Quantification and Display of Emotions in Music

A thesis presented

by

Danielle Bragg

To

Applied Mathematics

in partial fulfillment of the honors requirements

for the degree of

Bachelor of Arts

Harvard College

Cambridge, Massachusetts

1 April, 2010

Abstract

In this project, we quantify emotion in music, and portray our findings in a visualization system. The work can be divided into three main stages: processing music files in MIDI format to extract the desired musical properties, mapping these properties to emotions, and producing an appropriate and effective visualization of these emotional quantities. The MIDI processing stage provides continuous data on musical content by performing file segmentation followed by property extraction. A mathematical model we develop then maps the extracted properties to quantitative emotional values. The use of vectors and matrices in the structure of the model uniquely suits the problem's demand for a mapping between lists of quantitative data. The numerical values used in the model are derived directly from cumulative documented results of experimental studies. The final visualization presents the numerical results in a Processing sketch, where emotions are encoded with color, and fluid dynamics infuse it with dynamism.

Acknowledgments

I thank my advisor Professor Pfister for his guidance and direction throughout this project. I also thank Miriah Meyer for being a supportive mentor, and making time to discuss my work through countless meetings and e-mails. Both Professor Pfister and Miriah Meyer helped make this research an exciting and rewarding process.

I am also deeply grateful for the constant encouragement from my family and friends. I thank my parents for the wonderful education and many opportunities they have given me, and for their constant support through my endeavors. I thank my friends for growing, learning, and laughing with me. Their love and support has made all the difference.

Contents

1 Introduction **1**
- 1.1 Contributions . . . 2
- 1.2 Overview . . . 3

2 Related Work **6**
- 2.1 Computational Music Analysis . . . 6
- 2.2 Emotional Meaning of Music . . . 7
- 2.3 Visualizing Music . . . 8

3 MIDI Files to Musical Properties **11**
- 3.1 MIDI files . . . 11
  - 3.1.1 What is MIDI . . . 12
  - 3.1.2 Preprocessing - MIDI Segmentation . . . 12
- 3.2 Extracting Musical Properties . . . 14
  - 3.2.1 jSymbolic . . . 14
  - 3.2.2 Our Extraction Helper . . . 15
- 3.3 Converting between Lists of Musical Properties . . . 15
  - 3.3.1 Juslin and Laukka Background . . . 16
  - 3.3.2 jSymbolic to Juslin and Laukka . . . 16

4 The Model - From Musical Properties to Emotions **18**
- 4.1 Problem Exploration . . . 18
- 4.2 Defining Categorical Data . . . 19
  - 4.2.1 Possible Solutions . . . 19
  - 4.2.2 Our Solution - An Empirical Study . . . 20
  - 4.2.3 Results . . . 21
- 4.3 Musical Property Classification Vector . . . 23
- 4.4 The Matrix . . . 24
- 4.5 Emotion Vector . . . 27
- 4.6 Validating the Model . . . 28
  - 4.6.1 Perception of Anger . . . 28
  - 4.6.2 Perception of Fear . . . 28
  - 4.6.3 Perception of Happiness . . . 29
  - 4.6.4 Perception of Sadness and Tenderness . . . 30

5 The Visualization **31**
5.1 Emotion to Color Mappings . . . 31
5.2 Design . . . 33
5.2.1 Bouncing Balls . . . 33
5.2.2 Rain . . . 34
5.2.3 Interactive Visualization . . . 35
5.2.4 Randomized Fluid Dynamics . . . 36
5.2.5 Whirlwind Fluid Dynamics . . . 36
5.2.6 Final Plan . . . 37
5.3 Basic Setup . . . 37
5.3.1 Processing . . . 38
5.3.2 MSAFluid Library . . . 38
5.4 Customization . . . 39
5.4.1 Input Functions and Data Storage . . . 39
5.4.2 Dictating Hue . . . 40
5.4.3 Additional Fluid Forces . . . 40
5.4.4 Initially Normalized Magnitude of Fluid Forces . . . 41
5.4.5 Displaying Emotional Data . . . 42
5.4.6 Keeping it Fresh with Randomization . . . 43
5.4.7 Further Improvements . . . 44
5.4.8 Running the Visualization Tool . . . 44
5.5 Feedback . . . 45
5.5.1 Study Setup . . . 46
5.5.2 Results and Analysis . . . 47

6 Conclusion **49**
6.1 General Discussion . . . 49
6.2 Future Work . . . 50
6.2.1 Music File Preprocessing . . . 51
6.2.2 Mathematical Model . . . 51
6.2.3 Visualization . . . 52
6.3 Applications of Work . . . 53

Bibliography **54**

A Correlations between Musical Properties and Emotions **59**

B Distributions of Musical Properties **63**

C Visualization Code **68**
C.1 ThesisSwirl.pde . . . 69
C.2 Imports.pde . . . 76
C.3 Particle.pde . . . 77
C.4 ParticleSystem.pde . . . 79

Chapter 1

Introduction

This thesis explores and develops methods for quantifying and presenting the emotional content of music. The emotional power of music is arguably greater than the emotional power of any other medium. Music can bring a person to tears. It can lull a baby to sleep, and it can make people of all ages want to get up and dance. Its effects are felt clearly, but the methods it employs to touch our hearts are more elusive.

Because music itself evokes emotional responses, we explore techniques for musical analysis and property extraction. However, automated musical analysis is a challenge. Popular file formats for music storage like WAV and MP3 are difficult to analyze electronically. Because they are audio files, they require signal analysis to yield symbolic data of the musical events. Less popular formats like MIDI provide symbolic data without preprocessing, thus facilitating the extraction of musical properties. Analysis is still a challenge though, as accessing MIDI file data requires a specialized environment. Furthermore, MIDI files do not all conform to a standard format, and instead store information in different places and in different ways. Even with the help of software that extracts musical properties from a MIDI file, obtaining an analysis of different sections of the piece requires breaking the piece into smaller portions for analysis. This segmentation is further frustrated by the fact that the original file's setup information might not hold for a segment halfway through the piece, once features like tempo have changed. Notes that play through segmentation break points also provide a challenge. In this thesis, we present a solution to MIDI file segmentation and property extraction.

Once the musical properties of the music are exposed, determining emotional composition based on those musical properties is another challenge. Indeed, many musicians study and practice for years to develop techniques that will effectively reach their audiences. Tempo, dynamics, articulation, and note duration are a few of the many musical properties that can be manipulated to communicate different messages. The possible ranges of each of these properties is large, and musical properties weighted in different ways combine to create unique effects. Furthermore, it is unlikely that a piece of music communicates a pure emotion, but rather a mix of emotions in different proportions. To encapsulate these dimensions of musical language, we present a mathematical model that maps musical property values to a vector of quantitative emotional values.

Finally, we faced the task of presenting our quantitative emotional data in a suitable format. Such a presentation could help listeners better understand the emotional content of music. Though music can potentially enrich a person's life, it is not always so accessible. In particular, the deeper meaning of classical music can be elusive. This disconnect often lies in a lack of understanding of the music. Visualization techniques create a fitting environment for presenting quantified emotional content. Their capacity for creativity, flexibility, and multidimensionality facilitates the portrayal of abstract data. The final contribution of this work on the quantification of emotion in music is a visualization tool that bases its display on the emotional content of a piece of music.

1.1 Contributions

This work addresses the problem of quantifying the emotional content of music, and presents a visual method for presenting emotional findings. In tackling this task, we present the following contributions:

- **Software to perform MIDI segmentation.** With the help of a MIDIoriented programming environment, our software segments a MIDI file into smaller MIDI files of uniform playback length. A user can easily customize the playback length to suit his needs. (Chapter 3)

- **A software package for extracting a comprehensive list of musical properties from MIDI files.** Supplementing existing music analysis software with our own property extraction software, we create a more complete software package. It extracts the musical properties necessary for drawing conclusions about higher level properties of the music. (Chapter 3)

- **An empirical method to define cutoff points for categorizing musical property values.** Based on a list of user studies discussed in the literature, we create histograms of the distributions of musical properties. Cutoff points are defined so that each categorical range encompasses an equal number of data points. (Chapter 4)

- **A method for quantifying the emotions in music.** We define a mathematical model based on linear algebra that maps a vector of musical property values to a vector of quantified emotions. The matrix is defined by statistical data on the impact of the musical properties on the emotions. The model's computations yield a vector of the cumulative effects of the musical properties on each of the emotions. (Chapter 4)

- **A visualization that relays quantitative information about emotions.** Based on mappings from emotions to colors, the visualization uses quantity of color on the screen to indicate the amount of emotion present. Its use of fluid dynamics facilitates color mixing and heightens the intrigue of the display. (Chapter 5)

1.2 Overview

Our work to quantify emotion occurred in three major stages: MIDI file segmentation and extraction of musical properties from these segments, design of a model mapping musical properties to quantities of emotions, and finally the development of a system to display those emotional quantities. Figure 1.1 provides an overview of this work flow.

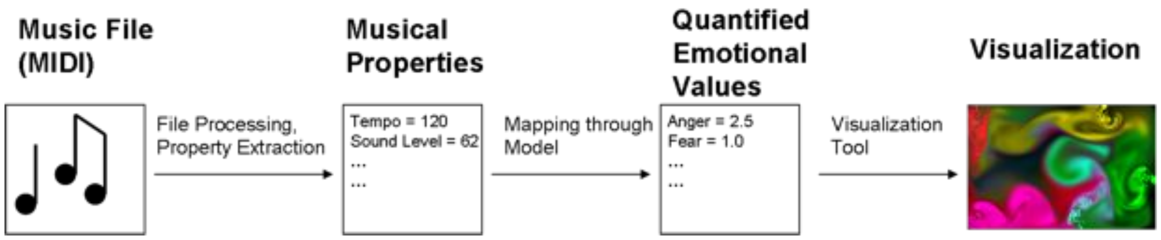

Figure 1.1: Overview of Workflow

Chapter 3 presents the techniques employed to move from a MIDI file to a list of musical properties. It provides background on MIDI files and describes the difficulties encountered in processing them. Because we wanted to perform musical analysis at various points throughout a work, we describe a tactic for segmenting a MIDI file into smaller MIDI chunks. We then present a software package that extracts a comprehensive list of musical properties from each of these MIDI files. In order to make sense of these properties in light of the literature on musical properties and emotional meaning, we also create a conversion between sets of property names.

Chapter 4 presents the mathematical model that maps these extracted property values to quantified emotions. Given data on correlations between musical properties and emotions, we explored the problem through a series of bar charts. Since the problem calls for a mapping between sets of quantitative data, we determined that a mathematical model based on linear algebra most appropriately addresses the problem. In Chapter 4, we describe the components of the model and demonstrate its validity.

Chapter 5 describes the final visualization built on our quantified emotional values. Visualization provides a unique medium for the presentation of our quantified emotions. We describe the design decisions behind the visualization system, as well as its implementation. The chapter concludes with a brief user study and discussion of results.

Finally, Chapter 6 summarizes the challenges we faced in quantifying emotions in music, and discusses future directions of work. Discussion of the greater significance of our work is also included, touching on applications to music performance training and musical composition.

Chapter 2

Related Work

In this project, we present musical property extraction software for performing computational music analysis, a mathematical model for mapping extracted musical properties to emotions, and a visualization tool to display emotional content. In this section, we explore existing work that uses different methodologies to address similar problems.

2.1 Computational Music Analysis

Inspired by growing interest in computational music analysis, researchers have improved techniques accomplish this task. Packages that perform tonal and harmonic analysis [55, 48, 47] operate with varying degrees of success. Other packages extract and analyze musical properties like tempo and meter [58, 39, 54]. These pieces of software extract unique types of information and accept specific file types as input. In our work we use McKay's jSymbolic package [39] for its comprehensive list of 160 musical features, encompassing all the properties necessary for music classification.

Techniques for music extraction are also used to organize music libraries based on properties of the music [46, 10, 37, 49, 43, 62, 60, 36, 32]. Given the limitations of vocabulary to describe music, text-based sorting systems are frequently inadequate for interactions with music collections. Genre definitions overlap, and sometimes we forget the name of a particular artist or song, making it difficult to find the music we seek [53]. To solve these problems, researchers have developed visual techniques for presenting music libraries in terms of musical similarity. These systems sort the library spatially so that similar songs are located close together while dissimilar ones are situated further apart. In order to determine likeness, some applications use text-based properties like year, artist, or genre [46, 10, 60] to evade the task of computational musical analysis. Other applications that do employ computational analysis frequently customize and combine pre-existing methods to suit their project [46, 37, 43], a technique that we adopt in our work.

2.2 Emotional Meaning of Music

On top of this foundation of musical analysis rests theory of musical communication of emotion. Based on close musical analysis, researchers have attempted to answer questions of how music conveys emotion. What cues do performers use to reach their audience? What musical properties convey anger, or fear, or happiness? Much work has been done to answer these questions [41, 65, 64, 66, 20, 22, 21, 14, 15, 51, 34], and some of this work even makes use of visualization techniques to present findings [51, 34]. This research explores the effects of various musical properties and structures on the listener's experience of music. The effects of culture and individuality are also considered.

Some work focuses in particular on defining emotions based on musical properties [40, 25, 31, 33, 56, 26, 5, 38, 19, 18, 35, 3, 24, 28, 63]. To examine the effects of particular properties on the emotional experience of music, researchers frequently conduct case studies. Some research efforts rely on rhythm to help identify high-level features [14], while other efforts depend on different aspects of the music. Yet other work presents evidence from multiple studies linking musical properties to emotions [15, 19, 29]. One project even develops a statistical model of emotional content, based on distributions of user feedback [63]. However, none of these works present a deterministic mathematical model to produce quantitative data on a range of emotions, as we do in this work.

Our visualization is based on the findings presented in Juslin and Laukka's "Communication of emotions in vocal expression and music performance: Different channels, same code?" [29], selected for its comprehensiveness. The work presents findings of 41 unique, independently run experiments that determine the effects of various musical properties on various emotions. On the basis of this experimental evidence, we were able to contribute a mathematical model.

2.3 Visualizing Music

Visualization techniques have also been employed to portray musical content [7, 41, 42, 23, 9, 57, 22, 27, 44, 21, 51, 34, 4]. The score is perhaps the most obvious means of music visualization [27]. Such a visualization consists of a series of instructions that a musician can read in order to execute an accurate performance of a piece. A performer can read it like a book, from start to finish, and it contains information detailing when and how notes should be played. However, scores generally exclude higher level information about the music. Which emotions or moods are evoked? What story is the music telling? Other visualization techniques take over in order to portray this higher-level information about the music.

One such growing form of musical visualization is digital animation. Many music players come with a visualizer to enhance the musical experience. For example, iTunes and Windows Media Player are popular music players that also provide visualization applications. The displays that come with many built-in music players are generally fun to watch, and succeed at securing the user's attention. However, they are mostly based on low-level properties of the music, such as tempo and location of strong beats. Like a score, they generally fail to communicate emotion and other high-level properties.

Nonetheless, researchers have tackled this problem of presenting the emotional content of music. For example, in "Emotion-Based Music Visualization Using Photos" [7], Chen et al. explore the effects of pairing music with specific photographs. They gather emotionally charged photographs, and pair these pictures with various music selections. They determine the emotional content of musical properties by using a statistical model [63], rather than a deterministic one. Through their user study, Chen et al. found that pairing the emotionally charged photographs with music enhances the listener's experience of the music. From the study, they were able to conclude that "emotion-based music visualization enriches users' listening experience" [7]. This evidence of the power of music visualization supports a visual presentation of our emotional quantification.

Other researchers have delved deeper into the content of music using different types of visualizations. For instance, in order to better understand and present the emotions that are implicit in many musical works, one team of researchers designed a series of "joint semantic spaces" [41]. The axes of a space represent different emotional properties, and a point's location in a space indicates the corresponding emotional subtext. Specifically, a musical work is represented by two spaces, one indicating "valence" and "arousal" on its axes, and the other indicating "kinematics" and "energy" on its axes. While informative, this type of abstract analysis is not necessarily intuitive, and its visualization is not aesthetically pleasing.

Other visual techniques focus more on engaging the listener. Concert simulation is one such tactic that can recreate the performer's experience [57, 57] or the audience's perspective [67]. Concert experiences of various musical works have been created based on their musical properties [67]. Characteristics of the performance and of the crowd vary greatly depending on the type of music playing. For instance, the audience for a metallic rock song will have a more rebellious look than that of a pop song. Other visualizations act out the effects of music on an individual human-like dummy [57], creating a vicarious experience for viewers.

Many applications that perform music analysis to organize music collections also employ visualization techniques to subsequently present the libraries. Some layouts are based on cartography [46], where a geographic map includes islands of similar pieces separated by oceans of musical difference. Physical attributes often embody musical attributes. More abstract methods, sometimes based in physics [10], allow designers to portray abstract data like emotion with greater flexibility and ease. For this reason, we employ abstract visualization techniques to present our quantified emotions in music.

Chapter 3

MIDI Files to Musical Properties

This chapter describes the music analysis performed to facilitate the determination and quantification of emotional content. Because the music itself communicates emotion, determination of emotional content requires knowledge of musical content. In this chapter, we explore different types of music files, and settle on MIDI files for the rich processed data they provide. We also examine software packages for music analysis and property extraction, and describe a method for extracting the properties we seek.

3.1 MIDI files

In this section, we provide a brief description of MIDI, the format of our music, as well as the necessary preprocessing of these files. We choose MIDI files because they offer symbolic information without the need for the signal analysis that audio files require. Because the emotional message of a musical piece can change over the course of the work, an overall analysis of the work would lose data on local changes. Instead, we want to perform musical analysis on small portions of the piece throughout its playback time. Furthermore, because musical extraction software typically takes a MIDI file as input, such a continuous analysis requires segmentation of our MIDI files into small chunks that would subsequently be run through the analysis software.

