STUDY ON THE PROCEDURAL GENERATION OF VISUALIZATION FROM MUSICAL INPUT USING GENERATIVE ART TECHNIQUES

A Thesis

by

CHRISTOPHER MICHAEL GARCIA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2011

Major Subject: Visualization

Study on the Procedural Generation of Visualization

from Musical Input using Generative Art Techniques

STUDY ON THE PROCEDURAL GENERATION OF VISUALIZATION FROM MUSICAL INPUT USING GENERATIVE ART TECHNIQUES

A Thesis

by

CHRISTOPHER MICHAEL GARCIA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Philip Galanter
Committee Members,	Joshua Bienko
	Jeff Morris
Head of Department,	Tim McLaughlin

May 2011

Major Subject: Visualization

ABSTRACT

Study on the Procedural Generation of Visualization
from Musical Input using Generative Art Techniques. (May 2011)
Christopher Michael Garcia, B.S., Texas A&M University
Chair of Committee: Prof. Philip Galanter

The purpose of this study was to create a new method for visualizing music. Although many music visualizations already exist, this research was focused on creating high-quality, high-complexity animations that cannot be matched by real-time systems. There should be an obvious similarity between the input music and the final animation, based on the music information which the user decides to extract and visualize. This project includes a pipeline for music data extraction and creation of an editable visualization file.

Within the pipeline, a music file is read into a custom analysis tool and time-based data is extracted. This data is output and then read into Autodesk Maya. The user may then manipulate the visualization as they see fit using the tools within Maya and render out a final animation.

The default result of this process is a Maya scene file which makes use of the dynamics systems available to warp and contort a jelly-like cube. A variety of other visualizations may be obtained by mapping the data to different object

attributes within the Maya interface. When rendered out and overlaid onto the music, there was a recognizable correlation between elements in the music and the animations in the video.

This study shows that an accurate musical visualization may be achieved using this pipeline. Also, any number of different music visualizations may be obtained with relative ease when compared to the manual analysis of a music file or the manual animation of Maya objects to match elements in the music.

NOMENCLATURE

AS3	ActionScript 3.0
FFT	Fast Fourier Transform
3D	Three Dimensional
GUI	Graphical User Interface
MEL	Maya Embedded Language

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

1. INTRODUCTION

Music is one of the most powerful forms of artistic expression. Music has the ability to evoke emotion and provide incredible experiences to those who listen. John Whitney, a well-known researcher in the music visualization field, says, "Music is the supreme example of movement become pattern. Music is time given sublime shape."[1] Visual art and music have much to relate them to one another, even though they appeal to completely different senses of hearing and sight. They both deal with creativity in their composition, and both can influence the mood or attitude of the ones who experience them. For example, audiences of either medium may be saddened when experiencing a gloomily themed work or excited by a fast paced beat or by bright colors. In most cases, music and art contain a form of internal structure that makes the piece complex and intriguing. The combination of these two methods of artistic media can provide stunning results, some of which I have explored with this project.

Generative art is generally defined as "any art practice where the artist uses a system, such as a set of natural language rules, a computer program, a machine, or other procedural invention, which is set into motion with some degree of autonomy contributing to or resulting in a completed work of art[2].

This thesis follows the style of *IEEE Transactions on Visualization and Computer Graphics*.

Many music visualizers available today implement different forms of generative art, such as fractal systems [3].

Mapping between two domains such as audio and video art is considered to be a form of generative art. The system I have implemented includes a randomization element in the visualization creation phase that can be used to produce a variety of results. There is a complex system that controls the final visual output of the project. Namely, this system is the dynamics system in Maya that can be used to simulate real-world physics as well as stretch and bend those physical laws. Using this system, every data set that is input can produce a variety of different results.

Music visualization is the representation of musical information in a visual form. There are many ways to go about visualizing music. Many things you might not think of as music visualization could be accurately described as such. Dancing, or the actions of moving one's body along with a piece of music, is, at its core, a type of music visualization. As the dancer moves through space in time with music, an observer might be able to pick out some details, such as rhythm, about the music even if they weren't able to hear it themselves [4]. One of most basic, and perhaps most accurate forms is traditional musical notation (Figure 1.1), consisting of notes and scores [5]. Apart from differences in interpretation by an individual musician, musical notation can be used to reproduce the music it represents in its tone, rhythm, and volume [6]. Many well

trained musicians can simply look at a piece of musical notation, and understand what a song would sound like if it was played aloud. While there are some who possess this skill, certainly most of the general populace doesn't know much about reading this particular notation in all of its intricacy. Most music visualization seeks to represent the music in a simpler, more aesthetic mode. In general, music visualization usually takes the form of a picture or video where information from the musical score is represented by shapes, colors, and a variety of other means.

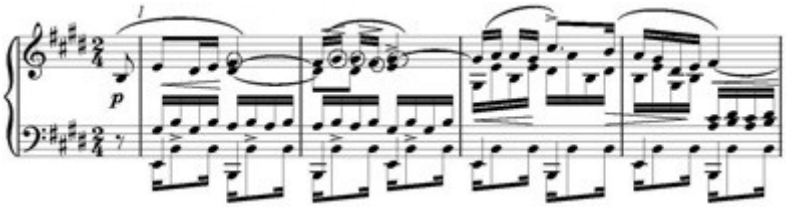

Figure 1.1: An example of normal musical notation

Ever since I was a boy, I have been fascinated by music visualization. Be it the simple visualization plug-in used in a media player or a professionally made music video, there is something about putting images to music that can increase the impact of each to a higher level than might have been possible with either alone. As my focus for the past few years has been learning all about the theory and creation of visual media, I wanted to create a music visualization that emphasized complex visual elements in a way that most typical, real-time plug-ins could not. Throughout my graduate program, I have become familiar with

many professional level applications such as Autodesk Maya and the Adobe Creative Suite and I wanted to combine my knowledge of these applications with my programming skills in order to create something unique.

The goal of this study was to produce a stable, easy-to-use pipeline for visualizing music that provides an alternative to manual music analysis and animation. While it may be possible to produce a more accurate or dynamic animation based on manual techniques, the time required will be much greater. This pipeline may also be used as a baseline tool for a more complex type of music visualization by allowing the user to tweak the final scene data to influence the rendered results. The final result of this pipeline should be a video set to a piece of music that has an aesthetically useful correlation to the sound (Figure 1.2). In this figure, a bouncing soft body object is distorted by a turbulence field that is controlled by data from the FFT analysis of low frequency ranges.

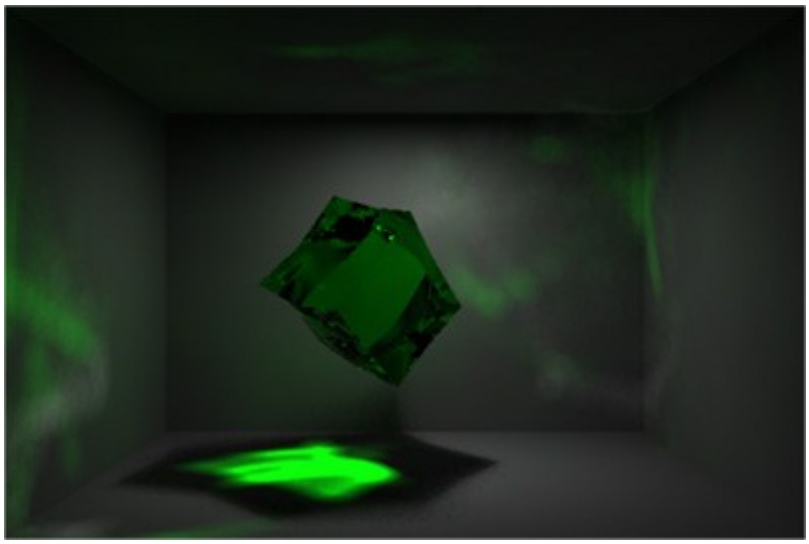

Figure 1.2: A bouncing soft body object

A second goal of this project was to make a visualization that is visually complex. Most music visualization takes place in real time with the music and therefore cannot take advantage of processor intensive rendering techniques such as ray tracing, caustics, and soft shadows. In this project, the pipeline not only allows for advanced physical simulations, but also gives the user an opportunity to render in high quality at any resolution. Due to the high-memory, high-processing requirements of such a render, the rendering cannot be done in real time. Another form of complexity that is available in the pipeline is the ability to track any number of frequency bands during music analysis and map them to practically any Maya attribute. During the music analysis portion of the pipeline, the user views FFT spectrum data and may pick any number of spans along the

spectrum to monitor and record. On the other side of the pipeline, all of this span data may be linked to any Maya object attribute, allowing for a plethora of mapping scenarios.

This project encompasses the creation of a pipeline for music analysis and visualization. This pipeline is comprised of two major parts. The first part of the pipeline is used to select and analyze piece of music and export the normalized data in a form that is compatible with the second stage. Starting from an .mp3 file, the user extracts information from the music through the use of an interactive Adobe Flash application. As I do not have much experience with music analysis or music in general, this was a challenging portion of the project. I designed the music analysis portion to be user interactive. This means that the analysis isn't solely done procedurally, and the user must provide some assistance to extract data from any particular piece of music. Using this method allows the data extraction to be much less time consuming and reduces the opportunity for error based on noisy or incomplete music data. I had done some basic work with AS3's compute spectrum functionality for my generative art class and have quite a bit of experience developing interfaces in Flash from websites and other course work, so I decided to use that as a base for my analysis tool.

The second part of this project's pipeline is the application of the data analysis in the creation of a 3D scene file that can be manipulated by the user to

render out the final visualization. Through the use of a customized GUI, users may set up some of the key parameters important to the visualization or simply randomize all of the variables and see what comes out. After the input phase is complete, the scripting takes control to create a scene that incorporates the user's choices into a dynamic scene file. At this point the user may change any aspect of the file they see fit before rendering. For example, a user can add a particle emitter, fluid, or hair system to the scene that would be influenced by the fields that the script has created. If the user wishes the final output of the pipeline may be changed and re-rendered multiple times. These renders can involve different object, materials, lighting, and camera angles. Making a final composite out of multiple renders can increase the overall impact of the piece.

