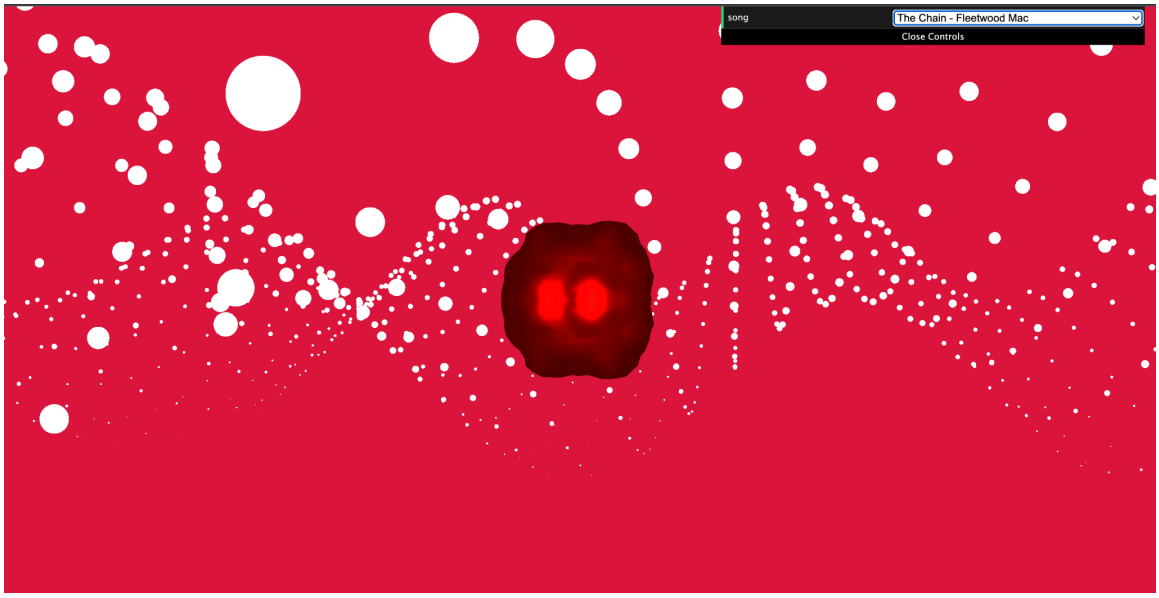
Nostalgia Media Player: Written Report

Brianna Butler and Natalie Reptak

Github: https://github.com/bributler927/nostalgic_music_player
Live Demo: https://bributler927.github.io/nostalgic_music_player/



Abstract

This project will allow for users to watch some of their favorite songs come to life! The Nostalgia Media Player was inspired by Windows Media Player's Visualization feature which provided cool animations to sound files. Our project contains 7 popular songs and allows the user to choose from any one of them. Then it visualizes those songs with the help of our Spotify Data dataset. The visualizations include a 3D shape, a wave in the background, and differing colors for these objects and the background. For songs that have higher danceability scores, the object distorts and “bubbles” faster. For more energetic songs, the bubbles grow on the object. For higher valence, the visualization includes brighter and "happier" colors. For loudness, the wave in the background grows or shrinks in size. For now we utilize an icosahedron as our main object. However, in the future we plan on Spotify compatibility, adding different shapes and allowing for more interactions with the user.

Introduction

For this project, we attempted to create an audio visualizer that takes into account characteristics of songs such as their valence, danceability, energy, and frequency statistics. This visualizer includes a 3D point wave in the background with a spherical Blob, which develops certain spikes and bubbles, as the main centerpoint. Our project took a look at seven different

popular songs and utilized a Spotify dataset as well as Web Audio API to gather song information. With this information, the background and blob colors would vary based on valence, the wave speed and amplitude would vary on BPM, and the blob would change shape based on energy and danceability. For songs with higher valence, the blob and background will have brighter and happier colors, with blue being more negative, red having a mid-positivity, and lime green being the happiest. For energy and danceability, the more danceable a song is, the more spikey/bubbly the blob, and the more energetic, the larger these bubbles and spikes get. Finally, for the wave, we look at the BPM measurement for the song and change the wave speed and amplitude to be higher the larger the BPM. With all of these functionalities in place, the user will be able to move their camera around the lit-up bubbling blob whilst witnessing the large point waves in the background moving to the music. In addition, we are in the process of utilizing Web Audio's Analyser to make the wave and/or blob vary based on FFT analysis of the song.

