The 1996 Mississippi State University Conference on

Digital Signal Processing

What: EE 4773/6773 Project Presentations
Where: Simrall Auditorium, Mississippi State University
When: December 2, 1996 — 1:00 to 4:00 PM

SUMMARY

The Department of Electrical and Computer Engineering invites you to attend a mini-conference on Digital Signal Processing, being given by students in EE 6773 — Introduction to Digital Signal Processing. Papers will be presented on:

- parallel implementations of fast Fourier transforms;
- real-time audible frequency detection and classification;
- analysis of forestry images for scenic content.

Students will present their semester-long projects at this conference. Each group will give a 12 minute presentation, followed by 18 minutes of discussion. After the talks, each group will be available for a live-input real-time demonstration of their project. These projects account for 50% of their course grade, so critical evaluations of the projects are welcome.

Session Overview

1:00 PM — 1:10 PM: J. Picone, Introduction

1:15 PM — 1:45 PM: Michael Balducci, Ajitha Choudary, and **Jon Hamaker**, “Comparative Analysis of FFT Algorithms In Sequential and Parallel Form”

1:45 PM — 2:15 PM: **David Gray**, Craig McKnight, and Stephen Wood, “Audible Frequency Detection and Classification”

2:15 PM — 2:45 PM: Yaquin Hong, **Nirmala Kalidindi**, and Liang Zheng, “An Algorithm To Determine The Scenic Quality Of Images“

3:00 PM — 4:00 PM: Demonstrations in 434 Simrall

AUTHOR INDEX

Volume 2

Digital Signal Processing

Table of Contents

COMPARATIVE ANALYSIS OF FFT ALGORITHMS IN SEQUENTIAL AND PARALLEL FORM

Michael Balducci, Ajitha Choudary, Jonathan Hamaker

Parallel DSP Group
Department of Electrical and Computer Engineering
Mississippi State University
Mississippi State, Mississippi 39762
{balducci, ajitha, hamaker}@erc.msstate.edu

ABSTRACT

There have been a large number of Fast Fourier Transform (FFT) algorithms which have been developed over the years. Among these are the Radix-2 algorithm, Radix-4 algorithm, Split-Radix algorithm, Decimation-in-Time-Frequency algorithm (DITF), Quick Fourier Transform (QFT), and the Fast Hartley Transform (FHT). However, there has not been much prior work where the user is given a single interface to poly-functional implementation that transparently optimizes space and time complexity.

In this paper we present the implementation and benchmarking of the sequential and parallel versions of the above mentioned FFT algorithms. All algorithms have been rigorously compared based on computational time, object size, code size, data dependence (real or complex) and the number of mathematical operations involved in the computations. The results of this endeavor will serve as a frame for creating an object-oriented FFT environment which will automatically choose the most efficient algorithm for a given platform, data set, and order or other user-specified criteria.

1. INTRODUCTION

The development of Fast Fourier Transform (FFT) has evolved over many years. The first major breakthrough was the Cooley-Tukey algorithm developed in the mid-sixties which resulted in a flurry of activity on FFT's. [1] [2] Further research led to the development of the Fast Hartley Transform and Split-Radix algorithm. Recently two new algorithms have also emerged: Quick Fourier Transform and the Decimation-in-Time-Frequency algorithm. Now research has to be geared towards finding which of these algorithms is most efficient. The problem that arises is the inherent trade-off between computation speed, memory usage and the algorithm complexity. Instead, we must attempt to find the most efficient algorithm under the constraints of a particular application. Our efforts will accomplish this task by benchmarking each algorithm under a variety of constraints. The benchmark statistics will be used to create an automated environment capable of choosing the most efficient algorithm for a given application.

The FFT is one of the important and most widely used Digital Signal Processing (DSP) algorithms. FFT algorithms are efficient methods of calculating the DFT. The DFT converts the input signal from the discrete time domain, x(n), to the discrete frequency domain, X(w), and vice versa. This is very useful in eliminating the unwanted noise signal from any communication signal (information bearing signal) being analyzed. Once the signal is converted into the frequency domain the noise or unwanted signal frequencies can be effectively filtered. Then, by using the inverse FFT, the communication signal can be converted back to the time domain. The FFT has many wide-ranging applications in nearly every signal processing field including speech, image processing, communications, cellular phones, modems, and digital control systems. [3]

For most applications computation time plays a significant role in the use of FFT's. The more time spent computing means more efficient utilization of resources; hence, more data can be processed in the same time. The computation time can be reduced using the symmetry, periodicity, etc. of the DFT. The computation time can also be reduced using parallelism in FFT's. By using the FFT algorithms in parallel, the data set can be separated into smaller blocks. The different blocks of data can then be processed at the same time across multiple processors. There is, of course, a trade-off with this method: overhead of communication between processes. To make efficient use of the parallel method the communication time must be minimized.

1. Theory of FFTs

The Fourier transform, *H(w)*, of a signal, *h(t*), is given by the equation of Figure (1a), where *h(t)* is the time-domain signal, *t* is the time and *w* is the angular frequency. The assertion of this equation is that any time-domain signal can be "transformed" into a function of frequency. A plot of this frequency-dependent function gives the frequency content of a particular signal over the entire frequency spectrum. The resulting function contains the magnitude and phase information for each frequency point in the spectrum. This process of conversion from the time to frequency-domain is invertible using the Inverse Fourier Transform of Figure 1b. By the inverse transform, a frequency-domain signal is transformed back to the time domain. [1] [3]

$$H(w) = \int_{-\infty}^{\infty} h(t)\, e^{j2\pi t} dt \qquad 1a$$

$$h(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} H(w)\, e^{-j\omega t} \qquad 1b$$

Figure 1. Forward and Inverse Fourier Transform

The Discrete Fourier transform (DFT) is the digital equivalent of the Fourier Transform. It is a bounded length sequence which is more practical than the infinite summation of the Fourier Transform. The DFT assumes that the input signal is periodic with a period equal to the length of the input sequence. The Discrete Fourier transform is defined as shown in Figure (2a) and its inverse is shown in Figure (2b) [7].

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{\frac{-j2\pi kn}{N}} \qquad 2a$$

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)\, e^{\frac{j2\pi kn}{N}} \qquad 2b$$

Figure 2 Forward and Inverse Discrete Fourier Transform

The DFT is one of the most important concepts in digital signal processing but it is not useful for practical applications. In computing the DFT directly, there are $4N^2$ multiplications and N(4N-1) additions, therefore the computation time is of the order N^2. This type of complexity is not satisfactory for large N. We desire complexities which are linear with data size. At this time, no algorithm is available with such performance but we can derive a class of algorithms which give an efficient alternative to the DFT. These algorithms are collectively known as Fast Fourier Transforms (FFTs).

2. Algorithms

The Fast Fourier Transforms included here use several different approaches in reducing the computation cost of calculating the fourier coefficients. One method employed by several of the algorithms is the divide-and-conquer approach. These algorithms use the fact that a input sequence of length N can generally be broken into a number of smaller sequences. Since the DFT has an order of complexity N^2, breaking the summation into β sections of length-N/β results in a reduction of complexity as shown in Figure 3. [4]

$$\text{Complexity} = \left(\frac{N}{\beta}\right)_0^2 + \left(\frac{N}{\beta}\right)_1^2 + \underset{\cdots}{\cdots} + \left(\frac{N}{\beta}\right)_\beta^2 = \frac{N^2}{\beta}$$

Figure 3 Complexity of Divide-and-Conquer approach. Note the reduction in complexity from the $O(N^2)$ of the DFT.

Another approach is to utilize the periodicity of the DFT. From Figure 4, it can be seen that the multiplying factors of the DFT exhibit both horizontal and vertical symmetry about the unit circle when N is a power of two. Thus, by realizing that the coefficients repeat, redundant calculations can be eliminated. [3]

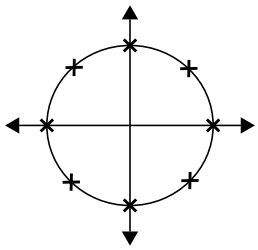


Figure 4 Plot of W_N on the unit circle. This complex function exhibits both horizontal and vertical symmetry when N is a power of two.

2.1. Radix-2 and 4 Algorithms

2.1.a. Sequential Form

By limiting our data-length to the form $N = R^V$ we can define a class of FFTs known as radix algorithms. These algorithms successively decompose a single N-point DFT into R segments of N/R-point DFTs. The most widely used of these radix algorithms is the Radix-2 and the Radix-4. Each uses the periodic properties of the DFT to attain higher efficiency levels. Radix algorithms can be implemented by either using Decimation-In-Time (Cooley and Tukey) or by Decimation-In-Frequency (Sande and Tukey). Each of these algorithms reduces the number of operations from $O(N^2)$ to $O(N \log_2 N)$. The drawback is that the data must be of a specified length. This problem can be avoided by zero-padding with no loss of information. [1] [5]

2.1.b. Parallel Form

The parallel structure of the radix algorithms is understood by considering the fact that the DFT can be decomposed into smaller independent DFTs. By performing each of these smaller DFTs concurrently, we can take advantage of this parallelism. Figure 5 shows two stages of a Decimation-In-Time FFT. In this figure, one can see that the 8-point DFT is decomposed into two 4-point DFTs. These, in turn, are each decomposed into two 2-point DFTs.[6]

.

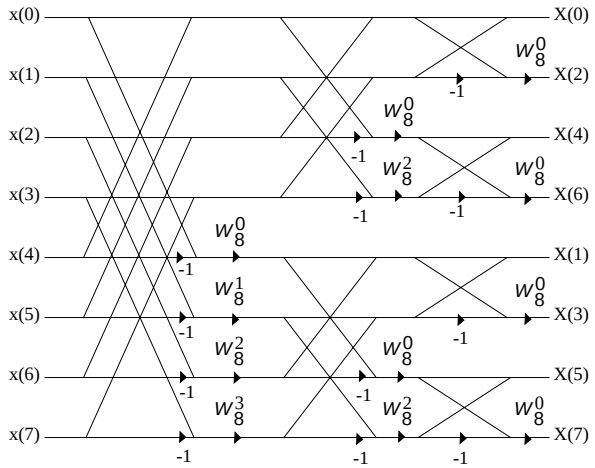


Figure 5 Flowgraph of an 8-point Decimation-in-Frequency FFT. The independence of the odd and even sample calculations allows for evaluation on separate processors

2.2. Split-Radix Algorithm

2.2.a. Sequential Form

By observing Figure 5, it can be seen that even index points can be calculated independently of the odd indexed points. This leads to the possibility of making use of more than one algorithm for the data set. The increase in computational efficiency of the higher order Radix-4 is attractive, but the limitation in data sequence lengths is a hindrance. The Split-Radix utilizes the fact that the data points can be decomposed into even and odd indices to employ both Radix-2 and 4 algorithms. A Radix-2 is performed on the even index points. The odd points are then decomposed into two N/4 point sequences where a Radix-4 approach is taken. In making use of these two techniques, the Split-Radix acquires an increase in computational efficiency over the Radix-2 while retaining the its ability to perform on any power of two. [7]

2.2.b. Parallel Form

As discussed above, the Split-Radix algorithm is composed of a Radix-2 and two Radix-4 components. These components are independent of each other and, thus, can be performed in parallel separate processes. Figure 6 shows the flow of this process. Also, notice that the Radix-2 and Radix-4 components can each be performed as parallel computations (see section 2.1.b). In this manner, we can achieve maximum utilization of parallel hardware.

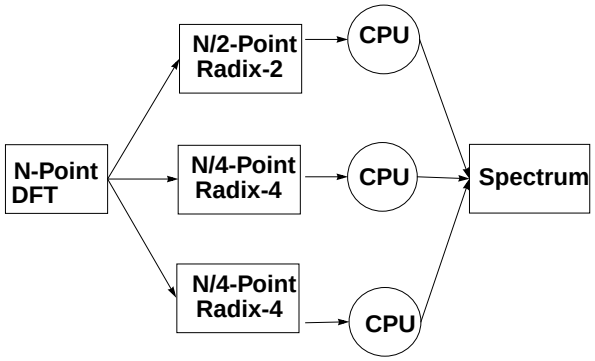


Figure 6 Parallel composition of the Split-Radix FFT and FHT.

2.3. Fast Hartley Transform

2.3.a. Sequential Form

The Discrete Hartley Transform (DHT), shown in Figure 7, takes the approach that fewer is better. Since complex arithmetic requires four real multiplications for every complex multiplication and two real additions for every complex addition, it is computationally very expensive. In addition to requiring more operations, complex numbers also require more memory since a complex number consists of two real coefficients. The DHT reduces the number of computations and memory used by simplifying the kernel of the DFT to real valued variables. [8]

$$X_H(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n \left[\cos\left(\frac{2\pi kn}{N}\right) + \sin\left(\frac{2\pi kn}{N}\right) \right]$$

Figure 7 The Discrete Hartley Transform

This new transform shares many of the properties of the DFT. Due to the similarity between the DFT and the DHT, the techniques employed by FFT to calculate the DFT can be applied to the DHT. This allows for an even greater reduction in the number of computations. Although the DHT requires fewer computations that the DFT, there is a drawback in calculating the fourier coefficients using the DHT. Additional computations are required to convert from the DHT to the DFT. However, since the relationship is linear, the computation cost is minimal. The relationship between the DHT and DFT is shown in Figure 8. [8]

$$Re(X(k)) = \frac{(X_H(k) + X_H(N-k))}{2}$$

$$Im(X(k)) = \frac{(X_H(k) - X_H(N-k))}{2}$$

Figure 8 Relations for conversion between Hartley coefficients and Fourier coefficients

2.3.b. Parallel Form

In our efforts, we chose the Split-Radix form of the FHT. The parallel form of this algorithm follows the form of the Split-Radix DFT shown in Figure 6. The difference occurs in the end when the Hartley coefficients must be converted to Fourier coefficients. This step introduces additional computations which can be performed in parallel. [9]

2.4. Quick Fourier Transform

2.4.a. Sequential Form

Whereas most FFTs use the periodic properties of the DFT to reduce complexity, the Quick Fourier Transform (QFT) uses the symmetry of the cosine and sine terms in the DFT to decrease the number of complex calculations. The QFT is constructed by breaking the data set into real cosine terms and imaginary sine terms and those into their even and odd parts. The length-(N+1) cosine terms and length-(N-1) sine terms can be recursively decomposed into two length-(N/2 +1) Discrete Cosine Transforms and two length-(N/2-l) Discrete Sine Transforms respectively as shown in Figure 9. Using this decomposition process we reduce the complexity of the DFT to a complexity of O(N $\log_2$N). Another important aspect of the QFT is that all complex operations occur at the last stage of the decomposition. This makes it well suited for real data. However, complex data can be processed by taking real transforms of the real and imaginary portions of the input separately and combining the results. [10]

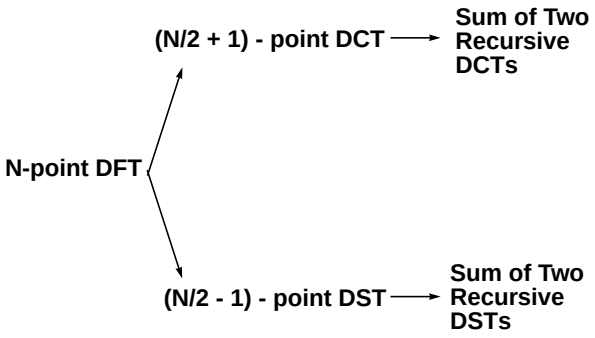


Figure 9. Flow Diagram of the Quick Fourier Transform

2.4.b. Parallel Form

The QFT is recursively decomposed into DCTs and DSTs according to the tree structure shown in Figure 10. We see from this figure that the operations at the leaves of the tree are independent of the other leaves' operations. Thus each leaf can be evaluated by a separate process and the results returned to the parent. We take the approach of allowing all processors to

traverse the QFT's tree until a point is reached where all processors can be utilized by a separate leaf. This approach does limit the number of processors to be a power of two but this is a reasonable limitation for today's hardware. Figure 10 shows how each processor is allocated to a branch of each subtree and the communication paths between each processor.

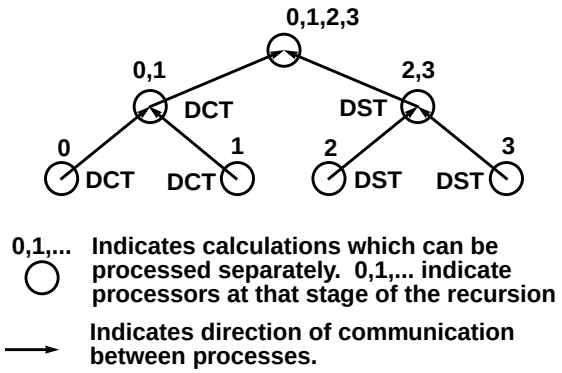


Figure 10 Parallel structure of the QFT and DITF algorithms

2.5. Decimation-in-Time-Frequency Algorithm

2.5.a. Sequential Form

The Decimation-In-Time-Frequency (DITF) algorithm uses both the Radix-2 Decimation-in-Time (DIT) and Decimation-in-Frequency (DIF) algorithms to form a new recursive algorithm. The DITF is based on the observation that the DIT algorithm has a majority of its complex operations towards the end of the computation cycle and the DIF algorithm has a majority towards the beginning. The DITF makes use of this fact by performing the DIT at the outset and then switching to a DIF to complete the transform as shown by the block diagram of Figure 11. Combining these algorithms comes at the cost of computing complex conversion factors at the point of switching. [11]

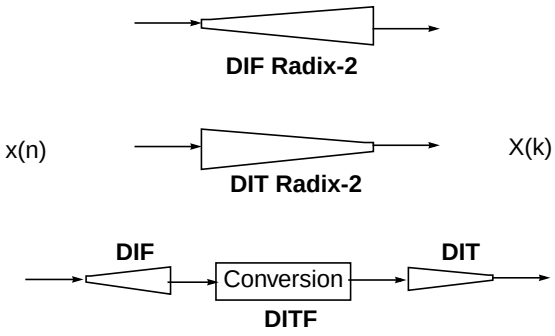


Figure 11 Block Diagram of DITF algorithm. Notice that the portions of the DIT and DIF with fewer computations are used by the DITF.