2. PRIOR WORK

The idea of putting music and images or colors together has been around since the time of Aristotle [7]. We see one of the very first examples of music visualization in the form of mapping visual media to specific elements in music from Isaac Newton. In 1704, Newton wrote *Opticks* which proposed a distribution of different colors to different musical notes. Observing the natural correspondence between the seven prismatic rays and scale of musical notes, he paired them as follows: red to C, orange to D, yellow to E, green to F, blue to G, indigo to A, and violet to B (Figure 2.1) [8].

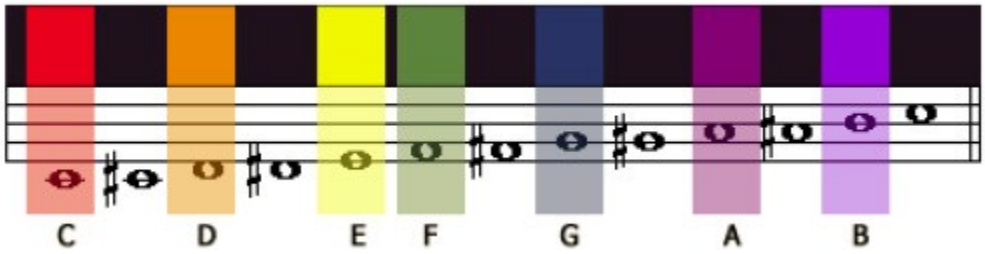

Figure 2.1: Maura McDonnell, Newton's color/note correspondences [8]

Newton's ideas were popular, and people like Louis-Bertrand Castel (1688–1757) created his own spin on these note assignments. Castel's scale featured a similar distribution of colors along the range of notes, but included 12 distinct

separations. Castel also believed that the color blue should be the representative for the C note, and modified his scale accordingly (Figure 2.2).

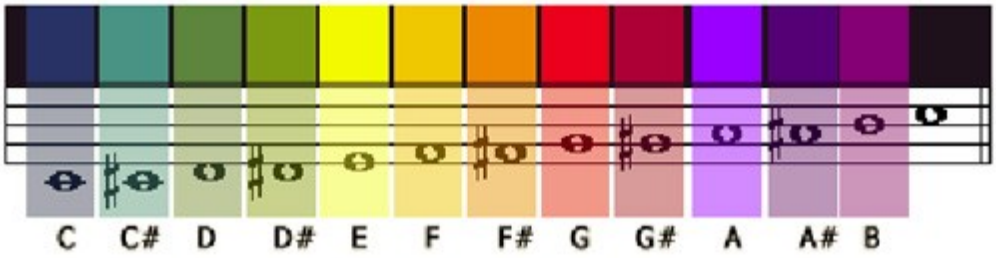

Figure 2.2: Maura McDonnell, Castel's revision to Newton's scale [8]

Similar to this approach, painters such as Wassily Kandinsky and Paul Klee used musical structure as an inspiration for some of their works. Kandinsky had a structural method of composition in which he used a rhythmic organization of different forms and colors to try to express emotions similar to the ones he heard in music [9]. Kandinsky is noted for saying, "Color is the keyboard, the eyes are the harmonies, the soul is the piano with many strings. The artist is the hand that plays, touching one key or another, to cause vibrations in the soul" [10]. Klee was interested in musical harmony that comes from polyphony and symbolized the beauty he heard there by replacing the sounds with blended, harmonious coloring schemes [8].

One of the first automated devices capable of creating visual music, the color organ (Figure 2.3), was made in 1877 by Bainbridge Bishop. Rather than painting his own impressions of music, he made a device that fit on top of an organ and projected different colored light onto a screen based on the key played. With this first model, Bishop also provided a control mechanism for the musician that could modify the light project, giving them some control over what to emphasize in the final visualization.

Figure 2.3: Maura McDonnell, A color organ [14]

In the early twentieth century, many men such as Walter Ruttmann, Leopold Survage, Len Lye, and Oskar Fischinger used the idea of animation to

represent various aspects of music. They realized that motion could be used to enhance the expressive power of their art. John and James Whitney pushed this idea into the next generation, investigating what made animation and music work together and closely synchronizing their abstract visuals to the music. Thomas Wilfred's "lumia" recitals combined music with a synchronized light show. Films, such as Disney's *Fantasia* in 1940, thrilled audiences and sparked new interest in the subject[11]. Studios like Disney and Warner Brothers also produced many of their cartoons to music. Extensive planning was required so as to ensure that the animation synced perfectly with the music to the extent that artists working on a particular animation were given a number of frames that they had to fit their animation into to match the beat. This of course, couldn't always work and in that case, the music might be changed slightly to accommodate the need for more time [12].

In 1968, John Whitney's film *Permutations* used imagery produced by a computer analysis of sound oscillations. He writes, "Thus, I believe that computer powers of calculation take us to the threshold of a world of stunning possibilities" [1]. The use of computer analysis opened up a new era for music visualization [13]. Many new artists began to put their own spin on visual music.

Beginning in the 1960's, across the world, in night clubs, concert halls, festivals, and stadiums a new kind of artist began to emerge. VJs, or video jockeys, began to find new and exciting ways to visualize music. At first, these

were simple analog light shows that accompanied the music, but as technology developed, Vjs began mixing in prerecorded video in the 1980's and computer animation in the 1990s. While the exact form varies from person to person, each VJ uses images and sounds to communicate message or provide entertainment. Some artists use computer generated visuals along with elements such as production lighting and special effect like fog machines and pyrotechnics to produce their unique music visualization. Other artists prefer to use camera footage or found footage that is mixed together by hand or algorithmically to correspond to the musical elements of the performance. Michael Faulkner, a video artist commented to say that, "VJs can take their place alongside the most adventurous graphic designers, illustrators, photographers, and Web designers, at the forefront of radical visual experimentation" [14].

Through the use of sophisticated algorithms, new classifications of color/music scales were suggested. Some new programs that were created could take a song as input and generate a unique color key or chromatic index of a song that would feature blending colors, much like the painters in the early days did [15]. In 2002, students of architecture began to transform music data into spatial representations, crating digital objects whose scale, density, and shape were determined by the music [7].

While some focused on trying to represent music's beauty in the form of hues and gradients, others, such as Philipp Kolhoff, were more focused on an

accurate visualization of a songs attributes relative to other songs. In 2006, he introduced a program that could generate procedural audio glyphs from a song based on multiple sampled attributes. In theory, a user could understand some basic facts about the song, such as tempo or loudness, simply by viewing the glyph [16].

Due to the increasing computational power of the machines used to create these visualizations, more impressive and complex output became possible [17]. Generally, these advances help both the overall quality of the images as well as the time it takes to generate them. Some focused heavily on audio data extraction such as meter and rhythm [18], while other leaned more toward visual complexity. Many real-time techniques became available. Some of these techniques involved the manipulation of images or video based on music data. In 2007, Allison Beane created a system for the real-time analysis of a live symphony orchestra performance and the conversion of the music data into video filtering techniques such as layering, filtering, and warping (Figure 2.4) [19]

.

Figure 2.4: Allison Beane, warp filter [19]

Using 3D art to visualize music has become an increasingly popular method. Many new artists have used programming languages such as OpenGL to bring their 3D animations to life [20]. One of the most notable early 3D visualizations was Pipe Dream (Figure 2.5) by Animusic [12] which featured auto choreographed animation based on a musical instrument digital interface or MIDI music file. Unlike normal music tracks that contain an audio signal, MIDI files contain flags and event message for instruments, making them easy to analyze and obtain accurate information from.

Figure 2.5: Eric Farrar, Pipe Dream Animusic [4]

In 2000, a pipeline that is roughly similar to the one used in this project was proposed, using separate analysis, 3D scene manipulation, and visualization modules. This project included use of fountain type of particle generators to visualize the music [21]. This type of visualization can be easily replicated using the pipeline from this project, along with a large number of new visualization styles. In 2008, Matthew Bain created a similar pipeline for the use of real time visualization of a symphony orchestra. Using Max/MSP, Bain did a simple audio analysis which was fed into Maya and used to generate real time effects (Figure 2.6) [9].

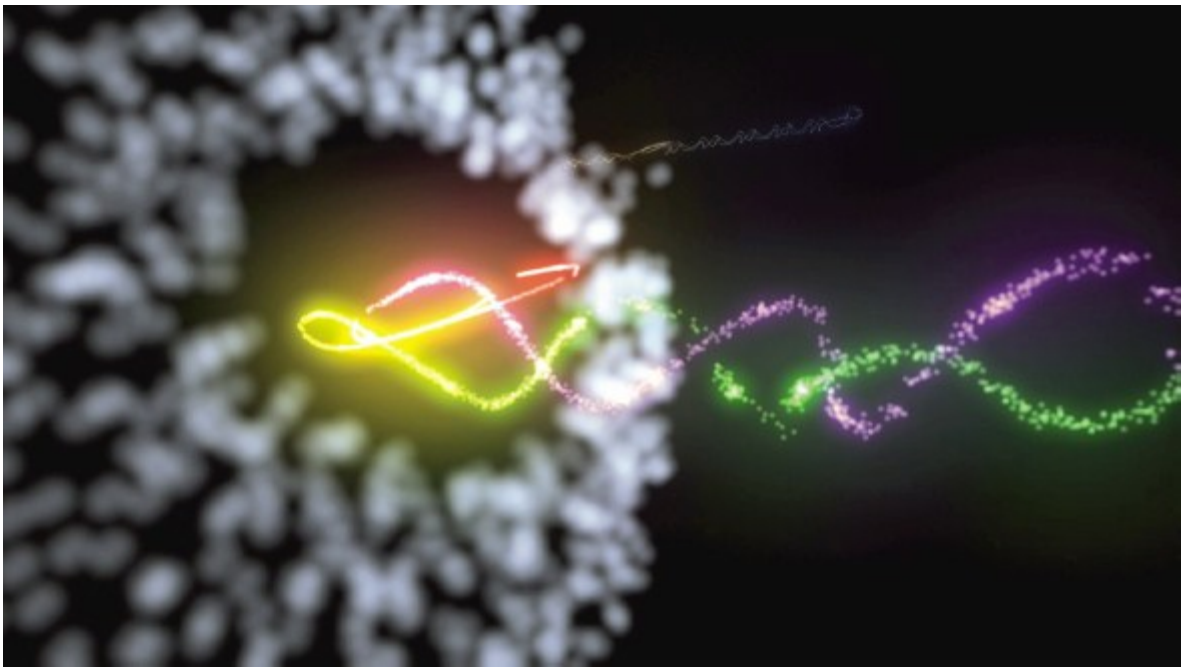

Figure 2.6: Matthew Bain, a particle emitter [9]

While I admire the difficulty involved in generating real time 3D visuals, such as some of the example above, I have not seen much from the field that appealed to me in the area of visual complexity. While many of these visualizations are entertaining and even interesting, I wanted to create something that incorporated some of the more complex areas of computer visualization. Using techniques like caustics, ray tracing, and final gathering, the output that can be obtained from my pipeline will have the potential to achieve a level of quality that I have not yet seen achieved by a real time system.

