Digital Image Sonification

Sarah Lucioni

GenEd 1080

December 11, 2019

Table of Contents

Abstract

This project aims to sonify famous paintings such as The Starry Night (Appendix, fig. 1), Mona Lisa (Appendix, fig. 2), The Creation of Adam (Appendix, fig. 3), and other visual forms. Digital image representations of these paintings will be used which will extend this projects ability to sonify any image such as a photograph. The image is the compositional map. We will try to uncover the hidden composition in the map. To create sound from the image map, a Python program using the Pyo audio processing library will be produced which will convert the RGB values of the image into sound. First, we will present current research and background into this topic. Then, we will examine influences that motivate this project. Next, we will dive into the implementation, analysis, and demonstration of the sonification of art. Finally, we will conclude with reflections.

Motivation and Description

Art of any form is exhilarating to experience. People of all creeds, cultures, and ability can experience music because it is a form which one can feel, hear, and see. On the other hand, paintings and visual art are restrictive because they require the ability of sight. Most paintings (especially famous paintings) are furthermore not allowed to be touched. Compositional maps are a visual form of music which reflect the composition in a myriad of ways. Some maps, such as musical scores, have a main purpose of helping others recreate the intended music. Others convey the feeling of the composition. We see that music is commonly transformed into a visual form which inspires this project to sonify paintings. Now, the painting is the compositional map and the music awaits to be created.

To create this physical experience with paintings, I will write a Python script that will turn a digital image into a song by converting the RGB values of the image into audio frequencies. Through the lens of this compositional map, each pixel is a note. A pixel will be broken into its three color channels: Red, Green, and Blue (RGB). Then, each channel will correspond to a pitch. Therefore, three pitches will play for one pixel. To mimic the sense that dark colors are deep and scary, darker values (which are RGB values closer to 0) will correspond to a lower frequency and thus a deeper and lower tone. Similarly, higher RGB values (closer to

255) will convert to higher frequencies. Sonifying each pixel would take a long time, so I will compensate by taking the averages of small chunks of pixels which should maintain the general sound of the digital image. The set of constructed notes will then be strung together using a sine wave because this is the purest sounding wave. The constructed songs will be analyzed to uncover trends apparent in the visual art form.

Background Research

To research the sonification of digital images, I first investigated other projects regarding this topic. Then, I explored the field of image processing. Next, I searched for an open source audio processing library to use in Python. After completing this background research, I had a clearer outline of goals for my project and felt comfortable proceeding forward with the implementation.

Pixelsynth

Olivia Jack created Pixelsynth which converts grayscale images into music (Jack, "Pixelsynth"). She chooses to emit a frequency when a brighter spot in the image is detected. Furthermore, she reads the image in vertical lines. This means that she only has to do one left to right pass over the image to read and convert the entire image to sound. Reading such an abundance of pixels at once emits a cacophonous sound. She chooses to pitch the pixels based on relative location in the image. A white spot emits a pitch. If the white spot is toward the top of the image, the frequency is higher. If the white spot is toward the bottom of the image, the frequency is lower.

Her user interface is quite engaging and works smoothly. She provides three overall options for the user to experiment with changing: the image, the sound, and the brush for drawing. Let us look first at the image options. The user has a choice of 13 black and white images. The selected images are noticeably blacker with high contrast between the black and white spots. This makes sense because Jack's software turns the white pixels into sound. Having less white in the image will create a relatively more interesting and less cacophonous sound. The user can also upload an image. The image will be converted to black and white. Due to the

explained high black content constraint, user uploaded images that are mainly black create more enticing sounds. Jack provides options to invert the image, alter the brightness and contrast of the image, change the rotation as well as repetition, spacing, and offset of the image. Experimenting with all of these options alters the sound and allows the user to gain an understanding of how her program functions.

