Music Visualizers in Unity

Ian Rapoport
Computational Media
University of California Santa Cruz
Santa Cruz California United States
irapopor@ucsc.edu

Jacob Wynd
Computational Media
University of California Santa Cruz
Santa Cruz California United States
jwynd@ucsc.edu

James Coulter Petnic
Computation Media
University of California Santa Cruz
Santa Cruz California United States
jpetnic@ucsc.edu

Kindon Smith
Computational Media
University of California Santa Cruz
Santa Cruz California United States
kimasmit@ucsc.edu

Abstract

For our final project, we wanted to explore different ways to visualize music in Unity. Our visualizers include a trigonometric beat detecting audio visualizer, a vertex shader that uses shadergraph and particles to create interesting deformation, a series of particle systems that respond to different pitches, and a bar graph style shader that should be reflective of the different frequencies.

KEYWORDS

Music Visualization, Unity,

1 Bar Graph Style Visualization by Ian Rapoport

This visualizer was meant to create a image similar to the one below in Unity.

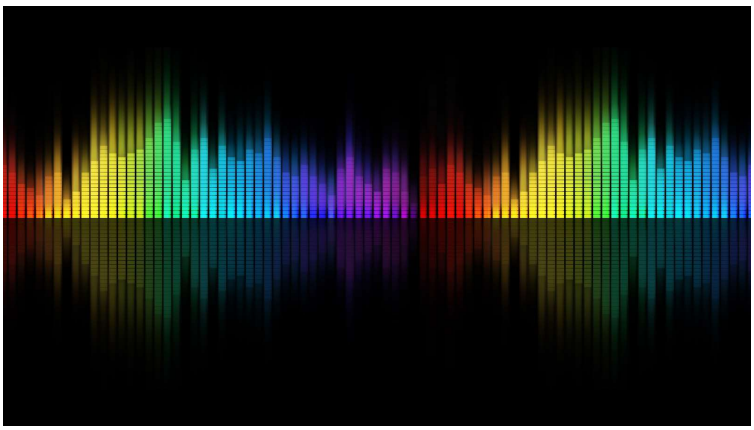


Figure 1: **A reference photo found at:** **https://steamuserimages-a.akamaihd.net/ugc/1006036902670958 54/9D7B84FE1FCE7E15EA2A1B18210B13C1C12242F1/**

This was accomplished using a c# script that would scan and quantify audio that would then pass this information to a shader that would assign colors to regions to create bars based on audio data.

1.1 C# Audio Reader: ConvertAudio.cs

This was accomplished by adapting the code given for the palm tree lab assignment. This involved taking a audio source in Unity and getting its spectrum data with the GetSpectrumData function. This turned the spectrum data into a size 256 float array. The next step was taking this array and converting it to an array of the users specification. This meant taking values from the array and averaging them into buckets. The user was given the following customization controls:

- Color[] colors: An array of the colors used for the visualizer (the number of colors determine the number of buckets to reduce the spectrum data too).
- Magnitudes[] magnitudes: An array that tracked the minimum magnitude and maximum magnitude for each bucket. Magnitudes is a custom class that simply stores a min float and max float.
- Float length: Designates the horizontal length for the bar graph.
- Vector3 center: Designates the center point for the bar graph.
- Float top: Designates the max y position of the bar graph.
- Float bottom: Designates the minimum y position of the bar graph.

- Bool lerpColors: Tells the shader whether or not it should use all the colors listed in colors, or simple make all the colors lerped values from the first and last color.
- Bool lerpMagnitudes: Decide whether or not the bars should simply reflect the magnitudes or should lerp between values to get more smooth movement.
- Bool useWorldPositions: Decide if the visualizer should use the models object position or world position.
- Float lerpSpeed: If the magnitudes are being lerpe, how fast should they lerp.
- Renderer[] renderers: The array of renderers that the visualizer should be applied to.

These values are passed into the shader.
The values in each bucket are turned into a value from 0 to 1 based on what there position is inbetween the high and low magnitudes.