2.5.b. Parallel Form

Similar to the QFT, the DITF uses a recursive approach to the decomposition of the DFT by using both a recursive DIT and a recursive DIF. As with the QFT, the parallel structure of the DITF is described by Figure 10. Unlike the QFT, the DITF is highly communication dependent. Each leaf process must communicate at least once for each frequency point calculated. Thus, the cost of the tree structured approach for the DITF is too high. Instead the approach we take for the DITF is one of data splitting. Each processor is allocated N/p frequency data points to calculate, where p is the number of processors and assuming (N mod p) = 0.

3. Implementation

Implementation of the FFT algorithms consisted of two phases: sequential coding and parallel coding. The sequential FFT routines have been freely available in the public-domain for years. These FFT routines have been refined over a matter of months and years and each has been tweaked to the point of maximal efficiency.

The parallel implementations were developed directly from the sequential code. To do this, we first carefully analyzed each FFT routine for structural parallelism. Secondly, we analyzed the required communication costs of the parallel structure. Taking both of these into account, we designed the parallel routines to best take advantage of the algorithm's parallelism while minimizing communication costs.

3.1. Parallel Communication Tool

In order to use multiple processors, the data must first be distributed. Thus, there is a need for a method by which to transfer data between processes. The Message Passing Interface (MPI) provides a means by which to transfer data between processes conveniently. Since MPI is a communication library, it allows for inner-workings of the communication protocol to occur transparent to the calling code. MPI processes are "self-aware" in the sense that each is assigned a numeric rank at start-up that uniquely identifies it. The transfer of data using MPI is accomplished by making a call to one of the library's function. The calling code specifies the transfer by passing sending process's rank, the receiving process's rank, the data type, and the number of data elements to be passed. In addition to point-to-point communication, MPI supports collective and group communication. This allows for a reduction in the overhead in sending to a group (broadcast) and receive from a group (gather) of process. [12] [13]

Since the MPI libraries have been ported to many operating systems (OS)/achetectures, porting a parallel program to another does not require any change in the communication calls. MPI takes care of the hardware details, allowing the programmer to worry only about the software implementation. This allows for MPI code to transcend platforms.

4. Testing Procedures

Giving an objective evaluation of the algorithms requires an extensive knowledge of how each compares over a wide range of circumstances. In particular we want to collect statistics which relate directly to application constraints including factors such as memory, speed, data length, and hardware capabilities. We also desired test methods which give consistent and repeatable results.

4.1. Criteria

In choosing criteria by which to evaluate the various algorithms, consideration was given to the different constraints that would be imposed by a particular application. The criteria for benchmarking were computations speed, memory usage, number of processors available, input data size, code size, object size, and number of mathematical operations (adds, multiplications, and binary shifts). Computation speed was selected as the core criteria for comparison since the most efficient method is generally the most desirable one. However, since the amount of memory available is not static from machine to machine, memory was also included as a measure of efficiency. For many parallel applications, an increase in the number of processors available directly correlates to the speed-up. This is due to the fact that the number of processors limits the degree to which the data sequence can be decomposed. As was illustrated in theory section, decreasing the length of the data sequence for an $O(N^2)$ operation reduces the net number of operations which must be performed. The number of mathematical operations was included since it is directly related to the computation time and the hardware requirements. The additions and multiplications were broken in to floating point and integer operations due to the fact that floating point operations are much more costly in computation time than are integer operations. Input sequences of differing lengths were included to examine increase/ decrease in the overall cost of the communication.

4.2. Testing Methods

Evaluation of the above criteria involves many issues which are not readily apparent. Key amongst these is processor loading. One of our most important criteria is speed. Unfortunately this measurement cannot be completely decoupled from the loading of the hardware. To obtain consistent measurements of speed, we must eliminate the effects of fluctuations in processor loading. We accomplish this in two ways: test on unloaded processors and use an iterative testing method.

We ran our timing tests on relatively unloaded processors. These processors were loaded only by our programs and system operations. This eliminated the latency introduced when another user’s program is executing on our test machines.

An iterative approach to testing was also used to reduce the transients of processor loading. This method involved running each test for each algorithm for a large number of iterations. For instance, a compute intensive system operation may have been started in the middle of one test. This test taken alone would produce invalid and inconsistent results. On the other hand, performing this same test over a large number of iterations would average out that invalid result and produce repeatable results.

We also used a variety of methods for calculation of the other statistics. Computation speed is measured using the standard unix system utility ‘clock()’. For evaluation of memory usage, we developed a floating point class which has the feature of accumulating a count for each byte of memory allocated. This gives a

very efficient method of viewing the dynamic memory management. Counting of mathematical operations was accomplished in a manner similar to the memory counts.

5. Results

The observed computation time, shown in Figure 12 and Table 1, illustrates that execution time was of order Nlog(N) for all algorithms with respect to the data size The Radix-4 had the least computation time of all algorithms evaluated, but is limited to data sizes which are a power of four. The Radix-2 was somewhat slower than the more computationally efficient Radix-4. For the odd powers of two, the Split- Radix had the lowest computation time. Overall, the Split-Radix ranked between the Radix-2 and Radix-4. This was expected since the Split-Radix makes uses of both the Radix-2 and Radix-4. Therefore, the computation time for the Split-Radix is a weighted average of the computation time of its two sub-components. The FHT, which utilizes a Split-Radix topology, is somewhat slower than the Split-Radix FFT. This is due to additional computation time necessary to transform the Hartley coefficients to Fourier coefficients.

The ranking of the algorithms based on lowest memory usage, shown in Figure 13 and Table 2, follows the Ranking for least computation time with one exception. The DFT was the slowest, but uses the least amount of memory.

Algorithm	Speed N = 64	Speed N = 128	Speed N = 256	Speed N = 512	Speed N = 1024	Speed N = 2048	Speed N = 4096
DFT	61900	248600	996400	3994300	16009500	21280728	30576662
RADIX-2	1000	2100	4400	9400	19900	42100	90900
RADIX-4	700	---	3500	---	16000	---	72200
SRFFT	900	1800	3800	7900	16800	35400	76400
FHT	1000	2100	4600	9500	20500	43600	97800
QFT	1100	2500	5600	12300	26900	60700	129300
DITF	49400	165700	596900	2214000	6889400	7735873	11668681

Table 1: Computation Time for Each Algorithm (See Figure 12)

Algorithm	Data Type	Float Mults	Float Adds	Int Mults	Int Adds	Bin Shifts	Memory (bytes)
DFT	Real	2097152	2097152	0	1049600	0	8
	Complex	4194304	4194304	0	1049600	0	8
RADIX-2	Real	20480	30720	0	15357	1024	12308
	Complex	20480	30720	0	15357	1024	12308
RADIX-4	Real	15701	28842	336	8877	2738	4172
	Complex	15701	28842	336	8877	2738	4172
SRFFT	Real	4668	11722	494	11545	2335	12332
	Complex	10016	25488	502	12448	2937	12332
FHT	Real	9352	15006	0	4695	2123	4328
	Complex	18704	32056	0	8367	4246	16416
QFT	Real	4224	14722	8	34517	157	12288
	Complex	8448	31492	16	70058	316	24576
DITF	Real	48894	47163	0	16927	1	446464
	Complex	52996	51796	0	34878	2	462848

Table 2: Table of mathematical operations for each algorithm

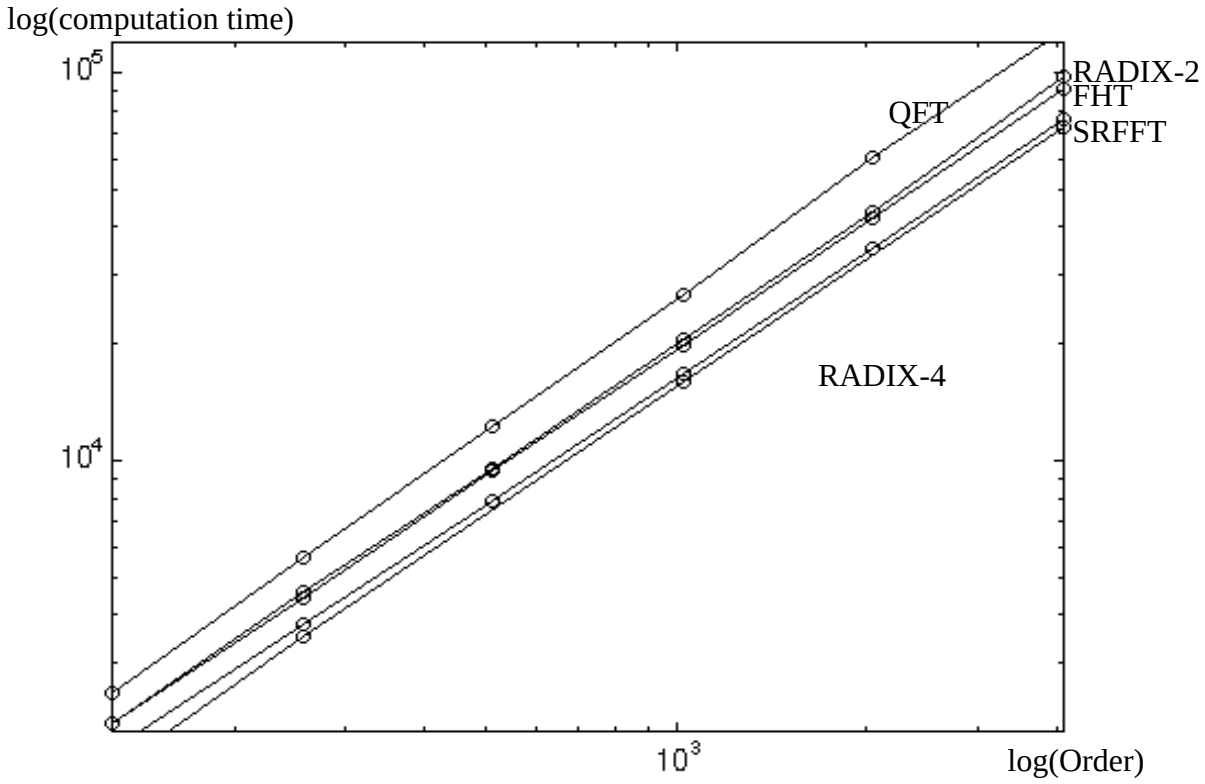


Figure 12 Log-Log Plot of Computation Time vs Order. Notice the N log(N) shape of the curves.

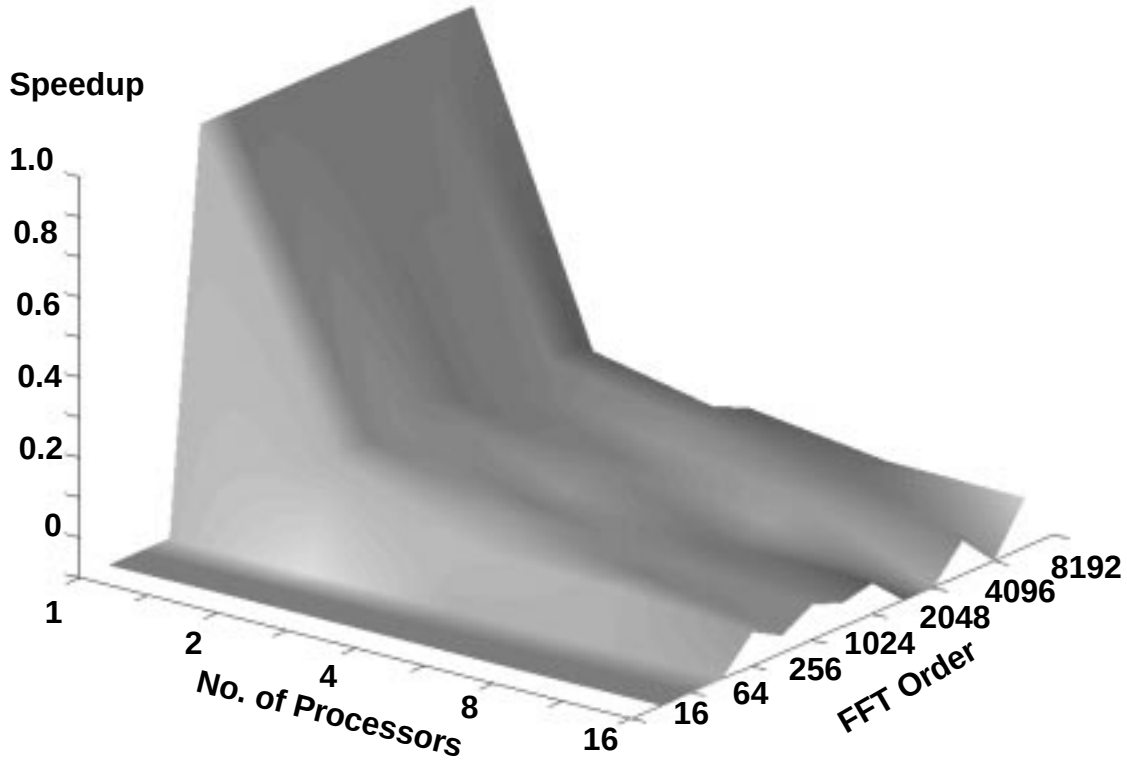
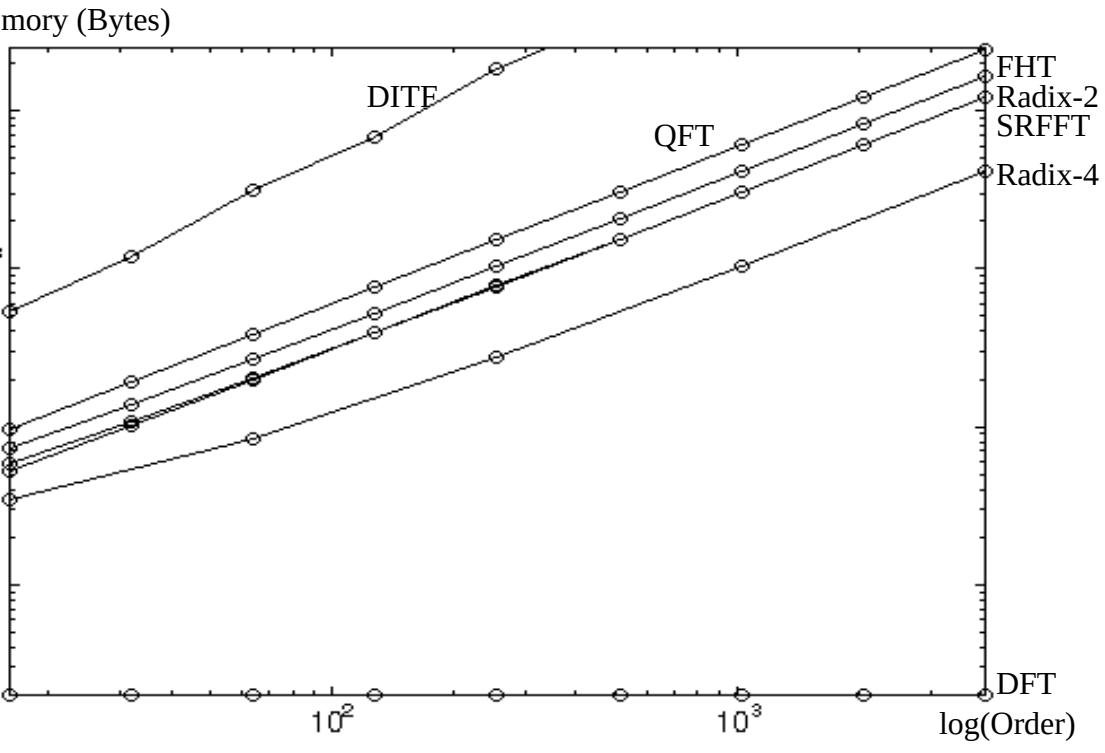


Figure 15 Typical results for the parallel algorithms. No speedup was achieved for the majority of the parallel algorithms