3.1.1 What is MIDI

MIDI, or Musical Instrument Digital Interface, is a file type for digital storage and transmission of music. A piece of music is stored as a series of messages or commands, and the execution of these commands yields the performance of a piece of music. There are many types of MIDI messages, and each corresponds to a musical event. For example, a Note-On message is a call for a note to be played. Other types of messages include instructions to add vibrato or intensity, or a cue for a note to end. Each message includes several parameters. Frequently, the first indicates the channel or voice to which the action should apply; the second specifies the pitch; and the third encodes the desired effect [50]. Standard MIDI is equipped with 16 channels, so the channel specification is a number between 0 and 15. Each of the other properties is encoded in a value from 0 to 127. In the case of pitch, for example, "middle C" normally takes a value of 60 [50].

A MIDI file contains a list of these messages whose execution produces a performance of a particular piece or song. Because there are many different ways that the commands can be encoded and listed, the MIDI Manufacturers Association (MMA) has proposed standards for MIDI files. In accordance with these standards, and for the sake of consistency, we use Standard MIDI Files (SMFs) in this work.

3.1.2 Preprocessing - MIDI Segmentation

In order to allow for the analysis of small pieces of MIDI files, we had to break the files into smaller files that could be run through analysis software. To the best of our knowledge, no publicly available software exists that segments MIDI files into smaller MIDI files according to user preference. Sequencers like Cubase allow users to load and manually edit MIDI files, but this method is impractical given the imprecision of manual input and the large number of segmentations we need to perform. As a result, we decided to create our own MIDI segmentation software.

Because MIDI files consist of a series of MIDI messages, and are not written in standard programming languages, we needed an interface to access MIDI information.

For this project, we chose improv, a C++ environment designed by Craig Sapp [52]. [1] We used Sapp's environment to create software that can segment a MIDI file into uniform-size chunks in terms of playback time. This software takes a single MIDI file as input, and outputs a series of MIDI files. Each of these output files has a playback length of 8.31 seconds, except possibly the last one, which could be shorter. According to experimental results, this is the amount of time that a person must listen to a piece of music before he can form a perception of its emotional message [3].

The segmentation software we created is based on Sapp's example files *midi2text.cpp* and *text2midi.cpp*. *midi2text.cpp* takes a MIDI file and outputs it as a text file of readable commands. *text2midi.cpp* takes such a text file and converts it back to a MIDI file. Our tool first uses the code of *midi2text.cpp* to convert a MIDI file to a textual list of commands. The software then takes this text file and calculates how many 8.31-second segments (MIDI-segments) it divides into. It creates a list of text files, where the nth file stores the events for the nth MIDI-segment of the original MIDI file. It then examines each event listed in the original text file, and determines which text file should store the event command by taking the floor of the event's time divided by 8.31. It also computes the event time to be stored within the smaller text file by calculating the event's time in the original text file modulo 8.31, since this quantity indicates how many seconds after the start of the smaller file the event should occur. The software then stores the event command in the appropriate text file. For example, if an event occurs 11.31 seconds into the piece, the event will be stored in the second text file with an event time of 3.00 seconds. Finally, the software uses the code of *text2midi.cpp* to convert these smaller text files back to MIDI format.

We note that the improv library flattens the tempo dimension. If the library converts a MIDI file to text and back, the data on tempo becomes standardized throughout the piece. The entire work assumes the tempo from the start of the clip. The pieces we use maintain a relatively constant tempo, so the impact of this

[1]Sapp's software is available for public use at *http://improv.sapp.org/*. It provides functions to input and output MIDI files, data structures to store information about MIDI files, functions to access and alter MIDI data, and other structures to facilitate interaction with MIDI files.

flattening is minimal.

While we chose to segment the music files into chunks of equal playback length, other segmentation schemes were also possible. Musical works are comprised of a series of phrases, which occur in a variety of lengths, even within a single piece. Methods have been developed to segment music into musically logical and cohesive sections ([12], [2], [16], [1], [8]), and the implementation and incorporation of different segmentation techniques is an area for future work.

3.2 Extracting Musical Properties

Like the segmentation of our MIDI files, the extraction of musical properties from those MIDI segments also requires a specialized programming environment. As discussed above, the creation and processing of MIDI typically requires MIDI editors, sequencers, and players. For this reason, we explored existing musical analysis software for MIDI files. Some packages like Melisma [55] produce a harmonic analysis rather than a list of the desired musical properties. Ultimately, we chose jSymbolic [39] for the extensive list of properties it produces. Though jSymbolic extracts many desired properties, it neglects dynamics, or volume, so with the help of the improv MIDI library, we created an Extraction Helper to produce this data.

3.2.1 jSymbolic

We chose to use Cory McKay's jSymbolic package [39] to extract the musical properties of the MIDI files. This software takes a MIDI file as input, and extracts 160 musical properties of the piece. These properties describe the work's instrumentation, texture, rhythm, dynamics, pitch statistics, melody, and chords. [2]

The jSymbolic software produces an XML file as its output. To extract the data stored in the XML file, we created a Python script using Python's lightweight DOM API [3]. The DOM, or Document Object Model, is a way of representing several types

[2]A more detailed description of the jSymbolic software and the features it extracts can be found at *http://jmir.sourceforge.net/jSymbolic.html*.

[3]found at *http://docs.python.org/library/xml.dom.minidom.html*

of markup language files, including XML files. Specifically, it represents the file as a tree, where nodes represent segments of the file's information, and child nodes represent more specific information within the segment represented by a parent node. Python's API provides functions for storing XML files as DOMs, as well as functions to access specific portions of the DOM tree, and to retrieve data stored at end nodes.

3.2.2 Our Extraction Helper

Because jSymbolic does not extract all of the musical properties that we need, we created an Extraction Helper to produce the remaining desired properties. Specifically, jSymbolic fails to provide data on Sound Level, or dynamics, of the piece. It does provide information on Variation of Dynamics, defined as the "standard deviation of loudness levels of all notes" [39], but ignores the average loudness.

To provide this missing data, we created a helper file. Again, Craig Sapp's Improv library [52] provided the appropriate environment for accessing and manipulating MIDI file data. To generate data on Sound Level, we averaged the sound levels of all notes played within the file.

3.3 Converting between Lists of Musical Properties

The vocabulary used to describe musical properties differed between jSymbolic and the literature about the emotional messages of musical properties. To facilitate the ensuing mapping to emotional messages, we defined the output from jSymbolic and our Extraction Helper with the terms used in the literature. We also eliminated properties not shared by both, as well as properties like nuances of tone that might exhibit themselves in a human performance but are lost to our electronic MIDI files.

3.3.1 Juslin and Laukka Background

To create a mapping from the musical properties found by jSymbolic to emotions, we used the work presented by Juslin and Laukka [29]. We chose this work because it presents the findings from 41 different musical performance studies. The accumulation of results from these studies provides us with a rich foundation of data for the project.

In the work, the authors present five emotions commonly portrayed in music: anger, fear, happiness, sadness, and tenderness. They also present a series of musical cues or properties used to portray emotions: tempo, sound level, sound level variability, high-frequency energy, pitch, pitch variability, pitch contours, tone attack, microstructural regularity, articulation, articulation variation, vibrato, timing variability, and duration contrasts. For each of these types of musical cues, they create categories for the various possible effects. For example, tempo can be "fast," "medium" or "slow." For each of these categories, Juslin and Laukka then list the studies that find that this type of cue communicates a particular emotion.

3.3.2 jSymbolic to Juslin and Laukka

Juslin and Laukka have a specific list of musical cues that they use to determine the emotion that the music conveys. However, the names that Juslin and Laukka give these cues do not match the names that jSymbolic uses for its outputs. We matched the cues used by Juslin and Laukka to those produced by jSymbolic or our Extraction Helper. Table 3.1 demonstrates this correspondence for each property that we will use.

We note that some of the musical cues examined in Juslin and Laukka are not applicable to our project, since deal solely with MIDI files. For instance, Timing Variability refers to the amount that a performer's execution differs from the written work. Since we are not analyzing scores and performances, this information is not relevant. We similarly disregard High-Frequency Energy, which refers to the fraction of acoustic energy above a certain frequency. Because MIDI files are produced by electronic musical instruments and computers, High-Frequency Energy, which is a

Table 3.1: Conversion between Juslin and Laukka and jSymbolic

Name in Juslin and Laukka	Name in jSymbolic or Extraction Helper
Tempo	Initial Tempo
Sound Level	Sound Level
Sound Level Variability	Variation of Dynamics
Pitch	Primary Register
Pitch Variability	Pitch Variety
Pitch Contours	Direction of Motion
Articulation Variation	Staccato Incidence
Duration Contrasts	Variability of Note Duration

property of sound quality, is typically constant. It is not a means for communicating emotion in MIDI files, so we disregard it.

Chapter 4

The Model - From Musical Properties to Emotions

This chapter describes the development of a mathematical model to map musical properties or cues to emotions. With musical properties extracted from MIDI files as described in Chapter 3, we wanted to quantify the emotional meaning of those properties. Because the literature classifies the values of musical properties into ranges, or categories, without specifying their cutoff points, we first define these cutoff points based on the statistical distribution of an empirical survey. To map this categorized data to emotional quantities, we then develop a mathematical model based on the literature. Because the problem calls for a mapping between sets of quantitative data, we base our model in linear algebra. The numerical values used in the model are derived from an accumulation of experimental data correlating musical properties with emotions.

4.1 Problem Exploration

To gain a sense of the problem of mapping musical properties to emotions, we created a series of bar charts. Each chart displays the impact of Juslin and Laukka's musical properties on a particular emotion, and are included in Appendix A. From these charts, we gained an appreciation for the complexity of the problem. The multidimensionality of both input (musical properties) and output (emotions) suggests a model rooted in linear algebra and involving vectors and matrices. The equation

$$\overline{v}M = \overline{e} \tag{4.1}$$

describes the general form of the model. It takes a vector $\overline{v}$ of musical properties, and multiplies it by matrix M to obtain a vector $\overline{e}$ of emotional values. The specifications of each component of the model are described in greater detail throughout this chapter.

4.2 Defining Categorical Data

As discussed above, Juslin and Laukka create categories for different possible ranges of musical properties, but fail to define those categories concretely. For instance, a tempo can be "fast," "medium," or "slow," and Juslin and Laukka present all of their empirical results according to these categories. However, they do not specify which tempo ranges define these three categories. Instead, they classify the findings of each study by making a judgment call based on the study's write-up and experimental procedures. In order to use the data presented by Juslin and Laukka, we needed to automate the process of property value classification and define concrete ranges for each of the property classes, or categories.

4.2.1 Possible Solutions

One approach to defining these categories is to rely on musical definitions of the terms. For instance, a musical dictionary provides definitions for fast tempo markings like "allegro," and those definitions typically include tempo ranges, in terms of beats per second. Based on these musical definitions, we could define cutoff points for "fast," "medium," and "slow" tempos. However, not all musical properties have readily applicable musical definitions. For instance, "articulation variation" is not a conventional musical term. Because the tactic of drawing on traditional musical definitions provides only a partial solution, we were forced to explore other alternatives.

Another more subjective option would be to base the cutoff points on selected portions of music that demonstrate the various categories of each quality. We would select pieces of music that are "fast," others that are of a "medium" speed, and others that are "slow." Based on an analysis of these groups of pieces, we would define specific numerical ranges for "fast," "medium," and "slow" tempos. The disadvantage of this strategy lies in its subjectivity. One person might feel that a particular piece is "fast," while another would classify it as "medium" or even "slow." The determinations of a music performer or professor might carry some authority, but ultimately the judgments would still be subjective by nature. Furthermore, some musical properties are very difficult to sense and classify. For instance, duration contrasts or articulation variation are difficult to sense by listening to a clip of music. Because of its subjectivity, this solution was abandoned in favor of a more objective method.

4.2.2 Our Solution - An Empirical Study

We ultimately decided to approach this task by evaluating a distribution of property values based on an analysis of real data. The data was comprised of a selection of MIDI files that demonstrate a wide musical range. It is important that the selection spans the entire range of the musical properties because the selection defines the spectrum of each musical property. The chosen works were movements from Franz Schubert's sonatas, specifically the first, second, third, and fourth movements of each of Sonata D.894, Sonata D.959, and Sonata D.960.[1] Schubert, who lived in the early part of the Romantic period, communicates emotion through familiar combinations of musical properties and large-scale organization, we expect his music to comply with our model. We chose to examine his sonatas in particular because they are multi-movement works, consisting of movements that have their own tempo and character.

We base our cutoff points on the distribution of property values extracted from these files. We first split each of these pieces into 8.31-second chunks, the same time

[1]The MIDI files used can be found in the free public MIDI library at *http://www.kunstderfuge.com/schubert.htm.*

required to perceive emotion in music [3]. This created a total of 797 chunks. Our property analyzing software, jSymbolic plus the Extraction Helper described above, then processed these chunks to output the desired musical properties. This output was loaded into MATLAB for further analysis. For each property, a histogram was created displaying the spread of results across the 797 chunks, as displayed in Appendix B. From these histograms, we calculate the cutoff points for the categories so that each category contains an equal number of data points. For each musical property with n categories, we calculate $n - 1$ cutoff points. The fraction of data points between each cutoff point is $\frac{1}{n}$. To determine these cutoff points, we use the Cumulative Distribution Function (CDF), defined as $F_X(x) = P(x \leq X)$, or the probability that a random variable x takes a value less than or equal to X. For each property, we calculated these values of X such that $P(x \leq X) = \frac{k}{n}$, for $k = 1, 2, 3, ..., n - 1$, where x is that property's value for a random piece of music. These values of X become our cutoff points for the musical property's categories.

The one exception to this method is the definition of categories for Direction of Motion. Because Direction of Motion is defined as the fraction of intervals rising in pitch, and is divided into two categories, it suggests a natural definition of these categories. The "higher" category contains all clips with more rising intervals than falling intervals, and the "lower" category contains all clips with more falling intervals than rising intervals.

4.2.3 Results

The cutoff points calculated for the categories of property values are listed in Table 4.1. The units for each property are those used in jSymbolic. Because Juslin and Laukka categorize every musical property value they present, we suspected that the distributions obtained would suggest their own categorical ranges. For example, Juslin and Laukka classify Pitch into three categories. When we analyzed Primary Register, the jSymbolic equivalent of Pitch, we hypothesized that the data would concentrate in three distinct areas of the graph, perhaps creating three "bumps" in the histogram.

Table 4.1: Categorical Cutoff Points

Name in J and L	Name in jSymbolic	Category	Range
Tempo	Initial Tempo	Fast	≥ 159
		Medium	$137 - 159$
		Slow	137
Sound Level	N/A	High	≥ 60.5305
		Medium	$46.0371 - 60.5305$
		Low	< 46.0371
Sound Level Variability	Variation of Dynamics	High	≥ 13.8867
		Medium	$8.7410 - 13.8867$
		Low	< 8.7410
Pitch	Primary Register	Sharp	≥ 66
		Precise	$61 - 66$
		Flat	< 61
Pitch Variability	Pitch Variety	High	≥ 25
		Medium	$20 - 25$
		Low	< 20
Pitch Contours	Direction of Motion	Up	≥ 0.5
		Down	< 0.5
Articulation Variation	Staccato Incidence	Large	≥ 0.2054
		Medium	$0.0677 - 0.2054$
		Low	< 0.0677
Duration Contrasts	Variability of Note Duration	Sharp	≥ 0.2457
		Soft	< 0.2457

Through our data analysis, we found that the range of musical properties is much more continuous than we had expected. Each property produces a relatively smooth range of values. No gaps indicating clear cutoff points are present, and most graphs demonstrate a single mode value. The one notable exception to the continuous nature of the properties can be seen in Figure B.1. This histogram displays a more segmented distribution for Initial Tempo, with notable gaps along the x-axis. However, this segmentation is a direct consequence of the tempo flattening performed by Craig Sapp's improv library during the MIDI file preprocessing.

Despite these similarities, each histogram is distinct from the others. For instance, Figure B.7, the histogram for Staccato Incidence, demonstrates a distribution with a large positive skew. Intuitively, there is virtually no limit to how long notes can linger, but there is a limit to how short they can be. Consequently, distribution of Staccato Incidence demonstrates a dominance of longer notes. At the same time, other distributions like Figure B.4, the histogram for Primary Register, are more centered. This histogram's absence of a long, unbalanced tail reflects the scarcity of both very high and very low notes. The differences in the histograms of each musical property coupled with the continuity of the spectrum of values for each property demonstrate the nontriviality of defining meaningful ranges for musical properties.

As described above, the histograms were segmented into sections where each section contained an equal number of data points. However, this is not the only possible segmentation scheme. Perhaps middle ranges deserve a larger or smaller portion of the data points. Or perhaps categories spanning sections of the tail should contain fewer data points. These modifications would change the classification of musical clips. For example, consider reducing the number of data points allotted to categories spanning the tail of Figure B.7, the histogram for Staccato Incidence. This change would reclassify many clips considered to have a "large" Staccato Incidence to a "Medium" level of Staccato Incidence. Such a change in musical assessment would also impact the evaluation of emotional composition. However, due to the diverse nature of the histograms, making such assessments proved difficult. Exploring different statistical strategies for segmenting the histograms is an area for future work.

4.3 Musical Property Classification Vector

As described above, Juslin and Laukka express the values of musical properties by classifying their values into ordered categories. With the help of our musical property category definitions, we converted each set of numerical property values into a vector displaying their classifications. We call this vector our "Musical Property Classification Vector." It expresses the classification of the eight musical properties used from Juslin and Laukka. Every index in the vector is a Boolean variable, taking a value of 1 or 0 for true and false as appropriate. Each one represents a musical property classification, and each property yields as many indices as it has categories for classification. Six of the musical properties have three categories of classification, and two have two categories of classification. This yields a Musical Property Classification Vector of dimensions 1 by $22 = 6 \cdot 3 + 2 \cdot 2$.

The equation

$$\overline{v} = \begin{pmatrix} Tempo_{Fast} & Tempo_{Medium} & Tempo_{Slow} & SoundLevel_{High} & \cdots \end{pmatrix} \tag{4.2}$$

displays the general format for our vector. The primary name of each index refers to the musical property, and the subscript indicates the category into which that property value falls. The order of the musical properties is Tempo, Sound Level, Sound Level Variability, Pitch, Pitch Variability, Pitch Contours, Articulation Variation, and finally Duration Contrasts. The categories appear in order from greatest to smallest in terms of their numerical ranges.

