AN ABSTRACT OF THE THESIS OF

Soraya J. Matos G. for the degree of Master of Science in Electrical and Computer Engineering presented on March 13, 1997. Title: A Graphic User Interface for Monophonic Music Analysis.

Abstract approved: Redacted for Privacy

Shih-Lien Lu

A Graphic User Interface is developed to determine the existence of a particular sequence of piano notes within a monophonic sound waveform. Such waveforms are recorded within the Graphic User Interface and then passed to the monophonic analysis engine. The first phase of analysis segments the PCM sound data to localize the potential note locations. The second phase of analysis takes the segmented note locations, moves them to the frequency-domain, and utilizes a probabilistic identification process to determine the identity of each note. Two sound files can be processed together to decide if any notes are common between them. A frequency-based comparison model allows flexibility in finding overlap between the files. Theoretical concepts are visualized using the Graphic User Interface making it a tool for developing additional insight into the analysis of music.

A Graphic User Interface for Monophonic Music Analysis

by

Soraya J. Matos G.

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented March 13, 1997
Commencement June 1997

Master of Science thesis of Soraya J. Matos G. presented on March 13, 1997

APPROVED:

Redacted for Privacy

Major Professor, representing Electrical and Computer Engineering

Redacted for Privacy

Head of Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Soraya J. Matos G., Author

ACKNOWLEDGMENT

My faithful thanks to God, who guides me and gives me spiritual strength.

I give special thanks to Fundación Gran Mariscal de Ayacucho (FGMA) for recognizing my efforts in my undergraduate studies and granting me a scholarship to pursue my graduate studies outside of my country.

I also give my special thanks to Dr. Lu, my major professor, who sparked me to choose the topic of this work. Particular thanks to Dr. Lee who lifted my confidence to start my studies in Computer Engineering, in a meeting that I will not forget. Thanks to Dr. Jack Kenney and Dr. Joseph Nibler for dedicating part of their time to participate in my graduate committee.

Thanks to Otto Gygax who gave me the opportunity to be a member of the Computer Support Staff at ECE, where I became familiar with computer operating systems and networking.

To all my friends, especially Patricia Zárate, for sharing wonderful moments and making me feel at home.

To the people that I loved the most, my family, who followed my progress and gave me their unconditional support to continue.

To my faithful and adorable husband, Julian B. Sessions, for giving me his precious kindness, for sharing his valuable knowledge, for supporting me in good and bad moments, and for being there at all times.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

LIST OF TABLES

LIST OF APPENDIX FIGURES

DEDICATION

To my mother, and my husband

1. INTRODUCTION

Chapter one serves as an introduction to this thesis. It describes the motivation, terminology, objectives, and organization.

1.1 Motivation

Advances in technology have facilitated information accessibility. With the vast amount of information available, automated methods are necessary to locate and identify relevant topics efficiently. Searching for keywords, for example, is quite common in a research library environment. Techniques for examining numeric and text databases, however, are not necessarily directly applicable to other forms of information.

One form of storage which merits special attention is that of musical recording. When music is sampled and converted to a digital format, each musical representation is unique. Digital storage of such recordings requires much more space than text does. The extra memory is not, in itself, a difficulty. The problem arises since the recordings are large as well as unique. For example, consider that a musical sound is digitized and stored in a database. Now suppose that a part of the original sound is reproduced and digitized a second time. It can not be guaranteed to find the second version as a subset of

the original version. Even if the extent of the search is known beforehand, it is highly unlikely to locate the second version represented exactly within the original database.

As a solution to this problem, consider the equivalence of words in a text document with musical notes in digitized sound data. Musical notes can be the fundamental building blocks in sound data as words are in a text document. Thus, if it is possible to characterize the musical data in terms of notes, it could be possible to accomplish matching in simpler terms. It is the focus of this thesis to investigate these identification and matching techniques for digital musical data.

As a practical use, I have often found myself wondering to which song belongs a part that I remember. Others share this curiosity too. This everyday activity is a complex process. My motivation is to implement this process that seems to be so natural.

When we listen to musical sounds, we can easily place them in a determined song. The work behind the recognition, for example, belongs primarily to the ear which processes all the sounds we hear. Our ear by nature operates as a temporal and frequency analyzer which is able to recognize different frequencies, timbres, and pitches [1]. Research in music perception has become more advanced in the last few decades. Artificial music perception involves simulation of the entire human auditory system, from the external ear

up to the brain cortex. Recognition, or identification, then became the first part of my research.

In order to compare a portion of a song with the song itself, I have to analyze the song in terms of musical notes played. Next I must choose a matching algorithm that will determine the existence of the desired portion. This process can be visually presented with a computer program. This motivated me to create the user interface which allows a person to interact with the analytical sections of this work. The contribution of three major areas: musical note recognition, searching techniques, and the creation of a user interface, sufficiently captured my attention to dedicate time to researching them.

1.2 Musical Sounds

There are a wide variety of musical sounds. They come from traditional instruments, for example, the piano, guitar, and flute, as well as modern devices such as the synthesizer. In the development of this thesis, a piano was chosen as the basic test instrument. Certain assumptions were made based on this choice. One assumption is the range of notes that the instrument can play. Since the piano is a western musical instrument, it operates on a scale of 12 notes per octave for a total of 88 notes. An octave is defined as a frequency range between f_1 and f_2 such that f_2 is twice f_1. Another assumption is the

existence of a standard specification for those octaves and notes. For more information on the piano specifications, please refer to the Appendix.

Musical sounds are composed of complex tones. A complex tone consists of a fundamental frequency tone combined with its harmonics. The harmonics are integer multiples of the fundamental frequency. The fundamental frequency is also called the first harmonic.

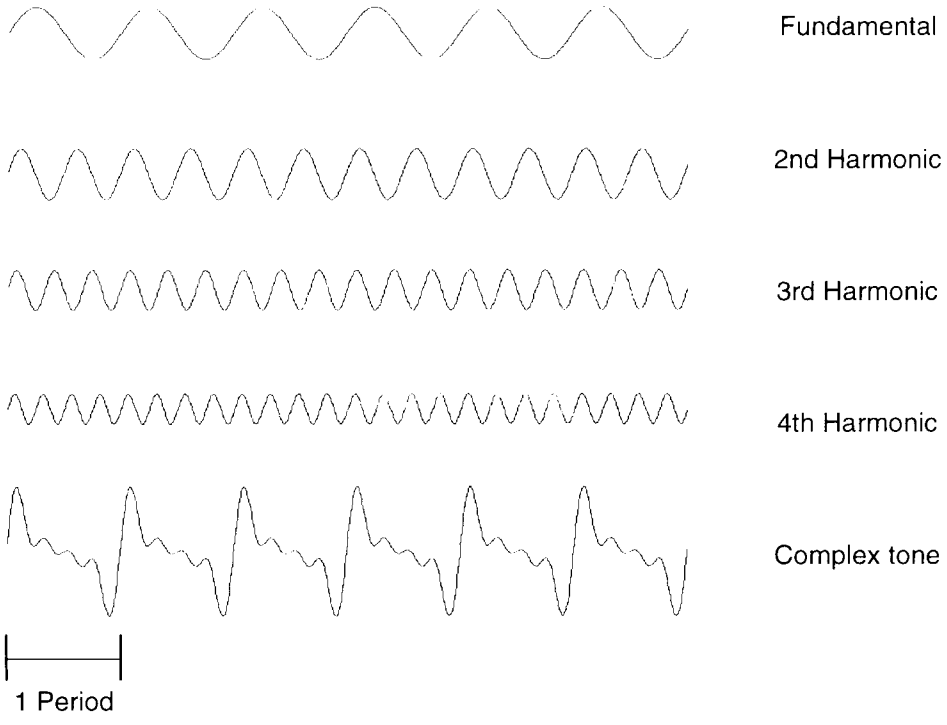


Figure 1.1
Example of a Complex Tone

Consider the example shown in figure 1.1. The complex periodic tone contains four harmonics. In this example, the fundamental frequency is 110 Hz;

therefore, the second harmonic is 220 Hz, the third harmonic is 330 Hz, and the fourth harmonic is 440 Hz. The complex tone is the sum of the fundamental and its harmonics. The periodicity of the complex tone results from the fact that the harmonics are integer multiples of the fundamental frequency.

Psychoacoustic study considers the psychological and acoustical aspects of music. It models the way humans perceive sounds as subjective responses to frequencies, amplitude, duration, wave shape, and location of the sound [2]. In psychoacoustic terminology, these parameters are respectively called: pitch, loudness, timbre, and auditory localization. Psychoacoustics has been incorporated in many studies related to speech analysis, and currently in music analysis [1]. References for psychoacoustic study can be found in [3] [4] [5] [6].

Pitch of a sound is a subjective response to frequency. Loudness is related to the amount of energy received by the ear, which on other hand, refers to the amount of energy a musical sound generates when played. Timbre is the perception of the shape of a sound waveform, also known as the envelope. It reflects the characteristic tone quality of a particular class of sounds. According to Helmhotz [7], a sound contains at least a rise time, or attack, steady-state, and a decay time as shown in figure 1.2. His characterization work is considered the foundation for modern studies of timbre. He concluded that timbre is intrinsically related to the spectral description of a sound. The last physical parameter mentioned is auditory localization, which is defined as the human perception of the placement of the sound source.

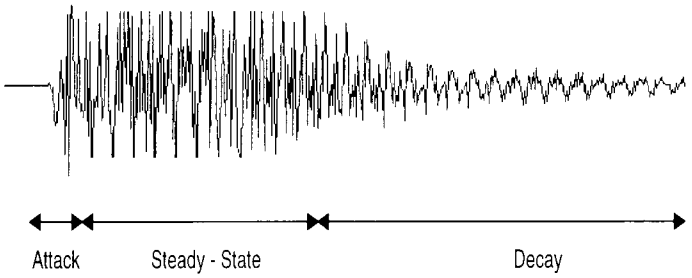


Figure 1.2
Three Principal Segments of a Tone According to Helmhotz

Another aspect of musical sounds is related to the way they are played. This primary classification divides musical sounds into two groups, monophonic and polyphonic. A monophonic sound occurs when notes are played one-at-a-time with little or no overlap, and are generated from a single source. The monophonic model requires that one note decays almost completely before the next one is played. This last statement is important to remember since musical notes are easily overlapped. It takes special attention not to play the next note before the current note has sufficiently decayed.

An example of a monophonic sound is shown in figure 1.3. The note sequence was C_4-D_4-E_4-F_4-G_4-A_4 on the piano scale. Overlapping between harmonics of different tones produces polyphonic musical sounds. In general, polyphonic sounds are a combination of sounds that come from different instruments as well as sounds that are played together, such as musical chords.

Figure 1.3
Example of a Piano Monophonic Sound

1.3 Objective of this Work

The work developed during this thesis focused on creating a Graphic User Interface to analyze monophonic musical sounds. The main objective is to create an environment where a person can interact naturally with the analysis of monophonic musical sounds from the piano. A secondary objective was to create a useful, real-world computer application. The user interface became the physical link in transforming abstracts ideas into a concrete presentation. In more detail this thesis covers the following objectives:

- Analyze monophonic musical sounds in the context of signal processing.
- Apply appropriate matching techniques to determine the existence of one monophonic musical sound within another monophonic musical sound.
- Build a Graphic User Interface to facilitate the analysis of monophonic musical sounds.

1.4 Thesis Organization

Chapter two presents a background in signal processing beginning with the theory of Fourier Series and ending with a complete analysis of a decimation-in-time Fast Fourier Transform (FFT). Chapter three describes the analysis of a monophonic musical sound to transform digital music into discrete musical notes. Chapter four studies some searching techniques that can be applied to determine if there is an exact match or approximate match between two sets of musical notes. Chapter five describes the implementation of a Graphic User Interface written for the Microsoft Windows NT environment. The purpose of the GUI is to graphically show the analysis of monophonic musical sounds and to illustrate the results of the matching algorithm. Chapter six is dedicated to conclusions and some words regarding future work.

2. BACKGROUND

As seen in chapter one, musical sounds are composed of complex tones. A complex tone contains a fundamental frequency and its harmonics. There exists a one-to-one mapping between a note and its fundamental frequency. For instance, the musical note known as A_3 on a piano keyboard has a fundamental frequency of 110 Hz. A complete key-frequency table is given in the Appendix. Thus, an analysis of musical sounds can involve an examination of the frequency spectrum. This chapter presents a overview of basic theory in signal processing to provide the reader with the necessary background to understand the following chapters. The goal on this chapter is to present an overview of the digital signal processing topics that will be applied to the analysis of monophonic music. This overview starts with the classical Fourier Series and ends with an implementation of the Fast Fourier Transform (FFT), highlighting the bit reversal algorithm which is applied either before or after the computation of the FFT.

2.1 Fourier Series

Fourier Series is a representation of a periodic signal in terms of a weighted sum of harmonically related sinusoids. It can also be expressed in terms of complex exponentials. For instance, let $x(t)$ be a continuous-time

periodic signal with fundamental frequency F_0. Thus, $x(t)$ can be rewritten using Fourier Series as in equation 2.1.

$$x(t) = \sum_{-\infty}^{+\infty} c_k e^{j2\pi k F_0 t} \qquad \text{(eq. 2.1)}$$

where c_k, the coefficients of the series, are given by,

$$c_k = \frac{1}{T_p} \int_{T_p} x(t) e^{-j2\pi k F_0 t} dt \qquad \text{(eq. 2.2)}$$

There are several conditions that must be satisfied in order to guarantee the existence of the Fourier Series. These are known as Dirichlet conditions [8]. The conditions are the following:

a. The signal $x(t)$ has a finite number of discontinuities in any period.

b. The signal $x(t)$ contains a finite number of maxima and minima during any period.

c. The signal $x(t)$ is absolutely integrable in any period, e.g., $\int |x(t)|\, dt < \infty$. Therefore, the energy of the signal $\int |x(t)|^2 dt < \infty$ is finite.

Recall the example of the complex tone given in the previous chapter, section 1.2. This complex tone may be represented as a Fourier Series consisting of four sinusoids.

2.2 Fourier Transform

The Fourier Transform is an extension of Fourier Series which is applied to non-periodic signals. This transform is derived from the Fourier Series by

letting the period of the signal tend to ∞ [8]. The Fourier Transform is defined in equation 2.3,

$$X(F) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi Ft} dt \qquad \text{(eq. 2.3)}$$

and the Inverse Fourier Transform,

$$x(t) = \int_{-\infty}^{\infty} X(F) e^{j2\pi Ft} dF \qquad \text{(eq. 2.4)}$$

In order to guarantee the existence of *X(F)*, *x(t)* has to satisfy the Dirichlet conditions mentioned before.

2.3 Discrete Fourier Transform

For the convenience of digital simulation, it is necessary to manipulate the continuous signal in discrete fashion. This is not possible with the strict definition of the Fourier Transform. The Discrete Fourier Transform (DFT) is known as a pair of transforms that map a discrete-time signal to the discrete frequency-domain, and vice-versa. DFT analysis combines Fourier Series theory with sampled data analysis by Shannon [9].

The first step to determine the DFT is to convert the continuous-time signal *x(t)* into a discrete signal *x(n)* by sampling with a frequency, *fs*. Thus,

$$x(n) = \begin{cases} x(t) \quad at \quad t = \dfrac{n}{fs} \\ 0 \quad otherwise \end{cases}$$

According to Shannon's Sampling Theory [10] , the signal *x(t)* must be sampling at $fs \geq 2B$, where B is the highest frequency encountered in *x(t)*, so that the signal *x(t)* can be recovered from its time samples.

Next I present a summary for the derivation of the DFT. I recommend a paper written by J. R. Deller [11] where he describes the meaning of the DFT in an easy manner. Other literature [8] [12] was also useful in obtaining this formulation.

By definition, the Fourier Transform of a non periodic discrete time signal *x(n)* is:

$$X(\omega) = \sum_{-\infty}^{\infty} x(n) e^{-j\omega n} \qquad \text{(eq. 2.5)}$$

which is continuous in time. This transform is periodic with period 2π since,

$$X(\omega + 2\pi) = \sum_{-\infty}^{\infty} x(n) e^{-j(\omega + 2\pi k)n} = \sum_{-\infty}^{\infty} x(n) e^{-j\omega n} e^{-j2\pi k} = \sum_{-\infty}^{\infty} x(n) e^{-j\omega n} = X(\omega)$$

Discrete signals in the time-domain have a periodic representation in the frequency-domain and vice-versa. Since *X(w)* is analog, it must be sampled to operate with discrete signals in the frequency-domain. The signal is sampled *N* times within 2π, which is the fundamental period. Thus, *X(w)* is sampled at $\omega = \frac{2\pi k}{N}$. Since *X(w)* is periodic and not bounded in time, it can be represented as an infinite summation of its periodic segments shifted in time.

$$X\left(\frac{2\pi k}{N}\right) = \sum_{l=-\infty}^{\infty} \sum_{n=0}^{N-1} x(n-lN) e^{-j2\pi k(n-lN)/N} = \sum_{l=-\infty}^{\infty} \sum_{n=0}^{N-1} x(n-lN) e^{-j2\pi kn/N}$$

$$X\left(\frac{2\pi k}{N}\right) = \sum_{n=0}^{N-1} \left[\sum_{l=-\infty}^{\infty} x(n-lN) \right] e^{-j2\pi k/N} \qquad \text{(eq. 2.6)}$$

The expression enclosed in brackets in eq. 2.6 is a periodic version of *x(n)*. This means that by having *N* samples of the spectrum of *x(n)*, a periodic version of *x(n)*, $x_p(n)$, can be recovered. The data sequence *x(n)* can be recovered from its frequency spectrum only if there is no aliasing in the time-domain.

Applying Fourier Series to $x_p(n)$,

$$x_p(n) = \sum_{l=-\infty}^{\infty} x(n-lN) = \sum_{k=0}^{N-1} c_k e^{j2\pi kn/N} \quad \text{and}$$

$$c_k = \frac{1}{N} \sum_{n=0}^{N-1} x_p(n) e^{-j2\pi kn/N} = \frac{1}{N} X\left(\frac{2\pi k}{N}\right)$$

therefore,

$$x_p(n) = \frac{1}{N} \sum_{k=0}^{N-1} X\left(\frac{2\pi k}{N}\right) e^{2\pi kn/N}$$

since,

$$x_p(n) = \begin{cases} x(n) & 0 \le n \le L-1 \\ 0 & L \le n < N \end{cases}$$

We can reconstruct *x(n)* from its periodic representation by restricting the limits of the summation to *N-1*. Thus,

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X\left(\frac{2\pi k}{N}\right) e^{2\pi kn/N} \qquad \text{(eq. 2.7)}$$

Since $x_p(n)$ equals $x(n)$ for $0 \le n \le L-1$, equation 2.5 can be rewritten as follows,

$$X(\omega) = \sum_{n=0}^{L-1} x(n) e^{-j\omega n} \qquad \text{(eq. 2.8)}$$

Moreover, extending the upper limit to N will not affect the summation since $x_p(n)$ is zero for $n > L-1$. Sampling the frequency spectrum at $\omega_k = \frac{2\pi k}{N}$, equation 2.8 becomes,

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \qquad \text{(eq. 2.9)}$$

Equations 2.8 and 2.9 are known as the pair of N-point DFT transforms.

2.4 Fast Fourier Transform

The Fast Fourier Transform algorithm was developed in order to compute the N-point DFT in an efficient manner. The FFT reduces the computational complexity of the DFT algorithm from $O(N^2)$ to $O(N*log(N))$. This reduction in computational requirements has made the DFT very practical and especially important for this thesis. Recall the definition of a N-point DFT for $0 \le k \le N-1$,

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \qquad \text{(eq. 2.9 repeated)}$$

Substituting the exponential, $W_N = e^{-j\frac{2\pi}{N}}$,

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}$$

If the sequence *x(n)*, is not complex, this transform requires *2N* real multiplications and *N-1* complex additions (*4N-2* real additions) for each *k*. It means that to compute a *N*-point FFT, this transform requires $2N^2$ real multiplications, and $4N^2$-*2N* real additions. This represents a heavy computation load, $O(N^2)$

The main purpose of the FFT is to reduce the computational complexity, which in turn, decreases the computational time. This is achieved by breaking the DFT into smaller DFTs and by taking advantage of the symmetry and periodicity properties of W_N [8].

These properties are:

$$W_N^{k+N/2} = -W_N \qquad \text{(a)}$$

$$W_N^{k+N} = W_N \qquad \text{(b)}$$

$$W_N^{lk} = W_{\frac{N}{l}}^{k} \qquad \text{(c)}$$

$$W_N^{N/2} = -1 \qquad \text{(d)}$$

The method described here is the *radix-2 algorithm*. The N-point data input, *x(n)*, is divided into even-indexes, and odd-indexes by replacing *n=2n* for even-indexes, and *n=2n+1* for odd-indexes. Thus, *X(k)* can be rewritten as follows,

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_N^{k(2n)} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_N^{k(2n+1)}$$

Applying property (c) , it becomes,

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_{\frac{N}{2}}^{kn} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_{\frac{N}{2}}^{kn} \qquad \text{(eq. 2.10)}$$

$$X(k) = \quad D_1(k) \quad + \quad W_N^k D_2(k)$$

According to the definition, summations D_1 and D_2 each represent a *N/2*-point FFT. Thus, a *N*-point FFT has been divided into two *N/2*-point FFTs which reduces the number of multiplications by half. For a *N/2*-point FFT, the range of *k* is $0 \le k \le \frac{N}{2} - 1$. For values $k > \frac{N}{2}$ the following analysis is applied,

$$X(k + N/2) = D_1(k + N/2) + W_N^{(k+N/2)} D_2(k + N/2)$$

$$X(k + N/2) = -W_N^{(k+N/2)} D_2(k)$$

Recall that a *N*-point DFT is periodic with period *N*. Therefore, a *N/2*-point DFT is periodic with period *N/2*. The process consists of continuing to

divide the *N*/2-point DFT into even-indexes, and odd-indexes by replacing *n* for *2n* (even case), and *n* for *2n+1* (odd case). This limits each sum to half the number of points. This procedure continues until a 2-point DFT is reached. This is achieved after $log_2 N - 1$ divisions (even and odd indexes). However, there is a multiplication element W_N^k at each odd division.

