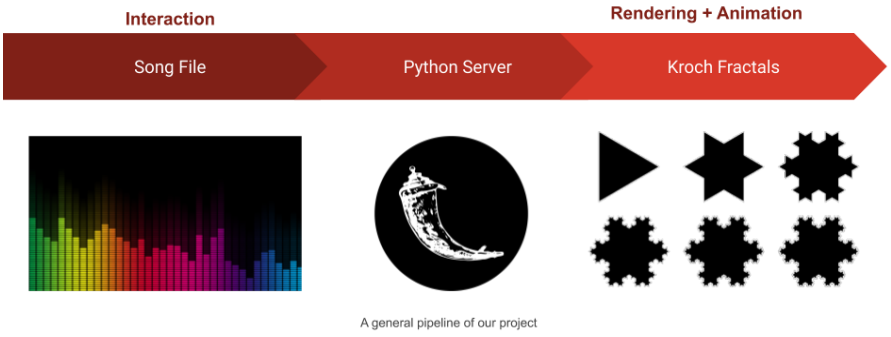
CS4621 Project Final Report

Team Members:

Riley Niu(hn263), Jinwei Shen(js3559), Dongqing Wang(dw532), Cynthia He (qh43)

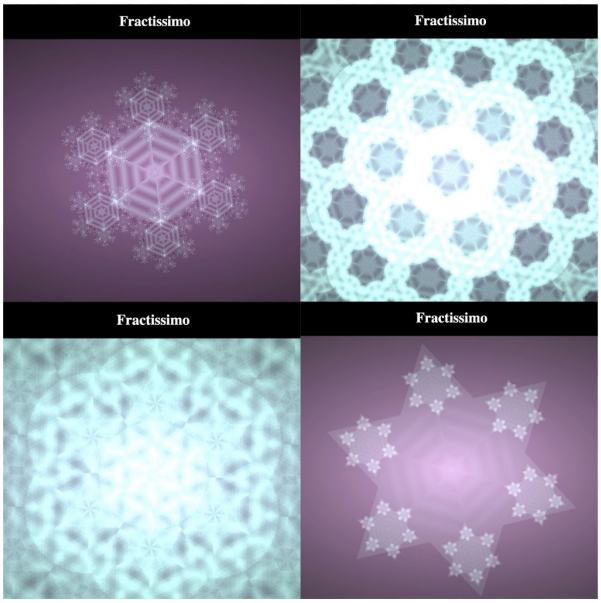
Goal of the project:



A general pipeline of our project

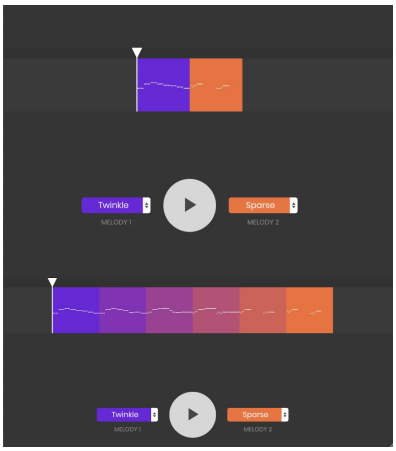
This project proposes a 2D music visualization application which generates iterative shapes and fractals that are rendered in real time and synchronized with the music as it is played. We made use of a music analysis open source library, Essentia, as well as the Web Audio API in processing the audio files and extracting the information. Our overall goal is to make a creative, artistic, and interesting graphical interpretation of the music pieces using specific algorithms that transform results from musical analysis (frequency-domain information with an awareness of "spikes" in the audio that often correspond to percussion hits, beats-per-minute for the visualizer's rotation, and "danceability" for the colors of the visualizer) into graphical elements, or in particular, fractals. Fractals, as a family of iterative geometric figures, has been a popular and useful math/computer graphics concept in modeling structures in which similar patterns recur at progressively smaller scales such as a snowflake, and in describing partly random or chaotic phenomena such as crystal growth or fluid turbulence. By visualizing music using 2D fractal elements, we hope to explore mathematical beauty in the form of algorithmic art by calculating fractal objects and representing the calculation results as real-time animations, as well as demonstrate our learning outcomes in computer graphics practicum class.

Description of what we accomplished + Technical Details:



As discussed during our final demo, we have worked on combining different parts of the pipeline that we have been working on individually, including music generation, music analysis, server connection, and graphics rendering, and eventually decide to make use of some of them and discarding the rest in order to achieve better outcomes. For the sake of showing our efforts, we will detail both the components that we decide to include and the ones that didn't make it into our final application. In general, we successfully completed our goal: **a fractal music visualizer**!

Music Generation (discarded)



- Two options for user music generation:
  - Local generation: collect sample of 3-part music composition with the length of 16 bars
  - Web application: preset 4-bar long music melody sampel that are capable of generating long sequence of music.
  - We choose to not include this part due to the lack of musicality of the song generated by this model which potentially works against our visualizer.

