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

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

proposal for

Comparative Analysis of FFT Algorithms In Sequential and Parallel Form

submitted to fulfill the semester project requirement for

EE 4773/6773: Digital Signal Processing

September 21, 1996

submitted to:

Dr. Joseph Picone

Department of Electrical and Computer Engineering
413 Simrall, Hardy Rd.
Mississippi State University
Box 9571
MS State, MS 39762

submitted by:

Michael Balducci, Ajitha Choudary, Jonathan Hamaker

Parallel DSP Group
NSF Engineering Research Center
The Institute for Signal and Information Processing

Mississippi State University
Box 6176
Mississippi State, Mississippi 39762
Tel: 601-325-2081
Fax: 601-325-7692
email: {balducci, ajitha, hamaker}@erc.msstate.edu

I. ABSTRACT

This project will provide a comparative analysis of seven FFT algorithms (Quick Fourier Transform, Fast Hartley Transform, Prime Factor algorithm, Decimation-in-Time-Frequency algorithm, Radix-2 algorithm, Radix-4 algorithm, and Split-Radix algorithm). All algorithms will be compared in sequential and parallel form based on computational time and complexity. This task will be accomplished by 1) implementing the FFT algorithms in sequential form, 2)Implementing the FFT algorithms in parallel form, 3) Running a battery of performance tests on each of the implementations and 4) collecting statistical data by which we may compare the algorithms in each of their forms. Using the data collected, we will determine which algorithm is best suited for a given implementation and data set. The information we obtain will have future implications in the areas of image processing, speech processing, radar applications, and most every facet of signal processing.

II. INTRODUCTION

Background

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. This is possible, because once the signal is converted into the frequency domain the noise or unwanted signal frequencies can effectively be 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 [7].

For most applications computation time plays a significant role in the use of FFT's. The less time spent computing means less 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 FFT. 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 on a single processor or by spreading the data 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 [7].

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. Using the Fourier Transform,

the time-domain signal is converted into the frequency domain. Likewise, the time-domain signal is resolved into exponential components denoting amplitude as the frequency varies. The inverse fourier transform, which returns the frequency-domain signal back to the time-domain, is given by Figure (1b).

(a) $$H(\omega) = \int_{-\infty}^{\infty} h(t)e^{j\omega t}dt$$

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

Figure 1 Fourier Transform 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 because the computations involved are significantly lesser and more manageable. For a periodic signal, the period of the signal is equal to the length of the sequence. The discrete Fourier transform is defined as shown in Figure (2a) and its inverse is shown in Figure (2b) [7].

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

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

Figure 2 Discrete Fourier Transform and Inverse Discrete Fourier Transform

Computation cost is a factor which greatly affects which DSP tool an engineer uses. 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. For large N the computational costs are extremely high. The Fast Fourier Transforms (FFTs) have been developed to reduce these computational costs. [7]

Fast Fourier Transforms

FFTs use several different properties of DFT's to minimize the computations. The FFT serves to increase the efficiency of the DFT by splitting the data set into smaller DFT's. The FFT algorithms utilize properties of DFT such as periodicity and the symmetry of cosine and sine terms present to increase efficiency. For this project we will be using the following seven algorithms **1)** Prime Factor algorithm (PFA) **2)** Quick Fourier Transform algorithm (QFT) **3)** Fast Hartley Transform **4)** Decimation-In-Time-Frequency FFT algorithm (DITF) **5)** Radix-2 algorithm **6)** Radix-4 algorithm and **7)** Split-Radix algorithm.

Prime Factor Algorithm is used when N can be factored into mutually prime factors. N should be able to be represented in the form of 2**p * 3**q *5**r * 7**s * 11**t * 13**u * 16**v etc. where each coefficient is a prime number. If the data size is of that type, the Prime Factor Algorithm is considerably faster than the others. Modules are written for 2,3,4,5,7 etc. and the data is transformed using these in nested loops. The computational complexity of PFA is of the order $O(N\log_2 N)$ [8].

The **Quick Fourier Transform** uses the symmetry of the cosine and sine terms in the DFT to decrease the complexity of the DFT calculations. The data set is first broken into real cosine terms and imaginary sine terms and those into their even and odd parts. The length-(N+1) Discrete Cosine Transform (DCT) terms and length-(N-1) Discrete Sine Transform (DST) terms are recursively broken into a (N/2 +1) DCTs and (N/2-l) DSTs respectively. In this manner, the number of additions and multiplications is dramatically decreased. Its developers claim that it is faster by a factor of two over direct methods providing an order of complexity of $O(N\log_2 N)$. The developers also claim that it is the best algorithm for prime length data sets. The number of multiplications involved in computation of the QFT are N^2 and number of additions involved is N^2+4N [4].

The **Hartley Transform** is very similar to the FFT. The major difference is that the FHT reduces the computational cost of the FFT's kernel by eliminating the complex arithmetic. The FHT reduces the number of calculations involved by performing real arithmetic instead of the complex calculations performed by the FFT. For a given point, the FFT must perform 4 multiplications and two additions. For the same point, the FHT performs two additions while only performing one multiplication. The one draw back to the FHT is that additional computations are required to convert the results to equivalent FFT results. The FHT has a complexity of $O(N\log_2 N)$ [6].

The **Decimation-In-Time-Frequency FFT Algorithm** is a combination of the decimation-in-time (DIT) and the decimation-in-frequency (DIF) algorithms. The DITF algorithm begins the breakdown of the DFT using the DIT and switches at some intermediate point to the DIF. This is

effective because the bulk of computation time for the DIT is at the end of the algorithm and the bulk for the DIF is at the beginning. Thus, switching at some mid-point allows one to miss both of the bulk regions of computation time. This gives a great decrease in the number of multiplications. [5]

Radix-2 and Radix-4 algorithm

The Radix 2 and 4 algorithms can be implemented by either using decimation in time (Cooley and Tukey) or by decimation in frequency (Sande and Tukey). These algorithms use the data of the form of 2^N. If the data is not of the form 2^N then the zero padding is done. Using the fact that the data is of the form 2^N, the summation is split into even and odd components and then the computations are performed. By using these algorithms the number of computations are reduced greatly compared to the Discrete Fourier Transform. The Radix-2 algorithm needs $(N/2)\log_2(N)$ complex multiplications and $N\log_2 N$ additions for both decimation in time and decimation in frequency. The Radix-4 differs from Radix-2 in the way the samples are split. In case of Radix-4 they are split into four. The Radix-4 algorithm requires $3N/8\ \log_2 N$ complex multiplications and $3N/2\ \log_2 N$ additions for both decimation in time and decimation in frequency. There is an increase in number of additions but there is a decrease in number of multiplications compared to the Radix-2 [7].

