Spring 1999

Implementation and performance study of image data hiding/ watermarking schemes

Arkadiusz Edward Komenda
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

IMPLEMENTATION AND PERFORMANCE STUDY OF IMAGE DATA HIDING / WATERMARKING SCHEMES

by
Arkadiusz Edward Komenda

Two data hiding / watermarking techniques for grayscale and color images are presented. One of them is DCT based, another uses DFT to embed data. Both methods were implemented in software utilizing C/C++. The complete listings of these programs are included. A comprehensive reliability analysis was performed on both schemes, subjecting watermarked images to JPEG, SPIHT and MPEG-2 compressions. In addition, the pictures were examined by exposing them to common signal processing operations such as image resizing, rotation, histogram equalization and stretching, random, uniform and Gaussian noise addition, brightness and contrast variations, gamma correction, image sharpening and softening, edge enhanccement, manipulation of a channel bit number and others. Methods were compared to each other. It has been shown that the DCT method is more robust and, hence, suitable for watermarking purposes. The DFT scheme exhibits less robustness, but due to its higher capacity is perfect for data hiding purposes.

IMPLEMENTATION AND PERFORMANCE STUDY OF IMAGE DATA HIDING / WATERMARKING SCHEMES

by
Arkadiusz Edward Komenda

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering

Department of Electrical and Computer Engineering

May 1999

APPROVAL PAGE

IMPLEMENTATION AND PERFORMANCE STUDY OF IMAGE DATA HIDING / WATERMARKING SCHEMES

Arkadiusz Edward Komenda

Dr. Ah N. Akansu, Thesis Advisor — Date
Professor of Electrical and Computer Engineering, NJIT

Dr. Yuguang Fang, Committee Member — Date
Assistant Professor of Electrical and Computer Engineering, NJIT

Dr. Richard A. Haddad, Committee Member — Date
Professor of Electrical and Computer Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Arkadiusz Edward Komenda

Degree: Master of Science in Electrical Engineering

Date: May 1999

Undergraduate and Graduate Education:

- Master of Science in Electrical Engineering,
  Department of Electrical and Computer Engineering,
  New Jersey Institute of Technology, Newark, NJ, May 1999

- Bachelor of Science in Electrical Engineering,
  Department of Electrical and Computer Engineering,
  New Jersey Institute of Technology, Newark, NJ, May 1998

Major: Electrical Engineering

To my beloved parents Jan and Grazyna,
my dearest grandparents Edward and Zofia Kaca, and
Waclaw and Maria Komenda
I love all of you very much

Dla moich ukochanych rodzicow Jana i Grazyny
oraz
kochanych dziadkow Edwarda i Zofii Kaca
i
Waclawa i Marii Komenda
Bardzo was wszystkich kocham

ACKNOWLEDGMENT

I would like to thank my advisor, Dr. Ali N. Akansu, for giving me the opportunity to work on such an interesting project, and helping me through guidance and strong support to complete this work in such a short time. Moreover, I found his insights into other aspects of life eye-opening and inspiring. I would not have been able to do it without his help.

Many thanks to Dr. Richard A. Haddad, who I met a few years ago when he came to NJIT. It is a pity that we only recently met in class as an instructor and a student. He is the best professor I have had. The students really appreciate his good teaching.

Special thanks to Dr. Yuguang Fang for persistent questioning of speakers during the seminars and actively participating in my committee. Those who ask don't wander aimlessly.

I would like to thank Dr. Alex Gelman and his crew at Panasonic Research Institute, Plainsboro, N.J. for their support and fruitful discussions.

Finally, I would like to thank my friend, Mahalingam Ramkumar, who introduced me to data hiding and image processing, provided me with guidance and help throughout the entire year and always found time to answer my never ending questions. He will make a great professor in the future.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

Figure **Page**

Figure **Page**

CHAPTER 1

INTRODUCTION

Data hiding sometimes also called *steganography* is a relatively new and quickly growing field in digital image processing. The main task is to design a scheme which effectively hides data in images without downgrading their quality [1, 2, 3]. Moreover, the hidden data should not be easily retrievable by just anyone. The method should be complex enough to enable only authorized and debriefed individuals to extract the hidden information. At the same time, hidden data should not be changed or completely lost due to the common image processing operations such as compression, image enhancement, filtering, etc. However, it is practically impossible to design a method that would guarantee safety and correct detection of data subjected to every possible image alteration. Therefore, as in the case of all new fields of application, we aimed to design a steganographic scheme that shows itself to be robust, at least, against to the most commonly encountered image processing operations.

1.1 Data Hiding Applications

Applications for data hiding in images are enormous, (Table 1.1). The most obvious one is *cataloging* or indexing the images. The hidden information can contain the type of the picture (cartoon, landscape, in-door, etc.), author's name, owner's name, date and location it was taken, and so on. Steganography can also be used to prove *image ownership* in the court of law. For example, the professional photographer takes a picture in the digital form and posts it on its website for viewing by the potential customers. If one of the customers wants to purchase the picture, he pays the photographer a negotiated price and now is free to download the image for commercial use. The website image contains a watermark which in case of a copyright infringement can be used by the photographer to prove that he's the actual owner.

Table 1.1 Data Hiding Applications

Data Hiding Applications
Cataloging
Image Ownership
Watermarking
Intelligent Agents

This scheme should protect the rightful owners from unauthorized use of their image content.

This example brings us to the next application, namely *watermarking.* Watermarking is a way of tagging an image for authenticity, copyright [4] or identification purposes. It has a lot in common with data hiding; however, it is also subject to attacks. Attacks in this instance encompass counterfeiting, altering and destroying the watermark. Typical hidden data which is not a watermark might be exposed to such attacks too; however, it is less likely, since changing the data might not bring any financial gain or other benefits. This distinction is based on the assumption that since the particular hidden data is not a watermark the ownership rights cannot be resolved on its bases. Since the watermarks are much more likely to be attacked, their embedding methods should be more complex and their survival rate under the attack much higher than a typical data hiding application. Most common watermark alteration attempts include compression, image reshaping, filtering, etc.

The fourth area of application for data hiding is of *intelligent agents.* In this scenario, images are carriers of small executive programs (intelligent agents) that perform specific tasks. For example, while displaying an image, the agent checks whether the computer performing the operation is the one that belongs to the person licensed to use the image. If not, it stop displaying operation and notifies the user that he is not authorized to use the image. Another very appealing application for intelligent agents is connected to e-commerce and Internet. For instance, a customer would like to watch a movie that can be downloaded from a pay-per-view website.

First, he pays for the movie, second downloads it, and then watches. Once the movie is finished the intelligent agent hidden inside the computer video file, e.g. MPEG, self-destructs the movie sequence. Since the viewer paid for only one viewing of the movie, he is no longer authorized to have a viewable copy of it.

1.2 The Scope of Thesis

This work presents two data hiding schemes complete with their implementation and performance analysis. The first scheme embeds data in low frequency image coefficients in the DCT domain [5]. The detection of data is a non-linear, oblivious operation. The second scheme uses maximally separable points of the signal constellation for steganography in combination with FFT based cyclic correlation for the detection operation [6, 7]. This method likewise does not require the original image to be present during the detection stage. Both methods were implemented in C/C++ and program listings can be found in the appendices. However, the main objective of this thesis is to present a thorough performance study of watermark and hidden data survival rates under various image signal processing operations for these two data hiding techniques. Watermarked pictures are exposed to various common image processing operations such as JPEG and SPIHT compressions, brightness and contrast manipulations, gamma correction, resizing, rotation, noise addition, to name a few. All testing is performed on standard test images.

CHAPTER 2

OVERVIEW OF DATA HIDING TECHNIQUES FOR IMAGES

Over the years, there have been developed many techniques for hiding digital data inside images. The first differentiation among them is based on data detection procedure (Table 2.1). While extracting data or a watermark, the user might be asked to provide an original, unwatermarked image which is necessary for the watermark retrieval operation. Such scheme [8, 9] is categorized as a *cover image escrow* method. On the other hand, the steganographic algorithm where the original image is not needed [10, 11] for a successful watermark extraction is called an *oblivious detection* scheme.

Table 2.1 Hidden Data Retrieval Methods

Hidden Data Retrieval Methods
Cover Image Escrow
Oblivious Detection

2.1 Classification of Data Hiding Techniques

Steganographic schemes can be divided into the following classes and subclasses:

- Spatial domain signature embedding
- Transform domain signature embedding techniques
  - Discrete Cosine Transform (DCT)
  - Wavelet transforms
  - Other transforms
- Other approaches

2.1.1 Spatial Domain Signature Embedding

The first data hiding methods were designed to operate in the spatial domain (without decomposition). They were relying on modification of certain pixels' bits, usually the least significant bits (LSB), to hide data. In Ref. [12], the authors present two schemes to hide data in images. The first one replaces the least significant bits of the image with an m-sequence while the second adds the m-sequence to the least significant bits of the image and uses auto-correlation to detect it later on. In Ref. [14], the author alters the LSB of sets of pixels selected by a random key in such a way to increase their contrast. Famous "Patchwork" operates on pairs of pixels, increasing the value of one and decreasing the value of the other one, in order to hide a bit [13]. However, these methods are not too resistant to common image processing operations, such as compression, addition of noise, and others.

2.1.2 Discrete Cosine Transform Based Techniques

The most popular approach to image steganography to date is through the use of the DCT. The original image is transformed into the DCT domain in which the actual data hiding takes place [5]. In Ref. [2], Cox *et. al.* apply a steganographic method to both a video and audio signals. The signature is inserted into still pictures using discrete cosine transform techniques. It is shown that the watermark can still be recovered after various common processing operations such as scaling, JPEG, rotation, translation and clipping. Even when the image is printed, photocopied and scanned, a signature is still recoverable. In Ref. [15] the authors use human visual system's (HVS) characteristics to ensure that the watermark is invisible to the human eye. The watermark's signature, added in the DCT domain, is of the pseudo-noise form shaped by using HVS's frequency masking. Ruanaidh *et. al.* in Ref. [16] present a watermark embedding scheme in the DCT domain by modulating the transform coefficients with a bi-directional coding. In Ref. [17] the authors hide

a signature inside the middle frequency DCT coefficients. This scheme is resistant to JPEG compression and resizing.

2.1.3 Wavelet Transform Based Solutions

More and more data hiding algorithms incorporate wavelet transforms. Inoue *et. al.* in Ref. [18] propose a discrete wavelet transform based method for embedding a digital watermark inside an image. Data is hidden in the lowest frequency components of the picture by using a controlled quantization process. Detection is done in an oblivious mode by quantization and mean amplitude of the lowest frequency components. In Ref. [19], the authors present two wavelet based techniques for steganography. They divide wavelet coefficients into significant and insignificant ones. In the first method, they hide data utilizing the insignificant coefficients. While in the second scheme, they introduce a threshold and modify significant coefficients for data hiding purposes. This method is shown to be robust to JPEG compression. Swanson *et. al.* in Ref. [20] present a video watermarking scheme in which a perceptual model determines where the strength of the watermark can be enhanced within the image. The method performs well even in the event of frame skipping.

2.1.4 Other Transforms

Another popular transform for data hiding is the Discrete Fourier Transform (DFT). Real images generally give a complex valued coefficients after the application of the DFT. Ramkumar *et. al.* in Ref. [7] use only the DFT magnitudes for data hiding purposes. However, Ruanaidh *et. al.* in Ref. [21] propose hiding data in the phase of the DFT coefficients. In Ref. [16], Hadamard Transform and Daubechies Wavelet Transform are utilized.

2.1.5 Other Approaches

A more novel approach has been presented by Voyatzis and Pitas [22, 23]. In this case, the watermark is an image consisting of black and white pixels only. First, the watermark is scrambled by chaotic transformation using toral automorphism. Second, the scrambled watermark is combined with the original picture by changing the gray levels to form a watermarked image. The mark is said to be resistant to filtering and JPEG compression. Interestingly, Davern and Scott [24] use fractal compression scheme for data hiding.

2.2 Summary

As we can see there are many various approaches to effective information hiding. Each presents us with different benefits and immunities to certain signal processing operations. However, no one has yet came up with the scheme which would be totally resistant to tampering. In this thesis it is aimed to come closer to the ideal fully-robust data hiding/watermarking method.

CHAPTER 3

TWO NEW ROBUST DATA HIDING / WATERMARKING METHODS

The following chapter introduces two steganographic techniques proposed in Ref. [5] and [6, 7]. They were both implemented in C/C++ and multiple experiments were conducted on watermarked images produced by those programs. The objective was to analyze the performance and resistance of the watermarks and hidden data to common signal processing attacks.

3.1 Method I: DCT Based Robust Watermarking Technique

The first data hiding method utilizes DCT transformation and its detection procedure is oblivious in nature [5]. In a typical scenario, an 8x8 block DCT is performed on the original image. Each block is then zigzag scanned so as to form 64 bands of coefficients. The first 8 bands of coefficients with the highest energy levels except the DC coefficient band are joined together to form a row vector. This vector is then randomly mixed, i.e. the coefficients are moved from one position to another in a random fashion. However, their original order is saved for later use in the reverse reordering operation. Next, the mixed coefficient vector is subjected to an all-pass filter scrambling. After all these operations, the vector is divided into segments with 128 coefficients in each. In each such a segment one bit of data is embedded. During the embedding small magnitude coefficients are manipulated as to have correct signs. For example, if we want to embed a '0' bit, we have to end up with the majority of coefficients in the segment being negative. This is achieved by reversing the positive sign of all small magnitude coefficients. To hide a '1' bit, the majority of the segment's coefficients should be positive, so we reverse signs of all the negative small magnitude coefficients (Figure 3.1).

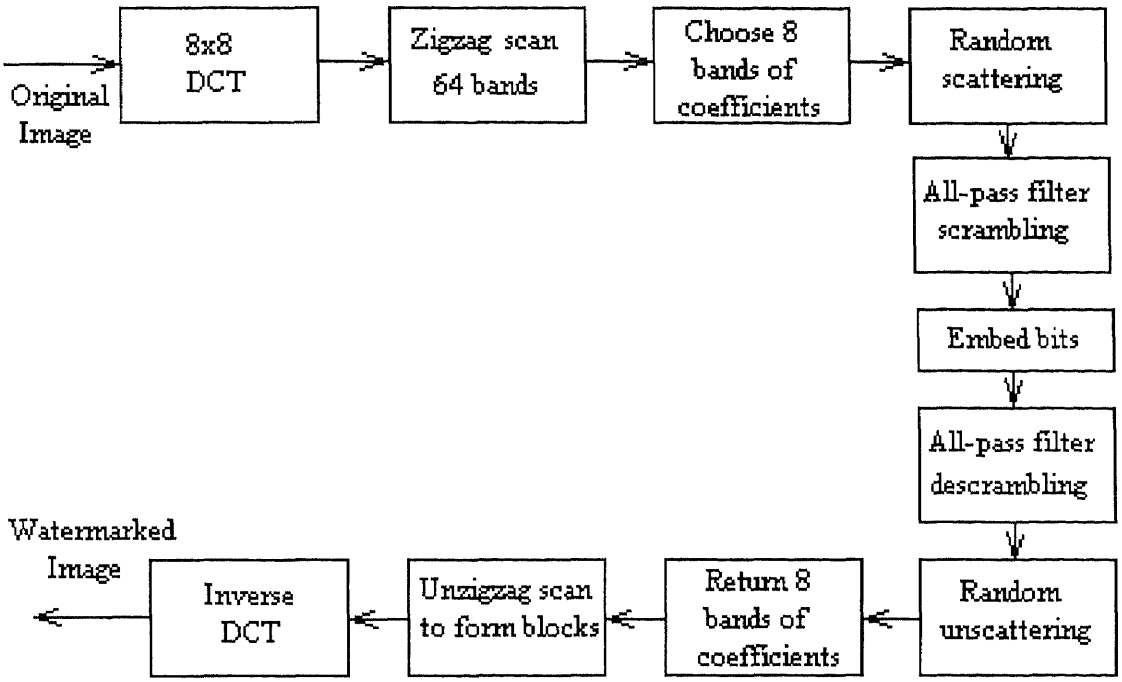

Figure 3.1 Method I - Data Hiding Scheme

After the embedding, the segments are put together to form a vector again. This vector is then inverse all-pass filtered and reordered. The new 2 through 9 bands together with the rest of the other bands from the original image are unzigzag scanned to form 8x8 blocks which are subjected to the inverse DCT. In this way a watermarked image is obtained.

Detection proceeds along the lines of the procedure described in the first paragraph of this section. However, instead of embedding, the bits are detected on the bases of the dominant sign in the segment. If in a 128 coefficient segment most of the coefficients are negative, then it is a '0' bit, otherwise it is a '1' bit (Figure 3.2).

3.2 Method II: Novel DFT Based Data Hiding Technique

As in the previous method, the detection step of the DFT based scheme does not need the original, unwatermarked image to be present for successful data extraction [6, 7].

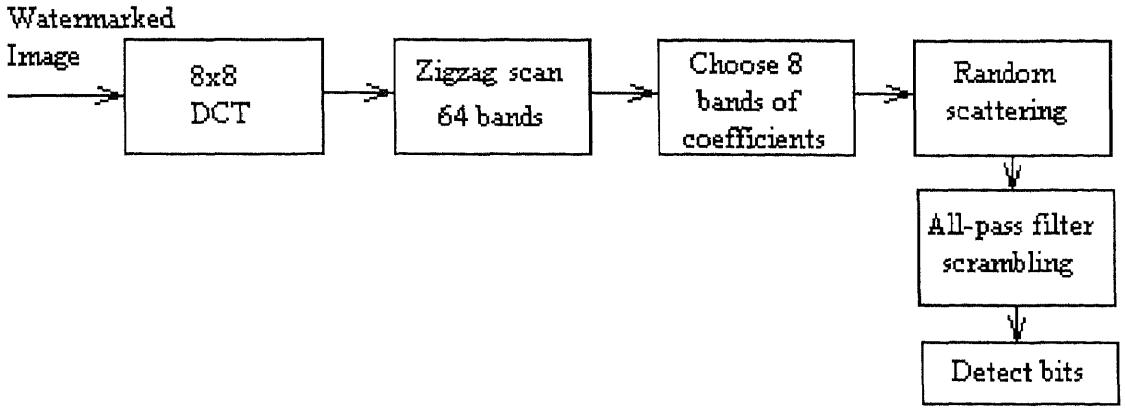

Figure 3.2 Method I - Data Detection Procedure

During the data hiding operation, the original picture undergoes 8x8 block DFT transformation. From each block 34 unique coefficients are extracted and divided into 34 bands. The end result is a 34 by number-of-8x8-blocks matrix. This matrix is column-wise decorrelated using a KLT (Karhunen-Loeve Transform) matrix and then row-wise decorrelated using Daubechies filter bank. In the next step, the matrix is reshaped to form a single vector. The vector is randomly reordered (saving the original coefficient ordering for later reverse reordering operation) and divided into segments of length 2048 coefficients. In each segment, one 12 bit data-word is hidden. The data hiding procedure uses a seed to generate a 2048 long binary sequence, which is all-pass filtered. The 12 bit data-word is converted into its integer equivalent and the binary sequence is then:

- cyclically shifted by the integer's value, if the integer is less than 2048,
- or multiplied by a negative sign and cyclically shifted by the *4095 - integer's value* amount, if the integer is greater than or equal to 2048.

After shifting, the new binary sequence is combined with the segment's coefficients (Figure 3.3).

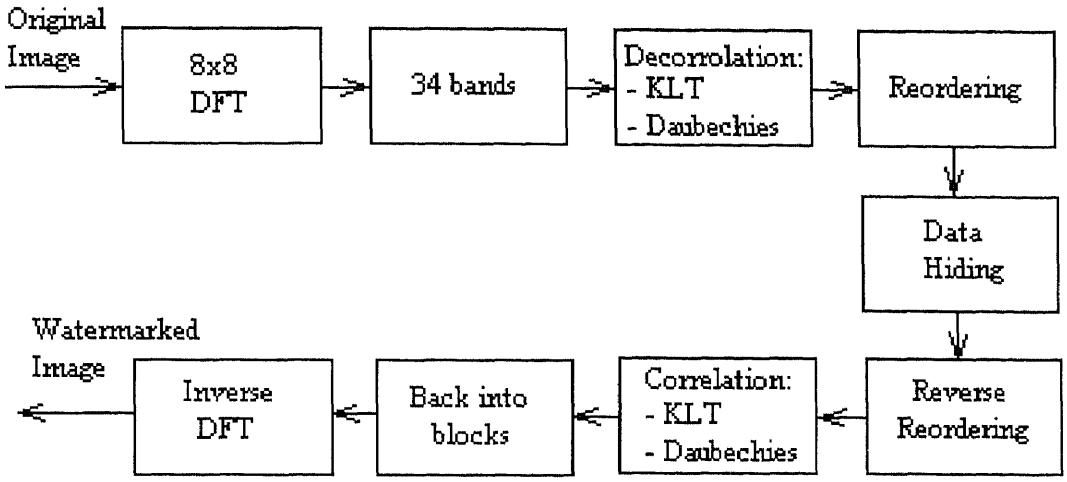

Figure 3.3 Method II - Data Hiding Scheme

All segments are combined back into a vector which is then reverse reordered and reshaped into a matrix. The matrix is correlated row-wise using Daubechies filter bank and column-wise by KLT. Finally, 8x8 blocks are formed and the inverse DFT performed on them to generate the watermarked image.

The detection part (Figure 3.4) goes along the lines of first paragraph in this section. However, only 6 least significant bits of each of the coefficients in the segment are used for data retrieval. The all-passed binary sequence is created using the same seed (the seed is an input to be given by the user, or, rather, owner of the image) as during the data hiding operation. Then the cyclic correlation of the binary sequence is performed on the coefficient segment. The amount of shift which gives the highest magnitude of correlation, is the hidden integer number. If the sign of the highest magnitude correlation is:

- positive, then the integer number (amount of shift) converted into a 12 bit data-word is the hidden data,

- negative, then the result of *4095 - integer's value* converted into a 12 bit data-word gives the hidden data.

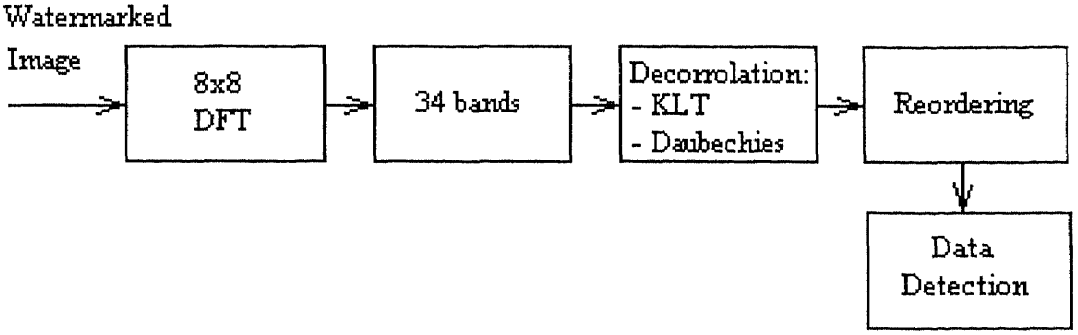

Figure 3.4 Method II - Data Detection Procedure

CHAPTER 4

SOFTWARE IMPLEMENTATION OF BOTH METHODS

Fully functional computer packages for both data hiding algorithms were built so as to further test their robustness, conduct a thorough performance research and find a means for easy distribution to all interested parties in our current research. Software for DCT and DFT steganographic methods is based on a combination of C and C++ languages. However, DCT and DFT packages were written separately and for different systems. Hence, their inner code does not resemble each other. They also function differently and, as will be shown, have different interfaces, although, some parallels can be drawn between them.

4.1 Method I - DCT Windows-based Software Package

Software package (*DHS1*) for the first watermarking method was written for Windows based systems. This includes:

- Windows 95,
- Windows 98,
- Windows NT 3.51 (Intel),
- Windows NT 4.0 (Intel).

DHS1 stands for *Data Hiding Software 1.* It is a fully functional, redistributable application software complete with the installation and setup routines (Figure 4.1). Version 1.10 of the software fits onto one 3.5" diskette with 1.44 MBytes of disk space. After installation, simple double-clicking on DHS1.EXE file in Windows Explorer runs the application (Figure 4.2).

Figure 4.1 DHS1 - Setup Splash Screen

4.1.1 DHS1: Usage

The program supports embedding data in both still images and sequences of images (video). To embed data in just one image choose **Image** from the main screen's *menu bar*, Figure 4.3; to hide data inside video frames select **Video** from the menu bar. Choosing **Image** or **Video** from the main menu bar leads us to an *operation submenu* (Figure 4.4) where choices are:

- **Hide**, for embedding data into an image,
- **Detect**, for extracting data from a picture.

Selecting **Hide** or **Detect** mode activates the *block size submenu* (Figure 4.5). The data hiding algorithm was slightly modified to allow various DCT block sizes to be used during the data hiding and extracting operation. It is up to the user to decide what block size option to use. In the next chapter, the usage of various block sizes and the corresponding data rate detection are investigated.

If we chose to hide data inside an image with the block size options of 8x8, 16x16 and 32x32 Figure 4.6 window will appear. Here we have an option of hiding data inside a black-and-white picture (grayscale) with a *pgm* format or a *ppm* color picture.