3. RESEARCH METHODOLOGY

3.1. Music Analysis

The pipeline for this project begins with the selection of an .mp3 type audio file by the user. I chose .mp3 because it has been one of the most common file types used to store music data since the mid-1990s. Another reason I chose this file type is that Adobe Flash is pre-programmed with the capability to load and read this particular type of file. When I began my initial research, I experimented with many different music analysis options. I considered developing my own analysis tool from the ground up in a language like C++, or making use of several other audio related languages such as Supercollider. While many of these options seemed seductive at first, offering unlimited customizability, I soon realized that the backtracking that I might have to do in terms of “reinventing the wheel” would most likely outweigh the benefits. Using a tool like Adobe Flash allowed me to skip worrying about ground that has already been covered several times, such as recognizing and decoding music files like the .mp3 type I have chosen to utilize. The file should follow all standards of header and information formatting of a standard .mp3 file. Through experimentation, I have found that a music file that is normalized to a maximum of -3 dB works best when doing the music analysis. There are several easy methods of converting a music file to such a normalized form. A popular method

would be through the use of a tool called Audacity[1] which is a free, open source music editing program available online.

I am very familiar with Adobe Flash and have done several projects using the FFT analysis function provided by AS3. For this project, I wanted to take the collection of sound data to a level of complexity that I had not previously explored. I decided to implement a beat detection algorithm that is based on an AS3 library that I found online[2]. While I was initially impressed with the capabilities of the library, as I dug deeper I found that several of the default interface options could be improved to give the user more capabilities. I also discovered a few errors in the original code, which were hindering the proper display and recording of spectrum data. Specifically, the method used to access data from the spectrum array after analysis was flawed, skipping over some values.

3.1.1. Interface Design

I decided to add several new functions to the original library. The first and most important is the ability to monitor and record frequency ranges. The user may specify a frequency or a range of frequencies inside of the music analysis application and create a specific name for each one. To do so, the user may manually enter start and end bounds in to the interface, or simply click on two

[1] Audacity is available at http://audacity.sourceforge.net
[2] The beat detection library was programmed by julapy and is available at http://julapy.com/beatdetection/BeatDetection.zip

points along the spectrum display. After the start and end points are determined, the user may enter a name for the specific range and then press the create button to initialize the range. Once established, the frequency monitor appears as a slightly transparent, blue strip across the frequency spectrum, with the end values clearly identified and the name in the center. Each band will collect information about the frequency range specified. For multiple frequencies, the monitor records an average energy value over all of the frequencies included. The user may specify an unlimited number of these bands, although the interface may get slightly crowded depending on the width of the bands specified and the number added. Using these frequency monitors, the user can pick out interesting frequencies in a piece of music. For example, a piece that contains only flute music might not have many notes that fall in the lower bass frequency range. Using frequency monitors, a user can easily isolate and monitor the mid and upper level frequencies of the flute, while leaving the base frequencies unobserved. Down the line, this improves the pipeline by simplifying the number of visualization options to only the ones which show interesting activity.

The beat detection portion of the application involves detecting energy deviations in the spectrum. In general, the term “beat” may mean one of several things, one of the most common being a measured rhythm to which all of the instruments in a piece of music play. In this case, I am using the term beat detection to refer to the detection of distinct differences in energy in a part of the spectrum as an individual sound event. Using this detection method, it is

possible to separate out beats form any portion of the spectrum. The user may choose to focus on the top of the spectrum to pick out the beats of a cymbal, or try to isolate another type of instrument that is clearly delineated on the spectrum.

Using an adjustable bounding line, the user may select a frequency range to detect beats. The user is able to adjust the position and size of the range bar using the left and right arrow keys and the 1 and 2 keys respectively. There is also a text display in the top left of the interface that shows specific range and threshold data. This part of the pipeline relies on the user's ability to isolate frequencies in which beat information is present. This feature allows for accurate detection of beats in from any part of the spectrum. For example, the beat in one song may be the rhythmic tapping of a high-hat cymbal, while another could be a thrumming baseline at the other end of the spectrum. Using an adjustable beat detection range, the user may easily recognize the area of the spectrum associated with the beat and isolate it. A beat is detected every time the instant sound energy is a certain threshold over the average sound energy [22]. The average energy is simply an average of the last ten sampled energies, and is used to provide a short history for the instant energy to be compared against. In the interface, the instant sound energy is represented by a red line that spans the samples frequency range, while the average energy is represented by a white line. As the music plays, these sound lines react to show the amount of sound energy within their spans by moving up and down (higher is more

energy).Using the up and down arrow keys, the user can adjust the beat detection threshold, or the minimum amount of difference between the instant and average energy that needs to exist for a beat to be recognized. When a beat is recognized, a green circle pulses in the beat portion of the interface, indicating that a beat was detected and the magnitude of the beat. The magnitude, in this case, is defined as the amount that the instant energy difference surpassed the user defined threshold.

For the final interface design (Figure 3.1), I included a large scale spectrum display that is 1024 pixels across. I implemented it this way so that the user can easily identify areas of activity on the band and create frequency monitors more accurately. There is a simple input box for the user to input a filename and a play button to begin the music. The user may type the name and extension of a music file that resides in the same directory as the application, or type the files full path. To perform an analysis, the user plays the music and watches the spectrum to determine the areas where they would like frequency monitors and the range over which the beat detection algorithm should span. After the user has set as many ranges as they like and adjusted any other settings, such as beat threshold, they may click the record button which begins the analysis.

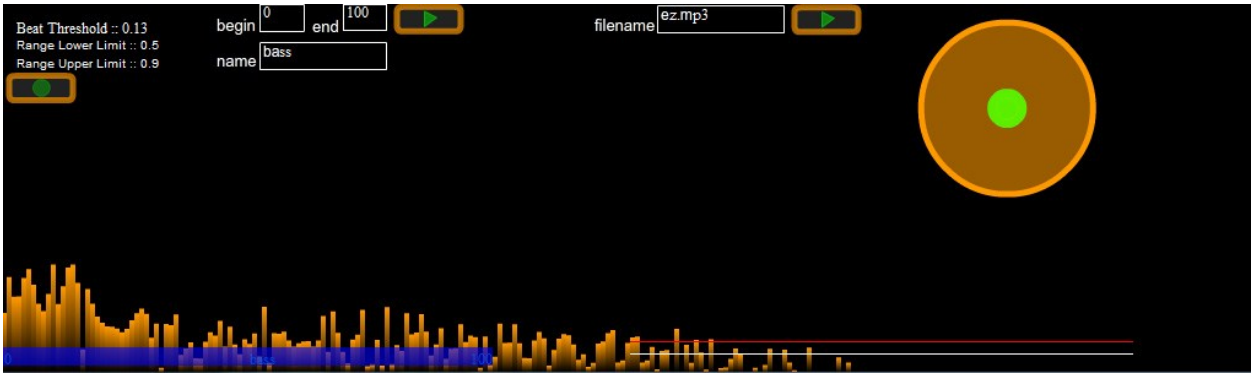

Figure 3.1: Final interface design for music analysis

3.1.2. Data Analysis and Manipulation

After the user begins the analysis process, no additional user input is required. Every twenty milliseconds beat and frequency data is stored into expanding arrays. For beat data, if there has been no beat detected in the time window, then the program simply records a zero value, while if a beat was detected, then the magnitude of the beat is stored. For frequency ranges, a simple average of the energy data across the range is recorded. After the song has completed, the data is normalized so that all the values fall between 0 and 1 with three decimal points of precision. While this function does not maintain a ratio of the values of one frequency span's energy when compared with another, I believe that this equalization can lead to a more interesting final visualization. If the user wishes, they may easily adjust the overall energy values of any range within the visualization portion of the pipeline.

I have written a couple of data manipulation functions that may be used to make the data more interesting to visualize. The first is a function that increases data variance from the mean. So, for example, suppose there is a frequency range had an average value of .5, with all values falling within a .1 range around the mean. Through the use of this function, the user may specify the amount of variance desired and the function will spread the data around the mean (Figure 3.2). Obviously, this could lead to the introduction of noise into the data if overused (since the values cap at 0 and 1), and so this function is not used by default within the music analysis tool. A second function that I have included is an invert data function. Visualizations are, in general, more interesting when they focus on dynamic points of change within a set of data. While this is the behavior that is present by using the straight data from a fairly inactive set, data from a very active frequency band tends to become somewhat homogenous with a few outlier values. Using the invert function, users may in essence swap the outliers with the average value (Figure 3.3). This means that the outlying values become the drivers of action in the visualization, rather than a constant homogenous input. While I believe that this function will be useful to advanced users, I want the interface to be easy to pick up and use by anyone without extensive research into the tool. Therefore, I have disabled the invert in the default version of the application.