A graphical representation for a 2-point FFT can be represented as in figure 2.1. It represents the non-reducible core of the FFT algorithm.

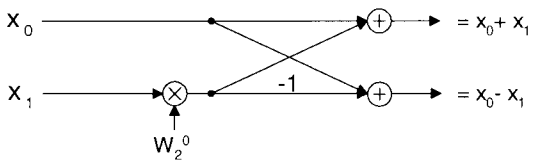


Figure 2.1
Butterfly Representation for a 2-point FFT

2.4.1 Example for an 8-point DFT Using a Radix-2 Algorithm

The number of divisions performed on a N-point FFT is given by $\log_2 N - 1$. Therefore, the number of divisions of an 8-point FFT is 2. Using the first division procedure described by eq. 2.10, we have,

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_N^{k(2n)} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_N^{k(2n)} \qquad \text{(eq. 2.11)}$$

Second division: for each sum, $n = 2n$ for even-indexes and $n = 2n+1$ for odd-indexes; $X(k)$ becomes,

$$X(k) = \sum_{n=0}^{\frac{N}{4}-1} x(4n) W_N^{k(4n)} + W_N^{2k} \sum_{n=0}^{\frac{N}{4}-1} x(4n+2) W_N^{k(4n)} + W_N^{k} \left[\sum_{n=0}^{\frac{N}{4}-1} x(4n+1) W_N^{k(4n)} + W_N^{2k} \sum_{n=0}^{\frac{N}{2}-1} x(4n+3) W_N^{k(4n)} \right]$$

replacing N=8,

$$X(k) = \sum_{n=0}^{1} x(4n) W_8^{k(4n)} + W_8^{2k} \sum_{n=0}^{1} x(4n+2) W_8^{k(4n)} + W_8^{k} \left[\sum_{n=0}^{1} x(4n+1) W_8^{k(4n)} + W_N^{2k} \sum_{n=0}^{1} x(4n+3) W_8^{k(4n)} \right]$$

(eq. 2.13)

Each sum represents a 2-point DFT as desired. Thus,

$$X(0) = \left[x(0) + x(4) W_8^{0} \right] + W_8^{0} \left[x(2) + x(6) W_8^{0} \right] + W_8^{0} \left\{ \left[x(1) + x(3) W_8^{0} \right] + W_8^{0} \left[x(3) + x(7) W_8^{0} \right] \right\}$$

$$X(1) = \left[x(0) + x(4) W_8^{4} \right] + W_8^{2} \left[x(2) + x(6) W_8^{4} \right] + W_8^{1} \left\{ \left[x(1) + x(3) W_8^{4} \right] + W_8^{3} \left[x(3) + x(7) W_8^{4} \right] \right\}$$

Notice that $W_8^{4} = -1$. Therefore, $X(1)$ is simplified. Thus,

$$X(0) = \left\{ \left[x(0) + x(4) \right] + W_8^{0} \left[x(2) + x(6) \right] \right\} + W_8^{0} \left\{ \left[x(1) + x(3) \right] + W_8^{0} \left[x(3) + x(7) \right] \right\}$$

pair pair pair pair

$$X(1) = \left\{ \left[x(0) - x(4) \right] + W_8^{2} \left[x(2) - x(6) \right] \right\} + W_8^{1} \left\{ \left[x(1) - x(3) \right] + W_8^{3} \left[x(3) - x(7) \right] \right\}$$

Each pair corresponds to a 2-point FFT which is the core of the FFT algorithm. The results are then combined into a 4-point and then an 8-point FFT. This is illustrated in figure 2.2.

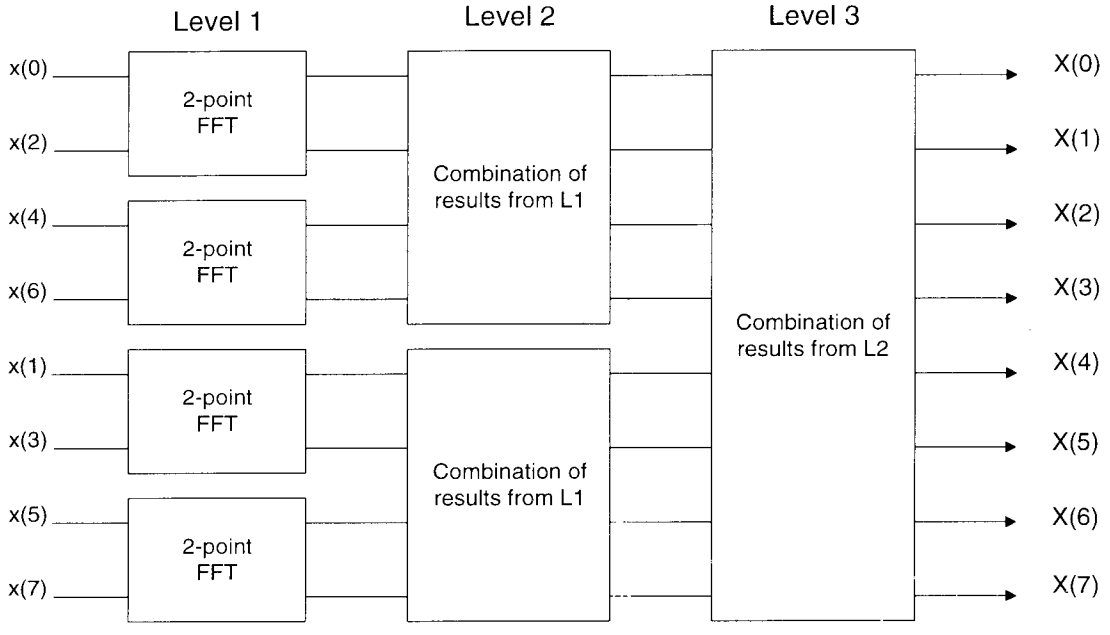


Figure 2.2
Levels for an 8-point Decimation-in-Time FFT

The results of this method are *X(0)* and *X(1)*. To obtain pairs *X(2)* and *X(3)*, the following procedure is applied:

1. Replace $k = k + 2$ in eq. 2.13
2. Evaluate $k = 0$ to obtain *X(2)*
3. Evaluate $k = 1$ to obtain *X(3)*

For $k = 0$,

$$X(2) = \{[x(0) + x(4)] - [x(2) + x(6)]\} + W_8^2 \{[x(1) + x(3)] - [x(3) + x(7)]\}$$

For $k = 1$,

$$X(3) = \{[x(0) - x(4)] - W_8^2[x(2) - x(6)]\} + W_8^3 \{[x(1) - x(3)] - W_8^2[x(3) - x(7)]\}$$

Notice that the second level of computation (pairs of { }) for *X(0)* and *X(2)* is equivalent to a 2-point FFT. The difference is encountered in the factor W_8^2. This behavior is repeated for all even-indexes when reducing a *N*-point FFT down to series of 2-point FFTs. To obtain the remaining pairs *X(4) ... X(7)*, *k* is replaced by *k + 4* in eq. 2.13, and it is evaluated from *k = 0* to *k = 3*. In general, *k* is incremented by a multiple of 2 as many times as divisions were performed to reduce the N-point FFT to a series of 2-point FFTs. *k* is then evaluated from 0 to the value of the increment - 1.

For an 8-point FFT the following method is applied:

Level 1: Calculate the 2-point FFT (expression between [ ])

Level 2: Combine results from level 1 (expression between { })

Level 3: Combine results from level 2

A flow graph of this formulation has a regular structure. A complete radix-2 formulation of the 8-point FFT is illustrated in figure 2.3.

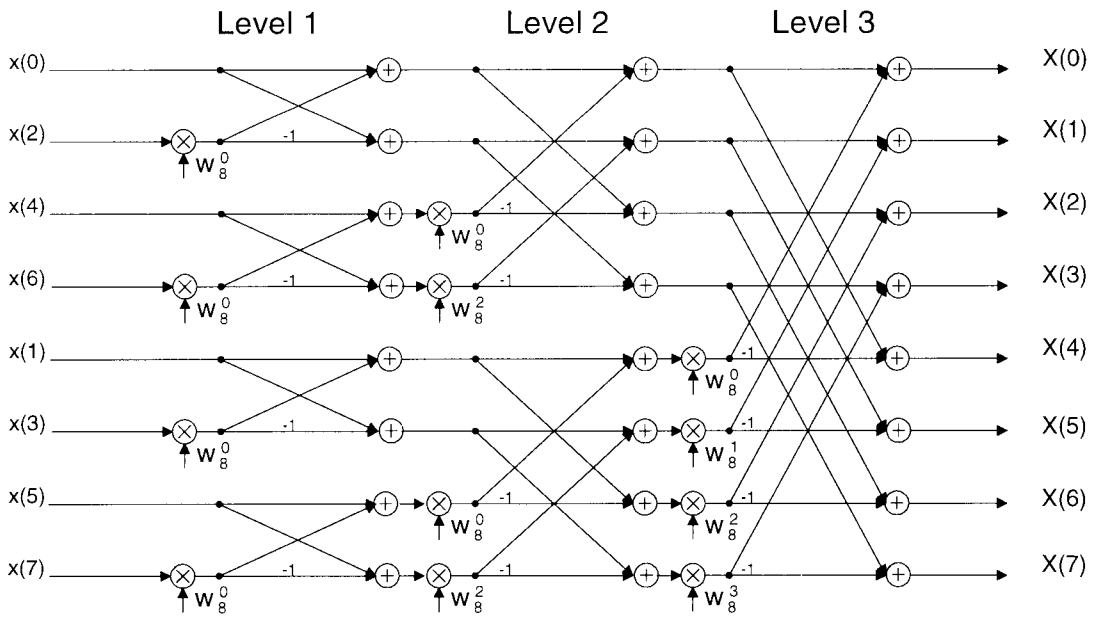


Figure 2.3
Flow Graph for an 8-point Decimation-in-Time FFT

Notice that data input is not consecutive. Instead there is a decimation-in-time. The data ordering is generated using the bit reversal algorithm, which is explained later in this chapter.

2.4.2 Formulation for a 16-point FFT

As shown above, the formulation for a radix-2 FFT algorithm has a regular structure. A 16-point decimation-in-time FFT is now shown. By having this example it is easier to visualize the regularity of this algorithm. Text books usually describe the formulation for an 8-point FFT. I extended this graphical description to a 16-point FFT to give the reader a better feeling for the structure of *N*-point FFTs.

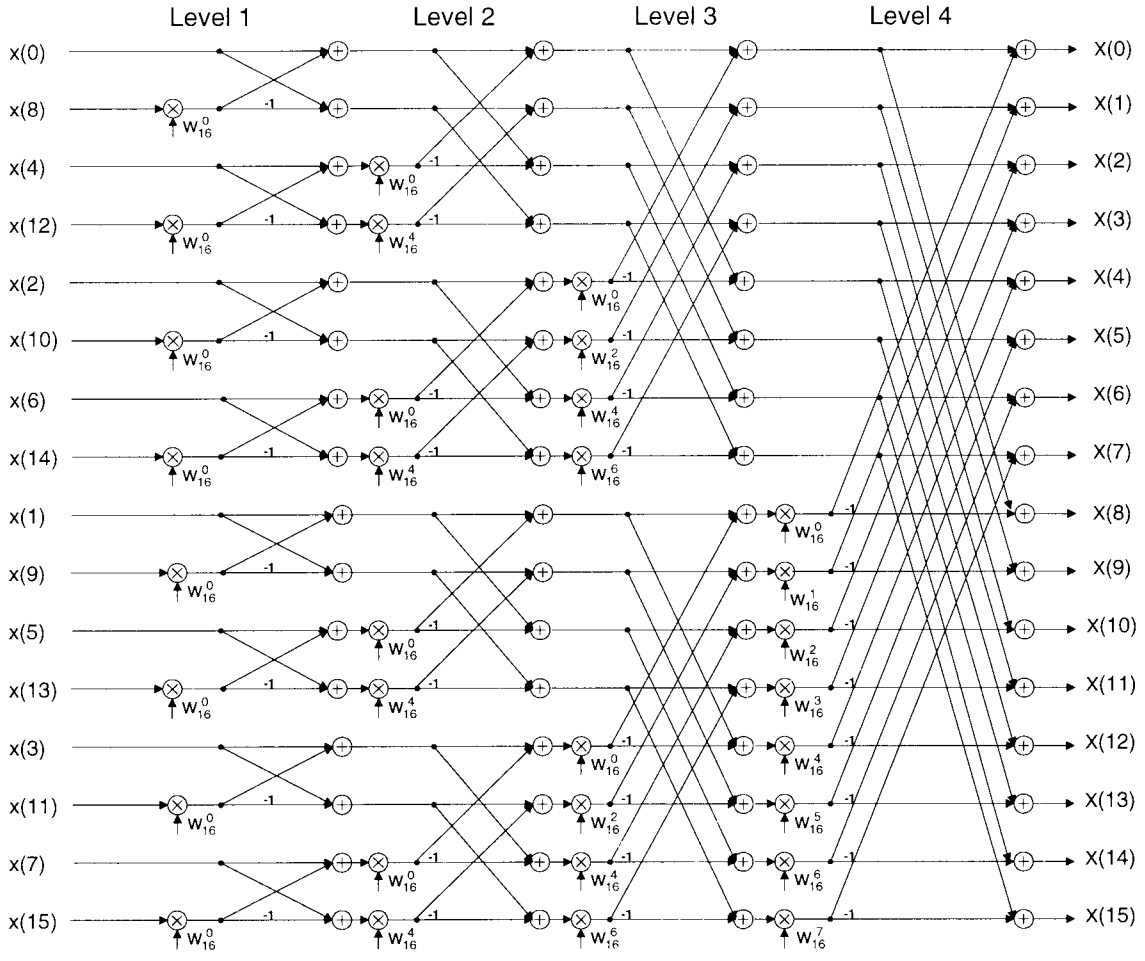


Figure 2.4
Flow Graph for a 16-point Decimation-in-Time FFT

In general, the N-point, radix-2 FFT has the following structural characteristics:

a. Number of levels: $log_2 N$

b. Number of butterflies per level = $\frac{N}{2^{current\ level}}$

c. Number of W_N per level = $2^{current\ level-1}$

2.4.3 Bit Reversal Algorithm

The input data must be reordered in order to compute the radix-2 FFT using a decimation-in-time algorithm. This is called bit reversal [13] [14] and must be done before, during, or after the FFT computation. As a matter of preference, the bit reversal is done prior to the FFT. For an 8-point FFT:

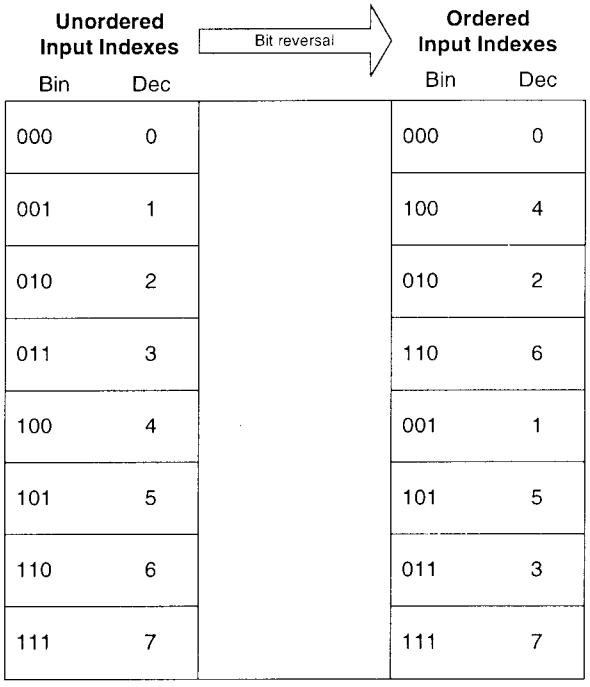


Figure 2.5
Bit Reversal for an 8-point Radix-2 Decimation-in-Time FFT

In general, a tree can describe this algorithm. Each level, L, in the bit reversal tree corresponds to series of 2^{L-1} values. The tree is shown below in figure 2.6, and the algorithm which generates this tree is also described. Starting from the root, each node has two children. The left child always has

the previous value, while the right has the previous value plus 2^{L-1}. In this thesis, the 8192-point FFT is the maximum size required.

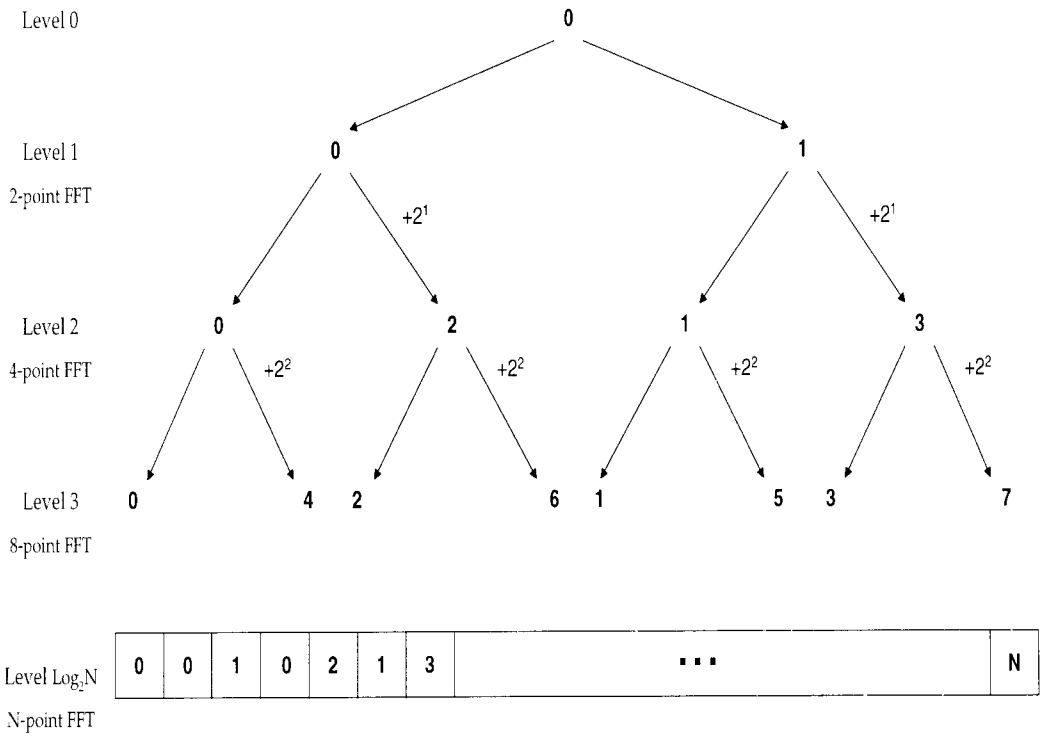


Figure 2.6
General Bit Reversal Tree

In order to generate the bit reversal indexes for a N-point FFT at any level, all previous levels must be generated in the process. Therefore, all necessary bit reversal indexes for FFT sizes up to 8192 points will be stored in a array of 2*8192-1 locations. An example of this array for an 8-point FFT is shown in figure 2.7. The array holds 15 values. The bit reversal indexes are found starting at location 8, and continuing until location 15. In general, the bit reversal indexes, for a N-point FFT, are contained in the array region [N,2N-1].

Once the array is generated for the maximum sized radix-2 FFT, any subsequent, smaller N-point FFTs will have their bit reversal indexes already calculated and conveniently located in the array starting at location N.

Location:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Input Indexes	0	0	1	0	2	1	3	0	4	2	6	1	5	3	7

Figure 2.7
Bit Reversal Array for an 8-point FFT

The overhead of the bit-reversal consists of filling the lookup array. The algorithm used to calculate this array is as follows:

Initialization:

```
offset = 1;          // amount to add to the right child
start  = 1;          // first location to fill
finish = 1;          // last location to fill
write  = 1;          // array is filled sequentially, this is the index
table[write] = 0;    // assume level 1 as base case

for level = 2 to 13          // up to 8192-point FFT (2^13)

    for i=start to finish
        table [++ write] = table [i];           // left branch = previous value
        table [++ write] = table [i] + offset;  // right branch = previous + 2^(level-1)
    end

    finish = write;
    start  = start << 1;
    offset = offset << 1;
end
```

Since the bit reversal algorithm was chosen to be done prior to the computation of the FFT, we can gain some speed when calculating the FFT for different data sets. This algorithm requires keeping a lookup table so the bit reversal is faster. I considered it important to describe the bit reversal implementation here since the algorithm was not found in the referenced literature. Implementation of the FFT core can be found in [15].

3. NOTE RECOGNITION

This chapter describes the method used to identify the notes present in musical sounds. Musical sounds used in this work correspond to monophonic signals from either a piano or a synthesizer.

Monophonic sounds occur when notes are played one-at-a-time. A music note is a complex tone that consists of a fundamental frequency and its partials, which are harmonically related. In monophonic musical sounds, a note decays before the next note is played; therefore, harmonics from one musical tone do not mix with harmonics of successive tones.

3.1 Data Collection

A synthesizer was utilized because it was more readily accessible than a piano. A synthesizer, is not, however, a realistic piano. It appears to the analysis as an ideal, well-tuned piano. Nevertheless, it was helpful to test the algorithms developed throughout this thesis and also in the development of the Graphic User Interface. However, the majority of musical sounds studied in this thesis correspond to monophonic signals recorded from an actual piano.

