Music Visualizer

DH2323: Computer Graphics & Interaction Final Project

Ayushi Singh: ayushis@kth.se
Aparajitha Adiraju: adiraju@kth.se
Rohin Bhasin: rohinb@kth.se

Table of Contents

Project Specification

What we did

For our final project, we created a music visualizer that takes in music as an input to our program and generates an associated output. We used C# in Unity to visualize a continuous generation of a geometric shape/image. To do this, we used a procedural content generation technique - specifically, we created a Lloyd relaxation of a 2D Voronoi diagram. We used existing resources and Unity inputs to process the music input and get the spectrum, pitch, and database values.

Features

- ➔ Use the beat of the music to be shown through the audio spectrum
- ➔ Corresponded pitch value to colors in audio spectrum
- ➔ Generate a continuous Voronoi diagram as the background using Lloyd relaxation

How work was split

- ➔ Ayushi worked on audio processing, as well as the blog.
- ➔ Aparajitha worked on integrating the csDelaunay library into the music visualizer as well as altering existing Unity components to fit the scope of our project.
- ➔ Rohin worked towards altering the Delaunay library to allow for necessary modifications to the Voronoi diagram. We were not able to integrate this ultimately due to the implementation constraints of the imported library. However, we used this color inspiration elsewhere in our project.
- ➔ We all worked on various smaller parts of the project, such as getting different colors working, setting up the Unity scenes, and debugging different issues.

Blog

Our project specification, progress reports, screenshots, and a screen recording of our project can be found at https://graphicsmusicvisualizer.wordpress.com/.

Aiming for: D-E

Introduction

We liked the idea of visualizing music and using that as a way to model our objects.

Voronoi Diagrams

A Voronoi diagram is an algorithm that partitions a space of geometric objects into cells, where each of these cells contain points closer to one specific object in comparison to any others. Specifically, it takes an arbitrary collection of points and counts the nearest point for each pixel and assigns that pixel to that point. There are different ways to compute the "nearest" point, such as euclidean and manhattan distance. The goal of this algorithm is to produce an even distribution of previously provided points. For this reason, Voronoi diagrams typically look more organic-looking divisions in comparison to systematic rectangular or circular methods. There are a variety of algorithms that can implement this concept, such as Fortune's algorithm.

Lloyd Relaxation

Lloyd's relaxation algorithm determines the Voronoi diagram for a set of points and repeatedly moves each point toward the centroid of its Voronoi region. This eventually converges to a standstill Voronoi diagram, and we incorporated this into our project to generate a dynamic Voronoi diagram where the shapes would move. However, ours is continuously moving and reset, and therefore never converges.

Music input

For the music input, we were initially intended to use the NAudio library to do the audio processing but ran into lots of issues with that which are listed later in this report. Instead, we followed a YouTube tutorial (linked on our blog) to write a script that extracted the spectrum, pitch, and database values from the audio file that was provided as input, as well as create a simple music visualizer.

Implementation and Challenges

Our Unity engine has a quad rendered with a circle of cubes that propagate based on the noise level of the music. We read the audio from an AudioSource and then calculate the noise level and the pitch. The pitch is then used to change the color of the wheel. We used a csDelaunay library (1) since it was a platform agnostic codebase built in C#. After that we found a basic Voronoi diagram class that allowed for a static diagram to build, and we used that and extended it to add a Lloyd relaxation animation in the background of our visualizer. There is also more information in the blog on implementation specifics and challenges.

How Unity works

Setting up Unity was one of the hardest parts of getting our project to come together. We were all unfamiliar with the Game Engine, and it was hard to connect the scripts to the Game Engine and work with built-in libraries. It was also challenging to get existing projects incorporated effectively.

Voronoi implementations

Given that this is a pretty complex technique, there was a wide variety of different implementations and resources surrounding this and Delaunay and Lloyd relaxations. After a lot of research, we came across a Unity library that would fit well into our project and that we could modify to fit our music visualizer.

NAudio

Originally, we were looking at Naudio as a potential audio processing library, as we noticed that the API provided a lot of helpful outputs. However, we all develop on Macs and we did not realize that this was a Windows-specific library used to take Wav input, so we looked into other sources to aid in the audio processing and extraction.

Polygon coloring using rasterization

We also attempted to color individual polygons within the Voronoi diagram. We tried to use the technique from Lab 3 in this: using the edges to draw all the lines as rows within the polygon. There were a couple problems we encountered in this. Firstly, the library we used doesn't properly implement clipping of edges along the sides of the shape, and therefore we aren't able to consistently get the edges and color in the polygon. Moreover, when trying to implement this ourselves, we are not able to effectively determine which line segments between vertices are actually edges, and which go through the polygon. This was not a problem in the lab because everything was represented as a triangle.

Related Work

Other music visualizers

Before embarking on this project, we did some research into other existing music visualizers and how they work. The main research that is being done with music visualizers is primarily on getting more details from the music quicker. Getting things like beats, percussion spikes, and interesting rhythmic information from music is challenging in conjunction with graphic visualization because the graphics have to be generated real time. Some preprocessing of the

audio file enables more complex visualization from the music. Using the information extracted from the music, there are a variety of visualizations that can be applied.

Voronoi

Voronoi diagrams and their implementations have a lot of uses outside of just the realms of interesting graphics visualizations. From reference (6) we can see that there is a technique called polynomiography that focuses on the relationship between Voronoi diagrams and polynomial roots. Since polynomial roots are such a fundamental concept in mathematics and other sciences, Voronoi diagrams can often be used to visualize complex scientific interactions.

Future Work

There are a few key things that we can add to this project to enhance the user experience and add more to this project.

We could modify the type of visualization we have using different parameters. For example, the visualization “cubes” could be placed at the centroids of the Voronoi diagram instead.
One thing we wanted to add was to have tempo detection to affect the pace of the Lloyd relaxation. Right now, we recreated the Voronoi diagram every n iterations, where n is a number that we picked to make it look good. In the future, we would want to adjust n based on the tempo of the song. This is a more difficult problem to solve as beat detection is an ongoing algorithm, and we discovered a couple existing implementations that could be integrated into our current C#/Unity platform.

We can also try and dive even deeper into the Voronoi implementation to see if we can decrease the speed at which it converges, and use that in conjunction with the previous parameter to manipulate the image based on the tempo. We could also add some algorithmic implementations of these different techniques within to script to extend existing libraries or create our own. There are a variety of algorithms that implement this method.
Another idea we had was to take in keyboard input from the user so they can have some control over the color scheme that appears.

References

1. https://github.com/PouletFrit/csDelaunay
2. https://forum.unity.com/threads/delaunay-voronoi-diagram-library-for-unity.248962/
3. https://www.youtube.com/watch?v=wtXirrO-iNA
4. https://answers.unity.com/questions/157940/getoutputdata-and-getspectrumdata-they-represent-t.html
5. http://pcg.wikidot.com/pcg-algorithm:voronoi-diagram
6. https://www.ics.uci.edu/~eppstein/gina/voronoi.html
7. https://link.springer.com/content/pdf/10.1007%2F978-3-642-41905-8_1.pdf