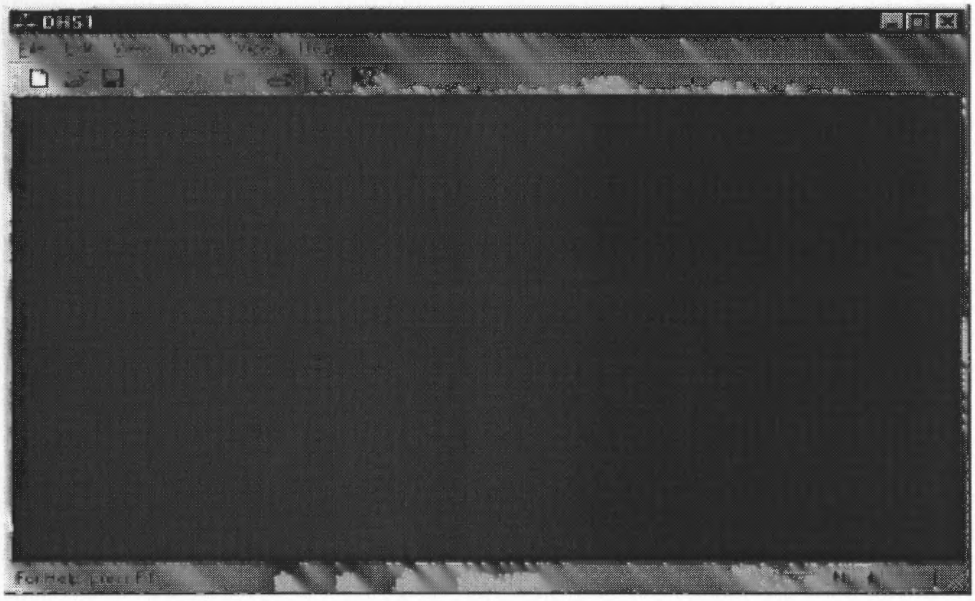

Figure 4.2 DHS1 - Main Screen

Figure 4.3 DHS1 - Main Screen Menu Bar

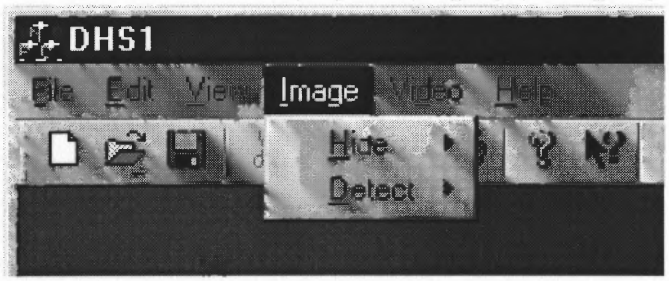

Figure 4.4 DHS1 - Image Operation Submenu

Figure 4.5 DHS1 - Block Size Submenu

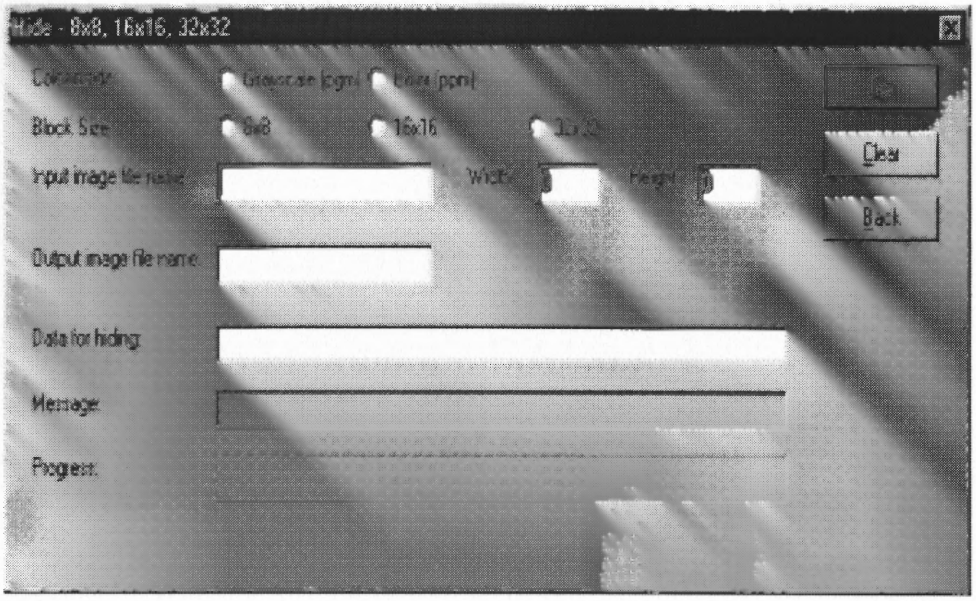

Figure 4.6 DHS1 - Data Hiding Inside an Image

A complete (with an extension) input file name of the original, unwatermarked image has to be provided as well as the output file name under which the watermarked image is to be saved. If the input file format is incorrect or the file does not exist an appropriate error message will be displayed in the *Message* box. Original image width and height has to be entered into the appropriate fields. Data to be hidden should be written inside the *Data for Hiding* box. There is no length restriction during entering the data. However, the program only hides the first n possible characters depending on the image size, which determines the image capacity for hiding. Note that the program will give an error if no data was provided for hiding. To proceed with data hiding operation press **Go**, to return to the main screen press **Back**. The progress of hiding can be followed on the *Progress* bar.

Detection operation follows similar steps. Press **Image** on the main screen *menu bar*, then select **Detect** and the appropriate block size option, in this case, **8x8, 16x16, 32x32**. The Figure 4.7 window will appear. Choose the file format, *pgm* or *ppm* and enter the name of the watermarked file (with the correct extension). Most importantly, select the block size which was used during the data hiding stage,

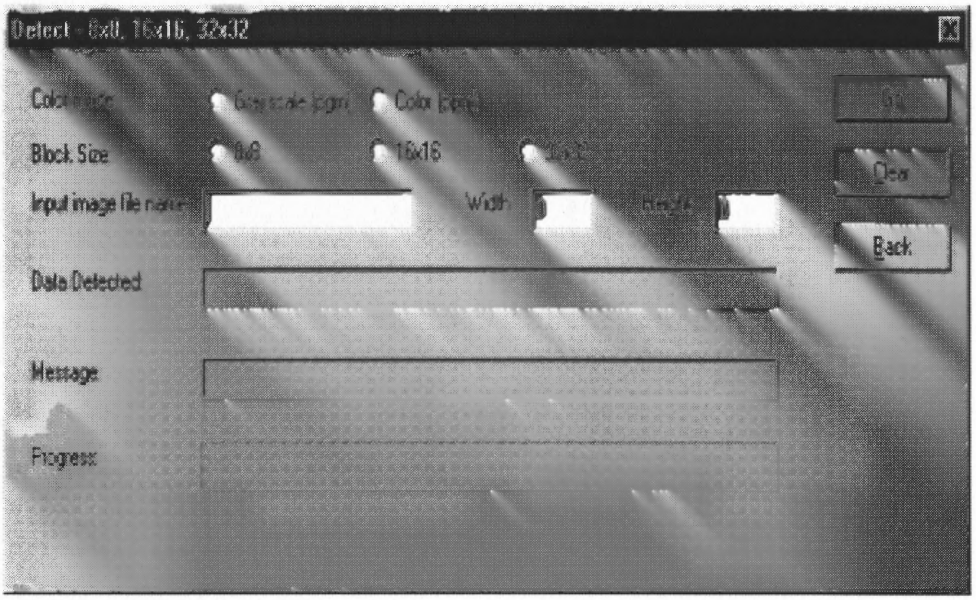

Figure 4.7 DHS1 - Data Detection from an Image

otherwise the detected data displayed in the *Data Detected* field will be incorrect. The correct width and height must also be entered in the assigned fields. Note that width and height for the watermarked image is the same as that of the original, unwatermarked image. To proceed with the detection press **Go**, to go back to the main screen click on **Back**. Detection progress can be followed on the *Progress* bar.

Hiding data in the video frames is somewhat similar. First we choose **Video** from the *menu bar*, then **Hide** and the block size option. Figure 4.8 window is displayed. File formats available for input are *pgm*, *ppm* and *.y.u.v*. In the last file format, one frame is stored in 3 separate files. The one with extension *.y* holds the luminance data, *.u* color difference between luminance and a red color, and *.v* color difference between luminance and a blue color. For *Common input file name* enter the name of the file with the changing frame numbers substituted by a percent sign, *%*, see Table 4.1. In the case of *.y.u.v* format, after the file name put *.y* extension, since the data is only being hidden in the luminance component. *Common output file name* is governed by the same rules as the input file names (see Table 4.1). Provide the image size in *Height* and *Width* fields. In *First file's number* and *Last file's*

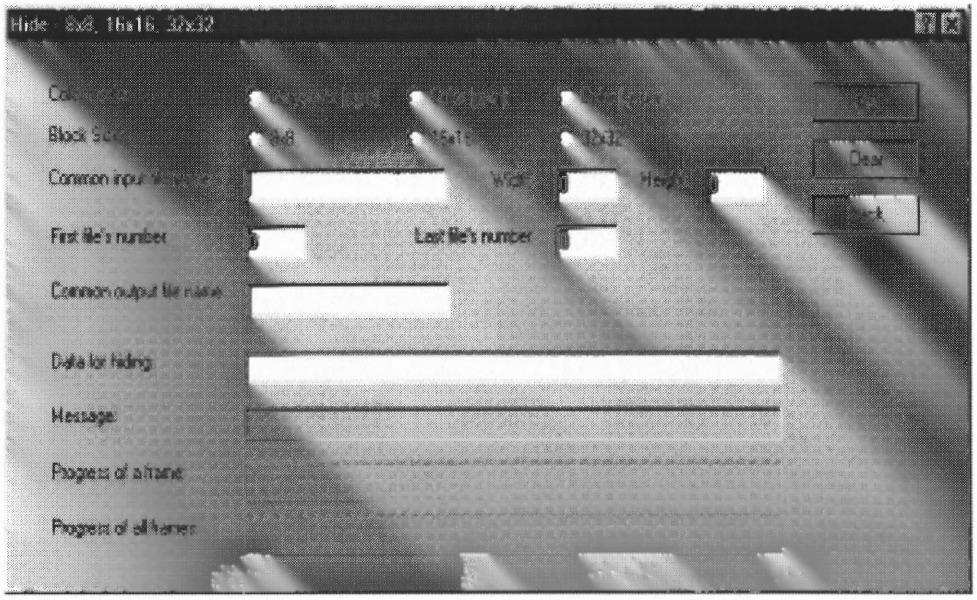

Figure 4.8 DHS1 - Data Hiding In Video

number do not enter the leading zeros if any. Data for hiding ought to be inserted into *Data for hiding* box. As in the single image case, there is no restriction on the length of inputed data, the program itself later determines how many characters it can hide in the provided video sequence. Data hiding progress of the current frame can be seen in *Progress of a frame* field. The progress of the whole movie can be seen on the *Progress of all frames* bar. To begin steganography click on **Go**, to clear all fields press **Clear**, to return to the main screen click on **Back**.

Table 4.1 Video Files Naming Convention

Files	**Naming Convention**
tp0000.pgm tp0001.pgm tp0002.pgm tp0003.pgm	tp%%%%.pgm
tp8.pgm tp9.pgm tp10.pgm tp11.pgm	tp%.pgm

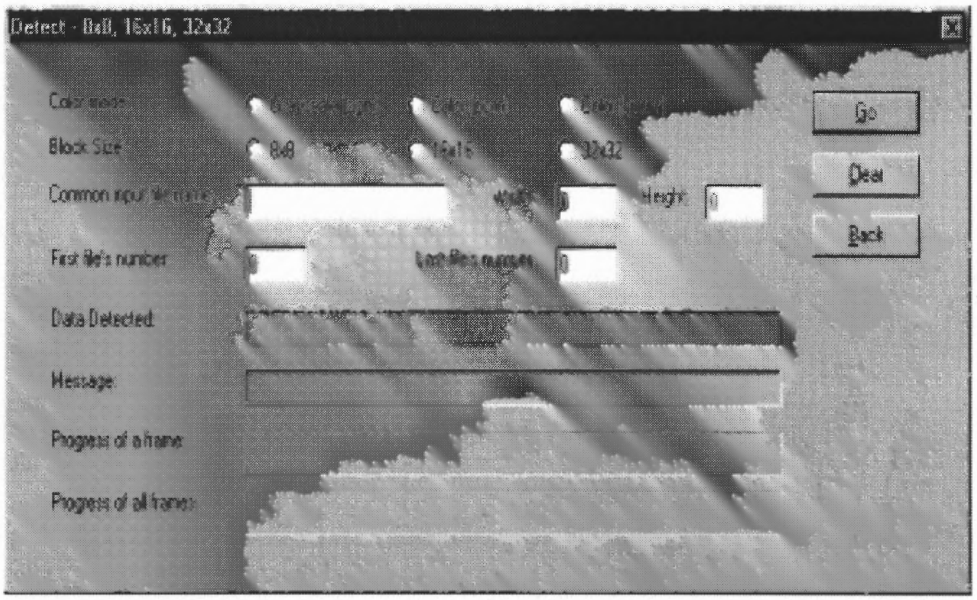

Figure 4.9 DHS1 - Data Detection In Video

Video detection screen Figure 4.9 can be reached by clicking on **Video**, then **Detect** and the block size option used while hiding data. All the fields follow the same rules as in the video data hiding window's case. A few points worth mentioning are:

- select the same block size as during the hiding operation,
- *Common input file name* string must be the same as the *Common output file name* used during the data embedding stage,
- *Width* and *Height* are the same as for the original data files,
- provide correct number of frames spanning between *First file's number* and *Last file's number* (do not put leading zeros).

The detected data will be displayed in the *Data Detected* field.

4.1.2 DHS1: Inside Structure and Workings

The whole DHS1 program comprises of many source files whose listings can be found in Appendix A. Before DHS1 was written in C/C++, a working copy was made in

Matlab. This way, it was possible to troubleshoot C/C++ code for logical errors if the results did not match with a Matlab program. Writing mathematical programs in Matlab is considerably faster than in C/C++.

The maximum height of the picture that the program can take as an input is 1000 pixels. There is no limit in the horizontal dimension. The assumption was that in practice pictures are usually wider than their vertical size. Hence, it is less likely that the picture will have 1000 pixels vertically than horizontally. The maximum height is controlled by the constant, *ISIZE*. The image is actually kept sideways in the array, that is, throughout the program operations are done on the transpose of the image. The reason is that only the number of rows of the array can be set dynamically in C++. The number of columns is a constant that can't be changed or manipulated from the inside of the program.

The program accepts *pgm*, *ppm* and *.y.u.v* (in the video case) file formats. These are pretty popular graphical formats and can be displayed by many commercially available graphics packages, for instance, Lview Pro and Paint Shop Pro. *.y* file format can be displayed as a RAW file. The reason for entering width and height of the images is that, first, *.y.u.v* files do not have size information embedded in themselves and, second, the program has been built in such a way that RAW file input format could be easily added. In addition, vertical and horizontal dimensions do not have to be powers of 2. The program is written to handle any size. This is done by using the maximum size of the image which is still divisible into an integer number of blocks of the current block size. For example, for the 8x8 block size option, for a 388x190 picture, the program only uses the 384x184 image portion.

For block sizes 8x8, 16x16, 32x32 and 64x64, 128x128 separate subroutines were written to speed up the execution of the program. During the trial runs it was noticed that a single subroutine for all different block sizes resulted in very long execution times under 8x8 and 16x16 options. This led to a split of the subroutine

into two as a necessity. The potential to use block sizes other than 8x8 makes the program more universal and capable of investigating their properties.

The program also checks whether the minimum margin of majority of the appropriate sign prevails in a segment after the embedding operation. If it is detected that the hiding operation did not create large enough majority of the appropriately signed coefficients, the data hiding operation is repeated and more coefficients' signs are reversed.

Overall there are 15 source files, 14 header files, 4 resource files, 13 classes and 16 global functions. The total amount of written code verges on 7800 lines. This reflects the complexity of the program (Figure 4.10).

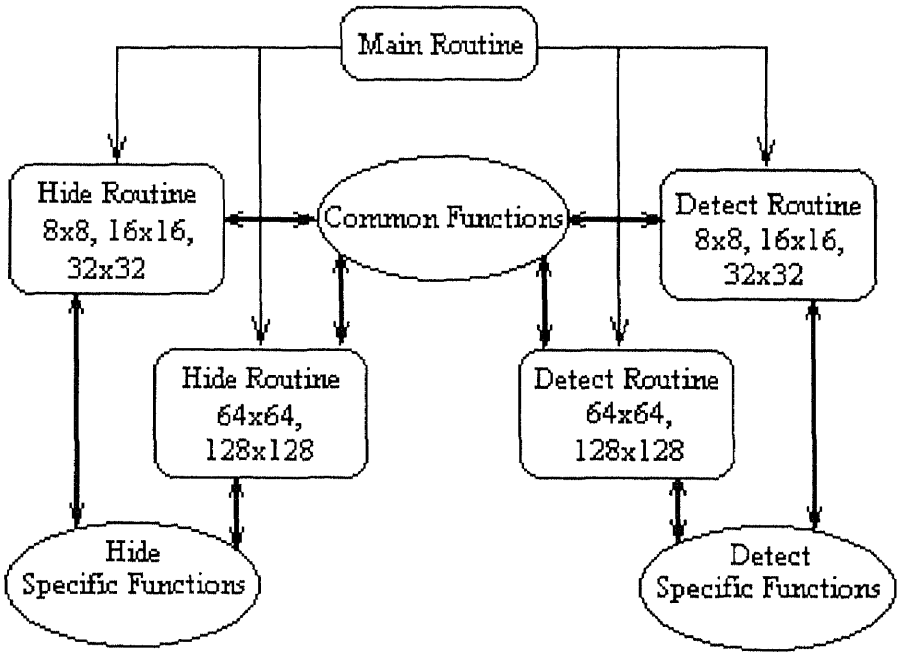

Figure 4.10 DHS1 - Program Structure

4.2 Method II - DFT Web-based Software Package

For the second method's data hiding algorithm a Web-based package was built. You can test run it at *http://www.njcmr.org/dhs/*. The software was written in ANSI

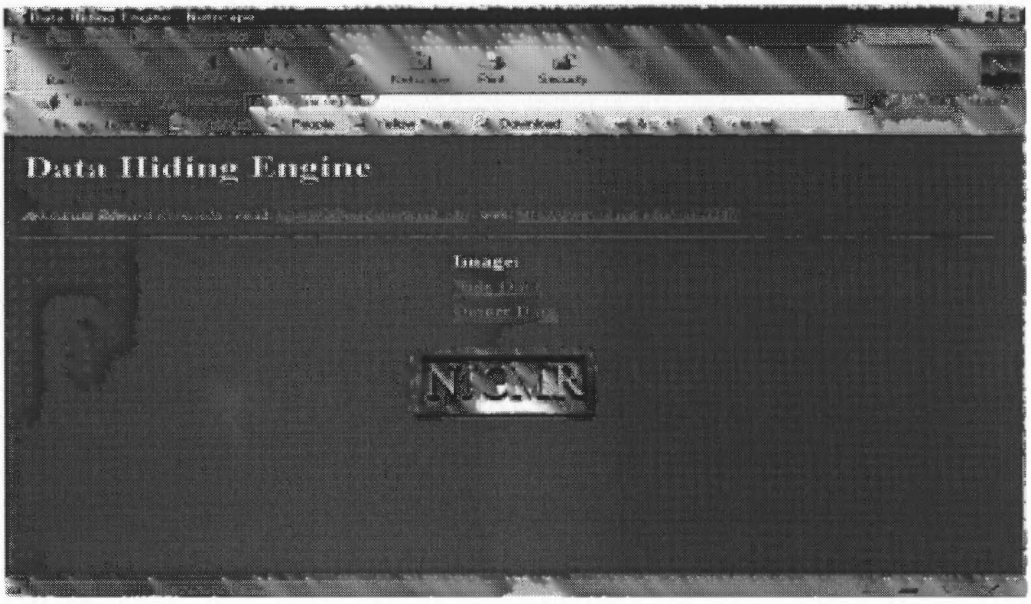

Figure 4.11 DHS5 - Main Screen

(American National Standards Institute) C/C++ for it to run on Unix machines. However, this made the task more complex than it would have been under the Windows-based system. ANSI standard does not support many convenient and useful functions available in Windows. This necessitated their development. The software package code named *DHS5* (*Data Hiding Software 5*) uses CGI (Common Gateway Interface) interface to allow a Web server to run a script on the server and send the output to a user's Web browser.

4.2.1 DHS5: Interface

The package can be accessed via the Internet at *http://www.njcmr.org/dhs/* (Figure 4.11). Data can only be embedded into single images. There is no video sequence option, since CGI Timeout prevents time consuming operations over the Internet causing a break in connection.

For steganography choose **Hide Data** link which will lead to Figure 4.12 screen. Graphical file formats supported are: *pgm* for black-and-white images, *ppm* and *.y.u.v* for color images. A detailed explanation of the *.y.u.v* format can be found in Section

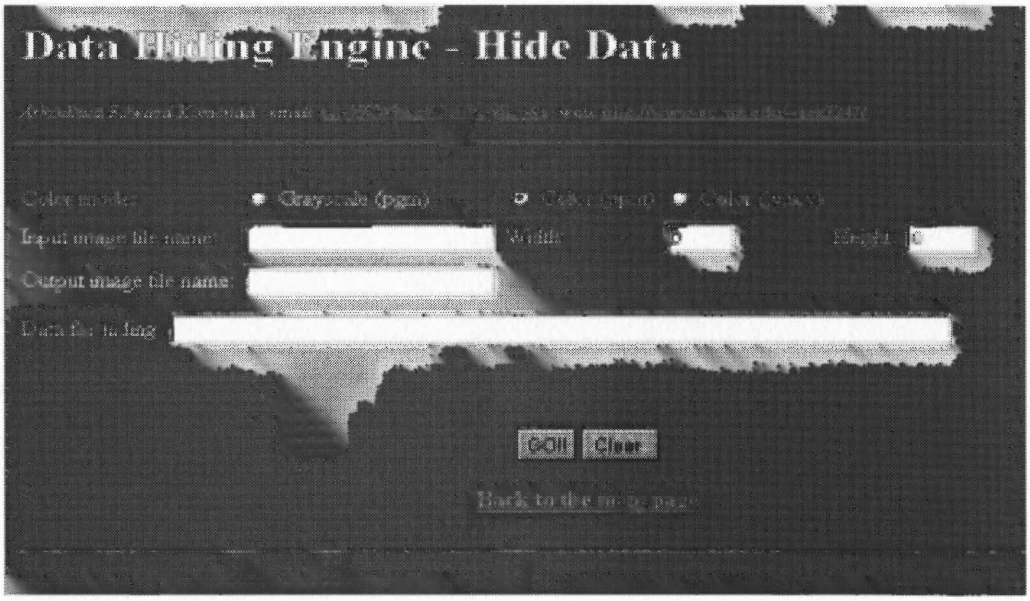

Figure 4.12 DHS5 - Data Hiding Screen

4.1.1. Note that if you would like to hide data inside a *raw* file, just change the file's extension to *.y* and run the program under the *.y.u.v* color mode. This is possible since the program hides data only in the luminance component which in the case of *.y.u.v* is solely represented by the file with a *.y* extension. Hence *.u* and *.v* files are not used by the program at all. The program will treat data from the *raw* file as the luminance data (as is the case in many instances) and perform the normal embedding. In the *Input image file name* field enter the original, unwatermarked image file name complete with the extension. For the *.y.u.v* case the extension ought to be *.y*. An incorrect file name results in the display of the error screen. Output file's name has to be provided in the *Output image file name* box. Enter width and height of the original image into the appropriate fields. The data to be hidden goes into the *Data for hiding* field. To proceed with the steganography press **Go** key, to clear all entries click on **Clear**.

After a successful run the following window appears, see Figure 4.13. The name of the file with hidden data inside, "lenaout.ppm", is provided in the upper portion

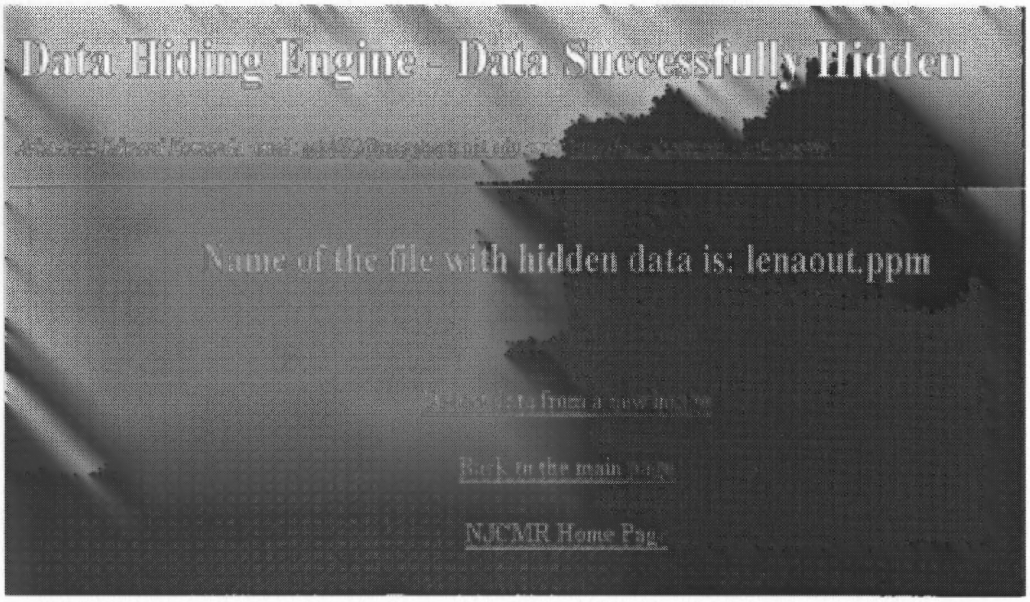

Figure 4.13 DHS5 - Screen after Data Hiding

of the page. To hide data in the next file click on the **Hide data in the next image** link, to proceed with data detection press on **Detect data from a new image**.