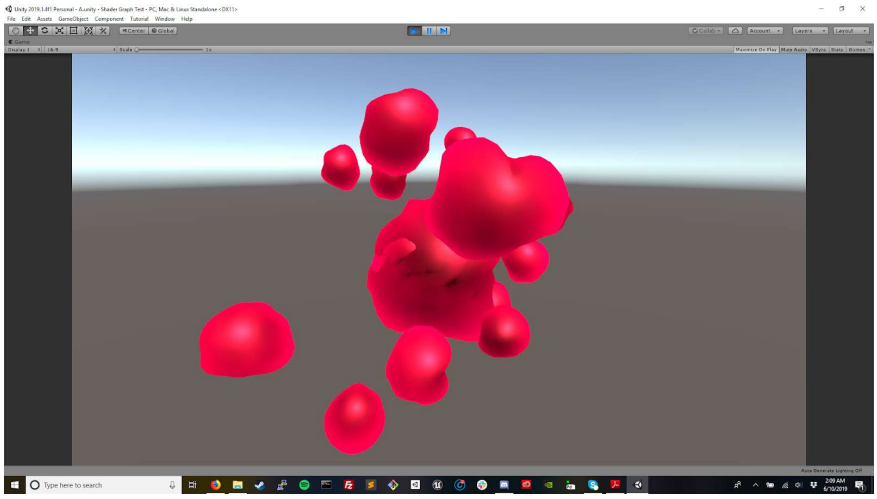
1.2 The Shader: IanVisualizer.shader

The only shader value that is set as a property is the base color (the color of the object when the visualizer isn't over it). All other values are assigned via ConvertAudio.cs. This way the visualizer can be consistent across multiple objects. The shader simply takes these values, determines if the fragment is within the range of the visualizer determined by the center, top, bottom and length. The magnitude dictates what percentage of area from the bottom to the top should be colored via the visualizer. The specific visualizer colors is decided by seeing how far the position is from the left most point of the visualizer, and using the count of colors to see which color it falls under.

Figure 2: **6 different objects in a scene with the visualizer. The two cubes, capsule and sphere in the center show a visualizer using world coordinates with all colors. The two spheres on the sides show a visualizer with local coordinates and the colors lerped.**

2 Vertex Displacement Using Audio by James Coulter Petnic

My goal was to create a shader that, similar to the one we created in our lab sections, would deform an object in a scene using values taken from an audio file. Initially the final result was not clear, but simple experiments with particles and the shader graph quickly led to inspiration:



The only pre-established goal was to create something using the shader graph feature made available in more recent Unity versions.

2.1 TreeBend.cs

As implied by the name, this script was made from the palm tree script provided in the lab section of class. It takes the spectrum data from an audio source and divides it into an array of "buckets," the float values of which can be used to manipulate elements of the object and its shader in the scene.

Only one bucket is currently used to affect the shader. Its value is checked against a threshold value, and when the threshold is reached the object vertices expand based on a noise function.