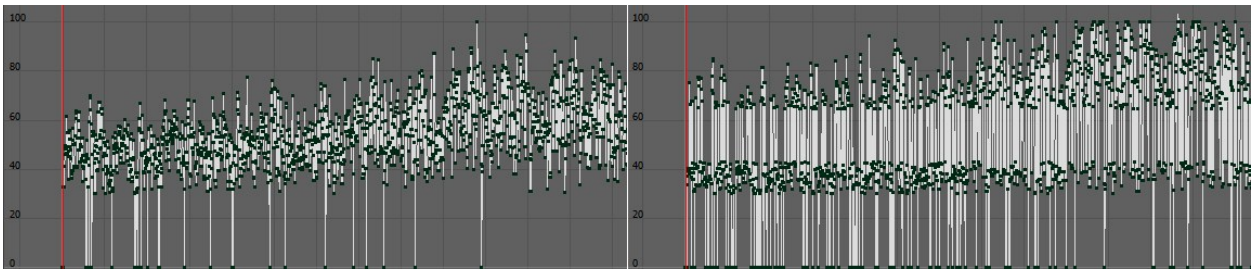

Figure 3.2: Frequency data before and after variance function

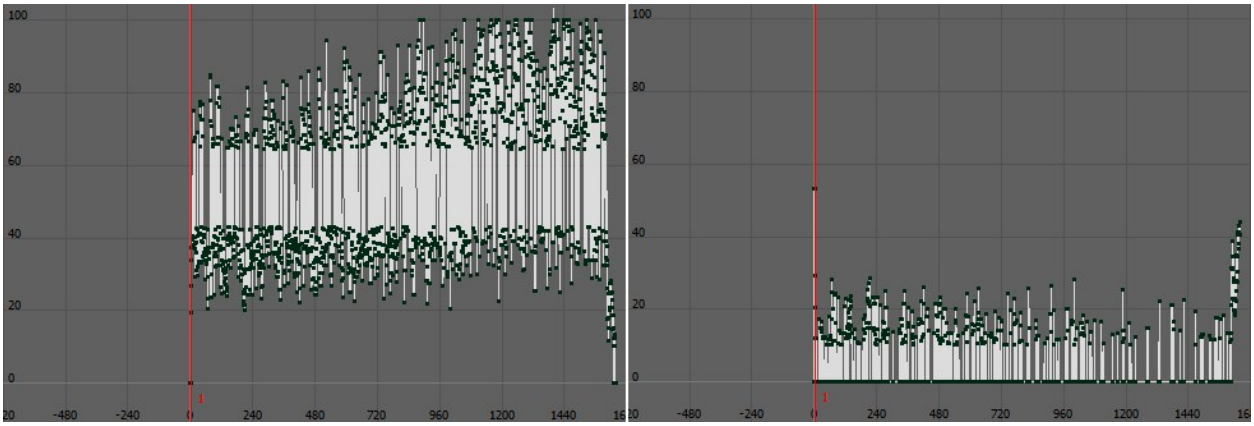

Figure 3.3: Frequency data before and after invert function

After the data collection and normalization have finished, the application writes a basic text file to the user's desktop. This file includes some metadata at the top, namely the number of user defined frequency ranges and the number of samples taken over the entire analysis session as well as the names that the user assigned to each frequency range. The data section of the file is formatted as a column list, the first column containing beat data followed by space

separated columns with values for each of the frequency bands. Each new line in the file represents a detection window time space in the algorithm.

3.2. Visualization

3.2.1. Interface

To start the visualization portion of the pipeline, the user will open up Maya and run the custom interface script that I have coded in Python. I chose Python over MEL, because of Python's advanced capabilities, data structures, and easy-to-use syntax. The Python script may be executed using the source and reload commands in the script editor, or the user may attach this functionality to a button on any of Maya's shelf interfaces. When the GUI first initiates, there is no other functionality than the ability to load a music data file. The user may select a file using a basic file chooser window (Figure 3.4).

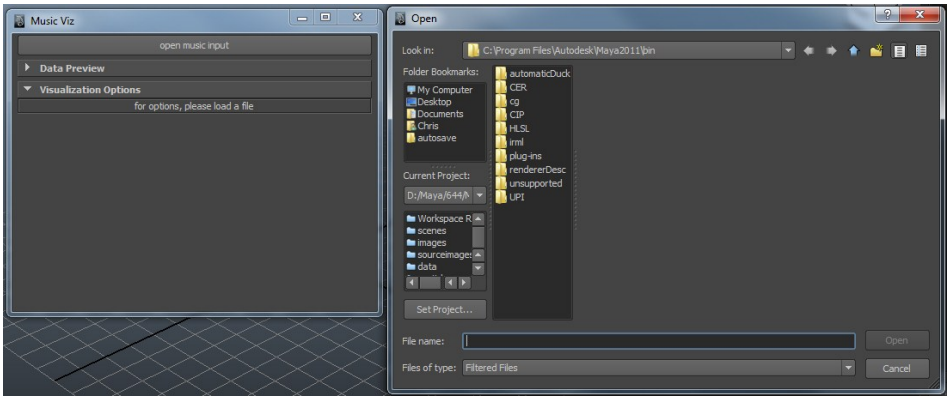

Figure 3.4: Initial GUI state and file selection window

Once a file is selected, the interface automatically updates to include specialized options for visualization of beat data, as well as any frequency range data the user create during the music analysis section. Using the visualization portion interface, the user may select one of four different types of fields whose magnitude will be mapped to the music data (Figure 3.5).

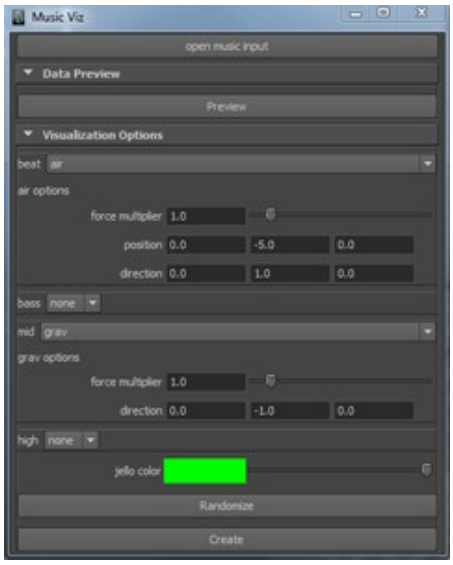

Figure 3.5: GUI options

These fields may be used to interact with many types of dynamic objects in Maya, including dynamic bodies, particles, hair, and cloth. The user may select from a gravity field, an air field, a turbulence field, and a vortex field. A gravity field is one of the most basic types of fields, simply exerting a certain force in a

given direction (usually down). An air field is much like a gravity field, with the exceptions that the emitter's position is relevant and an attenuation factor which allows for the field to act on different objects or areas of an object differently. This means that an air field may also cause spinning in the objects that it pushes against. A turbulence field is a more complex type of field that generates noisy, turbulent forces around it. This means that a turbulence field might push things on top of it away, while attracting objects to the side or spinning objects directly below. Given a large amount of noise, a turbulence field may produce various and unexpected resulting forces. A vortex field is a special type of field that was created to mimic forces generated by tornadoes. In general, vortex fields push objects away, while spinning them around a central axis. When selecting any of these types of fields, the user interface will display some important, field-specific options. Using these options, the user may be able to get an idea of the kind of visualization that will be created. For example, if the user selects a gravity field for the beat data, then they may assume that the objects in the scene will be pulled in a certain direction with each beat. This is not a common interface function in a Maya GUI. In order to automatically load and unload options based on a specific field, each frequency range is placed inside of its own frame layout. When a user selects a specific field, a function removes all children from the layout and repopulates it with the correct interface controls. Aside from field and field options dialogues, the visualization interface also includes a color selection box. By default, this visualization module creates a soft body object in Maya that

shares many visual characteristic with jelly or gelatin. Using this color picker, a user may specify a color for the jelly object. The visualization tab also includes a randomize button. Pressing this button will assign a random value to any editable fields in the interface, including the color box. While the values assigned are random, they have upper and lower bounds within reasonable limits. An air field, for example, will not have its initial position moved outside of the collision walls that the interface generates as part of the default visualization.

While the user may be able to get a feel for what the final data for a frequency or beat array will look like by watching the music analysis tool, many times a user will have no idea what the data set might look like. For this reason, I have implemented a data preview function. After loading music data, a user may click this button under the data preview tab to instantly see the basic data. This function automatically creates an empty group (a type of null node, which has no effect on rendering time) and adds custom attributes according to the beat and frequency data in the file. The values will appear in Maya graph editor as a continuous line with each data point represented as a key frame (Figure 3.6).

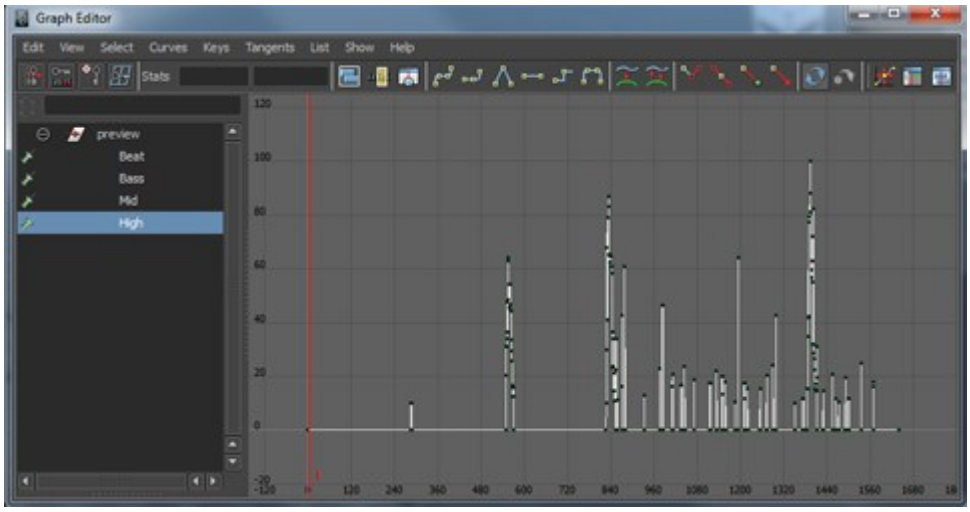

Figure 3.6: Data preview in Maya's graph editor

Using this function, a user may be able to recognize the best use of a particular data track. For example, if a data track has sparse and infrequent peaks, then the user might decide to map it to a strong field that will be easily noticeable, so that the effects of the data are obvious in the final visualization.

After the user is satisfied with the visualization options, they will then click the create button. The create function automatically creates a default scene containing a collision-enabled room, a soft body, jellylike object, and a light (Figure 3.7). While this alone can provide an interesting visualization, the user will most likely want to add objects or attributes to the scene after the initial work done by the algorithm. Shaders are created for every object in the scene, some of them including complex shading networks that involve ray-tracing. The

algorithm automatically creates a new default rendering camera, set directly in front if the scene. Render settings are automatically adjusted to high quality, with caustics and final gathering enabled. The initial resolution is set to 640 by 480 pixels, but this, like all other attributes setup by the function, can be changed by the user before final rendering. All the shadow and caustic attributes are added to the light and the quality levels are increased beyond the bare minimum default level. The timeline is automatically lengthened to span the course of the entire song and the final render frame limits are changed accordingly. Additionally, depending on the frequency ranges the user has chosen to visualize, field generators will be created. Each generator's magnitude attribute will be procedurally mapped to the data from the music analysis. While untouched this would lead to a dense cluster of key frames spanning the whole timeline, duplicate values are weeded out using the simplify curves functionality available in Maya. This simplifies the process for adding additional user animations or tweaking data values if so desired. Each field is named using the user-chosen name for the frequency monitor set up in the data collection phase as a prefix, followed by the field type.