An alternative way to reach the detection screen (Figure 4.14) is to click on **Detect Data** on the main screen. Select the appropriate *Color mode* and enter the *Input image file name* which should be identical to the *Output image file name* used at the data hiding stage. Width and height sizes must be equal to the ones of the original image. To activate data extraction press **Go**, for clearing the entries click on **Clear**. The retrieved data, "Testing run !!", is shown on the first line of the screen below the header, see Figure 4.15. To further detect data from other files click on **Detect data from the next image**, to switch to hiding mode press **Hide data in a new image**.

4.2.2 DHS5: Package Structure

The graphic file formats *pgm* and *ppm* were chosen as input files for their acceptance in academic circles and popular availability in the commercial image editing software. $.y.u.v$ format was also incorporated in preparation for the next release of the software

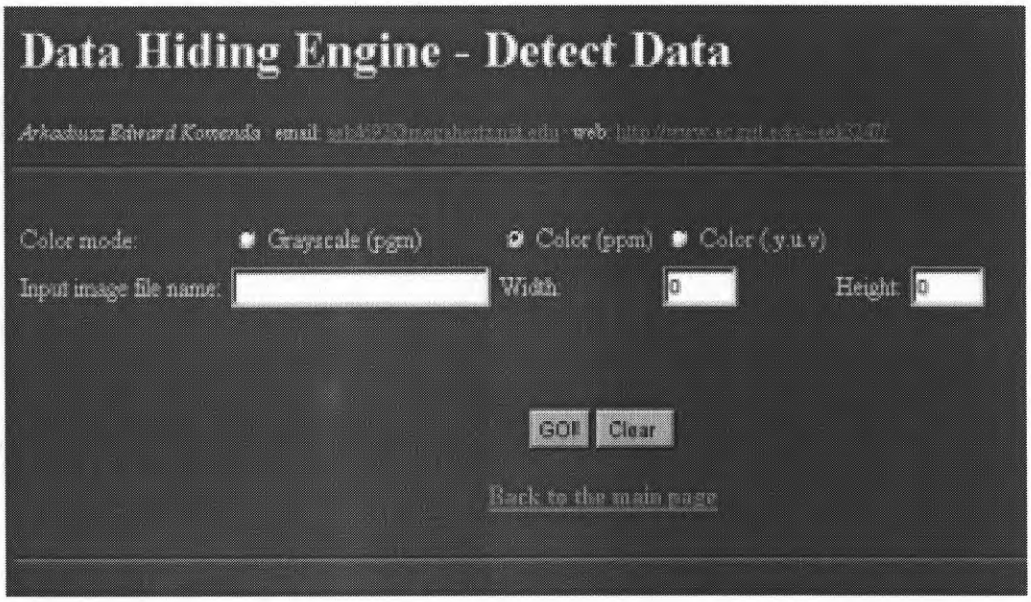

Figure 4.14 DHS5 - Data Detection Screen

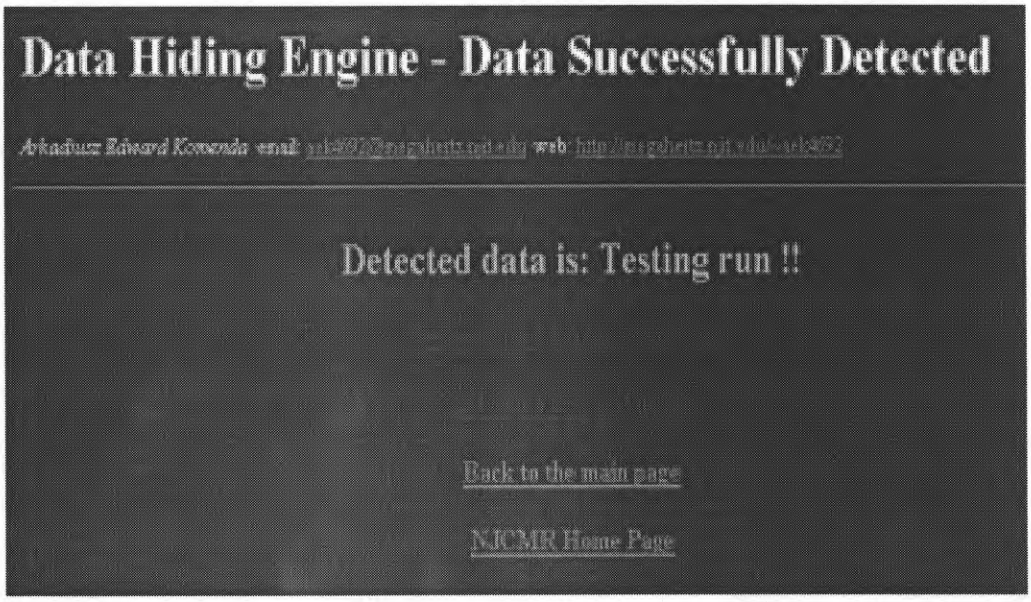

Figure 4.15 DHS5 - Screen after Data Detection

for Windows-based systems (due to CGI Timeout issue, see Section 4.2.1) which would include steganography in video sequences.

The need to input picture dimensions stems from their unavailability in $.y.u.v$ format and a plan to include *raw* file format as one of the input options. The DFT is performed on the 8x8 blocks of pixels. However, this does not limit image sizes to ones divisible into blocks of 8x8. This is possible due to the program's subroutine which narrows used picture area, to include the largest feasible, but still divisible into 8x8 blocks image portion, for data hiding. The rest of the picture is saved for later use in the construction of the watermarked picture.

Similarly to DHS1, the DHS5 program operates on the transposed image. This is a consequence of C/C++ limitation which allows only the array's row number to be set dynamically, while the number of columns has to be fixed. Since it's more likely that the picture is wider than higher, it was decided to set a vertical limit for the picture to 1000 pixels and let the horizontal dimension be only limited by hardware constraints. The only possible way of implementing this was by loading the image into an array rotated by 90 degrees (transposed).

The whole package consists of about 3700 lines of code in both C/C++ and HTML, and 17 different functions. It is divided into 3 standalone parts (Figure 4.16):

- main screen written in HTML with CGI,
- data hiding subprogram in C/C++ and HTML plus CGI,
- data extraction subprogram in C/C++ and HTML plus CGI.

A complete C/C++, HTML and CGI listing of the program can be found in Appendix B. The Matlab equivalent which was used for testing integrity and reliability of the Web-based system was also written.

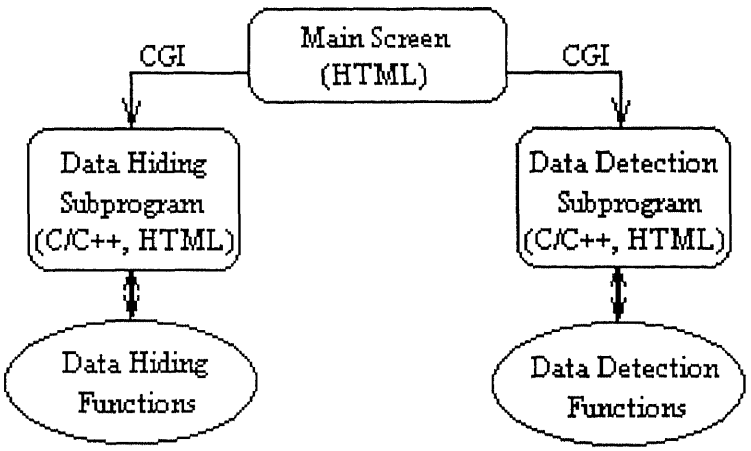

Figure 4.16 DHS5 - Package Structure

CHAPTER 5

A COMPARATIVE PERFORMANCE STUDY FOR TWO DATA HIDING SCHEMES IN IMAGE AND VIDEO

A scrupulous analysis of both steganographic techniques was performed with a special focus on hidden data survival under the JPEG and SPIHT compressions. In addition, an MPEG-2 video compression was tested for the first method. Both schemes were also subjected to an array of common image editing processes which included, but were not limited to, resizing, rotation, noise addition, sharpening and bluring. In all cases Paint Shop Pro Version 5.01 was used as an image editing tool.

5.1 Determination of the Majority Sign Threshold for Method I

The DCT technique has a majority sign threshold (dominance threshold) constant that had to be determined experimentally, before the algorithm could be used. This constant simply tells what the minimal number difference between the coefficients with the correct signs and ones with the opposite signs within a segment should be. For example, is there supposed to be 5, 10 or 20 more positive coefficients than negative ones while embedding a '1' bit?

5.1.1 Case of JPEG Compression (Grayscale Images)

An experiment was conducted on 3 grayscale 256x256 test images, namely Lena, Tree and Baboon, to determine the optimal majority sign threshold. In each image a 64-bit long binary signature was hidden using an 8x8 DCT block transform. Then the image was subjected to a JPEG compression with the quality value of 10. Next, the image was decompressed into a pgm file and data detection was performed on it. This procedure was repeated for the JPEG quality factors of 20, 30, 40, ..., 100. In addition different block sizes of 16x16, 32x32, 64x64 and 128x128 were also experimented. And just to improve the reliability of data, the whole process was

repeated for a new different signature. The thresholds tried were: 10, 22, 27, 32, 37, 42, 47, 52, 67, 72, 77, 82, 87, 92, 97, 102, 107, 112, 117, 122.

For results see Figures 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 5.10. It is seen that the optimal value for the majority sign threshold constant is around 20. In particular, see the last four figures for block sizes of 64x64 and 128x128.

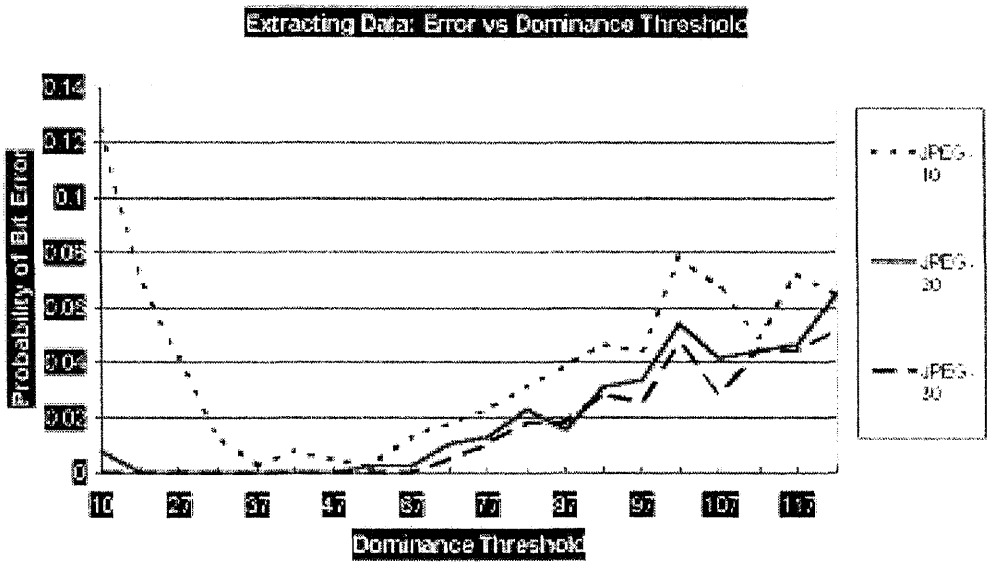

Figure 5.1 JPEG DCT - 8x8: Testing Various Majority Sign Thresholds (Part 1)

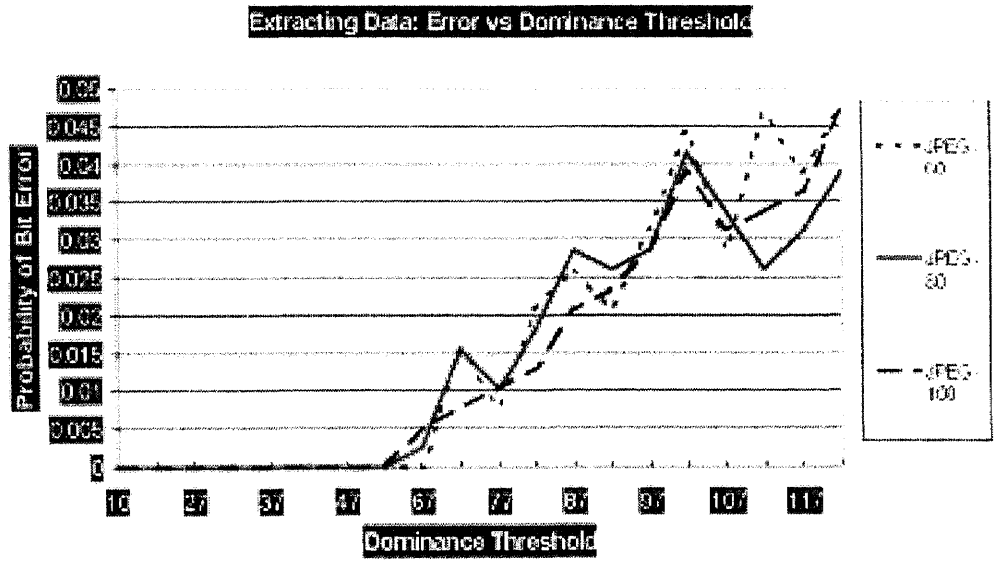

Figure 5.2 JPEG DCT - 8x8: Testing Various Majority Sign Thresholds (Part 2)

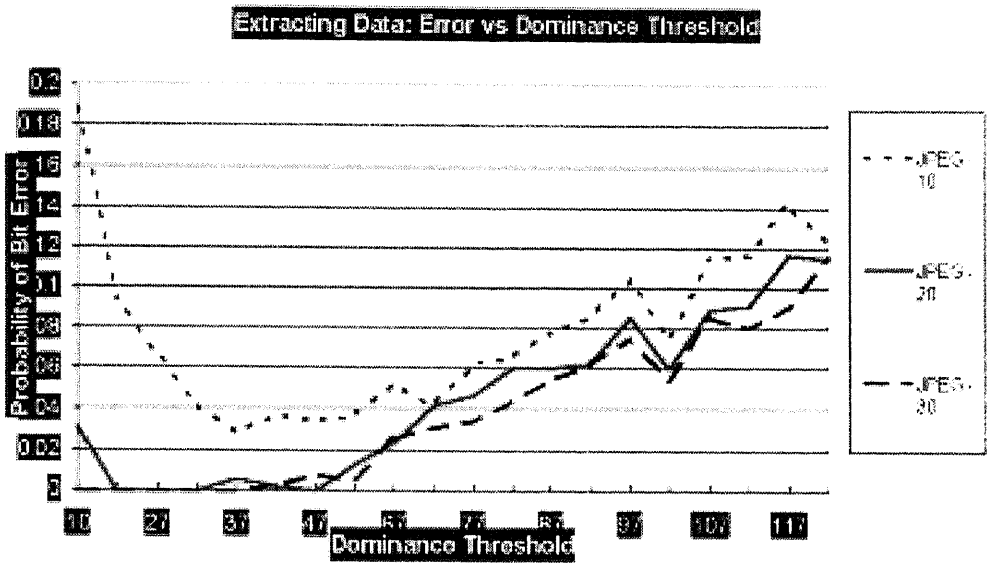

Figure 5.3 JPEG DCT - 16x16: Testing Various Majority Sign Thresholds (Part 1)

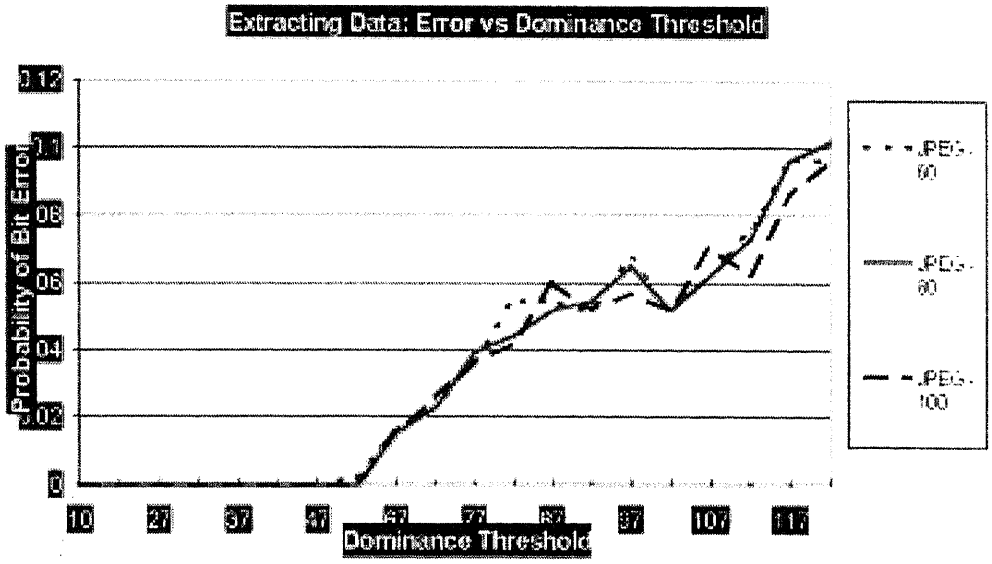

Figure 5.4 JPEG DCT - 16x16: Testing Various Majority Sign Thresholds (Part 2)

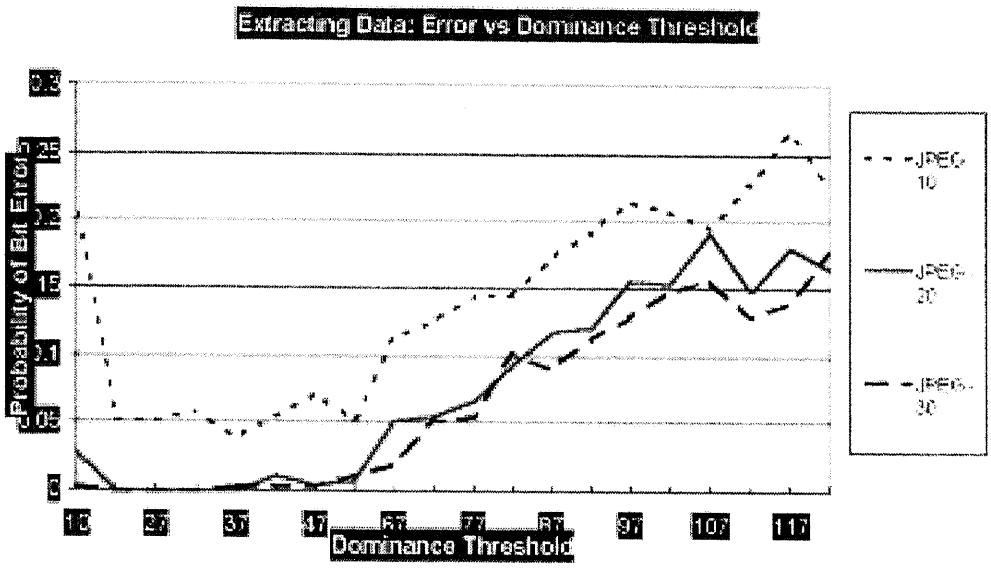

Figure 5.5 JPEG DCT - 32x32: Testing Various Majority Sign Thresholds (Part 1)

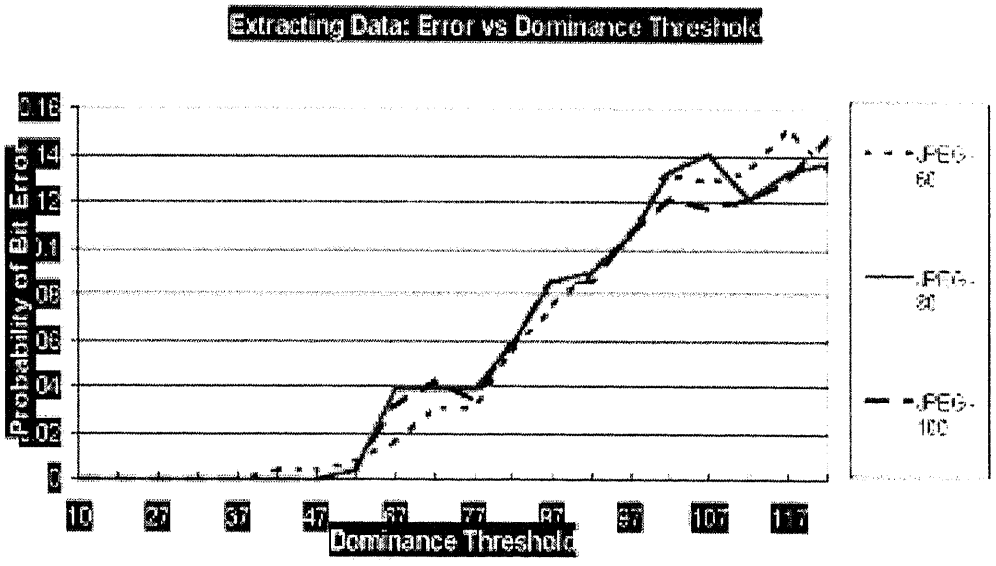

Figure 5.6 JPEG DCT - 32x32: Testing Various Majority Sign Thresholds (Part 2)

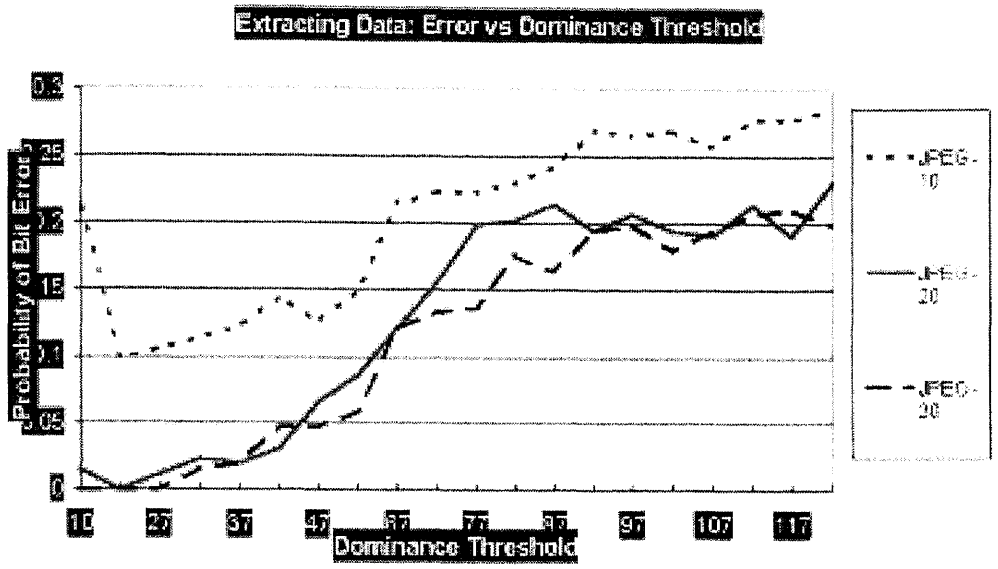

Figure 5.7 JPEG DCT - 64x64: Testing Various Majority Sign Thresholds (Part 1)

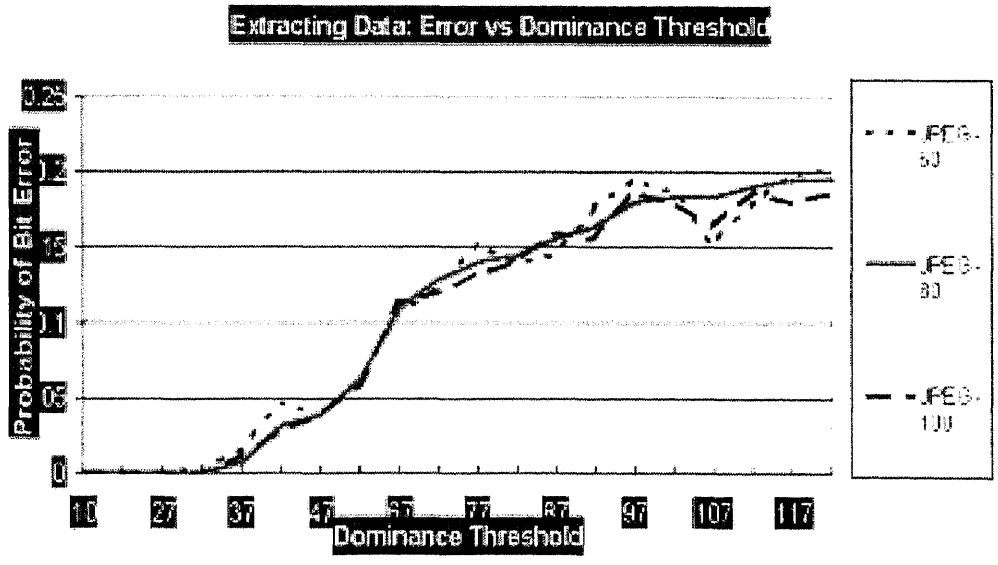

Figure 5.8 JPEG DCT - 64x64: Testing Various Majority Sign Thresholds (Part 2)

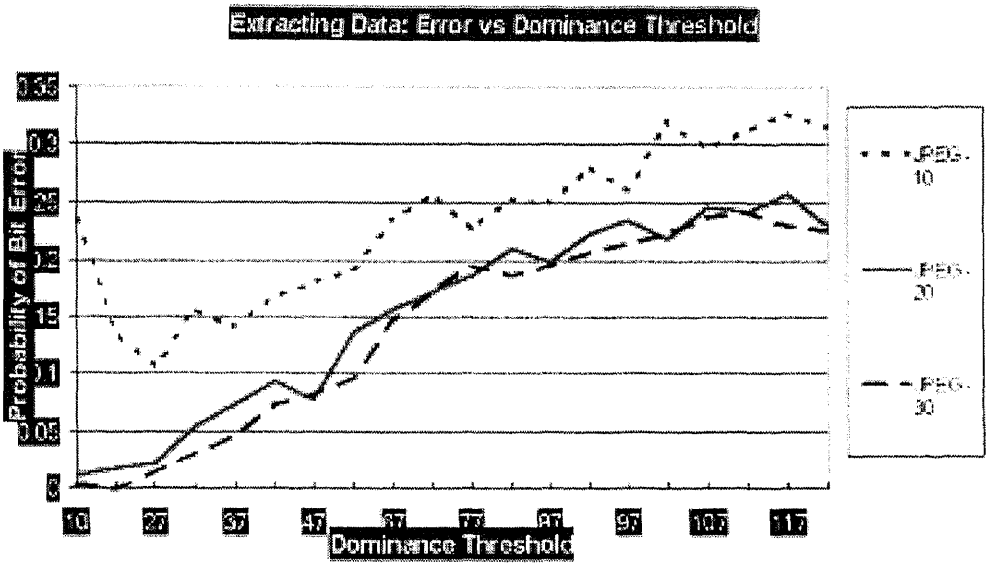

Figure 5.9 JPEG DCT - 128x128: Testing Various Majority Sign Thresholds (Part 1)

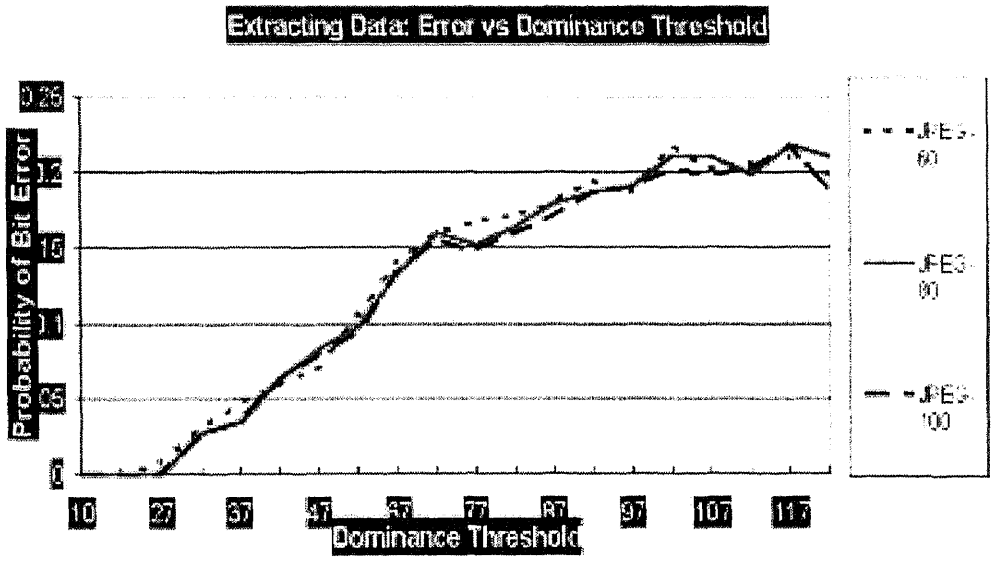

Figure 5.10 JPEG DCT - 128x128: Testing Various Majority Sign Thresholds (Part 2)

5.1.2 Case of SPIHT Compression (Grayscale Images)

The same experiment was conducted for SPIHT compression. However, this time we used:

- 7 grayscale 256x256 test images: Lena, Baboon, Tree, Splash, Tiffany, Girl and Peppers,
- 10 different 64-bit long binary signatures,
- 5 different DCT block sizes: 8x8, 16x16, 32x32, 64x64, 128x128,
- 6 SPIHT bitrates: 0.25, 0.50, 0.75, 1.00, 1.25, 1.50.

