Grid into a cylindrical shape and to add some of the finer details which would make this site a truly immersive musical experience. We are two musicians who study computer science and this was the first project of many combining these passions of ours. It was exciting to create a unique visualization of harmony that could be used to help others better understand music and we both plan to keep working on this project. Our general next steps include providing new visualizations for harmony as well as making the playing experience more immersive and enjoyable for casual playing, like being able to connect a real 88-key MIDI keyboard to this simulation.

6. Contributions

Diego implemented the first version of the Tonnetz, the keyboard layout labels, and the 2D instant and animated spirographs. Aaron implemented the 3D spirograph as well as all of its animations. We both contributed to the GUI and Tone.js for the virtual instruments. Joanna Kuo was very helpful as an advisor in pointing out ways in which we could make our project more vivid, immersive, and graphically interesting.

7. Works Cited

Works Cited

Tone.js Docs, tonejs.github.io/docs/14.7.77/.

"6. Three Simultaneous Key Presses and Ghosting." *Three Simultaneous Key Presses and*

Ghosting, www.dribin.org/dave/keyboard/html/ghosting.html.

“Home - Documentation.” *Guide Your Users through a Tour of Your App*, shepherdjs.dev/docs.

“Home The 100% JavaScript MIDI Player Using W3C Web Audio.” *MIDIjs*, www.midijs.net/.

“Pure Fourier Series Animation Montage.” *YouTube*, YouTube, 2 July 2019, www.youtube.com/watch?v=-qgreAUpPwM.

“Three.js – JavaScript 3D Library.” *Three.js – JavaScript 3D Library*, threejs.org/.

“Tonnetz.” *Wikipedia*, Wikimedia Foundation, 20 Feb. 2021, en.wikipedia.org/wiki/Tonnetz.

reality3d. “reality3d/3d-Piano-Player.” *GitHub*, github.com/reality3d/3d-piano-player/tree/master/js.