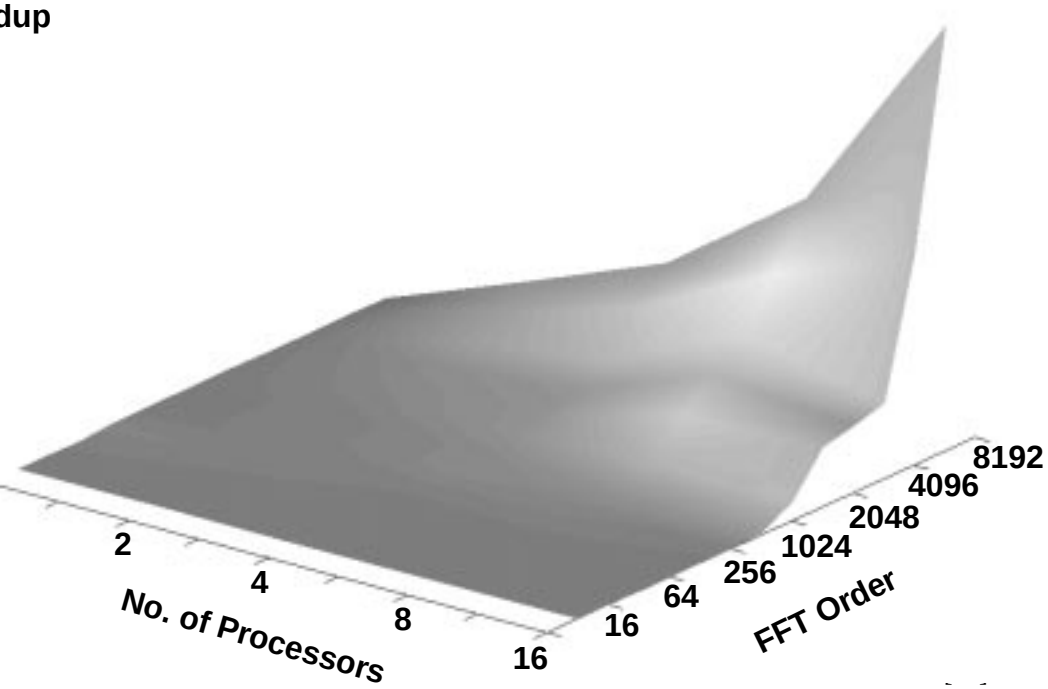
Log-Log Plot of Memory Usage vs Order for Each Algorithm.

alternative approach would be to distribute the data unevenly to allow more communication / execution overlap. That is, the master node would keep a larger portion of the data for itself. This would allow the

dup

inually process data while the other ess, and send back their portions. de is completing its computations at the processed data from other nodes n uneven data distribution approach re overlap, but still suffers from the den.

tion that eliminates the need for ta between processes is multi- reading uses several processes, all of to the same memory. Therefore, ne process transmitting its results to esult will be written to a memory processes can access. The processes o communicate with one another to ion between stages. That is, one o wait for another to complete the proceed. The limitation to this that the number of processes would number of processors on any one



4 Best case results for parallel algorithms (DFT). These results occur because data splitting was possible for the DFT.

6. Conclusions

Overall, the sequential algorithms proved to faster than their parallel counterparts. This can be accounted for by the additional time required to communicate between nodes. With the parallel algorithms, one node acts as a master distributing the data to other nodes. The master then becomes a slave which processes an equal portion of the data. The communication latency was such that the master node would complete its data processing cycle long before any of the other nodes since a send requires much less time than a receive. After completing its data processing cycle, the master node goes into a receive mode waiting to recollect the data processed by other nodes. This causes yet another delay due the imbalance of the send and receive times mentioned. The net result was sequential execution with waits inserted due to the communications latency.

In order for any parallel algorithm to be beneficial, there must be sufficient overlap between execution and communication. An even distribution of data provides for virtually no overlap. Given these results, an

7. Future Considerations

In our efforts thus far we have assembled the first unified collection of publicly available parallel FFT algorithms. The sequential code provides public-domain FFT code for a variety of algorithms under a single framework. The parallel code will give insight into the parallel coding of the various FFT algorithms.

Secondly, we have developed hard statistics for the complexity of the FFT algorithms. These statistics give the developer a clear picture of what will be required if a particular algorithm is used. Having these statistics takes the guess work out of development which was necessary to overcome the ambiguity of the big-O notation.

Perhaps, most importantly our work has laid the foundation for a class of “intelligent” programs which will automatically choose the best algorithm and execute it for a given set of user constraints. As the hardware available becomes more diverse, this type of software is becoming a necessity. This constraint driven program will also be a beneficial tool to the developer because it will give a quick way to test performance under changing design constraints.

There are various possibilities that must be explored

before the constraint driven software will become a reality. Foremost among these is to re-evaluate the parallel structure of the FFT code. Trying to cast the "tight" sequential code into a parallel form is not likely to be the best option. We should redevelop the implementation to use the parallel structure of the algorithm more efficiently.

Also, we must explore other parallel processing techniques. We have seen that the major impedance to parallel processing of FFTs is communication costs. We will explore other techniques (such as shared memory) which allow us to perform the FFTs with a reduced number of communication calls.

8. Acknowledgments

We would like to thank the following persons for their support: Dr. Joseph Picone, Dr. Tony Skjellum, and the members of each of their groups. Most importantly we thank Aravind Ganapathiraju for his leadership and interest in our efforts.

9. References

1. Oppenheim, Alan V., Ronald W. Shafer. *Discrete-Time Signal Processing.* Prentice Hall. Englewood Cliffs, New Jersey 1989. pp 587-610.

2. Bracewell, Ronald N. *The Fourier Transform and Its Applications, Second Edition.* McGraw-Hill Book Company. New York, 1978. pp356-381.

3. Proakis, John G. and Dimitris G. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications, Third Edition.* Prentice Hall, Upper Saddle River, New Jersey, 1996. pp230-256, 394-494.

4. Roberts, Richard A., Clifford T. Mullis. *Digital Signal Processing.* Addison Wesley, Reading, Massachusetts, 1987. pp 148-162.

5. Blahut, Richard E. *Fast Algorithms for Digital Signal Processing.* Addison Wesley, Reading, Massachusetts, 1985. pp 114-152, 240-280.

6. Tatyana D. Roziner, et. al., "Fast Fourier Transforms Over Finite Groups by Multiprocessor Systems," *IEEE Trans. on ASSP*, vol. 38, no. 2, February 1990. pp 226-239.

7. P. Duhamel and H.Hollomann, "Split radix FFT algorithm," *Electron. Lett.*, vol. 20, pp. 14-16, Jan. 1984.

8. R. Bracewell. *The Hartley Transform* Oxford, England: Oxford Press, 1985, chapter 4.

9. "Implementing 2-D and 3-D Discrete Hartley Transforms on a Massively Parallel SIMD Mesh Computer." *Technical Report CS-TR-95-01*, University of Central Florida, Orlando, FL.

10. H. Guo, G.A. Sitton, C.S. Burrus. "The Quick Discrete Fourier Transform." *ICASSP94 Digital Signal Processing.* vol III. Institute for Electrical and Electronics Engineers. pp. 445-447, 1994.

11. Saidi, Ali. "Decimation-In-Time-Frequency FFT Algorithm." *ICASSP94*
Digital Signal Processing. vol III. Institute for Electrical and Electronics Engineers.
pp. 453-456, 1994.

12. "Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. Technical Report Computer Science Department" *Technical Report CS-94-230*, University of Tennessee, Knoxville, TN, May 5 1994.

13. Snir, Marc. et. al. *MPI: The Complete Reference.* The MIT Press, Cambridge,Massachusetts, 1996.

14. Fox, Geoffrey C. et. al. *Solving Problems On Concurrent Processors*. Prentice Hall, Englewood Cliffs, New Jersey, 1988.

Audible Frequency Detection and Classification

David E. Gray, W. Craig McKnight, Stephen R. Wood

Audible Frequency Detection and Classification Group
Department of Electrical and Computer Engineering
Mississippi State University
216 Simrall, Hardy Rd.
Mississippi State, Mississippi 39762
{gray, wcm1, srw1}@ece.msstate.edu

1. Abstract

Current music software relies on external input from MIDI capable devices. Because traditional musical instruments are inherently analog, the interaction of musicians and computers is rare.

The purpose of this project is to develop a software package for music education utilizing an acoustical instrument interface so that players of all instruments can begin to utilize the computing power of today's world. Musicians who play tones into a microphone will see those tones analyzed in the areas of relative and absolute pitch.

2. Overview

2.1 A Brief History of Computers in Music

There has been an interest in using computers to aid in the creation of music for more than thirty years. Bell Telephone Laboratories developed a program called *Music 4* as early as the 1960's. Through various updates and revisions, this program eventually developed into what is now *CSound*. *CSound* allows the user to write programs that represent arrangements of music for different instruments that will be simulated by the computer. One of the new features of the program allows the user to input information into the computer via a MIDI (Musical Instrument Digital Interface) equipped instrument [1].

One area where the use of computers in music holds great promise is in education. *Listen,* by Imaja, is an educational software package that teaches students relative pitch, harmony, intervals between notes, and chord structure in an interactive environment. For example, the computer would give an interval to a student studying intervals. The student would then play the interval on the keyboard or other MIDI instrument and the computer would tell the student if the interval played was correct or if one of the notes was too high or too low. This package relies on MIDI codes being sent to the computer from a MIDI capable device [2].

MIDI codes are a form of communication protocol decided upon by the music manufacturing industry. The codes are transmitted serially at a 31.25 K bits/s data rate and contain information about which key was played, when the event started, when the event stopped, what voice patch to use, and other aesthetics involved with playing a note [3] [4].

In the case of a piano style keyboard being connected to a computer using MIDI, the information traveling to the computer is generated digitally, transmitted digitally, and manipulated on a digital computer. Throughout the entire signal chain from the instrument to the computer, the signal is never analog. This is an obstacle for musical instruments in general since most are inherently analog. MIDI controllers for instruments such as the guitar have been developed and in some cases the entire instruments themselves have been designed as the controllers, but an interface to accommodate all instruments without changing hardware is not in wide use.

The goal of this project is to create a program that uses a computer to recognize musical notes originating from an analog source. Users of the program are able to play notes on an instrument into a microphone. They are then informed of what note was played and if the note was flat, sharp, or in tune. Another option allows the user to learn about the intervals between a group of notes. The notes can either be played simultaneously or sequentially.

This project could serve as foundational research for the creation of a complete music education and notation

software package for instruments that are not MIDI capable.

2.2 Overall System Algorithm

Figure 1 illustrates the process by which musical notes are analyzed with the software developed in this project. A musical, analog signal is played into a microphone or other analog transducer. The analog signal is then converted to a digital signal sampled at a given rate by the recording feature of *Network Audio*.

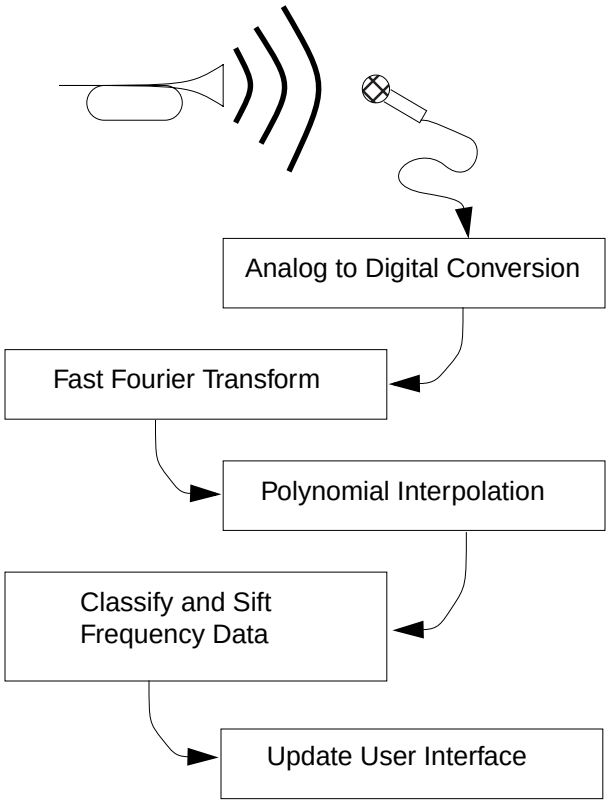


Figure 1: Block diagram of the implemented system showing the main steps in analyzing the musical signal.

After sampling, a Fast Fourier Transform is used to convert the time domain signal into the frequency domain. Once the signal is in the frequency domain, the peaks in the frequency spectrum are found and passed to a polynomial interpolation routine to increase the accuracy of the program. A list of possible notes is then built. The possible notes in the note list are classified as notes if they pass given criteria corresponding to magnitude and overtone patterns that imply musical signals.

Once the note list has been created, each note is given a note name corresponding to its frequency. Depending on the mode of operation, notes are found to be in tune or not, or the intervals between notes played either sequentially or simultaneously are determined.

3. Musical Signals

3.1 Musical Notes

When describing or measuring musical notes, only a single frequency is usually mentioned. For example, the musical note A4 is generally accepted to be the frequency corresponding to 440 Hz. However, when a single musical note is played on an instrument, more than one frequency is produced. The additional frequencies occur at integer multiples of the lowest frequency, which is the frequency used to name the note (See Figure 2).

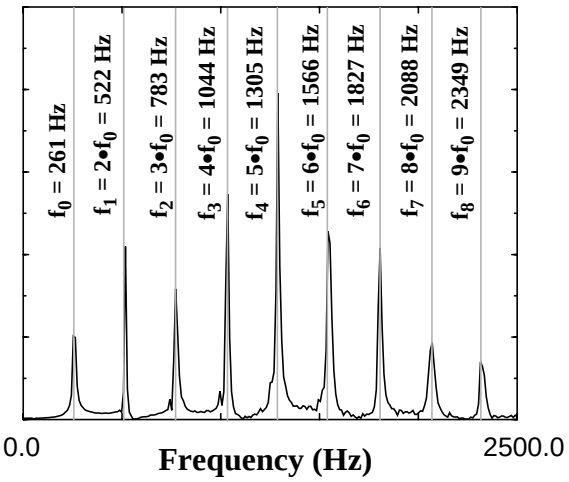


Figure 2: Frequency Spectrum of a C4 played on a Trumpet. Notice the integer relationship of each overtone frequency to the fundamental frequency at 261 Hz.

The lowest frequency is called the fundamental frequency and the frequencies occurring at integer multiples of the fundamental are called harmonics or overtones. Given the frequency, f_f, of the fundamental, the frequency of the n^{th} overtone can be calculated as follows:

$$f_n = (n+1) f_f \quad (1)$$

The relative amplitude of the different harmonics produced varies from instrument to instrument and this amplitude pattern defines the *timbre* of an instrument. Timbre is the quality, or color, of a sound based on a harmonic series.

Because musical notes produce a particular pattern of frequencies and because the energy of noise is distributed randomly throughout the frequency spectrum, it is possible to distinguish musical notes from noise [5]. Characteristic spectra for white noise, random noise and a musical note are shown in Figure 3. The distinguishing characteristic of the overtone pattern was used to identify and classify notes in this project.

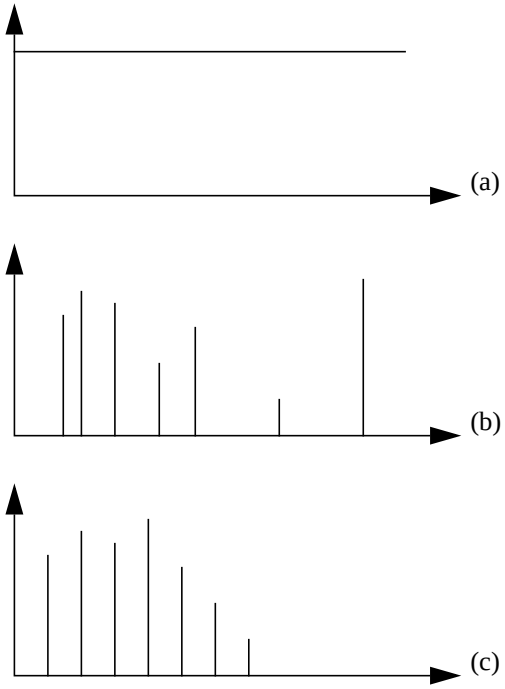


Figure 3: Frequency spectra of (a) white noise (b) noise with irregular energy concentration (c) musical note.

3.2 Even-Tempered Tuning

Perhaps the most important interval in music is the octave. The octave serves to define the musical scale. It is defined as the interval between two notes where the higher note is exactly twice the frequency of the lower note. In Western music, there are 12 notes, called semitones, which divide up the range between a note and its octave counter part. Patterns of these notes played sequentially make up a musical scale. The distance between two adjacent notes is called a semitone and the distance between two notes with one in between is called a tone or whole tone.

The major scale is defined by the following pattern of tones and semitones: T, T, S, T, T, T, S. If the eight frequencies representing the notes in the scale are picked to be musically pleasing, the notes are tuned to *ideal temperament*. The fourth column of Table 1 shows the ratios of the frequency of notes in the major scale in relation to the root or lowest note of the scale [6] [7].

Table 1: Comparison of Equal and Ideal Temperament for C Major Scale

Note Name	Equal Temperament		Ideal Temperament
	Ratio	Frequency	Ratio
C	1.0000	261.63	1.0000
C#	1.0595	277.18	
D	1.1225	293.66	1.1250
D#	1.1892	311.13	
E	1.2599	329.63	1.2500
F	1.3348	349.23	$1.333\overline{3}$
F#	1.4142	369.99	
G	1.4983	391.99	1.5000
G#	1.5874	415.31	
A	1.6818	440.00	$1.666\overline{6}$
A#	1.7818	466.16	
B	1.8877	493.88	1.8750
C	2.0000	523.25	2.0000

A problem is encountered when an instrument is tuned using ideal temperament ratios. If the instrument is tuned to the key of C and all the notes in between C and its corresponding octave obey the ratios of ideal temperament, the instrument will perform well for music written in the key of C. However, if a piece of music is to be played in a different key using the ideal temperament tuning for the key of C, the ratios between the notes for the new key do not correspond to ideal temperament for the key of C. For example, the difference in the ratio values for the notes C and D is

1.125 - 1.000 = 0.125. If a piece of music was in the key of G, the difference in the ratios between the notes of G and A is $1.66\overline{6}$ - 1.500 = $0.16\overline{6}$, which is clearly different. Therefore, using ideal temperament tuning, an instrument would have to be retuned anytime it played in a different key. This is quite undesirable and makes playing a piece of music that changes keys impossible.