For results see Figures 5.11, 5.12, 5.13, 5.14, 5.15, 5.16, 5.17, 5.18, 5.19, 5.20. It can be seen that the optimal value for the majority sign threshold constant for SPIHT is around 25, note figures for block sizes of 64x64 and 128x128.

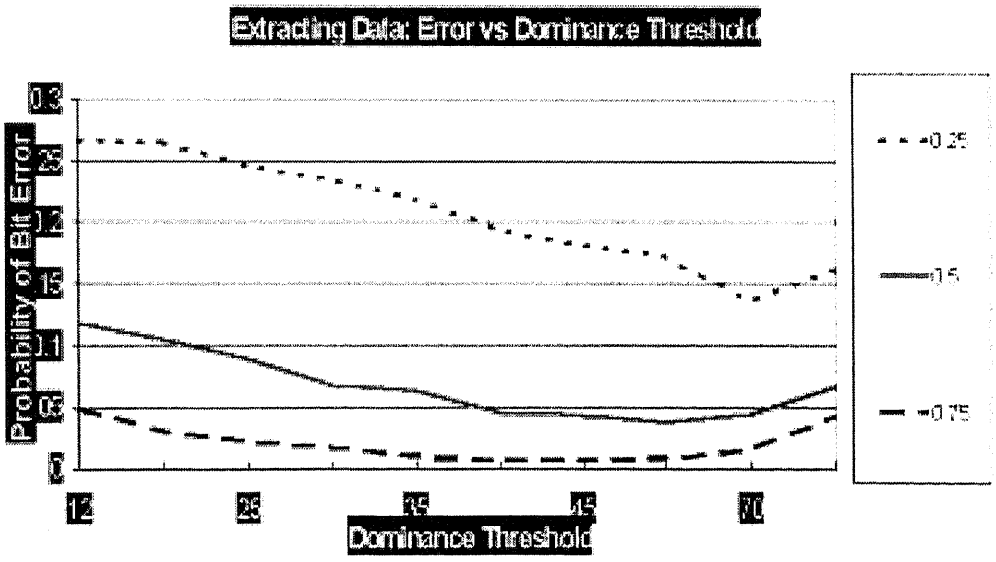

Figure 5.11 SPIHT DCT - 8x8: Testing Various Majority Sign Thresholds (Part 1)

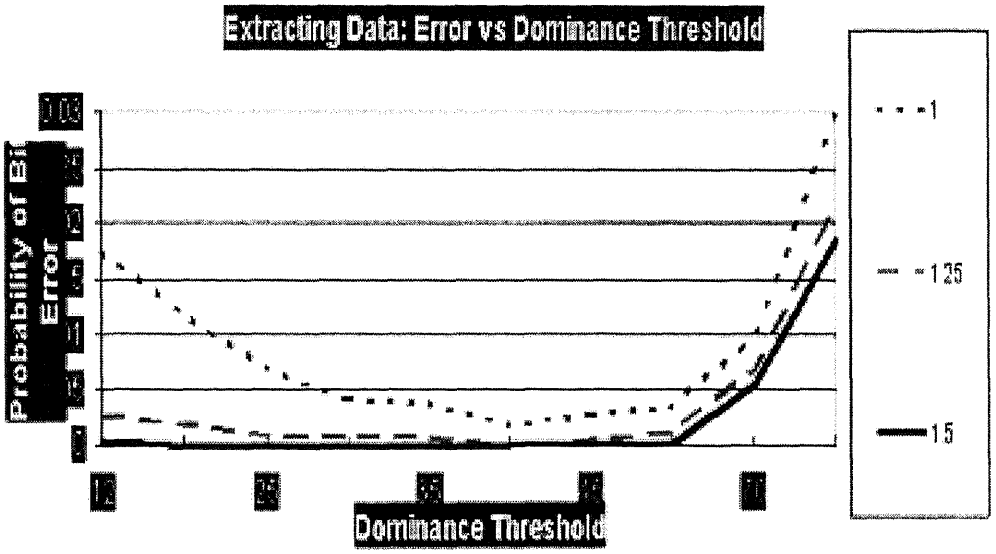

Figure 5.12 SPIHT DCT - 8x8: Testing Various Majority Sign Thresholds (Part 2)

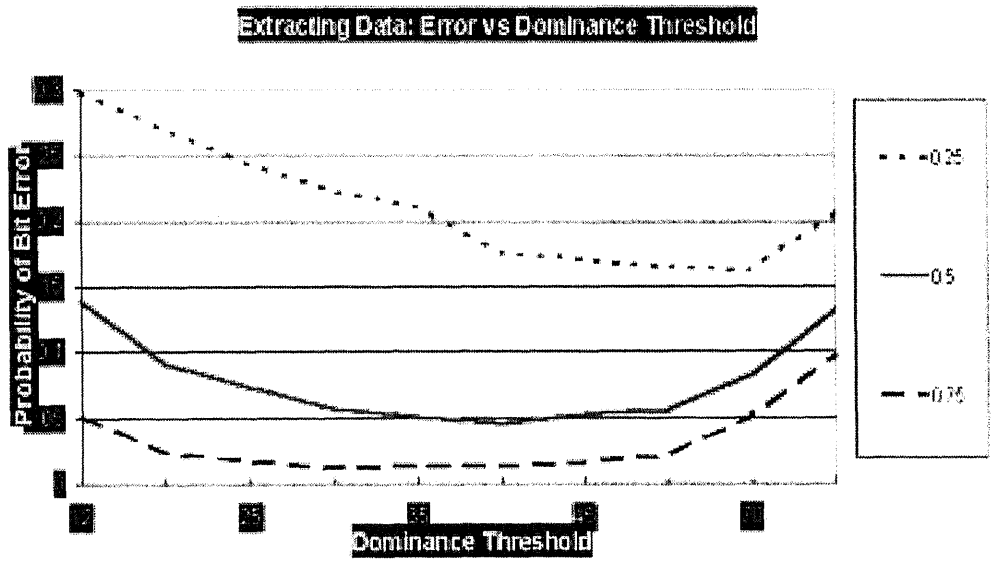

Figure 5.13 SPIHT DCT - 16x16: Testing Various Majority Sign Thresholds (Part 1)

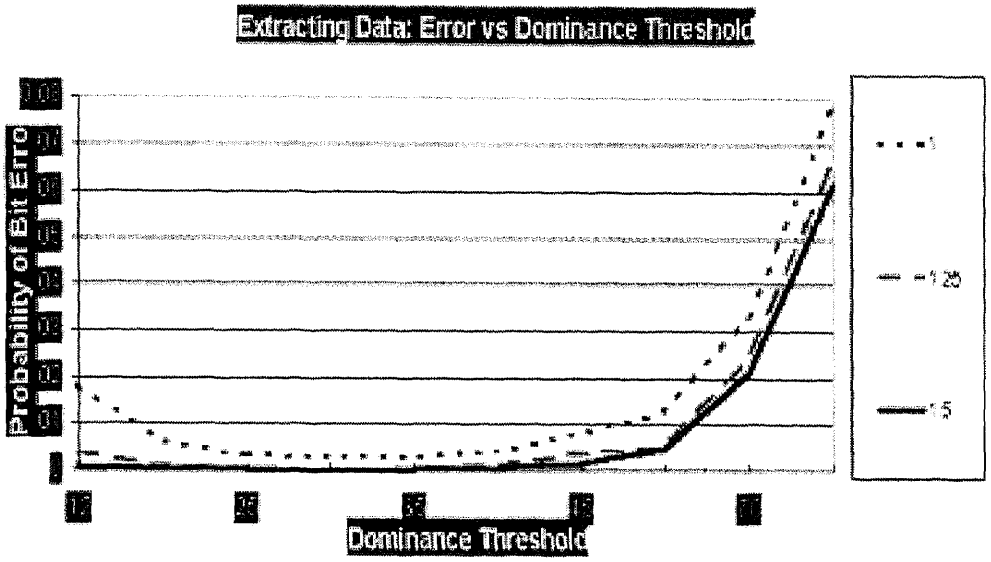

Figure 5.14 SPIHT DCT - 16x16: Testing Various Majority Sign Thresholds (Part 2)

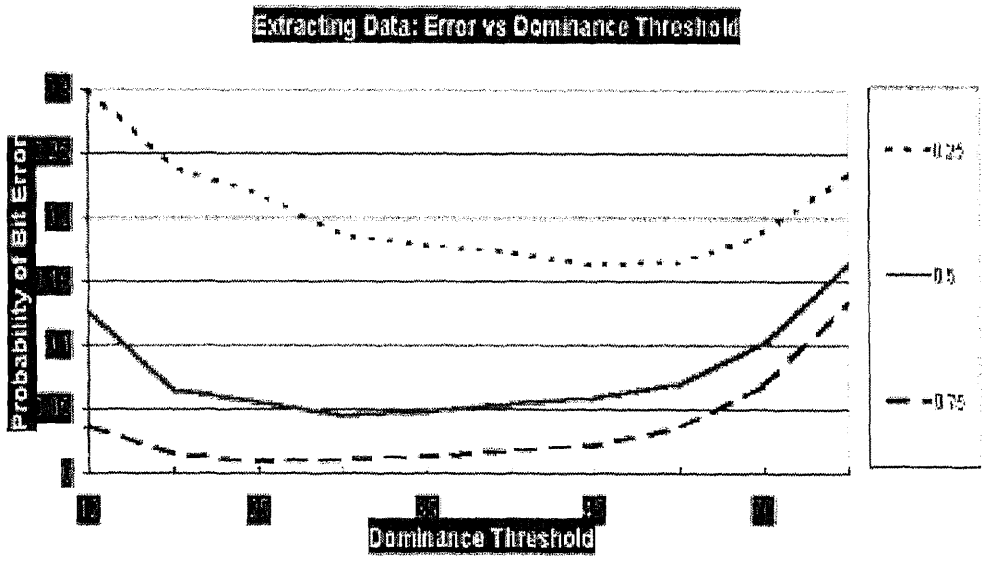

Figure 5.15 SPIHT DCT - 32x32: Testing Various Majority Sign Thresholds (Part 1)

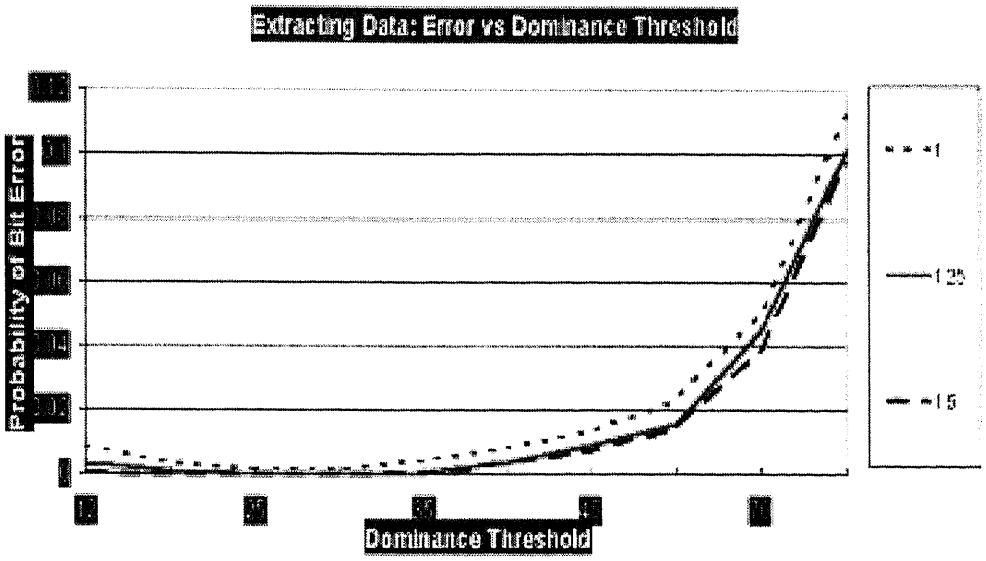

Figure 5.16 SPIHT DCT - 32x32: Testing Various Majority Sign Thresholds (Part 2)

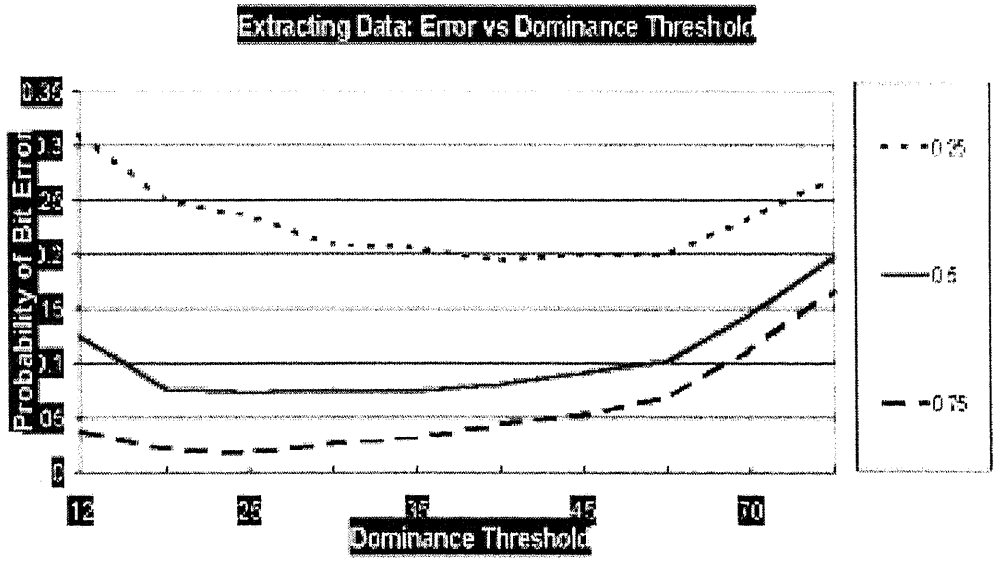

Figure 5.17 SPIHT DCT - 64x64: Testing Various Majority Sign Thresholds (Part 1)

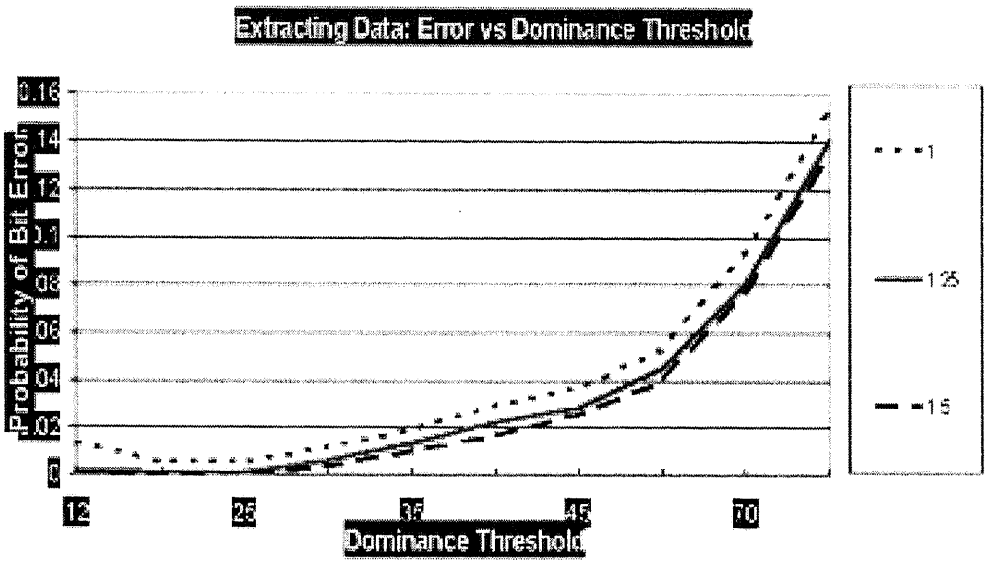

Figure 5.18 SPIHT DCT - 64x64: Testing Various Majority Sign Thresholds (Part 2)

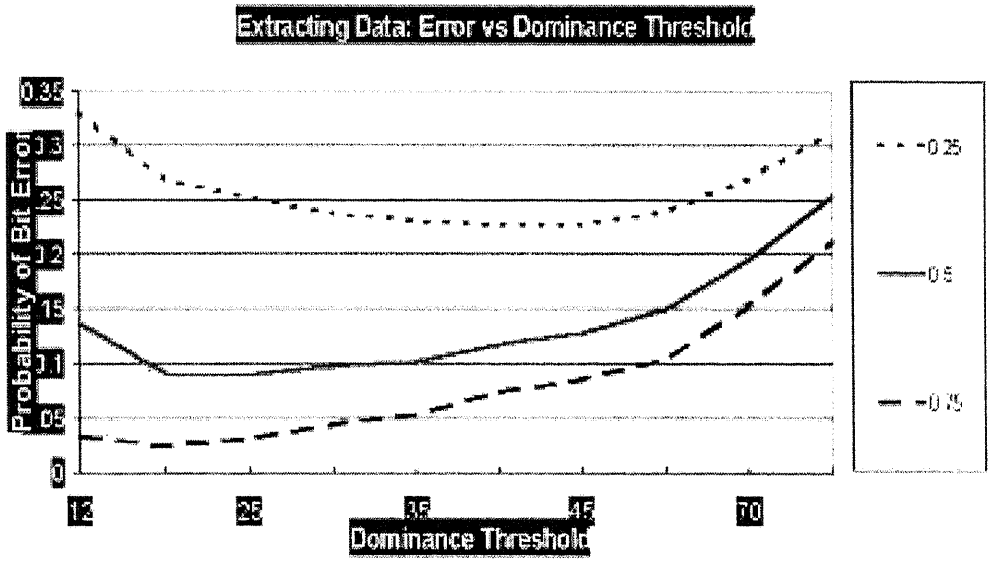

Figure 5.19 SPIHT DCT - 128x128: Testing Various Majority Sign Thresholds (Part 1)

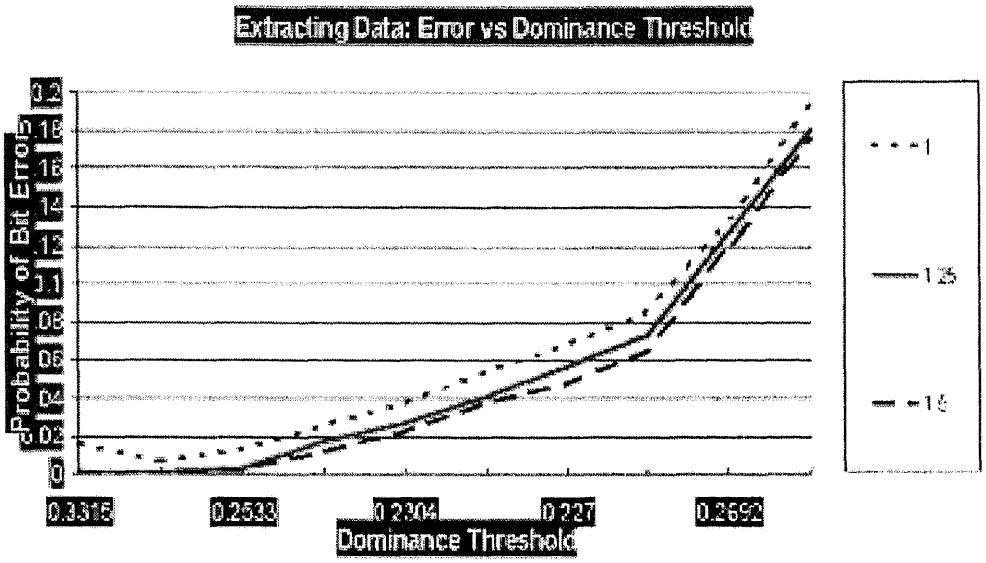

Figure 5.20 SPIHT DCT - 128x128: Testing Various Majority Sign Thresholds (Part 2)

5.1.3 Case of JPEG Compression (Color Images)

Similar experiment to the two previous ones was performed on color images which underwent JPEG compression. The following options were used:

- 2 color (ppm) 256x256 images: Lena and Tree,
- 2 different 64-bit long binary signatures,
- 5 different DCT block sizes: 8x8, 16x16, 32x32, 64x64, 128x128,
- 10 JPEG quality factors: 10, 20, 30, ..., 100.

See Figures 5.21, 5.22, 5.23, 5.24, 5.25, 5.26, 5.27, 5.28, 5.29, 5.30 for results. From the last 4 graphs it can be seen that the majority sign threshold constant should be chosen to be 20.