To gain an intuition for the vector's structure, consider the musical property of tempo. This property is classified into three categories, so three indices in $\overline{v}$ specify tempo classification. The first indicates whether or not the tempo is "fast"; the second indicates whether the tempo is "medium"; and the third indicates if the tempo is "slow." Exactly one of these three indices will take a value of 1, and the others will take a value of 0, since tempo can fall into exactly one of these categories.

The musical property data for each MIDI-segment was converted into a Musical Property Classification Vector to facilitate further processing. MATLAB was used to perform the necessary calculations and conversions.

4.4 The Matrix

In order to determine the values of the matrix M in Equation 4.1, we calculated the proportion of studies that found that a particular category of musical cue communicates a particular emotion.

The equation

$$M = \begin{pmatrix} Anger_{TempoFast} & Fear_{TempoFast} & \dots & Tenderness_{TempoFast} \\ Anger_{TempoMedium} & Fear_{TempoMedium} & \dots & Tenderness_{TempoMedium} \\ Anger_{TempoSlow} & \dots & & \dots \\ Anger_{SoundLevelHigh} & & \dots & \dots \\ \dots & \dots & \dots & Tenderness_{DurationContrastsSoft} \end{pmatrix} \tag{4.3}$$

defines the general form of the matrix M. Each column of M corresponds to an emotion, and each row corresponds to a category of a musical property, like "fast tempo" or "low pitch."For example, 20 of Juslin and Laukka's studies find that anger has a "fast" tempo, 1 finds that anger has a "medium" tempo, and none finds that anger has a "slow" tempo. Since there are 21 studies producing data on the impact of tempo on anger, $\frac{20}{21}$, $\frac{1}{21}$, and 0 become three indices in our matrix. All three results will appear in the column corresponding to anger. The values $\frac{20}{21}$, $\frac{1}{21}$, and 0 will appear in the rows corresponding to "fast tempo," "medium tempo," and "slow tempo," respectively.

The columns of M correspond to Anger, Fear, Happiness, Sadness, and Tenderness in this order. In other words, the first column includes data about Anger, the second about Fear, and so on. This ordering produces an Emotion Vector whose indices hold data about these same emotions in this same order. Similarly, the rows, corresponding to musical properties, are listed in the order of Tempo, Sound Level, Sound Level Variability, Pitch, Pitch Variability, Pitch Contours, Articulation Variation, and Duration Contrasts, with categories listed from "largest" to "smallest." This is the same ordering given to the Musical Property Classification Vector. Because the orderings are the same and differ only in horizontal or vertical direction, when we multiply the Musical Property Classification Vector $\overline{v}$ by M, the indices encoding data about the same property category are multiplied together.

For the data provided by Juslin and Laukka, M takes the values listed in equation

$$M = \begin{pmatrix} \frac{20}{21} & \frac{9}{14} & \frac{23}{25} & 0 & 0 \\ \frac{1}{21} & \frac{3}{14} & \frac{2}{25} & 0 & 0 \\ 0 & \frac{2}{14} & 0 & 1 & 1 \\ 1 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{12} & \frac{13}{21} & \frac{2}{21} & 0 \\ 0 & \frac{11}{12} & \frac{1}{21} & \frac{19}{21} & 1 \\ \frac{3}{5} & 1 & \frac{3}{8} & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{8} & 0 & 0 \\ \frac{2}{5} & 0 & \frac{1}{2} & \frac{2}{3} & 1 \\ \frac{1}{2} & \frac{2}{3} & 1 & 0 & 0 \\ \frac{1}{4} & 0 & 0 & 0 & 0 \\ \frac{1}{4} & \frac{1}{3} & 0 & 1 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{3}{4} & 0 & 0 \\ \frac{1}{3} & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{2}{3} & \frac{1}{4} & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & \frac{3}{5} & 0 & 0 \\ 0 & 0 & \frac{2}{5} & 1 & 1i \end{pmatrix} \tag{4.4}$$

.

This definition of M weights each musical property equally. However, it is possible that certain musical properties have a larger role in determining the emotional composition of a clip of music. For instance, we can weigh each musical property based on the proportion of studies that find a correlation between that property and emotional response. For instance, there are 21 studies linking tempo to anger, but only 4 studies linking pitch to anger. Similar ratios hold for the influence of tempo on the other emotions in comparison to the influence of pitch on those emotions. This difference in the amount of empirical support implies that tempo has a stronger effect on the communication of anger than tempo, in a ratio of $\frac{21}{4}$.

Fortunately, our model was designed with the capacity to adopt different weighting schemes. One possible method to introduce weights is to scale the entries of the Musical Property Classification Vector $\overline{v}$. Before $\overline{v}$ is multiplied by the matrix M, it can be multiplied by a weighting matrix W, according to the equation

$$\overline{v}WM = \overline{e}. \tag{4.5}$$

The nth entry in the nth column of W contains the weight for the nth entry of $\overline{v}$, and the rest of the entries are 0. Alternatively, this scheme can be seen as a scaling of M rather than of $\overline{v}$. From this perspective, the nth entry in the nth column of W contains the weight for the entries of nth row of M. Given the design of our model, other weighting methods can easily be developed to scale the entries of M individually, and not by row. Further experimentation with different weighting schemes is an area for future work.

4.5 Emotion Vector

When we multiply our Musical Property Classification Vector by our matrix as described in Equation 4.1, we obtain a vector $\overline{e}$ of resulting emotions. We call this vector of emotion quantities an "Emotion Vector." This vector has dimensions 1 by 5, since the model produces information about the five emotions described by Juslin and Laukka. Each index of $\overline{e}$ indicates the presence of a particular emotion in a clip of music. More specifically, each quantified emotional value is the sum of the contributions of each musical property to that emotion. The relative values of the vector's indices reveal the dominant emotions in the music. Our model is set up so that the first index represents anger, the second fear, the third happiness, the fourth sadness, and the fifth tenderness.

4.6 Validating the Model

We tested the model by using it to produce Emotion Vectors based on sample data with particular emotional intentions. For each emotion, we produce a Music Property Classification Vector consistent with properties typically used to communicate that emotion. In each case, the Emotion Vectors produced by the model show a dominance for the corresponding emotion, and provide validation for our model.

4.6.1 Perception of Anger

According to Juslin and Laukka, anger is generally characterized by fast tempo, high sound level, much sound level variability, and high pitch level. In accordance with this description, we created Music Property Classification Vector

$$\overline{v} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{4.6}$$

Multiplying by our matrix M, we obtain the Emotion Vector

$$\overline{e} = \begin{pmatrix} 3.0524 & 2.3095 & 2.6283 & 0.3333 & 0 \end{pmatrix}. \tag{4.7}$$

The first index, quantified anger, is larger than any other index, proving that our model perceives communication of anger.

4.6.2 Perception of Fear

According to Juslin and Laukka's studies, fear is characterized by fast tempo, low sound level, much sound level variability, high pitch level, little pitch variability, and rising pitch contour. According to these specifications, we can create the Music Property Classification Vector

$$\overline{v} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{4.8}$$

Applying our model by multiplication by matrix M produces Emotion Vector

$$\overline{e} = \begin{pmatrix} 3.3857 & 4.8929 & 3.5926 & 2.2381 & 2.0000 \end{pmatrix}. \tag{4.9}$$

In this example, the second index, quantified fear, is the largest index. Thus, our model accurately perceives and quantifies fear.

4.6.3 Perception of Happiness

Juslin and Laukka find that happiness is generally communicated through a fast tempo, medium to high sound level, high pitch level, much pitch variability, and rising pitch contour. A high sound level with these other property values yields Music Property Classification Vector

$$\overline{v_1} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{4.10}$$

while a medium sound level yields Music Property Classification Vector

$$\overline{v_2} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{4.11}$$

Again, applying the model through multiplication by M yields the Emotion Vector

$$\overline{e_1} = \begin{pmatrix} 3.7857 & 2.6429 & 4.0033 & 0 & 0 \end{pmatrix} \tag{4.12}$$

produced from $\overline{v_1}$ in Equation 4.10, and Emotion Vector

$$\overline{e_2} = \begin{pmatrix} 2.7857 & 2.7262 & 4.2890 & 0.0952 & 0 \end{pmatrix} \tag{4.13}$$

produced from $\overline{v_2}$ in Equation 4.11. In each case, the third index, quantified happiness, is the dominant emotional value, demonstrating the model's ability to perceive musical cues for happiness.

4.6.4 Perception of Sadness and Tenderness

Juslin and Laukka find that sadness and tenderness are both communicated through slow tempo, low sound level, little sound level variability, low pitch level, little pitch variability, and falling pitch contour. This yields the Music Property Classification Vector

$$\overline{v} = \left(\begin{array}{cccccccccccccccccccccc} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{array} \right). \tag{4.14}$$

When we multiply this vector by M according to the model, we obtain the Emotion Vector

$$\overline{e} = \left(\begin{array}{ccccc} 0.9833 & 2.0595 & 0.7976 & 5.5714 & 5.0000 \end{array} \right). \tag{4.15}$$

The fourth and fifth indices, quantified sadness and tenderness respectively, are significantly larger than the other indices of the vector, and this dominance indicates that our model perceives cues for sadness and tenderness.

Juslin and Laukka recognize that there is little differentiation between cues for sadness and tenderness. In order to better discriminate between the two emotions, our model requires a richer musical analysis and further research on musical properties that communicate these two emotions. Involving harmonic or melodic analysis and increasing the number of musical cues might help in this process by increasing the data available to the model.

Chapter 5

The Visualization

Once we had successfully produced quantitative emotional data for MIDI files, we wanted an interactive method to present our findings. Visualization provides a unique medium for presenting abstract data, so we designed a visualization system in Processing to display emotional content through color. In this chapter, we describe the design and implementation of a visualization system that presents the quantitative emotional data extracted by methods presented in Chapters 3 and 4. We conclude this section with a brief analysis and discussion of user feedback.

5.1 Emotion to Color Mappings

In order to determine which colors best represent our five emotions, we used correlations found between various emotions and colors [61, 11, 6, 59, 30, 13, 45]. These studies examine different properties of color and determine their ties to human experience of emotion. Evidence for a positive correlation between bright colors and positive emotions, and dark colors with negative emotions was found in one study [6]. In terms of hue, green proved to have the most positive associations, while yellow was the most negative [30]. At the same time, the red-yellow spectrum consists of the warmest colors, while the green-blue spectrum consists of the coolest colors [45]. The mappings presented in this section are summarized in Table 5.1. They are largely based on Andrade's work [11], and the specifics of his findings are applied to the emotions we use in the following paragraphs.

According to Andrade, people in the United States generally associate anger with dark and saturated colors. Since our research is being produced in this country, we will rely on this association. Anger is a negative emotion, and he also finds that people harbor negative connotations for purplish and yellow-red colors. Thus, these "negative" color ranges complement the negativity of anger. Since dark red is a saturated color within the yellow-red color range, we choose it to represent anger. The warmth of the color red [45] also compliments the heat of anger.

Fear is a weak emotion, and according to Andrade people associate weakness with light and unsaturated colors. Thus, they are likely to associate fear with light and unsaturated colors. Like anger, fear is also a negative emotion, so its representation by a color in the purplish or yellow-red color range is also appropriate. Yellow is a light color in the yellow-red color range, so we represent fear with yellow. This mapping resonates with [30]'s finding that yellow evokes extremely negative emotions.

Andrade also claims that people tend to associate happiness with saturated colors. Furthermore, they typically associate the blue-green color range with positive emotions. Happiness is a positive emotion, so a mapping to a color within this hue range is fitting. Ultimately, we chose to represent happiness with a saturated green. The mapping of green to the most positive emotion agrees with [30]'s findings that green evokes the most positive emotions.

Sadness and tenderness are softer emotions, typically associated with light and unsaturated colors according to Andrade. Sadness is also a negative emotion, again typically associated with purplish and yellow-red colors, so we represent sadness with light purple. On the other hand, tenderness is a more positive emotion, which again corresponds to the blue-green color range. Thus, light blue provides an appropriate mapping for tenderness.

We recognize that this assignment of colors to emotions is not the only viable one. The colors that we chose to represent the different emotions satisfy the saturation and hue requirements of the literature that we examined, so they are appropriate assignments. However different assignments also meet these requirements. For example, we

Table 5.1: Mappings from Emotions to Colors

Emotion	Color
Anger	Deep Red
Fear	Yellow
Happiness	Vibrant Green
Sadness	Light Purple
Tenderness	Light Blue

could represent anger with a maroon or even yellow rather than dark red and still comply with the guidelines set by the literature. Further experimentation with color assignments is an area for future developments.

5.2 Design

With the preprocessing of the music files completed and an emotional encoding system established, we began to build our visualization tool to present the emotional composition. In order to create an effective tool, we examined several possible plans for the visualization and selected the one that best captured the essence of our emotional data. The ideal visualization would encode the emotional categories through a qualitative parameter like color. It would also portray the quantitative data from the analysis of emotional content. On top of being informative, the visualization would appeal to users, helping them become more involved in the music. In the following section we present and describe some of the visualization plans considered in the formulation of our final design.

5.2.1 Bouncing Balls

Our first visualization plan consists of a series of balls that bounce around the screen. Color encodes emotion. The number of balls of a particular color corresponds to the amount of that emotion that is present. The balls bounce around randomly, and can travel faster or slower depending on the tempo of the section of the piece, as shown in Figure 5.1. Another option would be to have a single ball represent each color, where the size of the ball indicates the dominance of that emotion. This type of visualization is appealing for its clean simplicity. However, in comparison to more sophisticated visualizations it falls short in its appeal to potential users.

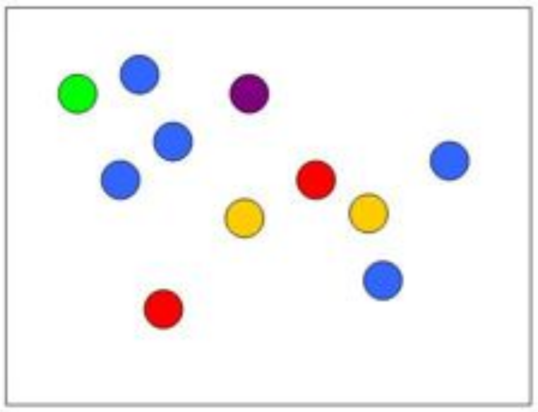

Figure 5.1: Bouncing Balls Design Sketch

5.2.2 Rain

Another potential visualization is reminiscent of the rain streaming down outside a window on a stormy day. The different colors of the raindrops indicate different emotions. The proportion of raindrops of various colors correlates to the significance of the encoded emotions. The design sketch in Figure 5.2 demonstrates streaks of raindrops of the same color. In an actual implementation, we might find it valuable to mix raindrops of different colors to provide a more integrated representation of the emotions. We may vary the intensity of downpour to correspond to the amount of sound being produced. We can also vary the speed at which the raindrops fall according to the tempo of the music. This visualization encodes the desired information, but presents the danger of becoming monotonous. Unless dramatic gusts of wind or other theatricalities can be introduced, this visualization lacks the intrigue of existing visualizations in commercial packages.

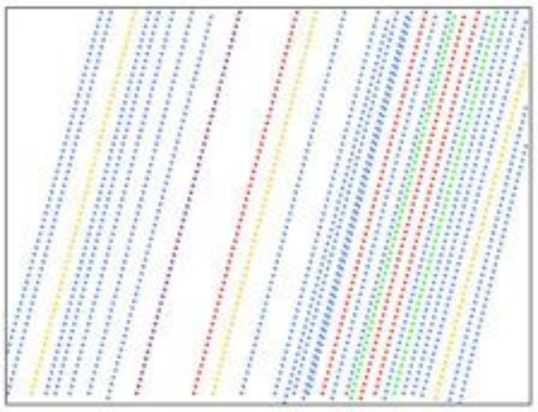

Figure 5.2: Rain Design Sketch

5.2.3 Interactive Visualization

In a more interactive visualization system, the user is given the opportunity to impact with the visualization. As the mouse moves across the screen, a rainbow of colors appears, as shown in the screen shot sketch in Figure 5.3. Each color represents a different emotion, and the proportion of each color corresponds to the emotion's prevalence in that section of the piece. A disadvantage of this visualization plan is that it lacks an intuitive way to encode extra data besides quantified emotions, like tempo. The user might also tire of constantly moving the mouse in order to see the visualization.

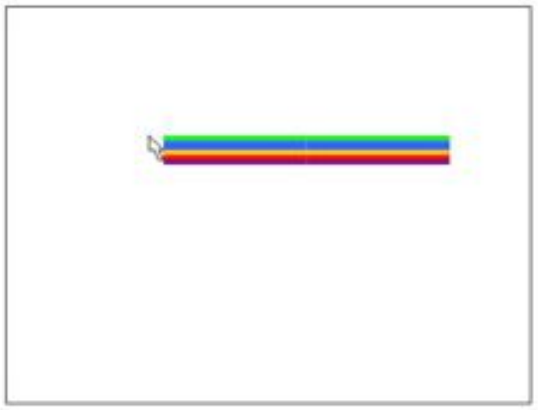

Figure 5.3: Interactive Visualization Design Sketch

5.2.4 Randomized Fluid Dynamics

Our later visualizations were inspired by fluid dynamics. In the randomized version sketched in Figure 5.4, the amount of fluid shown in a particular color corresponds to the dominance of the emotion it encodes. The movement of colors in this visualization allows for randomization. The fluids flow out of randomly appearing spouts on the perimeter of the screen. The colors will not necessarily all come together in one place. This separation of colors fails to represent the integrated nature of the emotions in the music. However, the color segregation offers the user a clear view of the differences in amounts of color, and corresponding emotional values.