The next feature that the user has the ability to change is the sound emitted. First, the user can turn the sound on and off. There is also an option to change the speed at which the image is read which changes the speed of the sound. This speed option also allows the user to change the direction in which the image is read (left to right or right to left). Next, the user has a choice of 10 scales: major, natural minor, harmonic minor, melodic minor, chromatic, whole tone, major pentatonic, minor pentatonic, kumoi pentatonic, and Chinese pentatonic.

A major scale is one of the most commonly used scales in Western music ("Major Scale"). It is a diatonic scale made up of seven notes in the step pattern: whole, whole, half, whole, whole, whole, half. A natural minor scale is built starting from the sixth degree of its relative major scale ("Minor Scale"). The step pattern for a natural minor is: whole, half, whole, whole, half, whole, whole. A harmonic minor scale is the same as a natural minor except the seventh note is raised by one semitone ("Minor Scale"). This changes the step pattern to: whole, half, whole, whole, half, augmented second, half. A melodic minor scale avoids the awkward augmented second step in the harmonic minor scale ("Minor Scale"). This feat is achieved by either raising the sixth note by a semitone or lowering the seventh note by a semitone. A chromatic scale is constructed of 12 semitones ("Chromatic Scale"). This scale covers all of the possible pitches in an equal temperament setting. Next, we have a whole tone scale which is constructed by taking a whole step between each subsequent note ("Whole Tone Scale"). A major pentatonic scale and a minor pentatonic scale both have five notes per octave ("Pentatonic Scale"). A major pentatonic scale takes out the fourth and seventh degree notes from a major scale and a minor pentatonic scale removes the second and sixth degree notes from a minor scale. The kumoi pentatonic scale is a Japanese scale which also removes the fourth and seventh degree, but is most likely derived from a melodic minor (Piper, "Kumoi Pentatonic"). Finally,

the Chinese pentatonic scale is similar to a major pentatonic scale, but the tuning is slightly different ("Chinese Musicology").

To further customize the scale on Pixelsynth, Jack provides options to choose the start note, start octave, and number of notes.

The final feature that the user may experiment with is customizing the brush used for drawing. A user can "draw" on the screen with this brush, and the software will play sounds reacting to the drawing. The basic options the user can change are the brush stroke width, number of repetitions, and opacity. This is a fun feature that encourages user participation.

Image Processing

Following researching Pixelsynth, I looked into image processing as a possible strategy to convert a digital image to sound. Image processing is a huge field in machine learning today. Currently, the main uses of image processing are classification, pattern recognition, and feature extraction (Geitgey, "Machine Learning is Fun!"). However, a neural network could be trained to classify images to different songs. This task is similar to how Spotify classifies music. One flaw I see with using image processing for art sonification is that interpretability may be lost because neural networks struggle with explaining their output. Due to this limitation, I decided to move forward with simple image processing using the PIL Python library by reading the RGB channels of each pixel and then transforming these values by hand.

Pyo

After researching the sonification of art and image processing, I set out to find an open source Python library for audio processing. I stumbled upon and liked Pyo because the creators heavily document this module. Pyo is a Python module for exploring audio processing (Belangeo, "Pyo"). Function descriptions and examples are provided in the API documentation which help build audio software, compose algorithmic music, and further explore audio processing ("Pyo 1.0.1 Documentation"). Being able to understand the functions relies on an understanding of synthesizers, electrical circuits, wave forms, sampling, and other topics covered in GenEd 1080. The key features to producing sound using Pyo is to first boot up the server.

Then, some sequence of frequencies (pitches) is needed. This sequence can be transformed using many Pyo functions. Once a waveform is created, the sound may be output. These steps are the basic steps I will take in transforming paintings into music.

Influence and Exploration

Multiple classes from GenEd 1080 influenced this project. The class regarding compositional maps and compositional choices greatly influenced me to sonify famous paintings. I have seen musical scores my whole life, but I never considered that other compositional maps existed. This class showed me that musical maps are not always intended to recreate the piece. A map may instead express the feeling of a piece. This inspired me to create music from other artistic pieces where the artistic piece took the place of a map. Creating maps for our acoustic and electronic instrument build allowed me to explore and enjoy the other path (music to map). This project lets me explore the map to music path.