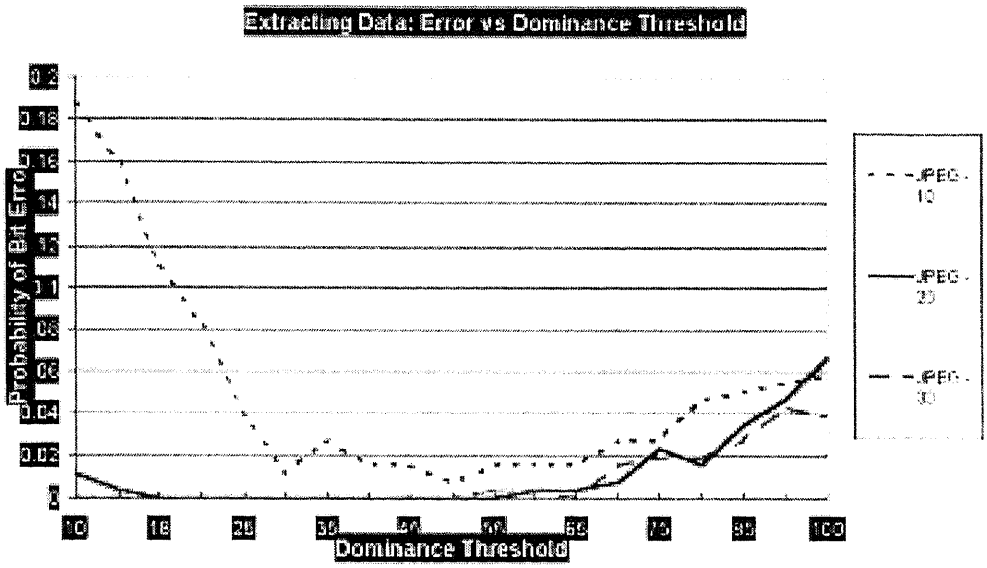

Figure 5.21 JPEG DCT - 8x8: Testing Various Majority Sign Thresholds (Part 1)

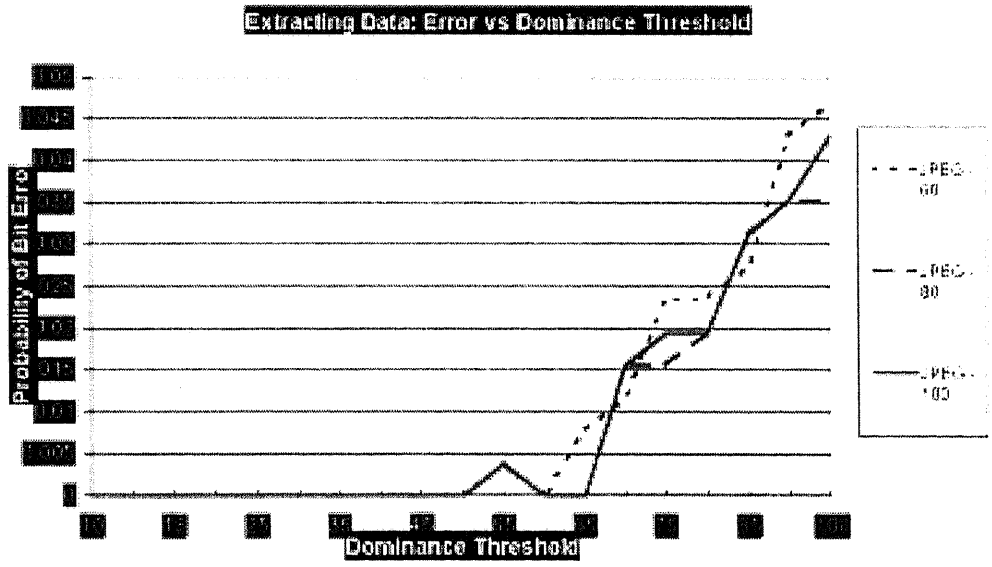

Figure 5.22 JPEG DCT - 8x8: Testing Various Majority Sign Thresholds (Part 2)

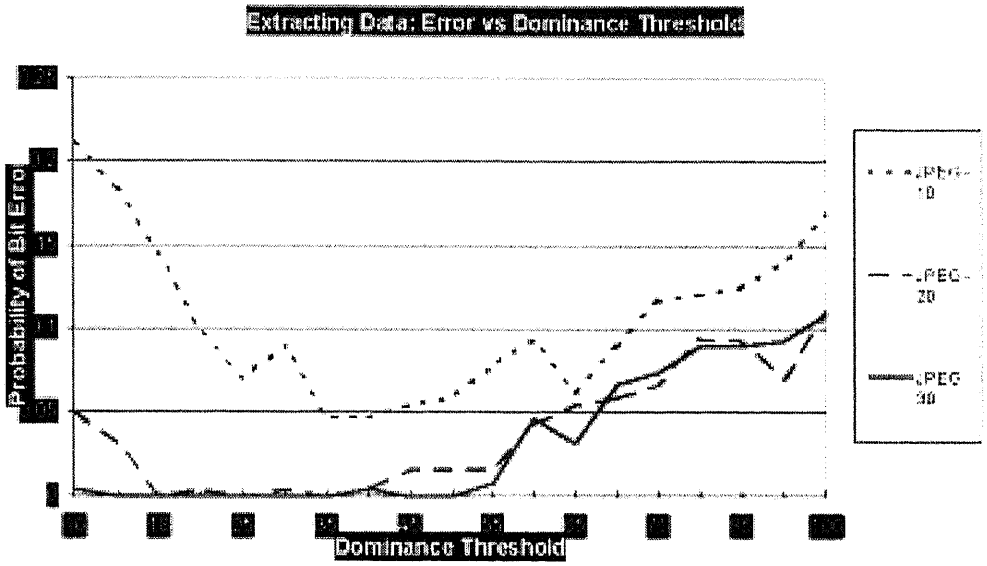

Figure 5.23 JPEG DCT - 16x16: Testing Various Majority Sign Thresholds (Part 1)

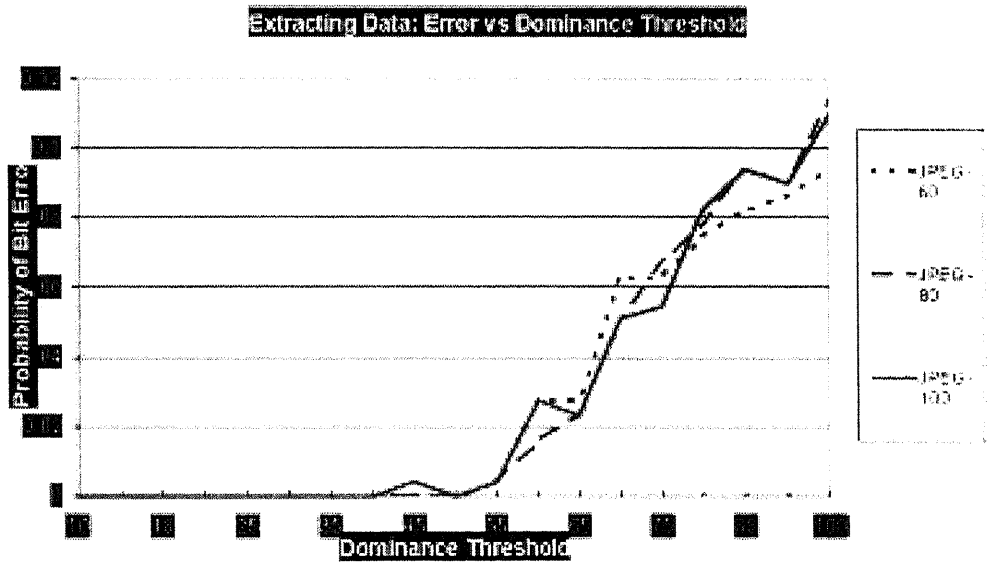

Figure 5.24 JPEG DCT - 16x16: Testing Various Majority Sign Thresholds (Part 2)

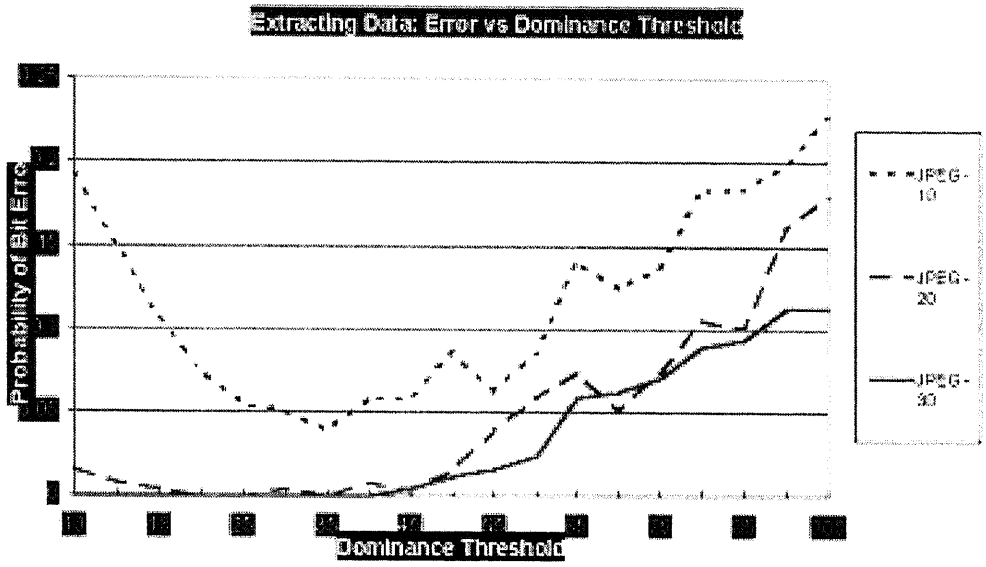

Figure 5.25 JPEG DCT - 32x32: Testing Various Majority Sign Thresholds (Part 1)

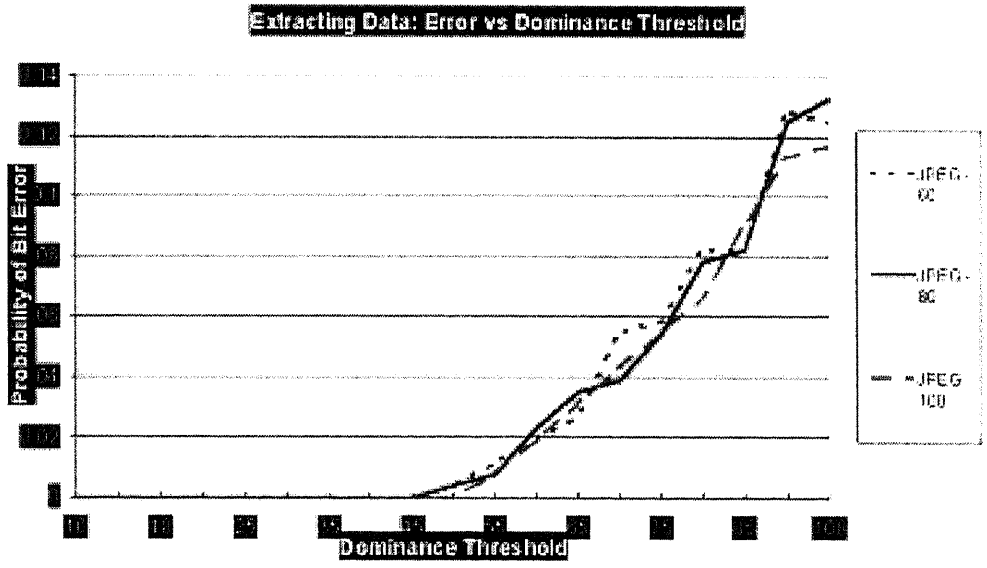

Figure 5.26 JPEG DCT - 32x32: Testing Various Majority Sign Thresholds (Part 2)

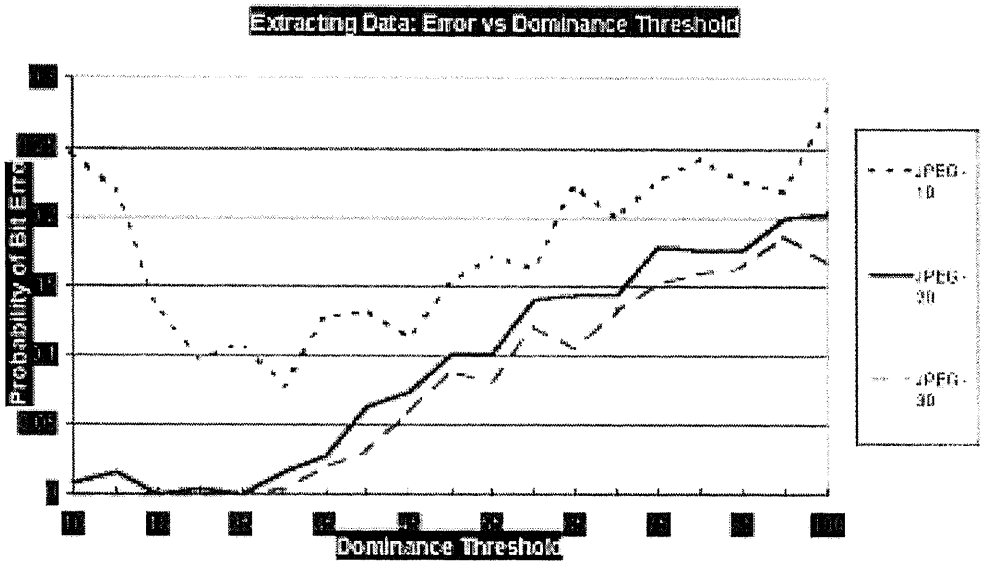

Figure 5.27 JPEG DCT - 64x64: Testing Various Majority Sign Thresholds (Part 1)

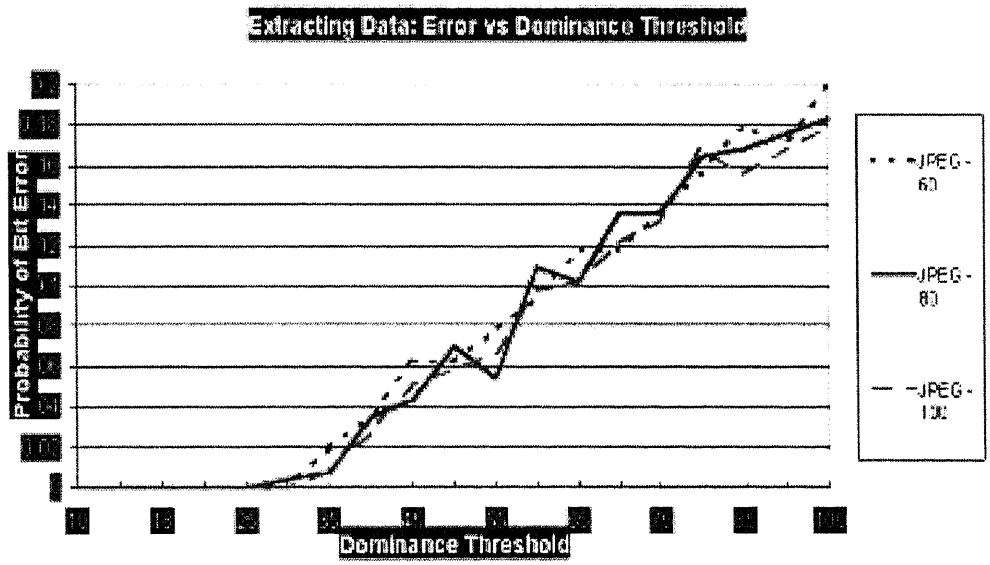

Figure 5.28 JPEG DCT - 64x64: Testing Various Majority Sign Thresholds (Part 2)

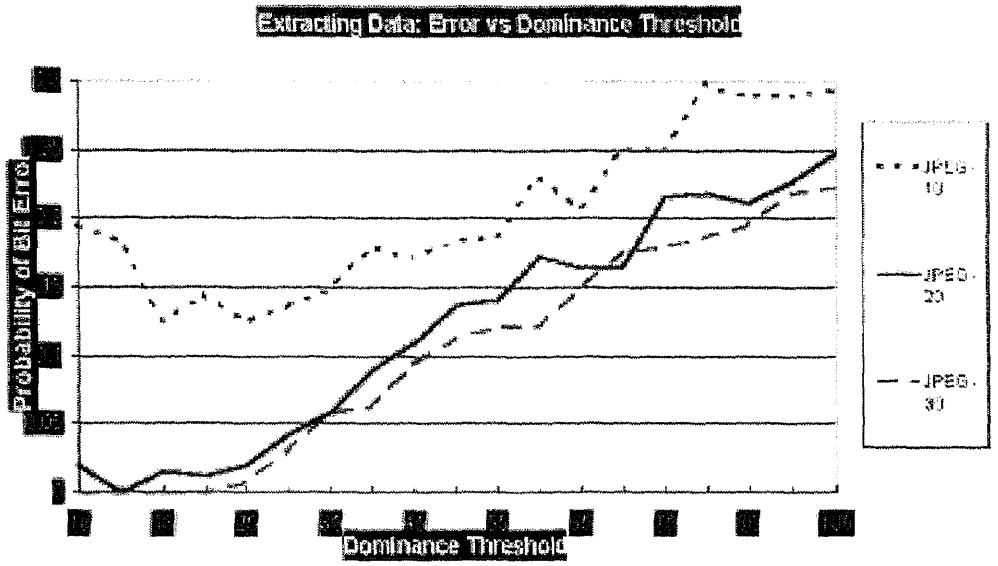

Figure 5.29 JPEG DCT - 128x128: Testing Various Majority Sign Thresholds (Part 1)

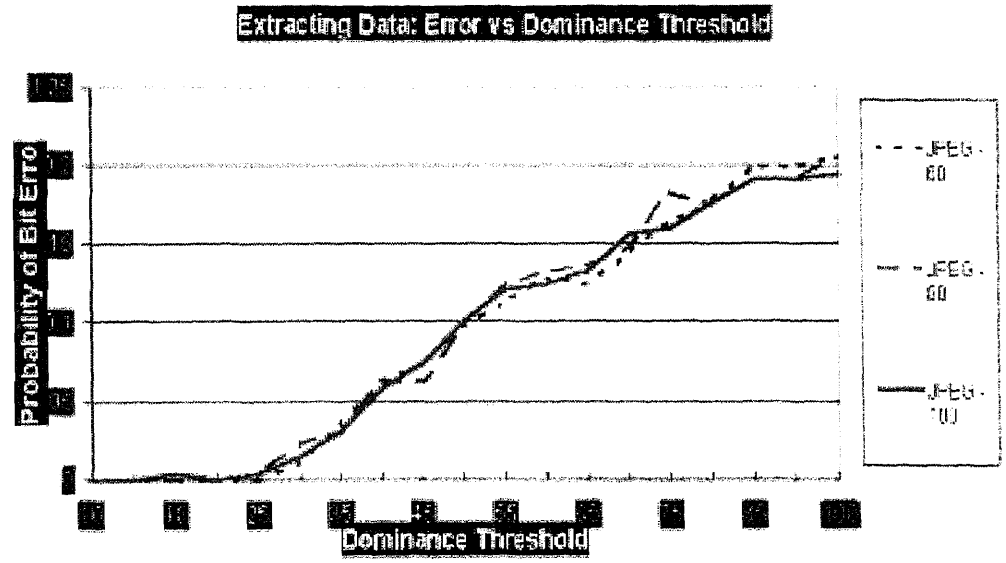

Figure 5.30 JPEG DCT - 128x128: Testing Various Majority Sign Thresholds (Part 2)

5.1.4 Case of SPIHT Compression (Color Images)

SPIHT compression was also used for finding the majority sign threshold. In this case we used:

- 3 color (ppm) 256x256 images: Lena, Baboon and Tree,
- 2 different 64-bit long binary signatures,
- 5 different DCT block sizes: 8x8, 16x16, 32x32, 64x64, 128x128,
- 6 SPIHT bitrates: 0.25, 0.50, 0.75, 1.00, 1.25, 1.50.

Results can be seen in Figures 5.31, 5.32, 5.33, 5.34, 5.35, 5.36, 5.37, 5.38, 5.39, 5.40. The majority sign threshold constant was chosen to be equal to 25 to perform well under SPIHT compression.

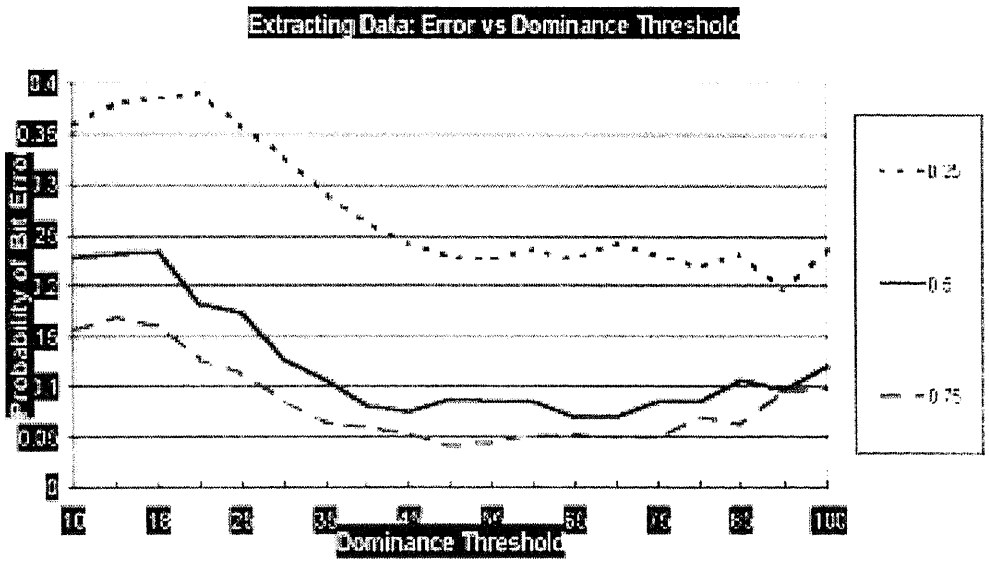

Figure 5.31 SPIHT DCT - 8x8: Testing Various Majority Sign Thresholds (Part 1)

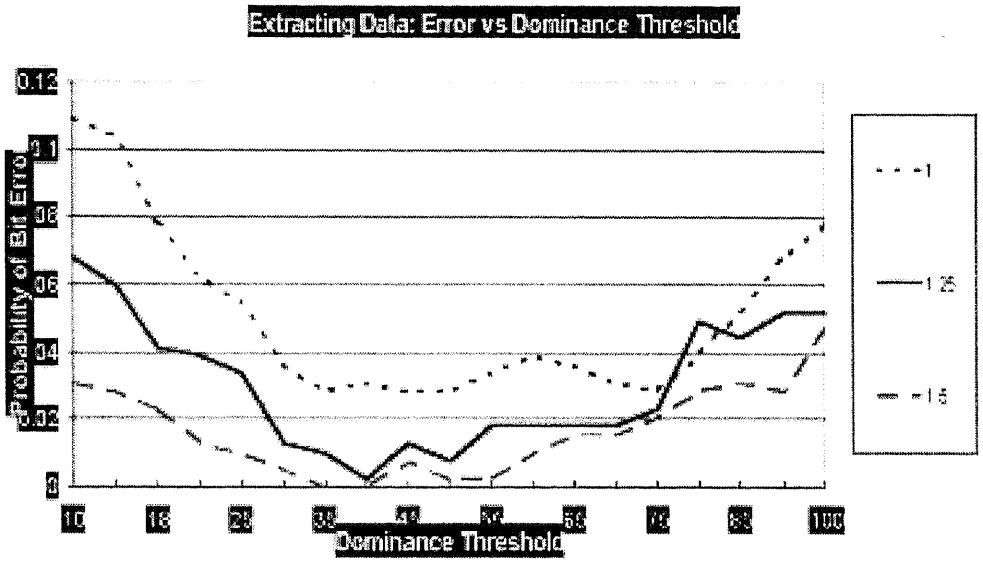

Figure 5.32 SPIHT DCT - 8x8: Testing Various Majority Sign Thresholds (Part 2)

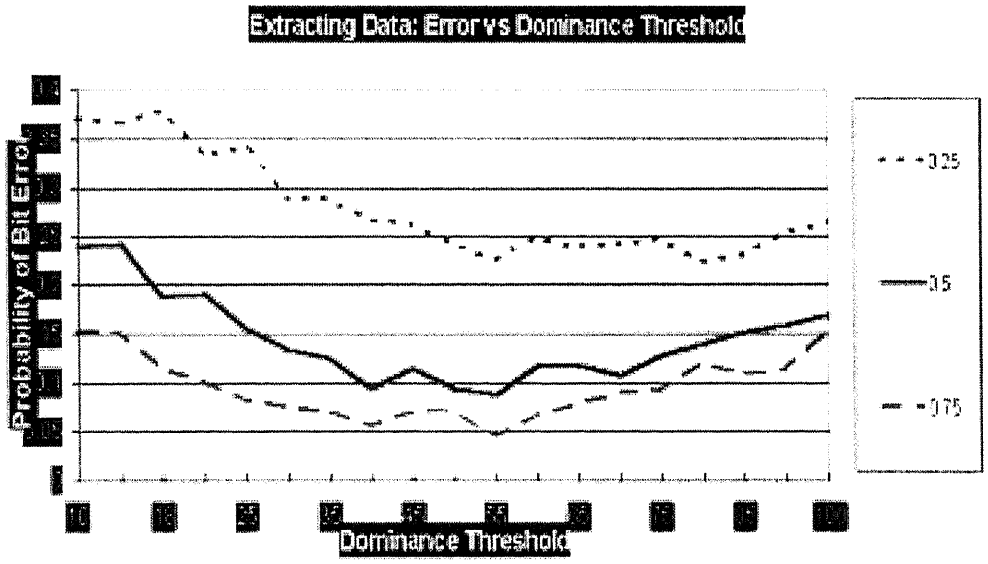

Figure 5.33 SPIHT DCT - 16x16: Testing Various Majority Sign Thresholds (Part 1)

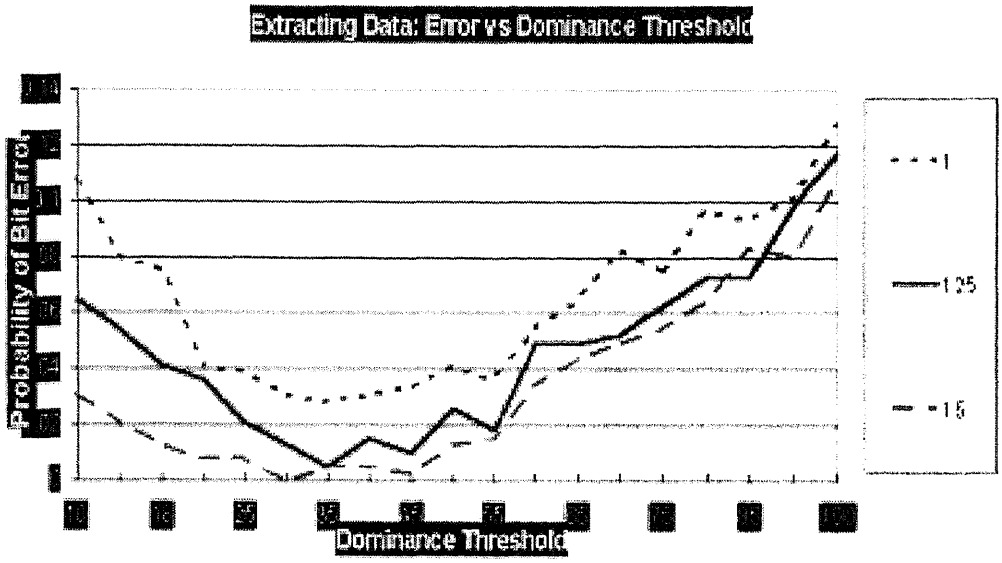

Figure 5.34 SPIHT DCT - 16x16: Testing Various Majority Sign Thresholds (Part 2)