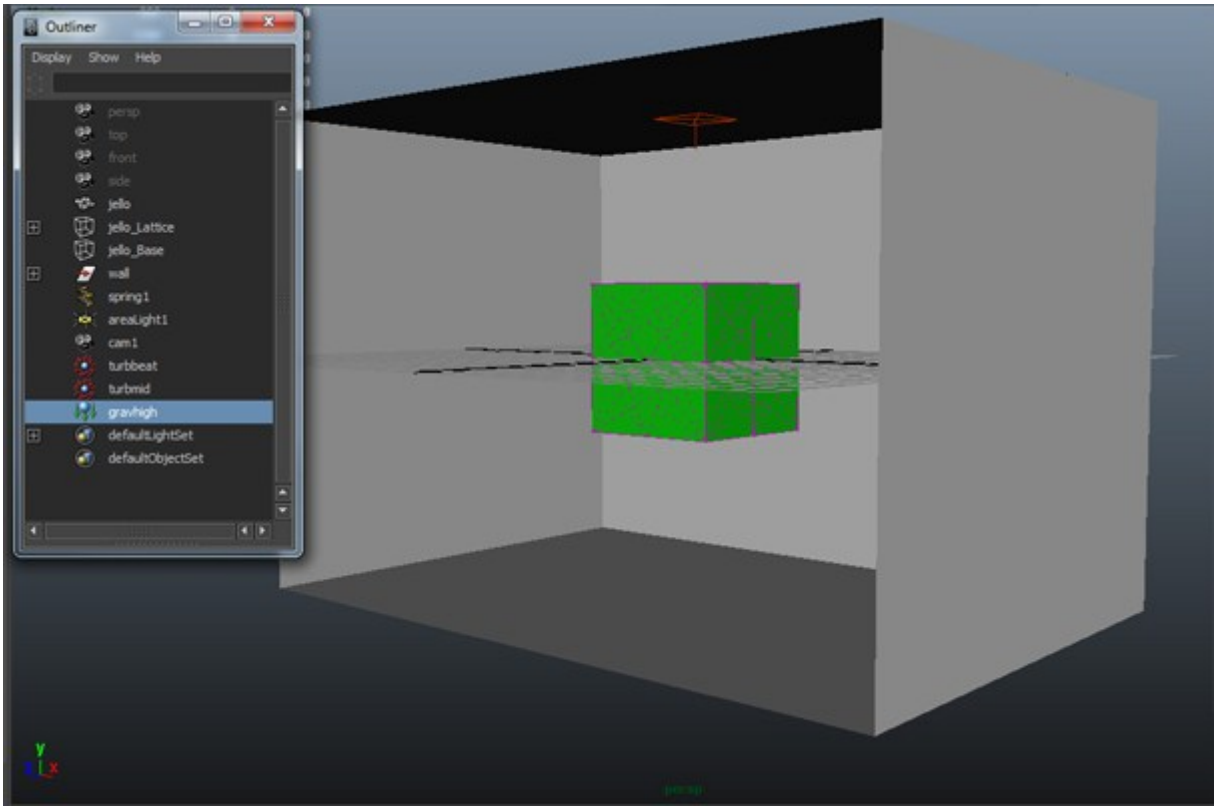

Figure 3.7: Basic scene created by the default visualization function

3.2.2. Manipulation

At this point in the pipeline, the user may play the animation to observe the effects of the fields on the soft body object. All aspects of the scene may be manipulated from this point. Options such as light position, number of lights, scene size, and field placement may be modified, along with a myriad of other options. User may move the camera anywhere inside the scene and adjust several camera attributes such as focal length and angle of view. Using the

graph editor, they may easily make changes to the overall magnitude keys. At this point, the user can decide how pronounced each aspect of the music analysis will be. For example, the user may decide that they want the bass frequencies of a particular data set to be present, but subtle. To accomplish this, the user can decrease all of the values for the bass data, so as to de-emphasize its influence in the final visualization. An interesting application of this modification is to center the mean of the data at 0 on the Y axis of the magnitude graph. Providing all of the values aren't exactly on the mean, this method can produce interesting actions. For example, if such frequency data was mapped to a gravity field, then the highest points could push the object upward, while the lowest points would pull in down. As mentioned before, users may add custom animation to the scene, and change any key values as they see fit.

Users may create any number of dynamic objects to interact with fields create by the visualization function. Instead of using the default bouncing jelly cube, the user will usually choose to replace it with their own objects, such as a particle emitter at the center of the scene. After a simple connection between the fields and new emitter, the particles from the scene would be moved by the fields according to the music analysis data. Additionally, users may choose to duplicate certain fields to make symmetrical effects for particle or fluid animations.

Aside from the field-based, default method of visualization provided by the main creation function, this pipeline allows for the creation of any number of visual variations using the input music data. Using the preview function, the user may obtain a node that contains all of the unaltered information from the music visualization portion. Using, the expression editor or any of several other methods in Maya, these key values may be assigned to virtually any attribute of any object in the Maya domain. Almost every attribute of every object may be keyed. Using simple attributes like radius, users may create a sphere whose size pulses to the music. Users can potentially use the sound data to control the lights in a rock concert scene, or even use beat data to synchronize the digital audience's animations. Data can be used to choreograph animation, such as dancing or exercises animation for digital characters. The potential uses of the data are nearly limitless, as it can be plugged into any type of function in Maya using MEL or Python.

3.2.3. Rendering

After the user has tweaked any setting that they see fit, they may use the render pipeline inside of Maya to generate final animation frames. Using the Mental Ray rendering interface, users may select any resolution they wish to render at. The user may also set up multiple cameras to render from. Additionally, they may enable or disable rendering features such as motion blur, final gathering, ray-trace accuracy, reflection and refraction limits, shadow

options, and many more high quality features. After selecting a data type for the final images, the final output of the entire pipeline is an image sequence that may be compiled standalone, or composited with other footage. The frames should be interpreted as a 24 frames-per-second video clip for use in any modern editing software package. In order to change the frames-per-second for an animation, the user will have to manually edit the Python script for the GUI.

4. RESULTS

4.1. Summary

When laid over the music used from the original analysis, the visuals share an obvious correspondence. This goal is essential to the effectiveness of any music visualization. The imposed time window used in the music analysis phase is included in the procedural animation script, ensuring the relationship stays accurate. The first stated goal of this project was to have a stable music visualization pipeline. Using the music analysis application, coupled with the GUI in Maya, anyone who is familiar with basic digital sound concepts and the Maya interface should be able to achieve some form of representational visualization. Though skill in Maya is a plus and needed to produce some of the more potentially interesting visualizations, the default visualization created by the GUI provides an automatic mapping that requires little real understanding of the software package. It is entirely expected that most users will want to implement their own visualizations using the analysis data and, for this purpose, Maya knowledge will be invaluable. Another goal for the project was that the visualizations attain a level of visual complexity that most real time systems would have trouble matching. The first part of achieving this complexity comes from allowing the user to select any number of frequency bands to analyze during the analysis phase. If they desire, the user may monitor any number of frequency ranges, allowing for a more intricate final visualization. The second

way visual complexity is achieved is by making available computational power that real-time systems cannot match. Using Metal Ray, images that come out of this pipeline can feature complex graphical effects such as motion blurring or caustics (Figure 4.1). Due to the non-real-time nature of the pipeline, users may generate the final images at any resolution. This process also allows for additional complexity to be added in the form of customized user animations or objects after the initial scene generation. In general, all of the goals that I set out to accomplish at the beginning of the project have been achieved.

There are several potential uses for the final output images. Frames from a simulation may be rendered out used as a series of still art pieces. Due to the 3D nature of the visualization portion of the pipeline, it is even feasible that the data could be used to construct a series of digital 3D objects which could be printed as a sculpture. The most common use will probably be rendering out animation. Users might choose to render out multiple kinds of animations based on the same musical data and compile them together.

Figure 4.1: An example of high quality output

4.2. Examples

In order to demonstrate the flexibility of the pipeline, I have created three different visualizations using Maya and the data from the music analysis portion. As I have mentioned, an experienced Maya user may find an almost limitless number of ways to visualize a certain data set, and I hope that these examples might help to demonstrate the range of visualizations that can be created.

The first visualization that I implemented resembles a classic analog spectrum display that can be seen on many sound boards or music players. In the music analysis portion of the pipeline, I simply set seven different frequency monitors spanning the entire length of the spectrum (Figure 4.2).

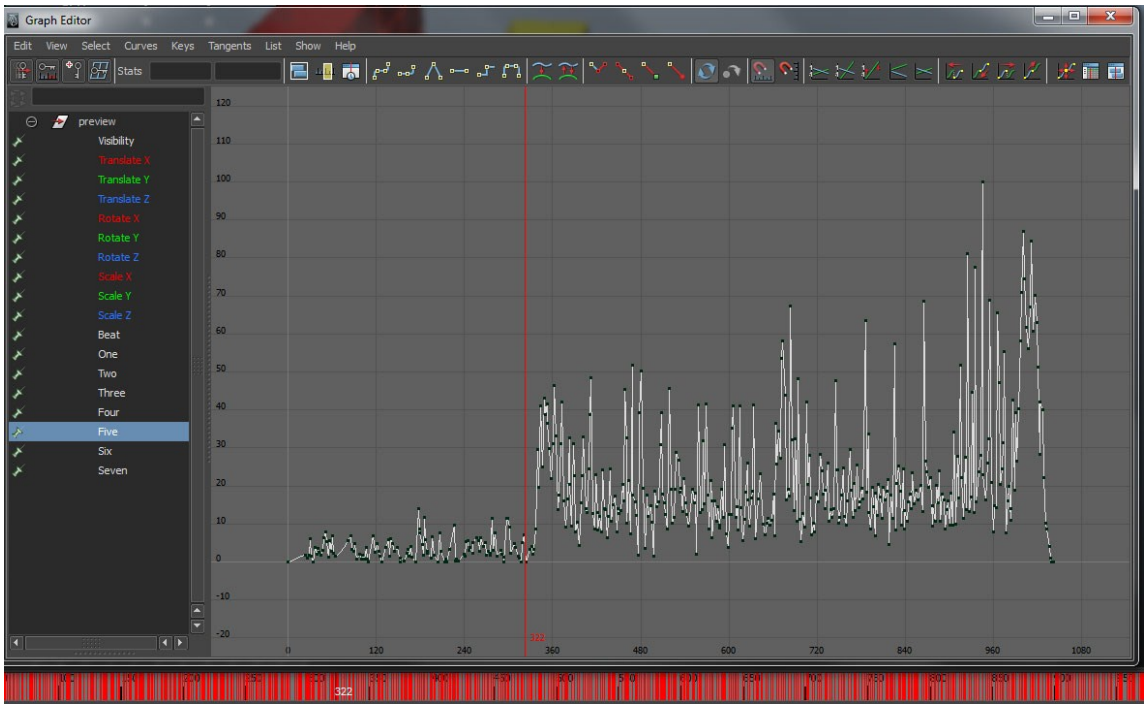

Figure 4.2: The preview node in Maya containing the seven frequency sets

In Maya, I created a complex custom hierarchy of cubes divided into separate columns. I gave each column a color (spanning the color spectrum) and then, using the expression editor, mapped the data from each of the seven

channels to the height value of each of the columns. After the initial mapping, I added some extra animations to the hierarchy that I had created earlier (Figure 4.3).