To remedy the tuning problem, a tuning method called *even temperament* was derived. Even temperament tuning divides the distance between semitones such that each semitone is a factor K larger than the previous semitone. That is,

$$f_1 = K \bullet f_0 \qquad (2)$$

where $K = \sqrt[12]{2} \approx 1.05946$.

While no note is exactly in tune using this system, each note is very close to being in tune. This allows musicians to play in various keys without having to retune their instruments. The resulting note frequency combinations are given in Table 2. The even tempered scale was used for classifying notes in the project because it is the accepted tuning scheme for Western music.

The range of notes listed in Table 2 contains the full range of a piano tuned using even tempered tuning. The area of interest for this project was limited to the notes between 80 Hz, a flat E2, and 2534 Hz, a sharp D#7. This restriction was made in the interest of frequency resolution which is discussed in Section 4. The musical notes not included (shaded in light gray) are on the extremes of the frequency range of piano and thus, used less often. Only frequencies within the unshaded range (the range of interest for this project) were classified as notes.

Table 2: Frequency Values for Even-Tempered Piano Scale

	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0	--	--	--	--	--	--	--	--	--	27.5	29.13524	30.86771
1	32.7032	34.64783	36.7081	38.89087	41.20344	43.65353	46.2493	48.99943	51.91309	55.0	58.27047	61.73541
2	65.40639	69.29566	73.41619	77.78175	82.40689	87.30706	92.49861	97.99886	103.8262	110.0	116.5409	123.4708
3	130.8128	138.5913	146.8324	155.5635	164.8138	174.6141	184.9972	195.9977	207.6523	220.0	233.0819	246.9417
4	261.6256	277.1826	293.6648	311.127	329.6276	349.2282	369.9944	391.9954	415.3047	440.0	466.1638	493.8833
5	523.2511	554.3653	587.3295	622.254	659.2551	698.4565	739.9888	783.9909	830.6094	880.0	932.3275	987.7666
6	1046.502	1108.731	1174.659	1244.508	1318.51	1396.913	1479.978	1567.982	1661.219	1760.0	1864.655	1975.533
7	2093.005	2217.461	2349.318	2489.016	2637.02	2793.826	2959.955	3135.963	3322.438	3520.0	3729.31	3951.066
8	4186.009	--	--	--	--	--	--	--	--	--	--	--

4. Internal Algorithms

4.1 Sample Frequency

The first consideration of the project was to decide on the frequency at which to sample the analog input signal. If an analog signal containing a maximum frequency, f_{max}, is to be recovered without aliasing, it must be sampled at a rate greater than its Nyquist rate, f_N [8]. The Nyquist rate for a signal is defined by the following equation:

$$f_N = 2 \bullet f_{max} \tag{3}$$

As stated in Section 3.2, the highest note that will be recognized is a D#7. The frequency for this note is 2489.02 Hz. For best performance, the first overtone of this note should also be detectable. The first overtone of a D#7 can be calculated from Equation (1) to be 4978.04 Hz. Therefore, the Nyquist rate is determined to be 9956.08 Hz. This is the minimum sample frequency.

Once the analog input signal is sampled, it will be transformed to the frequency domain via a Fast Fourier Transform (FFT). The frequency resolution for an FFT is defined as

$$\Delta f = \frac{f_s}{N} \tag{4}$$

where f_s is the sample frequency and N is the number of points of the FFT [9].

The peaks in the frequency spectrum of a signal that is transformed with an FFT appear to occur at integer multiples of the resolution. Therefore, the desired frequency resolution must also be considered along with the Nyquist rate when determining the sample frequency.

There are many algorithms that implement an FFT [10]. The Decimation in Time and Frequency (DITF) algorithm was implemented first [11]. This algorithm proved to be extremely slow when performing a 1024 point transform. For the project to execute in real time, a faster algorithm was necessary. The algorithm known as the Quick Fourier Transform (QFT) was tested to see if it had better speed performance [12]. For our application, and for 1024 points, the QFT was more than 10 times faster than the DITF. This speed enhancement was sufficient for the project to perform in real time; therefore, the QFT was implemented in the project.

The sample frequency was first set at 20 kHz. This choice would have allowed more overtones of the D#7 to be detected. However, from Equation (4), it would only offer a frequency resolution of 19.53 Hz. For the low end of the frequency range defined in Table 2, this resolution would allow up to three semitones to have a peak in the frequency spectrum at the same frequency value. For example, the frequency spectrum of an F#2, G2, and G#2 would all indicate a maximum at 97.66 Hz. This would not be acceptable.

The first two notes in the frequency detection range differ by approximately 5 Hz. Therefore, a resolution of at least 5 Hz would be needed in order for each note to have a maximum at a unique frequency. From Equation (4), this would require a sample frequency of 5120 Hz. However, the Nyquist rate, as discussed previously, sets a lower limit for the sample frequency of 9956 Hz. Therefore, the sample frequency was chosen to be 10 kHz.

The frequency resolution obtained by using a sample frequency of 10 kHz and a 1024 point FFT is given by Equation (4) to be 9.765625 Hz. This resolution will not provide a unique maximum point in the frequency spectrum for all the notes in the desired detectable range. Polynomial interpolation schemes provide a solution to this deficiency.

4.2 Polynomial Interpolation

The data points $(x_1,y_1),(x_2,y_2),...,(x_n,y_n)$ for a function, f(x), are the necessary information needed to compute an interpolating Polynomial, p(x), of order n-1. This polynomial will be unique and will agree with f(x) for all data points. A polynomial interpolation scheme is said to have Lagrange form if it can be written as

$$P_L(x) = \sum_{k=1}^{n} y_k L_k(x) \tag{5}$$

L_k represents a family of polynomials of degree n-1 which satisfy

$$L_k(x_j) = \begin{Bmatrix} 0, k \neq j \\ 1, k = j \end{Bmatrix} \quad \begin{matrix} j = 1, \ldots, n \\ k = 1, \ldots, n \end{matrix} \tag{6}$$

This definition insures that $p_L(x_j) = f(x_j) = x_j$ for j = 1,...,n.

For some arbitrary value of x,

$$L_k(x) = \prod_{\substack{j=1 \\ j \neq k}}^{n} \frac{x - x_j}{x_k - x_j} \tag{7}$$

Evaluation of $p_L(x)$ will require n+1 evaluations of L(x). The number of multiplies required to compute $p_L(x)$ is $n^2 + 2n - 1$ and the number of additions is $n^2 + n - 1$ [13].

If the data points $(x_1,y_1,y_1'),(x_2,y_2,y_2'),\ldots,(x_n,y_n,y_n')$ are known for a function, f(x), then it is possible to compute $p_H(x)$ of order 2n-1. This interpolating polynomial has Hermite form [13]. The Hermite polynomial satisfies the conditions $p_H(x_i) = f(x_i)$ as well as $p_H'(x_i) = f'(x_i)$ for all i = 1,2,...n.

The Hermite polynomial is represented by the following equation:

$$p_H(x) = \sum_{k=1}^{n} H_k(x) f(x_k) + \sum_{k=1}^{n} \hat{H}_k(x) f'(x_k) \tag{8}$$

where

$$H_k(x_j) = \begin{Bmatrix} 0, k \neq j \\ 1, k = j \end{Bmatrix} \quad \begin{matrix} j = 1, \ldots, n \\ k = 1, \ldots, n \end{matrix} \tag{9}$$

$$H_k'(x_j) = 0 \quad \begin{matrix} j = 1, \ldots, n \\ k = 1, \ldots, n \end{matrix} \tag{10}$$

$$\hat{H}_k'(x_j) = \begin{Bmatrix} 0, k \neq j \\ 1, k = j \end{Bmatrix} \quad \begin{matrix} j = 1, \ldots, n \\ k = 1, \ldots, n \end{matrix} \tag{11}$$

$$\hat{H}_k(x_j) = 0 \quad \begin{matrix} j = 1, \ldots, n \\ k = 1, \ldots, n \end{matrix} \tag{12}$$

For arbitrary values of x, the value of L_k defined in Equation (7) is used to define the value of H_k and $\hat{H}_k$ in the following manner:

$$H_k(x) = [1 - 2(x - x_k)L_k'(x_k)]L_k^2(x) \tag{13}$$

$$\hat{H}_k(x) = (x - x_k)L_k^2(x) \tag{14}$$

A 5-point Lagrange interpolation and a 3-point Hermite interpolation were tested to determine the increased accuracy that could be obtained by interpolating the data from the frequency spectrum. The goal of the Hermite scheme was to interpolate using three consecutive points (x_0,y_0), (x_1,y_1), and (x_2,y_2), where (x_1,y_1) is the maximum point in the frequency spectrum. However, the Hermite scheme requires knowledge of the derivative of f(x), as seen by Equation (8). This data is not directly available in the frequency spectrum. To produce a Hermite polynomial, the derivative at the points of interest was estimated using the centered difference formula for numerical differentiation given below [13]:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} \tag{15}$$

where h is the distance between consecutive data points and is the resolution of the FFT in this context.

For the Lagrange interpolation scheme, five consecutive data points, (x_0,y_0), (x_1,y_1), (x_2,y_2), (x_3,y_3), and (x_4,y_4) were used where (x_2,y_2) is the maximum point in the frequency spectrum.

Since the resolution of the FFT is approximately 10 Hz, the true frequency of the input signal will be within 5 Hz of the maximum in the frequency spectrum. Both of the interpolating polynomials were evaluated at 0.1 Hz increments across a 10 Hz range centered about the maximum data point. From this data, the maximum of

the interpolation polynomial is determined with an accuracy of 0.1 Hz.

The two interpolation routines were tested with computer generated sine waves of known frequency. The results are listed in Table 3

Table 3: Comparison among FFT, Lagrange, and Hermite

Actual Peak	FFT	5-point Lagrange	3-point Hermite
82.0	78.1	80.1	79.3
154.0	156.3	155.9	156.0
155.0	156.3	156.1	156.2
156.0	156.3	156.2	156.3
157.0	156.3	156.3	156.3
158.0	156.3	156.4	156.3
438.0	439.5	439.3	439.4
440.0	439.5	439.5	439.5
442.0	439.5	440.0	439.7
582.0	585.9	584.0	584.8
584.0	585.9	585.7	585.8
586.0	585.9	585.9	585.9
588.0	585.9	586.2	586.1
590.0	585.9	588.1	587.3
650.0	654.3	651.5	652.5
652.0	654.3	653.9	654.1
654.0	654.3	654.3	654.3

This data indicates that the 3-point Hermite polynomial never performs better than the 5-point Lagrange polynomial. The Hermite is also computationally more expensive, since it computes L_k as well as L_k' for $k = 1,...,n$ and must estimate 3 derivatives. Therefore, a 5-point Lagrange interpolating polynomial is used to determine the maximum point in the frequency spectrum more accurately.

The Lagrange interpolating algorithm was also tested on a range of frequencies given in Table 4. This table indicates the increased accuracy gained by interpolating as well as the error that still occurs in interpolated values.

Also, Table 4 shows that the FFT produces data points at 976.56 Hz and 986.33 Hz. Interpolation is most beneficial when the frequency to be detected is approximately half way between consecutive FFT data points. These results are illustrated below in Figure 4.

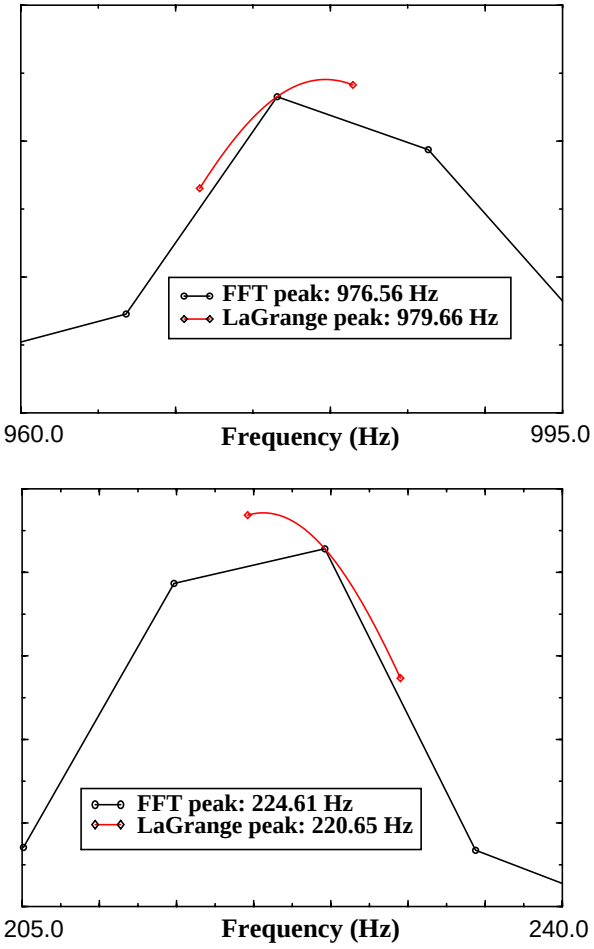


Figure 4: Two graphs comparing the accuracy of frequency data with and without Lagrange interpolation. (top) 981 Hz peak generated with a function generator. (bottom) A3, which is 220 Hz, played on a bass guitar.

A higher number of points could have been used in the calculation of the FFT in order to produce a better resolution as opposed to interpolation. However, even if this option is computationally feasible in real time, it is not as desirable as interpolation. If 2048 points were used, the resolution would be 4.88 Hz. A note that is one to two Hz sharp or flat would be likely to have the same maximum in the frequency spectrum as if it were played in tune. By interpolating the peak, however, it is

Table 4: Uninterpolated Frequency Values and Interpolated Frequency Values Using 5- point Lagrange Interpolation for Δf=9.766 Hz

Generated Frequency (Hz)	Uninterpolated Frequency (Hz)	Interpolated Frequency (Hz)	Uninterpolated Difference (Hz)	Interpolated Difference (Hz)	Error Ratio Magnitude
975.0	976.56	976.36	+1.56	+1.36	1.15
976.0	976.56	976.56	+0.56	+0.56	1.00
977.0	976.56	976.56	-0.44	-0.44	1.00
978.0	976.56	976.66	-1.44	-1.34	1.08
979.0	976.56	977.06	-2.44	-1.94	1.26
980.0	976.56	977.76	-3.44	-2.24	1.54
981.0	976.56	979.66	-4.44	-1.34	3.31
982.0	986.33	983.53	+4.33	+1.53	2.83
983.0	986.33	985.23	+3.33	+2.23	1.49
984.0	986.33	985.93	+2.33	+1.93	1.21
985.0	986.33	986.23	+1.33	+1.23	1.08
986.0	986.33	986.33	+0.33	+0.33	1.00
987.0	986.33	986.33	-0.67	-0.67	1.00

likely that a difference of one to two Hz in the signal will be discernible.

4.3 Sifting Algorithm

A sifting algorithm was implemented to identify and classify the notes that were played. The course of the algorithm is as follows. The frequency spectrum derived from the FFT is scanned to find peaks above a certain amplitude threshold. Once a peak is found, its frequency is compared with other peaks already placed in the note list to determine if it is an integer multiple of these peaks. If so the peak is classified as a harmonic or overtone of that note and placed in its array of overtones. The algorithm then progresses to see if the peak might be an overtone of any of the other possible notes in the note list. If, once the note list has been traversed, the peak has not been classified as an overtone, it is classified as a possible note.

Once all of the peaks out of the frequency spectrum have been classified as possible notes or overtones, the note list is scanned to eliminate any peaks which might be noise. To accomplish this, the amplitude of the peak and the number of peaks placed in its overtone array are tested.

To check the amplitude of the possible notes, the magnitude of the highest peak is used to normalize the magnitude of all of the peaks. If a peak does not have a magnitude of 0.15 once normalized and the peak doesn't have at least two overtones, the peak is classified as noise and is eliminated from the note list. If a peak in the note list satisfies both of these qualifications, then it is considered to be a note and is passed back to the calling function to be named.

4.4 Tuning Overtones

The data in Table 3 and Table 4 indicate that the interpolation algorithm detects a frequency with an absolute error less than 2.25 Hz. This error is quite significant in the lower end of the desired detection range. At higher frequencies, however, the error will be tolerable. Perhaps an example will illustrate this point.

A musician plays a note at 110 Hz, which is an A2 perfectly in tune. The minimum value that the

interpolation routine would indicate for this tone is 107.75 Hz. By searching a database, it is known that a frequency of 107.75 Hz should be 110 Hz to be in tune. The tuning quality of this note is calculated by

$$Q_f = \frac{f_c - f_T}{f_T - \left(\frac{f_T}{\sqrt[12]{2}}\right)} \qquad (16)$$

$$Q_s = \frac{f_c - f_T}{(\sqrt[12]{2})(f_T) - f_T} \qquad (17)$$

where Q_f is the quality of a note if it is flat, Q_s is the quality of a note if it is sharp, f_c is the frequency of the note played by the instrument, and f_T is the frequency of the note in even-tempered tuning that most closely matches f_c.

For the example under discussion, Equation (16) yields a quality of -0.36. The magnitude of the quality must be greater than or equal to 0.05 to be classified as in tune. Therefore, the tone played is characterized as very flat, when it was actually perfectly in tune.

