ITP2 - Final Music Visualization Project

Description of new components

GUI.js
This class replaces the ‘Controls and Input’ function. It draws a floating, movable window that contains buttons to switch between visualizations and a button to control playback. GUI.js is based on the concept of ‘Immediate Mode GUIs’ (IMGUI). Although there is some debate as to what constitutes an IMGUI, more information on the subject can be found here https://sol.gfxile.net/imgui/index.html and https://www.youtube.com/watch?v=Z1qyvQsjK5Y&ab_channel=CaseyMuratori

At its core GUI.js is a draggable rectangle that is an anchor for, and visually contains, widgets.The GUI also has an array of widgets (in our case, buttons) It tracks the status of the mouse and the state of the widgets (normal, hot (in our case this means mouse over), and active (being interacted with, in our case, clicked). The GUI has functions for drawing itself and it’s widgets, handling dragging, and region hit checking (is the mouse over something).
The widgets handle drawing themselves and decide their own interaction state.
The GUI also has visualization specific widgets.

My window didn’t drag smoothly, I found the solution was to use the *mousePressed* and *mouseReleased* functions from this example:
`https://editor.p5js.org/enickles/sketches/H1n19TObz`

Why didn’t I just use p5.gui? In the time required to figure out how it works, how its dependencies work and how to integrate it with my project (if that would even be possible) I could make my own gui, suited to my needs, that I can easily extend, modify, and reuse.

Rtt.js :Rapidly-Exploring Random Tree
This is easily the most complicated extension, implementing a k-d tree data structure to support a Rapidly-Exploring Random Tree (RTT).The implementations of these algorithms in this extension are not robust and were created to support the visualization. To be reused, they would need to have additional functionality and to be properly tested, this applies mainly to the kd-tree as the RTT is fairly simple.
Details about Rapidly-Exploring random trees may be found at this website written by it's creator: http://msl.cs.uiuc.edu/rrt/about.html
A video describing Rapidly-Exploring Random Trees can be found here:
https://www.youtube.com/watch?v=2ugFAYu8E6I&ab_channel=LearnlyLearnaboutmanythings

The kd-tree is not essential for the RTT, however, as more points are added, the time needed to compare closest points becomes unacceptably high for real-time use.
*It turns out that actually drawing the lines for this visualization is the real bottleneck.

The kd-tree turned out to be fairly easy to implement, there is ample information about the subject and
Wikipedia has a very detailed article about kd-trees: https://en.wikipedia.org/wiki/K-d_tree and this video series was very helpful:
https://www.youtube.com/playlist?list=PLguYJK7ydFE7R7KqRRVXw23kOrn6jiwqi

With all that said, I will give a high level description of the flow of this visualization.
1. To begin we have an array that stores points we have added to our scene . We seed this array with a starting point and add a root node to our RTT.
2. Next, we build a kd-tree based on the points in the array.
3 Then, a random point, *p*, is chosen and we find the closest point to *p* in the kd-tree, a point *closest*. A vector is created between point *closest* and point *p*. This vector is normalized and its magnitude is set so we can control the distance from the point *closest*. A new point, s, is created and its position is the point *closest* plus the vector. This point s is pushed on to the array and an RTT node is created based on this point.
4. A recursive function draws lines between the nodes in the RTT.
5. Goto step 2.

Wave3d.js: 3D Waves

This visualization is extremely simple except for some interesting functions that build custom polygon meshes.
The function *makeobj* takes a p5.Geometry object and populates it with vertex and face data, to create a grid/subdivided plane. This is based off an exploration I did with Unity several years ago (5 according to the file date) which in turn was based off a tutorial at https://catlikecoding.com/unity/tutorials/procedural-grid/ and much background knowledge gained from Nehe's old tutorials https://nehe.gamedev.net/ and it's ilk.

The function *makeNormals* takes a p5.Geometry object and computes the vertex normals. This is from https://www.iquilezles.org/www/articles/normals/normals.htm, the p5 calculate normals also uses this. (I suppose this has become the accepted way to do this)

https://github.com/processing/p5.js/issues/2369 had some information about how to manipulate the p5.Geometry object. Digging in the inspector and p5 source code also provided clues.

We keep an array of audio waveforms and map them onto our mesh every frame. It looked interesting but wasn't something I wanted to stare at for hours. I had a cool animated mesh but no idea how to use it. I moved on to other tasks and later decided to try again. This time I gave the mesh a black, specular material and mirrored the mesh vertically, and was very satisfied with the result.

Spectrum2.js: Spectrogram

This visualization is extremely simple, it draws the classic spectrogram view. The beauty of this visualization is, in my opinion, it's simplicity.
1. Each frame we draw the values from *fourier.analyze()* vertically along the right edge of the offscreen buffer.
2. We then draw the offscreen buffer to the main canvas.
3. On the next frame, without clearing the offscreen buffer, we draw it onto itself but translated 1 pixel to the left.
4. Goto step 1.

Spec_capsules.js : Spectrum Extension

This extension came about from a ‘discovery’ made while experimenting with ideas for another, now abandoned visualization. The ‘discovery’ is drawing a shape with a low fill opacity, moving it slightly and drawing it again, repeatedly will result in a smooth edge on the shape. Varying the colors between each repetition can give the illusion of depth and shading to the object. The effect was so striking to me that I had to follow it through and modify an extension using this technique.

Wavepattern2.js: “Wave Pattern WEBGL”

The inspiration for this visualization comes from the old WinAmp visualizers that made such an impression on me. This one is also fairly simple, aside from some buffer juggling.
The result was acceptable to me but this could be improved with a better understanding of GLSL.

ThreeDRoad.js: “3D Road”

This was inspired by some YouTube videos with a synthwave/techno/retro/80’s theme (whatever that really means, but there are a lot of them). *beginShape()* and *endShape()* are used to create the road and sides of the road. An offset is added to the uv’s to make the textures scroll. Arrays hold the position of the street lights and road decorations and we offset their positions until they pass our camera, leaving our view, then they are repositioned ‘down the road’. The streetlight models were made in a 3D modeling program. The background city image is a screenshot of an UnrealEngine city I made some time ago.

Wavesketch.js - ‘Wave Pattern Extension’

Included mostly to illustrate the progression in complexity as I became more familiar with javascript, P5 and the theme of music visualization. This was the first extension I did and it served as a test for more advanced ideas needed for the later visualizations. Those ideas include:

Using createGraphics to create an offscreen drawing buffer.
How would this affect the structure of the extension? How will this interact with the main visualization sketch? Are there any caveats to be aware of when using this?

Using an array to store waveforms.
How do arrays really work in javascript? What are their convenience functions?

Drawing things and seeing them in motion.
Does this look nice? Does it ‘visualize’ the music? What can be done with basic P5 drawing functions?

This extension was an invaluable first step and answered the initial questions I had while also generating new ideas and more questions. It also provided an opportunity to consider the workflow for the project.

Additional References
https://www.w3schools.com/js/default.asp