DH2323 - Computer Graphics and Interaction
Audio Visualization and Procedural Generation

Simon Edström Kawaji - kawaji@kth.se
Emil Jonsson - emiljonss@kth.se

May 2019

1 Introduction

We have created an audio visualizer program that can take an MP3 file as input and creates a real time, procedurally generated visualization of the music. The visualization consists of a tunnel that the main camera travels through while the tunnel creates a visual feedback to the music depending on the amplitude of the different frequencies from the currently played music.
To change the currently playing song, hover the mouse over the lower part of the screen and click on the song you want to play.

Blog:
https://jabroniboys.tumblr.com/
Video Demonstration:
https://www.youtube.com/watch?v=1disArJQr6Y
Video with just music:
https://vimeo.com/user60646127/review/338746771/69bcb0de29

2 Related Work

2.1 Conveyor Failure Diagnostics Using Sound Visualization Technique

[2] used Sound Visualization Techniques to help monitor the status of conveyor belts in industrial plants. Due to the fact that many of the conveyor belts are hard to access it becomes trouble some to perform diagnostics and monitor the belts. Something worth noting is that belts often produce much more noise when something is wrong in the machinery or there is a problem with control of correct operations. A solution proposed was to use acoustic cameras to identify and localize the source of the extra noise. A sound visualization would make it easy to notice when and where a problem occurred.

Both projects are related by that facts that it is important for the visualization to represent the sound as closely as possible. In Moravecs case it is important that its easy to distinguish when somethings goes wrong. In our program it is important that the user interprets the visualization as correct and for that it must also be a close representation of the audio.

3 Implementation and Results

The audio visualizer was made with the game development software Unity [4] which uses C# as a scripting language. We ended up with a product which was quite different from our original test we had in our project specification but is quite similar to the idea of the final product that we had. Our original vision was to have a procedurally generated tunnel that the camera traveled through, while the tunnel had some sort of changes depending on the music. Our final product is basically our original vision but with added features like smooth curves to the tunnel and particle effects that could travel on the tunnel lines.

3.1 Procedural Bézier Curve Generation

The tunnel in the visualizer is procedurally generated by randomly creating quadratic bézier curves and connecting them together. In our final version, we have three bézier curves that exist at the same time, always replacing the oldest one when a new is created. To make the transition from one curve to the other smooth, we made sure to keep the middle point of two connected curves to be on a straight line that goes through the point that connects the two curves like in 2. Once the camera has passed from one curve to the other, the old one will be deleted and a new curve will be generated at the end of the frontmost curve. The curves are divided into segments that generate a circle of randomly placed vertices and draw lines to a specified amount of neighbors. These vertices will also act as the spawn point for our particles that travel on the tunnel lines.

When we generate a new curve we a first only generate and compute the positions of the Bezier points. We do this by generating three random values. Two floats for units of length, and one for degree of rotation. ForwardL is the length moving in the direction of the camera at the first point of the Bezier curve and is used to get the second point in the bezier curve. The SideL is the second variable and tells the distance the third point will be from the second point on the curve. We then use the rotation variable to rotate this third point around the direction the curve is pointing in the first point as seen in 1.

Then we create one segment and its corresponding vertices per frame until we have filled the curve. This is to avoid creating it all at the same time which would make the program lag for a short while. When creating the vertices we also create the lines between them at the same time.

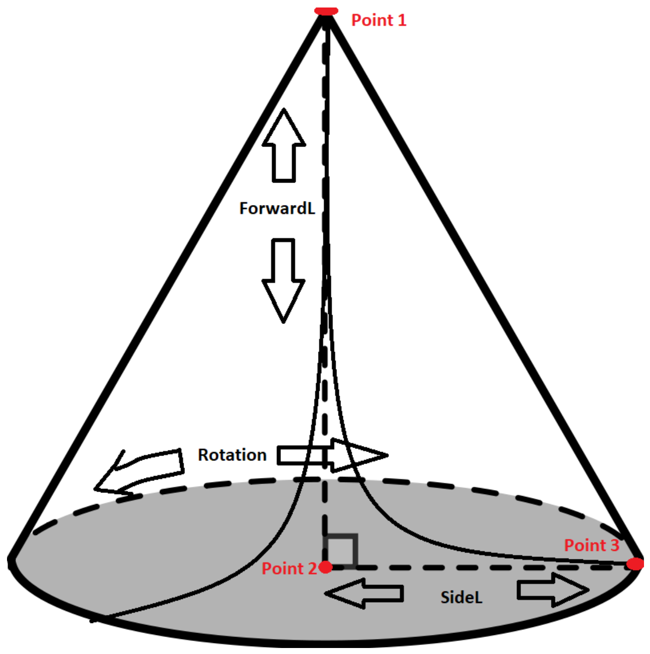

Figure 1: Generation of New Bezier Curves

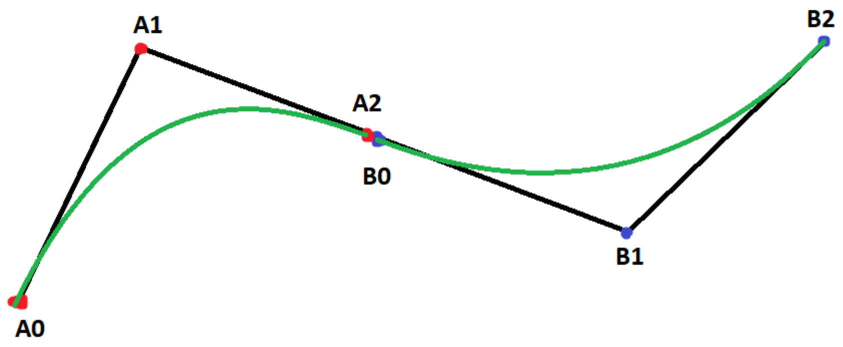

Figure 2: Two quadratic bézier curves that are connected like in our project

3.2 Audio Visualization

The music that has been chosen as input will be analysed by the Unity implementation of the Fast Fourier Transformation (FFT) function [1]. We then take 512 samples of different frequencies from the output of the FFT function and group them together into 8 groups of frequency levels. These 8 groups of frequencies is what we use for our visualization.