The musician plays another note at 880 Hz, which is an A5 perfectly in tune. The minimum interpolated value for this tone is 877.75 Hz. Using Equation (16), the quality of this tone is computed as -0.05 which would be characterized as in tune.

With this in mind, and remembering that tones have a predictable harmonic structure, tuning will be more accurate by tuning a particular overtone. Continuing the example, the musician again plays a note at 110 Hz. The seventh overtone of this tone is 880 Hz. Searching the database for the fundamental tone, returns that the tone should be an A2. Then, by tuning the seventh overtone, the fundamental can be classified as in tune.

This algorithm is solid as long as the overtone which is tuned is also a note in the database. Since the 2.25 Hz maximum error associated with the interpolation algorithm becomes less significant for each successive overtone, tuning to the highest detected overtone will provide the best result. However, only the 2^n-1 overtones of a tone correspond directly to notes in the database of notes found in even-tempered tuning.

This problem was investigated further by looking at the overtones for every tone in the desired detection range. All the overtones that occur within the bounds of the maximum frequency in the database were examined. The frequency of each overtone was determined and then the database was searched to see which tone most closely matched the overtone in frequency value. Then Equation (16) or Equation (17) was used to see how the overtone related to the tone found in the database.

Continuing the example for an A2, The eighth overtone occurs at 990 Hz. In even tempered tuning, no tone occurs at 990.00 Hz. If the database is searched for this value, a B5 will be returned which has frequency value 987.77 Hz. The quality of this note would be calculated as +0.0380. This quality represents an inherent error, ε, that is introduced when tuning the eighth overtone of an A2. If the 990 Hz frequency had been a fundamental tone, then the quality would accurately indicate that the tone was slightly sharp.

The data collected from this process was compiled and analyzed to see a remarkable trend. The error associated with a particular overtone is independent of the fundamental tone. The eighth overtone of any in tune note has a quality of 0.0380. Table 5 shows the error corresponding to a given overtone.

A fundamental tone, f_f, will have an overtone frequency, f_o, that will be used for tuning f_f. f_o will be compared to a frequency, f_c, from the database. If the inherent error, ε, is greater than 0, it represents the fraction of the difference between f_c and the next note in even tempered tuning that must be added to f_c in order to get f_o. This is seen by Equation (18).

$$f_0 = f_c + \varepsilon[(\sqrt[12]{2})(f_c) - f_c] \qquad (18)$$

If ε is less than 0, it represents the fraction of the difference between f_c and the previous note in even tempered tuning that must be subtracted from f_c in order to get f_o. This is seen by Equation (19).

$$f_0 = f_c - |\varepsilon|\left(f_c - \frac{f_c}{\sqrt[12]{2}}\right) \qquad (19)$$

Equations (20) and (21) are derived by solving Equations (18) and (19) respectively for f_c.

Table 5: Deviation of Overtones from Even-Tempered Scale

Overtone Number	Deviation from Even-Tempered Scale (%)
0	0.00
1	0.1021
2	1.9750
3	0.0511
4	-13.1604
5	1.9750
6	-29.9504
7	0.0511
8	3.8636
9	-13.1805
10	-46.5735
11	1.9580

$$f_c = \frac{f_0}{1 + |\varepsilon|(\sqrt[12]{2} - 1)} \qquad (20)$$

$$f_c = \frac{f_0}{1 + |\varepsilon|(1/(\sqrt[12]{2}) - 1)} \qquad (21)$$

If the overtone number, n, associated with the frequency f_o that is to be used in determining the quality of the fundamental tone, f_f, is known, then these equations allow f_o to be modified to a corrected value f_c by using the ε associated with the given n. Then, the new frequency, f_c, can be used for determining the tuning quality. When this frequency is compared to a frequency in the database, f_T, there will be no inherent error, ε, introduced. *Q*, the quality given by Equation (16) or (17), will accurately represent the quality of f_f.

4.5 Interval Detection

Musicians are often interested in the groups of notes or chords that they are playing. An option to identify the musical interval between two notes was implemented. When called for the first time, the interval detection code creates an interval list of all the notes currently in the note list. The interval between each of the notes in the interval list is then calculated.

Each successive time that the interval detection code is called, the note list and the interval list are compared to see if any new notes have been played. New notes are added to the current interval list and all intervals are recalculated. A function outside the interval detection code clears the interval list when the user desires to investigate a different combination of notes.

5. Project Database

To test the program during the process of development, a database was created. It consists of sine waves generated by a digital function generator and musical notes digitally recorded using a microphone and a sampling program called *Network Audio*. Files in the database range from containing single frequency data without instrument overtones to multiple frequency data with instrument overtones.

File names for the database were standardized to the following format: **<source>_<# of principle frequencies>_<tuning><note>**.raw. **<source>** is a three-character abbreviation for the instrument. **<# of principle frequencies>** is a numeral denoting the number of principle frequencies (i.e., the number of notes played) in the sample. **<tuning>** is a single-character denoting whether the sample is sharp, flat, or in tune: +, -, ~, respectively. **<note>** is a three-character denotation of the note name in the sample. For example, an A220 were represented as A3~. The file name for a sample collected from a trumpet playing a sharp A440 would be: tru_1_+A4~.raw. If the sample were a C261, but were in tune: tru_1_~C4~.raw. For a sample from an instrument capable of playing multiple tones at once, such as a guitar playing sharp A440 and in tune C#554 above it would be: acg_2_+A4~_~C5+.raw.

The archive was constructed of data to be used in three phases of the project. The first phase contained single tones to test the tuner function of the software. For

Phase I of the project, three instruments were used in the data archive: a guitar, a bass guitar and a trumpet. Each instrument was recorded in three sets of cases. For each case, a note was played in tune, then that note was played out of tune flat, and then it was played out of tune sharp. Phase I of the archive contains 36 samples in the range of interest.

In Phase II, two instruments, the guitar and bass guitar, were each recorded playing each of the twelve intervals less than an octave. Phase II of the archive contains 24 samples for interval testing.

Phase III of the archive contains sequential notes designed to test if the software is effective in detecting note changes. Again, three instruments were used, the guitar, bass guitar, and trumpet. For each file, a sequence of three notes was recorded. Each instrument played three sequences of intervals. The Phase III archive contains 27 samples.

Phase IV is an archive of chord data recorded from the guitar and the bass guitar. Each instrument played a major, minor and diminished triad based on three tones. The Phase IV archive contains 18 samples.

All of the data in the database was sampled at 20 KHz. Once the decision was made to run the program at a sampling rate of 10 KHz, the database files were down sampled to 10 KHz. This was accomplished through the use of a program called *sox*.

6. Results

Of 36 Phase I files, we named the correct note 88% of the time. Of these, 62.2% were tuned correctly. The note name in the acoustic guitar files were identified correctly 100% of the time and the tuning was correct 90.3% of these times. The note name in the bass guitar files were identified correctly 80.7% of the time and the tuning was correct 52% of these time. The note name for the trumpet files was identified correctly 87% of the time and the tuning was correct 83% of these times. This indicates that our algorithm is effective at frequencies greater than 130 Hz, but is less effective at lower frequencies.

Of 24 Phase II files, 33% of the notes and intervals were identified correctly. For any given two notes, the interval was always classified correctly. Of the 12 acoustic guitar files, 67% were correct. The four files that did not yield correct results could be special cases of overtone interaction or could be attributed to inaccurate naming of the files. None of the files for the bass guitar yielded correct results. The lower tone of each file was recorded an octave too low, placing it out of the range of interest. This accounts for the poor performance for these files.

Of the nine files in Phase III, all three notes in six of the files were identified correctly without any problems. One bass guitar file contains notes that were outside of the range of interest as defined by Table 2. Another bass guitar file causes an error in the software. Listening to this file indicates that there was a large amount of distortion present and this could account for the software error.

The files in Phase IV were not evaluated critically. However, the results for the Phase IV data are expected to be comparable to the results of the Phase II data. The addition of notes played simultaneously is not expected to adversely affect the performance of the software.

7. Future Research

It might prove useful to investigate frequency domain representations of our signals other than the FFT. Some alternatives might be spectral estimation and Prony's method [14] [15]. Estimating the spectrum with these techniques could eliminate the need for polynomial interpolation.

Reducing computation time for one spectrum will be key to integrating the software into a real-time music notation program. Faster FFT algorithms might yield this. Also the spectral estimation techniques could do this.

8. References

1. B. Vercoe, *MIT Media Lab Csound Manual*. http://www.leeds.ac.uk/music/Man/Csound/pref.
2. *Listen's Sound and MIDI Features*. http://www.imaja.com/Listen.html.
3. H. Chamberlin, *Musical Applications of Microprocessors*. Hayden Books., Hasbrouck Heights, NJ, USA, 1985.
4. S. Lehman, *Harmony Central: MIDI Tools and Resources*. http://www.harmony-central.com/MIDI/.
5. J. C. Brown, "Musical fundamental frequency tracking using a pattern recognition method." *The Journal of the Acoustical Society of America Vol 92, No. 3*. American Institute of Physics, Woodbury, NY, USA, 1994.
6. A. H. Benade, *Horns, Strings, and Harmony.* Dover, NY, 1960.
7. C. Taylor, *Exploring Music: The Science and Technology of Tones and Tunes*. Institute of Physics Publishing, Bristol, England, 1994.
8. J. G. Proakis, D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms and Applications Third Edition*. Prentice Hall, Upper Saddle River, NJ, USA, 1996.
9. R. E. Ziemer, W. H. Tranter, and D. R. Fannin, *Signals and Systems: Continuous and Discrete, 3rd Edition*. MacMillan, New York, NY, 1993.
10. W. W. Smith and J. M. Smith, *Handbook of Real Time Fast Fourier Transforms*. Institute of Electrical and Electronic Engineers, Inc., New York, NY, 1995.
11. Saidi, Ali. "Decimation-In-Time-Frequency FFT Algorithm." *ICASSP94 Digital Signal Processing. vol III*. Institute for Electrical and Electronics Engineers. New York, NY, 1994.
12. H. Guo, G.A. Sitton, C.S. Burrus. "The Quick Discrete Fourier Transform." *ICASSP94 Digital Signal Processing. vol III*. Institute for Electrical and Electronics Engineers, New York, NY, 1994.
13. N. S. Asaithambi, *Numerical Analysis Theory and Practice*, Saunders College, Fort Worth, TX, USA, 1995.
14. M. B. Priestley, *Spectral Analysis and Time Series, Vol. 1*. Academic Press, London, 1981.
15. S. L. Marple, Jr., *Digital Spectral Analysis With Applications*, Prentice Hall, Englewood Cliffs, NJ, USA, 1987.

9. Appendix--Raw Data from Analysis of Project Database

Table 6: Results of Analyzing Data from Phase I of Project Database

Filename (*.raw)	Number of FFT's per File	Number Correctly Identified	Number of Correctly Identified
acg_1_+C4~	18	18	18
acg_1_+D4~	16	16	16
acg_1_+E5~	7	7	7
acg_1_-C4~	18	18	18
acg_1_-D4~	16	16	16
acg_1_-E5~	8	8	8
acg_1_~C4~	20	20	7
acg_1_~D4~	24	24	22
acg_1_~E5~	7	7	6
bag_1_+A2~	28	27	1
bag_1_+A3~	22	22	22
bag_1_+C3~	29	29	26
bag_1_+C4~	28	28	28
bag_1_+E2~	31	19	19
bag_1_-A2~	30	30	30
bag_1_-A3~	22	22	22
bag_1_-C3~	27	0	0
bag_1_-C4~	20	20	0
bag_1_-E2~	28	0	0
bag_1_~A2~	31	30	3
bag_1_~A3~	26	26	18
bag_1_~C3"	32	30	0
bag_1_~C4~	21	21	0
bag_1_~E2~	26	16	0

Table 6: Results of Analyzing Data from Phase I of Project Database

Filename (*.raw)	Number of FFT's per File	Number Correctly Identified	Number of Correctly Identified
tru_1_+B4-	18	18	4
tru_1_+C4~	19	18	18
tru_1_+C5~	17	17	16
tru_1_+G4~	19	19	19
tru_1_-B4-	18	18	18
tru_1_-C4~	14	5	5
tru_1_-C5~	16	5	5
tru_1_-G4~	19	16	16
tru_1_~B4-	22	22	7
tru_1_~C4~	19	19	0
tru_1_~C5~	18	18	0
tru_1_~G4~	19	19	0

Table 7: Results of Analyzing Data from Phase II of Project Database

Filename (*.raw)	Low Note	High Note	Interval
acg_2_~C4~_~A4+	**C4**	**A4#**	**m7**
	A4#	**A4**	**P0**
acg_2_~C4~_~A4~	E3	B3	P5
	E3	G4#	M3
	B3	B3	P0
	B3	G4#	M6
	G4#	G4#	P0
acg_2_~C4~_~B4~	B3	A4#	M7
	A4#	A4#	P0

* these files cause a software error

Table 7: Results of Analyzing Data from Phase II of Project Database

Filename (*.raw)	Low Note	High Note	Interval
acg_2_~C4~_~C4+	C4	C5#	m2
	C5#	C5#	P0
acg_2_~C4~_~C5~	B3	F6#	P5
	F6#	F6#	P0
acg_2_~C4~_~D4+	**C4**	**D4#**	**m3**
	D4#	**D4#**	**P0**
acg_2_~C4~_~D4~	**C4**	**D4**	**M2**
	D4	**D4**	**P0**
acg_2_~C4~_~E4~	**C4**	**E4**	**M3**
	E4	**E4**	**P0**
acg_2_~C4~_~F4+	**C4**	**F4#**	**TT**
	F4#	**F4#**	**P0**
acg_2_~C4~_~F4~	**C4**	**F4**	**P4**
	F4	**F4**	**P0**
acg_2_~C4~_~G4+	**C4**	**G4#**	**m6**
	G4#	**G4#**	**P0**
acg_2_~C4~_~G4~	**C4**	**G4**	**P5**
	G4	**G4**	**P0**
bag_2_~C3~_~A3+	B2	G3	m6
	B2	A3#	M7
	G3	G3	P0
	G3	A3#	m3
	A3#	**A3#**	**P0**

* these files cause a software error

Table 7: Results of Analyzing Data from Phase II of Project Database

Filename (*.raw)	Low Note	High Note	Interval
bag_2_~C3~_~A3~	A2	B2	M2
	A2	G3	m7
	A2	A2	P0
	B2	B2	P0
	B2	G3	m6
	B2	A3	m7
	G3	G3	P0
	G3	A3	M2
	A3	**A3**	**P0**
bag_2_~C3~_~B3~	B2	G3	m6
	B2	B2	P0
	G3	G3	P0
	G3	B3	M3
	B3	**B3**	**P0**
bag_2_~C3~_~C3+	*		
bag_2_~C3~_~C4~	*		
bag_2_~C3~_~D3+	C3	G3	P5
	C3	A3#	m7
	G3	G3	P0
	G3	A3#	m7
	A3#	A3#	P0
bag_2_~C3~_~D3~	C3	A3	M6
	C3	D4	M2
	A3	A3	P0
	A3	D4	P4
	D4	D4	P0

* these files cause a software error

Table 7: Results of Analyzing Data from Phase II of Project Database

Filename (*.raw)	Low Note	High Note	Interval
bag_2_~C3~_~E3~	F2	E3	M7
	F2	B3	TT
	E3	**E3**	**P0**
	E3	B3	P5
	B3	B3	P0
bag_2_~C3~_~F3+	F2#	C3	TT
	C3	C3	P0
bag_2_~C3~_~F3~	F2	C3	P5
	F2	G3	M2
	C3	C3	P0
	C3	G3	P5
	G3	G3	P0
bag_2_~C3~_~G3+	G2#	C3	M3
	G2#	G2#	P0
	C3	C3	P0
	C3	G3#	m6
	G3#	G3#	P0
bag_2_~C3~_~G3~	G2	C3	P4
	C3	C3	P0

* these files cause a software error

Algorithm to Determine the Scenic Beauty of Images

Nirmala Kalidindi, Liang Zheng, Yaqin Hong

Digital Image Group
Department of Electrical and Computer Engineering
Mississippi State University
Box 9571
Mississippi State, MS 39762
434 Simrall, Hardy Rd.

{kaldindi@isip, zl3@ra, yh4@ra]

Abstract

The United States Forestry Service (USFS) wishes to determine the scenic quality of the images to preserve recreation and aesthetic resources in forest management. They want to determine a predefined pattern to cut the trees so as to still retain the scenic beauty even after cutting the forest by timber loggers. The scenic beauty will be determined on a scale from “0” to “1”. We have in our database 679 unique PPM images of different vegetations taken during all the seasons of the year. Each of the image is of 4.7 Mb. We have the subjective ratings available for all the images. These ratings are taken by showing each of the images to different groups of people and then converting them to a standardized scores by using Scenic Beauty Rating. We attempt to develop a systematic approach to determine the scenic quality and correlate them to the subjective ratings available. Some of the parameters to be considered are color, size of the trees etc. The effect of the color can be determined by doing histogram analysis of the image. The effect of the size of the trees can be studied by doing edge detection and computing the number of vertical lines in the image. Forest scenery that is undisturbed and having variety of natural features is preferred while scenery which obstructs the view of the forest with lot of foliage and bushes is not considered as scenic.and images which doesn’t obstruct the view are preferred as compared to the ones with more foliage and bushes. The evaluation is performed by running the program through the images in the database.