Split-Radix algorithm

The split-radix algorithm uses makes use of both radix-2 and radix-4 in one algorithm. It also, like the radix algorithms operates on data of the form 2^N . The zero padding is used for the data not in the required form. It uses radix-2 for the even samples of DFT and radix-4 for odd samples of DFT. It uses radix-4 for odd samples of DFT as these require multiplication by the twiddle factor. The radix-4 is used for the decomposition of odd samples to improve the efficiency. The split-radix algorithm is of the same order as radix-4. But it requires lesser than or equal computations as that of radix-4 depending on samples[7].

Parallel Systems

Concurrent or parallel processing is a process in which a large task is divided into smaller tasks and the smaller tasks are distributed over the available processors, or large data set is divided into smaller data sets and then processed on various processors. To distribute data or to gather results after or when computing, processors need to communicate. The communications between the processors can be provided by Message Passing Interface (MPI) [1]. MPI, as the acronym suggests, is a protocol with library calls that passes the data and instructions to the processors. The user has to specify the MPI calls to be utilized with all the parameters. The advantages of using MPI over other message passing systems are that it has the majority of the salient features of other message passing systems with a few exceptional features of its own. These features include support for libraries and inter-group communication [1][2].

The main issue that has to be considered in writing code using MPI is the communications cost involved in message passing. Even though FFT algorithms exhibit a high degree of parallelism, they require a lot of communications due to the high amount of data dependency. The

communication overhead must be kept in view when writing the applications and must be minimized or overlapped with those of computations to receive the benefits that parallel processing offers. Stability should also be considered when writing the applications. When the communication costs exceed the computation costs the parallel implementation is no longer a viable alternative to sequential code [1][2].

III. PROJECT SUMMARY

The purpose of this project is to make a performance comparison between FFT algorithms implemented in sequence and those implemented in parallel. This project is being conducted in conjunction with the Parallel DSP Project at Mississippi State University. The two sides of the project are being coordinated by Aravind Ganapathiraju and Dr. Joseph Picone. The main tasks of this project are set forth below:

- Implement the QFT, FHT, PFA, DITF, Radix-2, Radix-4, and Split-Radix FFT algorithms in a sequential form. The code for these algorithms will be written in C++ and compiled under Solaris 2.51 using the gnu compiler. A wrapper will be developed which will be a common interface to each of the algorithms. The wrapper will be capable of handling both real and complex single-dimensional data. The majority of each sequential program will be based on public-domain code. Each public-domain program must be tested and modified before it can be put in parallel form.

- Implement the same algorithms in a parallel form. The high degree of parallelism in FFTs will be used to develop parallel forms of the sequential code. The parallel code will split the tasks into sub-tasks which will be executed on separate processors. The inter-process communication will be accomplished using MPI (Message Passing Interface.) It is important to ensure that only the structure of the algorithm is changed and not the algorithm itself.

- Testing of sequential and parallel code. In order to test the code objectively, each algorithm will be tested (in sequential and parallel form) with the same data sets. The tests will each be run on equivalent, unloaded Sparc stations. Each algorithm implementation will be analyzed according to how fast it completes each data set and the number of arithmetic operations. Testing will be done on the following architectures: Supremacist, 2 CPU Sparc 20, Dual CPU Intel Pentium Pro, and a multiple CPU UltraSparc I.

- Analysis of test data. The test data will be compiled to determine the fastest algorithm, and the least complex algorithm among the sequential and parallel implementations.

IV. EVALUATION

In order to evaluate anything, a basis on which to compare must be chosen. We have elected to contrast the performance of the various algorithms based on their execution time, complexity and memory usage.

- **Execution Time** is a key factor in determining the efficiency of an algorithm. The execution

time, however is greatly affected by several factors. Chief among these are the characteristics of the data (real/complex), the machine on which the code is executed, and the number of parallel processors operating on the data. We will account for these factors by using the same data sets for each algorithm implementation, and by running a large number of repetitions for each implementation and data set. Execution time will be measured using the unix utility *gprof* for systems on which it is available. For test run on the SuperMSPARC the SuperMSPARC Performance Monitoring tools at the MSU Engineering Research Center will be used, and for all other systems,.the unix utility *utime()* will be used.

- **Complexity** is another key factor in choosing an algorithm. Two key indices of algorithm complexity are number of multiplications and additions. We plan to measure these using functions which take the place of the operators. These functions carry out the operations and increment a counter.

- **Memory usage** is a factor which is hard to quantify but is crucial to the performance of an algorithm. For instance, with infinite memory, one could just create a lookup table with all possible output values. However, memory is not infinite, thus it must be considered in algorithm coding and development. We plan to quantify memory by overloading the C++ *new()* operator. Much like the quantification of arithmetic operations we will implement a new version of the *new()* operator which increments a count each time it is called.

These evaluations will be made on each implementation of each algorithm and will be analyzed according to the following:

- **Sequential-vs-Parallel** - Each algorithm will first be tested to ensure that the data from the parallel implementation is equivalent to the data from the sequential implementation. Next, we will do simple statistical analysis to determine which implementation is faster for each algorithm. We will determine the speedup using the average results of testing. We expect the parallel implementation will be faster, but the communication overhead may prove otherwise.

- **Sequential-vs-Sequential & Parallel-vs-Parallel** - Next we will compare the algorithms against each other. We expect that the order of algorithm speed in sequential form will not be the same as in parallel form.

One of the systems to be used for testing is the SuperMSPARC. The SuperMSPARC is a 32-processor mulitcomputer specially equipped to allow for the monitoring of parallel code execution. The SuperMSPARC's performance monitoring tool's allow for an parallel application developer to observe performance related events graphically, such as interprocess communication and execution time, and link these events to particular locations of the user's code. This better enables to programmer to identify bottle necks and to examine the communication overhead inherent to parallel processing.

The SuperMSPARC uses a combination of custom designed hardware and software to collect event data. In order to collect data, the program must first be linked with the "probe" library at compile time. This will allow for event data to be transmitted the Probe Acquisition Board(PAB).

The formatting and transmitting of event data are the sole source of intrusiveness. The PAB is a small single-width card that sits on Sun's SBus. The job of the PAB is to collect, time stamp (at a resolution of 100ns), and transmit the probes to a central recording site. The probe data can then be analyzed by other software tools developed at the ERC.

The data sets will each be comprised of varying data types. Both single-dimensional and two-dimensional data will be tested. Each set will use four-byte float values of either real or complex data. Each data set will vary in length and type. The data type variations will aid in pointing out algorithms that may be more efficient for only certain data sets. By varying the size of the data set, the limitations of the parallel implementations should also be shown.