Musical sounds from the piano were recorded via a microphone to a tape recorder. Background noise was a factor when recording the data. With preliminary algorithms, I was only able to handle three octaves of the piano, middle C, and its adjacent octaves. However, with refined analysis, especially

with consideration of the missing fundamental phenomenon, explained later, I was able to handle the full range of piano notes, from 27.5 Hz to 4186.1 Hz. The sounds were played in *mezzo piano,* or moderately soft, intensity. At this volume, lower tones are more sensitive to noise than higher tones.

The set of data from the digital synthesizer was recorded by connecting the speaker line of the keyboard to the line-in of the sound card. Noise was a minimal factor in this configuration. The keyboard allowed a range of frequencies between 65Hz and 2058 Hz. As part of the user interface, software was developed to record the sounds originating at the line-in of the sound card. This allowed additional interaction with the user interface and seemed more satisfactory than restricting analysis to pre-recorded digital files.

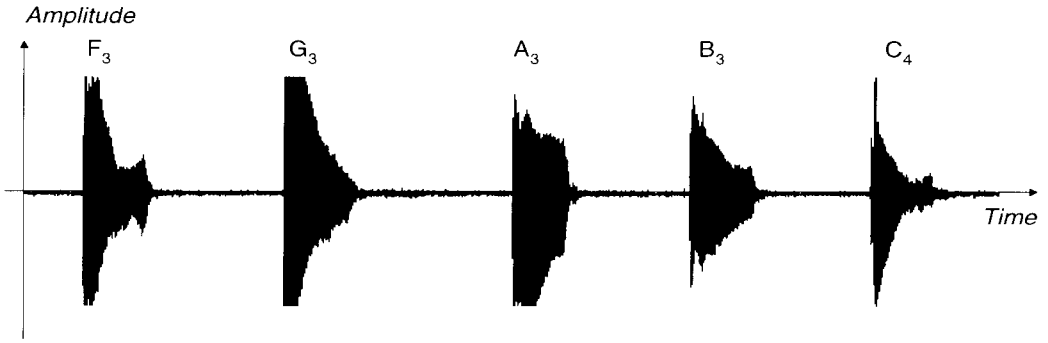


Figure 3.1
The Study Case Monophonic Sound

For the illustration of the algorithms developed in this chapter, a sound file shown in figure 3.1 was used. The sequence of tones was: F_3-G_3-A_3-B_3-C_4 and

is referred to as the *Study Case*. When testing the algorithm, many such sounds were used.

3.2 Sampling Rate

A starting point for note identification is to take series of piano notes and sample them. A sampling rate of 11.025 kHz was chosen since it preserves the frequency range of the piano while not placing overwhelming demands for data storage. Single channel data was recorded in 16 bit resolution at a rate of 22 Kbytes/second. According to Shannon's sampling theory [9], this sampling rate is adequate to prevent aliasing for the experimental data which covers the range 27 Hz - 4186 Hz.

3.3 Note Recognition Overview

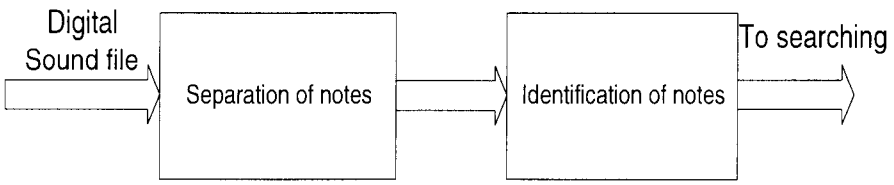


Figure 3.2
Block Diagram for Note Recognition Process

Figure 3.2 shows a block diagram that describes the identification process. Once the musical tones are recorded in a digital sound file, the notes

are separated for identification. Each note is then analyzed in the frequency-domain to extract the value of its fundamental frequency.

In the case of figure 3.1, the notes were played one-at-a-time, with no overlap between the consecutive notes. This is a simplification of music in general and makes monophonic music easier to analyze than polyphonic music. The simplification comes from the assumption that an automated recognizer could pick out sections of a recording that include notes and those sections that do not. An algorithm is then applied to each non-silent section in order to identify the note.

3.4 Separation of Notes

The process used to separate the notes is called segmentation. The purpose of segmentation is to find the boundaries of each note in the sound file. This is used to separate signal states [16]. Some other methods have been applied in speech recognition where more distinctive features are encountered. In a monophonic sound, there are basically two features or states: *note* or *silence*. Thus, segmentation is used to mark divisions between the two states. Once a section is located which contains data, it is assumed that it only contains a single note which can be enumerated on the piano scale. For monophonic sounds, the harmonics encountered in each segment will originate from a single piano note. This is because harmonics from different notes do not overlap.

As seen in figure 3.2, the first block for the identification process is the *Separation of Notes*. Once the notes are separated, identification can proceed independently for each note. For convenience, the notes in figure 3.3 have already been labeled. At this point in the analysis, however, the note identities are not known.

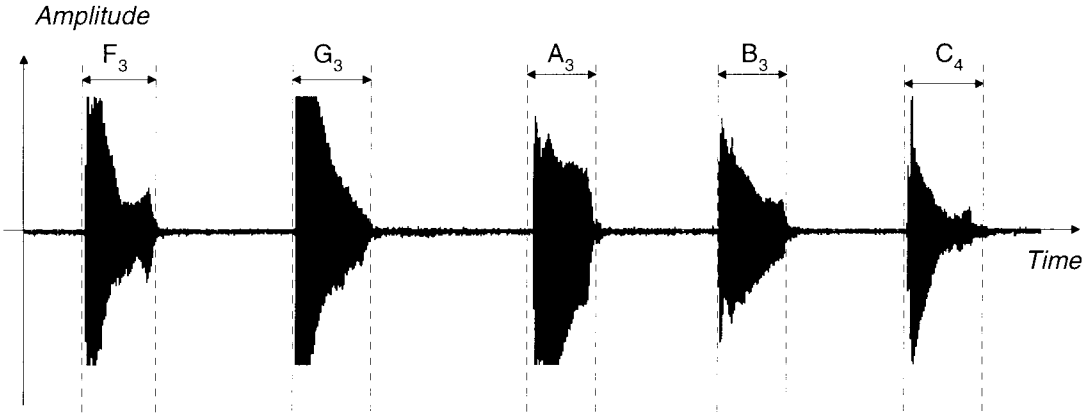


Figure 3.3
Visual Segmentation for the Study Case

By looking at figure 3.3, it is easy to notice that five notes were played since they are visually compressed in time. This compression is quite apparent. In fact, it is not possible to discern that the waveform is oscillating. A mechanism to determine when a note starts and ends can be realized by comparing the data with a threshold level. Since a monophonic sound occurs when a single note is played at a time; it decays considerably before the next one is played.

If a note decays with time, so does its energy. The energy of a signal *x(n)* over an interval [a,b] is given by,

$$E_{ab} = \sum_{n=a}^{n=b} |x(n)|^2$$

The average of the energy (power) on that interval is, $\overline{E}_{ab} = \frac{E_{ab}}{b-a+1}$. PCM samples are then divided into frames. Each frame has 100 samples in order to have adequate data to compute a short-term power signal.

This algorithm requires that the monophonic sound file has a period of silence, consisting of background noise, at the beginning of the sampled data. The length of silence is at least 25 ms [17]. This is a reasonable lag time before starting a music recording. The noise threshold is then calculated as the short-term power of the first 25 ms of data plus an offset to raise above most of the noise; in this case I used 10 dB.

The sound file is then divided into frames of 100 samples each. The number of frames is: $Floor\left(\frac{N}{100}\right)+1$, where N is the total number of samples within the sound file. A frame is marked as *note* or *silence* depending of its short-term power. For the last frame, the short-term power is computed using the number of samples remaining. Summarizing, the procedure for segmentation is as follows:

a. Calculate the short-term power for the first 25 ms, P_S

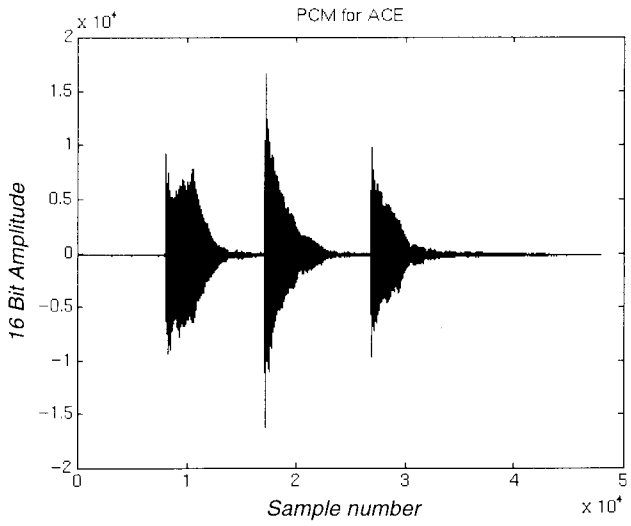
b. Divide the data into segments of 100 samples each and calculate the power for each segment.

c. Scan the data, segment-by-segment. If power of a segment is greater than or equal to P_S + 10 dB, mark the data within the segment as *note*. If the power of the segment is less than the threshold level, mark data within the segment as *silence*.

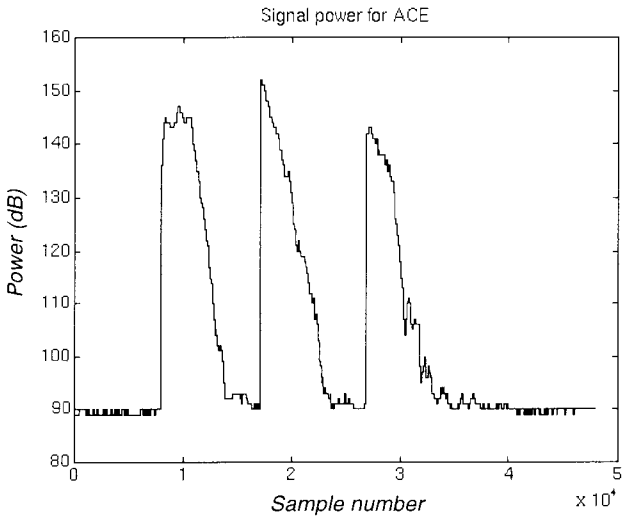
In practice, it is necessary to add 10 dB in the calculation of the threshold level due to the presence of noise in the recording. This level can be adjusted according to the conditions of the recording environment or instrument (piano/synthesizer).

This algorithm provided clean segmentation results for only a few sets of data. A segmentation which was successful is shown in figure 3.4 (a, b, c, d). Figure 3.4a shows PCM data which has three notes: A_4-C_4-E_4. Figure 3.4b shows its signal power, figure 3.4c shows the threshold level, and figure 3.4d shows the segmentation results.

For the study case, sequence F_3-G_3-A_3-B_3-C_4 shown in figure 3.1, results of the segmentation process are illustrated in figure 3.5. Data that is marked as *note* is represented by '1' , and *silence* is represented by '0'. Segmentation results represent the status (0 or 1) of each sample in the sound file.

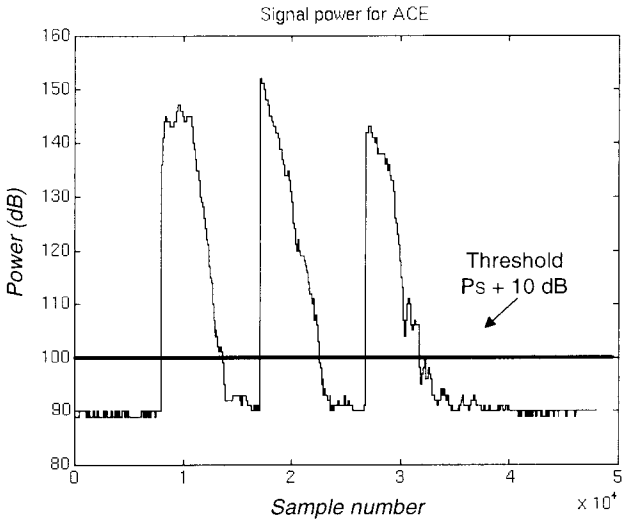


a. PCM data. Notes A_3-C_4-E_4

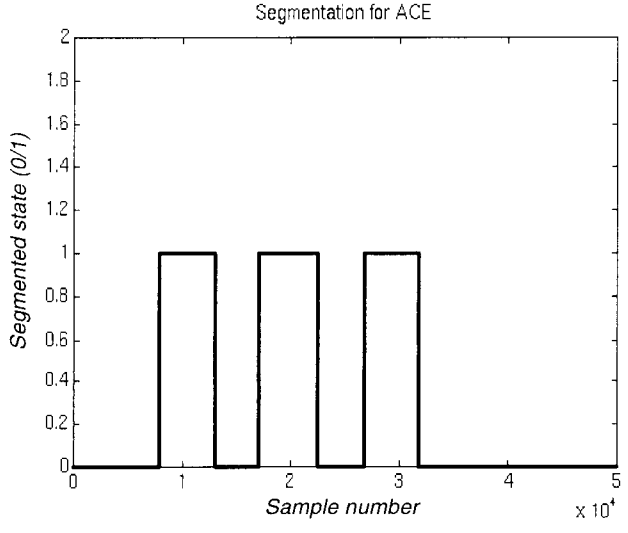


b. Short Term Power Signal

Figure 3.4
Segmentation Results for a Simple Example



c. Power Signal and Noise Level



d. Segmentation Results

Figure 3.4 (continued)

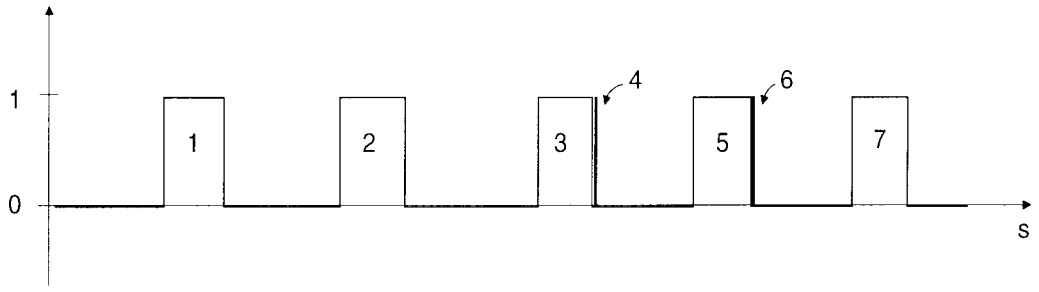


Figure 3.5
Segmentation Results for the Study Case

For the study case, the segmentation process produced erroneous results. Two sections of data were marked as new notes (regions labeled #4 and #6 in figure 3.5) even though they actually belonged to previous notes. This happened because there are some data samples inside a valid note that had low amplitudes. Therefore, some samples were marked as *silence* even though they were part of a note. See figure 3.6 to observe an example. This data was extracted from potential note #3 by magnification. The opposite situation, marking a section as *note* instead of *silence*, is also possible. It happens when some noise or spurious data is situated between notes. In order to correct this problem, a second revision of the results is performed based on the duration of a *note* or *silence*.

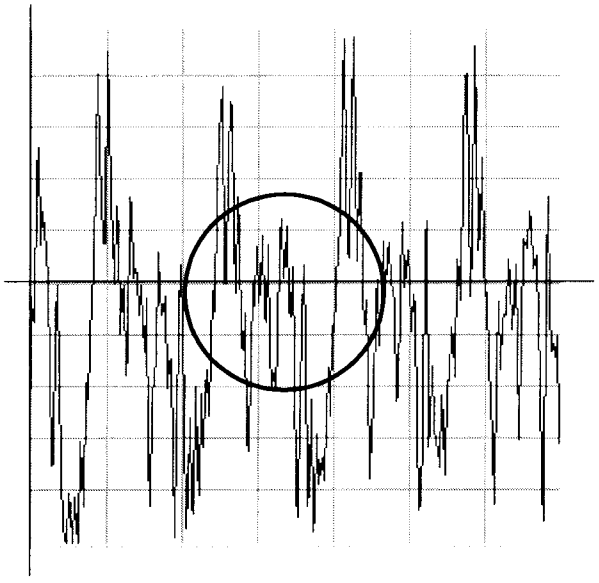


Figure 3.6
Low Level Samples Comprising a Note

The second pass, or revision, is performed to determine if the sections of data, marked as either *silence* or *note*, are valid. A section of data is considered to be valid if there are more than 100 ms of consecutive samples with the same state. If the size of a potential note is less than 100 ms, then it will maintain the state of the previous valid data. The reason for choosing 100 ms relies on the fact that a fast melody passage, in western music, is played at a rate not exceeding 10 notes per second. Thus, one note is played in no less time than 100 ms. Recall that the sampling frequency is 11,025 samples/sec. Therefore, 100 ms is equivalent to approximately 1100 samples. Thus, data is marked valid if the length of consecutive samples is greater than 1100 samples. Landmark points are localized on the transitions between levels. At each transition between *note* or *silence*, the previous valid state changes. Remember

that the initial state is assumed to be *silence*. A transition represents the testing point to check whether or not the state of a section is, in fact, valid.

Regarding the flowchart in figure 3.8, *Valid State* represents the valid state of a section, and it is initialized to *silence*. The state is validated if the section is long enough to maintain its current state; otherwise, the section will be assigned the last valid state encountered. The result of the second pass for the study case is shown in figure 3.7. The negative portion of the graphic illustrates regions where the state was changed to the previous valid state. They could have been either part of the previous *note*, or part of the previous *silence*. In this case, they were part of previous *note* states.

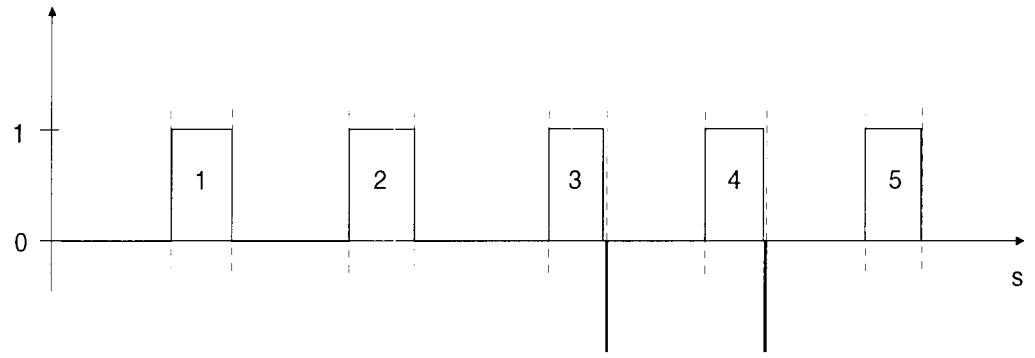


Figure 3.7
Second Pass for Note Segmentation

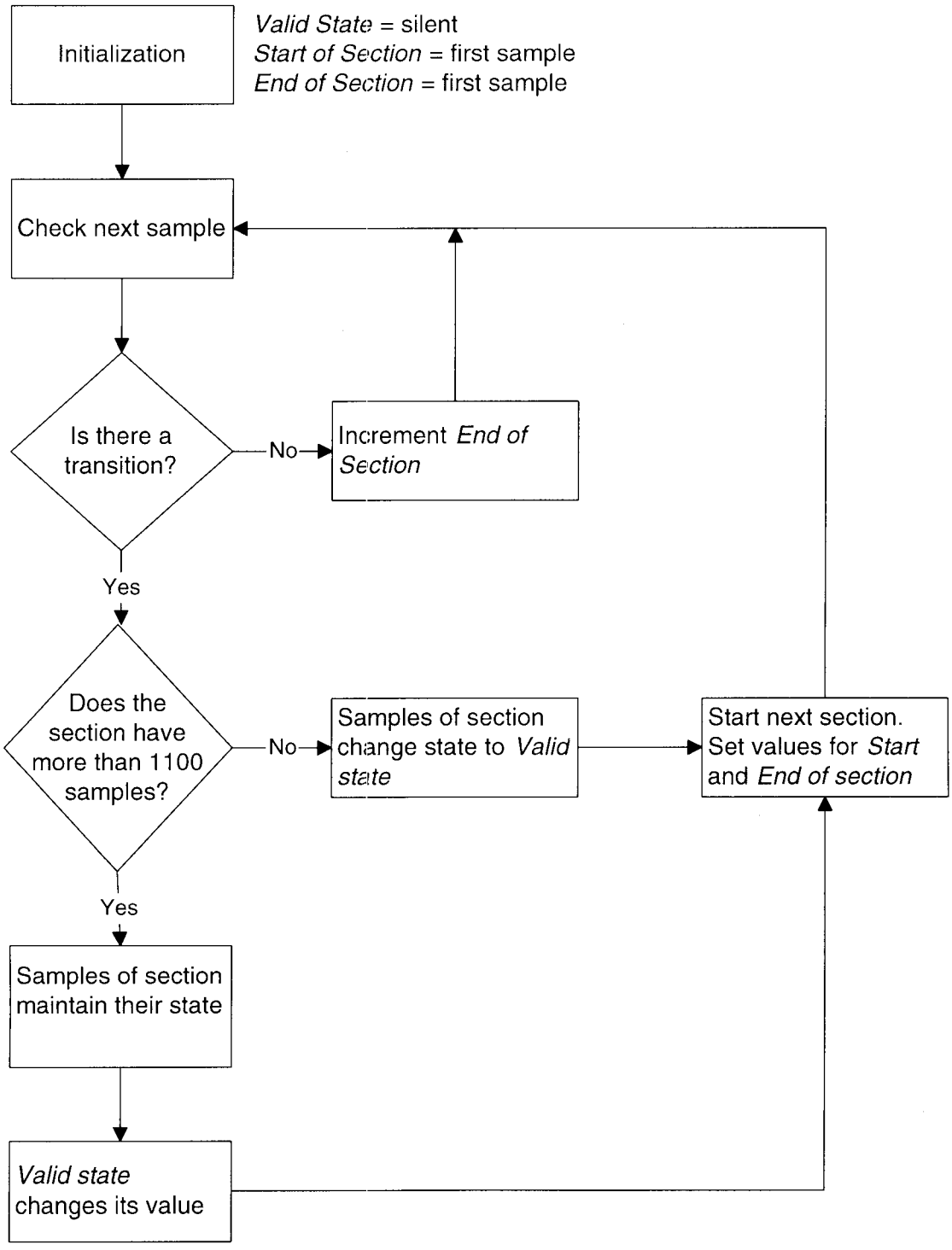


Figure 3.8
Flowchart for Second Pass of Segmentation

Once the sound file is separated into states of *note* and *silence*, the next step is to identify each of the notes. A frequency-domain analysis is then performed for each note to determine the value of its fundamental frequency. Finding the fundamental frequency is the means of discovering the identity of each potential note. Since the identification of a note requires additional information about the source instrument, a short section about the piano is included next.