1. Introduction

We attempted to develop an algorithm to determine the scenic quality of the image on a scale from “0” to “1”. The importance of forest recreation and landscape scenic quality is being recognized and thus efforts are made to preserve the scenic quality of the image.The statistical models suggest that the density and sawtimber-sized trees and the proportion and visual penetration are positively associated with scenic beauty while foliage, twig, small stem screening and the density of small-diameters tree are negatively associated with scenic beauty. Histogram analysis and edge detection methods were used to analyses the parameters such as color and the size of the trees in the image. The results from the subjective ratings showed that scenic beauty increases with the level of the hardwood retention and the summer, fall and

spring views were preferred over those taken during winter. Effort was made to correlate the output of the algorithm with the subjective ratings. We have the database with 679 unique images. The database has the scenic beauty rating for each of the image. Scenic beauty rating is a scaling procedure used to correlate the ratings from different groups of sessions. Baseline slides are used as reference for all the rating sessions.

2.PPM Images

The images given by the forestry department were in Kodak PhotoCD(PCD) format. The PCD format is a proprietary format and it could not be viewed with the imagetools available hence it was required to convert it into more familiar formats such as gif or ppm.We chose PPM format for this purpose. The PCD images were converted into PPM format through public domain software "hpcdtoppm" available in the net. This software tool is used to convert pcd files to ppm files. hpcdtoppm stands for "Hadmuts pcd to ppm" The software is available at the URL "*http://www.boutell.com/lsm/lsmbyid.cgi/000746*". This software also needs some netpbm utilities which are also available as public domain utilities. The netpbm utilities can be downloaded from the ftp site at "*ftp://ftp.cs.ubc.ca/ftp/archive/netpbm/*".Various resolutions of the ppm files can be obtained. Basically Base/16, Base/4, Base,4Base and 16 Base are available. We have taken the 4 Base option for our requirement.Any of the resolution can be obtained by adjusting the options while executing the "hpcdtoppm" software. The resolution of each of these options is given below.

1. Base/16, size 128 x 192 pixels
2. Base/4, size 256 x 384 pixels
3. Base, size 512 x 768 pixels
4. 4 Base, size 1024 x 1536 pixels
5. 16 Base, size 2048 x 3072 pixels

The format of the PPM images is described below:

- A "magic number" for identifying the file type. A PPM file magic number is the character string *P6*. It should be the first line of the PPM file.
- Whitespace characters such as blanks, TABs, carriage returns (CR), line feeds (LF) etc.
- The image width in number of pixels, formatted as ASCII characters in decimal.
- Whitespace characters.
- The height again in number of pixels, formatted as ASCII characters in decimal.
- Whitespace characters.
- The maximum color value which each of the colors in the pixels can have. This value is again in ASCII decimal. The maximum value which it can have is 255.
- Whitespace characters.
- Width x height pixels, each three ASCII decimal values having value between 0 and the specified maximum color value which it can take, starting at the top-left corner pixmap, proceeding in normal English reading order. The three values for each pixel represent red, green, and blue, respectively. A value of 0 means that the color is off, and the maximum value means that the color is at saturation level.
- Comments are also allowed in the

ppm file and they are indicated by the character "#". Any line starting with the character "#" to the next end-of-line are ignored.

- There is also a restriction that the line should not be longer than 70 characters.

Various resolutions of the ppm files can be obtained. Basically Base/16, Base/4, Base, 4Base and 16 Base are available. We have taken the 4 Base option for our requirement. The resolution of each of these options is given below.

1. Base/16, size 128 x 192 pixels
2. Base/4, size 256 x 384 pixels
3. Base, size 512 x 768 pixels
4. 4 Base, size 1024 x 1536 pixels
5. 16 Base, size 2048 x 3072 pixels

Any of the resolution can be obtained by adjusting the options while executing the hpcdtoppm software.

3. Histogram Analysis

Color is one of the noticeable features of a forest environment. It is affected by the temporal rhythm of seasons. Color variation by season is one of the most notable changes in forest vegetation. Summer, Fall and Spring views are judged as significantly more scenic than winter views. The preference is related to seasonal color patterns.As human preferences vary with color change, change of season is an important factor in determining the quality of an image.

Our approach was to extract the mean of each of the color in the image. Color has a major effect in determining the scenic quality of the image. The variation of seasons changes the colors in the image. It was observed from the subjective ratings that summer and spring are preferred over winter. This is based on the fact that people prefer green and blue color as compared to red and yellow which comes from the inclination of the people towards natural colors.

Each pixel in PPM is represented by three bytes one byte for each of the color of red. green and blue. Algorithm was developed to compute the mean of each of the three fundamental colors in the image and to correlate the mean of the colors to the scenic beauty of the image. For constructing the histogram the image is scanned in a single pass and a running count of the number of pixels found at each intensity value is kept. Some of the factors of the image like the naturalness of the image can be determined from the color as color is an aid in distinguishing between what is "natural" and what is "built in". In particular, a natural setting's continuous gradation in color is often very different from the sharper contrasts that are found in the built-in environment. Graph is drawn for the variation of the number of pixels for each value of the color from "0" to the maximum value of color in the image.

4. Edge Detection Overview

Edge detection is an important part in image analysis. Edges characterize object boundaries and are therefore useful for segmentation and identification of objects in scenes. Edge is defined as the boundary between two regions with relatively distinct gray-level discontinuity and the abrupt transition between two regions can be determined on the basis of gray-level discontinuity. The magnitude of the first derivative is used to detect the presence of an edge and the direction of the edge is determined by the sign of the derivative. Thus the two important properties for establishing similarity of edge pixels are the strength and the response of the gradient operator used to produce the edge pixel and the direction of the gradient.

Our approach was to extract the number of long trees in the image. We believed that computing the number of vertical lines in the image would

indicate the presence of the long trees and also the approximate length of the tree. We used the algorithm of canny-edge detector to compute edge detection.

The input to the canny edge detector is a PGM image hence we converted the PPM image to PGM image. The header of the PGM image is similar to the PPM file except for the magic number which is "P5" and there are width x height pixels each of 8 bytes representing the gray value. Edge detection was done to properly detect the edges and then thresholding them through a thresholding tracker to obtain both the horizontal and vertical lines. Algorithm was written to calculate the number of vertical lines after edge detection. By this way both the smaller as well as longer vertical lines are obtained. Hence to get the number of longer lines a threshold was considered to eliminate the shorter lines which correspond to foliage or small bushes in the image.

4.1 Conversion of PPM image to PGM image

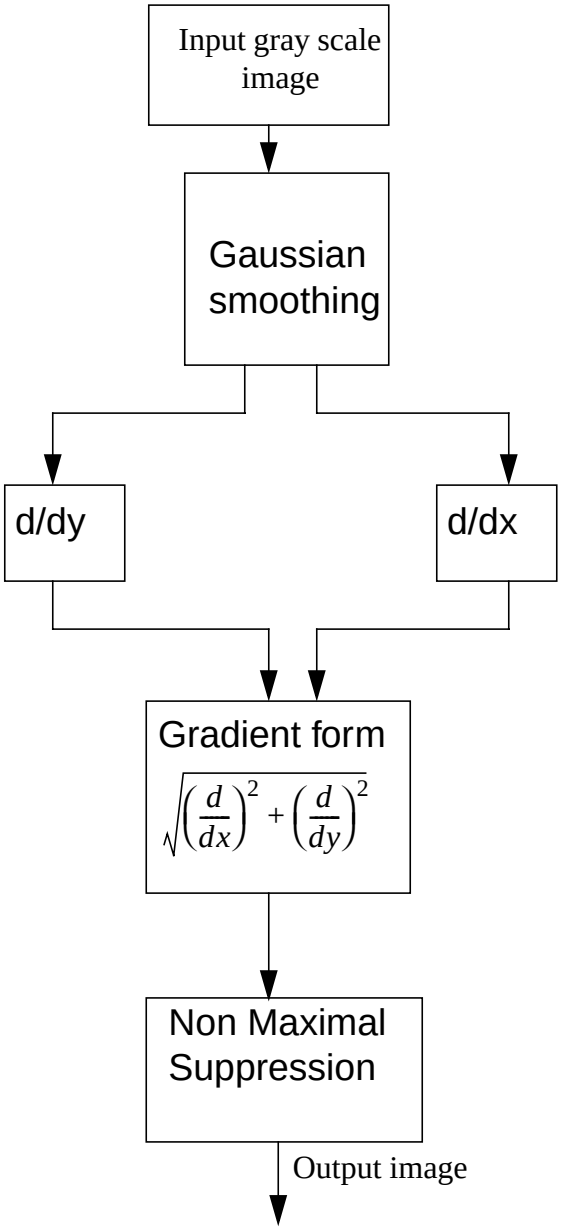
The first step of edge detection was to convert the PPM color image to the gray scale PGM image. This was done by applying the following matrix to each of the pixel of the PPM image.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (22)$$

The "Y" value in the above matrix represents the "luminance" or brightness of each pixel. The "RGB" represents the red, green and blue intensity of each pixel for the PPM image. Each pixel in the PPM file is read converted it into a gray scale value using the above relation and was written into the PGM file.

4.2 Doing edge detection

The canny-edge detector was used to perform the edge detection on the gray scale image. It first applies the masks in the horizontal and the vertical direction to retrieve the horizontal and vertical edges. These edges are then connected together to obtain the horizontal and vertical lines. A threshold value is used both for the magnitude and the orientation to detect an edge. The image after the edge detection contains only two color levels. The block diagram of the edge detection is shown below. Two masks which can



be used for the detection of edges in the horizontal and vertical direction are shown

below. The mask matrix as well as the image matrix ar given

$$\begin{bmatrix} I1 & I2 & I3 \\ I4 & I5 & I6 \\ I7 & I8 & I9 \end{bmatrix} \qquad \begin{bmatrix} W1 & W2 & W3 \\ W4 & W5 & W6 \\ W7 & W8 & W9 \end{bmatrix}$$

Where I is the image matrix and W is the mask matrix. The responses in the x and y direction can be obtained by applying the respective matrix. The mask matrix for the response in the x direction is given by the matrix [1] and the mask matrix for the response in the y direction is given by the matrix [2]. Gx and Gy, the gradients in the respective directions are calculated by the formulas shown below and the resultant gradient can be computed by the relation $\sqrt{G_x^{\,2} + G_y^{\,2}}$. The mask for calculating G_x is given by

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad [1]$$

and the mask for calculating G_y is given by

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

The G_x and G_y are given by the equations:

$$G_x = (W_7 + 2W_8 + W_9) + (W_1 + 2W_2 + W_3)$$

$$G_y = (W_3 + 2W_6 + W_9) + (W_1 + 2W_4 + W_7)$$

The angle gradient is given by the relation $\alpha(x, y) = tan^{-1}\left(\frac{G_x}{G_y}\right)$. The masks are moved through the entire image and the gradient for magnitude and orientation are obtained. We experimented the algorithm with various threshold values and it has been found to give a good edge detection at a threshold value of 120 for the orientation.

5. Vertical lines

The output of the canny edge detection algorithm is a combination of vertical as well as horizontal lines smoothened representing line segments. Algorithm was developed to compute the number of vertical lines. The total number of the vertical lines and the length of each of the line was computed. As the length of the lines is directly proportional to the length of the trees in the image they are useful in determining the number of tall trees and the short bushes.

Apparently tall trees have a positive effect on the scenic quality whereas the small bushes have negative effect. In calculating the length of the vertical lines care was taken of the deviation of the tree from the vertical by considering certain amount of deviation in the horizontal direction also. Also there might be a discontinuity in the length of the tree due to the presence of some obstacle before it. Care was taken for this also by giving a tolerance limit of about 5 pixels in the vertical direction.

6. Computing the scenic beauty

We had to obtain a relation for the scenic beauty from the analysis of the histogram and the edge detection. This has to be done so as to correlate the scenic beauty with the actual mean value of the rating available from the forestry department. The mean of each of the color in the image was obtained from the histogram analysis and the number of vertical lines in the image was obtained from the edge detection.

The mean of each of the color has been normalized inorder to get an exact percentage of the color in each of the image. The percentage of the long lines as well as the percentage of the short lines in relation to the total number of lines were computed. The facts that green color and tall trees has a positive impact on the scenic beauty and the red color and the short bushes has a negative impact on the scenic beauty is used for computing the weights of different parameters to be used for determining the scenic beauty.

7. Evaluation

The relation which we have developed by observing the dependency of the various parameters on the actual scenic rating was used for evaluating the scenic beauty on some other images in the database.We have taken about 20 images from the database and tried to adjust the weights of various parameters. After deriving a relation we implemented the relation on another 30 images. Though there was deviation from the actual scenic value they seem to be varying with the same proportion for all the images. We evaluated the program on plots 1 and 3 of block 1. We provide the results for all the images in plot 3 and some images in plot 1.

8. Database

An important aspect is the availability of extensive well organized database of the forestry images. These images are the pictures taken of the Ouachita forest in the Winona range. We have in our database 680 unique images. These are the images taken of four blocks in the range with each block having five plots. Each of the image is taken during all the seasons of the year and also with different angles so as to see the effect of the season.

There are also subjective scenic beauty ratings available for each of the images. These subjective ratings were obtained by showing the images to people from different walks of life and asking them to rate the images on a scale from 1 to 10. “1” indicating less scenic and “10” indicating the most scenic image. These scores are then converted into standard ratings using Scenic Beauty Rating technique.

Rating scales offer an efficient and widely used means of recording judgements about many kinds of images. Scenic Beauty Estimation(SBE) is one of the scaling procedure used. The main reason for the scaling procedures are that people will use the rating scale differently from one to another in the process of recording their perceptions of the images presented for assessment.

Scaling procedures are effective for adjusting some of these differences. All the information regarding the images such as the block number, plot number, SBE rating, angle and the time at which it is taken is included in the ppm file as comments. There are 4 blocks and 5 plots per block.The distribution of the plots in the blocks and the number of images is given in the following tables.

Directory b01 (block 1)	
Sub Directory	Number of Images
p001	32
p002	32
poo3	40
p004	39
p020	32

Directory b02 (block 2)	
Sub Directory	Number of Images
p005	40
p006	32
p007	32
p008	32
p019	32

Directory b03 (block 3)	
Sub Directory	Number of Images
p009	32
p010	32
p011	32
p012	32
p018	40

Directory b04 (block 4)	
Sub Directory	Number of Images
p013	32
p014	32
p015	40
p016	32
p017	32

There are 32 images of each plot. This relation comes as there are 4 angles per plot and the picture is taken 4 times in a year, we have 16 images of the same plot per year. As in the database we have the images for 2 years there are 32 images for each plot. Some of the plots have 40 images. These additional images are the baseline slides which are used as reference.

9. Results

This section contains charts and graphs of the various programs run on the images. For convenience sake we are including only the images from plots 3 and 1 in the website. The mean values of the color provided in the following chart are given in the same order as the images in the corresponding plots. We have taken the readings for 32 images from the plot 1 and the first ten images in the plot 3. Since the image names are long we have not included them here.We will just indicate them by numbers.

Number of the image	Red mean	Green mean	Blue mean
1	55	62	30
2	83	85	69
3	45	65	36
4	85	97	75
5	33	49	16
6	37	52	24
7	52	57	29
8	80	84	70
9	37	54	31
10	81	91	63
11	27	30	15
12	78	91	69
13	52	62	23
14	32	40	15
15	64	77	36
16	66	72	54
17	46	52	27
18	55	61	59
19	38	53	17
20	29	43	16
21	47	56	20
22	43	52	22
23	59	69	30
24	56	62	51
25	75	82	40

Number of the image	Red mean	Green mean	Blue mean
26	69	73	59
27	70	84	49
28	94	105	75
29	64	73	33
30	64	72	60
31	67	83	49
32	96	108	78
33	58	65	32
34	64	68	56
35	57	72	44
36	97	108	84
37	42	52	21
38	43	50	22
39	71	79	43
40	58	62	51
41	66	72	36
42	85	79	63
42	46	62	36
43	87	99	66
44	45	57	18
45	41	53	19
46	71	79	33
47	87	83	66
48	89	84	69
49	57	64	33

Number of the image	percent of long lines	percent of short lines
1	2.51	70.34
2	0.94	80.56
3	3.45	68.45
4	2.47	76.10
5	1.19	90.41
6	3.24	73.89
7	5.42	64.88
8	2.04	79.84
9	3.72	67.64
10	1.56	76.14
11	3.98	66.77
12	3.47	69.21
13	.0.88	82.04
14	2.12	74.21
15	2.95	68.72
16	3.08	72.24
17	3.75	70.34
18	0.84	81.12
19	0.52	83.30
20	2.10	74.34
21	2.70	78.88
22	2.21	72.02
23	2.51	71.13
24	4.99	68.42
25	4.30	66.81

Number of the image	percent of long lines	percent of short lines
26	2.78	78.39
27	4.43	67.57
28	2.17	71.71
29	2.50	74.30
30	3.56	69.98
31	4.36	65.62
32	2.29	73.66
33	2.95	70.72
34	1.41	77.89
35	3.31	68.18
36	2.67	73.08
37	1.97	77.13
38	2.45	72.03
39	2.46	72.57
40	2.03	77.37
41	2.74	72.93
42	2.34	74.79
43	3.05	70.30
44	1.10	77.98
45	1.42	77.13
46	2.57	69.58
47	2.69	68.00
48	2.28	75.70
49	3.24	71.04
50	1.96	76.23

Number of the image	Derived SBE	Actual SBE(mean)
1	0.42	4.68
2	0.33	4.64
3	0.50	6.59
4	0.37	4.27
5	0.56	7.25
6	0.51	7.58
7	0.40	4.15
8	0.33	3.94
9	0.49	6.35
10	0.38	7.21
11	0.41	4.58
12	0.33	3.76
13	0.46	6.41
14	0.48	6.00
15	0.45	5.19
16	0.36	4.81
17	0.41	3.22
18	0.33	2.63
19	0.54	6.17
20	0.55	6.73
21	0.46	6.95
22	0.46	6.38
23	0.44	5.49
24	0.35	6.02
25	0.41	4.47

Number of the image	Derived SBE	Actual SBE(mean)
26	0.34	4.15
27	0.43	4.75
28	0.38	4.13
29	0.43	5.72
30	0.36	5.00
31	0.44	5.00
32	0.38	4.41
33	0.42	5.25
34	0.34	4.15
35	0.44	5.74
36	0.36	5.24
37	0.47	6.65
38	0.44	6.61
39	0.40	5.85
40	0.34	6.04
41	0.47	4.37
42	0.35	4.47
43	0.54	4.80
44	0.46	4.65
45	0.57	6.09
46	0.57	5.94
47	0.50	4.76
48	0.36	3.60
49	0.50	5.16
50	0.48	5.05

The plots for the first five of the images in the table are given below. The first plot is the histogram plot which shows the number of pixels present in the image of each value of the color from zero to the maximum value of color in the image. The other plot shows the number of lines for each length of the line. The length of the line is on the X-axis and the number of lines with such length is on the Y-axis.