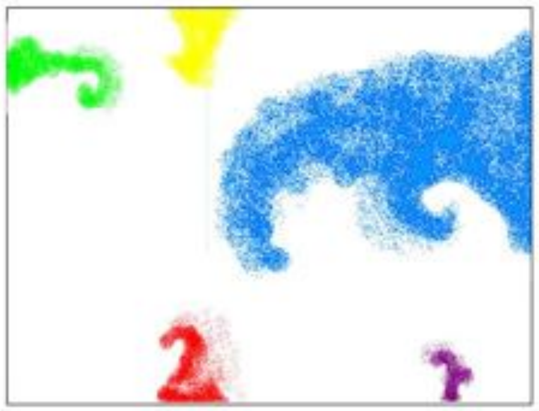

Figure 5.4: Randomized Fluid Dynamics Design Sketch

5.2.5 Whirlwind Fluid Dynamics

Like the “Randomized Fluid Dynamics” plan, the visualization sketched in Figure 5.5 is also inspired by fluid dynamics. Here we again encode emotion with color. Each color streams out of a fixed spout on the perimeter, and the amount of colored fluid that emanates corresponds to the dominance of the related emotions. The fluids are subject to a type of whirlwind effect, and mix at the center of the screen. This center region provides an integrated display of the colors, thus representing the mix of emotions in the music. In its potential to display the colors in an integrated fashion, this visualization is similar to the “Rain” visualization. The visualization’s capacity for random swirling shapes holds potential for the creation of a visually captivating presentation.

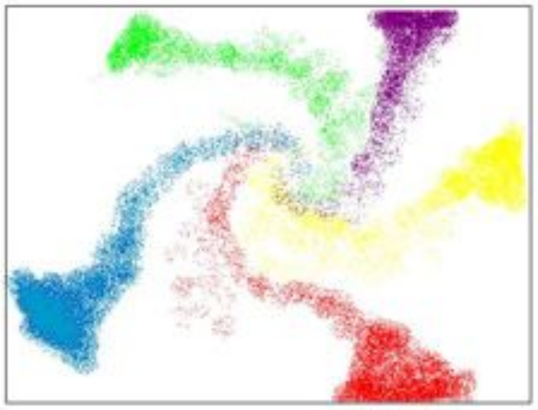

Figure 5.5: Whirlwind Fluid Dynamics Design Sketch

5.2.6 Final Plan

We chose to implement a mix of the “Whirlwind Fluid Dynamics” and “Randomized Fluid Dynamics” plans. The basis of these visualizations in fluid dynamics coupled with a capacity to incorporate randomization facilitates the creation of an engaging visual product. Our visualization plan includes five spouts, located equidistant from each other along the perimeter of the screen. Each spout spews fluid of a different color, corresponding to a particular emotion. The volume of fluid emitted from a particular spout correlates directly with the amount of emotion present. In accord with study findings, larger images trigger greater emotional responses [13], so our visualization associates the dominance of an emotion with greater amount of color.

5.3 Basic Setup

Processing was used to create the visualization portion of the project. The MSAFluid Processing library provided an accessible environment for manipulating fluid dynamics displays. We used a demo provided by the library as the skeleton for our implementation.

5.3.1 Processing

Processing is a programming language created by Ben Fry and Casey Reas. [1] This language is ideally suited for the creation of visually interactive products. Processing comes with built-in functions that draw various shapes, manipulate graphics, and detect user interaction through the keyboard and mouse. When a Processing file is run, a window is created that displays visual effects of commands in the file. Due to this convenient environment for creating sophisticated visual products, we chose Processing for this project.

Processing also contains specialized libraries that facilitated the creation of our visualization. It is an open-source language that is constantly growing and diversifying. As users apply the software to increasingly sophisticated and diverse projects, contributors upload new libraries for manipulation of specialized environments. The MSAFluid library by Memo Akten was particularly applicable to this project. MSAFluid is a library for solving real-time fluid dynamics simulations. Its computations use Navier-Stokes equations, but the details of the physics used by the fluid dynamics solver are beyond the scope of this work.

5.3.2 MSAFluid Library

The MSAFluid library is accompanied by a demo program, called MSAFluidDemo. When run, this sample program visualizes fluids moving across the window in accordance to motions of the mouse. Each time the user moves the mouse, a stream emanates from the mouse's location. When the mouse is still, existing streams continue to move, but no additional fluids are added. These fluids slow down according to the system solver's calculations. The color of the added fluid varies over time, as well.

Our visualization borrows code from this demo package. Specifically, we use the

[1] Processing documentation and downloads are available at *http://processing.org/*.

Particle and ParticleSystem classes. The Particle class holds all the relevant data and functions necessary for creating and manipulating a particle of fluid. It stores the particle's position, momentum, and size. The class contains functions to create new particles, update the data of a particle as time passes, and display a particle. The ParticleSystem class contains data structures for storing particle information and keeping track of the number of particles and other relevant system information. It contains function to create a system of particles, update the system, and add new particles.

Additional functions that we use are the addForce() and fadeToColor() functions. addForce() takes as input the necessary data about location and velocity of a force to be added, and updates the system. The fadeToColor() function dictates how color fades due to a particle's motion and lifespan. Functions in the ParticleSystem class make calls to fadeToColor() in order to update the display to incorporate the decay of fluid appearance. We also use the basic setup from the demo package that initializes the fluid space.

5.4 Customization

On top of the structural backbone provided by the MSAFluid library and demo, we added code to produce a system of fluids that matches our design. For further details on the code created, see Appendix C.

5.4.1 Input Functions and Data Storage

Because the visualization depends on a list of emotion vectors, it includes methods for inputting these vectors. A two-dimensional array called *emotions* stores the magnitudes of each emotion for each 8.31-second segment. The first dimension indicates the segment number, and the second dimension specifies the emotion. In this array and throughout the visualization, when emotions are referenced by numbers, the integers 0 through 4 correspond to anger, fear, happiness, sadness, and tenderness, respectively.

The visualization also accommodates the customization of different phrase lengths. In order to receive information on how the file has been broken up, the visualization accepts a list of segment lengths to accompany the list of emotion vectors. The nth element stores the length of time for which the nth vector represents the emotional composition of the piece. Throughout this thesis, files have been divided into 8.31-second segments for analysis, so each element in the list is 8.31 for this work.

To store the mappings from emotions to colors, the visualization also creates an array of hues. The vector contains one index for each of the five emotions, and again, the indices correspond to the emotions in the order described above. The importation and initialization of all values occurs in the file's setup() function.

5.4.2 Dictating Hue

In order to specify which emotion–and thus which colored fluid–appears on the screen, we added an input parameter to the addForce function. This parameter indicates the emotion that the added force will represent, and takes on values from 0 to 5. The same mapping of colors to integers is used for this input as well.

In the demo version, addForce determines the hue of the fluids that it adds as a cyclic function of the number of frames created. The following line of code is used to dictate the hue of the force being added, where x and y specify the coordinates of the force:

```
float hue = ((x + y) * 180 + frameCount) % 360; .
```

Because we wanted the color of the added fluid force to depend solely on the emotion it encodes, we replace the demo's code with:

```
float hue = hues[emotion]; .
```

5.4.3 Additional Fluid Forces

Our visualization also differs from the demo in the number of times that it adds fluid forces to the system. The demo version adds forces whenever the mouse is moved.

Thus, all of its addForce calls are located in the mouseMoved() function, a built-in function of Processing that is called whenever the mouse moves. The demo version specifies that a fluid force should be added at the location of the mouse with a force proportional to the velocity at which the mouse moved. Our design is independent of mouse action, so we removed the mouseMoved() function commands entirely.

Instead of relying on user interaction, our visualization adds fluid forces continually. To implement this, we call the addForce function in the draw() function, which is another built-in Processing function whose code executes continuously while the program runs. Since we call addForce from this function, addForce is continuously run to create the appearance of continuously flowing streams of fluids.

Our visualization consists of five spouts of fluid, each emitting a uniquely colored fluid corresponding to a different emotion. In order for each spout to have an equal influence on the composition of the display, the five spouts were equally spaced out along the perimeter of the screen. Initially, the first spout to be initialized was placed in the upper left-hand corner of the screen. Each subsequent was placed at an appropriate distance along the perimeter in a clockwise direction. For aesthetic reasons, we shifted these positions clockwise by 60 pixels. The code for these computations, replicated in Appendix C appears in the setup() and depends on the width and height settings of the screen.

5.4.4 Initially Normalized Magnitude of Fluid Forces

The position of each spout also determines the direction of the emitted fluid. So that each fluid maintains an equal opportunity to influence the composition of the screen, each spout points toward the center of the screen and emits fluid through calls to the addForce() function. This function takes as input the x and y components of the desired fluid force. The preliminary x component of the force emitted for each spout is simply the x position of the spout minus the x position of the center of the screen. Similarly, the preliminary y component is the y position of the spout minus the y position of the center of the screen.

These preliminary x and y values point all spouts towards the center of the screen, but give the emitted fluids varying magnitudes depending on the spout's distance from the center of the screen. To resolve this inconsistency, we scale all fluid forces to equal a constant c. Suppose x and y are the unscaled x an y components of the fluid emitted from a spout. We calculate our scaled x component to be $\frac{cx}{\sqrt{x^2+y^2}}$ and the scaled y component to be $\frac{cy}{\sqrt{x^2+y^2}}$. To verify the scaling effect, we compute the new magnitude of the force to be $\sqrt{(\frac{cx}{\sqrt{x^2+y^2}})^2 + (\frac{cy}{\sqrt{x^2+y^2}})^2} = \sqrt{\frac{c^2(x^2+y^2)}{x^2+y^2}} = \sqrt{c^2} = c$ [17]. The visualization performs this scaling for the x and y components of the fluid emitted by each fluid spout, and substitutes in the setup(). The value of c in the visualization is $\frac{1}{100}$, chosen for its aesthetic impact.

5.4.5 Displaying Emotional Data

As time passes, the emotional composition of the music changes, and the visualization must update the emotional data that it uses. To do this, it maintains a global variable `curr_seg` that keeps track of which segment of the piece is currently being played. The emotional data can then easily be accessed at `emotions[curr_seg]`. Each time the draw() function loops, it checks if the current playback time is within the temporal range of the `curr_seg`th segment of music. If not, it increments `curr_seg`. To access the playback time, the visualization relies on Processing's built-in millis() function. This function returns the number of milliseconds that have passed since the program began to run.

Based on the obtained emotional data, the magnitudes of the fluids emitted from the five spouts are scaled yet again. We want the magnitude of fluid coming from each spout to be proportional to the presence of the corresponding emotion. Since the magnitude of each force is normalized each time the draw() function runs, we can multiply the magnitude of each force by the magnitude of the corresponding emotion to obtain a new proportional magnitude. To see how we do this, suppose p is the emotional contribution of a particular emotion, and x and y are the normalized x and y components of the fluid from the corresponding spout. Our visualization replaces x with px and y with py. This creates a total magnitude of $\sqrt{(px)^2 + (py)^2} =$

$\sqrt{p^2(x^2+y^2)} = p\sqrt{x^2+y^2}$. Since $\sqrt{x^2+y^2}$ was the original magnitude of the force, these modifications of the x and y components of the force yield the desired change in magnitude. Furthermore, these changes do not affect the direction of the force because the ratio of the x and y components remains constant.

5.4.6 Keeping it Fresh with Randomization

In order to keep the visualization interesting, we introduced an element of randomness. Specifically, we constantly change the direction of the fluid spouts by a small, random amount. This allows the fluids to form new, interesting patterns as they mix. It also means that each time the visualization runs, it creates a slightly different presentation.

We want to modify the direction of each flow without affecting their magnitudes. Suppose that x is the x component of a force and y is the y component, and suppose we increase x by r. Then the y component must also be modified. Specifically, to maintain the magnitude of the force, the y component must be $\pm\sqrt{x^2+y^2-(x+r)^2}$. To verify this result, we see that the magnitude of the total force with these new x and y components is $\sqrt{(x+r)^2+(x^2+y^2-(x+r)^2)} = \sqrt{x^2+y^2}$. Since $\sqrt{x^2+y^2}$ was the original magnitude of the force, we know that we have not altered its magnitude. In our visualization, r takes a random value between -0.1 and 0.1. This value of r adds intrigue to the presentation without creating distractingly large variations in the directions of the flow.

For the spouts along the top and bottom borders of the screen, the visualization first randomizes the addition to the x component of the flow. It then calculates the y component, and place a + or - in front of it depending on whether the flow is on the top or bottom to ensure that it is pointing toward the center of the screen. Similarly, for spouts along the left and right borders of the screen, it randomizes an addition to the y component of the flow, and calculates the x component based on the new y value. Again it chooses the sign of the x component depending on whether the origin of the flow is on the left or right side of the screen to ensure that the flow points towards the center of the screen.

5.4.7 Further Improvements

To present a smoother arrangement of colors, the visualization also orders the spouts so that similar colors are adjacent to each other. In our implementation, the colors initially appeared in clockwise order of deep red, yellow, vibrant green, light purple, followed by light blue. This translates into emotions taking the clockwise ordering of anger, fear, happiness, sadness, followed by tenderness. Because purple is a mix of red and blue, its location between green and blue, rather than between red and blue, disrupted the natural flow of the visualization. As a result, we switched the location of the spouts spewing purple and blue fluids. This yields a clockwise ordering of emotions of anger, fear, happiness, tenderness, and sadness. Interestingly, this rearrangement yields a smoother ordering of emotions as well as colors, thus supporting our mapping between emotions and colors. Instead of appearing between happiness and tenderness, sadness sits between tenderness and anger. This location makes more sense because the negative connotations of sadness link to the negative connotations of anger, while the softer aspects of sadness also link it to tenderness.

5.4.8 Running the Visualization Tool

In order to run our visualization, the MSAFluid package must be downloaded and placed in the "libraries" folder of the "Processing sketchbook" folder.[2]

The software also requires two TSV (Tab Separated Values) files as input. One file contains quantitative emotional data in a MIDI file. Each row contains the emotional content of a MIDI-segment, and each column contains the data for a particular emotion. The columns contain information about anger, fear, happiness, sadness, and tenderness, respectively. The second TSV file contains a list of MIDI-segment lengths. For our visualization, all segments have a length of 8.31. Importing the lengths of the segments allows users to modify segment lengths without changing the visualization code, thereby maintaining the extraction barrier of our code.

Figures 5.6 and 5.7 display sample screen shots of the running visualization. In

[2]The entire library is available at *http://processing.org/reference/libraries/*.

Figure 5.6, all five colors mix and are relatively equal in their dominance. Contrastingly, Figure 5.7 presents a dominance of red and yellow. Since red encodes anger, and yellow encodes fear, we know that this screen shot accompanies a section of music dominated by these two emotions.

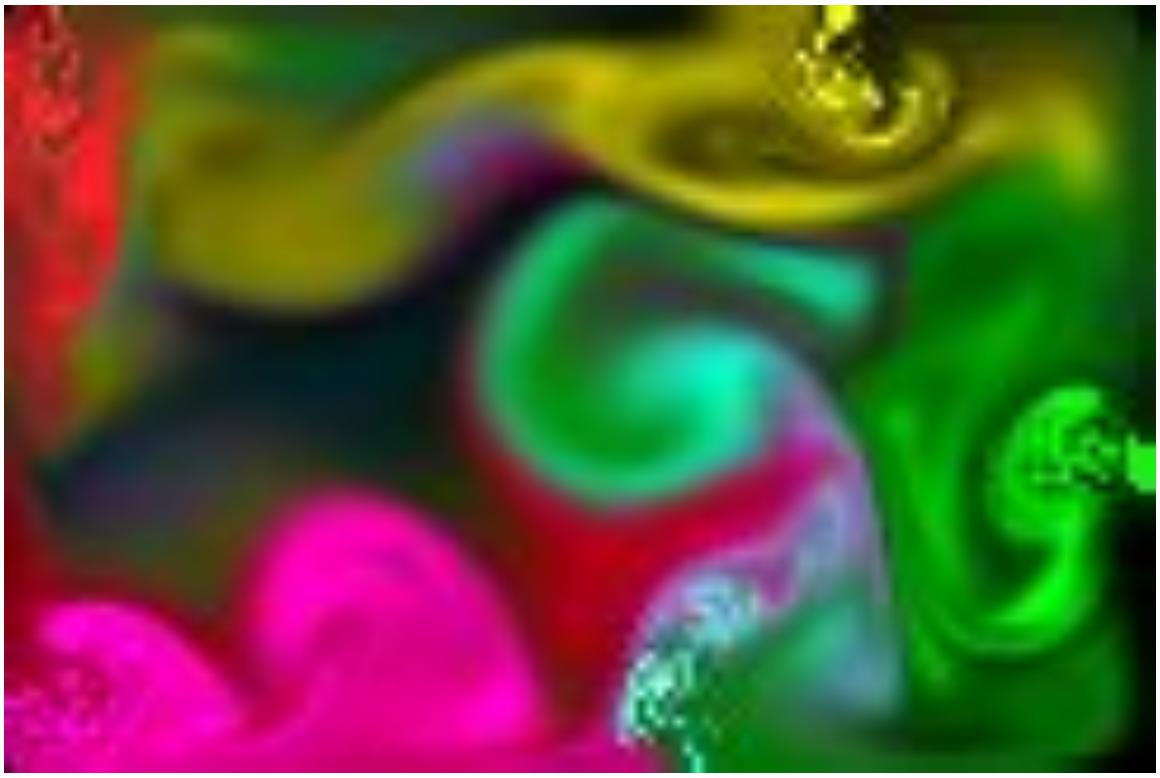

Figure 5.6: Screen Shot of the Visualization for the First Movement of Schubert's Sonata D.894. The balance of colors displays a balanced mix of emotions.