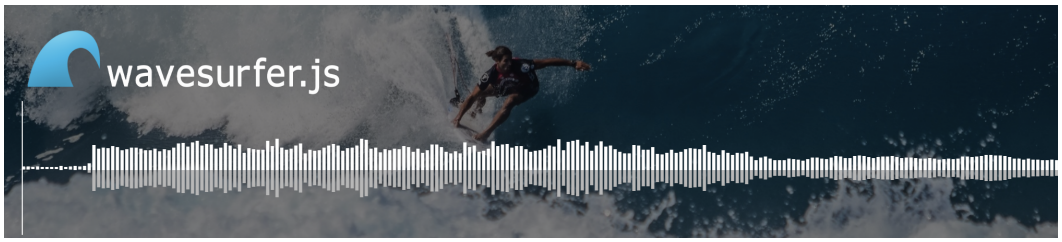
We do not have a specific target audience, nor do we see a particular demographic receiving benefits due to our project. We made sure to include well-known/popular songs of different genres so that anyone who opens our visualizer can play and see a song that they enjoy. Rather than discuss a specific social group benefitting from our project, we would like to point out the technological requirements for our project. Those who view our project should have access to a working internet connection, working speakers or, preferably, headphones, and an internet browser (Google Chrome is preferred).

Previous Work

As mentioned in the introduction, related work includes the Windows Media Player, a music visualizer available on Windows computers that was a staple part of most early 2000s kids' childhoods. The nostalgia-element to this visualizer is the inspiration for the name of our project. This media player featured explosion-like effects and changing colors that changed for each song played through the player.

Above: Image of Windows Media Player, one of the main inspirations for this project.

Various past music visualizers create visualizations in a variety of ways. For the Windows Media Player, most of the visualizations were completely random. Many visualizers online use the frequencies in the song audio itself to construct their visualizations. One of these includes wavesurfer, a customizable audio waveform visualization that is built on top of Web Audio API and HTML5 Canvas. Wavesurfer is known for its visualization that resembles that of a laptop music studio computer program. Wavesurfer is also available for programmers to use within their own individual projects. It is its own standalone visualizer, but it is also a tool.



Above: Image of wavesurfer home bar, which features one of the frequency wave visualizations you can make with its product.

Another audio visualizer we drew inspiration from was Specterr. Music visualizations made with Specterr and those that are similar to the ones made with Specterr are popular amongst Youtube videos that simply contain the audio to a song. These visualizations have two main characteristics: an eye-catching object in the foreground and a steady, yet adapting background visual playing behind the foreground object.

Above: Screenshot of a single frame of a Specterr visualizer example.

Our own visualizer follows this formula, but with a 3-D twist - most of these visualizations are limited to 2D visuals. In fact, the 2D limitation is where most of these visualizers “fail”. There are very few 3D visualizers available online. And if they do exist, they are not well known. Our visualizer will not only be 3D, but it will also allow the user to move the camera around the scene to view different perspectives of the visualizer.

Approach

For approaches, we first reasoned on how to gather song information. Our beginning step was attempting to receive the Spotify API key which we were not able to get in time. After this we approached the project by utilizing a combination of a Spotify song dataset on Kaggle as well as Web Audio Analyser. For the blob, we first tried to cover a sphere with a cloth and attempted to create the blob visualization this way. However, we found a more efficient way after viewing other ThreeJS examples. It was slightly difficult to set up the loading JSON dataset. We at first attempted to load and access it in our scene js file. However, after issues with asynchronous loading and promises, we decided on loading the dataset in our app js and accessing the songs in our scenes js. A couple other approaches and considerations are described in Methodology as well.

As of now, we know for a fact that if the user chooses only the songs listed in the GUI, there should be a song-personalized visualization available from the visualizer. We have only downloaded the audio files of the songs listed within the GUI. Because of the structure of the GUI, it's relatively impossible to choose a song we do not have an audio for. The only time the visualizer should not "work" is when the page initially loads. This is because no song information has been selected, and thus, the visualization cannot adjust its appearance based on the selected song's values.

Methodology

In order to execute our approach to this project, we needed 4 main pieces: song audio, song Spotify data, the foreground "blob", the background wave, and the frequency visualizer. Implementation of the song audio included downloading the wanted song files, loading those song files into the project, and playing those files. Song Spotify data was found through an open source dataset on Kaggle. The blob featured in the foreground of the project and the wave features in the background was implemented through the provided features of the THREE.js library.