3.5 Important Piano Observations

Examining the organization of piano frequencies provides insight for how to analyze, with sufficient resolution, piano music in the frequency-domain.

3.5.1 Piano Tuning

A piano keyboard has 88 keys, from A_0 at 27.5 Hz to C_8 at 4186.1 Hz. The piano keyboard is divided into octaves. An octave is composed of many notes such that the last note in the octave has twice the frequency of the first note in the octave.

There are different ways to tune a musical instrument, e.g. to define the frequency of each note. In western music a method called *Equal Temperament* is commonly used to tune keyboard instruments [2]. Tuning instruments by equal temperament divides an octave into N equally spaced intervals. For

keyboard instruments the number of intervals is twelve. Thus, an octave is divided into twelve equal semitone intervals. It means that two adjacent tones, or semitones, have the same ratio.

$$\frac{freq(note2)}{freq(note1)} = \frac{freq(note3)}{freq(note2)} = \frac{freq(note4)}{freq(note3)}$$

The ratio for an octave of twelve tones is $2^{1/12}$. The base, 2, comes from the definition of an octave where the last tone is at twice the frequency of the first note. The exponent is the inverse of number of intervals. If *note 1* has a fundamental frequency of 20 Hz, then *note 13* will have a fundamental frequency of 40 Hz, which is twice that of *note 1*. The integrity of the definition of an octave is maintained. Table 3.1 shows frequency ratios within an octave.

Note 1	1	*Note 2*	$2^{1/12}$
Note 3	$2^{2/12}$	*Note 4*	$2^{3/12}$
Note 5	$2^{4/12}$	*Note 6*	$2^{5/12}$
Note 7	$2^{6/12}$	*Note 8*	$2^{7/12}$
Note 9	$2^{8/12}$	*Note 10*	$2^{9/12}$
Note 11	$2^{10/12}$	*Note 12*	$2^{11/12}$

Table 3.1
Frequency Ratios within an Octave

A complete table for the actual frequencies of a piano using the twelve semitone equal-tempered system is shown in the Appendix.

3.5.2 Characterization of the Piano Scale

A piano scale can be characterized using a simple model. For each key on the piano keyboard there exists a fundamental frequency, F_0. Since each fundamental is unique on the piano, a simple method for matching a note with the key that would produce that sound can be devised. By having a table of the piano fundamentals a priori, a lookup can be made to determine the proper note. The lookup table can be generated by mapping each note to a bounded integer range centered on its fundamental frequency. For example, the fundamental frequency of B_2, which is 123.47 Hz, is mapped to integers 122, 123, and 124. This allows a close approximation of the fundamental to be mapped to the key which owns slots 122, 123, 124 in the lookup array. An example is realized for keys shown in figure 3.9. This table works by simply indexing of the rounded value of F_0. A maximum of 4190 entries are required for the lookup table. Recall, the highest frequency on a piano is 4186.1 Hz.

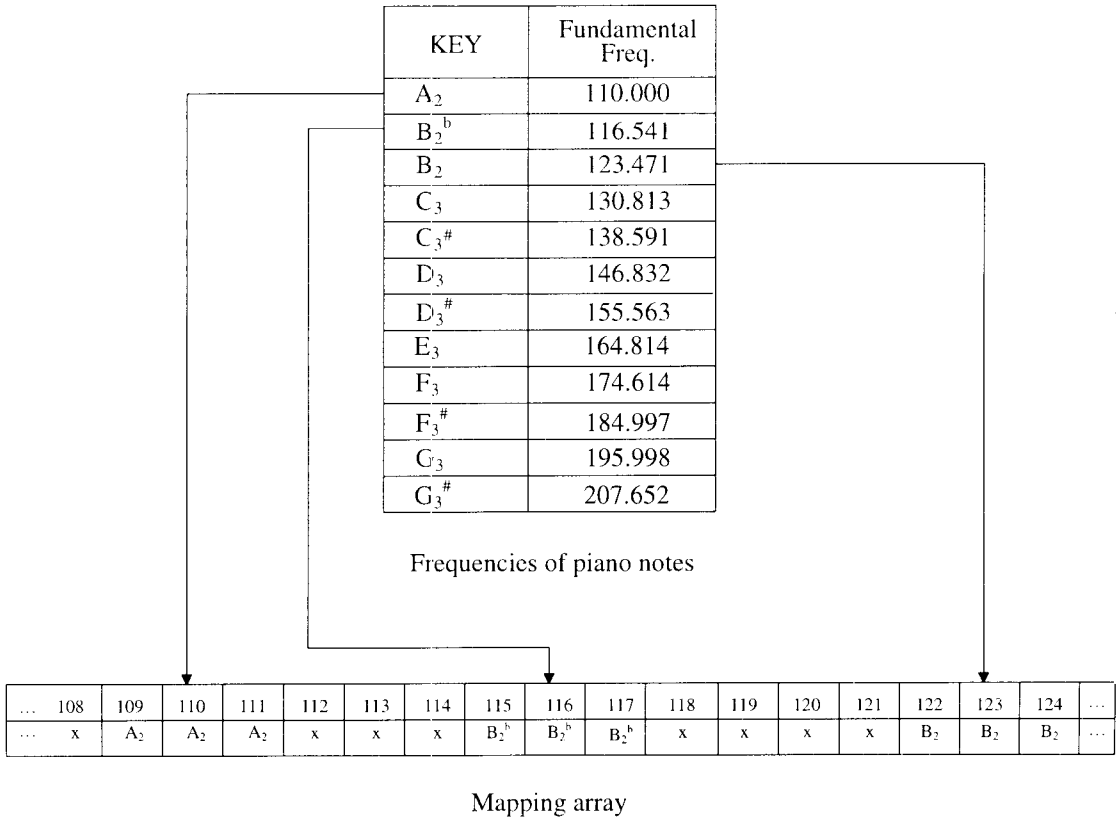


Figure 3.9
Mapping of Fundamental Frequency to Piano Key

A more precise table could be made by indexing with some multiple of the frequency. This would create more entries in the lookup table. For the table used in practice, a multiple of 10 was chosen. A key maps to a range of frequencies near the fundamental. This range increases at higher frequency since the spacing in frequency between consecutive keys increases. It happens because the frequency scale on the piano is geometric with a ratio of $2^{1/12}$ (1.059) between consecutive notes, or 6%. F_0 is first multiplied by 10 before indexing into the mapping array. It allows more flexibility and accuracy when indexing

the fundamental frequency given by the spectrum analysis for low frequency piano notes.

The range for each entry was chosen to be within 3% of its fundamental, using to 1.5% to cover lower frequencies and 1.5% to cover higher frequencies. In other words, the range for frequency at f_i is $[f_i - 1.5\%f_i, f_i + 1.5\%f_i]$. For example, key #48 whose theoretical fundamental frequency is 415.30 Hz is stored in the mapping table centered at position 41530. The valid range of the table mapping to key #48 is given by locations [40907, 42152] which actually correspond to the frequency range from 409.1 Hz to 421.5 Hz. By choosing a range as a percentage of the theoretical fundamental, the size of the ranges increase with the fundamental.

There is an upper limit on the number of possibilities for a correct identification, a "vocabulary" of notes. We will now investigate the identification algorithm.

3.6 Note Identification

This method to find the fundamental frequency of a note uses the Fast Fourier Transform (FFT) which converts PCM time-domain data into the frequency-domain. This is applied to each localized note. In order to illustrate this, the first note from the study case was extracted. Its frequency spectrum was computed by applying the algorithm described in chapter two. The results are shown in figures 3.10 and 3.11. The magnitude of the frequency spectrum

was normalized to 1. A peak is considered to be a harmonic if its magnitude is greater than 0.1.

Figure 3.10
PCM Data of the First Note in the Study Case

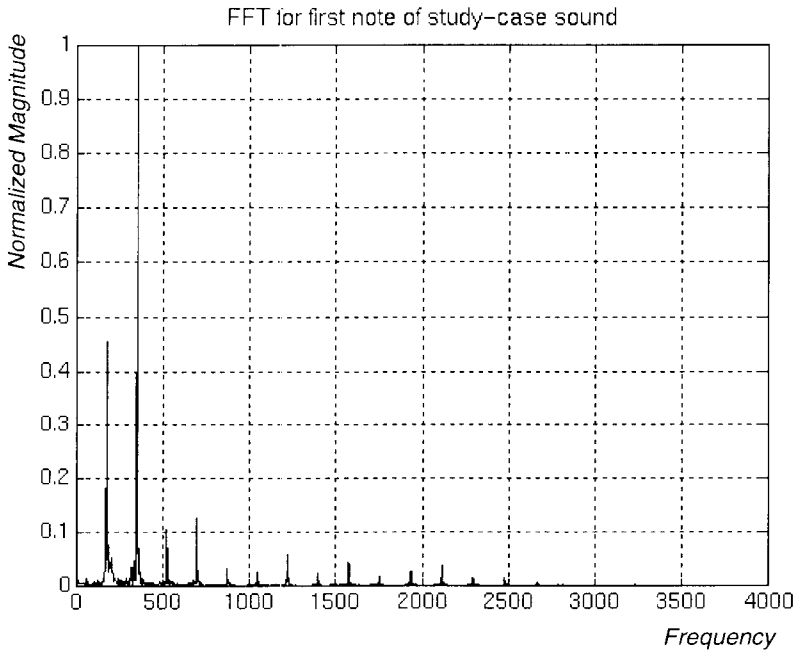


Figure 3.11
Fourier Transform of the First Note in the Study Case

The first peak corresponds to the fundamental frequency while the other peaks should correspond to the harmonics. In this case, the first peak is localized at 172.85 Hz, which correctly maps the fundamental to piano key F_3.

Unfortunately, this simple analysis is not robust, and could easily fail when confronted with a phenomenon called the *Missing Fundamental*. This phenomenon states that, while the fundamental frequency is sometimes missing in a complex tone, we can still hear it [18]. It happens because the remaining harmonics form a pattern of repetition at the greatest common divisor (GCD), which is the fundamental frequency [2]. Our ear is capable of sensing this missing frequency. The fundamental frequency, therefore, also may be calculated as the GCD of the remaining harmonics.

Ideally, the frequency spectrum of a complex tone is comprised by its fundamental and its harmonics. In practice, a noisy environment and internal characteristics of the physical instrument deviate from this ideal model by allowing other peaks to be present in the frequency spectrum. The ideal frequency spectrum will have well defined peaks at each harmonic, however, as a practical matter, spurious peaks will arise and must be filtered.

The algorithm developed in this section is based on the fact that harmonics of a complex tone occur consecutively at multiples of F_0. Therefore, the distance between harmonics of a musical tone corresponds to the value of F_0. Finding the harmonic spacing, the most common distance between adjacent peaks in the frequency spectrum, is the method to determine F_0.

3.6.1 Filtering Spurious Peaks

Since the actual frequency spectrum will have some spurious peaks that do not contribute to the process of resolving the fundamental frequency, a procedure is applied to filter the data. This procedure is as follows:

a. Ignore all frequency spectrum data if its magnitude is less than 10% of the maximum FFT magnitude.

b. Mark the segment where the magnitude first crosses the 10% level and then dips below. A peak is chosen as the maximum value within this segment.

c. If the distance of two consecutive peaks is less than 24 Hz, discard smaller peak. 24 Hz was selected because the minimum F_0 on a piano scale is 27.5 Hz. No valid piano note could generate a frequency below 24 Hz.

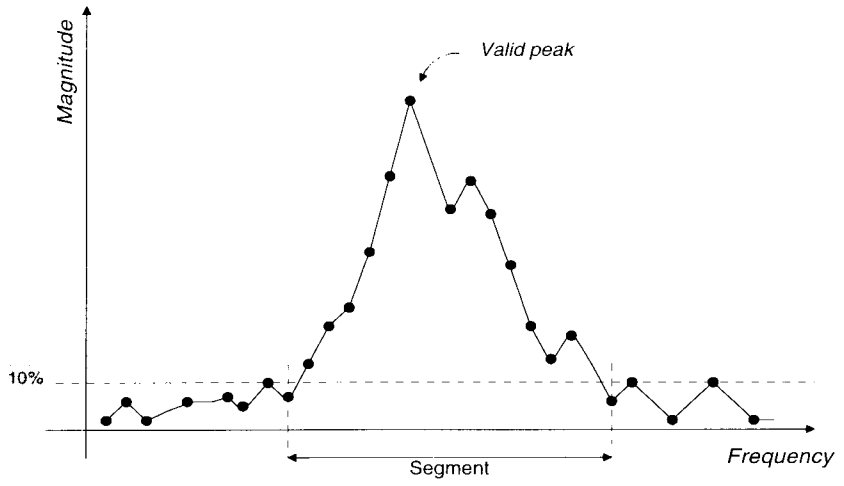


Figure 3.12
Localization of a Valid Peak.

Figure 3.12 shows an example of a valid peak. The graph is a partial frequency spectrum of piano key #42.

3.6.2 Resolving the Fundamental Frequency

In most cases, valid peaks are located at harmonic frequencies. This, however, cannot be guaranteed. Resolving F_0 from these peaks relies on finding the most common adjacent peak difference. Since we are working with floating-point data, the distances will not be identical, but they will be close. Therefore, an algorithm is designed to converge the distances to agree on a single frequency. Even though all the harmonics are not localized, the most frequent distance between peaks should tend to the value of F_0. Taking the median of a list of peak differences should give this value. This algorithm is as follows:

a. Compute and tabulate the distance in frequency between two adjacent peaks, *FD*, and calculate the average, *AFD*.

b. Map each *FD* within 4% of *AFD* to the same integer. In other words, round to the nearest 4% of *AFD*, the resulting number is called the *RFD*. Calculate the median.

c. Generate a new table using the original data. Only include entries whose *RFD* equals the median.

d. Repeat steps b. and c. For step b., the values are rounded to the nearest integer frequency.

e. Calculate the median of the new values. This *RFD* represents the resolved F_0.

This procedure is explained with an example. This example is based on figure 3.13. The data used in figure 3.13 correspond to the frequency spectrum of key #42 whose theoretical F_0 is 293.66 Hz. Seven peaks were detected.

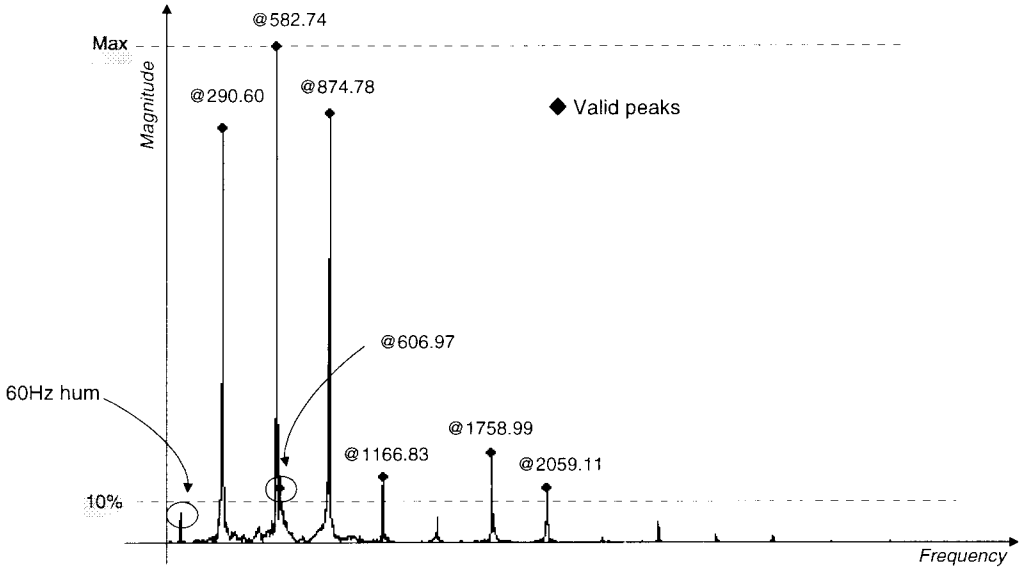


Figure 3.13
Frequency Spectrum for Piano Key #42

a. Compute and tabulate, from left to right, the distance in frequency between two adjacent peaks, *FD*, and calculate the average, *AFD*.

INDEX	$Peak_i$ (Hz)	$Peak_{i+1}$ (Hz)	FD_i (Hz)
1	290.74	582.74	292.04
2	582.74	606.97	24.22
3	606.97	874.78	267.81
4	874.78	1116.83	292.04
5	1116.83	1758.99	592.16
6	1758.99	2059.11	300.11

Table 3.2
Adjacent Frequency Distance

From table 3.2 the *AFD* is 294.73 Hz. This average value cannot be used as the resolved fundamental frequency since any single misplaced peak could substantially affect the calculation. This example also turned out to be very special since the *FD* between the second and third peak was greater than 24 Hz. Thus, the third peak was kept even though it was obviously wrong.

b. Generate rounded values for *FD* and calculate the median.

The goal is to reduce the *FD* table to the more common values. In order to allow similar values of *FD* to map to the same integer, each distance is rounded to a multiple of 4% of the *AFD*, called the rounded factor (*rf*). Table 3.3 contains these values for all the segments. In this case, *rf* is 11 Hz.

$$\text{rounded factor } (rf) = 4\% \text{ of the } AFD \qquad \text{(eq. 3.1)}$$

$$RFD_i = [Floor\ (FD_i/rf) + 0.5)]^{*}rf \qquad \text{(eq. 3.2)}$$

INDEX	FD_i (Hz)	RFD_i (Hz)
1	292.94	297
2	24.22	22
3	267.81	264
4	292.04	297
5	592.16	594
6	300.11	297

Table 3.3
Rounded Frequency Distance (1)

The median RFD is 297 Hz. Only entries with a RFD equal to 297 Hz will be considered as F_0 candidates in further analysis.

c. Generate a new table using the original data. Only include entries whose *RFD* equals the median.

Table 3.2 was reconstructed by choosing only the indexes of *FDs* that have a *RFD* equal to the median. In this case, the indexes are 1, 4, 6. This is shown in table 3.4 .

OLD INDEX	NEW INDEX	FD_i (Hz)
1	1	292.04
4	2	292.04
6	3	300.11

Table 3.4
Median Frequency Distance

d. Repeat steps b. and c. For step b., the values are rounded to the nearest integer frequency.

In this case, FD rounding is made to the nearest Hz. The result is labeled as RFD and it is displayed for each entry in table 3.5.

INDEX	RFD_i (Hz)
1	292
2	292
3	300

Table 3.5
Rounded Frequency Distance (2)

The new median is 292 Hz which represents the resolved frequency. This value is multiplied by 10 and becomes the index into the fundamental

frequency ↔ piano key mapping table discussed in section 3.5.2. In this case, the note is mapped to key #42 as desired.

This method overcomes the difficulties of missing harmonics. It also handles the task of matching floating-point numbers which are not equivalent, but rather close.

3.7 Resolution in Frequency

The resolution of the frequency analysis depends on the number of points of the FFT and the sampling frequency. For analysis, an 8192-point FFT was chosen. It means that in the frequency spectrum, the samples will be located at $f_k = \frac{sampling\ frequency * k}{N}$, where $0 \le k \le \frac{N}{2} - 1$. Therefore, the frequency resolution of the FFT is:

$$resolution = \frac{f}{N} = \frac{11025}{8192} = 1.346\ Hz$$

The smallest frequency difference on the piano scale is 1.6 Hz (between the lowest notes). Hence, this resolution is sufficient to distinguish the closest frequencies in the lowest scale. Moving from the lowest frequency to the higher frequencies in the piano keyboard, the difference between consecutive frequencies increases geometrically with a ratio of 1.056. For high frequencies, this resolution is more than necessary; fewer points for the FFT could have been used. In this thesis, however, a global analysis was applied using an 8192-point

FFT for the entire piano scale. The results of the experiments showed that this global method worked. All the frequencies in the piano scale were distinguished. Nevertheless, future work could consider the use of more sophisticated tools such as Constant Q Spectral Transform [25], and parametric analysis [8].

4. SEARCHING ALGORITHM

From the previous chapter, the musical notes comprised in a sound file were detected and identified. These files were recorded from a classical instrument, specifically the piano. The purpose of this chapter is to recognize a musical file given a sequence of musical notes that may be part of the file. In other words, the main idea is to implement an algorithm that allows a user to recognize a musical file which contains particular notes.

4.1 General Description

A block diagram, figure 4.1, shows the inputs and outputs of the searching process. The following terms are used throughout this chapter.