5.5 Feedback

To gauge the success of our visualization at effectively displaying quantitative emotional data, we conducted an informal user survey. Several MIDI files were played for users, accompanied by visualizations. Each MIDI file was played twice, and one of two distinct visualizations accompanied the music. One was our visualization, and the other was a randomized version of our tool. The order of the visualizations was randomized. Users were asked to give feedback on their experience, and to compare

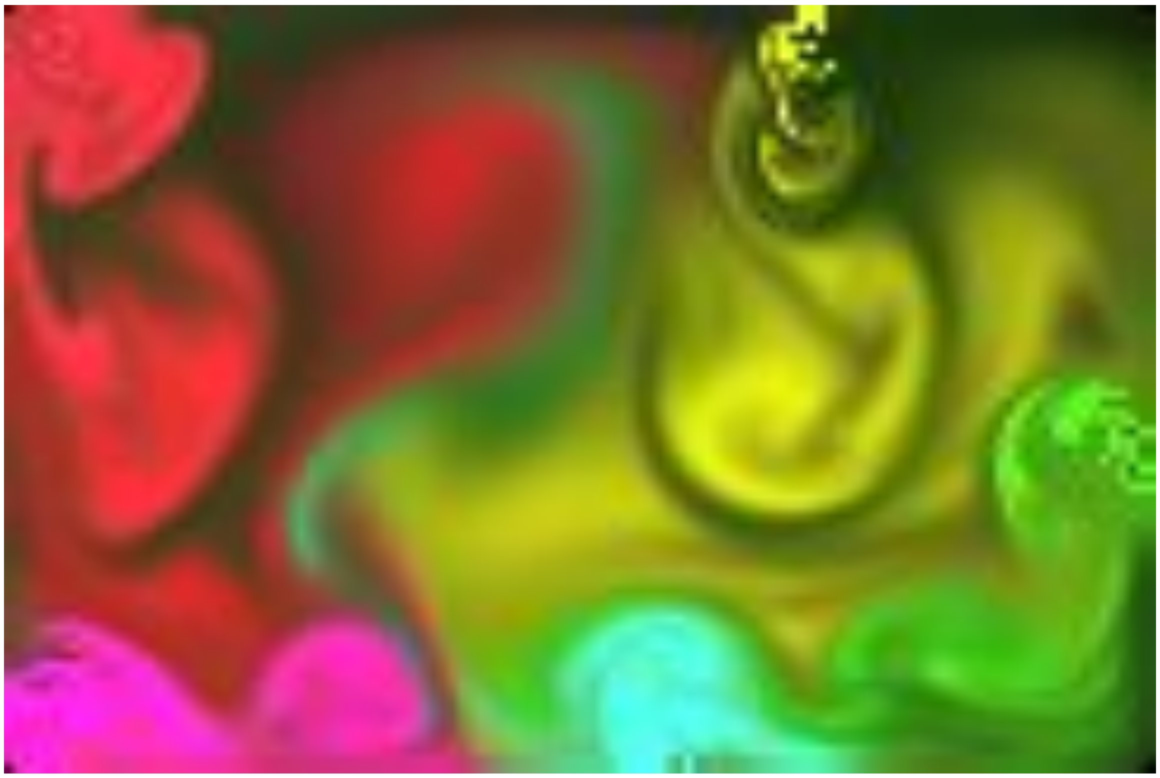

Figure 5.7: Screen Shot of the Visualization for the First Movement of Schubert's Sonata D.894. The dominance of red and yellow correlate to a dominance of anger and fear.

their experience of the two visualizations for each MIDI file.

5.5.1 Study Setup

The randomized visualization differed from ours in the emotional data used to generate the display. Our visualization based its display on the emotion vectors calculated for the piece. Contrastingly, the randomized version based its display on random emotional vectors. Since the numerical values of each emotion typically ranged from 0 to 5, it chose random numbers within this range. The temporal location of the emotion vectors is uniform throughout the visualizations; both visualizations compute and portray a new set of emotional values every 8.31 seconds. The divergent emotional values between the two visualizations caused different amounts of colored fluids to be emitted by each visualization for any given moment during each piece.

The MIDI files played for the users were the third movement of Schubert's Sonata D.894 and the second movement of Sonata D.960. These movements were chosen for their wide range of musical and emotional content. Playing several different MIDI files for users increased the reliability of the feedback, since the visualization might coincidentally perform very well or poorly with a particular piece of music. After the users listened to a piece of music and viewed both visualizations, they were asked for feedback on the experience. For each piece, the user was asked to select his preferred visualization, and to explain his choice.

The users we surveyed consisted of ten undergraduate students. They possessed varying degrees of affinity to and training in music, and their academic studies ranged across the humanities and sciences. They were not provided with the theory behind the making of the visualization, nor were they told that colors represent emotions, or that the visualization was meant to convey an emotional message.

5.5.2 Results and Analysis

The users who preferred the completely randomized visualization cited the display's dynamism as the source of its appeal. They liked that the visualization changed more drastically over the course of the music. In comparison, our visualization presented smoother transitions and fewer unexpected changes. The relative stability of our visualization system speaks to its success at relaying emotional content, since music is rarely a sequence of radical changes in emotional content.

Users also noticed that the basis of our visualization system differed from the basis of popular music visualizers that do not encode emotion. Many users referenced other music visualizers that encode the rhythm with a pulsing screen or changes in volume through the introduction of new motion. They drew upon past experience with music visualization systems to evaluate ours. However, the aim of our visualization system differed from those that the users had seen. The focus of those systems was to present an entertaining visual product that compliments the music, while the focus of our system was to convey deeper meaning in the music in the form of emotions. The fact that users noticed dissimilarities between our visualization and other systems they have seen before is a testament to the novelty of our approach.

In testing our system, we noticed that it is sometimes difficult to differentiate between the amounts of colored fluids. To increase the visible contrast between the amounts of fluids, we modified the velocities of fluid flow. Instead of assigning velocity to be proportional to the quantitative emotional values, we made velocity proportional to the square of the corresponding emotional value. We also scaled down the velocities so that the user would have more time to absorb the data. Figure 5.8 displays a screen shot of this visualization with increased contrast. Though this visualization heightens differentiation between colors, users found this type of visualization more boring than its faster paced predecessors.

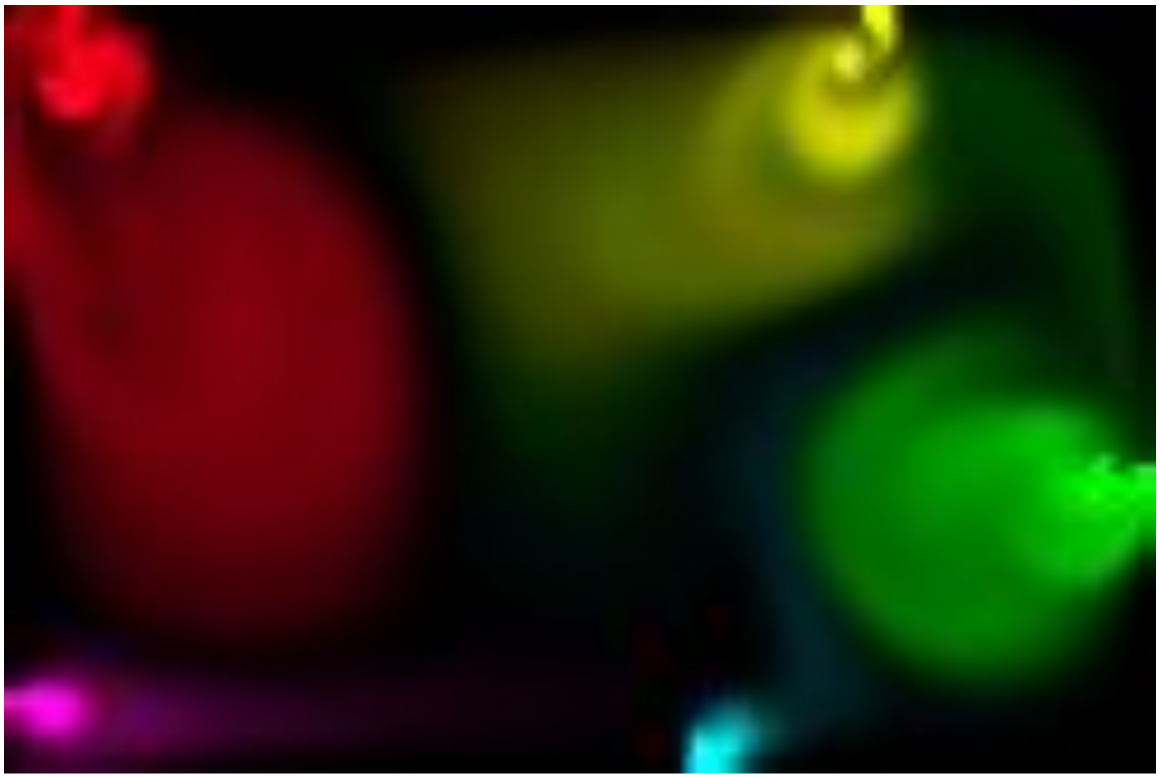

Figure 5.8: Screen Shot of the visualization with increased differentiation between velocities for the First Movement of Schubert's Sonata D.894.

Chapter 6

Conclusion

Through the development of software to quantify and display emotion in music, we encountered and addressed unexpected challenges with innovative solutions. In these challenges, we also found opportunities for future explorations and developments.

6.1 General Discussion

The first stage of our research consisted of extracting musical properties from MIDI files. Because we wanted to obtain a nearly continuous spread of data, we segmented the MIDI files into smaller chunks before performing musical analysis. This file segmentation proved unexpectedly difficult. MIDI files are unstandardized, and store their data across a number of tracks. The first track stores data to initialize the file, but commands that modify this setup occur throughout. Because the first segment's setup might not hold for subsequent segments, initializing these segments is a challenge. Tempo changes in particular confound the segmentation process. We circumvented these problems by using improv, a MIDI environment whose built-in functions flatten tempo. Once the files were segmented, the extraction of musical properties ran smoothly. The jSymbolic package extracted a large number of properties. With the aid of our Extraction Helper to determine average volume, we obtained a comprehensive list of musical properties, that can be applied to other problems involving low-level property extraction.

Once we had processed the MIDI files and extracted musical property values, we tackled the task of creating a model to map these values to emotional values. Because the literature categorizes property values into ranges, but fails to define cutoff points, we faced the unexpected challenge of defining these ranges and developed an original statistical approach to solve the problem. Our solution exposed aspects of the musical properties worthy of further exploration. With these ranges defined, we proceeded to build a model based on empirical results presented in the literature. The model takes a vector of musical property values, multiplies this vector by a matrix, and obtains a vector of quantified emotional values. It derives elegance and flexibility in its simplicity, and provides the innovation of a deterministic mathematical model quantifying emotion from low-level properties.

Finally, in creating a visualization tool to display the emotional data we had quantified, we faced design and implementation challenges. The task of encoding the abstract qualities of emotions was solved by a mapping of emotions to color. To provide an engaging but flexible setting, we based the visualization in fluid dynamics. Directing the fluids towards the center of the screen and incorporating randomness in their flow called for geometric calculations. A final survey of user feedback provided a sense of the innovation and effectiveness of the system.

6.2 Future Work

Each stage of quantifying and portraying emotion in music offered opportunities to enhance our work further. We began the process by segmenting and extracting musical properties from MIDI files. The domain of application for our work could be greatly increased by expanding these techniques to accommodate not only MIDI files, but other types of music files as well. Next, we created a mathematical model to map musical properties to quantify emotional values. This model could be enhanced by redefining its categorical ranges of musical properties, experimenting with the weighting of the properties, and increasing the total number of musical properties used. Finally, we created a visualization based on the product of our model. This display could be enhanced by providing richer data, and offers significant areas for application.

6.2.1 Music File Preprocessing

A major focus of this thesis has been the segmentation of music files and the extraction of their musical properties. The system that we present performs these operations for MIDI files. This file format was chosen for its explicit storage of musical events, which facilitated the extraction of many musical properties.

We are interested in expanding the methodology to accommodate other music file formats. Because MIDI makes up a small fraction of all music files, processing other types of files has a wide appeal. Once musical properties are extracted, our mathematical model mapping these properties to emotions holds, regardless of the original format of the music.

6.2.2 Mathematical Model

The development of a mathematical model mapping musical properties to emotions has been another major contribution of this work. The model takes a vector of musical property values and multiplies it by a matrix to produce a vector of emotional quantities. The literature categorized property values into ranges, and we defined those ranges based on an statistical analysis of empirical data of property values. The indices of the matrix were also defined by statistical data on the impact of property values on the emotions.

We would like to experiment with the definition of the ranges for the categorization of musical property values. The current ranges were defined based on the distribution of the surveyed data, such that each range contained an equal number of data points. However, it is possible that the different properties require different range definitions, based on the individuality of their histograms, listed in Appendix B. Modified range definitions would translate into different categorizations of musical property values, mapping to potentially more accurate quantified emotional values through our model.

The mathematical model could also be improved by weighting the effects of the musical properties on the emotions differently. Our current model values each musical property equally. However, it is possible that certain musical properties have a larger role in determining the emotional composition of a clip of music. For instance, a tempo might have a greater impact on emotional content than average note length. Furthermore, a property like tempo might have a greater or lesser effect on the presence of specific emotions. Our model is designed to incorporate weightings through the introduction of a weighting matrix W, as described in 4.5. We would like to examine the effects of different numerical assignments of the weighting matrix.

Expanding the list of musical properties used would also enrich our model. The current model maps 8 musical properties to the 5 emotions described by Juslin and Laukka. A higher number of musical properties would account for more dimensions of the music, and thus yield more accurate mappings. Among the properties added, elements of harmonic analysis could be included. Because MIDI files lose some of the music's subtler qualities, some musical properties were rendered inapplicable to our MIDI-based system. These properties were thus excluded from the model. Expanding our system to incorporate richer file formats, as discussed above, has the added appeal of facilitating the incorporation of more musical properties in our model. Other musical properties were excluded from the model because they required comparison of the performance with a score. Consequently, analysis of digital scores would also help expand the list of musical properties, and enrich our mathematical model.

6.2.3 Visualization

The visualization we created to display information about quantity of emotion represents emotions with colors, and amount of emotion with quantity of color. This system could be enhanced to provide the user with more information. Additional features like pie charts or bar chart at the side could be added to explicitly show the proportions of the emotions present. Exacerbating the difference between large and small amounts of color could also enhance the tool's effectiveness at relaying emotional quantities.

6.3 Applications of Work

Implications of our work extend beyond the production of quantified emotional data. For example, the tool could be used by musicians to check the effects of their playing. If a performer wanted to express a particular emotion in an excerpt, he could record himself and feed the recording to our emotional quantification system. According to the feedback the system provides, he would know if his intentions match his emotional impact. The system could also be used to create music from a set of emotional values. Given a set of quantified emotions, an inverse mapping to musical properties could be created. These properties could then be used as guidelines for composition. Other systems for measuring emotional responses to music could benefit from our tool. If other sources of data can produce quantified emotional values, our visualization system can be used to display their findings as well. For instance, neurological measurements of emotions in the brain while listening to music could use a visualization system like ours to explore its findings. The elegance of our system and the relative independence of its contributions provides opportunity for powerful applications of the work presented in this thesis.

Bibliography

[1] S. Abdallah, K. Noland, M. Sandler, M. Casey, and C. Rhodes. Theory and evaluation of a Bayesian music structure extractor. In *Proc. International Conference on Music Information Retrieval (ISMIR)*. Citeseer, 2005.

[2] J.J. Aucouturier and M. Sandler. Segmentation of musical signals using hidden Markov models. *Preprints-Audio Engineering Society*, 2001.

[3] J.P. Bachorik, M. Bangert, P. Loui, K. Larke, J. Berger, R. Rowe, and G. Schlaug. Emotion in Motion: Investigating the Time-Course of Emotional Judgments of Musical Stimuli. *Music Perception*, 26(4):355–364, 2009.

[4] T. Bergstrom and K. Karahalios. Seeing more: Visualizing audio cues. *Human-Computer Interaction–INTERACT 2007*, pages 29–42, 2007.

[5] E. Bigand, S. Vieillard, F. Madurell, J. Marozeau, and A. Dacquet. Multidimensional scaling of emotional responses to music: The effect of musical expertise and of the duration of the excerpts. *Cognition & Emotion*, 19(8):1113–1139, 2005.

[6] C.J. Boyatzis and R. Varghese. Children's emotional associations with colors. *Journal of Genetic Psychology*, 155(1):77–85, 1994.

[7] C.H. Chen, M.F. Weng, S.K. Jeng, and Y.Y. Chuang. Emotion-based music visualization using photos. *Advances in Multimedia Modeling*, pages 358–368, 2008.

[8] E. Chew. Regards on two regards by Messiaen: Post-tonal music segmentation using pitch context distances in the spiral array. *Journal of New Music Research*, 34(4):341–354, 2005.

[9] E. Chew and A.R.J. Francois. MuSA. RT: music on the spiral array. real-time. In *Proceedings of the eleventh ACM international conference on Multimedia*, page 449. ACM, 2003.

[10] L. Dalhuijsen and L. Velthoven. MusicalNodes. http://www.musicalnodes.com/, 2007.

[11] R. D'Andrade and M. Egan. The colors of emotion. *American ethnologist*, pages 49–63, 1974.

[12] R.B. Dannenberg and N. Hu. Pattern discovery techniques for music audio. *Journal of New Music Research*, 32(2):153–163, 2003.

[13] B.H. Detenber and B. Reeves. A bio-informational theory of emotion: Motion and image size effects on viewers. *Journal of Communication*, 46(3):66–84, 1996.

[14] S. Dixon, F. Gouyon, and G. Widmer. Towards characterisation of music via rhythmic patterns. In *Proc. ISMIR*, volume 5. Citeseer, 2004.

[15] Y. Feng, Y. Zhuang, and Y. Pan. Popular music retrieval by detecting mood. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 375–376. ACM, 2003.

[16] J. Foote. Automatic audio segmentation using a measure of audio novelty. In *Proceedings of IEEE International Conference on Multimedia and Expo*, volume 1, pages 452–455, 2000.

[17] D. Halliday, R. Resnick, and J. Walker. *Fundamentals of Physics*, volume 1. John Wiley and Sons, Inc., 7 edition, 2005.

[18] K. Hevner. Expression in music: a discussion of experimental studies and theories. *Psychological Review*, 42(2):186–204, 1935.

[19] K. Hevner. Experimental studies of the elements of expression in music. *The American Journal of Psychology*, 48(2):246–268, 1936.

[20] R. Hiraga. Case study: a look of performance expression. In *Proceedings of the conference on Visualization'02*, page 504. IEEE Computer Society, 2002.