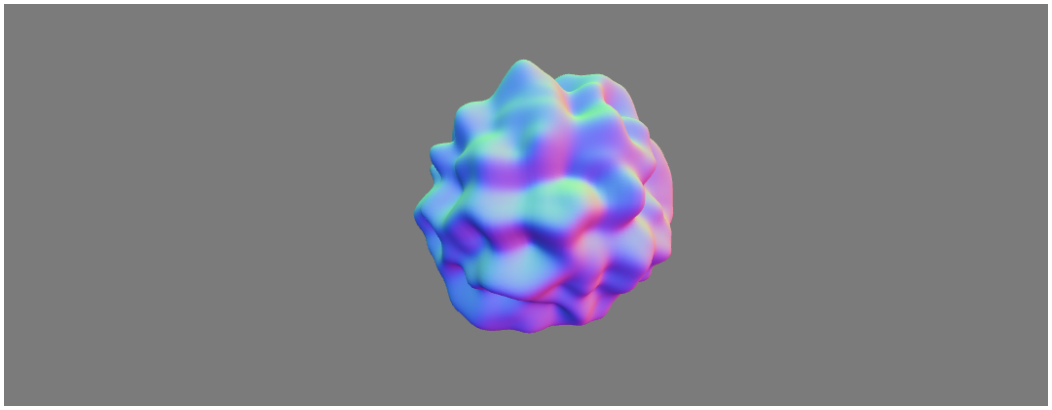
Each audio file was obtained through Youtube. Using an online Youtube to MP3 file converter, we were able to find the song audios we wanted, convert them, and download them for use in our project. For the loading of the audio files, there is mainly one way to play local mp3 files in a javascript project. Through utilizing the built-in Audio javascript class, we were able to create audios, play those audio files, and stop the audio from playing when necessary.

As mentioned above, we obtained our song data from a dataset titled "Spotify - All Time Top 2000s Mega Dataset" by user Sumat Singh. There are many datasets available on Kaggle that have Spotify data on hundreds to thousands of songs. We chose this specific dataset because it contains the data for the most popular songs on Spotify, enabling us to use songs that are well-known, and thus, be more likely to be popular with our users as well. The dataset contains these songs and Spotify factors/labels used to describe the characteristics of each of the songs including valence (mood of the song), danceability, energy, loudness, and BPM. These specified characteristics were the values we used to influence the look of our visualizer when each is chosen. There are also characteristics of the songs like year released, artist, top genre, and speechiness that we decided to not utilize within our project.

Because of the nature of visualizers, there were an infinite number of possibilities we could have gone with for the visualizations made by our project. There were a few driving factors for the objects and look chosen for our visualizer. First, we wanted to ensure our visualizer was unique and looked different from other visualizers currently available on the internet. Secondly, we wanted to work with objects that were easy to manipulate automatically whenever a song was chosen. So, this included animated objects that could change patterns and their look depending on a song's Spotify values. Third, from the beginning of the project, we had a relatively clear view of what we wanted the visualizer to look like, so we sought out objects and items that could best represent our original idea of the visualizer.

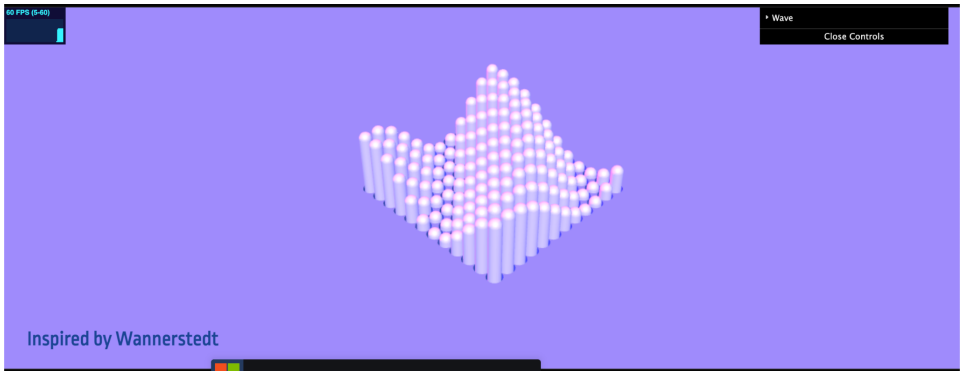
Firstly, in terms of gathering audio, the initial proposal of this project was to directly connect to Spotify through their API. Through the Spotify API, we would have been able to gather the user's ten most recently liked songs and retrieve the relevant statistics of those songs (danceability, energy, valence, etc.). This would then allow the user to choose between those two songs to play and visualize on the website. Once the user selects a song, the Web Playback SDK would be utilized to play it. However, this involved receiving the Spotify API key, which we still do not have access to. In addition, it would most likely lead to our group spending more time on just resolving how to play audio rather than working on the visualizations which are more important. We had another idea of then just allowing the users to upload their own MP3 and we would utilize ThreeJS's audio library to analyze it. However, this library only provides us the FFT of the audio, and no other information like a Spotify dataset. After meeting with our advisor, we decided to perform a mixed response. We found a Kaggle dataset that contained statistics about 2000 songs that would provide us the information the Spotify API would. With this, we would select a few from there and then only allow the user to select from those. After that, we would also still utilize the ThreeJS audio library to add more visualizations through FFT analysis. Hence this gives us the opportunity to use both song characteristics and frequency analysis. The only issue is that this limits the user to what songs they can play.