V. PROJECT DEMONSTRATION

Real-time analysis of this project's concepts will be made available through a graphical user interface (GUI). The GUI will be largely user configurable. The user will be able to decide which algorithms are compared, which architecture each is run on, the number of CPUs each will use and the type of data on which each will operate. The results of the real-time experiment will be available to the user in a series of frames. One frame will contain a real-time communication simulation which will represent communication between parallel processes and processor utilization. Another frame will give a real-time graphical representation of the number of arithmetic operations and memory accesses that each algorithm performs. Yet another frame will contain a running timer on each algorithm chosen. In addition to giving the user the option of algorithms, there will be an "auto" function which will determine the optimum algorithm for a given data set (chosen by the user). This GUI will be extremely useful not only as a visualization tool for novice users but also as an analytical tool for our group as we attempt to optimize the various algorithms.

VI. SCHEDULE

A schedule for the major tasks in this project are shown in Figure 4.

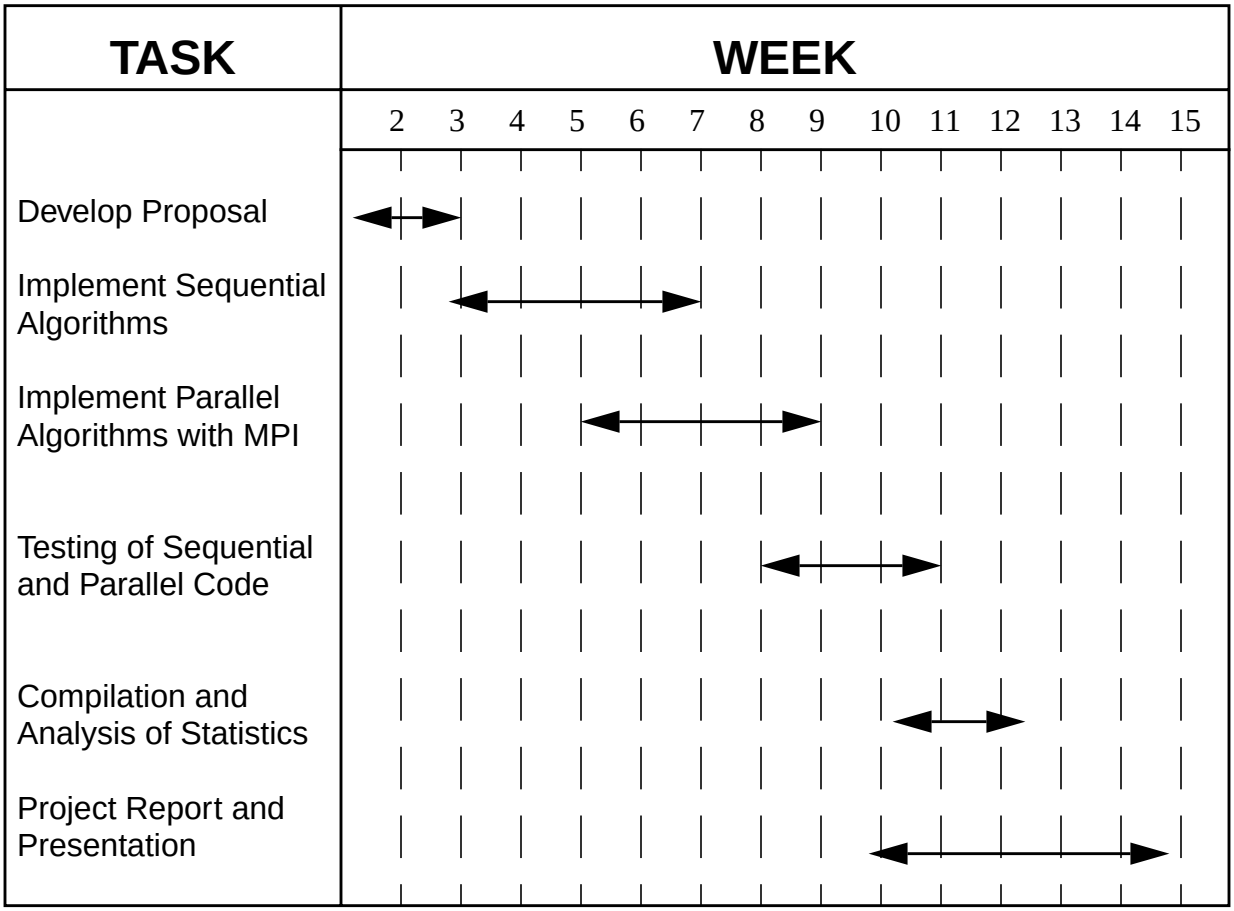


Figure 4. Schedule of Completion of Key Tasks.

VII. REFERENCES

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

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

[3] R. Bracewell. *The Hartley Transform* Oxford, England: Oxford Press, 1985, chapter 4.

[4] Patrick Bridges, Nathan Doss, William Gropp, Edward Karrels, Ewing Lusk, and Anthony Skjellum. "Users guide to MPICH, a Portable Implementation of MPI", 1994.

MSU Argonne Joint Documentation.

[5] "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.

[6] Press, William H., Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling. *Numerical Recipes in C, The Art of Scientific Computing.* Cambridge University Press, Cambridge, Mass., 1988. pp 407-447.

[7] 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.

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

[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] 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.

[11] Temperton, Clive. "Nesting Strategies for Prime Factor FFT algorithms." *Journal of Computational Physics*, vol 82, Number 2, June 1989.

[12] D. P. Kolba and T. W. Parks, "A prime factor FFT algorithm using high speed convolution," *IEEE Trans. on ASSP*, vol. 25, pp. 281-294, August 1977.

[13] Temperton, Clive. "A self sorting in-place prime factor real/half-complex FFT algorithm," *Journal of Computational Physics*, vol 75, pp. 199-216, 1988.

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

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

[16] Baher, H. *Analog and Digital Signal Processing*. John Wiley & Sons, New York, 1990. pp 349-356.

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

[18] Zheng, Rong. "A Study of Index Mapping for Prime Factor Algorithm of FFT." *The Proceedings of the 6th International Conference on Signal Processing Applications & Technology*, pp. 593, 1995.

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

[20] Elliot, Douglas F. and K. Ramamohan Rao. *Fast Transforms Algorithms, Analyses, Applications.* Academic Press, Inc., Orlando, 1982. pp 58-168, 447.

[21] Brigham, E. Oran. *The Fast Fourier Transform and Its Applications.* Prentice Hall. Englewood Cliffs, New Jersey 1988. pp 131-156.