[21] R. Hiraga and N. Matsuda. Visualization of music performance as an aid to listener's comprehension. In *Proceedings of the working conference on Advanced visual interfaces*, pages 103–106. ACM, 2004.

[22] R. Hiraga, R. Mizaki, and I. Fujishiro. Performance visualization: a new challenge to music through visualization. In *Proceedings of the tenth ACM international conference on Multimedia*, page 242. ACM, 2002.

[23] R. Hiraga, F. Watanabe, and I. Fujishiro. Music learning through visualization. *Proc. of WEB Delivering Music*, 2002.

[24] M.B. Holbrook and M.P. Gardner. Illustrating a dynamic model of the mood-updating process in consumer behavior. *Psychology and Marketing*, 17(3):165–194, 2000.

[25] F.S. Howes. Emotion. *Music and Letters*, 1:45–57, 1924.

[26] G. Husain, W.F. Thompson, and E.G. Schellenberg. Effects of musical tempo and mode on arousal, mood, and spatial abilities. *Music Perception*, 20(2):151–171, 2002.

[27] E.J. Isaacson. Content visualization in a digital music library. In *Third International Workshop on Information Visualization Interfaces for Retrieval and Analysis (IVIRA) at JCDL*. Citeseer, 2003.

[28] P.N. Juslin. *A functionalist perspective on emotional communication in music performance*. Acta Universitatis Upsaliensis, 1998.

[29] P.N. Juslin and P. Laukka. Communication of emotions in vocal expression and music performance: Different channels, same code?. *Psychological Bulletin*, 129(5):770–814, 2003.

[30] N. Kaya and H.H. Epps. Relationship between color and emotion: A study of college students. *College student journal*, 38(3):396–406, 2004.

[31] J.T. Kellaris and R.J. Kent. An exploratory investigation of responses elicited by music varying in tempo, tonality, and texture. *Journal of Consumer Psychology*, 2(4):381–402, 1993.

[32] P. Knees, M. Schedl, T. Pohle, and G. Widmer. An innovative three-dimensional user interface for exploring music collections enriched with meta-information from the web. In *ACM Multimedia*, pages 17–24. Citeseer, 2006.

[33] C.L. Krumhansl. Music: A link between cognition and emotion. *Current Directions in Psychological Science*, 11(2):45, 2002.

[34] J. Langner and W. Goebl. Visualizing expressive performance in tempo-loudness space. *Computer Music Journal*, 27(4):69–83, 2003.

[35] R.T. Laudon. The Elements of Expression in Music: A Psychological View/Element Ekspresije u Glazbi: Psihologijsko Stajalište. *International Review of the Aesthetics and Sociology of Music*, pages 123–133, 2006.

[36] S. Leitich and M. Topf. Globe of Music: Music Library Visualization Using GEOSOM. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR 2007)*. Citeseer, 2007.

[37] A. Lillie. *MusicBox: Navigating the space of your music*. PhD thesis, Massachusetts Institute of Technology, 2007.

[38] D. Liu, L. Lu, and H.J. Zhang. Automatic mood detection from acoustic music data. In *Proceedings of the International Symposium on Music Information Retrieval*, pages 81–7, 2003.

[39] C. McKay and I. Fujinaga. jSymbolic: A feature extractor for MIDI files. In *International Computer Music Conference*. Citeseer, 2006.

[40] L.B. Meyer. *Emotion and meaning in music*. University of Chicago Press, 1956.

[41] L. Mion and G. De Poli. Music Expression Understanding Based on a Joint Semantic Space. *AI* IA 2007: Artificial Intelligence and Human-Oriented Computing*, pages 614–625, 2007.

[42] JB Mitroo, N. Herman, and N.I. Badler. Movies from music: Visualizing musical compositions. In *Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, page 225. ACM, 1979.

[43] F. M
"orchen, A. Ultsch, M. N
"ocker, and C. Stamm. Databionic visualization of music collections according to perceptual distance. In *Proc. of the 6th International Conference on Music Information Retrieval (ISMIR05), London, UK*. Citeseer, 2005.

[44] K. Ohmi. Music Visualization in Style and Structure. *Journal of Visualization*, 10(3):257–258, 2007.

[45] L.C. Ou, M.R. Luo, A. Woodcock, and A. Wright. A study of colour emotion and colour preference. Part I: Colour emotions for single colours. *Color Research & Application*, 29(3):232–240, 2004.

[46] E. Pampalk. Islands of music: Analysis, organization, and visualization of music archives. *Journal of the Austrian Soc. for Artificial Intelligence*, 22(4):20–23, 2003.

[47] B. Pardo and W.P. Birmingham. Algorithms for chordal analysis. *Computer Music Journal*, 26(2):27–49, 2002.

[48] C. Raphael and J. Stoddard. Harmonic analysis with probabilistic graphical models. In *Proc. ISMIR*, pages 177–181. Citeseer, 2003.

[49] A. Rauber, E. Pampalk, and D. Merkl. The SOM-enhanced JukeBox: Organization and visualization of music collections based on perceptual models. *Journal of New Music Research*, 32(2):193–210, 2003.

[50] J. Rothstein. *MIDI: A Comprehensive Introduction.* Madison (WI): A-R Editions, 2 edition, 1995.

[51] C.S. Sapp. Harmonic visualizations of tonal music. In *Proceedings of the International Computer Music Conference*, volume 1001, pages 423–430. Citeseer, 2001.

[52] C.S. Sapp. Improv: Computer/Performer Interaction Programming with MIDI in C++. http://improv.sapp.org/, January 2005.

[53] M. Schedl and T. Pohle. Exploring Music Artists via Descriptive Terms and Multimedia Content. *Semantic Multimedia*, pages 137–148, 2008.

[54] E.D. Scheirer. Tempo and beat analysis of acoustic musical signals. *The Journal of the Acoustical Society of America*, 103:588, 1998.

[55] D. Sleator and D. Temperley. The Melisma Music Analyzer. *nd). Available online at¡ http://www. link. cs. cmu. edu/music-analysis/¿, accessed 30 May 2007.*

[56] J. Sloboda. Music: where cognition and emotion meet. In *Conference Proceedings: Opening the Umbrella; an Encompassing View of Music Education; Australian Society for Music Education, XII National Conference, University of Sydney, NSW, Australia, 09-13 July 1999*, page 175. Australian Society for Music Education, 1999.

[57] R. Taylor, P. Boulanger, and D. Torres. Visualizing emotion in musical performance using a virtual character. In *Smart Graphics*, pages 13–24. Springer, 2005.

[58] D. Temperley and D. Sleator. Modeling meter and harmony: A preference-rule approach. *Computer Music Journal*, 23(1):10–27, 1999.

[59] MM Terwogt and JB Hoeksma. Colors and emotions: preferences and combinations. *The Journal of general psychology*, 122(1):5, 1995.

[60] M. Torrens, P. Hertzog, and J.L. Arcos. Visualizing and exploring personal music libraries. In *Proc. ISMIR*, pages 421–424. Citeseer, 2004.

[61] P. Valdez and A. Mehrabian. Effects of color on emotions. *Journal of Experimental Psychology: General*, 123(4):394–409, 1994.

[62] R. van Gulik, F. Vignoli, and H. van de Wetering. Mapping music in the palm of your hand, explore and discover your collection. In *Proceedings of the 5th International Conference on Music Information Retrieval.* Citeseer, 2004.

[63] T.L. Wu and S.K. Jeng. Probabilistic estimation of a novel music emotion model. *Advances in Multimedia Modeling*, pages 487–497, 2008.

[64] Y.H. Yang, Y.C. Lin, Y.F. Su, and H.H. Chen. Music emotion classification: A regression approach. In *Proc. IEEE Int. Conf. Multimedia and Expo*, pages 208–211, 2007.

[65] Y.H. Yang, C.C. Liu, and H.H. Chen. Music emotion classification: a fuzzy approach. In *Proceedings of the 14th annual ACM international conference on Multimedia*, page 84. ACM, 2006.

[66] Y.H. Yang, Y.F. Su, Y.C. Lin, and H.H. Chen. Music emotion recognition: the role of individuality. In *Proceedings of the international workshop on Human-centered multimedia*, page 22. ACM, 2007.

[67] E. Yılmaz, Y. Çetin, Ç. Erdem, T. Erdem, and M.
"Ozkan. Music Driven Real-Time 3D Concert Simulation. *Multimedia Content Representation, Classification and Security*, pages 379–386, 2006.

Appendix A

Correlations between Musical Properties and Emotions

This appendix provides a series of bar charts used to explore the problem of mapping musical properties to emotions. Each chart displays the number of studies linking various musical property values to a particular emotion. The charts include the musical properties we incorporated in our model, as well as others examined by Juslin and Laukka.

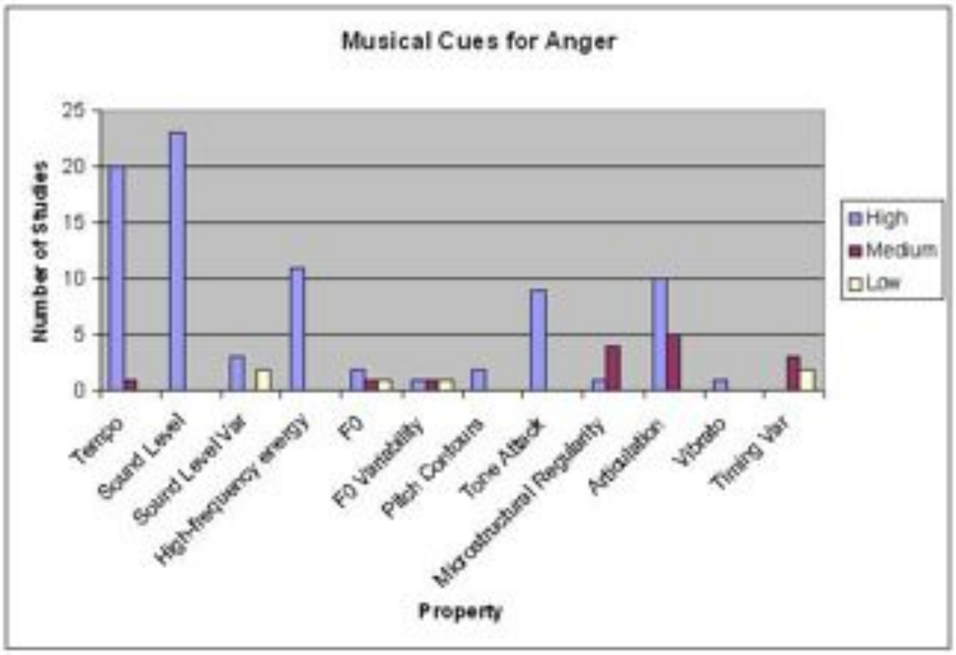

Figure A.1: Correlation between Musical Property Values and Anger

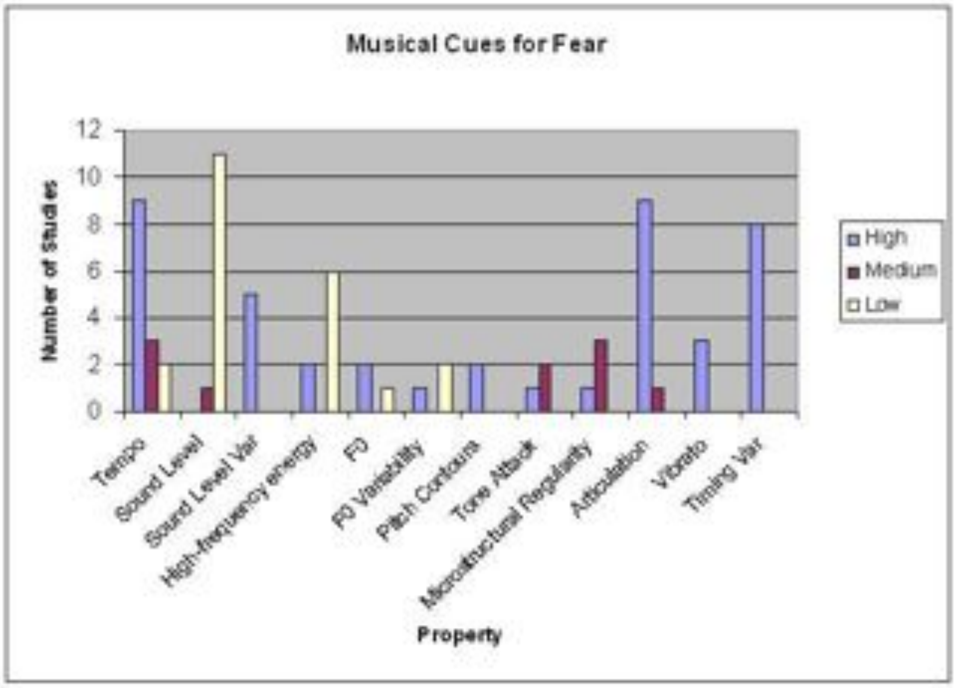

Figure A.2: Correlation between Musical Property Values and Fear

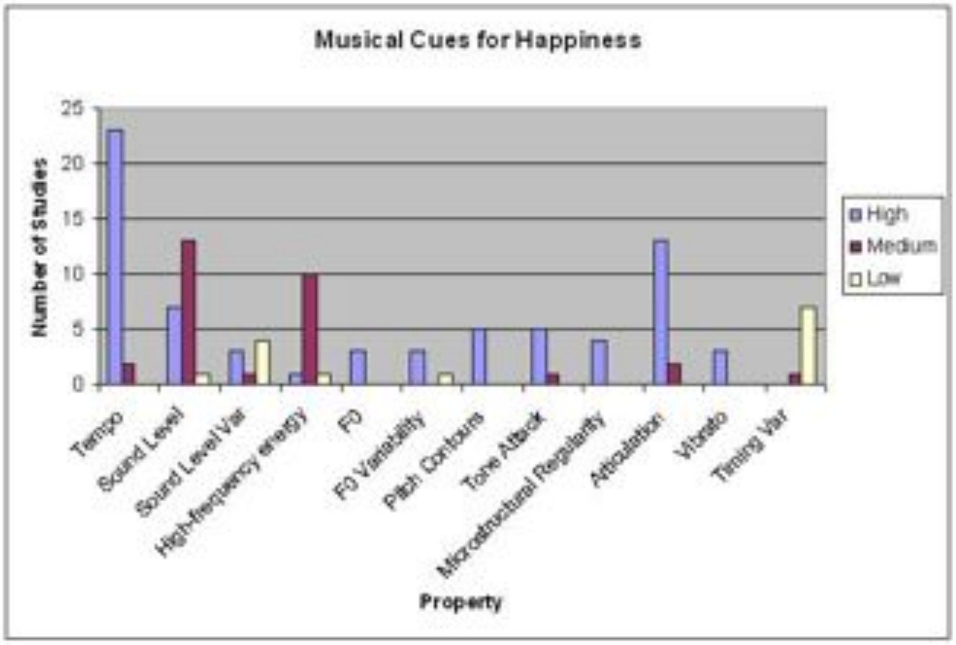

Figure A.3: Correlation between Musical Property Values and Happiness

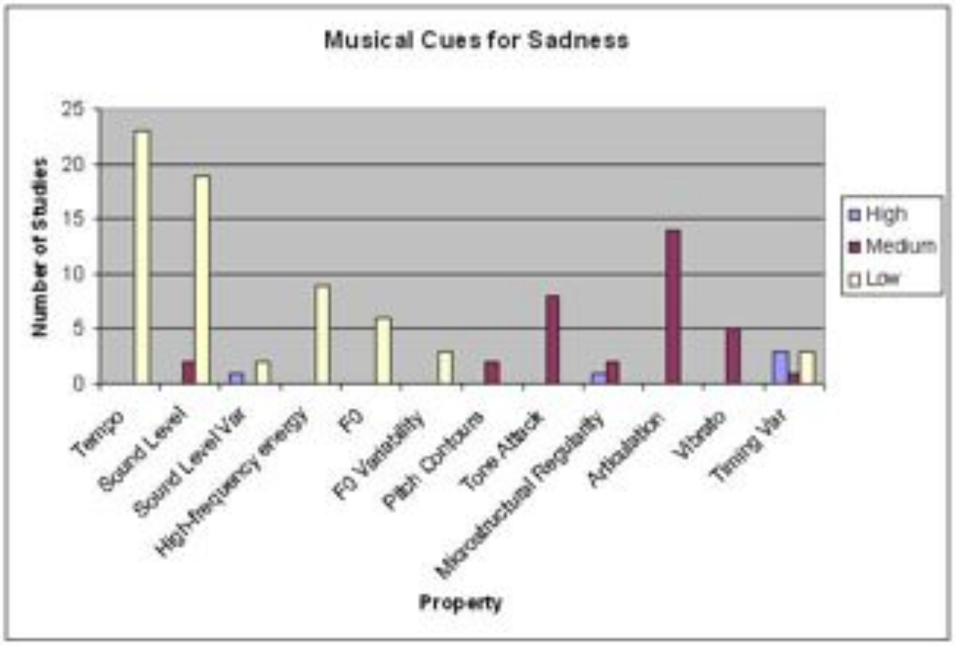

Figure A.4: Correlation between Musical Property Values and Sadness

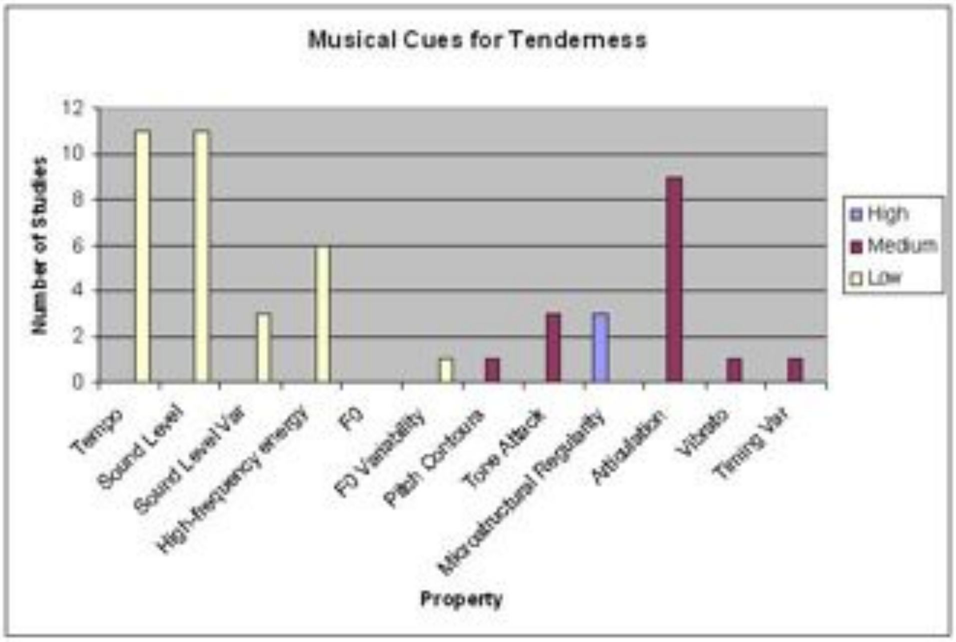

Figure A.5: Correlation between Musical Property Values and Tenderness

Appendix B

Distributions of Musical Properties

This appendix lists the distributions of the values of the musical properties used to define the ordinal categories of musical property values. The unit used for each musical property is the unit used by the jSymbolic property extraction software. Categorical ranges were defined so that each range encompasses an equal number of data points.