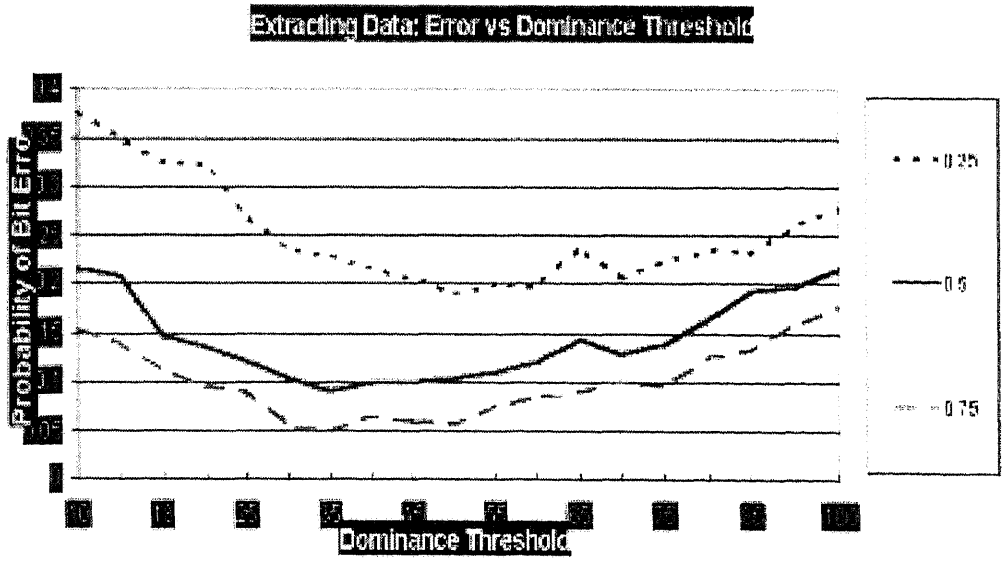

Figure 5.35 SPIHT DCT - 32x32: Testing Various Majority Sign Thresholds (Part 1)

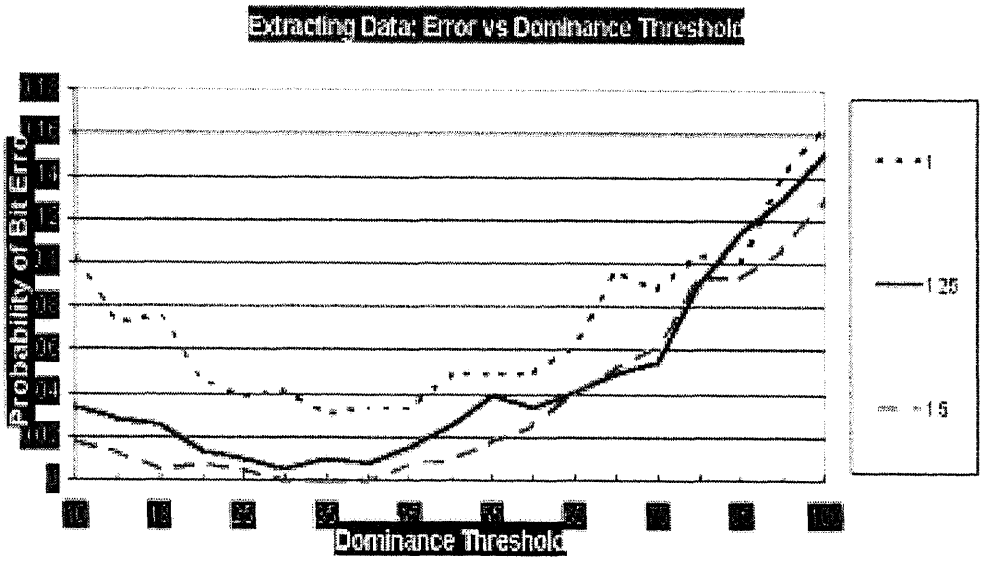

Figure 5.36 SPIHT DCT - 32x32: Testing Various Majority Sign Thresholds (Part 2)

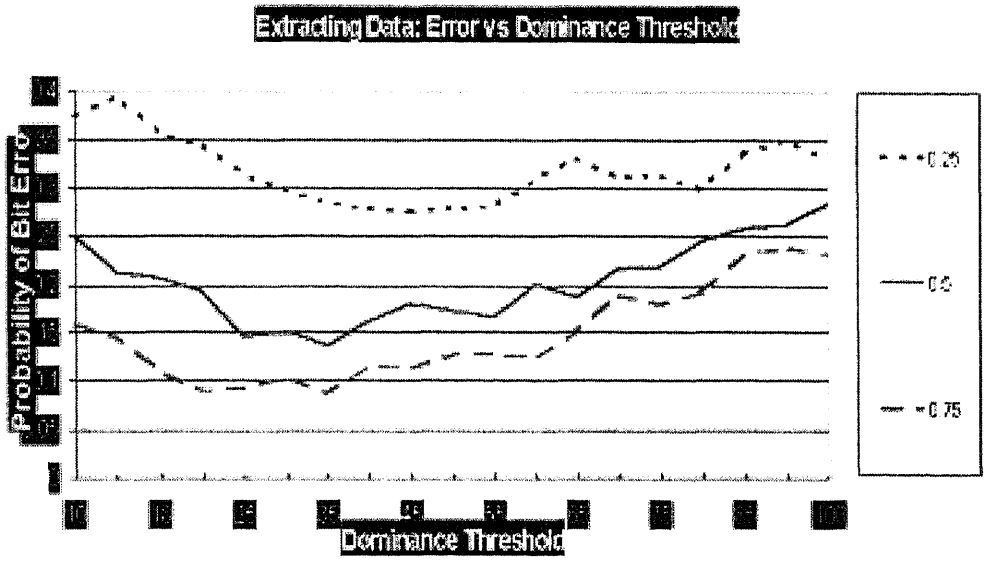

Figure 5.37 SPIHT DCT - 64x64: Testing Various Majority Sign Thresholds (Part 1)

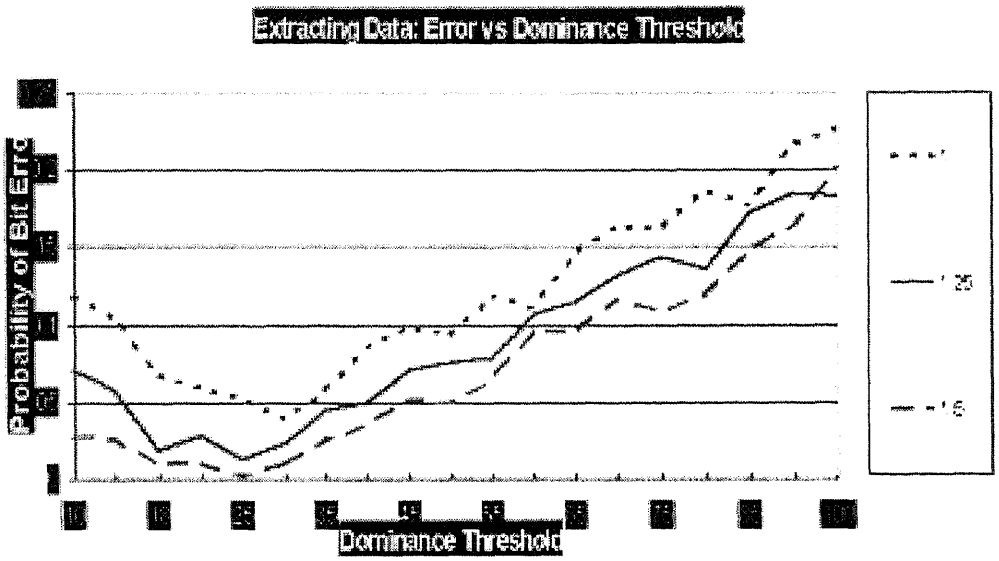

Figure 5.38 SPIHT DCT - 64x64: Testing Various Majority Sign Thresholds (Part 2)

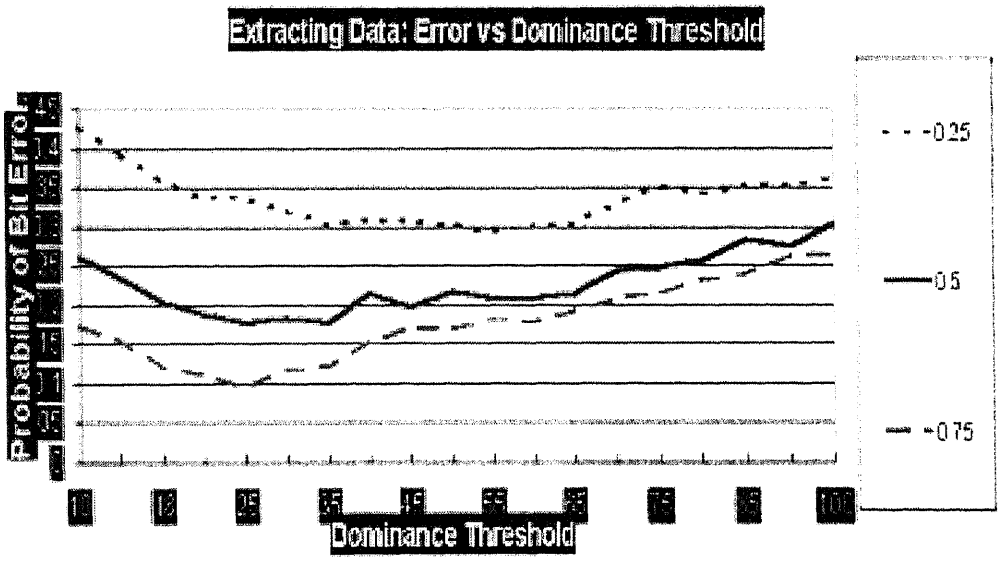

Figure 5.39 SPIHT DCT - 128x128: Testing Various Majority Sign Thresholds (Part 1)

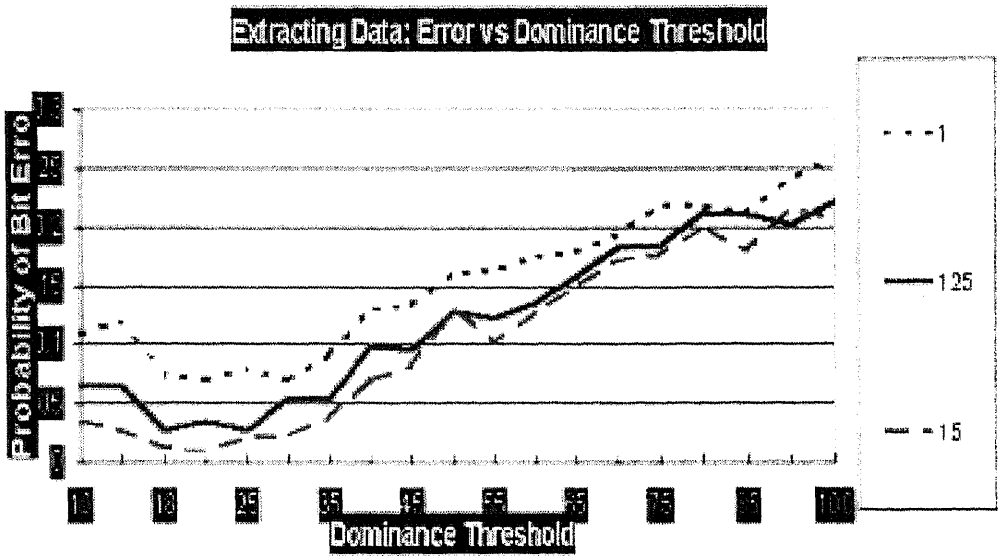

Figure 5.40 SPIHT DCT - 128x128: Testing Various Majority Sign Thresholds (Part 2)

5.2 Performance Results with Compression

In this section we present a thorough testing of DCT and DFT schemes subjected to JPEG, SPIHT and MPEG-2 compression schemes.

5.2.1 JPEG Grayscale Image Tests

Method I was used to hide a 64-bit long binary signature inside a grayscale (pgm) image, using a majority sign threshold (dominance threshold) constant set to 20. JPEG compression was then applied to the image, followed by decompression back into a pgm format. This procedure was repeated for:

- 7 black-and-white 256x256 images: Baboon, Lena, Tree, Girl, Splash, Tiffany and Peppers,
- 10 different signatures,
- 5 different DCT block sizes: 8x8, 16x16, 32x32, 64x64, 128x128,
- 10 various JPEG compression quality factors: 10, 20, 30, 40, ..., 100.

For image quality comparisons, see Figures 5.41, 5.42, 5.43. They display the original, unwatermarked image, next to the DHS1 watermarked image. Note, that the decrease in quality of the images due to the watermarking operation is minimal (there is no compression applied to these pictures).

Results of the comprehensive testing procedure described above can be seen in Figures 5.44, 5.45.

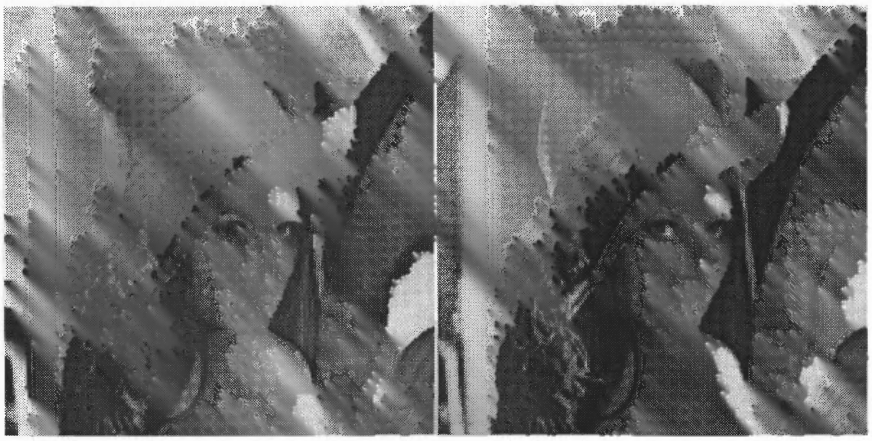

Figure 5.41 Original (left) and DHS1 Watermarked (right) 256x256 *pgm* LENA Image

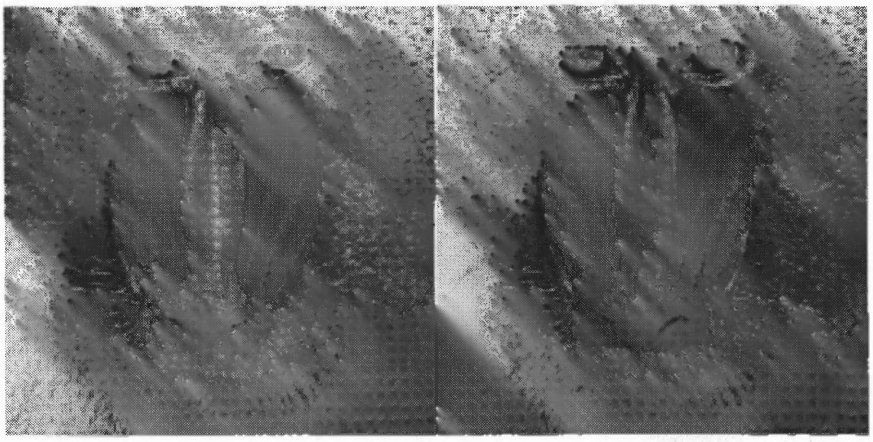

Figure 5.42 Original (left) and DHS1 Watermarked (right) 256x256 *pgm* BABOON Image

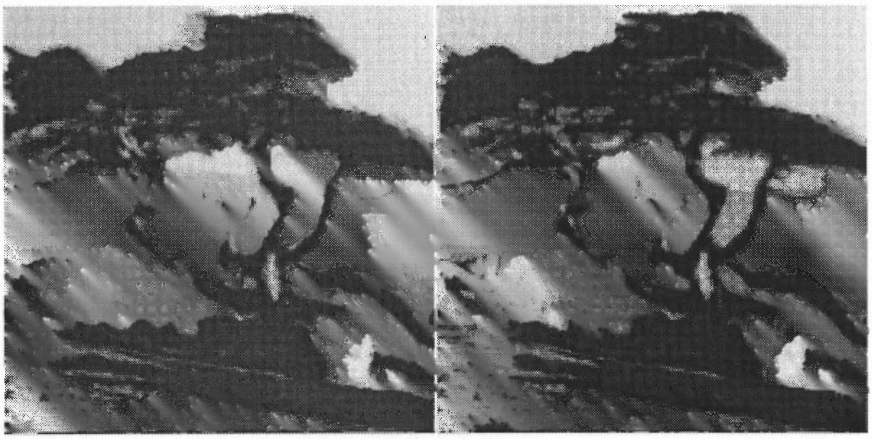

Figure 5.43 Original (left) and DHS1 Watermarked (right) 256x256 *pgm* TREE Image

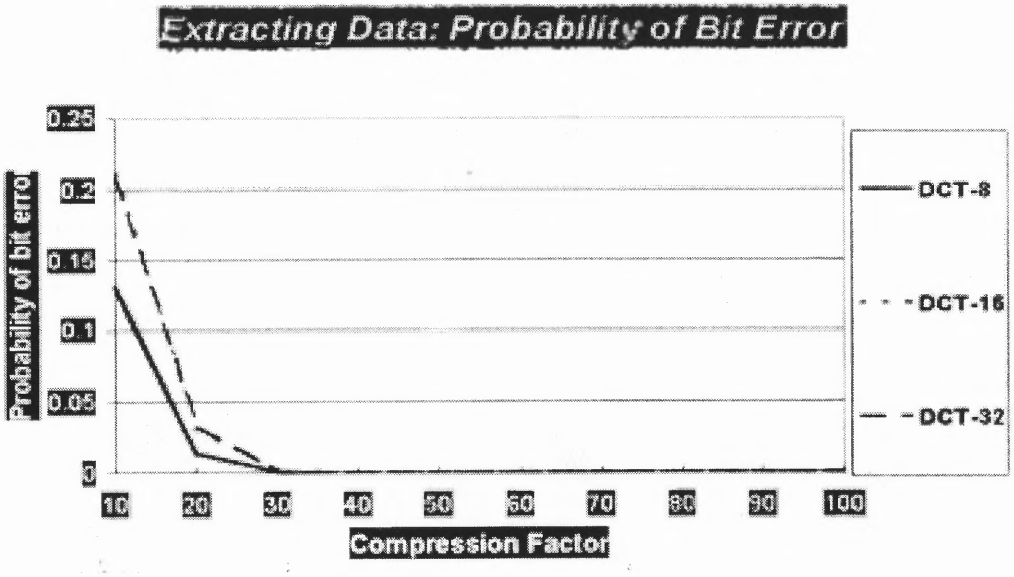

Figure 5.44 Probability of Bit Error vs. Different JPEG Quality Values for 8x8, 16x16, 32x32 DCT Block Sizes

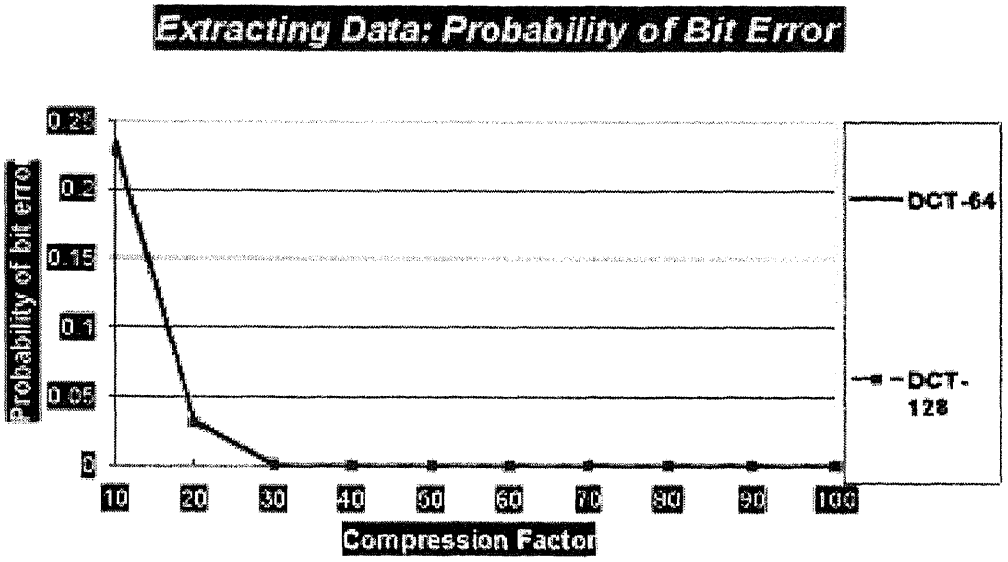

Figure 5.45 Probability of Bit Error vs. Different JPEG Quality Values for 64x64, 128x128 DCT Block Sizes

The same experiment was conducted for the data hiding Method II. In this case we used:

- 13 black-and-white 256x256 images: Airplane, Baboon, Lena, Tree, Girl, Girl2, Girl3, Couple, Splash, Tiffany, House, Sailboat and Peppers,
- 11 different signatures,
- 10 various JPEG compression quality factors: 10, 20, 30, 40, ..., 100.

For image quality comparisons, see Figures 5.46, 5.47, 5.48. They show the original, unwatermarked image, next to the DHS5 watermarked image. Note, that the decrease in quality of the images due to the watermarking operation is minimal (there is no compression applied to these pictures).

For error probability rates under DHS5 see Figure 5.49.

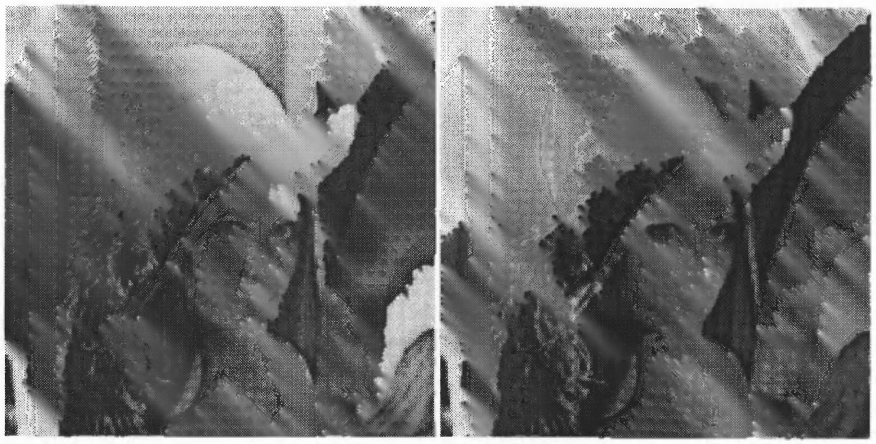

Figure 5.46 Original (left) and DHS5 Watermarked (right) 256x256 *pgm* LENA Image

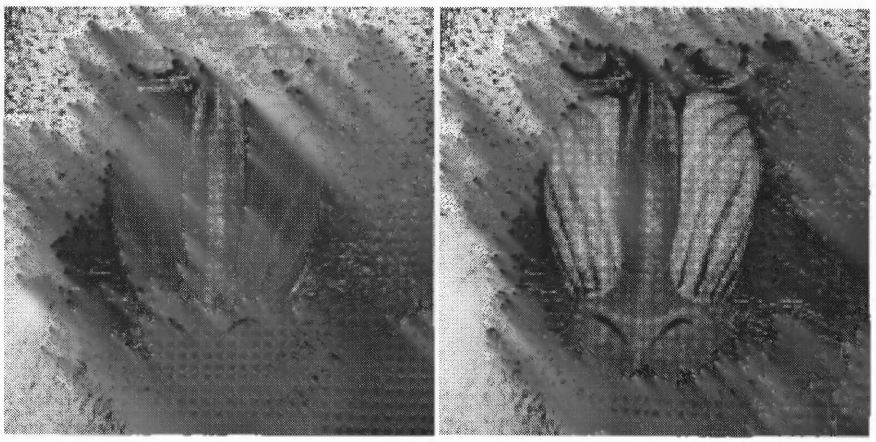

Figure 5.47 Original (left) and DHS5 Watermarked (right) 256x256 *pgm* BABOON Image

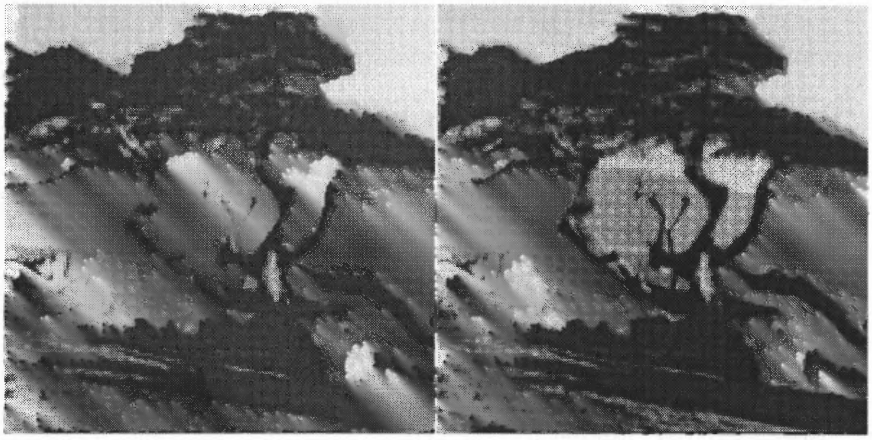

Figure 5.48 Original (left) and DHS5 Watermarked (right) 256x256 *pgm* TREE Image

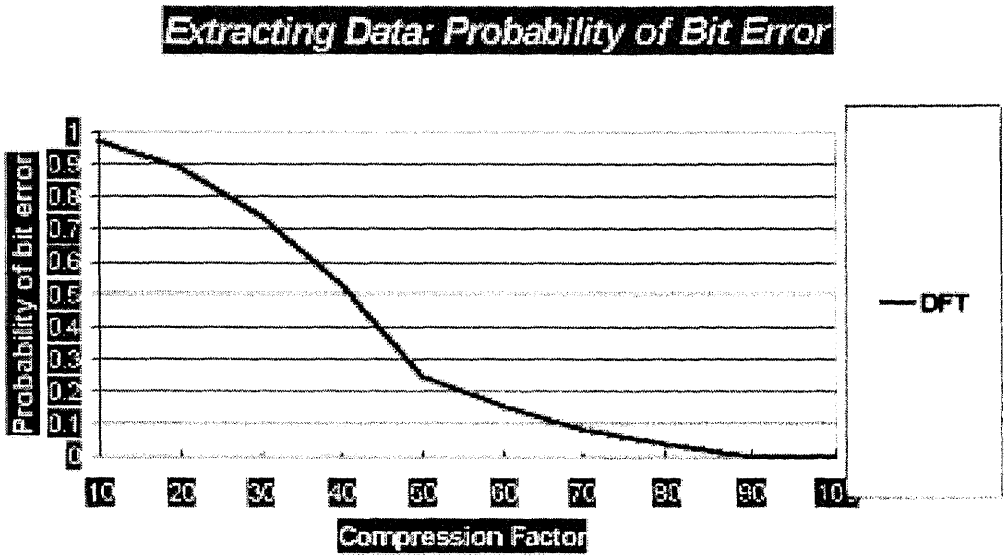

Figure 5.49 Probability of Bit Error vs. Different JPEG Quality Values for DFT Data Hiding Scheme

From the graphs it can be seen that Method I performs much better than Method II when the images are subjected to JPEG compression, irrespective of the quality factor.