2.2 New Shader Graph.shadergraph

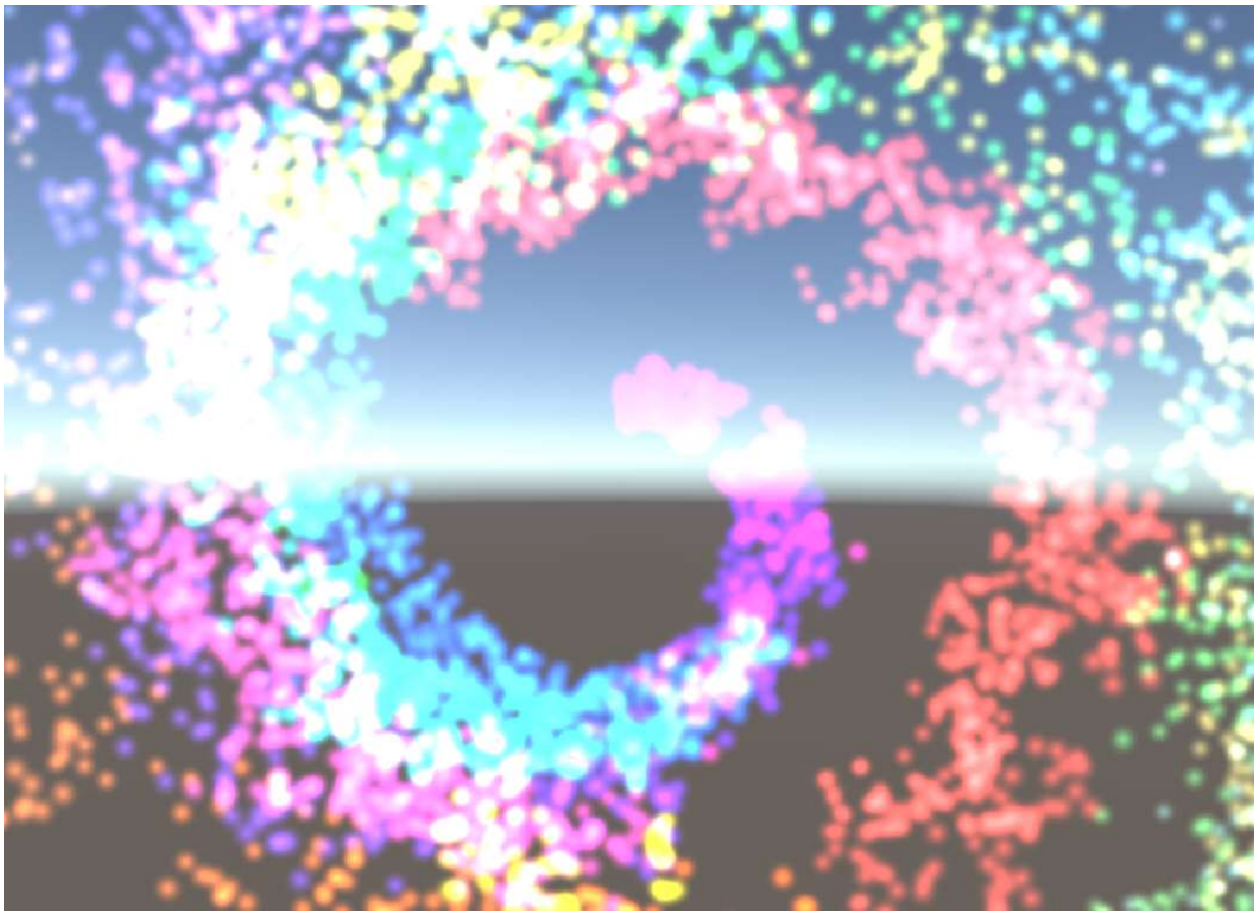
A primary motivating factor behind this project was to learn how to use the Unity shader graph feature, which allows for node-based scripting that seems to be far more intuitive than normal shader coding.

The shader works by using the output of a simple noise function multiplied by an amount acquired from the scene's audio source to displace the vertices of a given object along their normal vectors. The result is an orb that seems to flow and become distorted like liquid based on the sound file being played in the scene. The object also has a

particle system attached to it, and each particle has the same shader applied to it, all of which are controlled by the original TreeBend.cs script on the parent object. This leads to a semi-lava lamp effect where high spectrum data values from the audio cause the orb to deform and eject smaller orbs of liquid, which quickly rejoin with the parent object. The color of the object and its particles also changes based on the amount of displacement, and an exposed border value allows the user to change the amount of outer glow applied to the object. Overall the shader is meant to create a very kinetic and fluid visual display to accompany the audio.

3 Particle Systems Using Audio by Jacob Wynd

The goal here was to use particle systems that would clearly change with the music and make interesting visual effects. Ultimately this was achieved primarily through c# scripts capturing and operating on the audio data, and another c# script controlling when particles were emitted. The colors of the particles were also controlled by a shader.



3.1 W_AudioPeer.cs

This file captures audio data from the audio listener and stores it in a public static array so that it can be easily accessed by other files. It operates on the audio listener so that the system can be easily adapted to projects that might use multiple audio sources.

3.2 BucketMover.cs

This script takes the audio data gathered above, and reduces them to eight floating point values stored in the array called buckets. It determines how much of the spectrum data goes into each bucket based on the values entered into the cutoffs array. This allows the user to adjust the sensitivity of each individual particle system. At each frame, this script rotates a center of rotation object (of which the particle systems are children) in order to get the particle systems to move in concentric circles. BucketMover also calls a function each frame in EmitAtHeight in order to emit particles.

3.3 EmitAtHeight.cs

In an earlier version of this particle system, the emitters moved up and down with the music, giving this script its name. However it turned out that rotating the emitters in concentric circles had more visual appeal so now this script contains a function which takes a degree of rotation and emits particles based on how far the emitter has rotated each frame. Coefficients in this function were adjusted until it appeared as visually pleasing as possible.