[22] Peled, Abraham, Bede Liu. *Digital Signal Processing, Theory, Design, and Implementation.* John Wiley & Sons. New York, 1976. pp 142-171.

[23] Elliot, Douglas F. *Handbook of Digital Signal Processing Engineering Applications.* Academic Press, Inc. San Diego, 1987. pp 527-630.

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

[25] C.S. Burrus, "Unscrambling for Fast DFT Algorithms." *IEEE Trans. on ASSP*, vol. 36, no. 7. July 1988. pp 1086-1087.

[26] James S. Walker, "A New Bit Reversal Algorithm," *IEEE Trans. on ASSP*, vol 38, No. 8, August 1990. pp 1472-1473.

[27] Henrik V. Sorensen, et. al, "Real-Valued Fast Fourier Transform Algorithms," *IEEE Trans. on ASSP*, vol ASSP-35, No. 6, June 1987. pp 849-855.

[28] Jeffrey J. Rodriguez, "An Improved FFT Digit-Reversal Algorithm," *IEEE Trans. on ASSP*, vol. 37, no. 8, August 1989. pp 1298-1300.

[29] Martin Vettereli and Pierre Duhamel, "Split-Radix Algorithms for Length-p^m DFT's," *IEEE Trans. on ASSP*, vol. 37, no. 1, January 1989. pp 57-64.

[30] 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.

[31] Ivan W. Selesnick and C. Sidney Burrus, "Automatic Generation of Prime Length FFT Programs," *IEEE Trans. on Signal Processing*, vol. 44, no. 1, January 1996. pp 14-24.

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

proposal for

Audible Frequency Detection and Classification

submitted to fulfill the semester project requirement for

EE 4773/6773: Digital Signal Processing

September 30, 1996

submitted to:

Dr. Joseph Picone

Department of Electrical and Computer Engineering
413 Simrall, Hardy Rd.
Mississippi State University
Box 9571
MS State, MS 39762

submitted by:

David Gray, Craig McKnight, Stephen Wood

Audible Frequency Detection and Classification Group
Department of Electrical and Computer Engineering
Mississippi State University
216 Simrall, Hardy Rd.
Mississippi State, Mississippi 39762
Tel: 601-325-2081
Fax: 601-325-7692
email: {gray,wcm1,srw1}@ece.msstate.edu

I. ABSTRACT

Current music software relies on external input from MIDI capable devices. 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.

Tone recognition will be accomplished by transforming time domain data into frequency domain data and sifting out the overtones of instruments played into a microphone. The transformation will likely be accomplished using a Fast Fourier Transform (FFT), spectral estimation techniques, or Prony's Method. Once the note or notes are identified, the user will be given information about the frequency content of the tones played.

II. INTRODUCTION

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* is an education software package that teaches students relative pitch, harmony, intervals between notes, and chord structure in an interactive environment. For example, a student studying intervals would be given an interval to play by the computer. The student would then play the interval on the keyboard or other instrument and the computer would tell the student if the interval they 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 the playing of the note [3].

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. The signal is never analog during the entire signal chain from the instrument to the computer. 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.

III. PROJECT SUMMARY

Archive Development

An archive of time-domain data taken from various sources of interest will be key to developing a successful frequency detection and classification software package. An archive of sampled data files will be developed and used in this project for testing and evaluation procedures. It will be a collection of sampled data files from sources which will include a digital function generator, acoustic guitar, electric bass guitar, trumpet. Each file will focus on a particular aspect of the project. The files will range from containing single frequency data without instrument overtones to multiple frequency data with instrument overtones.

File names for the database will be standardized to the following format: **<source>_<# of principle frequencies>_<tuning><note>**.raw. **<source>** will be a three-character abbreviation for the instrument. **<# of principle frequencies>** will be a numeral denoting the number of principle frequencies (i.e., the number of notes played) in the sample. **<tuning>** will be a single-character denoting whether the sample is sharp, flat, or in tune: +, -, ~, respectively. **<note>** will be a three-character denotation of the note name in the sample. For example, an A220 will be 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: gui_2_+A4~_~C5+.raw.

The archive will be constructed of data to be used in three phases of the project. The first phase will be single tones to be used on the tuner function of the software. For phase I of the project, three instruments will be used in the data archive: a guitar, a bass guitar and a trumpet. Each instrument will be recorded in three sets of cases. For each case, a note will be played in tune, then that note will be played out of tune flat, and then it will be played out of tune sharp. When this is completed, phase I of the archive will contain 27 samples.

In Phase II, two instruments, the guitar and bass guitar, will each be recorded playing each of the twelve intervals less than an octave. This will complete an archive of 24 samples for Phase II testing.

Phase III of the testing will be an archive of sequential notes designed to test if the software is effective in detecting note changes. Again, three instruments will be used, the guitar, bass guitar, and trumpet. For each file, a sequence of three notes will be recorded. Each instrument will play three sequences of intervals. This will build a Phase III archive of 27 samples.

Phase IV will be an archive of chord data recorded from the guitar and the bass guitar again. Each instrument will play a major, minor and diminished triad based on three tones. This yields the Phase IV archive of 18 samples.

Algorithms and Strategies

As this archive of data is being built, software modules will be written that will evaluate the frequency content of the data in much the same way that a commercial tuner does. Initially, the software will focus on the evaluation of a single frequency. The software package will be written in ANSI standard C and compiled for use in a Unix Windows environment.

The general algorithm for detecting the frequencies that are played by the user will be to acquire time-domain data for a given window and transform the data into the frequency domain. The transformation to the frequency domain will initially be done using an FFT [4].

The resolution of an FFT is defined as

$$\Delta f = (f_S)/N \qquad (1)$$

where f_s is the sample frequency and N is the number of points in the FFT [5]. For example, a 512 point FFT using an 8kHz sample frequency has a resolution of 15.625 Hz. With this resolution it would be impossible to distinguish notes lower than middle C on a piano keyboard! Therfore there is some concern that an FFT will not give the frequency resolution needed for the project to be successful. So, spectral estimation techniques will also be investigated for determining the frequency content of the signal [6]. A linear prediction method known as Prony's Method is also a possibliity for transformation to the freqency domain [7].

Once the data has been converted into the frequency domain, the software will analyze the frequency data to determine the principal tone played. This will be based on the amplitude of the frequency components contained in the data.

When an instrument is played, it resonates with a principal or fundamental frequency, but it also resonates with high frequency tones. These high frequency signals are called overtones. Every instrument has a specific pattern of overtones associated with it. This overtone pattern differentiates the timbre of one instrument from the timbre of another, so the process of sifting the frequency content of an instrument tone is not as simple as removing specific frequencies [8].