5.2.2 SPIHT Grayscale Image Tests

Similar experiment was conducted for SPIHT compression using DHS1 (majority sign threshold constant set to 20). We used:

- 7 black-and-white 256x256 images: Baboon, Lena, Tree, Girl, Splash, Tiffany and Peppers,
- 10 different signatures,
- 5 different DCT block sizes: 8x8, 16x16, 32x32, 64x64, 128x128,
- 6 various SPIHT bitrates: 0.25, 0.50, 0.75, 1.00, 1.25, 1.50.

For different DCT block sizes the results are shown in Figures 5.50, 5.51.

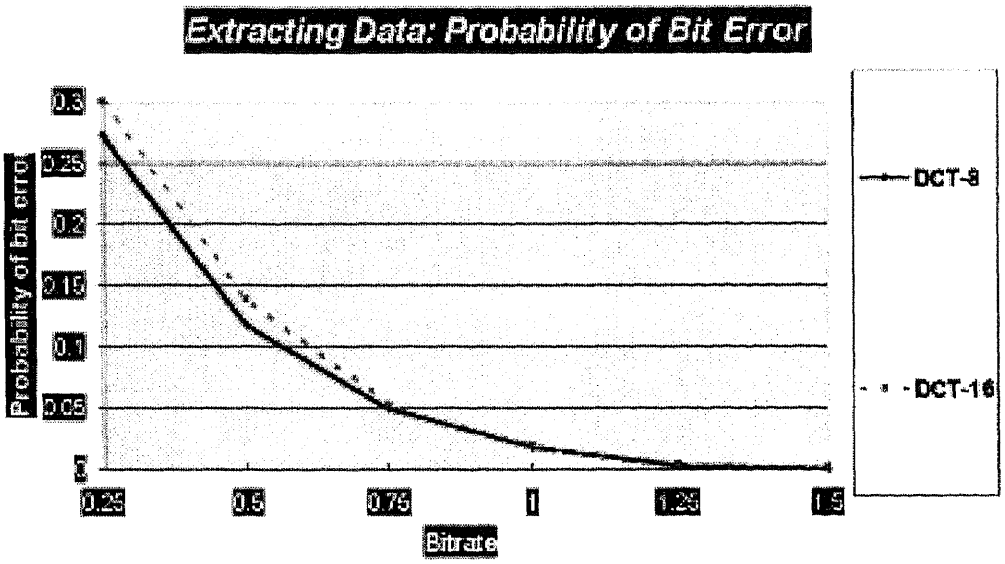

Figure 5.50 Probability of Bit Error vs. Different SPIHT Bitrates for 8x8, 16x16 DCT Block Sizes

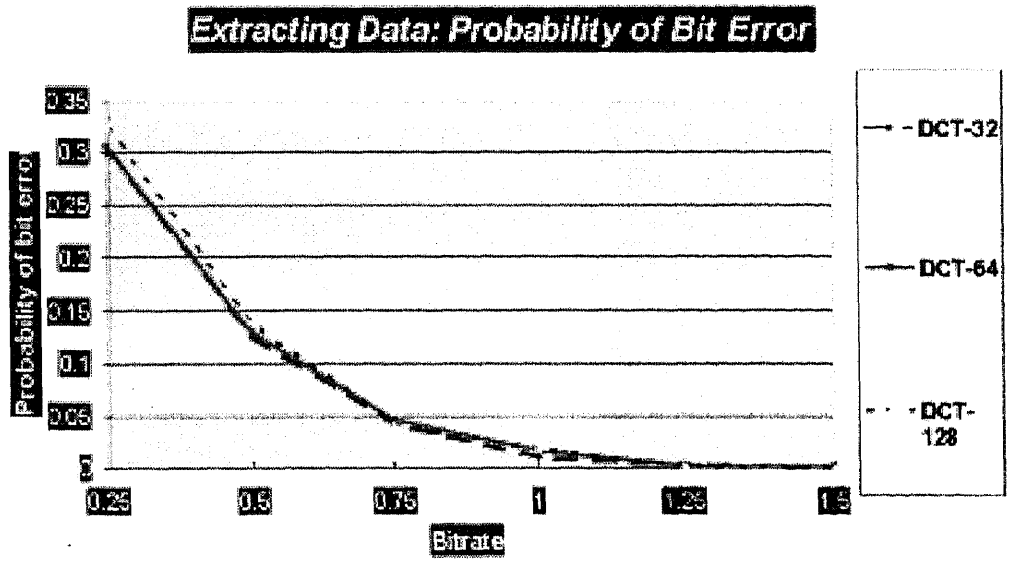

Figure 5.51 Probability of Bit Error vs. Different SPIHT Bitrates for 32x32, 64x64, 128x128 DCT Block Sizes

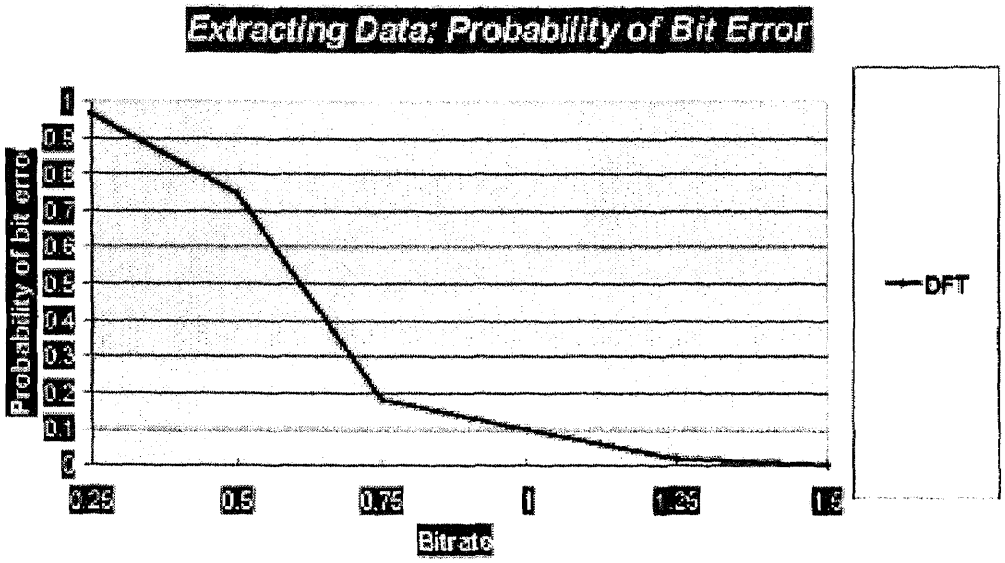

Figure 5.52 Probability of Bit Error vs. Different SPIHT Bitrates for DFT Data Hiding Scheme

Then, the procedure was repeated for Method II data embeded images, for:

- 13 black-and-white 256x256 images: Airplane, Baboon, Lena, Tree, Girl, Girl2, Girl3, Couple, Splash, Tiffany, House, Sailboat and Peppers,
- 11 different signatures,
- 6 various SPIHT bitrates: 0.25, 0.50, 0.75, 1.00, 1.25, 1.50.

Results are depicted in Figure 5.52.

Method I shows again an advantage over Method II, however, at 1.25 and 1.50 bitrates both schemes perform equally well.

5.2.3 JPEG Color Image Tests

JPEG compression test was conducted on Method I hidden data inside color (*ppm*) pictures using:

- 7 color 256x256 images: Baboon, Lena, Tree, Girl, Splash, Tiffany and Peppers,

- 10 64-bit long binary different signatures,
- 5 different DCT block sizes: 8x8, 16x16, 32x32, 64x64, 128x128,
- 10 various JPEG compression quality factors: 10, 20, 30, 40, ..., 100.

Original images next to Method I modified ones can be seen in Figures 5.53, 5.54, 5.55. They show the original, unwatermarked image, next to the DHS1 watermarked image.

Results of the comprehensive tests can be seen in Figures 5.56, 5.57.

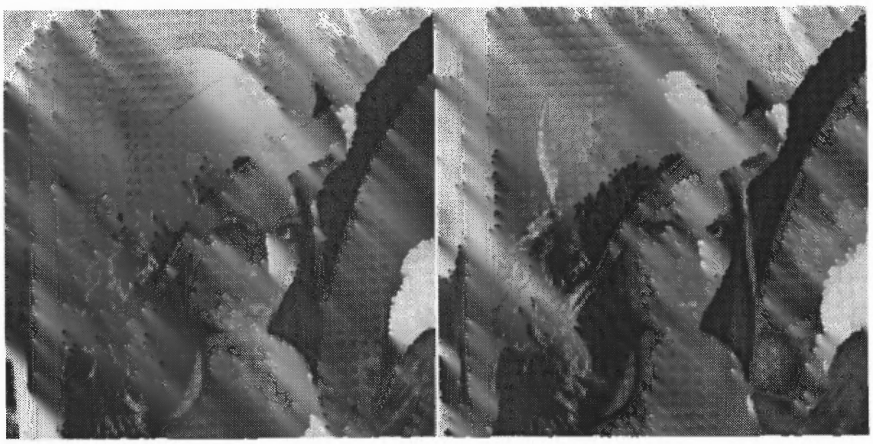

Figure 5.53 Original (left) and DHS1 Watermarked (right) 256x256 *ppm* LENA Image

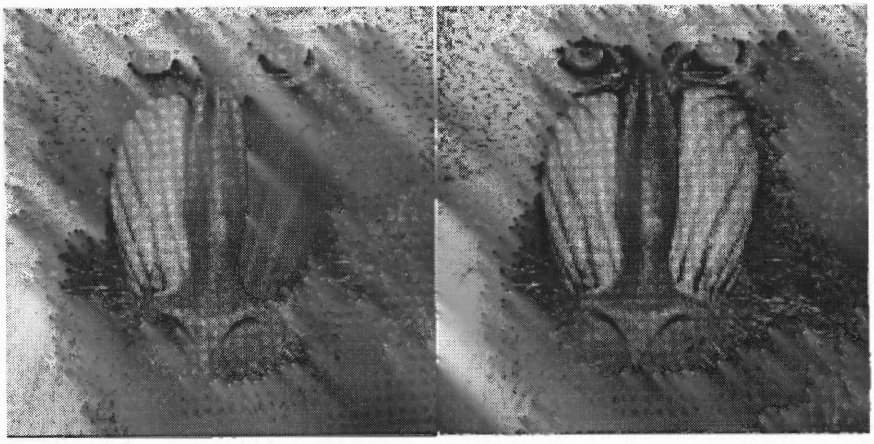

Figure 5.54 Original (left) and DHS1 Watermarked (right) 256x256 *ppm* BABOON Image

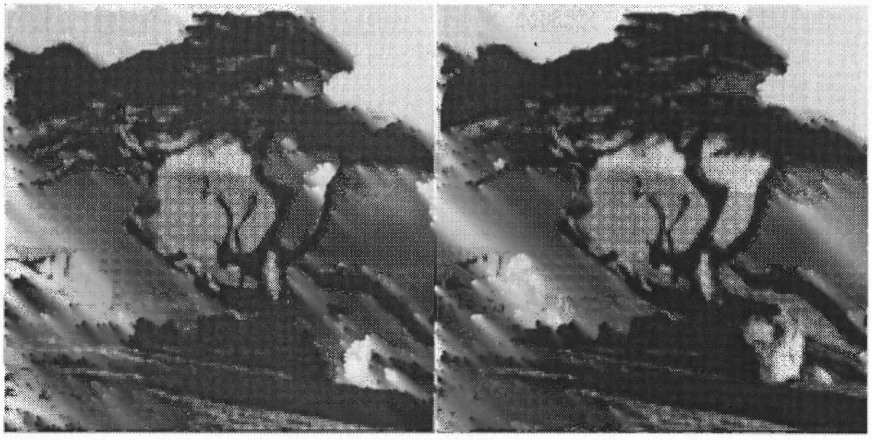

Figure 5.55 Original (left) and DHS1 Watermarked (right) 256x256 *ppm* TREE Image

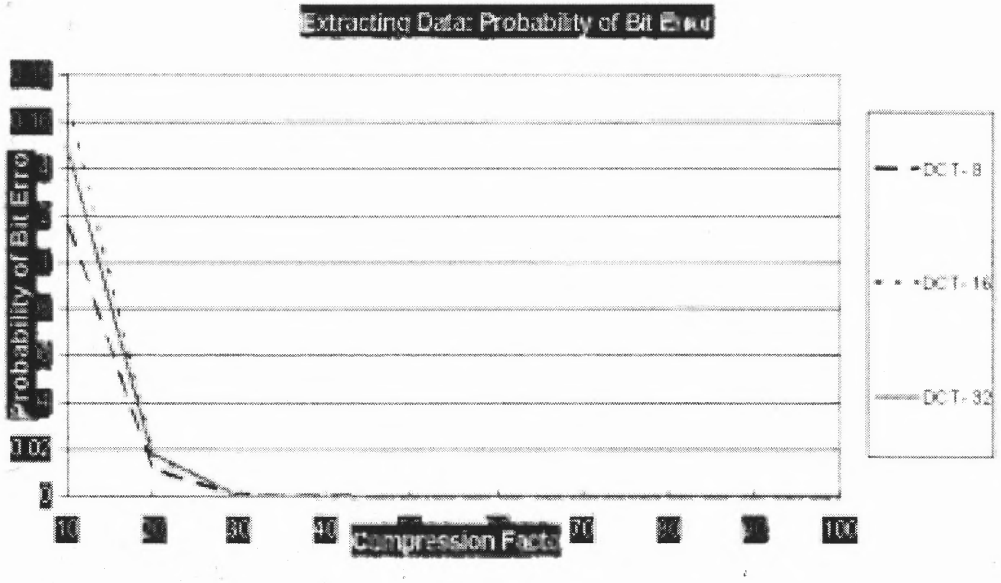

Figure 5.56 Probability of Bit Error vs. Different JPEG Quality Values for 8x8, 16x16, 32x32 DCT Block Sizes

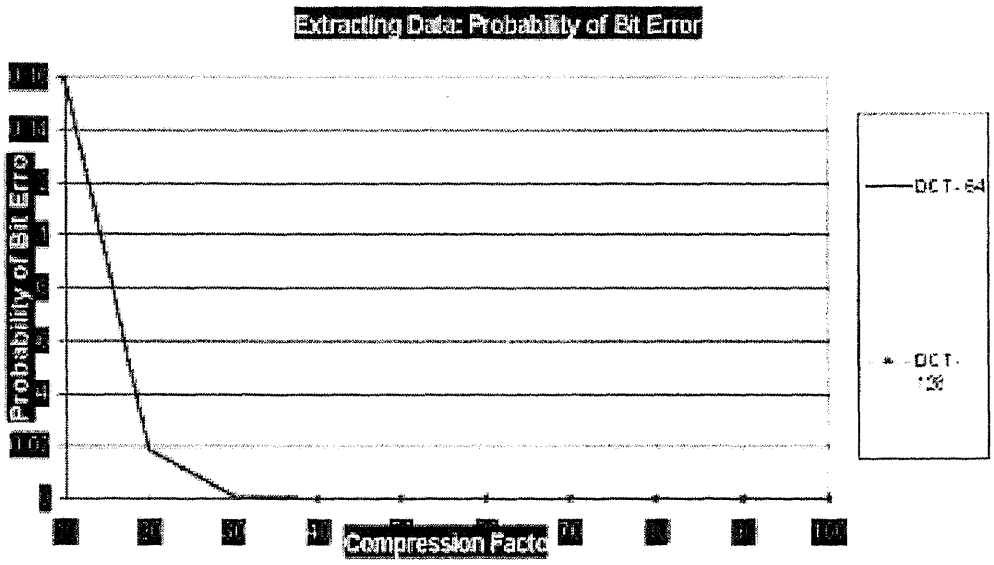

Figure 5.57 Probability of Bit Error vs. Different JPEG Quality Values for 64x64, 128x128 DCT Block Sizes

The test was repeated for Method II images:

- 13 color 256x256 images: Airplane, Baboon, Lena, Tree, Girl, Girl2, Girl3, Couple, Splash, Tiffany, House, Sailboat and Peppers,
- 11 different signatures,
- 10 various JPEG compression quality factors: 10, 20, 30, 40, ..., 100.

For quality reassurance, see the original and DHS5 modified images in Figures 5.58, 5.59, 5.60.

The probability of error test outcomes are portrayed in Figure 5.61.

DCT method performs much better than the DFT method.

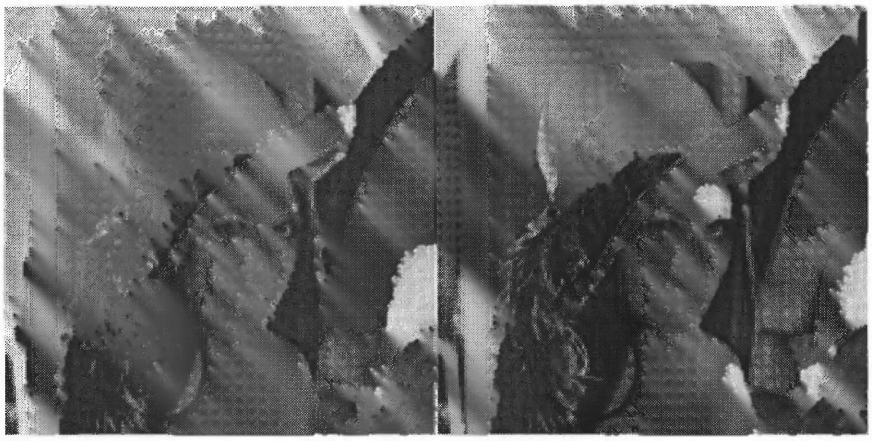

Figure 5.58 Original (left) and DHS5 Watermarked (right) 256x256 *ppm* LENA Image

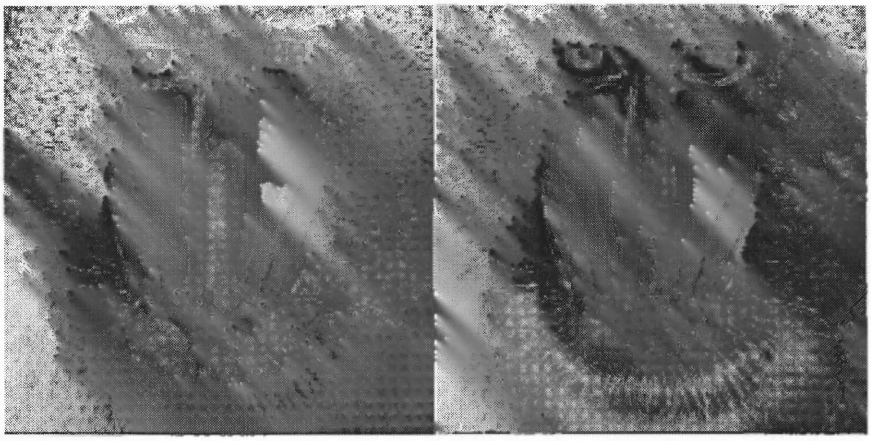

Figure 5.59 Original (left) and DHS5 Watermarked (right) 256x256 *ppm* BABOON Image

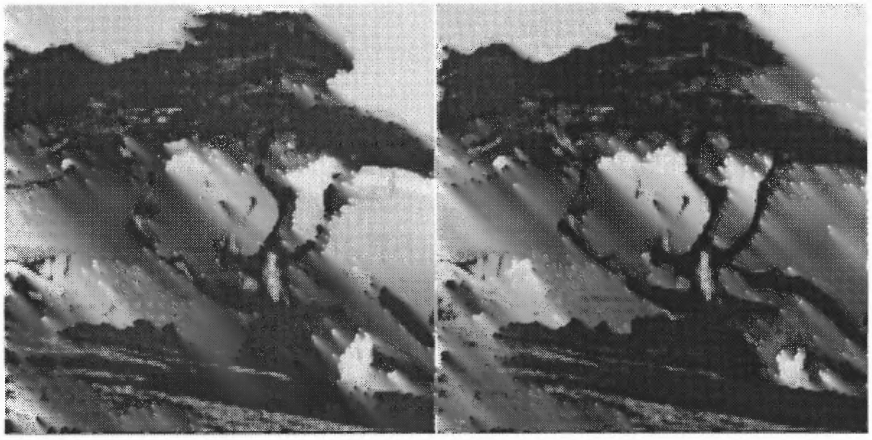

Figure 5.60 Original (left) and DHS5 Watermarked (right) 256x256 *ppm* TREE Image

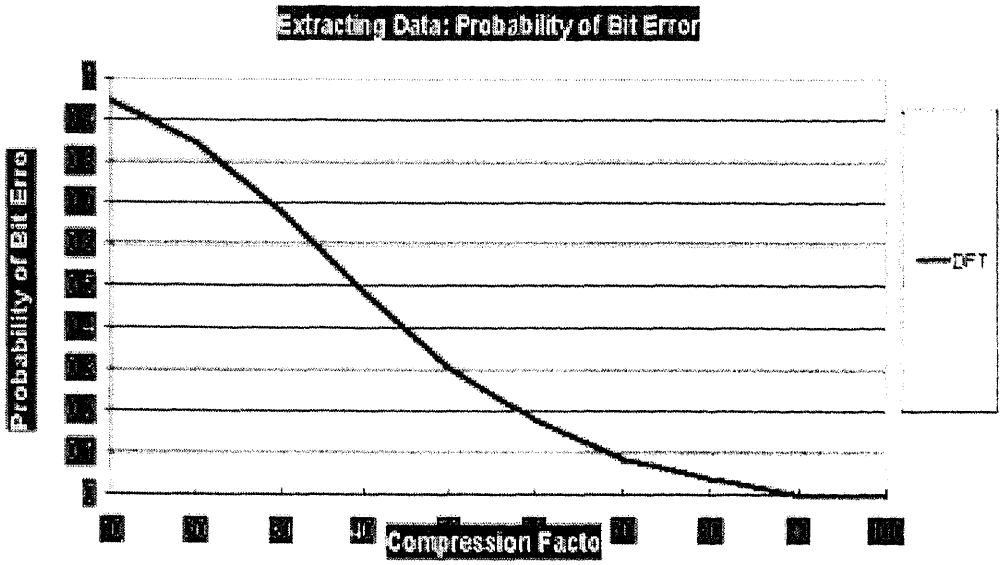

Figure 5.61 Probability of Bit Error vs. Different JPEG Quality Values for DFT Method

5.2.4 SPIHT Color Image Tests

Finally, the test was done for SPIHT compression for color images using DHS1 (majority sign threshold constant set to 25). We used:

- 7 color 256x256 images: Baboon, Lena, Tree, Girl, Splash, Tiffany and Peppers,
- 10 different 64-bit long binary signatures,
- 5 different DCT block sizes: 8x8, 16x16, 32x32, 64x64, 128x128,
- 6 various SPIHT bitrates: 0.25, 0.50, 0.75, 1.00, 1.25, 1.50.

For different DCT block sizes the results are shown in Figures 5.62, 5.63. Then, the procedure was repeated using DHS5 data embeded images, for:

- 13 color 256x256 images: Airplane, Baboon, Lena, Tree, Girl, Girl2, Girl3, Couple, Splash, Tiffany, House, Sailboat and Peppers,
- 11 different signatures,
- 6 various SPIHT bitrates: 0.25, 0.50, 0.75, 1.00, 1.25, 1.50.