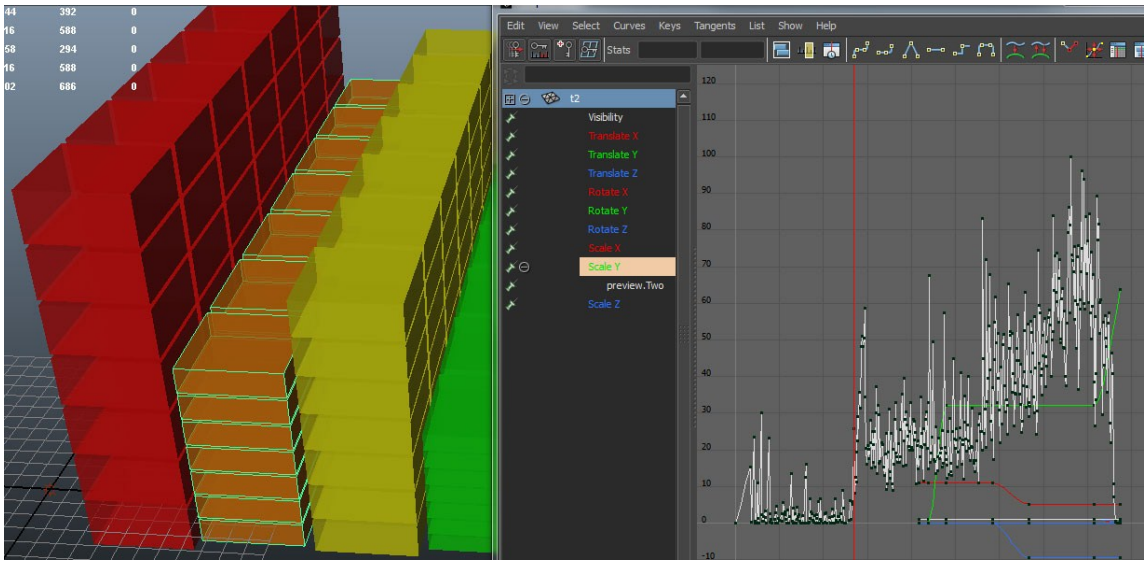

Figure 4.3: Custom animations added to column hierarchies

The final animation included a custom-animated camera that zooms and pans around the bars as they fluctuate, morph, spread out (Figure 4.4), and then finally shrink to nothingness. In the render settings, I made the boxes semi-transparent, added a glow effect, and boosted the overall quality to include some basic reflections and refractions.

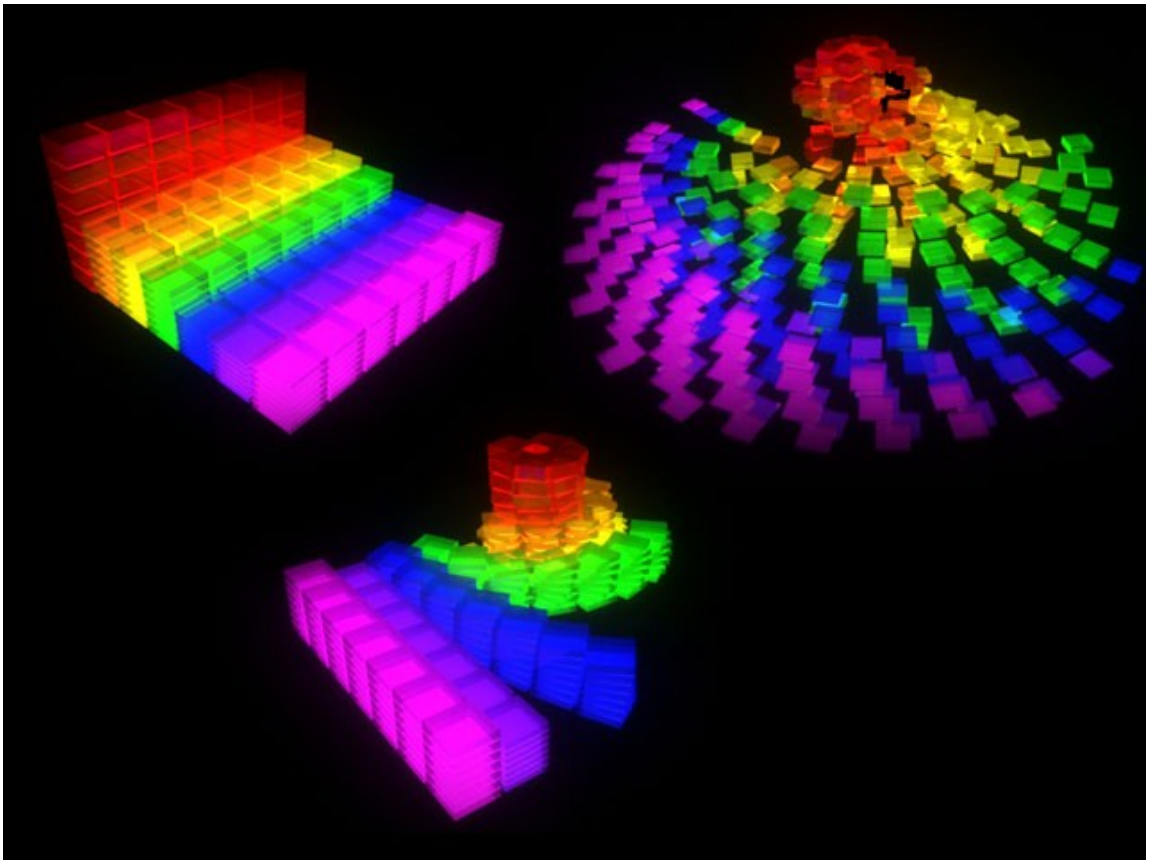

Figure 4.4: Three frames from the custom visualization example

This visualization demonstrates that the user may create an animation from scratch and easily plug in the data from the music analysis phase.

The second visualization that I created involves the use of the Maya fur system. I began by creating a custom fur definition in Maya and tweaking many of the default variables. After some experimentation, I decided to map three unique fur attributes including fur length, tip curl, and scraggle. Fur length is obviously the length of the fur on an object, the tip curl attribute is used to bend

or curl the end of fur and the scraggle attribute controls how bendy and crooked the fur appears. Using some basic arithmetic and the expression editor, I assigned the length to the mid-range frequency monitor, scraggle to the hi-frequency data, and tip curl to the beat (Figure 4.5).

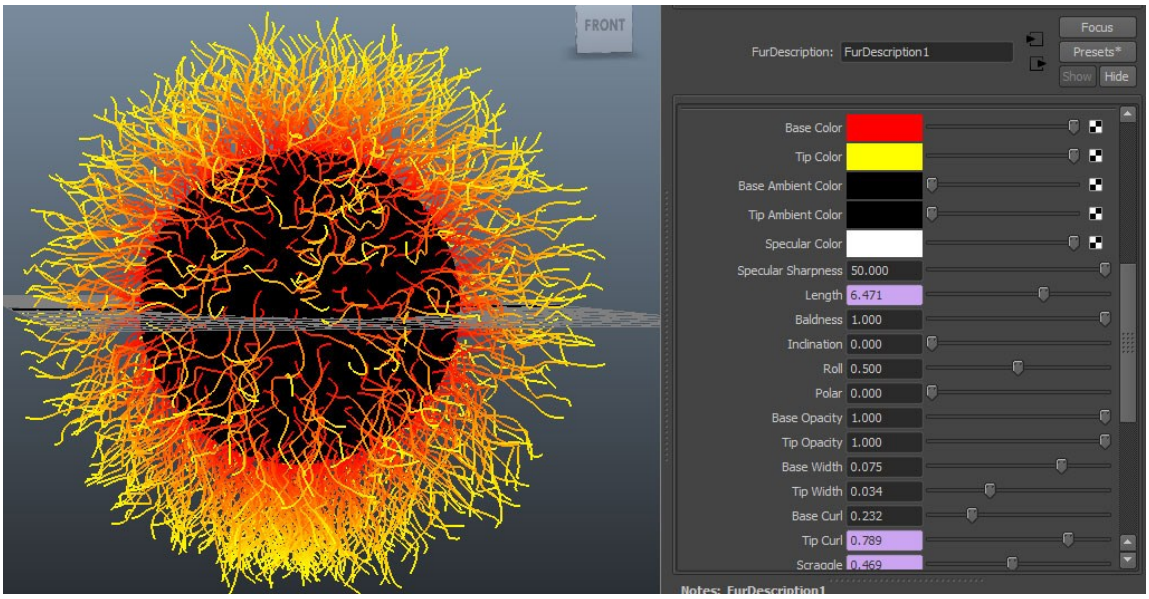

Figure 4.5: Maya fur with a high scraggle value applied to a sphere

In this example, I took the time to manually edit the animation keys, smoothing out specific regions in the data over a certain portion of the song to decrease the fur activity during that window (Figure 4.6). In the final animation, the fur is applied to a sphere and the fur grows and pulses along with the musical cues. After all of the fur attributes have been set in the fur description, it

is possible for me to apply it to any type of NURBS or polygon object within Maya (Figure 4.7). This visualization demonstrates that you can plug in the data from the preview function into any kind of specialized Maya object or system (Figure 4.8).