> *Primary*: a musical file comprised by P, a sequence of notes, where S may be found.
>
> *Secondary*: a musical file comprised by S, a sequence of notes, that might be contained in sequence P.
>
> *Approximation level*: the desired accuracy of the match. For a level of 100%, it is an exact match.

Recall that both primary and secondary musical files are transformed into sequences of musical notes using the *Note Recognition* algorithm. The results of the transformations on both files are passed to the *Searching* block, along with the *Approximation level* to find the possible matches of *Secondary* within the *Primary* file.

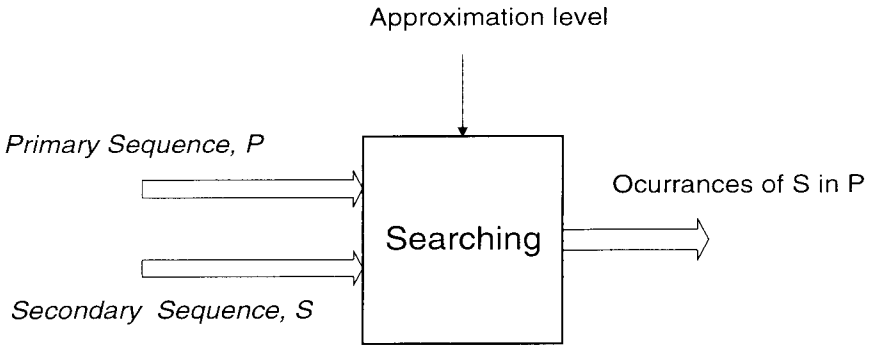


Figure 4.1
Block Diagram for Searching Algorithm

Since both sequences which enter the search block come from a transformation, *Music file* → *Sequence of Notes*, they might not be 100% accurate. Thus, searching techniques should not only find an exact match, but rather, they should allow approximate matches. This approach has two advantages: (1) to overcome possible transformation mistakes, and (2) to allow flexibility in the matching process. The latter comes about since a person may only remember a portion of the song (incorrectly); therefore, a user will be given several options. An exact match is a special case when the approximation level is chosen to be 100%.

Once the sound files are transformed into sequences of notes, we can describe these two sequences as strings of data. The content of each string is a sequence of characters from a finite character set. This character set, called Σ, is comprised with the 88 keys of the piano. We can apply searching methods used for string matching in text editors, pattern recognition in molecular biology, bibliography retrieval, and symbol manipulation. For these

applications the character set might be the English alphabet or DNA base pairs, for example.

This chapter studies two searching techniques. The first one corresponds to an algorithm for Exact String Searching called *Boyer-Moore algorithm* [19]. The implementation has the modification suggested by Horspool [20]. The second method is designed based on musical note characteristics.

In string matching, basically we are searching for a pattern inside a large string. In this work the pattern is called *secondary (S)*, and the large string is called *primary (P)*. Thus, the objective of string matching is to find all the occurrences of S in P. The size of S is m, and the size of P is n.

In order to explain the searching algorithms, a definition of the basic variables is necessary. Let S be the secondary sequence, $s_1s_2s_3...s_m$, inside the principal sequence, P, $p_1p_2p_3...p_n$. Both are sequences of characters inside a character set Σ that contains the musical notes. The objective of string searching is to find a set of locations L such that $p_ip_{i+1}...p_{i+m-1} = S$, where i is the pointer to the beginning of the match in P.

4.2 Boyer-Moore Algorithm

The basic characteristic of the Boyer-Moore algorithm (BM) is that the match is tested from the last character of S [19]. According to the authors, it allows large jumps throughout the primary text.

Initially, both S and P are aligned to the left, e.g. the first character of S, s_1 is aligned to the first character of P, p_1. The character of P at location m (length of S) is the target character called tc. At this point $i = m$. The Boyer-Moore algorithm contains several observations, each one is explained below. Musical notes in this section are contained entirely within an octave. The subscripts of the notes were omitted to provide better clarity.

4.2.1 Observation No. 1

If tc is not found in S, the next target character on P will be located at m locations after the current one, since there could not be an occurrence of S in P from locations 1 to m. Thus, the pointer i is incremented by m. An example of this case is shown in figure 4.2.

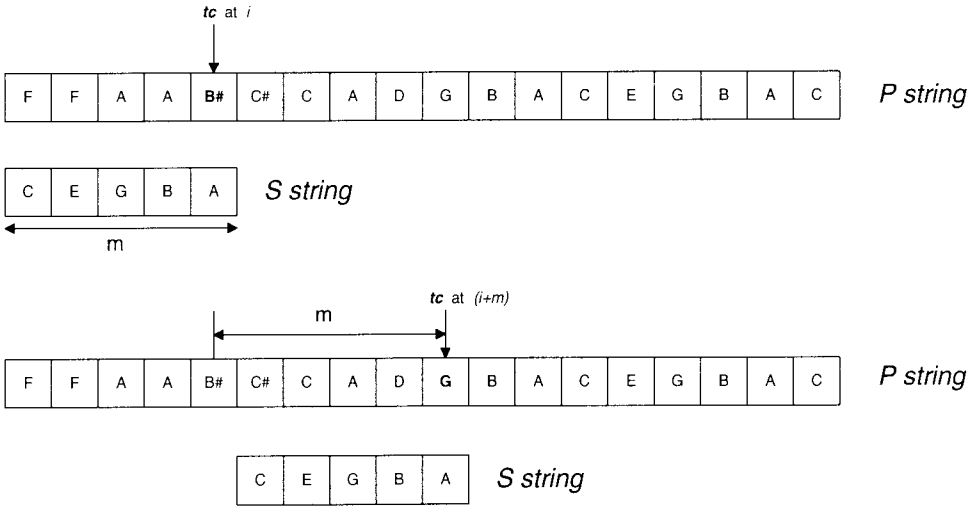


Figure 4.2
Example for Observation No. 1 of BM algorithm

4.2.2 Observation No. 2

This is a general rule for observation No. 1 when *tc* is found within the interval of *S*, at a distance of $delta_1$ from location of last character in *S*. Thus, *delta1 = m - location(tc)* in S. In this case, *S* is shifted to the right by $delta_1$, in other words, the pointer to *P*, *i*, is incremented by $delta_1$. Notice that if *tc* is not found, $delta_1$ equals *m*, as suggested in observation No. 1. Therefore, unless *tc* matches the last character of *S*, the pointer to P is incremented by $delta_1$. An example of this case is shown in figure 4.3.

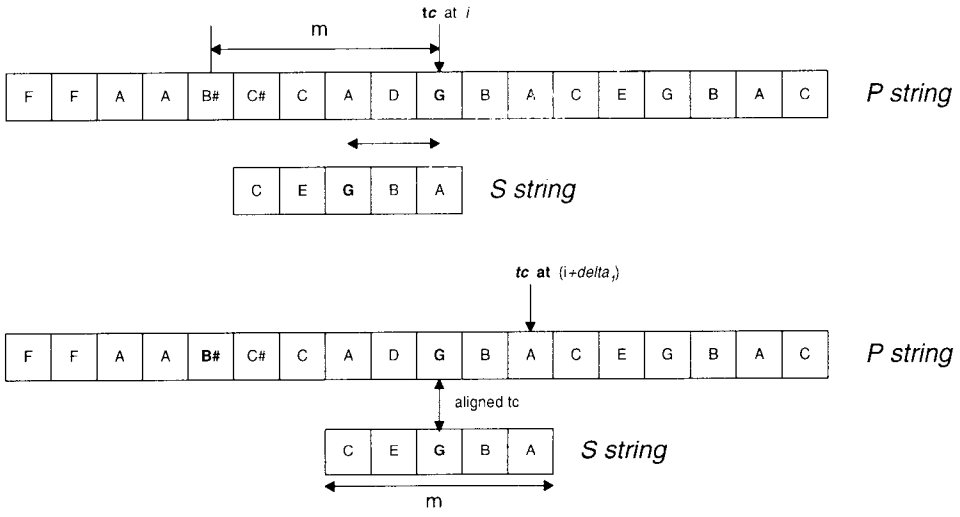


Figure 4.3
Example for Observation No. 2 of BM Algorithm

Actually this observation handles another case. If *tc* matches the last character of *S*, we have to determine if the previous character of *P* matches the previous character of *S*. If so, the process continues until a mismatch is

encountered. If no mismatch is found then there is an exact match and the next target, *tc*, is located *m+1* positions to the right of pointer *i*. *m* is added since the match always starts at the rightmost position of *S*. If there is not an exact match, a mismatch was found after matching the last *w* characters of *S*. This particular case is handled in observation No. 3. Notice that the character **G** in figure 4.3, was found two positions before ($delta_1$) with respect to *m*; therefore, *i* is moved two positions ($delta_1$) to the right.

Summarizing, observation No. 2 handles the general case for observation No.1 and a particular case of matching, exact matching. After an exact match is found, the pointer to *P* is increased by *m+1*.

4.2.3 Observation No. 3a

In this case, the increment of *i* is not based on the position of last character of *S*, but rather on the position of the mismatch found after matching *w* characters. The basic idea is applied again. This consists of aligning the occurrences of *tc* found in *S* and *P*. An example of this case is shown in figure 4.4. The pointer *i* is incremented by $delta_1$. Recall, $delta_1$ is defined as the distance between the last character of *S* at *m* and the position where *tc* was found in *S*. Therefore, *S* should be shifted by ($delta_1$ - *w*) positions, so the pointer *i* will be increased by [($delta_1$ -*w*) + *w*] = $delta_1$.

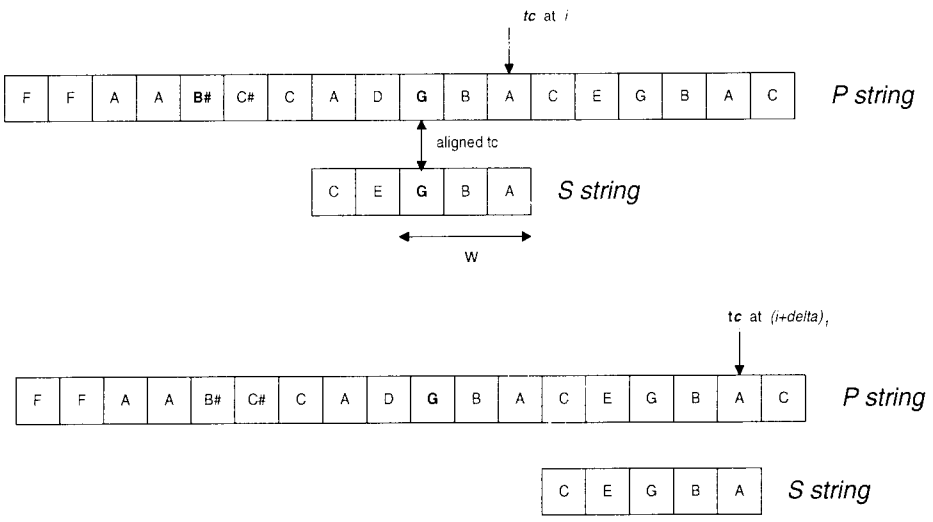


Figure 4.4
Example for Observation No. 3a of BM Algorithm

This observation also considers the possibility that the occurrence of *tc* in *S* might be at the right side of the mismatched character. If this happens, the pointer to *P* would be moved backwards to align *tc* with its match in *S*. This could end up in an endless loop where the pointer to *P* is moved back-and-forth within a certain range. This situation is avoided by Horspool by only comparing the last character of *S* and *P*. The main idea is to perform the jumps based on the last character of *P*.

Since $delta_1$ depends on *S* and the characters in Σ, all the possible $delta_1$ are preprocessed and stored in a table called $delta_1$. This table has an entry for each character in Σ.

4.2.4 Observation No. 3b

This observation considers the case where there is a substring of *S*, *subS*, that is repeated in S. An example of this case is *S*: FDEABFDE. The *subS* is FDE, a smaller pattern within *S*. Taking the substrings into account, another table, called $delta_2$, is processed. This table has as many entries as characters in *S*. The objective of $delta_2$ is to optimize the algorithm when substrings are encountered in *S*. Longer jumps for pointer *i* can be achieved. The main idea is to align the leftmost occurrence of *subS* in *S* with its occurrence in *P* after *w* characters have been matched. The logic of this table is explained in [19]. For the algorithm, pointer *i* is incremented by the greatest amount between $delta_1$ and $delta_2$. This part was omitted, since, as Horspool mentioned in his analysis, $delta_2$ does not contribute significantly to the speed of the algorithm [20]. Also, according to Baeza-Yates [21], the Horspool implementation of BM algorithm is considered the fastest. Horspool compared several approaches such as scanning for the first character, scanning for the least frequent character, and finally the BMH algorithm, (H) for Horspool.

4.3 Boyer-Moore-Horspool Algorithm

Boyer-Moore-Horspool Algorithm (BMH) is a string matching algorithm that searches a large block of text to find the first occurrence of a substring. This algorithm, however, was implemented to find all the possible exact

matches of S in P. The theory behind this algorithm was explained in section 4.2. This algorithm improved the BM algorithm by merging the two tables proposed by Boyer-Moore, $delta_1$ and $delta_2$, into one called $delta_{12}$. This is basically the $delta_1$ table with a slight change, $delta_1[s_m]$.

The implementation has the following modules:

MODULE NAME	MODULE DESCRIPTION
cal_delta12	calculates $delta_{12}$ for all characters contained in Σ. They are initially set to m since all the characters in Σ that do not appear in S produce an increment of m. This was described in observation No. 1.
BMH	executes the core of the BMH algorithm
AllExactMatch	extension of the BMH algorithm to find all the occurrences of S in P. Basically once an exact match is found, the BMH algorithm is applied to the rest of sequence P, starting at the location after the previous exact match in P.

Table 4.1
Description of Exact Match Modules Using BMH Algorithm

4.3.1 Pseudo-code for BMH algorithm

```
Initialization:
    int i = m - 1+ index          // i is a pointer to P;
    int j = m -1                  // j is a pointer to S;
                                  // index is used to extend algorithm
                                  // P and S are 0 based arrays
while (i <  last location of P)
     if ( P[i] = = S[j]){         // last character matches
       if( P[index... i] = S)  return i-m+1;  // there is an exact match
    }
    i = i + delta12[P[i]]
    j = m -1;
end while
return -1                         // if no exact match returns -1
```

4.3.2 Pseudo-code for AllExactMatch

```
 caldelta12                       // initialize table delta12
Initialization:
    int nm = 0                    // counter number of matches
    int index = 0                 // keeps track location of P
    int next_n = m-1              // possible next loc of P
   Macth[nm] = BMH(S, m, P, n, index) // finds first exact match
end Initialization

while ( a match exists   and   next_n < last loc of P)
     index = Match[nm] + 1        //increment to i to reach first loc. of
                                  // unexplored P
     nm++
    Match[nm] = BMH( S, m, P, n, index)
    next_n = Match[nm] + m
end while
return 0
```

4.4 Ranking Model Algorithm

The Ranking Model Algorithm is a weighted searching algorithm that allows approximate matches of S in P. It initially assigns an equal weight to each character in the substring of *P*, called *Ps*, that is being compared to S. These weights are then adjusted according to a model that determines the proximity between the characters. The initial weight for each character of *Ps*, in term of percentage, is given by,

$$initial\ weight = \frac{100\%}{length(S)} \qquad (\text{eq. } 4.2)$$

This is the maximum contribution of each character in *Ps*. The final percentage assigned to a particular match is the sum of the contributions of each character in *Ps*. The contribution factor is a vector that contains the contributions, *cf*, of each element of *Ps*. The match percentage is given by,

$$match\ percentage = initial\ weight * \sum_{j=0}^{m-1} cf_j \qquad (\text{eq. } 4.1)$$

4.4.1 A Simple Model

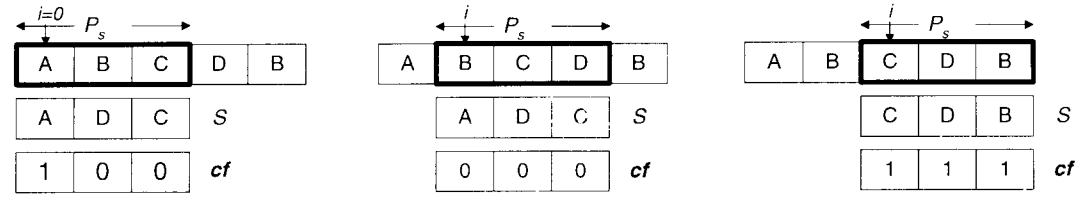
In order to illustrate the Ranking Model Approach, a simple model is described. In the next section, a more elaborate approach is presented.

A simple model is as follows:

$$cf_i = \begin{cases} 1 & if \quad p_{si} = s_i \\ 0 & if \quad p_{si} \neq s_i \end{cases}$$

Let P be the sequence: A B C D B, and S: C D B. Since the length of S is 3 characters, the initial weights for each character of *Ps* is 33.33%. *Ps* is a substring of *P* that is being compared to *S*. As in previous sections, let *i* be a pointer to *Ps*.

Figure 4.5 illustrates this example. For i = 0 the contribution vector is [1 0 0], therefore the match percentage at location 0 in P is 33.33% (eq. 4.1).



a. Compare at *i=0* b. Compare at *i=1* c. Compare at *i=2*

Figure 4.5
Approximate Matching Example for Simple Model

Percentage of S within P by starting location	Match Percentage
0	33.33%
1	0.00%
2	100.00%

Table 4.2
Matching Percentages for Simple Model

4.4.2 Frequency-Based Ranking Model

The model described in this section is based on:

a. The proximity of musical notes from the piano keyboard.

b. The frequency spectrum of musical notes

Since this model is based on frequency, the elements of P and S are compared as frequencies rather than as characters. The construction of the model starts by characterizing the proximity of the musical notes based on the theoretical fundamental frequency of each key in S (F_{S0}). Keys in P, whose resolved fundamental frequency (F_{P0}) are relatively close to F_{S0}, will have a contribution factor greater than the keys that are further away. Keys under test correspond to keys in the secondary musical file since they are the keys that are to be matched within the primary musical file. A possible model for this characterization is illustrated in figure 4.6. The curve is centered on the frequency of the key under test. Primary notes with frequencies on the curve are given contributions accordingly. The model is generated based on F_{S0}. The implementation of this model is time consuming since the exponential function is simulated as a piece-wise linear function. This exponential approach allows a contribution factor to those keys close to the key under test.

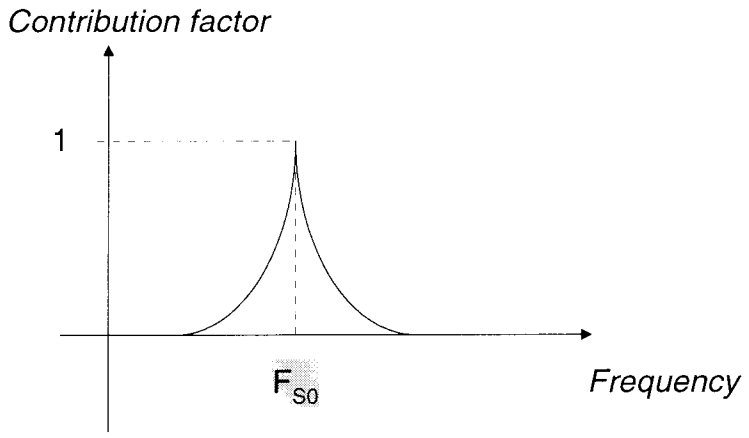


Figure 4.6
Frequency Proximity Characterization of Piano Key under Test

An approximation to this model is shown in figure 4.7. This model is simplified by replacing the exponential function with a simple linear function.

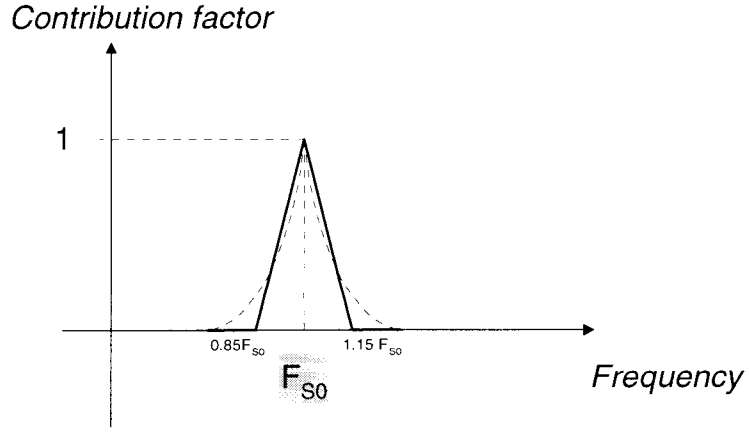


Figure 4.7
Simplification of Frequency Proximity Model of S

From the implementation point of view, this is less complex since the function can be easily generated using the endpoints of each line. The

contribution factor is non-zero for values of F_{P0} in the range [$0.85F_{S0}$, $1.15F_{S0}$]. This range was chosen empirically. To visualize this part of the model, let:

s_i be key #49, the key under test in S, with F_{S0} at 440 Hz

p_i be key #48, the key under test in P, with F_{P0} at 410 Hz.

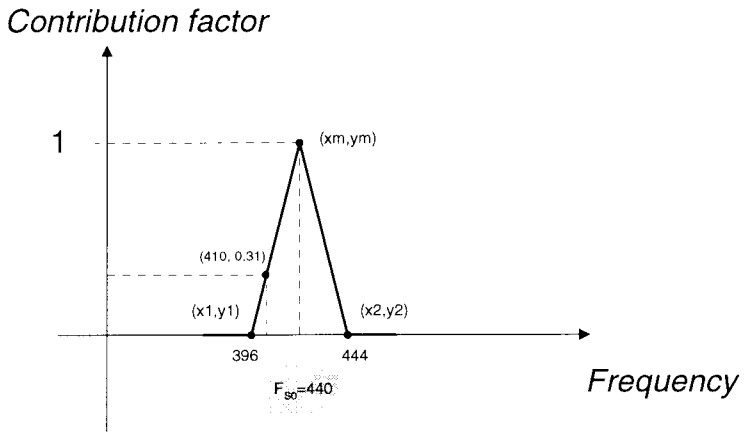


Figure 4.8
Frequency Proximity Model of Piano Key #48

By inspecting the model in figure 4.8, key #48 with F_{P0} = 410 Hz, has a contribution factor equals to 0.31. What this shows is that different keys can be compared with varying matching percentages. This model is therefore a function of both keys. *Frequency model* = $f(F_{S0}, F_{P0})$. The second part of the model will now be explained.