The procedurally generated tunnels' radius will temporarily increase depending on the amplitude of the low frequencies in the music that is currently playing. The tunnel will also create particles that travel between the network of vertices on the tunnel. The color of the tunnel and the particles are dependent on the amplitude of the different ranges of the music. We have assigned the red color value to be dependent on the low frequencies, the green color value to be dependent on the middle frequencies and the blue to be dependent on the high frequencies.

3.3 Particle Effects

At first we tried using the Unity particle and trail system to get the desired result, and after a long struggle we decided to create our own system. The particles that are used in this project are created by creating two points that travel by linearly interpolating between vertices in the tunnel. One of these two points has a delay, and will start moving later then the other point. Then we draw a line between these points which results in the particles that can be seen in our project. Their The particles will react to the music by changing color, spawn rate and movement speed. They also creates a "chain" effect by spawning a new particle at the vertex that they traveled to and moves towards a random neighboring vertex.

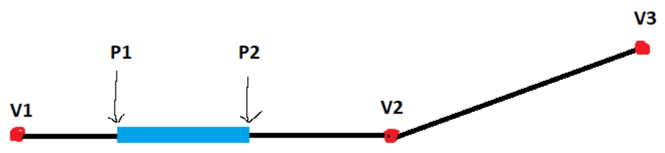

Figure 3: Three vertices V1,V2,V3 and the particle that is made of the points P1 and P2

4 Discussion

4.1 Solved Issues and Problems

4.1.1 Gimbal Lock

When we first started creating the Bézier Curves we were just rotating the different parts around the x,y,z axis. We realized that we sometimes got stuck just travelling in a straight line and no longer turning. Since we generated the curves using random degrees of rotation we didn't immediately realize what caused it since it only happened rarely. We read about Gimbal Lock and realized we could solve this by changing to rotations to instead use Quaternions. Unity had many features of these already implemented so it was not that difficult to switch.

4.1.2 Potential Improvements

One improvement could be to have the curves immediately react to switching a song. Since we draw curves ahead of time we will not instantly notice that a song has been changed, but a few seconds later when the new curves are reached. One solution to this could be to remove all current curves and draw entirely new curves upon switching a song.

Ideally we would also be able to get the BPM of an mp3 file so we could use this parameter when generating the visualization. There is no easy way to do this with Unity/c so we would have to implement this ourselves using some form of FFT system, or alternatively try seeing if someone else has created a solution to this problem.

We could also make the visualizer be more dependent on the music and use less randomness. For examples patterns could be generated on the tunnel walls depending on the amplitude of different frequencies in the music. This was a part that was difficult because using variables from the music could potentially create strange visualization scenarios and would require a lot of testing with different songs and fine tuning of variables.

5 Perceptual Study

A perception study could be conducted where we have users trying to determine which visualization correspond to which song. Several visualizations could be played back at the same time along with the song and the user would have to chose a one/several of these which the user thinks represent the song. Another way to do it is to have a user listen to a song and simulation and rate on a scale of 1 to 5 on a poll with how well the visualization represents the music, also having visualizations created for completely different songs. If users have a hard time determining which visualization goes with which song we could improve it by having the visualization depend even more on the exact properties of the song and less on randomness.

We could also do a study where we show one visualization at the time for the same song and then change variables such as camera speed, tunnel radius scale, particle spawn rate to see which of the visualizations the subject thinks corresponds best to the music. This study would also use a poll where the subjects answer questions like "Did the visualizer correspond well to the music?" and rate them on a scale from 1 to 5.

This would give insight on how well the visualizer captures the music and help us change and improve parts that the poll could help us find and fine tune the different variables in the program.

6 Individual Contributions and Resources used

Emil mostly worked on the procedural generation part, like generating Bezier Curves and creating vertices.

Simon mostly worked on the sound stuff and how our visualization interacts with the audio.

Much of the stuff we worked on together sitting beside each other or at least talking to each other and constantly discussing the different possible solutions.

6.1 Unity Post Processing Stack

We used a couple of effects found in the Post Processing Stack Library developed by Unity Technologies available on the Unity Store [3]. Specifically a Fog effect, Bloom, Chromatic Aberration and a Vignette effect.

References

[1] *Fast Fourier Transform.* 2019. URL: https://en.wikipedia.org/wiki/Fast_Fourier_transform.

[2] Marek Moravec et al. "Conveyor Failure Diagnostics Using Sound Visualization Technique". In: *Advances in Science and Technology Research Journal* 12.4 (2018), pp. 144–150. ISSN: 2080-4075. DOI: 10.12913/22998624/100352. URL: http://dx.doi.org/10.12913/22998624/100352.

[3] *Post Processing Stack.* 2019. URL: https://assetstore.unity.com/packages/essentials/post-processing-stack-83912.

[4] *Unity.* 2019. URL: https://unity.com/.