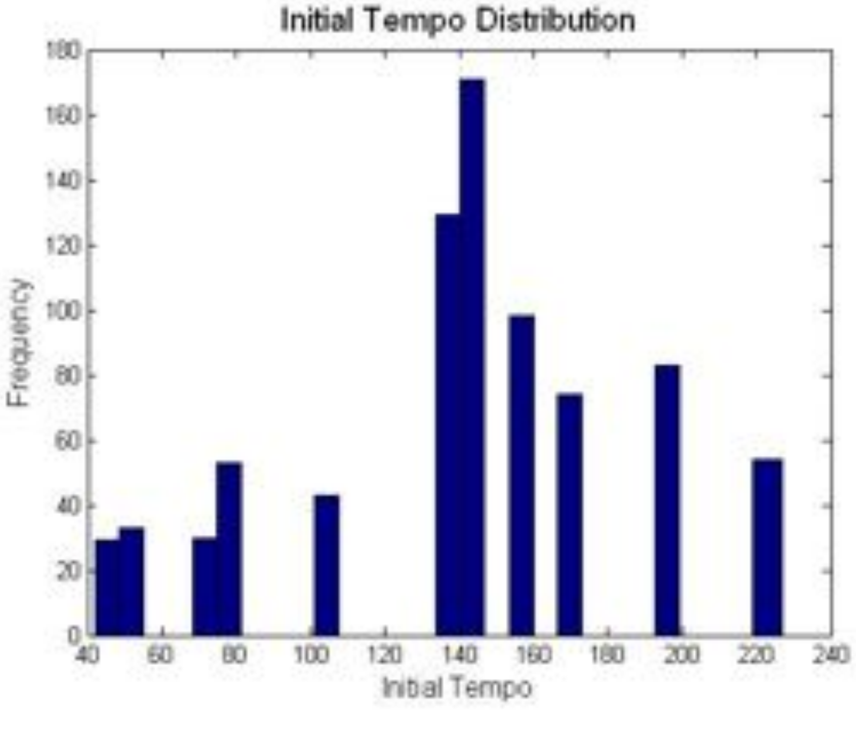

Figure B.1: Distribution of Initial Tempo across 797 Musical Sections

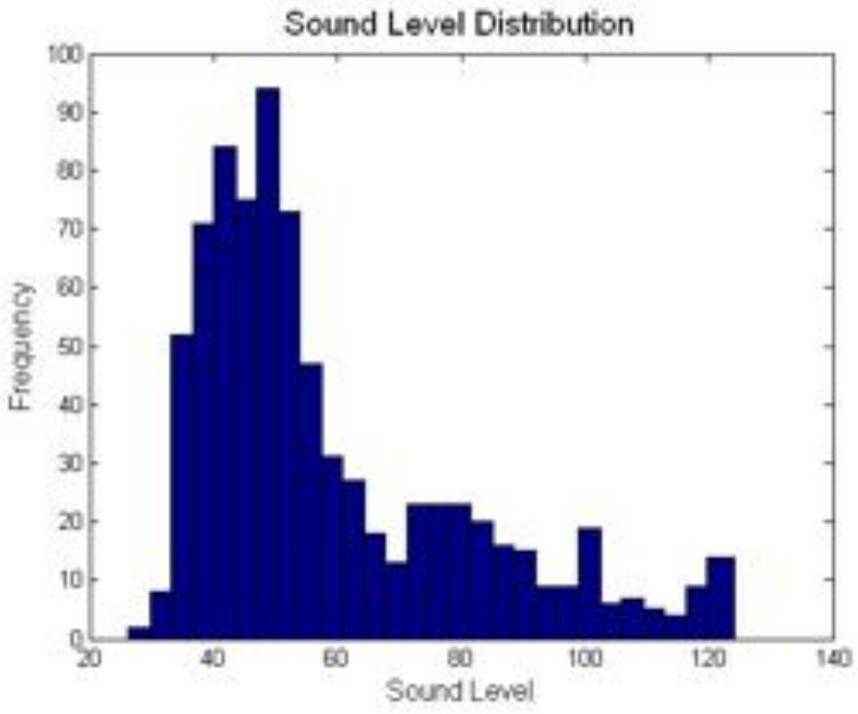

Figure B.2: Distribution of Sound Level across 797 Musical Sections

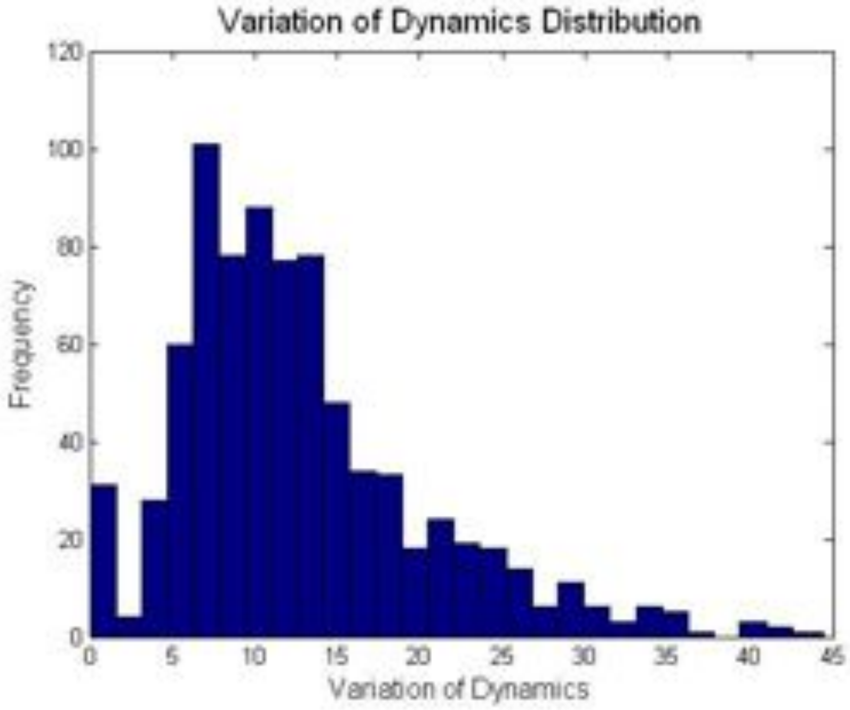

Figure B.3: Distribution of the Variation of Dynamics across 797 Musical Sections

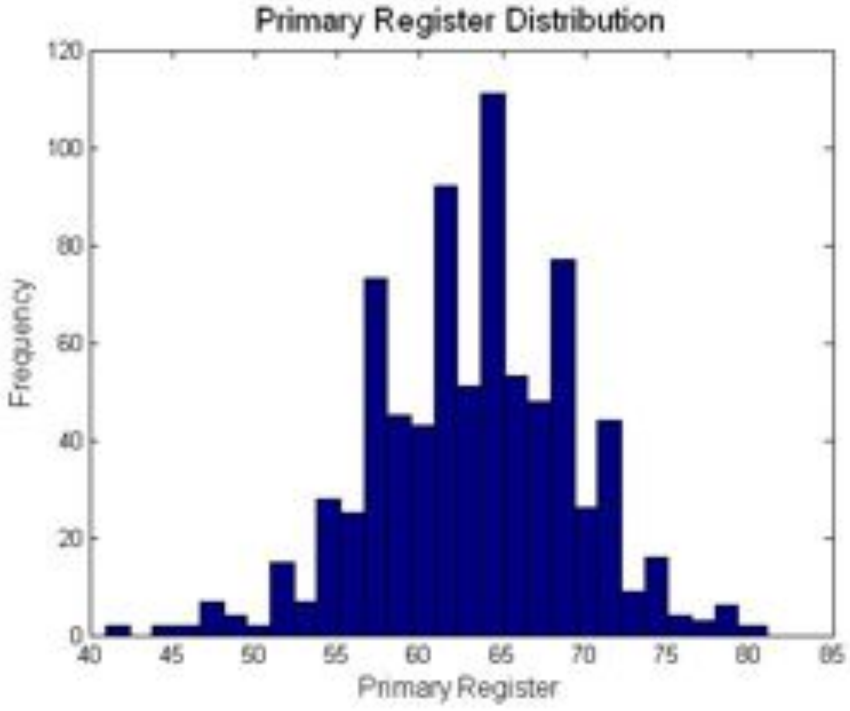

Figure B.4: Distribution of the Primary Register across 797 Musical Sections

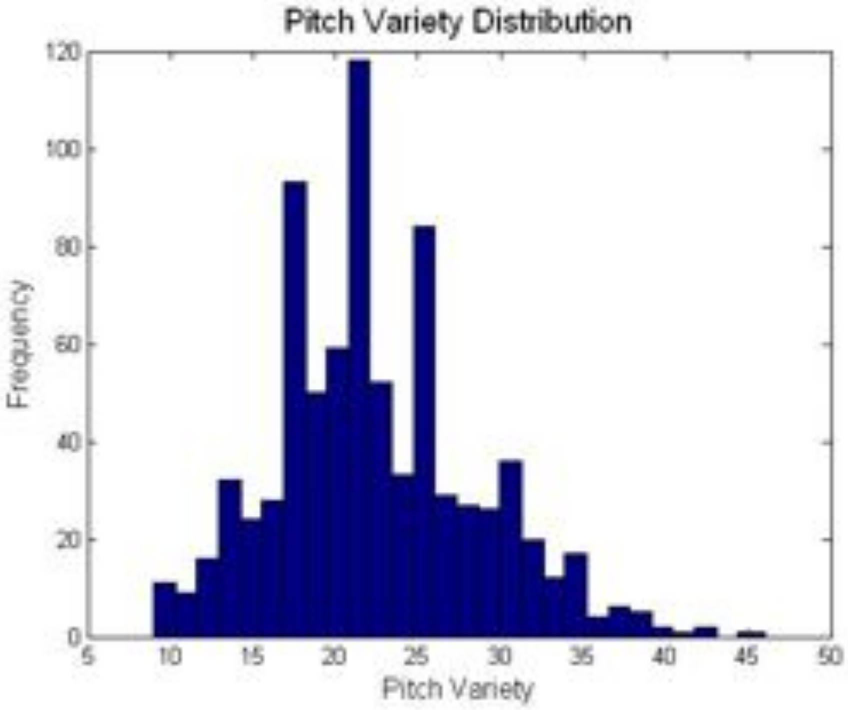

Figure B.5: Distribution of the Pitch Variety across 797 Musical Sections

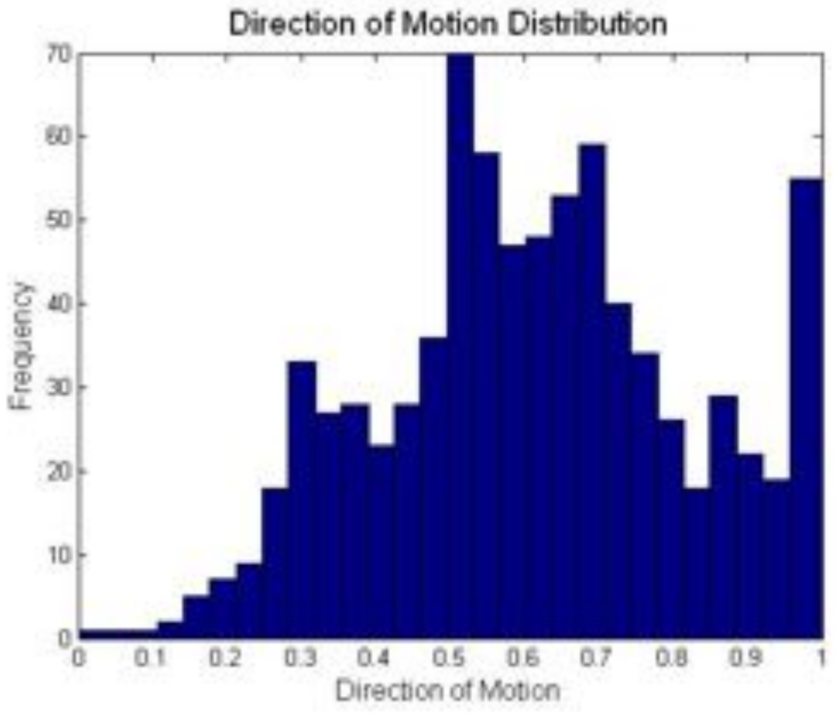

Figure B.6: Distribution of the Direction of Motion across 797 Musical Sections

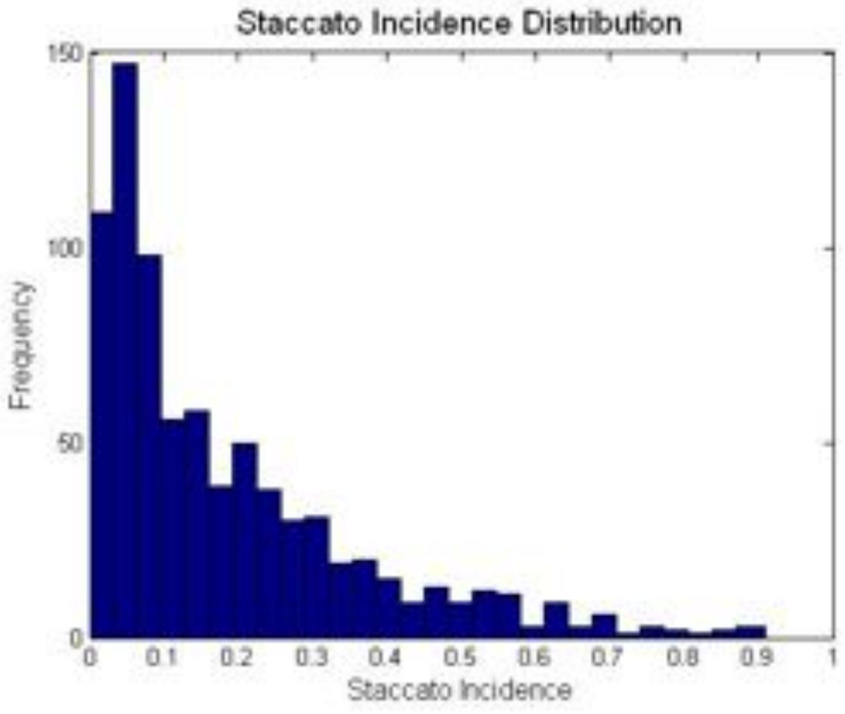

Figure B.7: Distribution of the Staccato Incidence across 797 Musical Sections

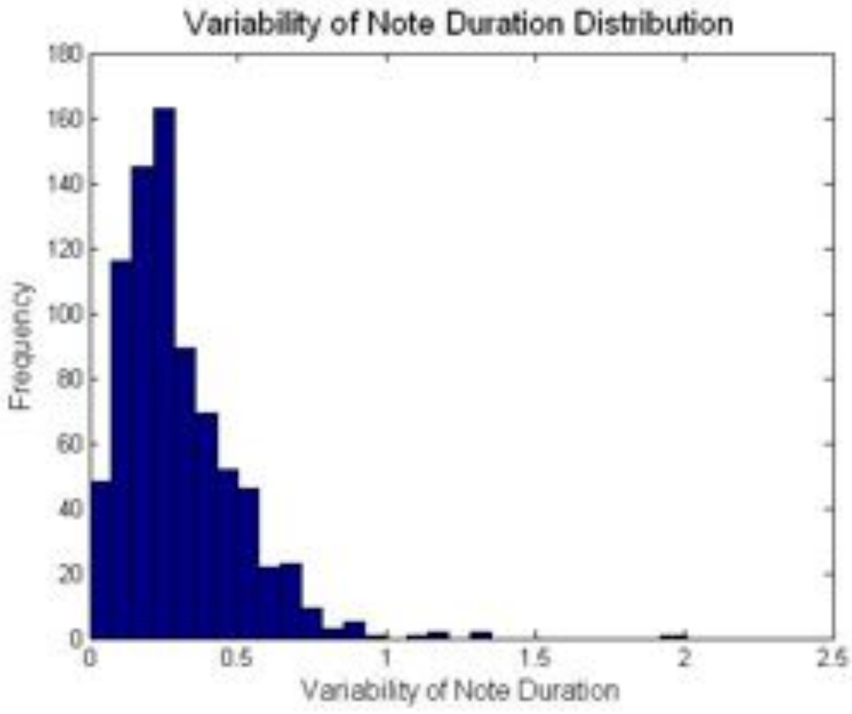

Figure B.8: Distribution of the Variability of Note Duration across 797 Musical Sections