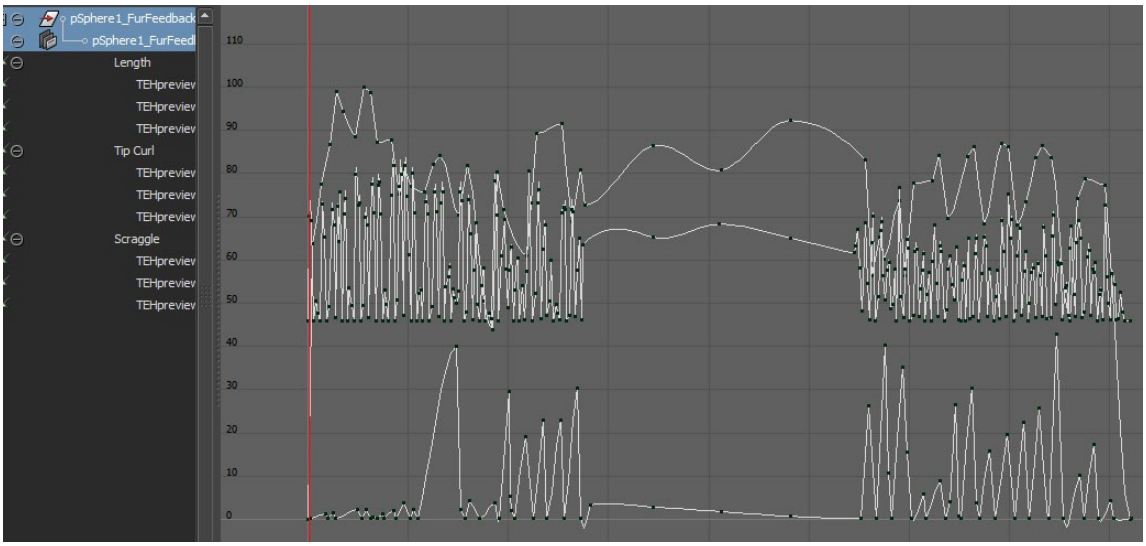

Figure 4.6: Key frame data for the fur visualization after manipulation

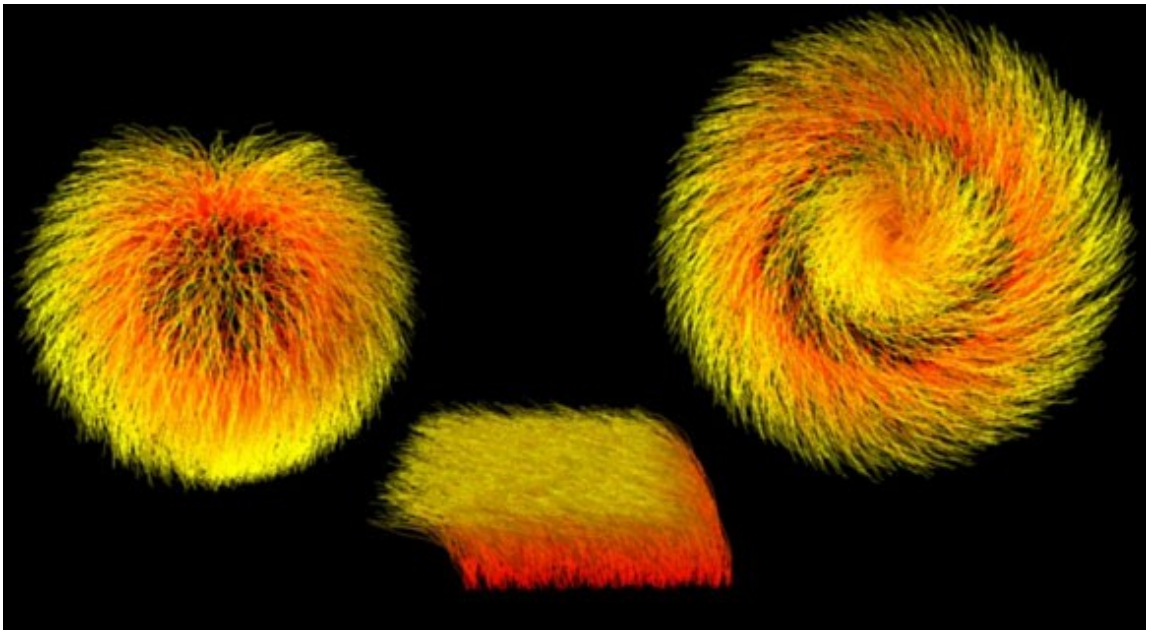

Figure 4.7: The custom fur description applied to three different objects

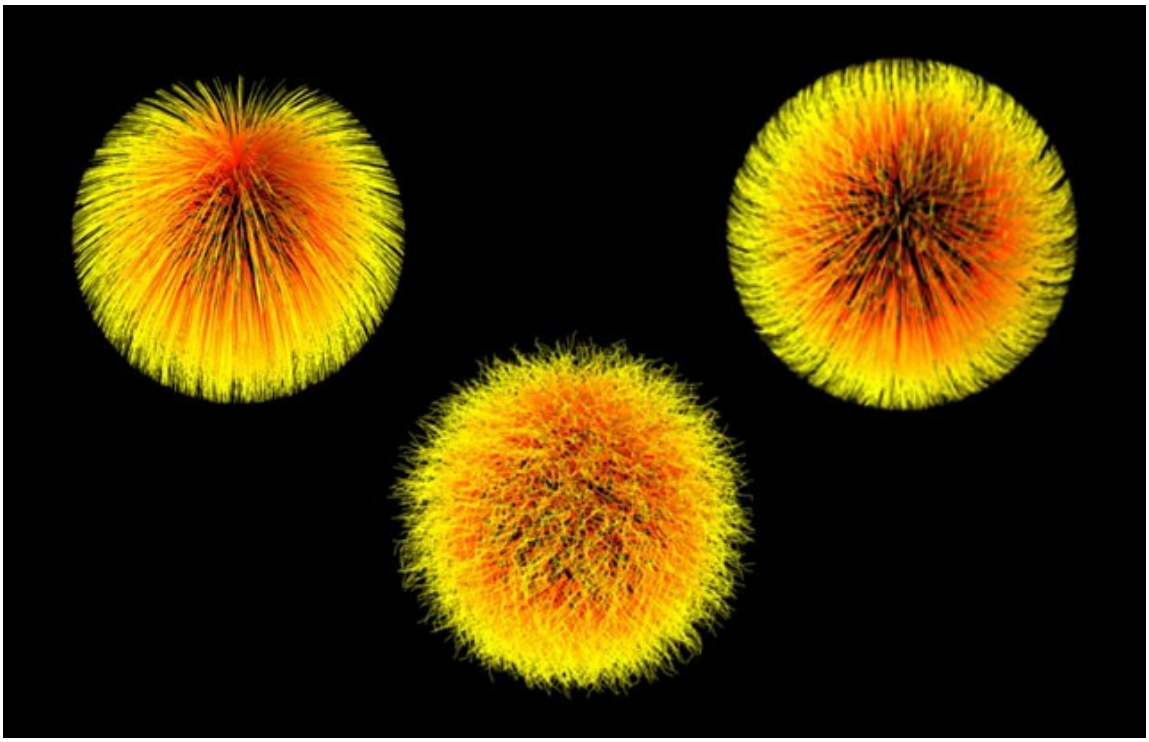

Figure 4.8: Three frames from the final fur visualization

The final visualization example that I made included the automatic manipulation of a character rig using sound data. I downloaded a premade rig called Liam from creativecrash.com[3] (Figure 4.9).

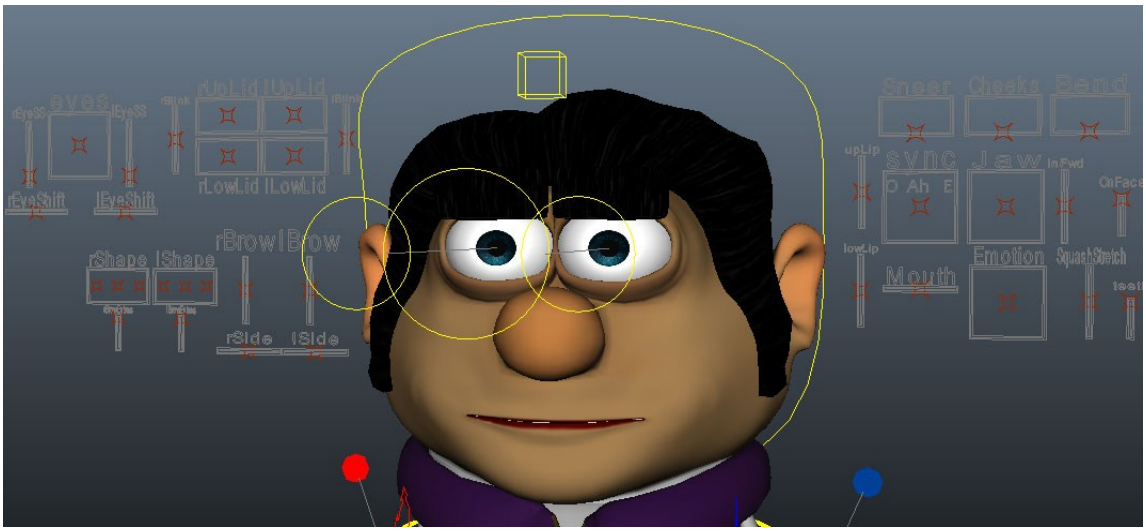

Figure 4.9: The “Liam” facial rig

After familiarizing myself with some of the basic rig functionality, I decided to map data from some mid-range frequencies to the eyes. The music input that I selected for this visualization had a strong melody that fell within these frequencies, and the overall effect was that the character’s eye closed and opened slightly along with the music. I then decided to map the beat data to a slight head rotation, making the character’s head bob along with the song.

[3] Liam rig available at http://www.creativecrash.com/maya/downloads/character-rigs/c/liam

Finally, I created a custom mapping that made the character smile when a certain high-range frequency was hit (Figure 4.10). I also created a custom attribute on the rig to switch the smile to either side of the character's face.