Analysis details:

We adopted a pre-existing open source library of algorithms and tools for audio and music analysis, description, and synthesis called Essentia, provided by the Universitat Pompeu Fabra in Barcelona. The library, developed in C++ and wrapped in Python and accessible through a Python interface, includes a number of music algorithms and tools, of which we used MonoLoader, EqloudLoader, RhythmExtractor2013, BeatsLoudness, Danceability, PredominantPitchMelodia, PitchContourSegmentation, and YamlOutput.

When the user uploads a song file (mp3, m4a, wav, aiff, or flac), a function in main.py saves the song file within the ./2D-music-visualizer/analyzer folder so that its address can be passed as a parameter to the function in essentia_python.py, where the analysis takes place.

Once the function in essentia_python.py receives the song file address, the above listed algorithms and tools are called with the file address as a parameter in order to perform the analysis and produce a value or array of values (i.e. beats-per-minute, frequency, danceability, etc.). All of the values are then compiled into “output.json” and outputted into ./2D-music-visualizer/analyzer to be referenced within JavaScript to alter the visualizer. If a pre-existing “output.json” from the previous song exists, the upload of the new song file will automatically delete it and create a new one so that no old values are re-referenced.

Server side details:

Our project relies on backend Python script to perform music analysis, so there has to be communication between JavaScript and Python script. When user uploads a song, the JavaScript needs to send the music file to the Python for analysis, which would output a json file containing relevant information and send it back to the JavaScript. One of the common ways is to use JQuery AJAX POST request, but the native Python http.server command cannot handle POST request. So we built a Python web server using Flask to handle AJAX requests and render the HTML webpage.

To render the HTML template correctly, we set up the file structure such that all HTML files are placed in a folder called “templates” and all static JavaScript and CSS files are placed in “static” folder.

In the Python script (main.py), we specified the port number for the web server and the HTML file to render, and defined a function (upload_file) to call the function in essentia_python.py to perform music analysis upon receiving the POST request. The result of this POST request creates and returns the “output.json” back into the JavaScript for manipulating the visualizer.

Then to send the POST request, we simply put the Python function name in the URL field, put the music file into a supported data structure, and wait for the response of the POST request to process the received data.

Visualization details:

Since our goal is a 2D visualizer, we used a full-quad shader as demonstrated in the previous lectures. Our vertex shader simply passes necessary information such as uv coordinates to the fragment shader, and most of our rendering algorithms are contained in the fragment shader. The most challenging part of graphics rendering for us is to achieve all fractal generations based on each pixel, which is an entirely different perspective from most of our original references that uses object-oriented languages.

We pass the results (i.e. frequency, danceability, BPM etc.) from the music analysis to the shader using uniform variables. In the fragment shader, we first defined various shape generation functions that will be used as base geometries for evolving fractals, including circles, triangles, hexagram, and skeleton, etc. These functions simply test for implicit representations of a given 2D position. We then linear interpolate these shapes and cross-fade their alpha based on the frequency in the blendshape function. We used Koch Fractal algorithm as our main fractal generation approach. The Koch snowflake is constructed by starting with an equilateral triangle, then recursively altering each line segment as follows:

- divide the line segment into three segments of equal length.
- draw an equilateral triangle that has the middle segment from step 1 as its base and points outward.
- remove the line segment that is the base of the triangle from step 2.

In the main function, we generate koch fractal with 5 iterations that use geometries generated from the blendshape function. The rotation speed of the visualizer depends on a ratio directly in proportion to the newly calculated BPM of the song, same as the fadeSpeed value in the function drawShape. The background color is added to a vec3 of current frequency value so that its intensity changes as well. We also used a uniform variable u_time that is constantly changing and multiplied it by the uniform variable dance, a ratio calculated by the newly generated danceability value of the song. By wrapping the sine function outside of it we are able to use it to add variety of the changing patterns animate the fractals constantly. The final pixel output is the koch fractal output mixed with background color and pre-defined fractal color.

Instructions to run the code:

1. (For some reason WebAudio Api sometimes doesn't work on Safari, so just to be safe, run the application on Chrome if at all possible).
2. Installations:
   a. `pip install flask`
   b. `pip install ipython numpy matplotlib pyyaml`
   c. `pip install essentia`
3. Navigate inside /2D-music-visualizer and run `python main.py`
4. Copy the prompted link into web browser to see application running (i.e. http://127.0.0.1:4625/). If it says "Address already in use", go into `main.py` and scroll down to the bottom and change the port number.
5. Upload a song file (mp3, m4a, wav, aiff, or flac) where it says "Choose file"
6. Wait while it's loading… Good things come to those who wait!
7. Enjoy a fractal visualization of your favorite song!
8. Feel free to upload another song file!

Reference:

- MusicVAE: Creating a palette for musical scores with machine learning. https://magenta.tensorflow.org/music-vae

- Koch Fractal: https://www.youtube.com/watch?v=5FtuP3AwPIE&list=PL3POsQzaCw53vmvWr-Ye-R0d3NPJzp25P&index=4
- Music Theory: https://www.musictheory.net/
- Essentia: https://essentia.upf.edu/documentation/index.html
- Koch Fractal: https://en.wikipedia.org/wiki/Koch_snowflake