The dark to light fluctuations in the paintings influenced me. I want to be able to maintain these patterns by conveying low to high pitch fluctuations. This further inspired me to keep the notes and pitches simplistic so that one can hear the ebb and flow of the painting.

Another speaker that influenced me was Susan Rogers, the last guest speaker of the class. She mentioned how painting moved from realistic to abstract once the camera was invented and then drew a parallel to music becoming abstract now that analog has moved to digital. My project combines these two statements by discovering the abstract music captured within a painting.

To explore my project, I used a handful of engineering principles. First, I learned how to work with an image processing library in Python (PIL) as well as an audio processing library discussed above (Pyo). These both pulled on knowledge of audio software, electronic circuits and synthesizers which we explored in the GenEd 1080 lab. This also required knowledge of Python and Jupyter Notebook. The GenEd 1080 toolkit contains useful resources for both of these tools. In addition, this project required composition and mapping techniques. This project

also required knowledge of the time, frequency duality to construct a song from the sequence of notes assembled from an image.

Implementation

I created a Python script which converts a digital image into a song. This song plays the music confined within a painting.

Steps

The script (Appendix, fig. 4) first reads in an image and separates the RGB channels into three separate arrays using the Python PIL library. The arrays are then reshaped to match the height and width of the image. To reduce the duration of the final composition, the next step in the program scans the image in small chunks of pixels and reduces each chunk to one averaged RGB value. This is known as *filtering* in a convolutional neural network. This effectively reduces the image size which provides the overall sense of the image in a shorter time span.

We now have three arrays of the averaged pixel RGB values for the inputted image. This is where the Pyo audio processing module steps in. RGB values range from 0 to 255. To transform this to an interesting frequency, we must rescale the RGB values. First, I mapped each of the three channels to the same frequency range of 100 to 800. However, this did not provide complete distinction between colors because a color value of (x, y, z) would produce the same pitch as a color value (z, y, x) even though the colors are different. To fix this issue, I mapped each of the three channels to slightly different frequency ranges. I chose the frequency ranges loosely corresponding to the spectrum of light. The red channel ranges from 200 to 550. The green channel range from 212 to 650. The blue channel ranges from 212 to 700.

The next step creates an iterative sequence of the pitches defined from the rescaled RGB values. The final step uses a sine wave signal generator to transform the sequence of pitches into a flowing song. All three channels are played at the same time which creates the song representing the image.

Challenges

Two main challenges that I faced while creating the above script were the duration of the final composition and the frequency ranges of the RGB values. At first, I wanted to convert each pixel value to one note. However, this method created songs that were way too long (over multiple days long). To shorten the time, I decided to read the image in chunks (still left to right). The image chunks are small enough to convey the same meaning, but do not take nearly as long. The output songs are still fairly long (30 minutes or longer), but this is a huge improvement over songs that last a few days. The compositions attached are short excerpts from the full song. The excerpts still convey the general feeling of the inputted painting.

The frequency range problem meant that different colors mapped to the same sound. To combat this problem, I chose slightly different frequency ranges for each of the three channels.

Analysis and Demonstration

Using the RGB values to transform an image into a range of frequencies worked well with a few tweaks as discussed above. Some of the Pyo functions did not work as expected. They muddied the sound which removed the recognition of dark to light patterns. This makes sense because the Pyo functions added more noise and modulated the already defined pitches. The modulation changed the overall sound. If I did not set the intention of the project to be able to slightly distinguish the pitches as they relate to the color, this modulation would have been okay. In terms of the overall composition, I liked the simplistic version best because it conveyed the general flow of an image.