Another possibility was involving how many blobs we would choose to use. Originally, the plan was to use several blobs throughout the canvas and have them move around randomly based on song characteristics. However, this turned out to look visually unappealing and messy, leading us to stick to having one blob in the middle of the screen. This one blob would then represent the characteristics of the song (in addition to the colors we utilized and our wave characteristics in the background). The moving aspect of the blob was inspired by this CodePen example. We liked the application of perlin noise in creating a "bubble" effect. There were other examples of dynamically changing balls online that we considered (like this one), but most were very difficult to understand or implement.



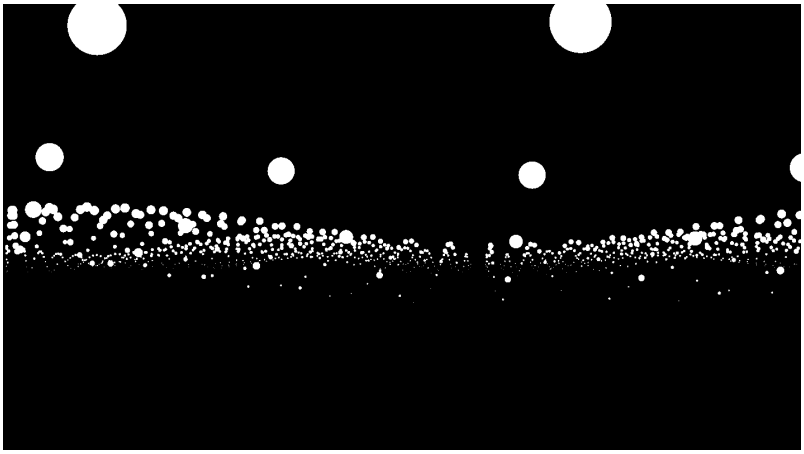
Above: Blob example we decided to draw our main inspiration from.

Regarding the wave, there were different wave examples we looked at and built our ideas off of. In our very first idea of this project, there would be just one wave line in the background, changing based on loudness and BPM. However, upon further research, a more 3D approach would look more visually appealing as well as looking more visually correlated to the songs. The types of wave backgrounds we looked at all varied but didn't quite match the background effect we wanted (such as this CodePen example and this one).



Above: One of the wave examples we passed on due to it not being a "background" effect.

They all rather appeared to be "main effects" and wouldn't be able to quietly fill in the background for the blob to take over the main part. However a ThreeJS example perfectly fit our idea of what we wanted from a background, it just required some visual tweaking and editing to be responsive to music.



Above: Wave example we decided on.

Results

We measured success for this project through how aesthetically pleasing and accurate our visualizations were with respect to the selected song. If the song was slow and sad, our visualization should be more mellow appearing as well, the background and the blob should be blue and the wave should move slowly with low amplitude. However, if the song was fast, the colors should be bright and the wave in the background should move quickly with high amplitude. We wanted to ensure that the song characteristics gathered from our dataset were vividly and accurately shown and that our music also played correctly. This success measurement involved us tweaking certain weightings for amplitudes and speed until we got something that properly showcased the different movement characteristics. We did not want the wave to move fast for a slow song or slow for a fast song, so we had to ensure the mathematical operation we fixated on for updating wave speed properly updated for slow and fast. A similar approach was done for the blob characteristics as well, with higher energy and danceability causing many large spikes.