Keeping this in mind, it seems appropriate to sift the frequency data based on the amplitudes of the frequency spectrum. Therefore, the software will analyze the of the frequency spectrum for the portion containing the principle tone. All of the other frequency data will be considered overtones, and will be ignored for the purpose of the tuning module and the interval relation module.

The software will be written in several stages. Initially, the software will be written to identify all frequency data in the sample. However, the samples will, in this stage be pure tones constructed using a digital function generator.

Once the problems are identified and resolved with this ideal data, the software will be tested with

data containing a principal frequency and overtones. This data will be generated by a digital function generator and data recorded from real instruments. The software will be enhanced to remove the overtone frequencies and to identify only the fundamental frequencies in the data. Removing the overtone frequencies could require implementation of very sharp, narrow filters.

Once the software is working properly in these cases, testing can begin using real time data. At this point, the software should be able to identify the principal frequency data of a given sample and to graphically display both the sample's frequency and how much the tone must be altered for it to match the frequency of the closest letter pitch. For example, if the principal frequency of the sample was 434 Hz, then the software should evaluate in real time and display this fact and also that the sample is flat of A440.

When this aspect of the software works properly, the first major section of the project will be complete. The next major section will be to enhance the software to detect multiple principle frequencies in the same sample. The procedure will be similar to the initial portion of the project.

The software will be evaluated with generated pure tone data again. This time, however, the data will contain more than one specific principle frequency. That is, the amplitudes of the signals will be proportional, whereas the amplitudes of the overtone frequencies in previous data were much smaller in comparison with the principle frequency amplitude. Once the software works properly for the ideal case (no overtones) and for generated data containing multiple frequencies and overtones, testing will proceed with real instrument data from data files, and finally to real instrument data in real time. When this phase of the project is complete, and a user plays intervals or chords into a microphone, the software will identify the tones played and graphically display the tones on a staff. Each tone will be identified, and the harmonic relationship between them will be displayed.

If the user has an instrument that only plays one note at a time, he or she will be able to take multiple samples and compare them with one another. The software should perform equally well with a single instrument, or comparing data from multiple instruments.

Demonstration of Software

The result of the project will be a software package that operates in two major modes. In the first, the software can be used to tune acoustical instruments in the same way a commercial tuner would be used. In the second, the software can be used to identify multiple tones and to classify the interval relationships between them.

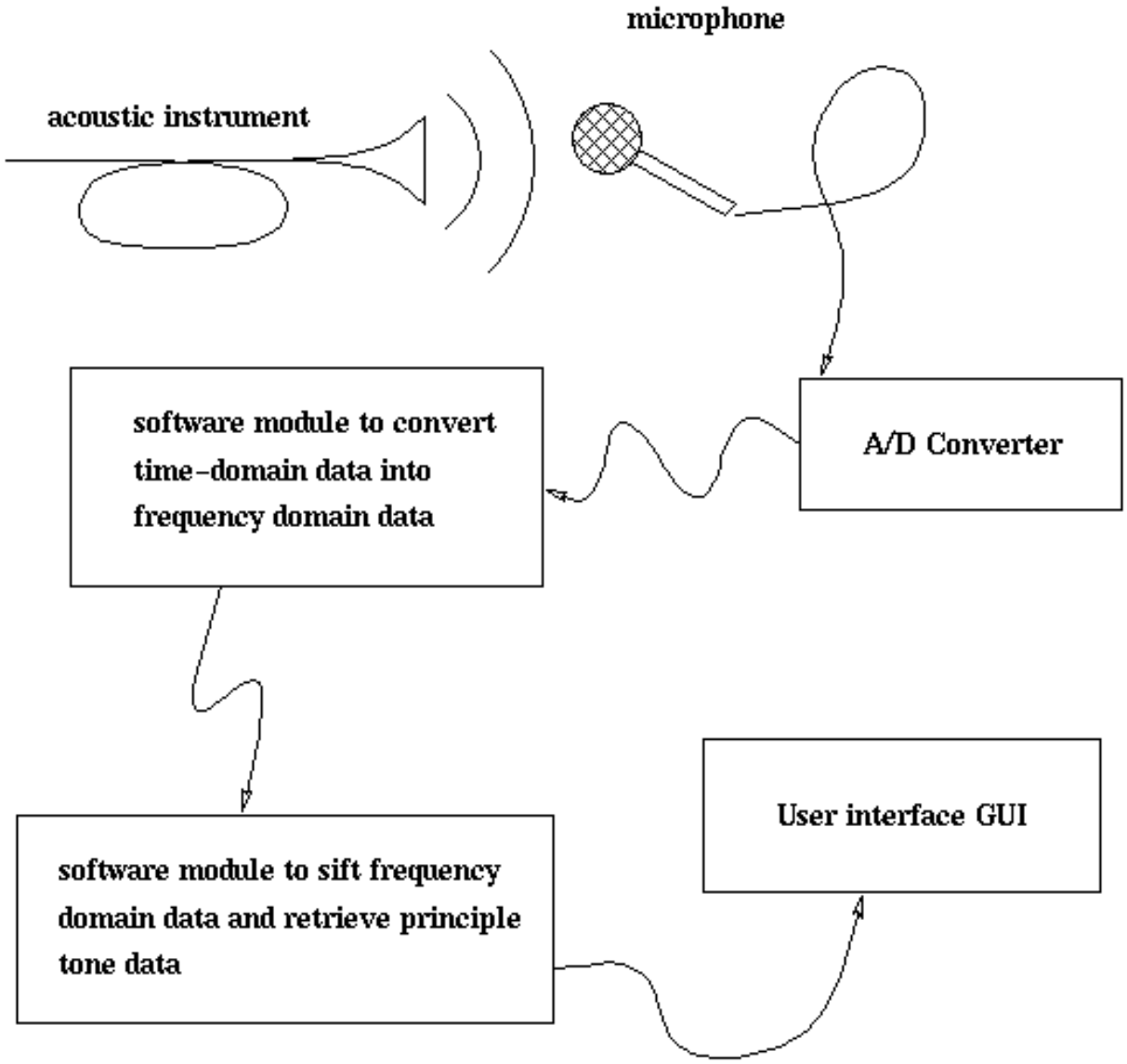
To demonstrate the tuner mode of the software, a user will play an instrument into a microphone at the terminal. The software will graphically tell the user what note is being played and whether the note is flat, sharp, or in tune. This, of course, will be performed in real time so that the user can adjust the instrument before the approaching performance!