For example, I started out by converting a simple rainbow of colors to sound (Appendix, fig. 5). This composition is in the "`rainbow.mp3`" file. Listening to this composition, one can hear how the rainbow of colors flows. At times, the rhythm feels like it's falling behind itself which speaks to how the rainbow changes from color to color, but it doesn't necessarily do so uniformly. The produced song sounds similar to the human "wa wa wa" tone that can be simulated with a synthesizer and different waveforms. It also sounds almost like a robot trying to exaggerate vibrato which ends up sounding like an alarm.

I enjoy The Starry Night composition. This is stored in "`starry-night.mp3.`" This composition reflects the brush strokes and overall tone of the original painting. The song sounds like a windy and stormy night to me. Looking at the painting, I have the same sense of a windy and stormy night. I find it interesting how there is a consistent pulse throughout the song. At some points, this is not the main focus, but one can still hear it in the background. To me, this relates to the stars of a starry night. They are always there, but one does not always notice them.

The Mona Lisa composition ("`mona-lisa.mp3`") is interesting because there are points where it is very quiet and then all of a sudden it becomes loud. When it is quiet, there is a sense of calmness with a bit of pressure in the background. When it becomes loud, there is a sense of urgency and pressure. This reflects the painting because The Mona Lisa has such a calm look on her face, yet, her constant eye contact raises a feeling of pressure and discomfort.

Inspired by the black and white images that Olivia Jack used to create Pixelsynth, I tested a few grayscale images. I used one image that was mostly white and one image that was mostly black. The mostly white one is of a black and white galaxy breaking out from a white sheet of paper (Appendix, fig. 6). The composition ("`bw-galaxy.mp3`") sounds similar to a heartbeat. The deep, low pitches convey where the galaxy breaks out. The higher pitches within the dark parts are the stars shining through. I think the mostly black image sounds even more appealing. The mostly black image is a black and white shot of a lioness (Appendix, fig. 7). The composition ("`bw-animal.mp3`") sounds like the purring and roars that would come from the pictured cat. This prompts a new way to choose a color scheme or lack thereof: to convey a sound from the image.

Reflections

Creating this project, I learned how to work with two Python libraries: Pyo and PIL. Using the audio processing software, Pyo, I discovered how to link and convert a sequence of numbers into frequencies and then into pitches. Furthermore, I learned how to transform the pitches into a sound wave. Using PIL, I learned how to process image in Python. I also learned how to work with Jupyter Notebooks.

I applied a handful of topics from the GenEd 1080 course throughout this project. First, I applied the ideas of composition and mapping techniques to understand and analyze this project. Using Pyo required me to utilize the topics we discussed regarding electronic circuits, synthesizers, and audio software. Many of the terms and functions available to use on Pyo were features we learned about in class (such as a high and low pass filter). Within Pyo, I also used knowledge of the time, frequency duality to choose the correct pitches and time for the sound I wanted to produce using a sine wave. Furthermore, I had to apply a knowledge of Python and Jupyter Notebook to construct this project.

I wish we dove deeper into how to use an audio processing software. I found it difficult to find a good, open source audio processing software. Once I did find Pyo, it was still confusing to get a program up and running. I wish we had explored some type of related software more so that I would feel more comfortable creating and exploring digital audio processing.

Overall, I enjoyed researching, creating, and sharing this project. I successfully sonified digital images and extracted intriguing results from the compositions. I find the sonification of art exciting. This project just begins to scratch the surface of the multitude of possibilities and songs that can come from other art forms.

Works Cited

[1] Belangeo. “Pyo.” *GitHub*, 29 Nov. 2019, https://github.com/belangeo/pyo.

[2] “Chinese Musicology.” *Wikipedia*, Wikimedia Foundation, 30 Sept. 2019, en.wikipedia.org/wiki/Chinese_musicology.

[3] “Chromatic Scale.” *Wikipedia*, Wikimedia Foundation, 5 Nov. 2019, en.wikipedia.org/wiki/Chromatic_scale.

