Audio Visualization. It’s a field that has fascinated artists and scientists alike for hundreds of years, and challenged programmers for nearly four decades. Like synesthesia, or the ability to sense a stimulus through a sense not usually employed for that stimulus, an audio visualizer gives a normal person a way to experience sound through light, and light through sound. I propose to create a program that not only visualizes music, but creates that music algorithmically, in real-time. The shapes and sounds created will be new every time the program is run, and the equations governing their evolutions will be based on fractal properties such as recursion, iterated function systems (IFS), probabilistic IFS, and fixed-dimension fractal construction. In this proposal, I will give a description of my tools, research, intended research, and aims, as well as an expected timeline of my progress.

Tools

The first and probably most important question is what language to build my visualizer in. I have had experience and feel most comfortable with C and Haskell. I am okay with C++, but if I used it for this project I would need to brush up a good amount to understand what’s really going on. My suspicion by reputation is that C/C++ would have better performance than Haskell, but I feel that this is outweighed by two things: first, I believe Haskell’s packages for audio representation, analysis, and manipulation are better to work with and more expressive than their C counterparts; and second, I personally find a deep attachment to Haskell and other functional programming languages as an extension of my mode of thought, and believe that their use, especially in a program involving so much mathematics, is entirely called for. Of course if the performance gap is so wide that it literally renders the program useless, a different language must be used. My first checkpoint is to write a program in Haskell with the same maximum complexity I’ll need in the visualizer and try to get a smooth rendering (see Aims/Timeline).

Another advantage of Haskell is that I am already well familiar with its Euterpea package, which provides an impressive structure for the composition and manipulation of MIDI music. I learned about and used Euterpea by taking CPSC 432, and I find it to be an elegant tool for composition. Through Euterpea, notes can easily be strung into chords, chords into sentences, sentences into sonatas, with full control at each level. Oscillators can be linked and filtered to produce never-before-heard sounds.

The two main graphics libraries I am considering are SDL and OpenGL, which are both fairly generic graphics libraries and are available for both Haskell and C++. I have used SDL for 2D work in Haskell, and OpenGL for 3D work in C++; my impression is that OpenGL will be better for 3D fractal renderings, so I plan on using the OpenGL package for Haskell.
Research

So far, most of what I know comes from classes here at Yale. I have an excellent understanding of how computers interface with sound thanks to my classes in Music Technology as well as Computer Music. I also have a decent background in composition, and have learned formal music theory from the music department. Abstract Algebra and Fractal Geometry (MATH 190 and 290) have inspired me to think of the mathematical properties inherent in everything in this world, even (or especially) the things that are most artistically beautiful. And I work well with functional programming, as I’ve already mentioned. These experiences have prepared me well for a project like this.

Still, I will need to learn much more, starting with other people who have worked towards the same goal. There are many programs now available that draw graphics to music, and even a few I’ve come across that create their music at the same time. I’ll examine these for what I do and don’t like, and draw inspiration from them. One book I expect to find very helpful is *The Haskell School of Expression*, which teaches Haskell through multimedia examples. I’ve found at least one web tutorial on using OpenGL in Haskell, and intend to complete several before using it in my project. Another book, *Multifractals and 1/f Noise* by Benoit Mandelbrot, should prove a valuable resource in realizing the different forms of fractal behavior and how those properties are implemented in the physical world. Finally, I plan to study the work of Richard Voss, who has examined the fractal properties of noise and music. His work suggests that a self-affinity and $1/f$ relationship is a core characteristic to much of the music we already know and appreciate.

Another avenue I hope to explore is a community of fractal animators who compete to make beautiful fractal exploration videos with programs of the smallest size. I’ve seen one video that comes as a 4 KB executable file, and when run, produces a four-minute high-definition voyage into a 3D fractal. This community of artists is very open about their work, and they post regularly on internet forums about their efforts. I’d very much like to reach out to them or adopt some of their practices, since saving space and in general making the program more elegant are always virtues.

Aims

My purpose for this project is both technical and artistic. I want to build a fairly complicated program to hone my coding skills. At the same time, I want to express myself through music and images, and one of the things I’d like to express most is my appreciation and awe for the beauty of mathematics. Taking a project from conception to a polished product is something I’ve never done before. I will have produced something that I can demonstrate and distribute, something that will hopefully give someone using it some of the joy that I put into it. And with the technical skills to pull off a program like this, I’ll be able to make more programs that deal with audio and video, which I’d certainly like to continue doing.
Timeline

1. Determine complexity of most computationally intensive task, implement some program of comparable complexity in Haskell and make sure that the computation and rendering are feasible. If results are outrageous, switch to C/C++. (September 30)

2. List, then refine through experiment, the qualities of the fractal that I would like to and will be able to use. List attributes of the audio/visual composition and connect traits that could be linked algorithmically. (October 7)

3. Learn as much as possible about other visualizers, especially how they tackle performance issues and what parts of the music they are using. Study Haskell and learn the OpenGL package through tutorial. (October 14)

4. Write a spec for my program: what its inputs and outputs are, what it is expected to do and how quickly. Detail important functions within the program and how they interact. Chart the control flow. (October 21)

5. Write code for computation of the music and graphics elements. Transform user inputs into data needed to play and display. (November 4)

6. Write code to render graphics and audio. (November 18)

List of Deliverables

Code (Haskell source file)
Final Executable
Written Report of Project