Demonstrating the interval detection mode of the software will be similar. The user will change the mode of the software by using the GUI. When everything is set, the user will play two notes simultaneously into the microphone. The software will evaluate the sounds, and graphically

display what notes are present, and the interval between them.

Of course, if there are no musicians present at the demonstration, the software can also be tested in each of it's modes using the arhcive that will have been built

.



Simple layout of the finished product, including the main modules of the software and what functions they will perform.

IV. EVALUATION

The methods of testing the tuning capability and the interval detection capability of the project will be similar to one another. In both cases, evaluation will proceed in stages. For the tuner, the first stage will be to input a file containing a digitized sinusoid consisting of only one frequency. The frequency will be a known value, but may be "out of tune."

The second stage will be to test the tuner with data files that contain signals that have multiple frequencies. These input signals will be data recorded from various instrument sources of interest. When the tuner can accurately classify these multiple frequency signals, it will be tested using real time input from instruments representing various instrument families. A commercial tuner will be used to verify the results of the software package in real time.

The interval detection and display function of the project will be tested in a similar manner. First, input files containing intervals of pure sinusoids will be evaluated. Then data files containing signals recorded from various musical instruments will be evaluated Finally, real time interval detection and display will be evaluated.

V. SCHEDULE

Figure 2 indicates the schedule that we plan to observe in order to complete the project by the designated date.

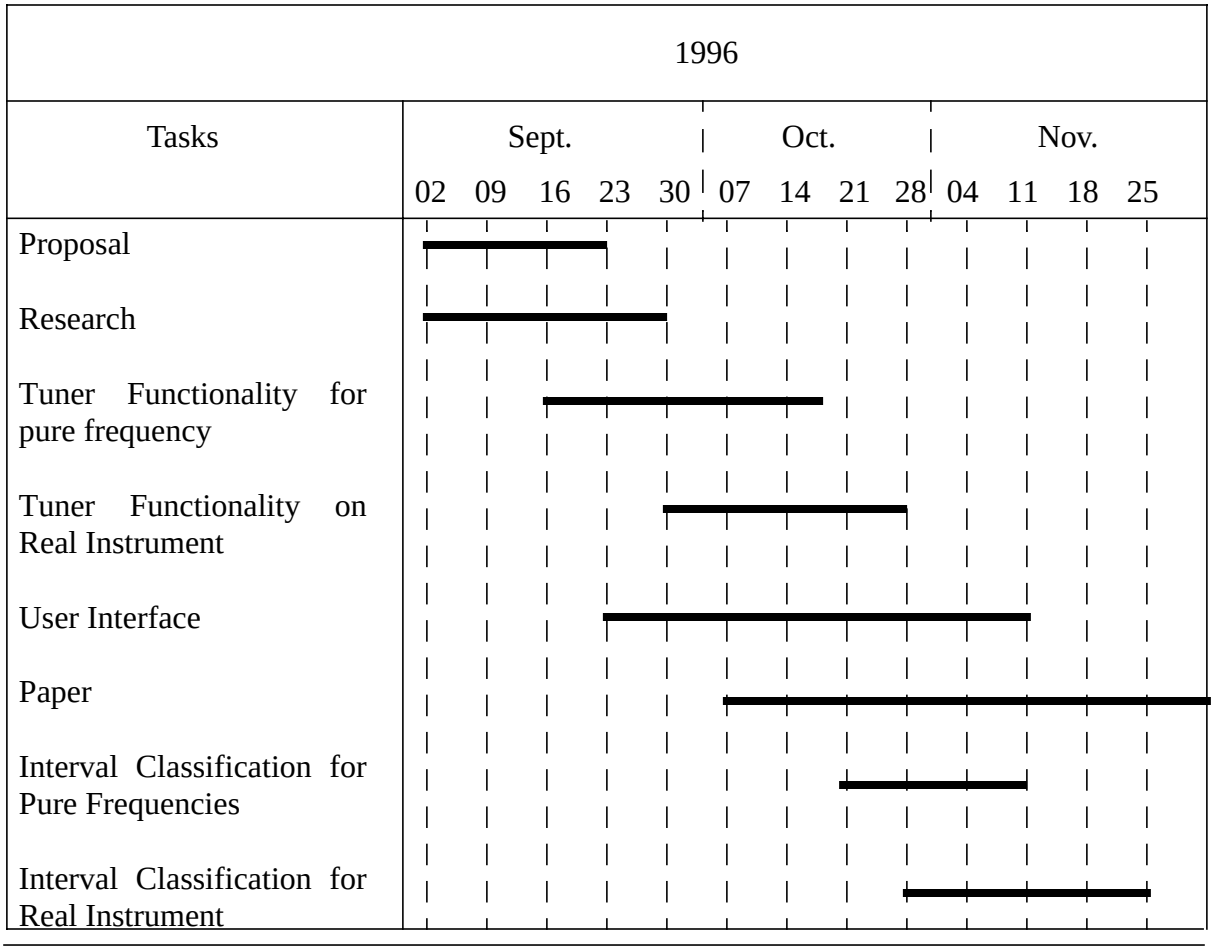


Figure 1: A time-line displaying the projected schedule for the Audible Frequency Detection and Classification software.

VI. 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*. Hasbrouck Heights, N.J.: Hayden Books, 1985, pp. 313-15.
4. W. W. Smith and J. M. Smith, *Handbook of Real Time Fast Fourier Transforms*. New York: Institute of Electrical and Electronic Engineers, Inc., 1995.
5. R. E. Ziemer, W. H. Tranter, and D. R. Fannin, *Signals and Systems: Continuous and Discrete, 3rd Edition*. New York: MacMillan, 1993, pp. 486-540.
6. M. B. Priestley, *Spectral Analysis and Time Series, Vol. 1*. London: Academic Press, 1981.
7. S. L. Marple, Jr., *Digital Spectral Analysis With Applications*, Englewood Cliffs, N.J.: Prentice Hall, 1987, pp. 303-349.
8. A. H. Benade, *Horns, Strings, and Harmony.* Garden City, NY: Doubleday, 1960.