[4] Da Vinci, Leonardo. “Mona Lisa.” *WikiArt*, www.wikiart.org/en/leonardo-da-vinci/mona-lisa.

[5] Geitgey, Adam. “Machine Learning Is Fun! Part 3: Deep Learning and Convolutional Neural Networks.” *Medium*, Medium, 7 Nov. 2018, medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutiona-neural-networks-f40359318721.

[5] Jack, Olivia. “Pixelsynth.” *Pixelsynth*, ojack.github.io/PIXELSYNTH/.

[6] “Major Scale.” *Wikipedia*, Wikimedia Foundation, 21 Sept. 2019, en.wikipedia.org/wiki/Major_scale.

[7] Michelangelo. “The Creation of Adam.” *WikiArt*, www.wikiart.org/en/michelangelo/sistine-chapel-ceiling-creation-of-adam-1510.

[8] “Minor Scale.” *Wikipedia*, Wikimedia Foundation, 18 Nov. 2019, en.wikipedia.org/wiki/Minor_scale.

[9] “Pentatonic Scale.” *Wikipedia*, Wikimedia Foundation, 28 Nov. 2019, en.wikipedia.org/wiki/Pentatonic_scale.

[10] Piper, Justin. “Kumoi Pentatonic.” 2016, www.justinpiper.com.

[11] “Pyo 1.0.1 Documentation.” *API Documentation*,

ajaxsoundstudio.com/pyodoc/api/classes/index.html.

[12] “Rainbow Image.” *Stack Overflow*, stackoverflow.com/questions/58036809/how-to-extract-color-using-opencv?noredirect&lq=1.

[13] Romero, Leonardo. “Leona” *Leonardoromerovg.wixsite.com*, leonardoromerovg.wixsite.com/mecbrush/3?lightbox=dataItem-jq9ko34a.

[14] Salah, Muhammed. “Black and White Galaxy.” *Instagram*, www.instagram.com/p/BfOq-kpFoJA/?taken-by=muhammedsalah_.

[15] Van Gogh, Vincent. “The Starry Night.” *Google Arts and Cultures*, artsandculture.google.com/asset/the-starry-night/bgEuwDxel93-Pg?hl=en.

[16] “Whole Tone Scale.” *Wikipedia*, Wikimedia Foundation, 17 Oct. 2019, en.wikipedia.org/wiki/Whole_tone_scale.

Appendix

Fig 1. The Starry Night (Van Gogh)

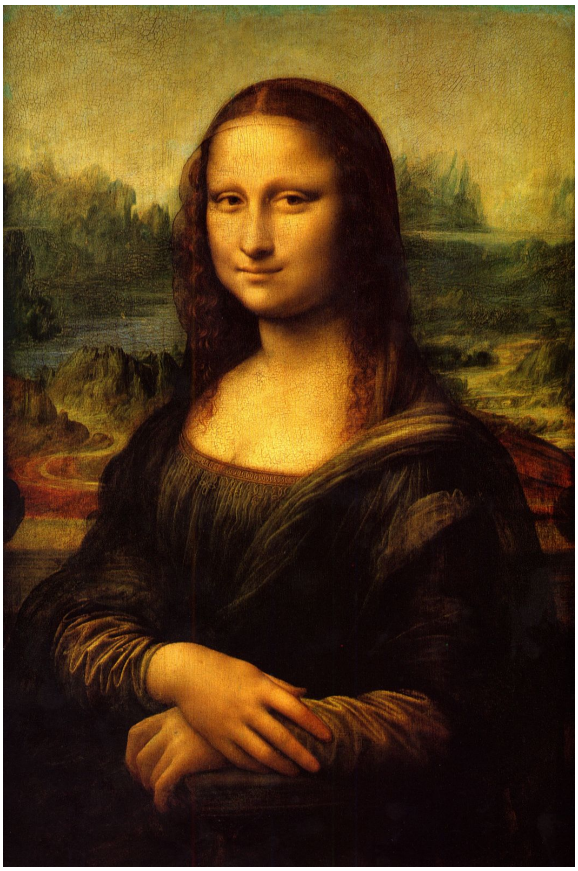

Fig. 2 - The Mona Lisa (Da Vinci)