According to the frequency spectrum of a musical note, it has a fundamental frequency component, and harmonics at integer multiples of its fundamental frequency. A model should also recognize this property by

allowing a contribution factor to those keys whose fundamental frequencies are harmonics of the key under test. A model is illustrated in figure 4.9. The contribution factor decreases with a ratio of ½ for subsequent harmonics.

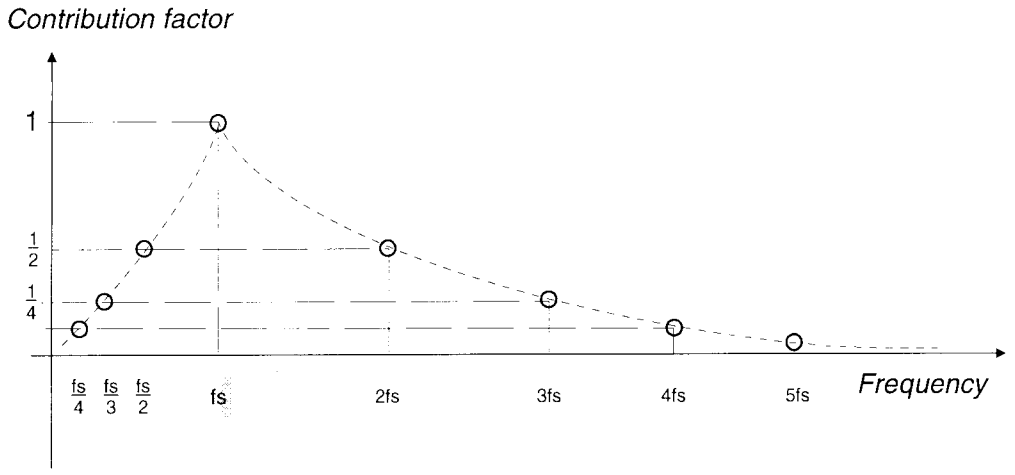


Figure 4.9
Characterization of Harmonics of Piano Key under Test

The two models (figure 4.7 and figure 4.9) are unified in order to create a ranking frequency model that will consider the proximity of keys in terms of frequency and the basic properties of a musical tone. In this model the characterization of harmonics will act as the envelope of the frequency model discussed previously. This model is illustrated in figure 4.10.

The boundaries on the frequency axis were determined empirically. Since the contribution factor decreases rapidly with the number of harmonics, the frequency-proximity model was computed for just a few harmonics.

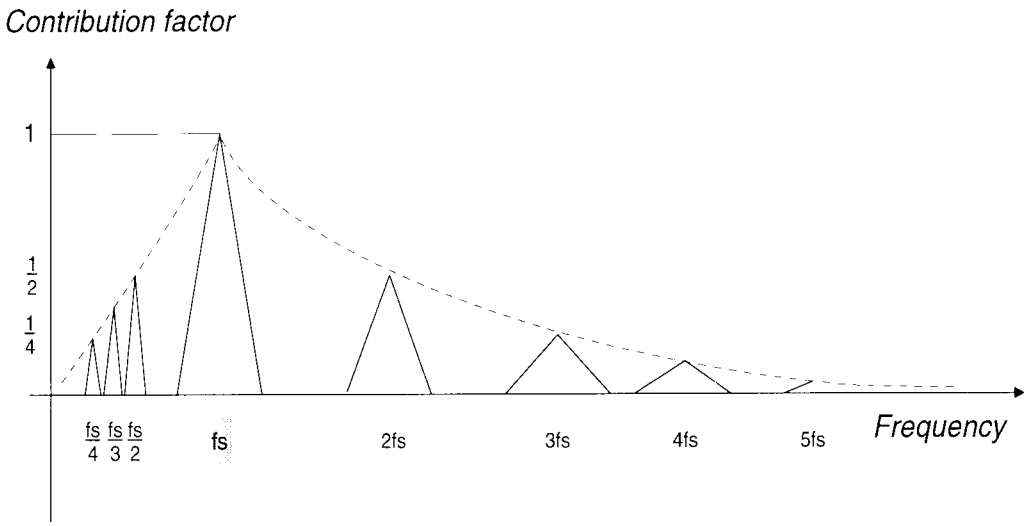


Figure 4.10
Frequency-Based Ranking Model

A limit of at most five harmonics became the boundaries for the frequency axis in either direction. The upper and lower boundaries are also restricted to the frequency span of the piano scale 27 Hz - 4186 Hz.

4.4.3 Implementation of the Frequency-Based Ranking Model

Each substring *Ps*, is compared with sequence *S*, as described in section 4.2.1. Initially a maximum percentage weight is given to each key in *Ps* (eq. 4.1) This percentage is adjusted according to the contribution factors given by the frequency ranking model. A vector, ***cf***, is generated for each *Ps*, and eq. 4.2 is applied. An accumulative count for the match percentage can be kept so that when the minimum approximation level is not possible to reach (see figure 4.1), the current comparison is aborted and the comparison moves to the next *Ps*.

A frequency-based ranking model is generated for each key in S and is computed as needed. For each curve the ending points are stored. The lines are stored according to arrangement shown in figure 4.11. This figure does not represent a properly scaled model.

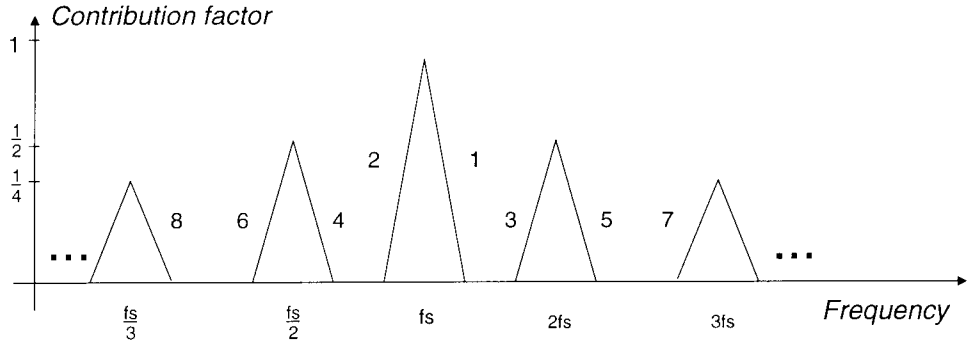


Figure 4.11
Line Generation Order for Frequency-Based Ranking (not to scale)

A search is made to determine in which region the primary frequency can be found. If the primary note is found within a region, its contribution factor is determined by linear interpolation. If the primary note is not found, it will have a contribution factor equal to zero.

4.4.4 Example for Frequency-Based Ranking Model

Let Primary and Secondary be sound files which have been converted into the following note sequences. Primary: { 40-40-42-40-45-44-40-40-42-40}, and Secondary: {40-40-54}

The result of the frequency-based ranking model for the first substring of P is shown in figure 4.12.

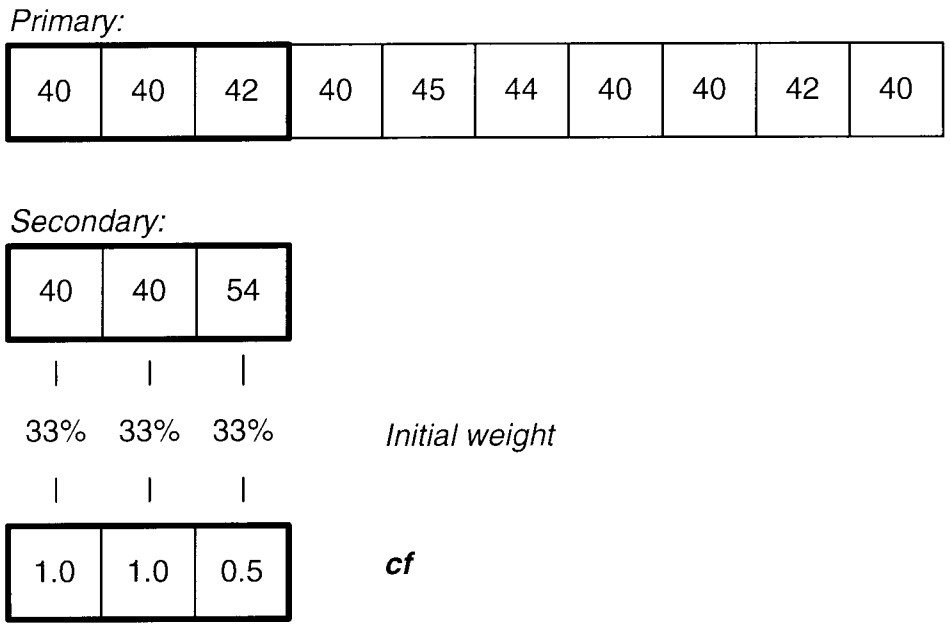


Figure 4.12
Example for Frequency-Based Ranking Model

Since the first and second notes are the same for both sequences, a contribution factor of 1.00 is assigned to each note. The third note in the Primary is an octave lower than the third note in the Secondary; therefore, the contribution factor for that note is 0.5. The final match percentage is:

$$match\ percentage = \left(\frac{1.0}{3} + \frac{1.0}{3} + \frac{0.5}{3}\right) * 100\% = 83\%$$

This algorithm is then applied to all the substrings of the Primary. A list of all the matches with a percentage greater than or equal to the desired approximation level is generated. This list, which is presented to the user, contains each match percentage and location.

5. GRAPHIC USER INTERFACE

The User Interface is a critical link in transforming abstracts ideas into a concrete presentation. It facilitates visualization that is not possible with pre-worked examples and static figures. This chapter describes the design and operation for *NoteIt*, the User Interface of this thesis.

5.1 Motivation

The User Interface is useful component of this thesis since it provides a method for the user to interact with the monophonic music recognition model. The goal of the User Interface is to provide a pleasant environment where the user has sufficient tools for problem formulation, analysis, and post-analysis presentation. In creating the GUI, I attempted to create a complete workshop for the recognition and placement of monophonic piano music.

A variation of a typical text-based User Interface is the *Graphic User Interface*. The Graphic User Interface, or GUI, is the modern technique of simultaneous presentation of text and graphics [22]. A GUI is appropriate here in order to easily interact with many graphical objects generated by the analysis. It seems to be a natural way to present the thesis, especially given the recent demand for GUI applications in the industry.

This GUI operates in the Windows NT environment. Windows NT is a widely accessible platform which provides multitasking. A contributing reason

for this choice is the availability of a superb software development environment, Microsoft Visual C++.

5.2 Organization

In this chapter the following topics are addressed: Getting Started, Handling Sound Files, Start-to-Finish Analysis, and Productivity Topics. Hard copies of actual screen displays are included to assist in the descriptions.

Getting Started

- Minimum computer equipment
- Starting the program
- Menu organization
- Setting user preferences
- Using different graphics colors

Handling Sound Files

- Supported sound formats
- Recording a sound file
- Trimming a sound file
- Saving the current sound file
- Loading a stored sound file
- Using sound playback

Start-to-Finish Analysis

- Starting a new project
- Loading the primary sound file
- Identifying notes within the primary file
- Loading the secondary sound file
- Identifying notes within the secondary file
- Searching for the secondary within the primary
- Selecting and viewing the search results
- Saving the project

Productivity Topics

- Using the previous project
- Performing an automated analysis
- Using the integrated FFT display tool

5.2.1 Getting Started

The GUI was created to run on an Intel Windows NT platform, specifically a Pentium or later processor, compatible with the x86 architecture.

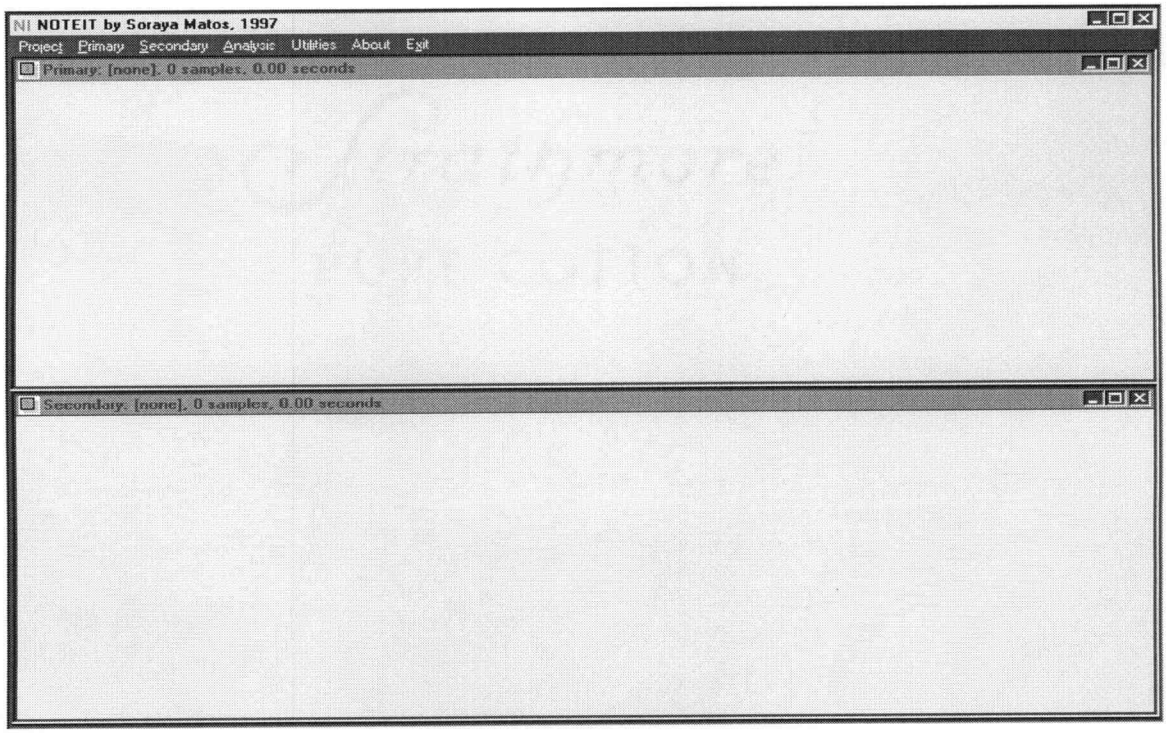


Figure 5.1
Initial GUI screen

NoteIt is the name of the GUI. When the GUI starts, it first shows an introduction screen, and then immediately displays its main menu. The GUI can be seen in figure 5.1. Also shown are two separate windows known as the Primary and Secondary windows. The top window, named Primary, contains a complete sample of musical notes, whereas the bottom window, named Secondary, only contains a subset of musical notes to be located within the Primary by the analysis.

If real-time recording is desired, a tuned piano and microphone should be available. If either recording or playback is desired, a standard Windows sound card is necessary. It is possible, however, to run the program without a sound card. In this case all inputs should be available as sound files prior to analysis. Low level Windows sound input/output routines are utilized to provide extensive control over the A/D and D/A interface [23] [24].

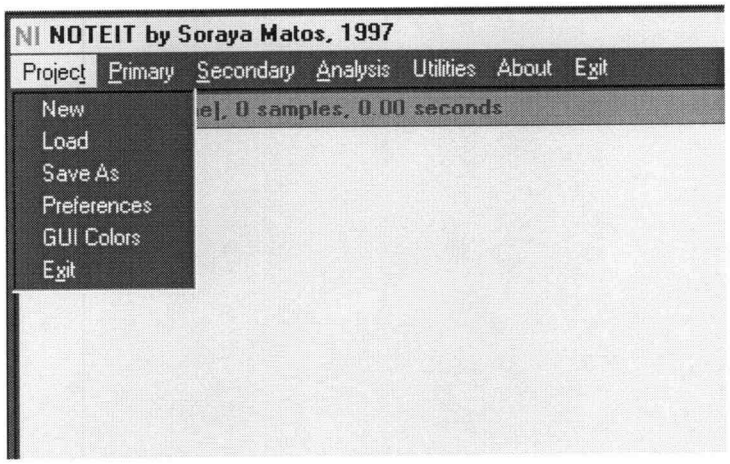


Figure 5.2
GUI Main Menu with Project Operations

The GUI main menu, seen at the top of figure 5.2, contains commands for both the Primary and Secondary windows. All options for the GUI can be found within these menus. Under the 'Project' menu, options for customizing the GUI are available.

The user may either select 'Preferences' or 'GUI Colors'. If Preferences is selected, the user can choose (1) between Time Domain or Frequency Domain for the audio monitor, discussed in Section 5.2.2, (2) instrument type, and (3) minimum acceptance matching level for the searching algorithm. The default values are for: time-domain, piano as the instrument, and 75% acceptance level. The Preferences dialog box is shown in figure 5.3.

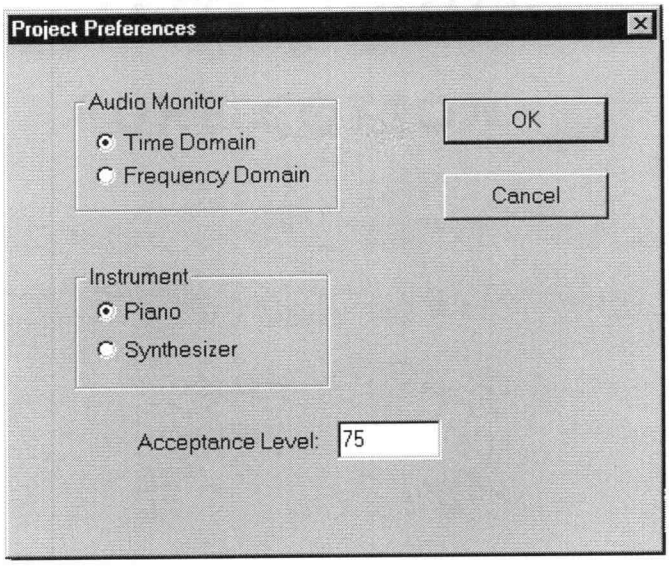


Figure 5.3
GUI Project Preferences

Colors may also be selected to by the user to aid with the presentation of graphics. Under the 'GUI Colors' option, the user may customize colors for drawing the Primary, Secondary, and Analysis results. Colors may be changed using the dialog box in figure 5.4

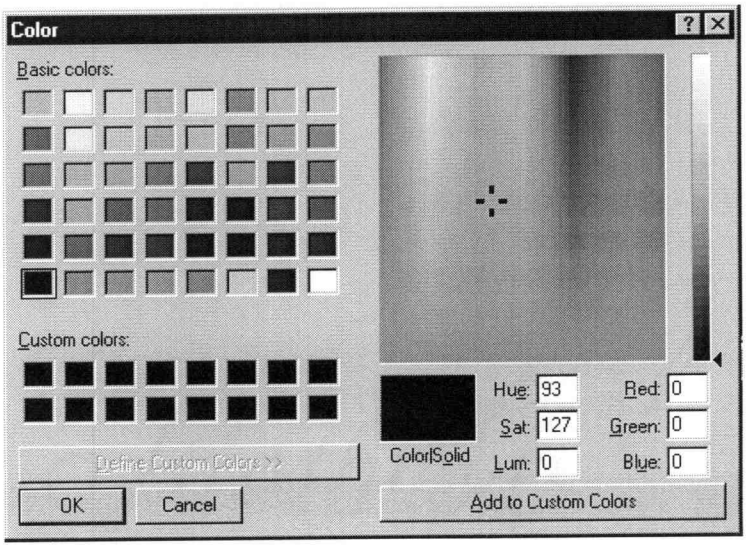


Figure 5.4
GUI Color Selection

5.2.2 Handling Sound Files

The GUI uses of two formats of sound files. The first format is the Riff Wave (.WAV) PCM format commonly used in Microsoft Windows. The second format is text mode with the extension .AUD. Wave files are stored in a non-compressed, binary format for quick loading and easy interface with other programs. Text format allows interfacing with programs which do not accept

WAV files. The sound configuration used within the GUI is by default WAV format with 16 bit A/D resolution with a sampling rate of 11.025 kHz.

The easiest way to access a new sound sample is to record it. The GUI can adequately handle samples as long as 300 seconds. The record option is available under either the Primary or Secondary menus as shown in figure 5.5

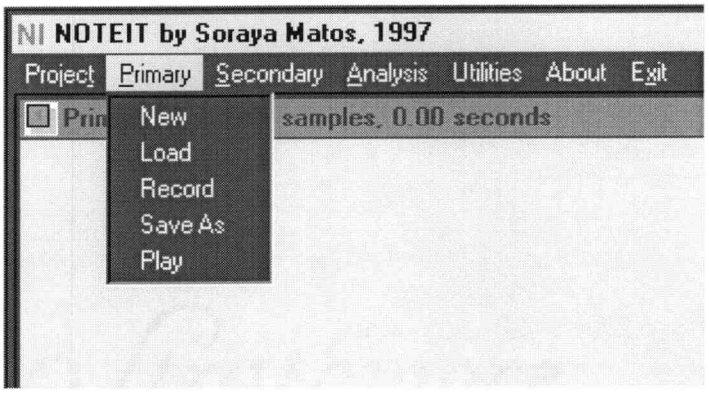


Figure 5.5
Primary Sound Operations

If record is selected, a dialog box appears asking the length (in seconds) for the recording. The recording process, once started, can not be interrupted. Trimming unused samples is permitted and will be discussed later. Once the recording commences, an audio monitor appears in the bottom left corner of the screen. The audio monitor uses either a time-domain or frequency-domain data representation, based on GUI preferences, for the incoming sound. This real-time monitor (figure 5.6) should be used to insure the recording is strictly monophonic; that is, each note finishes before the next note is started.