10. Future Work

10.1 Evaluating Scenic Beauty

The evaluation of scenic beauty has been done by only observing the various parameters computed and figuring out a relation between the actual scenic beauty and these parameters. The parameters are the mean value of the colors and the percentage of the vertical lines in the image. A better and efficient method would be to use the neural network as decision box. The parameters should be given as the input and the neural network should be trained with a fair amount of database to output the actual scenic beauty. It can then be used to test on the remaining images.

Also some other parameters like texture of the ground and frequency characteristics can be used to determine the scenic beauty. The effect of these parameters can be determined using various image analysis methods and their effect on the scenic beauty should be evaluated.The determination of more parameters helps in the effective determination of the scenic beauty and a better correlation to the actual scenic value.

11. REFERENCES

[16] R.J. Ray, D.J. Cengel, W.F. Watson, J. D. Clark, D.G. Hodges, and V. A. Rudis, “A Benefit-Cost Comparison of Providing Scenic Beauty in the Ouachita National Forest,” *Proceedings of the*

17th Annual Meeting of the Council on Forest Engineering, pp. 39-51, Corvallis Oregon, U.S., June 1994.

[17] T. A. Herrick and V. A. Rudis, "Visitor Preference for Forest Scenery in the Ouachita National Forest," *Proceedings of the Symposium on Ecosystem Management Research in the Ouachita Mountains: Pretreatment Conditions and Preliminary Findings*, pp. 212-222, Hot Springs, Arkansas, U.S., October 1993.

[18] J. H. Gramann and V. A. Rudis, "Effects of Hardwood Retention, Season of the Year, and Landform on the perceived Scenic Beauty of Forest Plots in the Ouachita Mountains," *Proceedings of the Symposium on Ecosystem Management Research in the Ouachita Mountains: Pretreatment Conditions and Preliminary Findings*, pp. 223-228, Hot Springs, Arkansas, U.S., October 1993.

[19] V. A. Rudis, J. H. Gramann, and T. A. Herrick, "Esthetics Evaluation", *Proceedings of the Symposium on Ecosystem Management Research in the Ouachita Mountains: Pretreatment Conditions and Preliminary Findings*; 1993 October 26-27; pp 202-211

[20] J. H. Gramann, W. Yhang, "The Effect of Forest Color on the Perceived Scenic Beauty of Recently Treated Pine-Oak Plots on the Ouachita National Forest, Arkansas", Final Report Prepared for Forest Inventory and Analysis unit, USDA Forest service.

[21] V. A. Rudis, "Sampling and Modeling Visual Component Dynamics of forested Areas", State of the Art Methodology of Forest Inventory; A Symposium Proceedings; 1989 July 30 - August 5; pp. 84-85

[22] T. C. Brown, T. C. Daniel, "Scaling of Ratings; Concepts and Methods", Research Paper RM-293, pp. 6-18.

[23] R. J. Schalkoff, "*Digital Image Processing and Computer Vision*", John Willey & Sons, Inc., New York, USA, 1989.

[24] R. C. Gonzalez, P. Wintz, "*Digital Image Processing, Second Edition*", Addison-Wesley Publishing Company, Massachusetts, USA, 1987.

[25] R. Chellappa, A. A. Sawchuk, "*Digital Image Processing and Analysis*, Vol 2: Digital Image Analysis", IEEE Computer Society Press, New York, USA, 1985.

[26] K. R. Castleman, "Digital Image Processing", Prentice hall, Inc., Englewood cliff, NJ, USA, 1979.

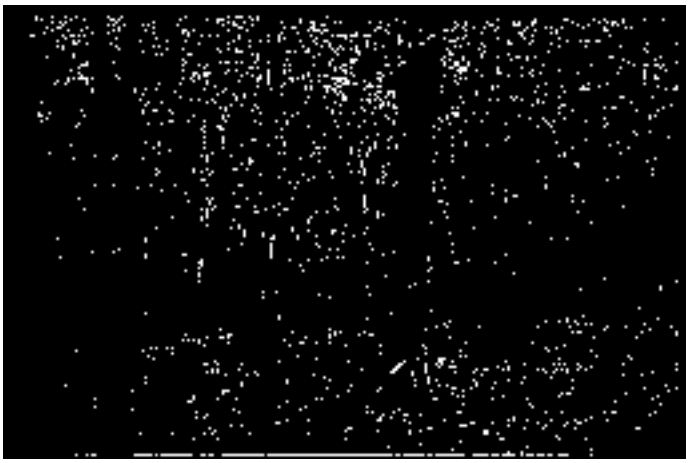
[27] W. K. Pratt, "Digital Image Processing", John Willey & Sons, Inc., New York, USA, 1978.

[28] A. K. Jain, "Digital Image processing", Prentice hall, Inc., Englewood cliff, NJ, USA, 1986.

Appendix A: Images Tested:

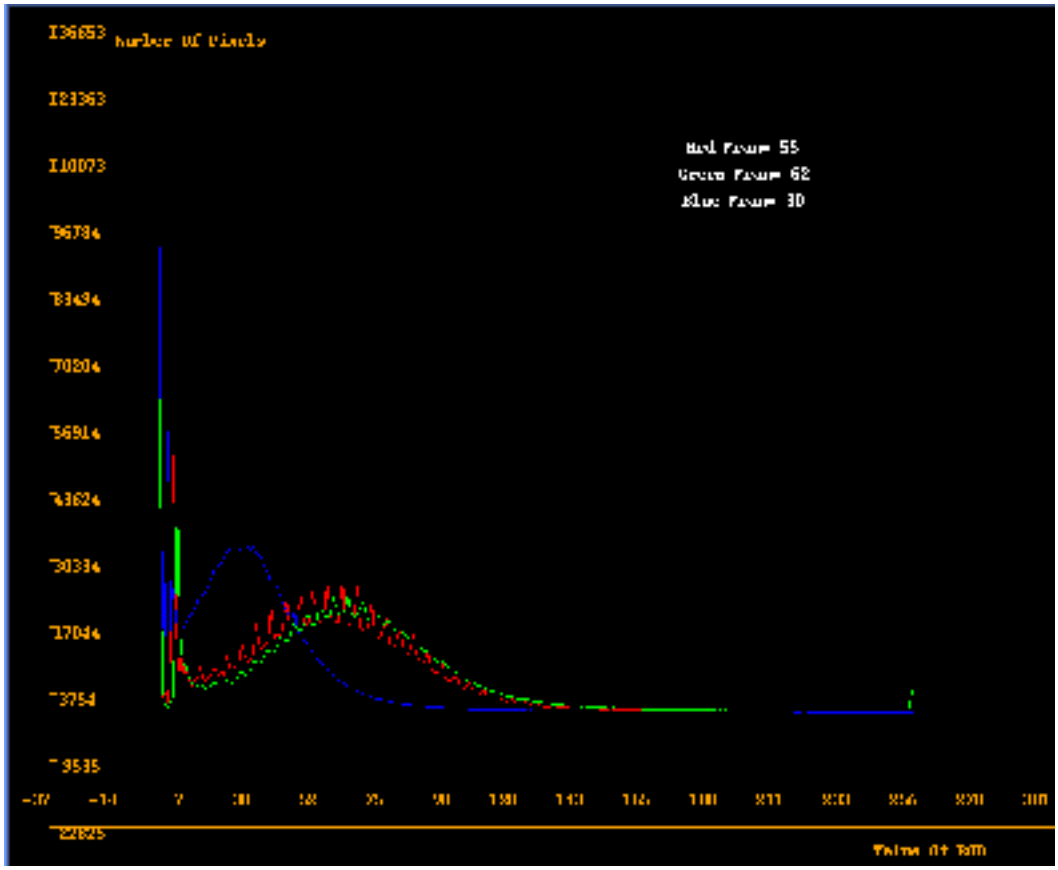
Red mean = 55
Green mean = 62
Blue mean = 30
% long lines = 2.51
% short lines = 70.34
SBE = 0.42

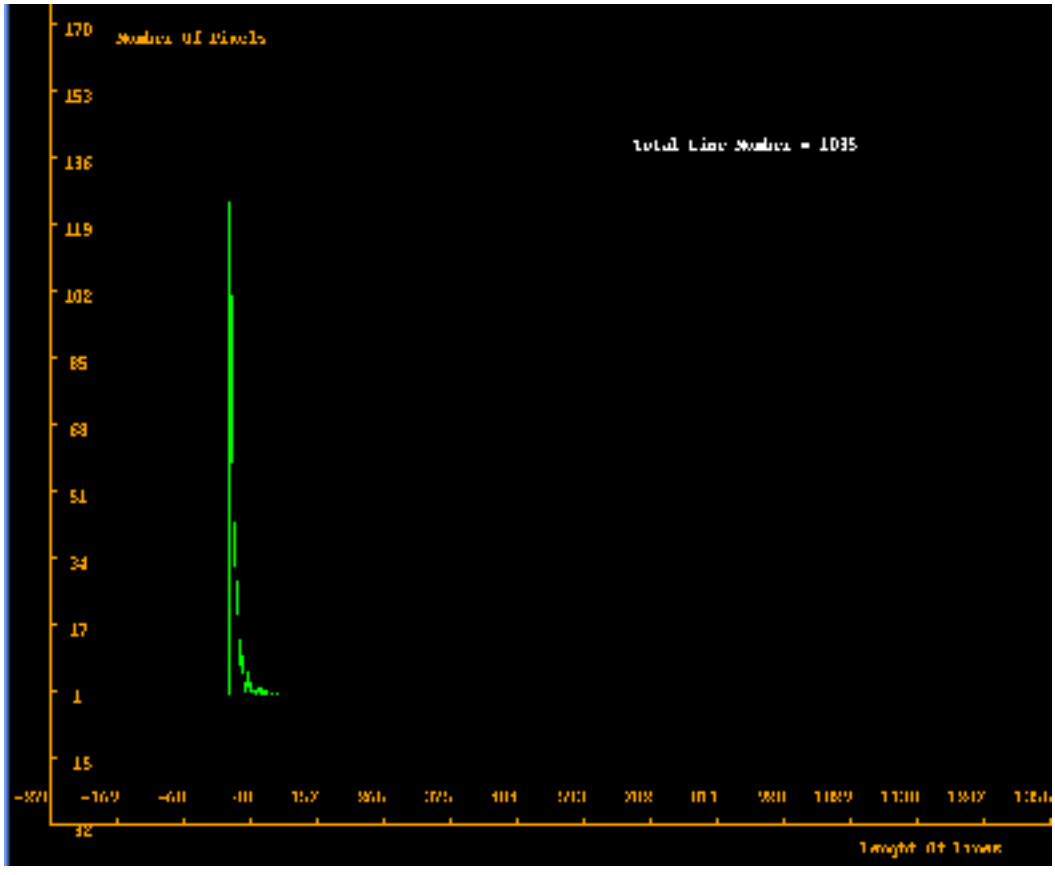
Gray scale image



Edge detected image

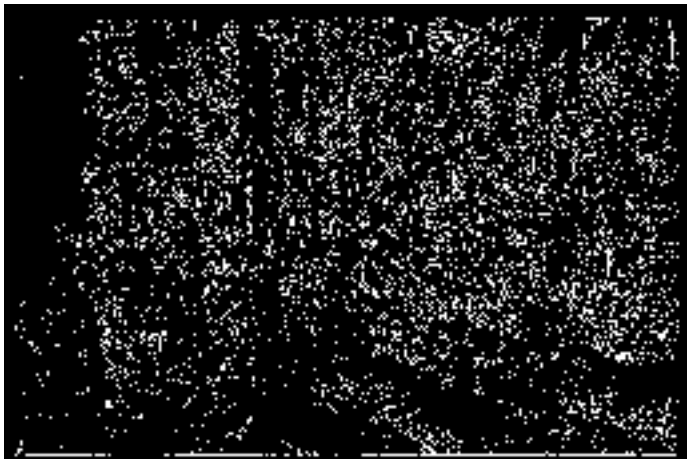
Plots showing the number of pixels of each color and the number of vertical lines





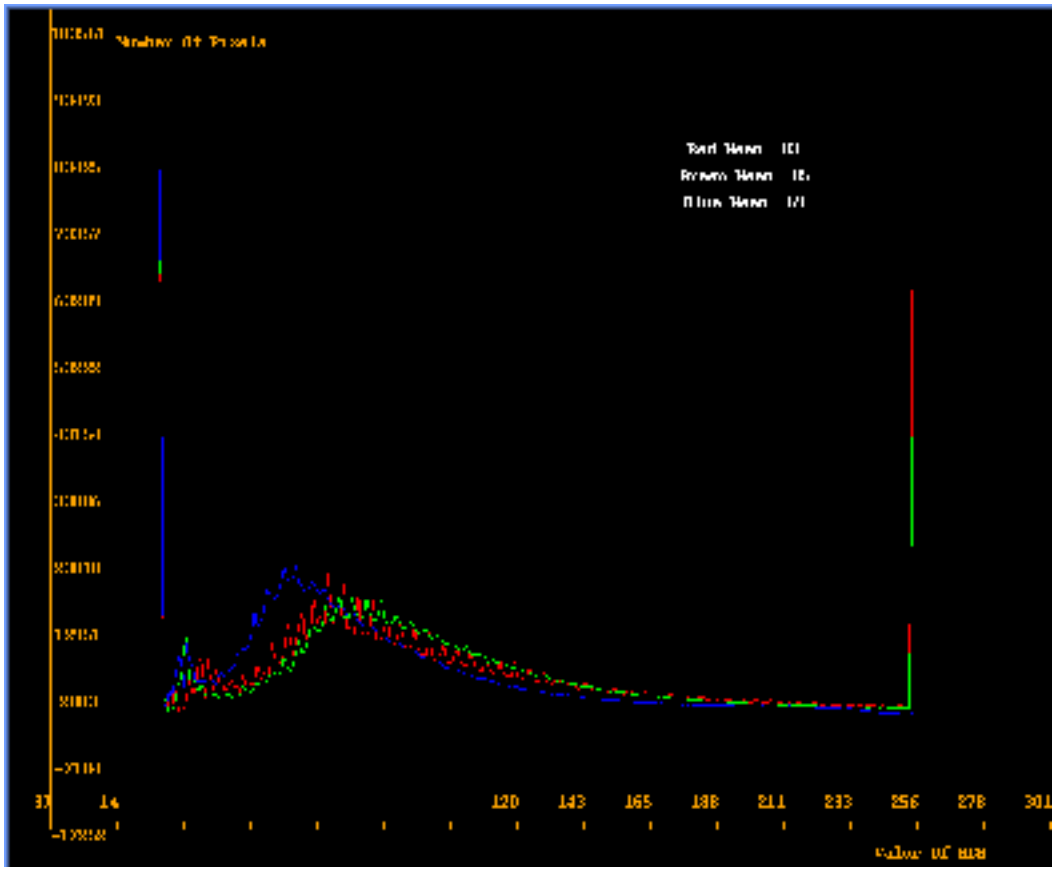
Red mean = 83
Green mean = 85
Blue mean = 69
% long lines = 0.94
% short lines = 80.56
SBE = 0.33