Fig. 3 - The Creation of Adam (Michelangelo)

```
# Extract the rgb values from an image
img = Image.open("images/bw-animal.jpg", 'r')
pix_val = list(img.getdata())
reds = []
blues = []
greens = []
for rgb in pix_val:
    reds.append(rgb[0])
    blues.append(rgb[1])
    greens.append(rgb[2])

# Image height and width
width, height = img.size
width_step = width // 60
height_step = height // 60
# Reshape RGB to arrays of the image size
reds_arr = np.asarray(reds).reshape(height, width)
blues_arr = np.asarray(blues).reshape(height, width)
greens_arr = np.asarray(greens).reshape(height, width)
reds_avg = []
blues_avg = []
greens_avg = []
# Average small chunks of the image instead of reading each individual pixel
for h in range(0, height, height_step):
    for w in range(0, width, width_step):
        red_slice = reds_arr[h:h + height_step, w:w + width_step]
        reds_avg.append(np.mean(red_slice))

        blue_slice = blues_arr[h:h + height_step, w:w + width_step]
        blues_avg.append(np.mean(red_slice))

        green_slice = greens_arr[h:h + height_step, w:w + width_step]
        greens_avg.append(np.mean(red_slice))
```

```
s = Server().boot()

# Uncomment for version which does not play unique pitches
# pits_r = rescale(reds, 0, 255, 100, 800, False, False)
# pits_b = rescale(blues, 0, 255, 100, 800, False, False)
# pits_g = rescale(greens, 0, 255, 100, 800, False, False)

# Uncomment for version which uses each pixel
# pits_r = rescale(reds, 0, 255, 200, 550, False, False)
# pits_b = rescale(blues, 0, 255, 212, 650, False, False)
# pits_g = rescale(greens, 0, 255, 212, 700, False, False)

# Version which uses the averaged chunks of the image
# Rescale RGB values to frequency values
pits_r = rescale(reds_avg, 0, 255, 200, 550, False, False)
pits_b = rescale(blues_avg, 0, 255, 212, 650, False, False)
pits_g = rescale(greens_avg, 0, 255, 212, 700, False, False)

# Create a sequence which will iterate the above pitches
seq = Seq(time=.01).play()
it_r = Iter(seq, choice=[pits_r])

it_b = Iter(seq, choice=[pits_b])

it_g = Iter(seq, choice=[pits_g])

# Print the chosen frequency
# pp = Print(it_r, method=1, message="Frequency")

wave_r = Sine(it_r).out()
wave_g = Sine(it_g).out()
wave_b = Sine(it_b).out()

# Uncomment to add a harmonizer on top of the pitches
# hr_r = Harmonizer(wave_r).out()
# hr_g = Harmonizer(wave_g).out()
# hr_b = Harmonizer(wave_b).out()

# Runs the audio system
s.gui(locals())
```

Fig. 4 - Python Script

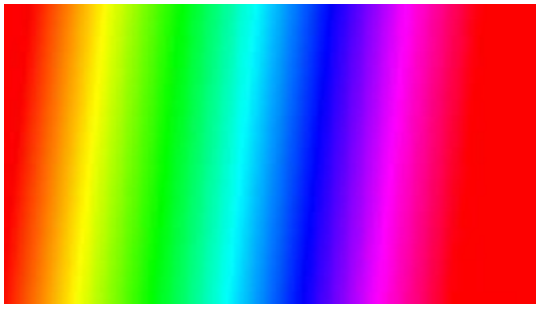

Fig. 5 - Rainbow (“Rainbow Image”)

Fig. 6 - Black and White Galaxy (Salah)

Fig. 7 - Leona (Romero)