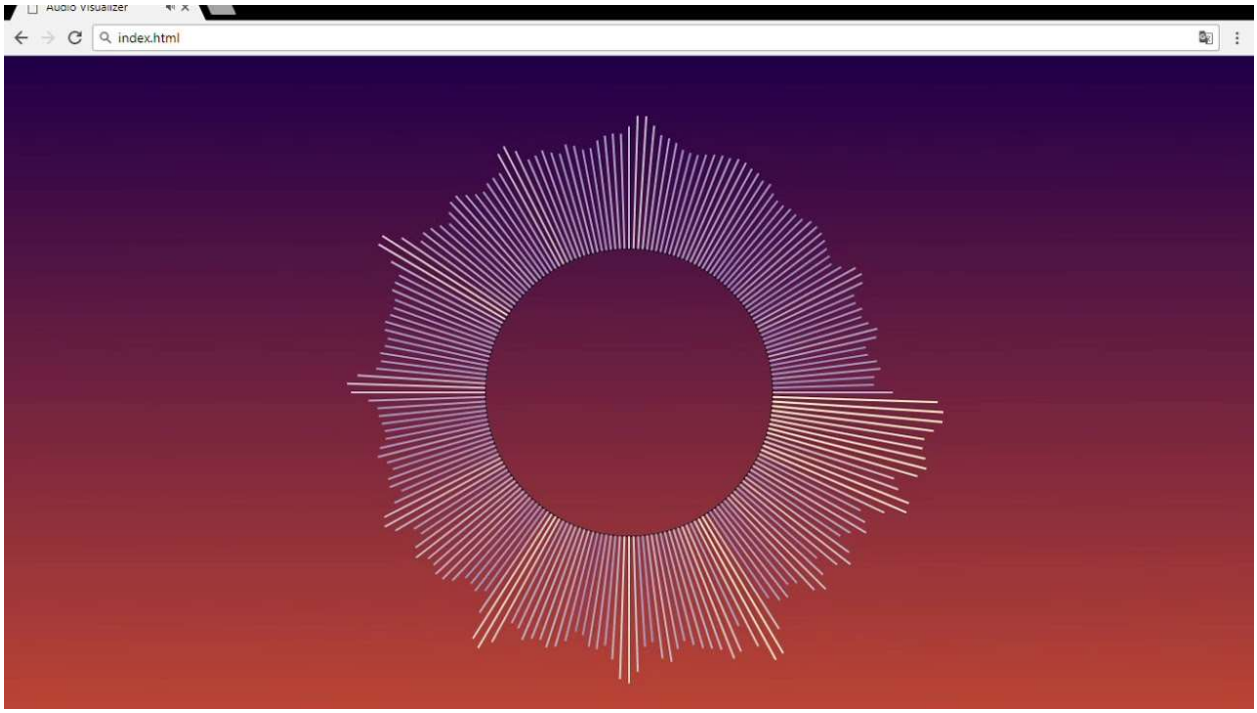
3.4 ParticleSmoke.shader

This shader interpolates between two colors based on the age of the particle, and reduces the alpha of the particle over time. For this system eight materials were made for the eight different particle systems, each with a different starting color, they all fade to a gray smoke color with a low alpha as they drift off screen. This fading is based on the current age of the particle in relation to its total lifetime.

4 Trigonometric Visualization of Audio by Kindon Smith

My goal was to create a unique sine wave visualizer for any given audio input. This visualizer would need to have specific frequency data and split input sounds into bands. This was accomplished by primarily c# scripts taking in audio data, spawning objects,

and manipulating them into a sine wave visualization.



4.1 SpectrumData.cs

Capture audio input data from an audio listener in Unity3D. This data is then put through a fourier transformation to prevent bleeding and used to assign bucket values for a variety of arrays and buffers. These arrays are used to set color, scale, size, and act as buffer i- betweens for lerping.

4.2 AudioSync.cs

Syncs audio data so it's not a constant surge of data input. There are stop/breakpoints where beat's are detected, but outside of those timeframes input data is recorded but disregarded as noise.

4.3 RingCubePopulator.cs

Populates the main scene area, rotates their Euler Angles based on the number of objects in the scene, and transforms them so they form a ring. Then takes Spectrumdata and applies transformations to them based on the buffer arrays. This gives the appearance of a double sided sin wave surrounding the scene that the camera is in.