For experiments, this involved switching through multiple songs and changing the viewing angle. So we randomly switched from certain songs such as “Hallelujah” to “Despacito” then to “The Chain” and back to none. This allowed us to check if our visualizations were properly updating to the song change. During these types of tests we realized that there were certain issues with our song data set. The BPMs listed for some songs were incorrect, “Take Me Home, Country Roads” had 164 BPM, but upon further research, the proper BPM is 82. This then required us to take a second look at our dataset and ensure that each characteristic listed is correct, which we did not plan on doing in the beginning of the project. Only selecting a small amount of songs to play provided a benefit here since it allowed us to double check if the dataset we are utilizing has correct values listed. After changing the BPM, we were able to see that our visualizations matched that song more accurately.

Another test was to move our camera position around, and see if the visualizations stay consistent. Our camera is focused on the blob in the center but we can move it around the blob and see if we still get nice visualizations from any angle. Once doing this, we realized that our lighting is only on the blob from a certain angle. This makes the blob look almost black in the

darker areas and makes the color and movement difficult to see. Therefore, this experiment allowed us to see that we should change the lighting to be from the camera hence ensuring the blob will be lit up from wherever the user views it from. In addition, by changing the viewing angle, we double checked that our wave background was operating correctly, and did produce a background effect. The particles extended far out in each horizontal direction, and properly produced wave effects.

Discussion

From the beginning of the conceptualization of the project, we wanted the nostalgia music visualizer to have compatibility with the Spotify API. However, due to time constraints and the time it would take to gain an access token, we were unable to incorporate Spotify compatibility with our final project. However, despite not being able to use the Spotify API, we did structure the alterations to our scene so that they are responsive to Spotify defined categories for the chosen song. Currently, we are using a JSON file with song data and their respective Spotify values to apply these scene alterations. Because of this, the obvious next step for our project would be to pursue our original plan of allowing users to choose any song available on Spotify through the Spotify API. This would allow the user to have nearly an unlimited amount of choices for a song to play and visualize. Unfortunately, with this next step, our users would need to have a Spotify account to use our music visualizer. The current version (the final project submission) uses local audio files, thus allowing anyone to use our visualizer.

Another future step we were thinking of adding to our project is to enable user interaction with the objects in the music visualizer. The original motivation for this project, the Windows Media Player contained amazing visual effects for the songs it played, but users were not able to interact with these effects or alter them, leaving users wanting more. For example, we think that, as a boredom killer, it would be nice to interact with the foreground blob, move it around, stretch it, throw it, etc. as the user listens to the song.

From this project we learned a variety of skills in and outside the realm of graphics. Outside our graphical learnings, we delved into JSON loading and parsing, Audio playback, basic javascript syntax and functions, Kaggle datasets utilization, and Web Audio API's functionalities. Through the Web Audio API, we were able to parse frequencies present in songs through the use of the Analyser Node. From the Kaggle dataset, we found out how to load large JSON data sets asynchronously and parse them correctly through javascript functions such as fetch(). We never previously worked with interpreting Kaggle datasets, specifically in a javascript environment so we learned about converting CSV files to JSON, and how to workaround the asynchronous fetching. We also learned how to properly filter the fetched results in order to get the characteristics of the song we wanted.

In the realm of graphics, we learned more about lighting, camera positions, materials, vertex/fragment shaders, IcosahedronGeometry, BufferGeometry, BufferAttribute, and Points. In particular, regarding lighting and camera, we learned the differences between certain lights and cameras. After figuring this out, we were able to figure out how to shine a light from the camera to ensure that whatever the user is looking at is illuminated through the use of point lighting

attached to the camera and ambient lighting. We also learned how to create a beautiful wave background utilizing many points, bufferGeometry,bufferAttribute, vertexShader, and fragmentShader. In particular, through use of vertexShader and fragmentShader we were able to figure out how to create ShaderMaterial, which allowed us to have our points change in terms of size (allowing them to fade in and out as seen). Then we were able to generate a series of points that were able to follow mathematical functions based on sine to get their respective position and scale. The wave was very difficult to understand at first, but once figuring it out, we were able to edit the speed in which the wave moved, its amplitude, and the size of the points – hence giving us the ability to properly represent the musical data.