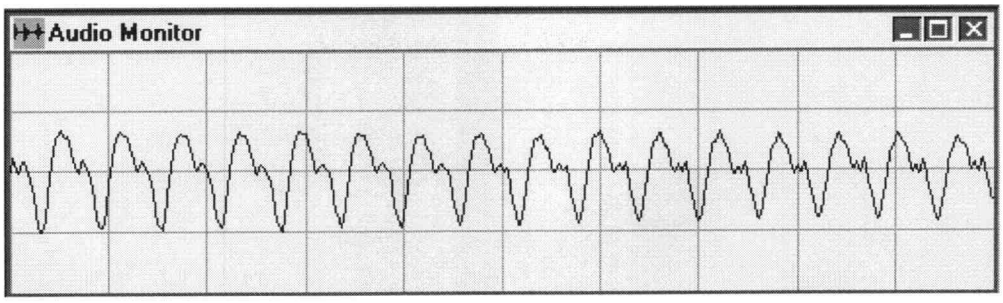


Figure 5.6
Real Time Audio Monitor

Once a recording is completed, it is displayed in the Primary or Secondary window. A grid is overlaid with horizontal divisions of 1 second in time. Vertical divisions are in multiples of 8192, 1/8 the maximum 16 bit PCM representation.

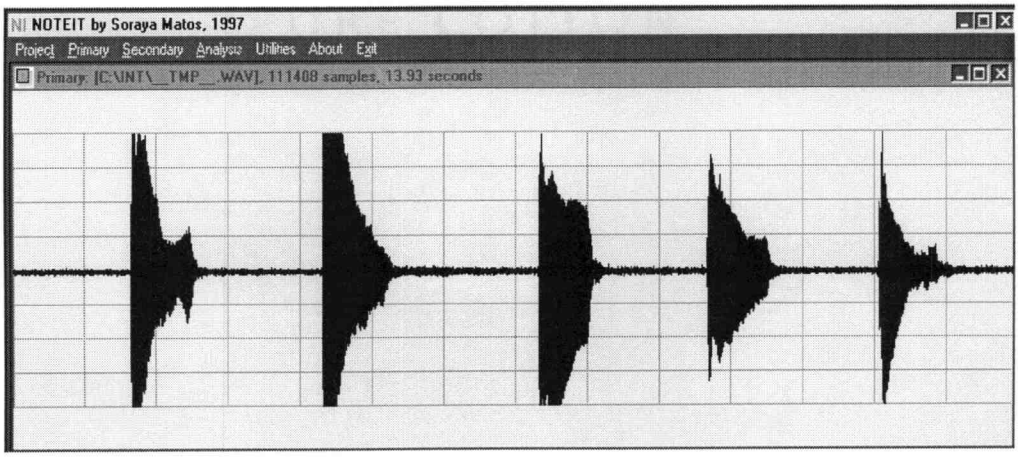


Figure 5.7
Monophonic Primary Sound After Recording

The recorded file is automatically given the name __TMP__.WAV and stored in the current disk directory. In most cases the user will want to store

the sound sample under a different name. This can be done with the 'Save As' option under either the Primary or Secondary menus. Prior to saving, the user may want to delete part of the recording. This is accomplished with the trim option. The trim option will keep a selected portion of the sample and delete the rest. To use the trim option, the user points to the start location with the left mouse button and clicks the right mouse button over the end location. This will determine the trim region. The trim region is shown on visually on the screen as an area enclosed between dashed lines as in figure 5.8. The user then selects the 'Trim' option from the menu. After a confirmation message, the sound samples *not selected* by the trim operation are deleted from memory. They, however, still exist in the disk file __TMP__.WAV. The user should then save the remaining sound samples into a different file.

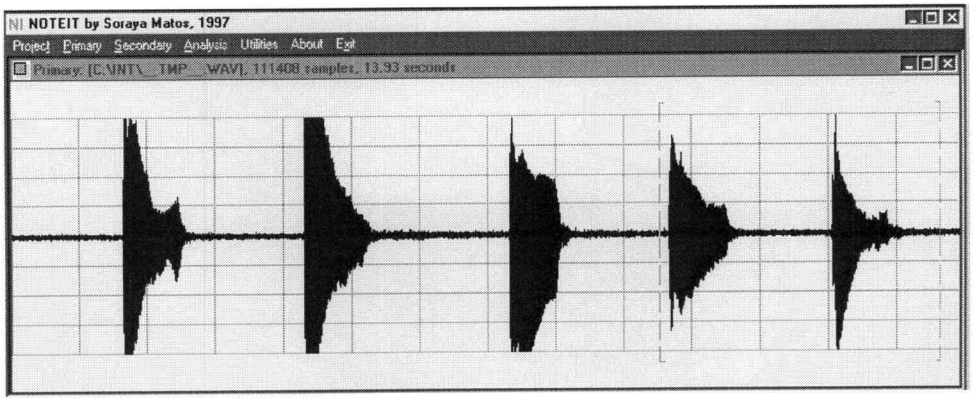


Figure 5.8
Selecting a Trim Region

Audio playback is the last option under the Primary and Secondary menus. It sends the sound sample to the D/A converter. Playback will also activate the Audio Monitor which was shown during the sound sample recording section (figure 5.6). While the sample is played back, a status line will be drawn across the display indicating the amount of playback remaining. It is not possible to terminate a playback once it has commenced. There is no difference between Primary and Secondary windows in the process of recording, loading, and storing sound samples.

5.2.3 Start-to-Finish Analysis

This section assumes that sound samples have been previously recorded and are suitable for performing an analysis. The first step is to create a new project. This will reset any GUI Preferences to their default state. A project contains a Primary sound sample and a Secondary sound sample. It is the duty of the analysis to perform segmentation and then analyze the samples to determine the appropriate musical notes that the samples represent.

This analysis will be accomplished step-by-step. In section 5.2.4 an automated analysis is discussed. After selecting 'New Project' from the main menu, the user should configure Preferences and then load a Primary sample. The load dialog box defaults to recognizing the WAV format. Files of this type are listed within the load box as seen in figure 5.9.

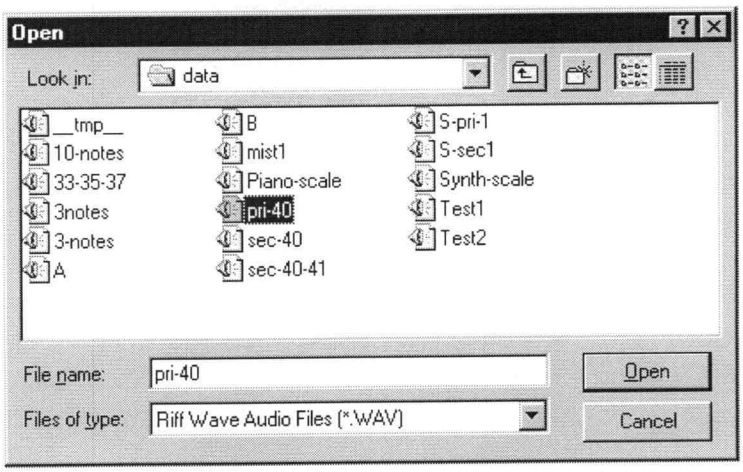


Figure 5.9
Loading a WAV File into the Primary Window

Once the file is selected and 'Open' is pressed, the sound sample will be shown in the main Primary window. The Primary file chosen for this example can be seen in figure 5.10. It is 25 seconds in length and has 11 monophonic notes defined. At this point the notes have not been identified.

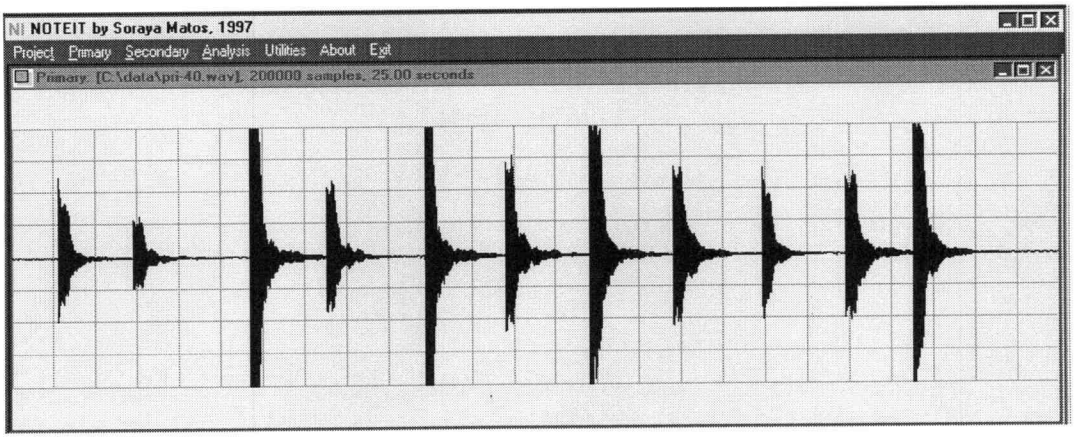


Figure 5.10
Primary Window with Sound Sample for Analysis

The next step is to analyze the Primary to determine the note identities. This is done by accessing the Analysis menu as shown in figure 5.11.

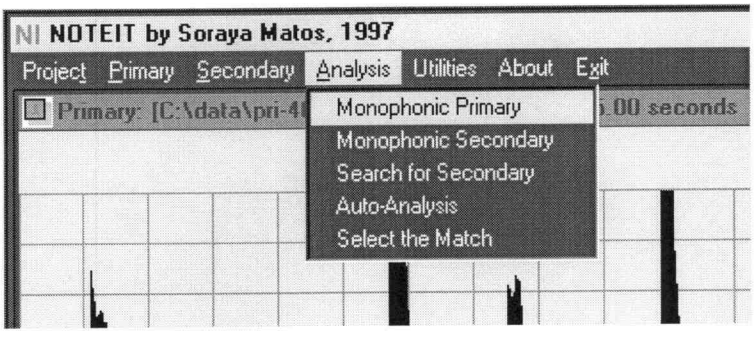


Figure 5.11
Selecting an Analysis Type

Once the option is selected to perform an analysis of the monophonic primary, a window is created to show the analysis steps. Refer to Chapter 3 for a detailed explanation of the analysis. When the analysis window appears, the first procedure is to click the top menu buttons in a left-to-right manner.

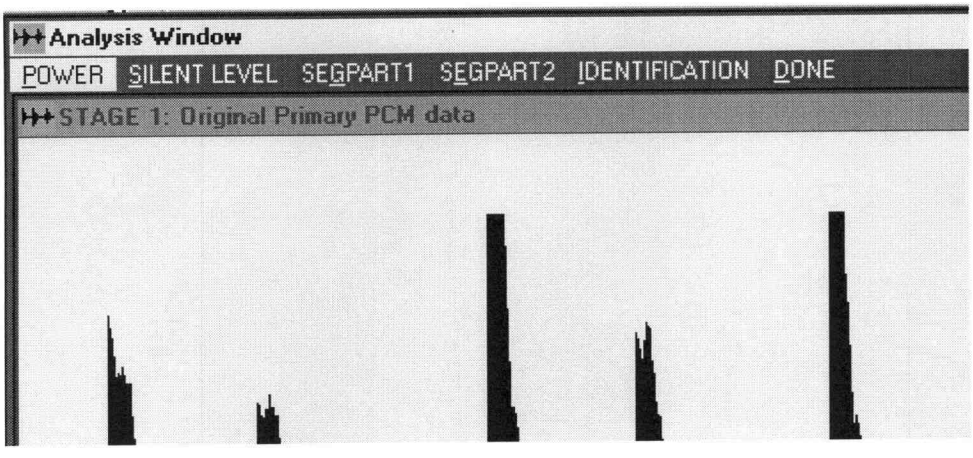


Figure 5.12
Monophonic Analysis Buttons

The user starts the analysis by clicking the POWER key as seen in figure 5.12. In the analysis window, the display is scrolled so that the previous step is located at the top of the window and the current step is located at the bottom of the window.

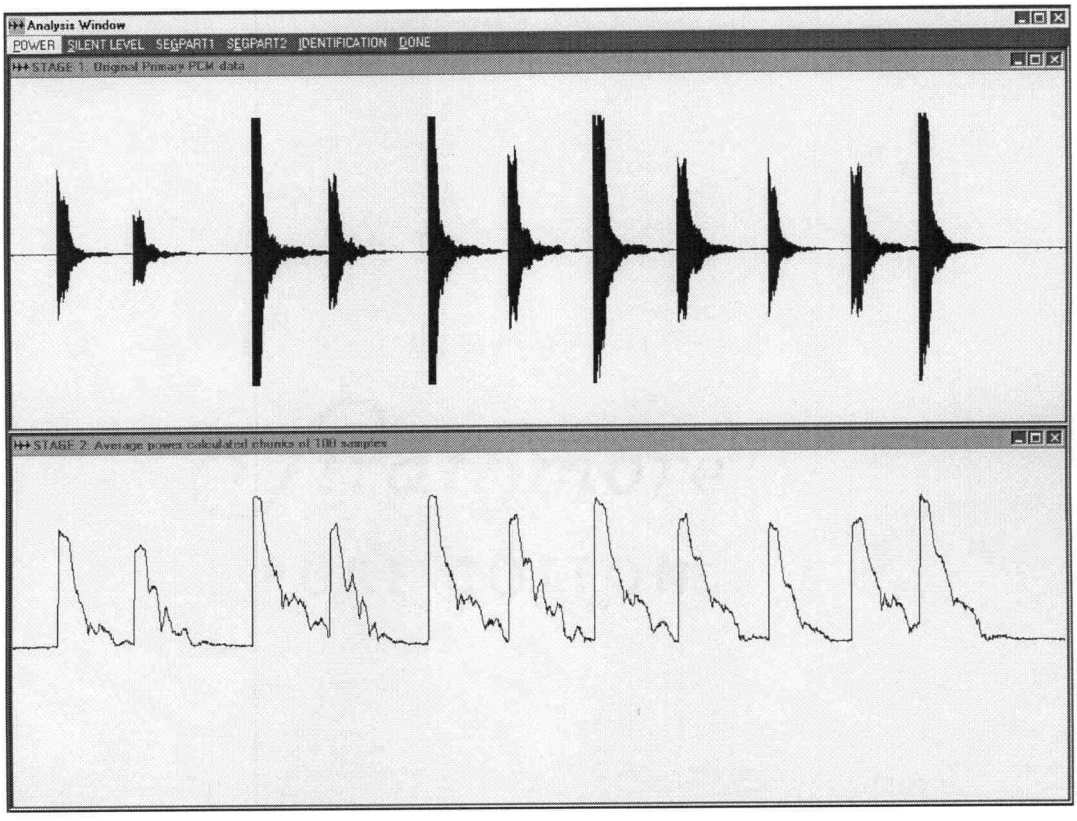


Figure 5.13
Monophonic Analysis: Average Power of the Primary

The first step was to open the monophonic analysis window and calculate the short-term power signal, the next step will be to determine the noise level and begin segmentation. The noise level is found by searching for the silent-state level at the beginning of the sound sample. This silence will

reflect a good estimate of the background noise. In figure 5.14 an upper limit to the noise power is determined and an initial segmentation is composed. This figure corresponds to pressing the SILENT LEVEL menu button followed by pressing of the SEGPART1 menu button.

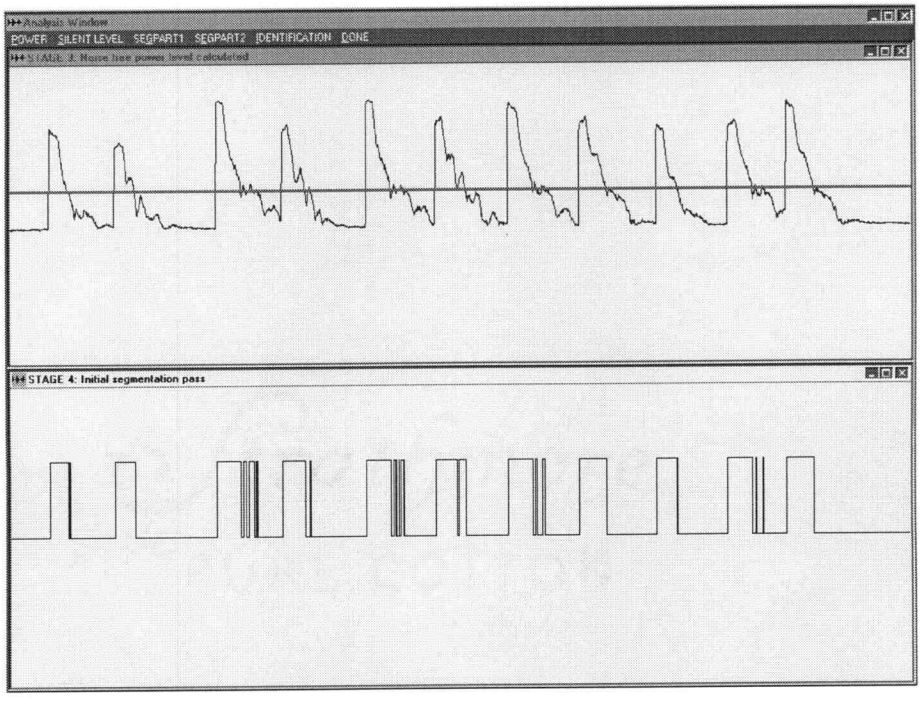


Figure 5.14
Monophonic Segmentation, Phase One

Figure 5.15 is then achieved by pressing SEGPART2 followed by IDENTIFICATION. If the same process is completed for the Secondary, an annotated graphic similar to the one in figure 5.16 is displayed. Above each note is its fundamental frequency. Below each note is the piano key that it represents (see Appendix for more information).

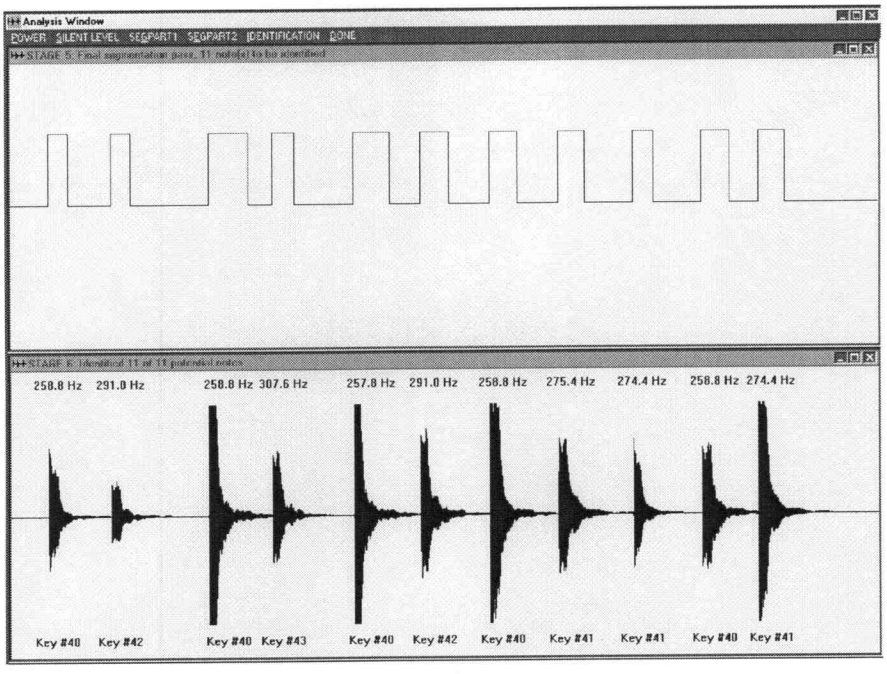


Figure 5.15
Monophonic Segmentation, Phase Two, with Note Identification

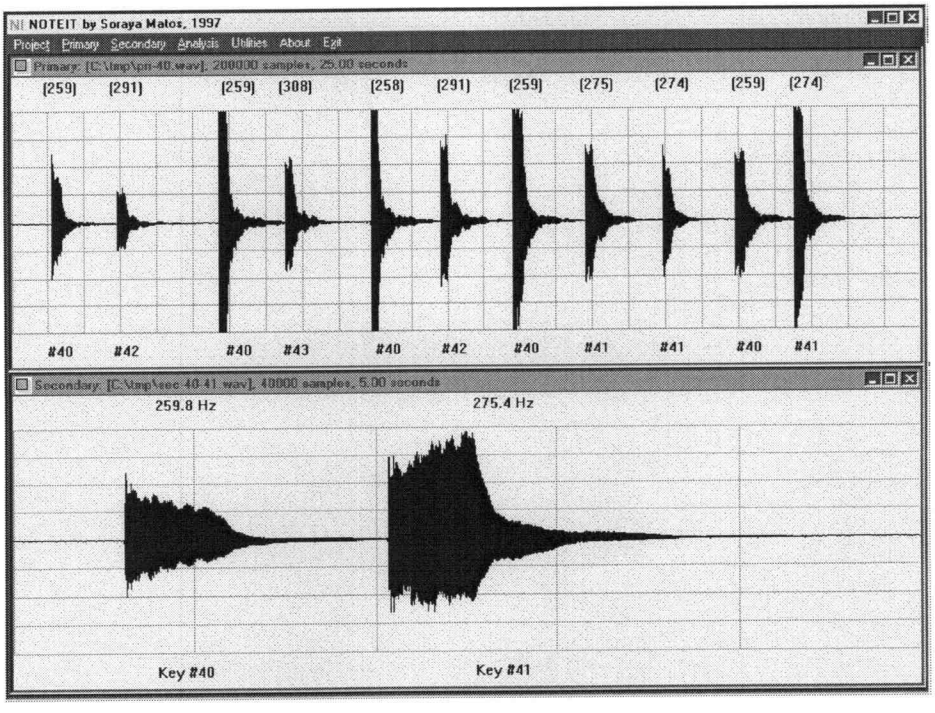


Figure 5.16
Note Identification for Primary and Secondary

The remaining topic is to search for the Secondary within the Primary. This can be accomplished by selecting the 'Search for Secondary' item from the analysis menu as seen in figure 5.17.