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

proposal for

An Algorithm To Determine The Scenic Quality Of Images

submitted to fulfill the semester the project requirement for

EE 4773/6773: Digital Signal Processing

September 23, 1996

submitted to:

Dr. Joseph Picone

Department of Electrical and Computer Engineering
413 Simrall, Hardy Rd.
Mississippi State University
Box 9571
MS State, MS 39762

submitted by

Nirmala Kalidindi, Yaqin Hong, Zheng Liang

Image Processing Group
Department of Electrical and Computer Engineering
Mississippi State University
Box 9571
434 Simrall, Hardy Rd.
Mississippi State, Mississippi 39762
Tel: 601-325-3149
Fax-601-325-3149
email: kaldindi@isip, yh4@ra,zl3@ra

I. ABSTRACT

This project determines the scenic quality of the images.The images are given by the forestry department taken over various seasons and also over different vegetation. The goal will be accomplished by taking into consideration the various factors given by the statistical models. These statistical models were constructed from interviewing various visitors which consisted of people from different walks of life.The statistical models tell us the parameters that are sensitive to visual preference. [9] Some of the parameters to be considered are color, treatment of the ground, size of the trees etc. Also, forest scenery that was undisturbed, containing variety of natural features was preferred.We will attempt to develop an algorithm to extract the various features of the image and analyse the scenic quality.The algorithm determines the scenic quality on a scale of 0 to 1.'1'indicates a high quality image whereas '0'indicates a lower quality one.This project is for the forest dept which will be useful to them for the cutting trees as they prefer to cut the forest in such a way so as to preserve the scenic content.[4]

The evaluation part consists of running the algorithms on the images available in the database and confirming the results with the statistical SBE ratings given.The user can enter the image and the histogram, output of the edge detection algorithm and the frequency response computed will be displayed. Also a bar showing the scenic quality of the image from '0' to '1' will be displayed.

II. INTRODUCTION

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. [8]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. Limited amount of downed wood enco- untered was positively associated with scenic beauty while the statistics show a decline in perceived scenic beauty during the summer season. Visual penetration was lower and foliage twig screening was higher in low elevation landform positions compared with high elevation landform positions.The results also showed that perceived scenic beauty increases with the level of hardwood retention and the summer, fall, and spring views were preferred over those taken during the winters. Various algorithms have to be developed, considering all the above factors to determine the scenic quality of the image[3].

Season of the year: Color is one of the most 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 of season is a factor in determining the scenic quality of an image.

Segmentation: segmentation process involves partitioning an image into disjoint regions.A region is a connected set of pixels.It is of importance in computing the number of vertical lines in an image The information such as number of vertical lines is useful as it affects the scenic rating of the image.Those images which are pervasive through the forest i.e which gives a clear pic-

ture rather than blocking the image are considered as scenic.

Frequency Response: Fourier transform is done to compute the frequency response characteristics of the image. The fourier transform is a linear integral transformation that establishes a unique correspondence between a function of time and a function of frequency.Since in images processing the inputs and outputs are two dimensional.2D fourier transforms will be employed.

III. Project Summary

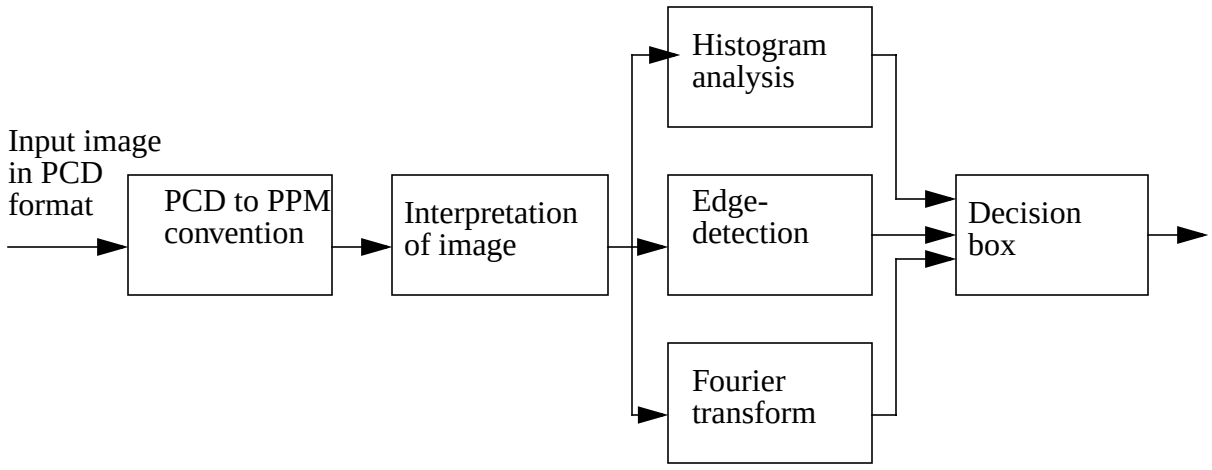


Figure I. System Architecture

The purpose of this project is to develop an algorithm to determine the scenic quality of the image. Figure I shows the architecture of the system, where the input is an PCD format image, the output will be decision represented by “0” or “1” to indicate the degree of scenic quality. To extract important features from image data which are given in PPM format, a description, interpretation or understanding of the scene is necessary.

The scenic beauty estimation is used to measure the visual preference for forested environments and to identify the need for measures that are sensitive to scenic beauty. Initially the images are given in PhotoCD format.As the format of the photoCD is not publicized it id required that the images should be converted to a suitable format for further processing. We have chosen Portable Pixel Map (PPM) for our project. The PCD format can be converted to PPM format using public domain software **hpcdtoppm**.The format of PPM has a header which consists of a magic number, width of the image, height of the image and the maximum level of colors the image can have. The image analysis basically involves feature extraction, segmentation and classification techniques. Individual colors form each pixel can be extracted by software. For the analysis of color histograms can be constructed for individual colors.Segmentation involves separation of different

objects by extracting their boundaries. Edge detection algorithms can be used to detect the number of vertical lines in an image.This can be used to find out the number of tall trees in the image.The Fourier transform of the image is done to compute the frequency response of the image. The outputs of all these algorithms should be fed to an decision box or image understanding system.This can be a neural network which should be trained by the images initially according to their standard beauty ratings.