For the blob we learned how to dynamically, randomly change objects in accordance with time and how to scale these changes for program performance. Through course lessons, we were able to understand the role of perlin noise in creating randomly generated domains, and we mapped these domains to our object through its vertices. Then we learned how to activate these changes in accordance to the timestamp logged by our computers. We also had to learn how to scale this time stamp in a way that allowed the objects to "move with the music". Performance limitations on the blob (for example, introducing too high of peaks slowed down the entire program) helped us learn how to find a good balance of extreme enough changes and performance of the program. Overall, this project required us to thoroughly understand the examples we looked at to be able to properly edit and apply them in a way we wanted to. In addition, this project required us to learn other JSON, Javascript, Web Audio tools in order to add our audio and its analysis.

Conclusion

The main project was more complex than initially thought, our data gathering and ensuring objects were properly created and responded correctly involved significant debugging time. Retrieving our JSON data asynchronously was a challenge due to complexities involving waiting for the data to load to ensure promises were fulfilled and then figuring out where we could fetch the data in an async context. Then once this data was loaded, after playing around with it, we realized that there were issues with the data set as well. Certain songs had an incorrect BPM listed, making a slow song appear very fast and then making our visualizations seem strange and incorrect.

In addition, when operating with different objects and examples, such as the particle wave, there were issues with incorporating it into a larger project. Many examples online had the wave as a standalone feature, and utilized html files and other script sections. This was especially noted when implementing our point wave in which many examples used fragment and vertical shaders written in html files to add to the wave functionality. However, since we were mostly using app.js and scene js from the starter code we had to edit this, and ensure our wave still had the appropriate shaders without needing to utilize certain code sections such as : "document.getElementById('vertexshader').textContent." Hence there was some digging needed to be done to incorporate these shaders without needing to add another document element. After that was done, with some other tweaks our wave functioned nicely, then we just needed to update certain mathematical operations to reflect music BPM and loudness.

Even with these minor setbacks, we were able to effectively and efficiently complete key features of our application, with FFT analysis in the final stages of development.The next steps would be to completely finish the frequency analysis and visualization of these specific frequencies as well as tweak certain scalars to better reflect song qualities (such as wave speed based on BPM). To add to this project, we could also add more songs and more interactive features, as described in Discussion. There are also issues such as ensuring correct info in the dataset and ensuring the details of the blob are more visible by tweaking our lighting. Overall, we are very happy with how our project turned out and its aesthetic qualities and hope to add many more functionalities to it in the future.

Contributions

In this section, we included a brief list describing each team member's contributions to this final project!

Bri's contributions:

- Finding the Spotify dataset
- Editing the GUI to include chosen songs
- Implemented audio player
- Construction of the blob
- Alteration of the blob and background by Spotify values

Natalie's contributions:

- Downloading song audios
- Loaded song data from Spotify JSON file
- Construction of the background wave
- Alteration of the wave by Spotify values
- Implemented frequency analyzer (WIP)
- Fix the lighting of the scene to have blob illuminated no matter camera's viewing angle

Both:

- Choosing songs to include in the visualizer
- Narrowing down which visualizations to include in visualizer based on examples online

Works Cited

Blob Inspiration: https://codepen.io/farisk/pen/vrbzwL

Playing Audio StackOverflow: https://stackoverflow.com/questions/9419263/how-to-play-audio

Specter: https://specterr.com/music-visualizer/

Spotify Dataset: https://www.kaggle.com/datasets/iamsumat/spotify-top-2000s-mega-dataset

ThreeJS Documentation: https://threejs.org/docs/index.html#manual/en/

- Specific Pages: Color, IcosahedronGeometry, MeshStandardMaterial

Wave Inspiration: https://threejs.org/examples/webgl_points_waves.html

- Bypassed Wave Example 1: https://codepen.io/iondrimba/pen/wVgKoZ?editors=1010
- Bypassed Wave Example 2: https://codepen.io/seanfree/pen/GNKydX

Youtube to MP3: https://y2mate.is/enAB1MR/

Web Audio Node Info: https://developer.mozilla.org/en-US/docs/Web/API/AudioNode

- https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API
- https://developer.mozilla.org/en-US/docs/Web/API/AnalyserNode
- https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Visualizations_with_Web_Audio_API

Lights: https://r105.threejsfundamentals.org/threejs/lessons/threejs-lights.html

- https://stackoverflow.com/questions/45039999/three-js-light-from-camera-straight-to-object