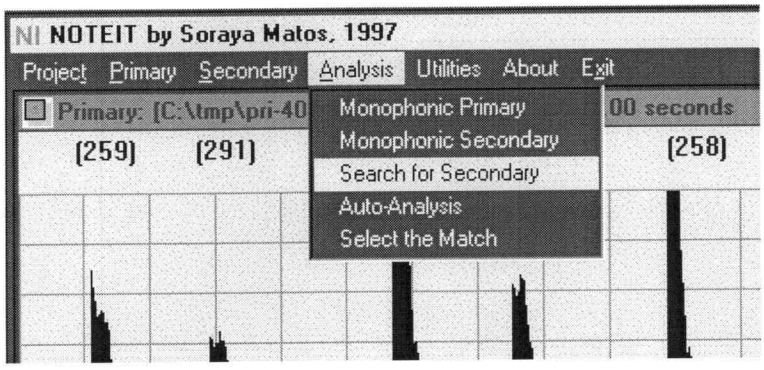


Figure 5.17
Searching for Secondary in Primary

When the search is completed a pop-up dialog box will show the results of the search process, figure 5.18.

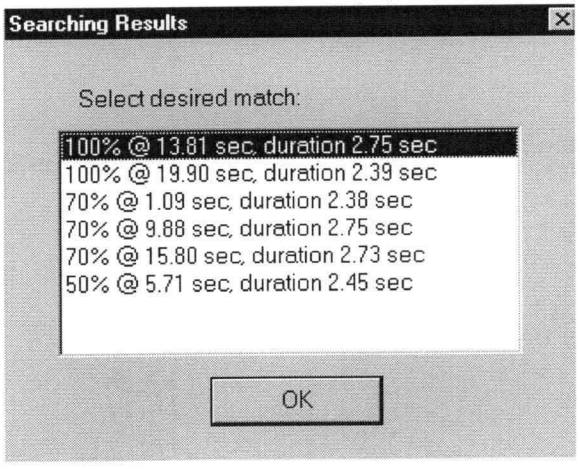


Figure 5.18
Searching Results

If the search preferences have been set to less than 100% acceptance level, there could be many entries listed. For clarity, the entries are sorted from best to worst match. Listed are: (1) the percent similarity, (2) the match location from the beginning of the Primary, (3) the length (duration) of the match within the Primary.

While looking at figure 5.18, the user is able to select one of the entries with the mouse. The matching segment is displayed in the Primary window for visual confirmation. Once dismissed, the results window can be made to reappear if the 'Select Match' option under the Analysis menu is selected.

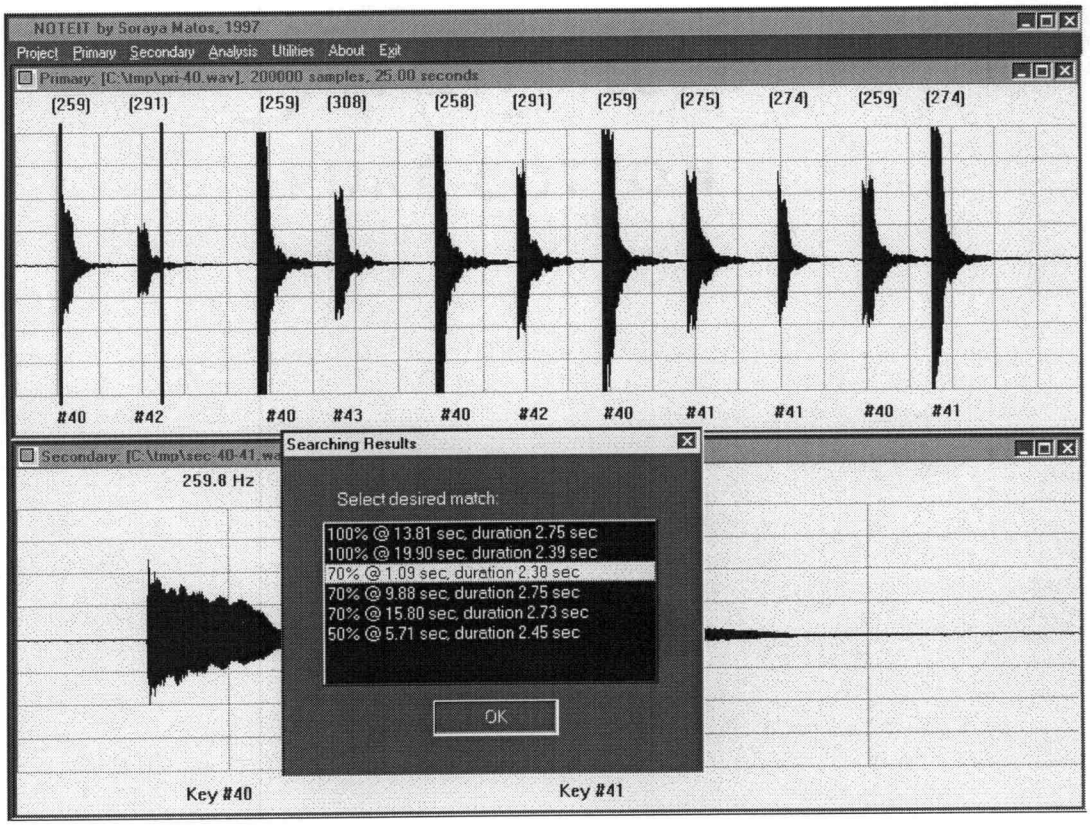


Figure 5.19
Automatic Identification and Searching Results

This section has demonstrated the steps and related graphic displays that are necessary to complete a manual analysis. The last step is to save the project for later retrieval. This can be done under the Project menu.

5.2.4 Productivity Topics

The number of analysis steps can be reduced by using some of the automatic services that the GUI provides. First, it loads the previous project. Then it loads the Primary and Secondary files used within the project. It does not re-run the analysis upon loading the project. There is, however, an option under the Analysis menu for performing an automatic analysis. An automatic analysis completes each of the steps between figures 5.13-5.19. The automatic analysis does not display the intermediate graphics and will jump directly to a graphic similar to figure 5.19. If the Secondary data can not be located within the Primary data, above the acceptance level, then a message is sent to the user.

As a debugging aid, an on-demand FFT viewer can be activated under the Utilities menu. Either Primary or Secondary sound samples can be specified as the FFT source by first selecting the FFT tool and then pressing the left mouse button at the beginning of the region to analyze. Various FFT data sizes and windowing functions are available. Consult the on-line documentation for more information about this.

6. CONCLUSIONS

Research for this thesis was undertaken with the goal of analyzing monophonic musical sounds. I desired to treat the sounds completely from start-to-finish, from their creation in the strings of the piano to their detection by frequency spectrum algorithms. The research culminated with the development of a Graphic User Interface which can be used for a practical application of search and retrieval. I will give an overview of the process which is used for monophonic sound data. This is similar to the steps that a user sees in the user interface.

6.1 Summary

Monophonic piano music was captured by microphone and digitized into 16 bit PCM samples at 11.025 kHz. The next step was to use the monophonic properties to extract each potential note from the PCM samples. To extract each musical note, the short-term power signal was calculated. The size of each segment was 100 samples. The result of this process was the definition of each region that contained a note in terms of power.

The boundaries of musical notes were resolved by assigning a specific state to samples in the PCM data. These states are *silence* and *note*. Silent level was calculated as the short-term power signal for the first 25 ms. The envelope was then compared with the silent level to determine the presence of a musical

note. This initial process was not sufficient since states could be improperly assigned due to noise during silent times or low amplitudes within the range of a musical note. It was corrected by passing the data through a filter that verified that the length of the data was sufficient to be considered a note. The algorithm of this filter was described with a flow chart shown in figure 3.8.

Once the boundaries of a note were determined, the frequency spectrum of each note was resolved by computing an 8192-point FFT of each potential note time-domain segment. Following the characteristics of a complex tone, probabilistic methods were applied to map its frequency spectrum to a particular musical note. For all analysis, an 8192-point FFT was sufficient to recognize the entire interval of frequencies comprising the piano keyboard.

Having two sound files available, each containing a list of musical notes, searching techniques were utilized to find the existence of one sound file within another. Two different techniques were used, text-based matching and frequency-based matching. Text-based techniques were only used for locating exact matches. The need for having a matching algorithm which could take into account the characteristics of musical notes was imperative. Therefore, a frequency based model was realized. This algorithm involves not only the proximity of a particular musical note with its neighbors, but also the relation of the fundamental with its harmonics. Approximate matching was accomplished with this method. The model for this type of matching was shown in chapter 4, figure 4.10.

6.2 Concluding Remarks

The Fast Fourier Transform was used as a base tool to analyze monophonic musical sounds. A careful analysis of the resulting spectrum allowed the identification of a musical note within the entire interval of the piano keyboard. This analysis was refined by collecting data from the full piano range and adjusting the algorithm to agree with each key pressed. The FFT was of great importance in this thesis and was implemented using a decimation-in-time radix-2 algorithm. A bit reversal algorithm of my own design kept data indexing operations to a minimum and enhanced the readability of the FFT algorithm. This algorithm was explained in detail in chapter 2.

The identification of a musical note involved a close examination of the frequency spectrum to select appropriate peaks. It used the distance between the peak locations to determine the fundamental frequency. This process basically simulates the calculation of the GCD of the frequencies where the peaks were found. It is a generic method to compute the GCD that involves floating-point data.

The Graphic User Interface is a beneficial tool to understand the concepts of analyzing musical sounds. It included, in a visual manner, the steps to transform a sound file from its raw PCM data to a list of musical notes. Moreover, it is a practical tool to implement data searching in the sense that a secondary sound file is to be located within a primary sound file. The searching

techniques use concepts from text matching as well as a model which was designed to perform approximate matching based on frequency characteristics of musical sounds.

The GUI includes modules that are not part of a analysis per se but create a robust environment to develop ideas and accomplish work related to monophonic musical analysis. Some of these modules are: create a project, define its properties, record, play, and trim a sound file, and useful FFT utilities with size and data windowing control.

6.3 Future Work

The GUI can be used demonstrate analysis concepts. It was intended to be a workshop for developing more research into the analysis of music. User interfaces can grow as much as users would like or as much as researchers would need.

As far as music is concerned, this work can be extended to allow transposition of the sound file. In other words, locating the sound file in a key signature different from the one that was recorded. Transposition could be included as a parameter in the searching model. A more practical solution will be to assign a key signature to the secondary sound file; in which case, the primary file would be transposed into that key signature before matching begins.

This work can be also extended to include other instruments besides the piano. Guitar, for example, could be studied immediately since it shares many traits with the piano. Furthermore, voice could be included. This would require more research to implement pitch detection. The GUI could then be used in vocal studies at music schools.

BIBLIOGRAPHY

1. Dixon, S., Multiphonic Note Identification, *Proceedings of 19th Australasian Computer Science Conference*, Melbourne, Australia, February 1996.

2. Dodge, C., Jersy, T. A., *Computer Music*, New York: Schirmer Books, 1985.

3. Roederer, J. C., *Introduction to the Physics and Psychophysics of Music*, Second Edition, New York: Springer Verlag, 1979.

4. Rigden, J. S., *Physics and the sound of Music*, Second Edition. New York: J. Wiley, 1985.

5. Moore, B., *An Introduction to the Psychology of Hearing*, Third Edition, London: Academic Press Ltd, 1991.

6. Hall, D. E., *Musical Acoustics: An Introduction*, C.A: Wadsworth Publishing Company, 1980.

7. Von Helmholtz, H., *On the Sensation of Tone as a Psychological Basis for the Theory of Music*, A.J. Ellis, transl. New York: Dover, 1954.

8. Proakis, J. G., and Manolakis, D. G., *Digital Signal Processing, Principles, Algorithm and Application*, Third Edition, New Jersey: Prentice Hall, 1996.

9. Shannon, C. E., Communication in the Presence of Noise, *Proceedings of the IRE*, Vol. 37, pp. 10-21, 1949.

10. Jerry, A. J., The Shannon Sampling Theorem - Its Various Extensions and Applications: A tutorial Review, *Proceeding of the IEEE*, Vol. 65, 1977, pp. 1565-1596.

11. Deller, J. R., Tom, Dick, and Mary Discover the DFT, *IEEE Signal Proceeding Magazine*, April 1994 , pp. 36-50.

12. Brigham, E. O., *The Fast Fourier Transform and Its Applications*, Englewood Cliffs, New Jersey: Prentice-Hall, 1988.

13. Rodriguez, J. J., An Improved FFT Digit-Reversal Algorithm, *IEEE Transactions on Acoustics, Speech and Signal Processing*. Vol. 37, No 8, August 1989, pp. 1298-00.

14. Rius, J. and De Porrata, D., New FFT Bit-Reversal Algorithm, *IEEE Transactions on Signal Processing*, Vol. 43, April 1995, pp. 991-4.

15. Press, W.H., et al. *Numerical Recipes in C*, Second Edition, Cambridge: University of Cambridge, 1992.

16. Lui, S. A., Landmark Detection for Distinctive Feature-Based Speech Recognition, *Journal of Acoustical Society of America*, Vol. 100, No. 5, November 1996, pp. 3417-30.

17. Pinto, N. B., Childers, D. G., and Lalwani, A., Formant Speech Synthesis: Improving Productivity Quality, *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 37, No. 12, December 1989, pp. 1870-87.

18. Pielemieir, W. J., Time-Frequency Analysis of Musical Signals, *Proceedings of the IEEE*, Vol. 24, September 1996, pp.1216-30.

19. Boyer, R., and Moore, J. S., A Fast String Searching Algorithm, *Communications of the ACM*, Vol. 20, No. 20, October 1977, pp. 762-772.

20. Horspool, R. N., Practical Fast Searching in Strings, *Software, Practice, and Experience*. Vol. 10, June 1980, pp. 501-506.

21. Baeza-Yates, R. A., and Gonnet, G. H., A New Approach to Text Searching. *Communications of the ACM*, Vol. 32, October 1992, pp. 74-82.

22. Schildt, H., Pappas, C., and Murray, W., *Osborne Windows NT Programming Series: Programming Fundamentals*, Vol. 1, New York: Osborne McGraw-Hill, 1994.

23. Schildt, H., Pappas, C., and Murray, W., *Osborne Windows NT Programming Series: General Purpose API Functions*, Vol. 2, New York: Osborne McGraw-Hill, 1994.

24. Schildt, H., Pappas, C., and Murray, W., *Osborne Windows NT Programming Series: Special Purpose API Functions*, Vol. 3, New York: Osborne McGraw-Hill, 1994.

25. Brown, Judith C., Calculation of a Constant Q Spectral Transform, *Journal of Acoustic Society of America*, Vol. 89, No. 1, January 1991, pp. 425-434

APPENDIX

The purpose of this appendix is to show the fundamental frequencies of an equal-tempered piano keyboard. The range of frequencies is 27.5 Hz - 4186.1 Hz, as shown in table A.1.

The piano usually consists of semi-alternating black and white keys. The keys are arranged in such a manner that the frequencies of the keys increase from left to right in a geometric series. The constant of the series is equal to the twelfth root of two, or approximately 1.059. This defines twelve keys per octave (a doubling of frequency). For example, there exists a white key on the keyboard called Middle-C for which the fundamental frequency is set to approximately 261.6 Hz. The key to the immediate right, a black key, and is set to (261.6 Hz) x 1.059, or approximately 277.2 Hz. If we proceed to the right, by the time we reach the thirteenth key, we double the frequency and thus cover an entire octave. This key pattern repeats every twelve keys.

If you chose the white key at 110 Hz, then the thirteenth key to the right will be at 220 Hz and your octave will have covered 110 Hz to 220 Hz. A graphical description of the piano keyboard is given in figure A.1.

The keys are enumerated consecutively from 1 to 88. The key on the far left of the piano is white and is labeled key #1, the immediate key to the right is black and it is called key #2, and so forth.

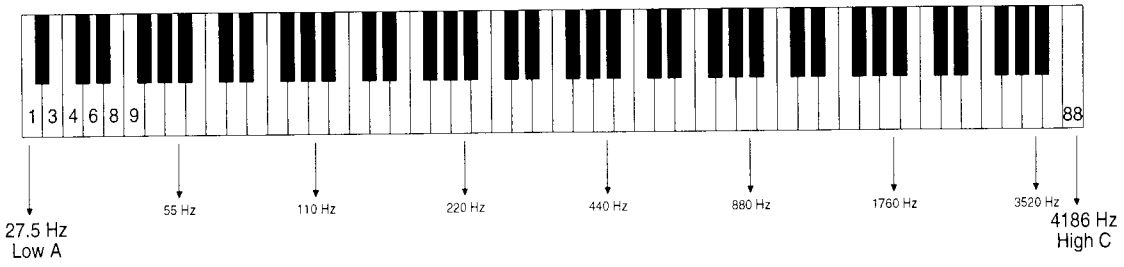


Figure A.1
The Typical Piano Keyboard.

Western music defines a standard octave at a fixed range of frequencies, called the 'Middle C octave'. It is also referred to as the Middle C scale. This scale is shown below in figure A.2.

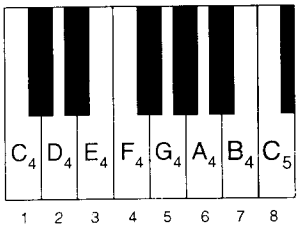


Figure A.2
The Middle-C Scale

An *octave* could contain notes between C_4 to C_5, since the fundamental frequency of C_5 is twice the fundamental frequency of C_4. In total, there are seven octaves on the piano. This gives 7 * 12 = 84 keys. There are 4 additional keys which do not cover a full octave, but are a part of the upper piano scale.

A standard digital sampling rate of 8 kHz is not sufficient to accurately capture the piano's spectrum. This is due to the fact that the highest frequency

(F_{max}) is 4186.1 Hz. A Nyquist sampling rate of at least $2F_{max}$ is necessary to avoid aliasing of high notes into low frequency. The next standard sampling rate above 8 kHz is 11.025 kHz, or ¼ the rate of standard digital audio, 44.1 kHz. 11.025 kHz was used within this thesis.

Key #	Note	Fund.
1	A_0	27.500
2	B_0^b	29.135
3	B_0	30.868
4	C_1	32.703
5	$C_1^\#$	34.648
6	D_1	36.708
7	$D_1^\#$	38.891
8	E_1	41.203
9	F_1	43.654
10	$F_1^\#$	46.249
11	G_1	48.999
12	$G_1^\#$	51.913

Key #	Note	Fund.
13	A_1	55.000
14	B_1^b	58.270
15	B_1	61.735
16	C_2	65.406
17	$C_2^\#$	69.296
18	D_2	73.416
19	$D_2^\#$	77.782
20	E_2	82.407
21	F_2	87.307
22	$F_2^\#$	92.499
23	G_2	97.999
24	$G_2^\#$	103.826

Key #	Note	Fund.
25	A_2	110.000
26	B_2^b	116.541
27	B_2	123.471
28	C_3	130.813
29	$C_3^\#$	138.591
30	D_3	146.832
31	$D_3^\#$	155.563
32	E_3	164.814
33	F_3	174.614
34	$F_3^\#$	184.997
35	G_3	195.998
36	$G_3^\#$	207.652

Key #	Note	Fund.
37	A_3	220.000
38	B_3^b	233.082
39	B_3	246.942
40	C_4	261.626
41	$C_4^\#$	277.183
42	D_4	293.665
43	$D_4^\#$	311.127
44	E_4	329.628
45	F_4	349.228
46	$F_4^\#$	369.994
47	G_4	391.995
48	$G_4^\#$	415.305

Table A.1
F_0 of the Equal Tempered Piano Keyboard

Key #	Note	Fund.
49	A_4	440.00
50	B_4^b	466.16
51	B_4	493.88
52	C_5	523.25
53	$C_5^\#$	554.37
54	D_5	587.33
55	$D_5^\#$	622.25
56	E_5	659.26
57	F_5	698.47
58	$F_5^\#$	739.99
59	G_5	783.99
60	$G_5^\#$	830.61

Key #	Note	Fund.
61	A_5	880.00
62	B_5^b	932.33
63	B_5	987.77
64	C_6	1046.5
65	$C_6^\#$	1108.7
66	D_6	1174.7
67	$D_6^\#$	1244.5
68	E_6	1318.5
69	F_6	1396.9
70	$F_6^\#$	1479.9
71	G_6	1567.9
72	$G_6^\#$	1661.2

Key #	Note	Fund.
73	A_6	1760.0
74	B_6^b	1864.7
75	B_6	1975.5
76	C_7	2093.0
77	$C_7^\#$	2217.5
78	D_7	2349.3
79	$D_7^\#$	2489.1
80	E_7	2637.1
81	F_7	2793.8
82	$F_7^\#$	2959.9
83	G_7	3135.9
84	$G_7^\#$	3322.4

Key #	Note	Fund.
85	A_7	3520.0
86	B_7^b	3729.3
87	B_7	3951.1
88	C_8	4186.1

Table A.1 (continued)