1. Color features:

The simplest and most useful features of an object is its color[10]. Color variations are one of the most notable changes in forest vegetation as the season progresses. The individual color information i.e the amount of red, green and blue in each pixel is extracted. Each pixel is represented by 24 bits. 8 bits each for red, green and blue colors.The color analysis can be done by constructing a **histogram** to determine the content of various colors in 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. This is then used to construct a suitable histogram. Some of the factors of the image like the naturalness of the image can be determined from 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. [1]

2. Edge detection:

Edge detection is an important part in image analysis. Edges characterize object boundaries and are therefore useful for segmentation, registration and identification of objects in scenes. Edge points can be thought of as pixel locations of abrupt gray-level change.We therefore convert the color image to black and white image by RGB - to - YIQ mapping. The RGB - to - YIQ mapping is defined as follows: [2]

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

The Y component of YIQ is the luminance. The chromaticity is encoded in I and Q. For a continuous image f(x,y) its derivatives assumes a local maximum in the direction of the edge. Edge is defined as the boundary between two regions with relatively distinct gray-level properties and the transitions between two regions can be determined on the basis of gray-level discontinuities.The magnitude of the first derivative can be used to detect the presence of an edge, while the sign of the second derivative cab be used to determine whether an edge pixel lies on the dark or light side

of an edge. [6]The sign of the second derivative is positive for pixels lying on the dark side of both the leading and trailing edges of the object, while the sign is negative for pixels on the light side of these edges. Therefore, one edge detection technique is to measure the gradient of f along r on a direction θ, that is [5],

$$\frac{\partial f}{\partial r} = \frac{\partial f}{\partial x} \bullet \frac{\partial n}{\partial r} + \frac{\partial f}{\partial y} \bullet \frac{\partial y}{\partial r} = fx\cos\theta + fy\sin\theta$$

Hence the two principal properties used for establishing similarity of edge pixels are the strength of the response of the gradient operator used to produce the edge pixel and the direction of the gradient. A point in the predefined neighborhood of (x, y) is linked to the pixel at (x, y) if both magnitude and direction are satisfied. This process is repeated for every location in the image. A record must be kept of linked points as the center of the neighborhood is moved from pixel to pixel.[7]

Boundaries are linked edges that characterize the shape of an object and they are useful in comp utation of geometry features such as size or orientation.Edge detection algorithms are typically followed by linking and other boundary detection procedures designed to assemble edge pixels into meaningful boundaries.

4. 2D FFTs

The fourier transform is used to compute the frequency response characteristics of the image. The neural network models using the Fourier Transform can be used for extraction and recognition stage. Artificial Neural Networks are capable of learning and making decisions. The two dimensional Fourier Transform distribution has some interesting properties which are used for extracting invariant features of the image.They are:

(1). The magnitude of the Fourier Transform is shift invariant.

(2). Higher input spatial frequencies correspond to high amplitude values further from the origin of the FT plane and

(3). As the input rotates the distribution in the FT plane also rotates.

Images are inherently two dimensional. The two dimensional discrete Fourier transform is given as

$$F(u,v) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)e^{-2\pi j\left(\frac{mu}{M}+\frac{nv}{N}\right)}$$

for u = 0,1,2,...M-1 and v = 0,1,2... N-1

The inverse Discrete Fourier transform is given by

$$f(m,n) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{\left[2\pi j\left(\frac{mu}{M}+\frac{nv}{N}\right)\right]}$$

for m = 0,1,2.... M-1 and n = 0,1,2...N-1.

The double summation can be written in the matrix form as

F = P f Q

Where P and Q are non -singular, matrices of the size M x M and N x N respectively, f represents the input image matrix and the elements of matrices P and Q are given by

$$q_{vn} = \frac{1}{Ne}^{\left(\frac{-2\pi jnv}{N}\right)}$$

where v = 0,1,2,...N-1 and n = 0,1,2,... N-1

$$p_{um} = \frac{1}{M} e^{\left(-\frac{2\pi jmu}{M}\right)}$$

where u = 0,1,2.....M-1 and m = 0,1,...M-1.

IV. EVALUATION

In order to evaluate the algorithm, the data provided by the forestry department of various images will be used. The data also has different versions of the same image taken during different seasons of the year.The algorithm developed will be applied to extract the various features of the image. We will attempt to demonstrate some of these features by computing histograms, passing it through an edge detection algorithm and computing the fourier transform.

The histogram of an image normally refers to the intensity values of the pixel.We will show the histogram for the individual colors of red, green and blue.We will also show the frequency response characteristics of the image by computing the fourier transform of the image.The image will be given to the edge detection algorithm and the characteristics of the image like the number of vertical lines will be computed.The database has total of about 600 images. Some of the images i.e about 20% can be used for training the neural network and the remaining can be used fro testing the algorithms.The neural network will be trained according to the SBE ratings already available.The scenic quality of the image will be shown on a scale of '0' to '1'.'0' indicating a low scenic and '1' indicating a high scenic image.It will be compared with the SBE ratings already available to confirm the validation of the algorithm.

V. SCHEDULE:

Table 1:

TASK	WEEK
	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Proposal	———
Feature extraction software	———————
Algorithm development	———————————
Testing and analysis	———————
Project report and presentation	———————

VI. REFERENCES

[1] Anil K. Jain, *Fundamentals of Digital Image Processing*, Prentice Hall, Englewood Cliffs, NJ, USA, 1989.

[2] Foley, Van Dam, Feiner, Hughes. *Computer Graphics: Principles and Practice*, Addison - Wesley Publishing Company, Inc, Menlo Park, California, USA, July, 1995

[3] Victor A. Rudis, James H. Gramann, Edward J. Ruddell, Joanne M. Westphal, "*Forest Inventory and Management-Based Visual Preference Models Of Southern Pine Stands*", Forest Science, vol 34, No 4, pp. 846-863, December 1988.

[4] Herrick, Thereasa A., Rudis, Victor A. "*Visitor Preference for Forest Scenery In The Ouachita National Forest*", Paper presented at the Symposium on Ecosystem Management Research in the Ouachita Mountains: Pretreatment Conditions and Preliminary Findings, Hot Springs, AR, October 26-27, 1993.

[5] Rafael C. Gonzalez, Paul Wintz, *Digital Image Processing*, Addison-Wesley Publishing Company, California, USA, 1987

[6] Rama Chellappa, Alexander A.Sawchuk, "*Tutorial Digital Image Processing and Analysis* Vol. 2, Computer Society Press, Washington D.C, 1985

[7] Kenneth R. Castleman, *Digital Image Processing*, Prentice Hall, Inc. Englewood Cliff, NJ, USA, 1979

[8] Rudis. Victor A, Gramann. James H Herrick, Theresa A, "*Esthetics Evaluation*", Paper Presented at the Proceedings of the Symposium on Ecosystem Management Research in the Ouachita Mountains: Pretreatment Conditions and preliminary Findings; 1993 October 26-27, pp 202-211.

[9] Gramann. James H, Rudis Victor A, "*Effects of Hardwood Retention, Season of Year, and Landform on the Perceived Scenic Beauty of Forest Plots in the Ouachita Mountains*", Paper Presented at the Proceedings of the Symposium on Ecosystem Management Research in the Ouachita Mountains: Pretreatment Conditions and preliminary findings; 1993 October 26-27 pp 223-228.

[10] Wiijoo Yhang, "*The Effect of Color on the Perceived Scenic Beauty of Pine-Oak Plots in the Ouachita National Forest, ARKANSAS*", Dissertation submitted to the office of Graduate Studies of Texas A & M University.