Appendix C

Visualization Code

The following section includes the code for the visualization product. The structure of the code in *ThesisSwirl.pde*, *Particle.pde*, and *ParticleSystem.pde* comes from the MSAFluid library Demo. Sections that were added to make this project work are indicated with "DBragg - start" and "DBragg - end" markers. The *Imports.pde* file was created from scratch.

```
/****************************************************************
 MSAFluid library's copyright information:

 Copyright (c) 2008, 2009, Memo Akten, www.memo.tv
 *** The Mega Super Awesome Visuals Company ***
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *     * Redistributions of source code must retain the above
 *   copyright notice, this list of conditions and the following
 *  disclaimer.
 *     * Redistributions in binary form must reproduce the above
 *  copyright notice, this list of conditions and the following
 *    disclaimer in the documentation and/or other materials
 *   provided
 *  with the distribution.
 *     * Neither the name of MSA Visuals nor the names of its
 *  contributors may be used to endorse or promote products
 *   derived from this software without specific prior written
 *  permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
 * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * ****************************************************************/
```

C.1 ThesisSwirl.pde

```
import msafluid.*;

import processing.opengl.*;
import javax.media.opengl.*;

final float FLUID_WIDTH = 120;

float invWidth, invHeight;    // inverse of screen dimensions
float aspectRatio, aspectRatio2;

MSAFluidSolver2D fluidSolver;

ParticleSystem particleSystem;

PImage imgFluid;

boolean drawFluid = true;

//DBragg - start

float[] hues;
//Coordinates of flows
float x1, x2, x3, x4, x5;
float y1, y2, y3, y4, y5;
float x1_vel, x2_vel, x3_vel, x4_vel, x5_vel;
float y1_vel, y2_vel, y3_vel, y4_vel, y5_vel;
//Variables for keeping track of time
float cumm_passed;
int curr_seg;

//DBragg - end



//960, 640
void setup() {
// use OPENGL rendering for bilinear filtering on texture
    size(960, 640, OPENGL);
//    size(screen.width * 49/50, screen.height * 49/50, OPENGL);
    hint( ENABLE_OPENGL_4X_SMOOTH );    // Turn on 4X antialiasing

    invWidth = 1.0f/width;
    invHeight = 1.0f/height;
    aspectRatio = width * invHeight;
    aspectRatio2 = aspectRatio * aspectRatio;

    // create fluid and set options
    fluidSolver = new MSAFluidSolver2D((int)(FLUID_WIDTH),
     (int)(FLUID_WIDTH * height/width));
    fluidSolver.enableRGB(true).setFadeSpeed(0.003).setDeltaT(0.5)
     .setVisc(0.0001);

    // create image to hold fluid picture
    imgFluid = createImage(fluidSolver.getWidth(),
     fluidSolver.getHeight(), RGB);

    // create particle system
    particleSystem = new ParticleSystem();

    // init TUIO
    initTUIO();

    //DBragg - start//

    //Initialize emotion-color mappings
    hues = new float[5];
    hues[0] = 357; //anger_hue
    hues[1] = 50; //fear_hue
```

```
hues[2] = 120; //happiness_hue
hues[3] = 340; //sadness_hue
hues[4] = 200; //tenderness_hue

//Calculate spacing of streams
//(equidistant from eachother around perimeter)
float total_length = 2*width + 2*height;
float interval = total_length/5;
float midX = width/2;
float midY = height/2;
//Initialize coordiates for 1st stream
float indent = 60;
x1 = indent;
y1 = 0;
//Initialize coordinates for 2nd stream
if (interval + indent <= width){
  x2 = interval + indent;
  y2 = 0;
}else if (interval + indent > width &&
 interval + indent <= width + height){
  x2 = width;
  y2 = interval - width + indent;
}else if (interval + indent > width + height &&
 interval + indent <= 2 * width + height){
  x2 = width - (interval - width - height) - indent;
  y2 = height;
}else{
  x2 = 0;
  y2 = total_length - interval - indent;
}
//Initialize coordiates for 3rd stream
if (2 * interval + indent <= width){
  x3 = 2 * interval + indent;
  y3 = 0;
}else if (2 * interval + indent > width &&
 2 * interval + indent <= width + height){
  x3 = width;
  y3 = 2 * interval - width + indent;
}else if (2 * interval + indent > width + height &&
 2 * interval + indent <= 2 * width + height){
  x3 = width - (2 * interval - width - height) - indent;
  y3 = height;
}else{
  x3 = 0;
  y3 = total_length - 2 * interval - indent;
}
//Initialize coordiates for 5th stream
//-- NOTE: Intuitive locations of 4th and 5th are switched.
if (3 * interval + indent <= width){
  x5 = 3 * interval + indent;
  y5 = 0;
}else if (3 * interval + indent > width &&
 3 * interval + indent <= width + height){
  x5 = width;
  y5 = 3 * interval - width + indent;
}else if (3 * interval + indent > width + height &&
 3 * interval + indent <= 2 * width + height){
  x5 = width - (3 * interval - width - height) - indent;
  y5 = height;
}else{
  x5 = 0;
  y5 = total_length - 3 * interval - indent;
}
//Initialize coordiates for 4th stream
if (4 * interval + indent <= width){
  x4 = 4 * interval + indent;
  y4 = 0;
}else if (4 * interval + indent > width &&
 4 * interval + indent <= width + height){
  x4 = width;
  y4 = 4 * interval - width + indent;
}else if (4 * interval + indent > width + height &&
 4 * interval + indent <= 2 * width + height){
  x4 = width - (4 * interval - width - height) - indent;
  y4 = height;
}else{
  x4 = 0;
  y4 = height - (4 * interval - 2 * width - height) - indent;
}

//Calculate directional velocities
//All velocities point towards the center of the screen.
//All velocities normalized so each has net velocity 1/10.
float scaler = 100;

x1_vel = midX - x1;
y1_vel = midY - y1;
float proportion = scaler * (x1_vel * x1_vel + y1_vel * y1_vel);
x1_vel = x1_vel / sqrt(proportion);
y1_vel = y1_vel / sqrt(proportion);
```

```
    x2_vel = midX - x2;
    y2_vel = midY - y2;
    proportion = scaler * (x2_vel * x2_vel + y2_vel * y2_vel);
    x2_vel = x2_vel / sqrt(proportion);
    y2_vel = y2_vel / sqrt(proportion);

    x3_vel = midX - x3;
    y3_vel = midY - y3;
    proportion = scaler * (x3_vel * x3_vel + y3_vel * y3_vel);
    x3_vel = x3_vel / sqrt(proportion);
    y3_vel = y3_vel / sqrt(proportion);

    x4_vel = midX - x4;
    y4_vel = midY - y4;
    proportion = scaler * (x4_vel * x4_vel + y4_vel * y4_vel);
    x4_vel = x4_vel / sqrt(proportion);
    y4_vel = y4_vel / sqrt(proportion);

    x5_vel = midX + 80 - x5;
    y5_vel = midY - y5;
    proportion = scaler * (x5_vel * x5_vel + y5_vel * y5_vel);
    x5_vel = x5_vel / sqrt(proportion);
    y5_vel = y5_vel / sqrt(proportion);

    //Import neccessary data
    import_emotions();
    import_times();

    //initialize time
    curr_seg = 0;
    cumm_passed = 0;

    //DBragg - end
}

void draw() {

    updateTUIO();
    fluidSolver.update();

    if(drawFluid) {
        for(int i=0; i<fluidSolver.getNumCells(); i++) {
            int d = 2;
            imgFluid.pixels[i] = color(fluidSolver.r[i] * d,
             fluidSolver.g[i] * d, fluidSolver.b[i] * d);
        }
        imgFluid.updatePixels();//  fastblur(imgFluid, 2);
        image(imgFluid, 0, 0, width, height);
    }

    particleSystem.updateAndDraw();

    //DBragg - start
    //Check if we have moved on to the next time segment
    if (millis() / 1000 >= cumm_passed + times[curr_seg]){
      println(millis() + " " + curr_seg);
      cumm_passed = cumm_passed + times[curr_seg];
      curr_seg = curr_seg + 1;
    }

    //Retreive information on the strengths of the emotions
    int scale_input = 100;
    float proportion1 = emotions[curr_seg][0]/scale_input;
    float proportion2 = emotions[curr_seg][1]/scale_input;
    float proportion3 = emotions[curr_seg][2]/scale_input;
    float proportion4 = emotions[curr_seg][3]/scale_input;
    float proportion5 = emotions[curr_seg][4]/scale_input;

    //Random factor of fluid outputs
    float random_range = .1;
    float random_factor_1 = random(-random_range, random_range);
    float x11_vel = proportion1 * x1_vel;
    float y11_vel = proportion1 * y1_vel;
    x11_vel = x11_vel + random_factor_1;
    y11_vel = sqrt(proportion1 - x11_vel * x11_vel);

    float random_factor_2 = random(-random_range, random_range);
    float x21_vel = proportion2 * x2_vel;
    float y21_vel = proportion2 * y2_vel;
    x21_vel = x21_vel + random_factor_2;
    y21_vel = sqrt(proportion2 - x21_vel * x21_vel);

    float random_factor_3 = random(-random_range, random_range);
    float x31_vel = proportion3 * x3_vel;
    float y31_vel = proportion3 * y3_vel;
    y31_vel = y31_vel + random_factor_3;
    x31_vel = -sqrt(proportion3 - y31_vel * y31_vel);

    float random_factor_4 = random(-random_range, random_range);
```

```
    float x41_vel = proportion4 * x4_vel;
    float y41_vel = proportion4 * y4_vel;
    y41_vel = y41_vel + random_factor_4;
    x41_vel = sqrt(proportion4 - y41_vel * y41_vel);

    float random_factor_5 = random(-random_range, random_range);
    float x51_vel = proportion5 * x5_vel;
    float y51_vel = proportion5 * y5_vel;
    x51_vel = x51_vel + random_factor_5;
    y51_vel = -sqrt(proportion5 - x51_vel * x51_vel);

    //Add the streams.
    //Color indicates emotion.
    //Velocity indicates amount of emotion.
    addForce(x1 * invWidth, y1 * invHeight, x11_vel, y11_vel, 0);
    addForce(x2 * invWidth, y2 * invHeight, x21_vel, y21_vel, 1);
    addForce(x3 * invWidth, y3 * invHeight, x31_vel, y31_vel, 2);
    addForce(x4 * invWidth, y4 * invHeight, x41_vel, y41_vel, 3);
    addForce(x5 * invWidth, y5 * invHeight, x51_vel, y51_vel, 4);

    //DBRAGG - end
}

void mousePressed() {
    drawFluid ^= true;
}

void keyPressed() {
    switch(key) {
    case 'r':
        renderUsingVA ^= true;
        println("renderUsingVA: " + renderUsingVA);
        break;
    }
    println(frameRate);
}


// add force and dye to fluid, and create particles
// DBragg - added input "emotion" parameter
void addForce(float x, float y, float dx, float dy, int emotion) {
// balance the x and y components of speed
// with the screen aspect ratio
    float speed = dx * dx  + dy * dy * aspectRatio2;
    if(speed > 0) {
        if(x<0) x = 0;
        else if(x>1) x = 1;
        if(y<0) y = 0;
        else if(y>1) y = 1;

        float colorMult = 5;
        float velocityMult = 30.0f;

        int index = fluidSolver.getIndexForNormalizedPosition(x, y);

        color drawColor;

        colorMode(HSB, 360, 1, 1);
        //DBragg - start
        float hue = hues[emotion];
        //DBragg - end
        drawColor = color(hue, 1, 1);
        colorMode(RGB, 1);

        fluidSolver.rOld[index]  += red(drawColor) * colorMult;
        fluidSolver.gOld[index]  += green(drawColor) * colorMult;
        fluidSolver.bOld[index]  += blue(drawColor) * colorMult;

        particleSystem.addParticles(x * width, y * height, 10);
        fluidSolver.uOld[index] += dx * velocityMult;
        fluidSolver.vOld[index] += dy * velocityMult;
    }
}
```

C.2 Imports.pde

```
\\DBragg - start

String[] emotion_lines;
String[] time_lines;
float[][] emotions;
float[] times;

//Import data on emotions from TSV file emotion_vectors.tsv.
//Store the data in emotions[][].
void import_emotions(){
  emotion_lines = loadStrings("emotion_vectors.tsv");
  emotions = new float[emotion_lines.length][5];
  for (int i = 0; i < emotion_lines.length; i++){
    String[] pieces = split(emotion_lines[i], "    ");
    for (int j = 1; j <= 5; j++){
       emotions[i][j-1] = float(pieces[j]);
    }
  }
}

//Import data on the time segmentation.
//For the original visualization,
//all entries in time_lines.length are 8.31.
void import_times(){
  time_lines = loadStrings("times.tsv");
  times = new float[time_lines.length];
  for (int i = 0; i < time_lines.length; i++){
    times[i] = float(time_lines[i]);
  }
}

\\DBragg - end
```

C.3 Particle.pde

```
class Particle {
    final static float MOMENTUM = 0.5;
    final static float FLUID_FORCE = 0.6;

    float x, y;
    float vx, vy;
    float radius;       // particle's size
    float alpha;
    float mass;

    void init(float x, float y) {
        this.x = x;
        this.y = y;
        vx = 0;
        vy = 0;
        radius = 5;
        alpha  = random(0.3, 1);
        mass = random(0.1, 1);
```

```
}


void update() {
    // only update if particle is visible
    if(alpha == 0) return;

    // read fluid info and add to velocity
    int fluidIndex = fluidSolver.getIndexForNormalizedPosition
     (x * invWidth, y * invHeight);
    vx = fluidSolver.u[fluidIndex] * width * mass
     * FLUID_FORCE + vx * MOMENTUM;
    vy = fluidSolver.v[fluidIndex] * height * mass
     * FLUID_FORCE + vy * MOMENTUM;

    // update position
    x += vx;
    y += vy;

    // bounce of edges
    if(x<0) {
        x = 0;
        vx *= -1;
    }
    else if(x > width) {
        x = width;
        vx *= -1;
    }

    if(y<0) {
        y = 0;
        vy *= -1;
    }
    else if(y > height) {
        y = height;
        vy *= -1;
    }

    // hackish way to make particles glitter when the slow down a lot
    if(vx * vx + vy * vy < 1) {
        vx = random(-1, 1);
        vy = random(-1, 1);
    }

    // fade out a bit (and kill if alpha == 0);
```

```
        alpha *= 0.999;
        if(alpha < 0.01) alpha = 0;

    }


    void updateVertexArrays(int i, FloatBuffer posBuffer,
     FloatBuffer colBuffer) {
        int vi = i * 4;
        posBuffer.put(vi++, x - vx);
        posBuffer.put(vi++, y - vy);
        posBuffer.put(vi++, x);
        posBuffer.put(vi++, y);

        int ci = i * 6;
        colBuffer.put(ci++, alpha);
        colBuffer.put(ci++, alpha);
        colBuffer.put(ci++, alpha);
        colBuffer.put(ci++, alpha);
        colBuffer.put(ci++, alpha);
        colBuffer.put(ci++, alpha);
    }


    void drawOldSchool(GL gl) {
        gl.glColor3f(alpha, alpha, alpha);
        gl.glVertex2f(x-vx, y-vy);
        gl.glVertex2f(x, y);
    }

}
```

C.4 ParticleSystem.pde

```
import java.nio.FloatBuffer;
import com.sun.opengl.util.*;

boolean renderUsingVA = true;

void fadeToColor(GL gl, float r, float g, float b, float speed) {
    gl.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE_MINUS_SRC_ALPHA);
    gl.glColor4f(r, g, b, speed);
    gl.glBegin(GL.GL_QUADS);
    gl.glVertex2f(0, 0);
    gl.glVertex2f(width, 0);
    gl.glVertex2f(width, height);
    gl.glVertex2f(0, height);
    gl.glEnd();
}


class ParticleSystem {
    FloatBuffer posArray;
    FloatBuffer colArray;

    final static int maxParticles = 5000;
    int curIndex;

    Particle[] particles;

    ParticleSystem() {
        particles = new Particle[maxParticles];
        for(int i=0; i<maxParticles; i++) particles[i] = new Particle();
        curIndex = 0;

// 2 coordinates per point, 2 points per particle
//(current and previous)
        posArray = BufferUtil.newFloatBuffer(maxParticles * 2 * 2);
        colArray = BufferUtil.newFloatBuffer(maxParticles * 3 * 2);
    }


    void updateAndDraw(){
        // processings opengl graphics object
        PGraphicsOpenGL pgl = (PGraphicsOpenGL) g;
        GL gl = pgl.beginGL();                // JOGL's GL object

        gl.glEnable( GL.GL_BLEND );             // enable blending
        if(!drawFluid) fadeToColor(gl, 0, 0, 0, 0.05);

// additive blending (ignore alpha)
        gl.glBlendFunc(GL.GL_ONE, GL.GL_ONE);
        gl.glEnable(GL.GL_LINE_SMOOTH);        // make points round
        gl.glLineWidth(1);


        if(renderUsingVA) {
```

```
        for(int i=0; i<maxParticles; i++) {
            if(particles[i].alpha > 0) {
                particles[i].update();
                particles[i].updateVertexArrays(i, posArray,
                 colArray);
            }
        }
        gl.glEnableClientState(GL.GL_VERTEX_ARRAY);
        gl.glVertexPointer(2, GL.GL_FLOAT, 0, posArray);

        gl.glEnableClientState(GL.GL_COLOR_ARRAY);
        gl.glColorPointer(3, GL.GL_FLOAT, 0, colArray);

        gl.glDrawArrays(GL.GL_LINES, 0, maxParticles * 2);
    }
    else {
     // start drawing points
        gl.glBegin(GL.GL_LINES);
        for(int i=0; i<maxParticles; i++) {
            if(particles[i].alpha > 0) {
                particles[i].update();
                // use oldschool renderng
                particles[i].drawOldSchool(gl);
            }
        }
        gl.glEnd();
    }

    gl.glDisable(GL.GL_BLEND);
    pgl.endGL();
}


void addParticles(float x, float y, int count ){
    for(int i=0; i<count; i++) addParticle(x + random(-15, 15),
     y + random(-15, 15));
}


void addParticle(float x, float y) {
    particles[curIndex].init(x, y);
    curIndex++;
    if(curIndex >= maxParticles) curIndex = 0;
}
```

```
}
```