Figure 4.10: Rig controls mapped to music data

This visualization demonstrates that the user may integrate their own sound data into objects or system that were built by others (Figure 4.11). All of these visualizations are examples of basic applications of the sound data to

different types of Maya objects. With more time, it would be possible to create more robust, higher quality visualizations.

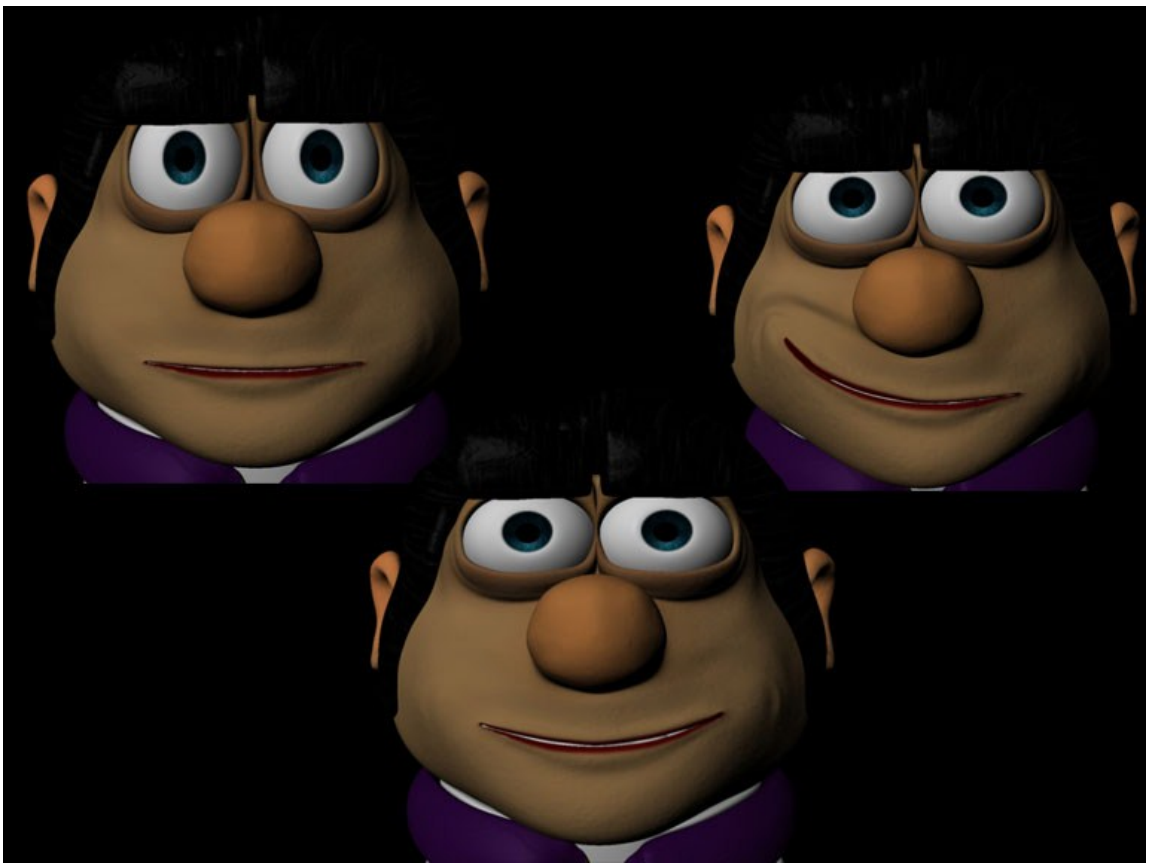

Figure 4.11: Three final frames from the “Liam” visualization

5. SUMMARY AND FUTURE WORK

Throughout the process of completing the project, there were several things that I realized could be added in order to improve the thoroughness of the music analysis, the quality of the final animation, or the pipeline in general. While I had time to add some of these improvements as I worked, there are a few that I still have to suggest to others interested in the research. The first thing that I would like to add to the pipeline is support for the .wav audio file type. This file type is one of the most popular kinds and adding support would broaden the usability of the project, or at least save the user the time it takes to convert from an .mp3. Also, if this idea were fleshed out in the pipeline, the song could be automatically appended to the timeline in Maya when the scene is generated and would play back with the animation. This could serve to make the process of tweaking animation and adding custom elements more quick and effective.

From a music analysis standpoint, one feature that I would have liked to add is support for separate left and right audio channel analysis. It is possible to get separate channel data in Flash and I believe that, if integrated properly, having differentiated left and right channel data could enhance the final visualization. Using this data, the user could more easily isolate specific instruments or cues in the music data, and also do some potentially interesting things in Maya. An example of a simple application of this data to the default visualization would be to assign competing gravity fields on each side of an

environment, and let the magnitude readings from each channel push or pull objects in opposite directions. Another feature of the music analysis which I worked on, but was unable to achieve reliable results from is a beat checker. After the initial analysis of the sound is complete, it would be helpful to have a reliable function to analyze the data set and extrapolate a tempo with which to check for beats that might have been missed during the recording process. This algorithm could analyze the intervals between detected beats to establish a general time interval with which to check for missed beats. A recording of the frequency data for the beat range would be easy to attain using the frequency monitor class that I have created. Backtracking through this data, the algorithm could look for expected beats and see if there was any kind of significant increase in energy at that time. Obtaining correct tempo information could improve the final data quality and help the visualization sync more closely with the audio. The final addition I would make to the analysis side of the pipeline would be better integration of the data manipulation functions. While these functions can be useful for certain types of data sets, they aren't immediately available to the user via the GUI. It might be helpful to implement per-monitor options for each new frequency band that the user initializes.

In the visualization side of the pipeline, I am fairly happy overall with the GUI functionality for creating the default visualization. However, one of the best features of the pipeline, the ability to key any Maya attribute to the music data, isn't obviously apparent when you load the interface. Users must open the

expressions editor or copy and paste frames from an object to correctly map animation. While this isn't a particularly daunting task, it might be helpful to provide an easy expression box extension to the GUI, allowing users to type in specific object attributes next to the frequency or beat data and automatically mapping the data.

REFERENCES

[1] J. Whitney, *Digital Harmony: on the Complementarity of Music and Visual Art.* Peterborough, NH: Byte Books, 1980.

[2] P. Galanter, "What is generative art? Complexity theory as a context for art theory," *Proc. 6th Generative Art Conference*, Milan, Italy, 2003.

[3] H. Situngkir. (2005, What is the relatedness of mathematics and art and why we should care? *BFI Working Paper Series WPK2005,* pp. 1-8. Available: http://cogprints.org/4679/

[4] L. Pocock-Williams, "Toward the automatic generation of visual music," *Leonardo (U.K.),* vol. 25 pp. 29-36, 1992.

[5] S. M. Smith and G. N. Williams, "A visualization of music," *Proc. IEEE Visualization '97*, Phoenix, AZ, pp. 499-503, 1997.

[6] E. Isaacson, "What you see is what you get: on visualizing music," *Proc. 6th International Conference on Music Information Retrieval*, London, pp. 389–395, 2005.

[7] M. Radojevic and R. Turner, "Spatial forms generated by music: The case study," *Proc. 5th Generative Art Conference*, Milan, Italy, pp. 28.1-28.10, 2002.

[8] M. McDonnell. (2007, 10 Nov. 2010). *Visual Music*. Available: http://www.soundingvisual.com/visualmusic/VisualMusicEssay.pdf

[9] M. Bain, "Real time music visualization: A study in the visual extension of music," Thesis, Ohio State University, Columbus, 2008.

[10] M. McDonnell. (2003, 10 Nov. 2010). *Notes for Lecture on Visual Music*. Available: http://www.soundingvisual.com/visualmusic/visualmusic2003_2004.pdf

[11] M. Clague, "Playing in 'toon: Walt Disney's "Fantasia" (1940) and the imagineering of classical music," *American Music,* vol. 22, pp. 91-109, 2004.

[12] E. Farrar. (2003, 10 Nov. 2010). *A Method for Mapping Expressive Qualities of Music to Expressive Qualities of Animation*. Available: http://www.accad.ohio-state.edu/~efarrar/thesis/process.htm

[13] J. Whitney, "To paint on water: The audiovisual duet of complementarity," *Computer Music Journal,* vol. 18, pp. 45-52, 1994.

[14] M. Faulkner and D-Fuse, *VJ: audio-visual art + VJ culture*. London: Laurence King, 2006.

[15] D. Politis, Margounakis, D., Mokos, K., "Visualizing the chromatic index of music," *Proc. 4th International Conference on Web Delivering of Music, 2004.*, Barcelona, Spain, pp. 102-109, 2004.

[16] P. Kolhoff, "Music icons: procedural glyphs for audio files," *Proc. Brazilian Symposium on Computer Graphics and Image Processing*, Amazonas, Brazil, pp. 289-296, 2006.

[17] J. Foote, "Visualizing music and audio using self-similarity," *Proc. 7th ACM International Conference on Multimedia (Part 1)*, Orlando, Florida, pp. 77-80, 1999.

[18] E. Guaus and E. Batlle, "Visualization of metre and other rhythm features," *Proc. 3rd IEEE International Symposium on Signal Processing and Information Technology*, Darmstadt, Germany, pp. 282-285, 2003.

[19] A. Beane, "Generating audio-responsive video images in real-time for a live symphony performance," Thesis, Texas A&M University, College Station, 2007.

[20] A. Johnson and S. Semwal, "Music as an input device," *Proc. VR Workshop Beyond Wand and Glove Based Interaction*, Chicago, IL, 2004.

[21] O. Kubelka. (2000, 10 Nov. 2010). *Interactive music visualization*. Available: http://www.cescg.org/CESCG-2000/OKubelka/

[22] F. Patin. (2003, 10 Nov. 2010). Beat Detection Algorithms. *flipcode*. Available: www.flipcode.com/misc/BeatDetectionAlgorithms.pdf

VITA

Name: Christopher Michael Garcia

Address: Department of Visualization
Texas A&M University, C108 Langford Center, 3137 TAMU
College Station, Texas 77843-3137
C/O Philip Galanter

Email Address: chrismg@tamu.edu

Education: B.S., Computer Science, Texas A&M University, 2008
M.S., Visualization Science, Texas A&M University, 2011