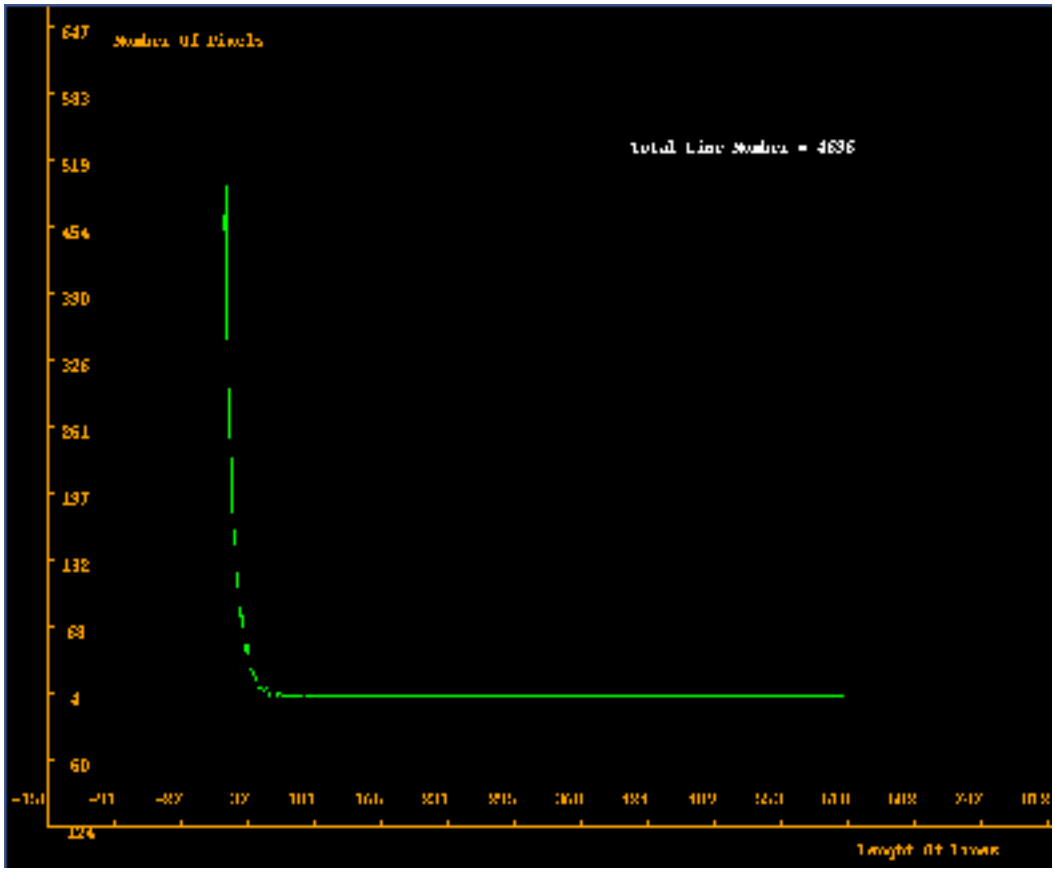
Gray scale image



Edge detected image

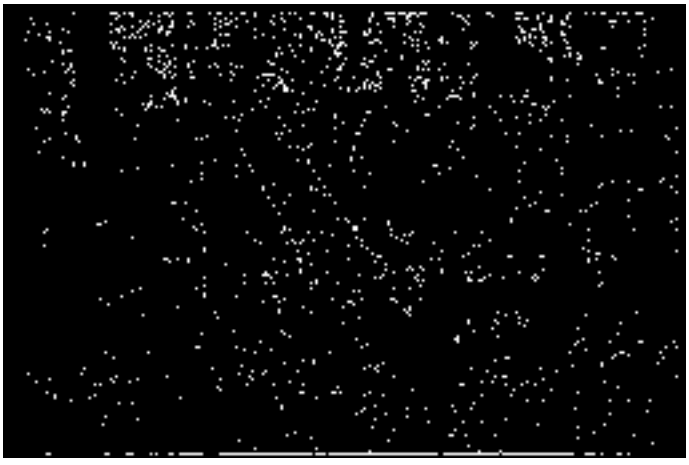
Plots showing the number of pixels of each color and the number of vertical lines





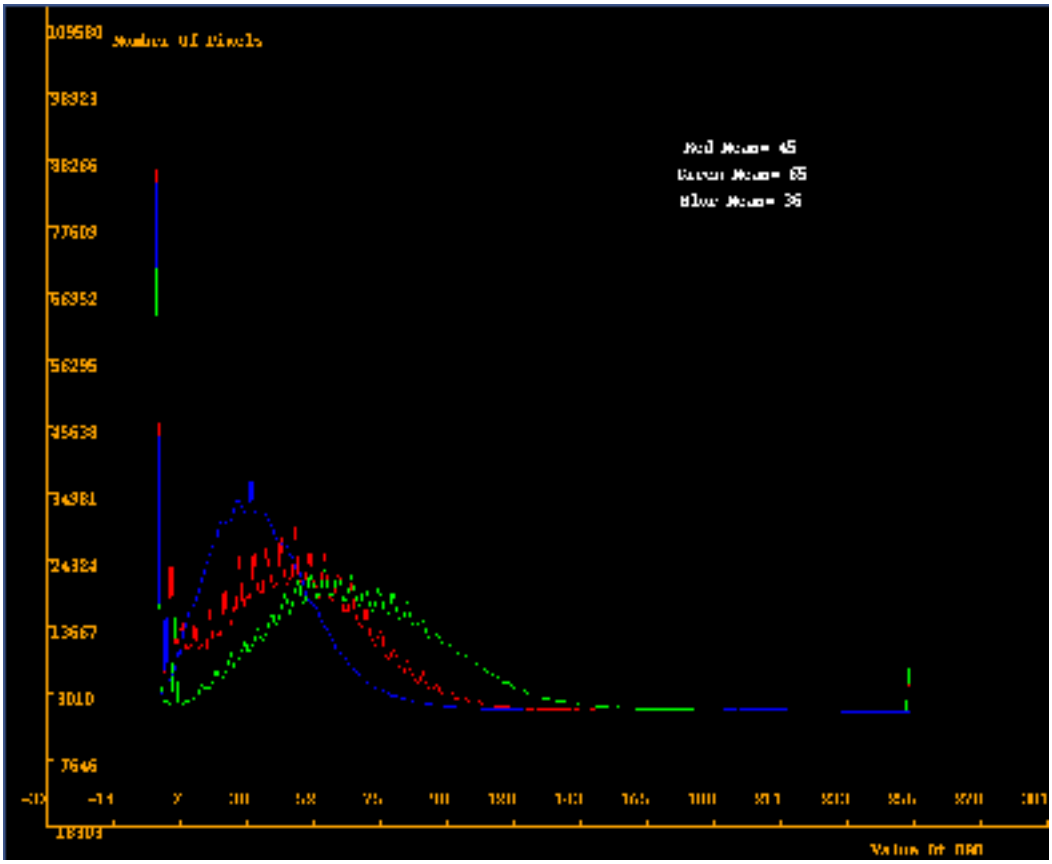
Red mean = 45
Green mean = 65
Blue mean = 36
% long lines = 3.45
% short lines = 68.45
SBE = 0.50

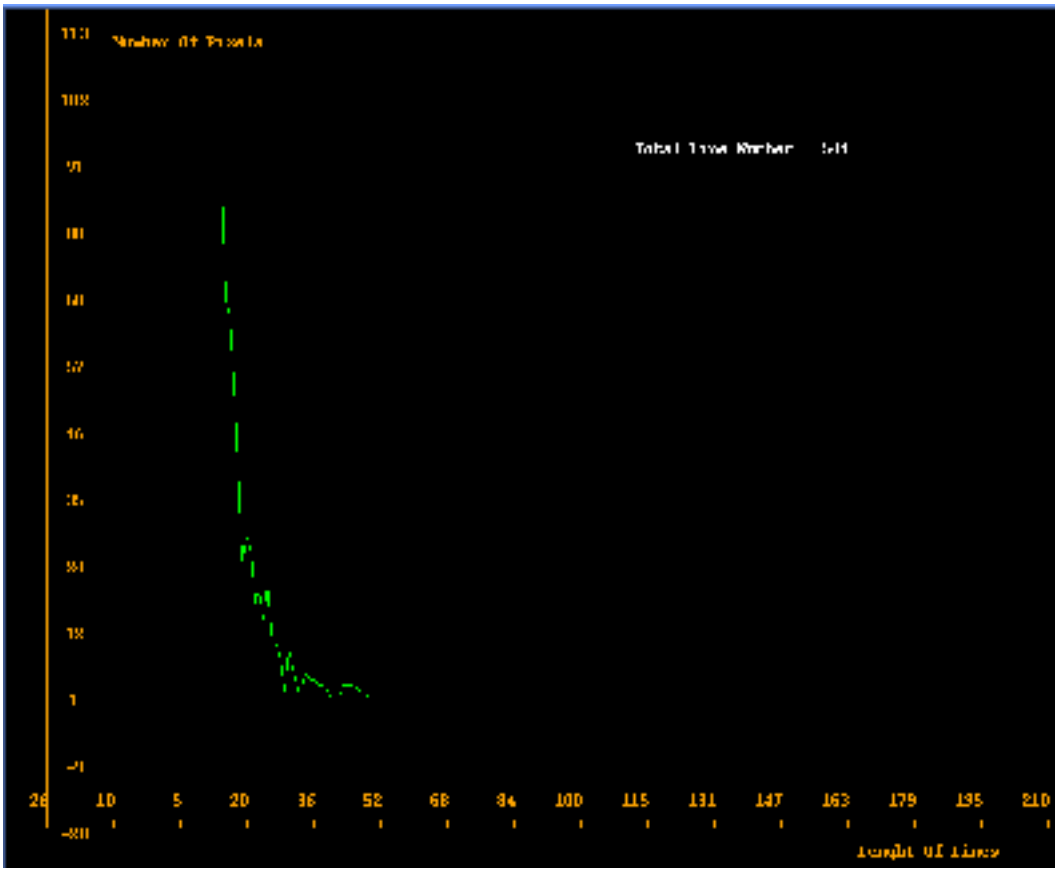
Gray scale image



Edge detected image

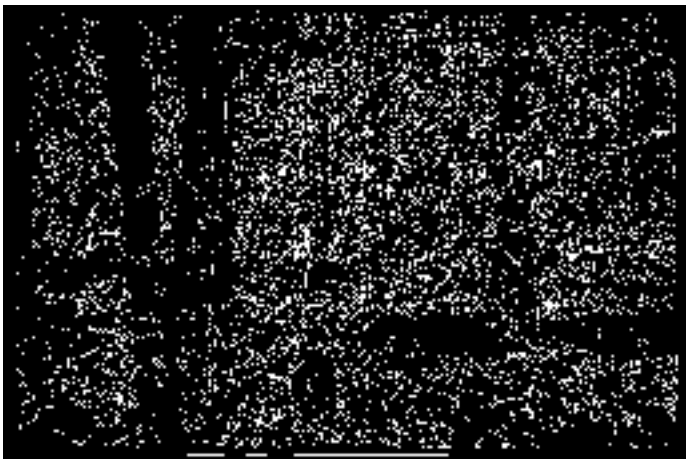
Plots showing the number of pixels of each color and the number of vertical lines





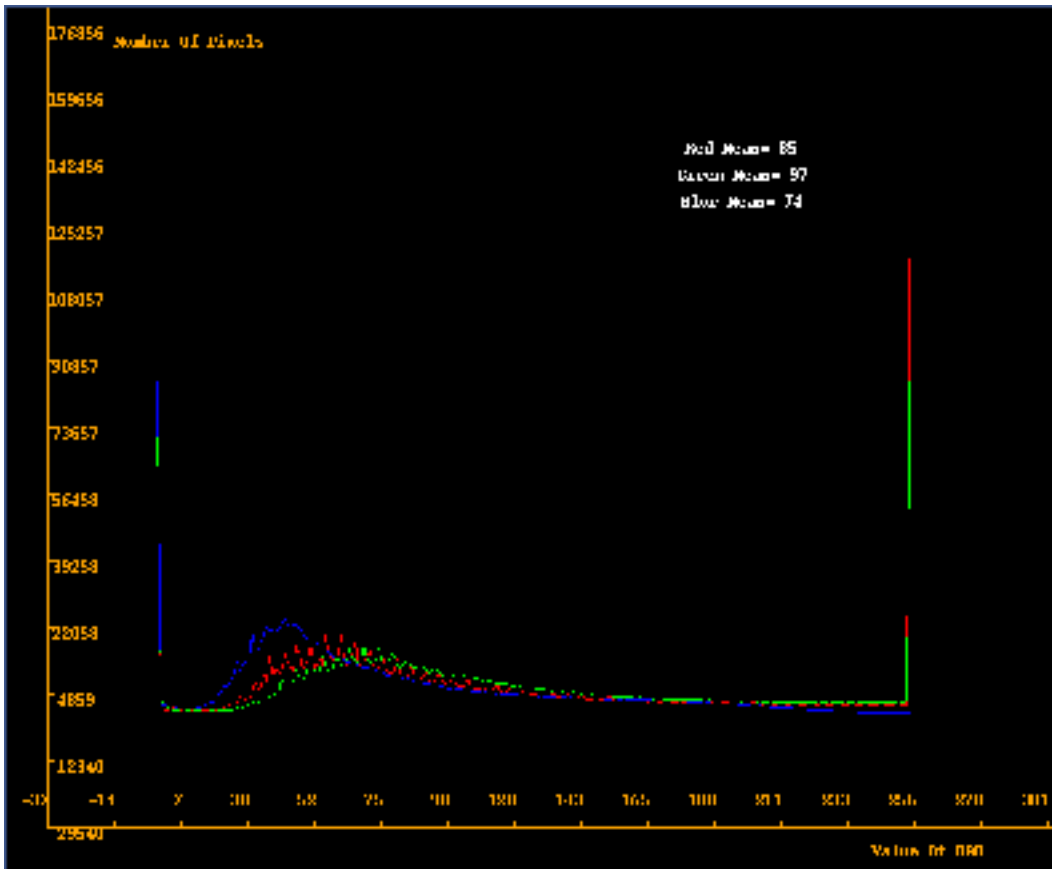
Red mean = 85
Green mean = 97
Blue mean = 16
% long lines = 2.47
% short lines = 76.10
SBE = 0.37

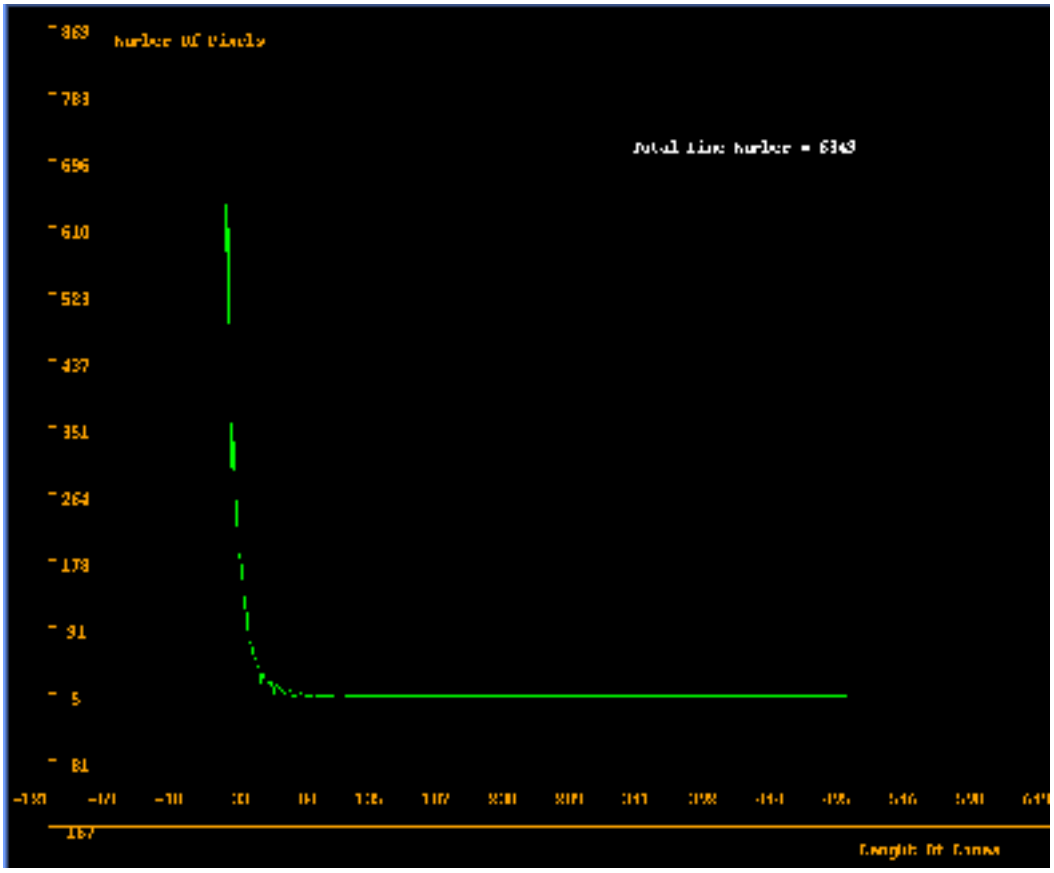
Gray scale image



Edge detected image

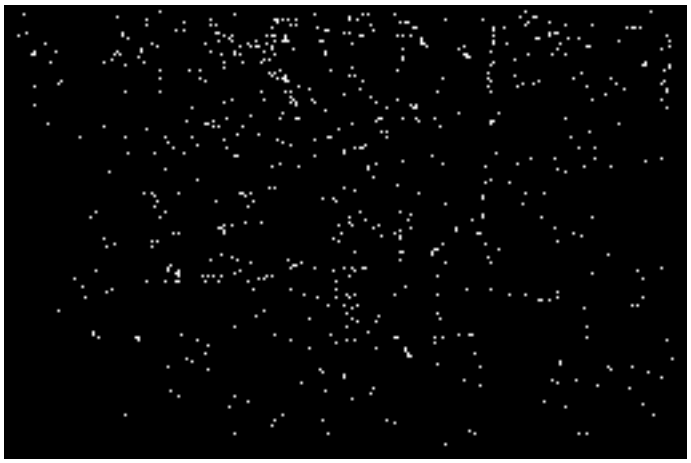
Plots showing the number of pixels of each color and the number of vertical lines





Red mean = 37
Green mean = 52
Blue mean = 24
% long lines = 3.24
% short lines = 73.89
SBE = 0.56

Gray scale image



Edge detected image

plots showing the number of pixels of each color and the number of vertical lines