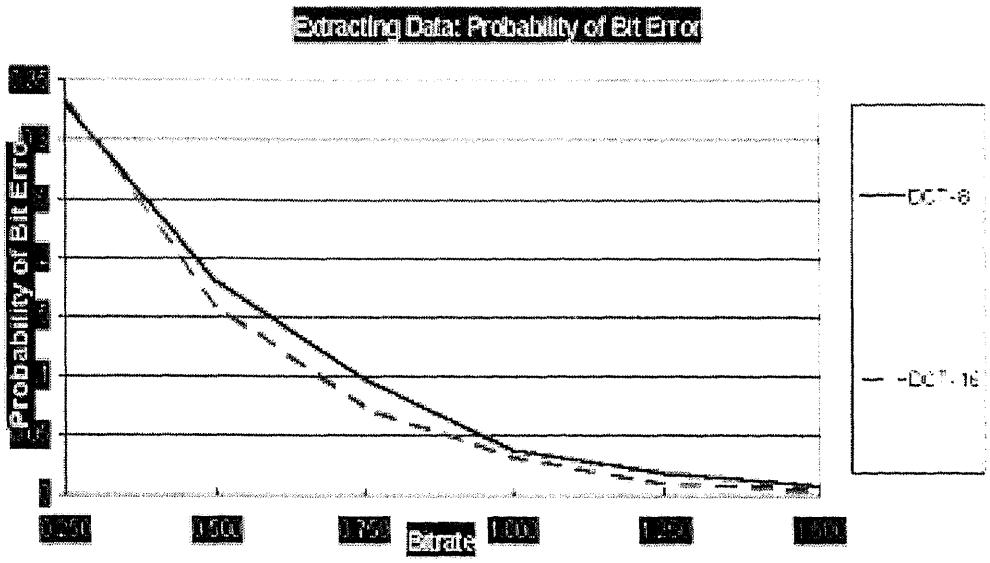

Figure 5.62 Probability of Bit Error vs. Different SPIHT Bitrates for 8x8, 16x16 DCT Block Sizes

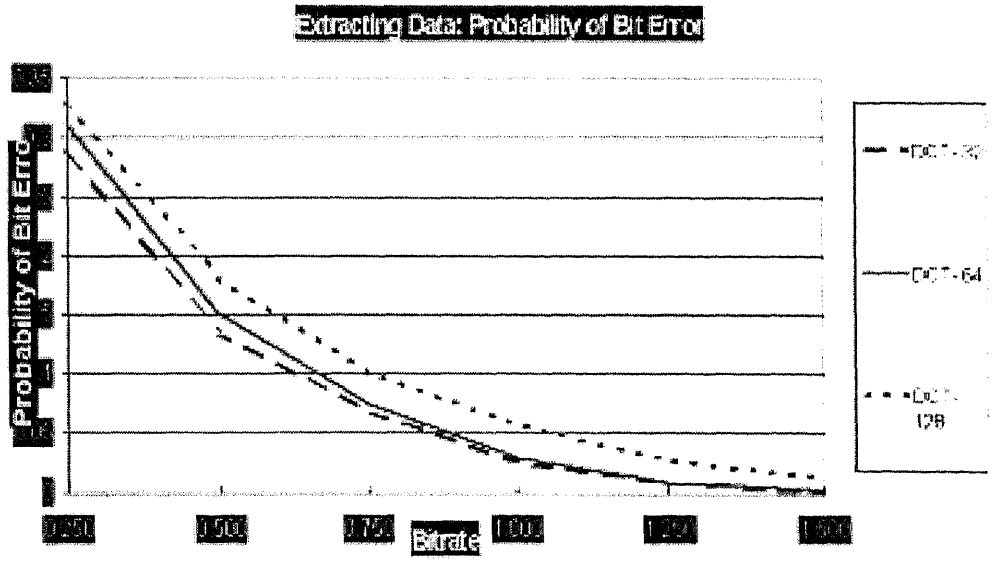

Figure 5.63 Probability of Bit Error vs. Different SPIHT Bitrates for 32x32, 64x64, 128x128 DCT Block Sizes

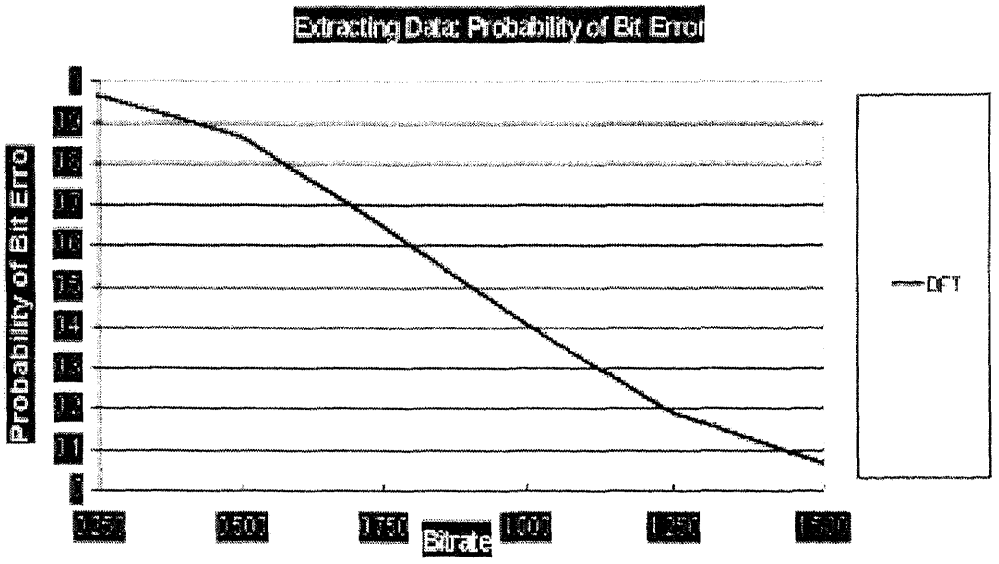

Figure 5.64 Probability of Bit Error vs. Different SPIHT Bitrates for DFT Data Hiding Scheme

Results are portrayed in Figure 5.64.

DHS1 is clearly better than DHS5 when it comes to the probability of error.

5.2.5 MPEG-2 Tests for Method I

Two movie sequences *Coastguard* and *Mother* 300 frames each were used for data hiding using DHS1. Each frame in either sequence is of 176x144 size. It was decided to try to hide 25 bits in such frame. Later the experiment was repeated for steganography of 35 bits in a frame. Using different majority sign threshold (MST) (dominance threshold) constants the outcomes are compared in Table 5.1. Table 5.2 gives the same information as Table 5.1, however this time the exact number of error bits is given. This should highlight the robustness of DCT data hiding method to MPEG-2 compression.

Sample original and watermarked frames can be seen in Figures 5.65, 5.66. Figure 5.67 shows the results of Tables 5.1 and 5.2 in a graphical format.

Clearly, the more bits per frame we try to hide the higher the probability of an error. Bigger MST means less errors since DCT coefficients are altered more,

Table 5.1 Probability of Bit Error after MPEG-2 Using DHS1

	coastguard		*mother*	
MST	No. of bits hidden per frame		No. of bits hidden per frame	
	25	35	25	35
5	0.0375	0.0482	0.0060	0.0062
10	0.0019	0.0016	0.0011	0.0009
15	0.0008	0.0010	0.0009	0.0007
20	0.0005	0.0007	0.0008	0.0006

however, the bigger MST means lower picture quality. Nonetheless, at MST equal to 20 and bitrates of 25 and 35 bits per frame the picture quality is good and when combined into a movie the interference associated with hidden data is insignificant.

Table 5.2 Exact Number of Bit Errors after MPEG-2 Using DHS1

	coastguard		*mother*	
MST	No. of bits hidden per frame		No. of bits hidden per frame	
	25	35	25	35
5	281	506	45	65
10	14	17	8	9
15	6	11	7	7
20	4	7	6	6
Total number of bits hidden in 300 frames:	7500	10500	7500	10500

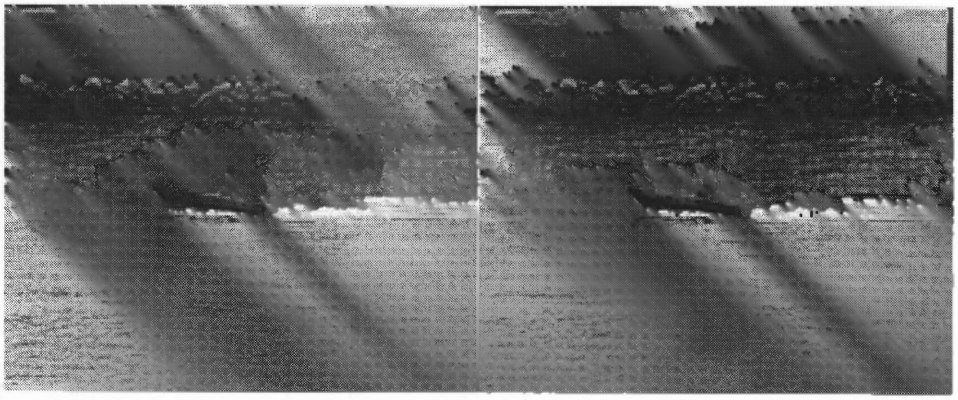

Figure 5.65 Original (left) and DHS1 Watermarked (right) 176x144 *.y.u.v* COASTGUARD Video Frame 0 (no. of bits in a frame = 25, MST = 20)

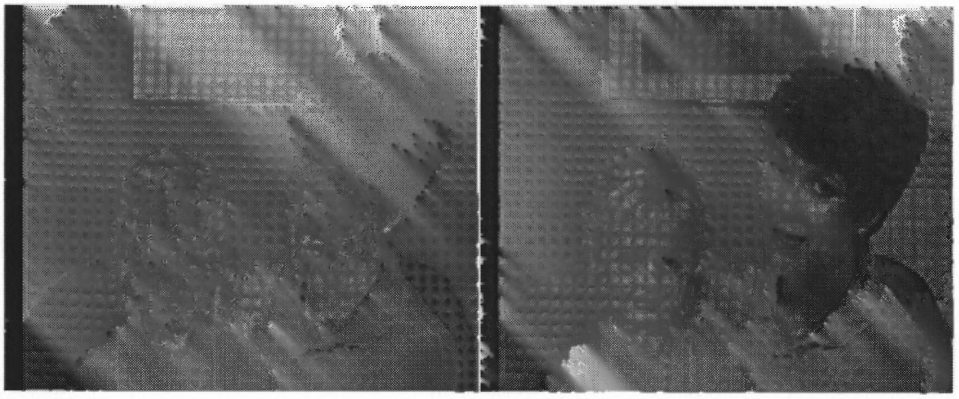

Figure 5.66 Original (left) and DHS1 Watermarked (right) 176x144 *.y.u.v* MOTHER Video Frame 0 (no. of bits in a frame = 25, MST = 20)

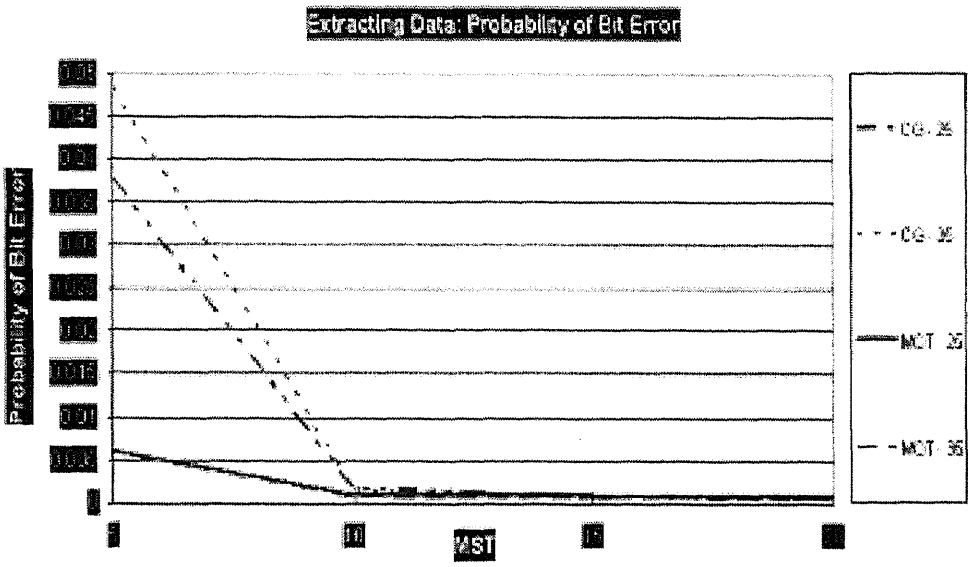

Figure 5.67 Probability of Bit Error vs. MST Constant for: CG - 25 (coastguard with 25 bits/frame), CG - 35 (coastguard with 35 bits/frame), MOT - 25 (mother with 25 bits/frame), MOT - 35 (mother with 35 bits/frame)

5.3 Data Survival under Common Signal Processing Operations

In this section we investigate the survival of hidden data after various image editing operations are performed on the data carrying image. We utilize commonly available Paint Shop Pro Version 5.01 as an image altering tool, since software like this will probably be an instrument of a hacker trying to destroy a watermark or hidden data. All the tests are performed on a color (*ppm*) 256x256 LENA picture. DCT and DFT methods are used and the data survival results are tabulated beside each other.

5.3.1 Resizing

Data is hidden in a 256x256 LENA picture which is then resized to the indicated dimensions. The new image is saved and then resized back into the original 256x256 size which is again saved. Due to these operations the new LENA's quality is worse than that of the original image, see Figures 5.68, 5.69. Results are shown in Table 5.3.

Figure 5.68 Method I: Watermarked (left) and after Resizing (right) to 100x100 and Back to 256x256, LENA Image

Figure 5.69 Method II: Watermarked (left) and after Resizing (right) to 100x100 and Back to 256x256, LENA Image

Table 5.3 Resizing: Probability of Bit Error

Resize to [pixels]	Method I	Method II
200x200	0.000	0.000
180x180	0.000	0.000
200x128	0.000	0.500
150x150	0.000	0.643
128x128	0.000	0.857
125x125	0.000	1.000
100x100	0.000	1.000
100x80	0.125	1.000
100x50	0.875	1.000
50x50	1.000	1.000

5.3.2 Rotation

LENA with hidden data was subjected to rotation after which data was tried to be extracted from the picture. Influence of rotation on data is depicted in Table 5.4.

Table 5.4 Rotation: Probability of Bit Error

Rotate rightwards [degrees]	Method I	Method II
0.01	0.000	0.000
0.10	0.000	0.000
0.50	1.000	1.000
1.00	1.000	1.000

5.3.3 Negative Image

The image was negated Figures 5.70, 5.71. Interestingly, for Method I all bits were wrong, but for Method II they were all correctly detected.

5.3.4 Text Superimposed over an Image

The image was altered by addition of a text on top of it, Figure 5.72,. All data under both steganographic schemes was correctly detected. More tests showed that

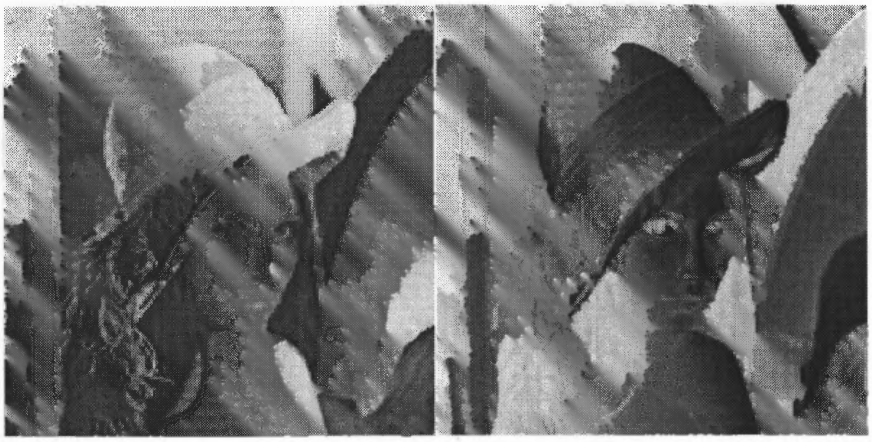

Figure 5.70 Method I: Watermarked (left) and Negated (right) LENA Image

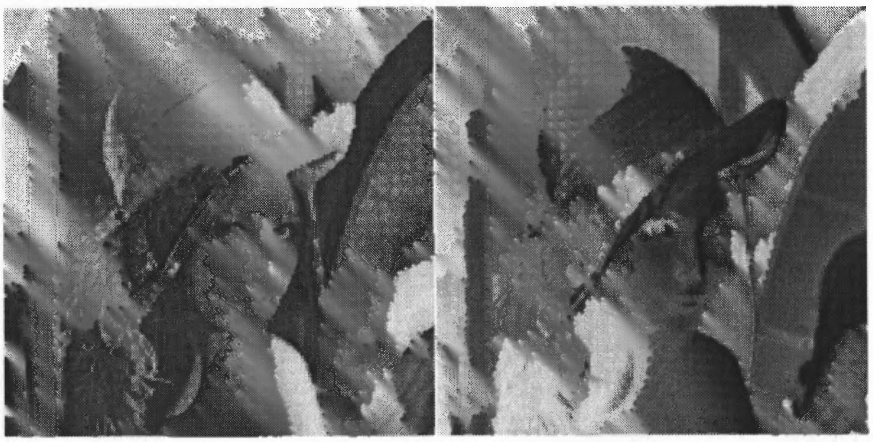

Figure 5.71 Method II: Watermarked (left) and Negated LENA Image

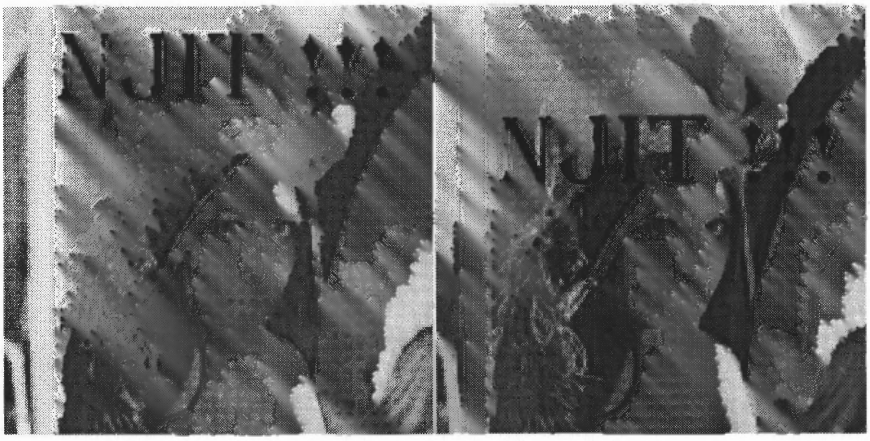

Figure 5.72 Method I Text Altered Image (left) and Method II Text Altered Image (right)

the schemes are immune to any alterations that involve additions of text to the picture.

5.3.5 Random and Uniform Noise Addition

Varying degrees of random (Figure 5.73) and uniform (Figure 5.74) noise were added to LENA. Both schemes are more immune to uniform noise than random one, however, overall, DCT method exhibits higher tolerance to noise than the DFT method.

5.3.6 Sharpen and Edge Enhancement

Either method's image when submitted to Paint Shop Pro's sharpening, sharpening more (Figure 5.75), edge enhancement and edge enhancement more (Figure 5.76) produces no errors upon information detection.

5.3.7 Blur and Gaussian Blur

For blur and more blur results of the experimentation can be seen in Table 5.5. Figure 5.77 compares an effect of varying radius in Gaussian blur (Figure 5.78) on

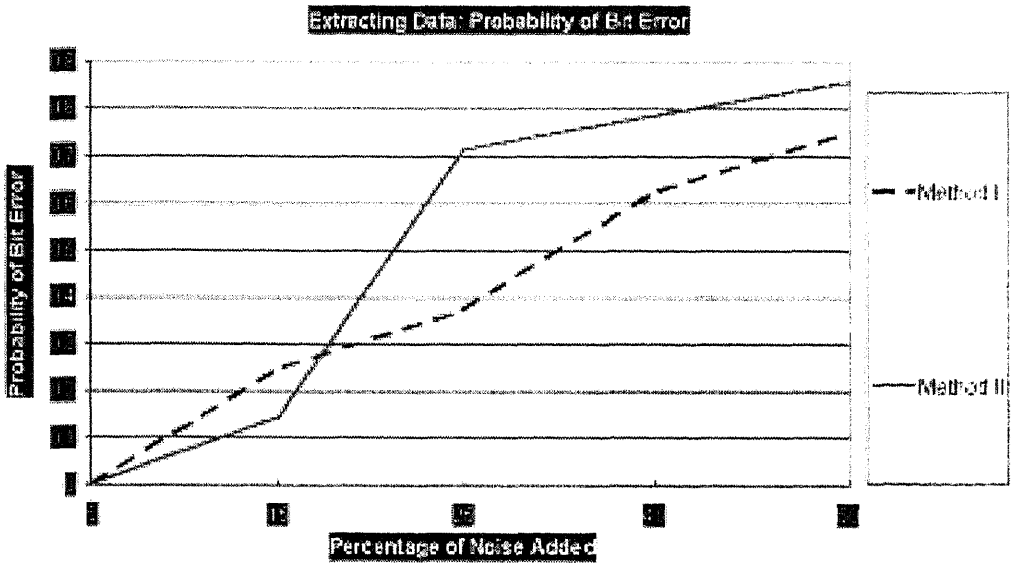

Figure 5.73 Influence of Random Noise Addition on Probability of Bit Error

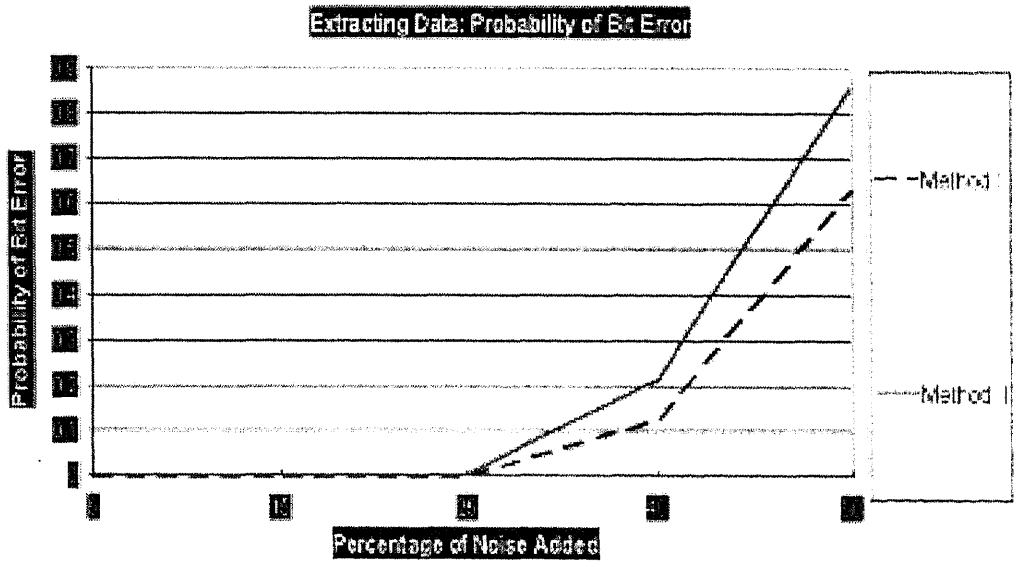

Figure 5.74 Influence of Uniform Noise Addition on Probability of Bit Error

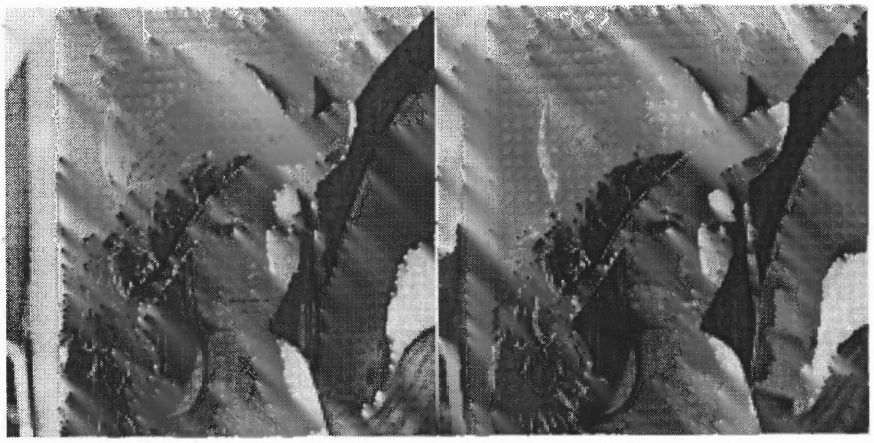

Figure 5.75 Method I (left) and Method II (right) Sharpened More Image

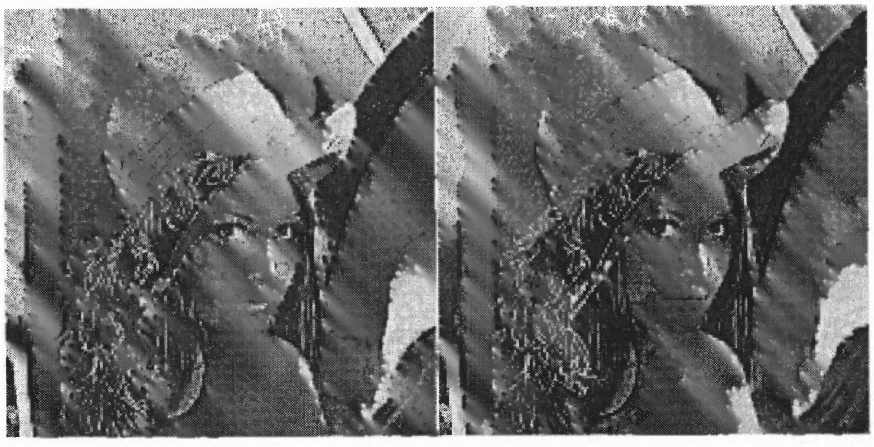

Figure 5.76 Method I (left) and Method II (right) Edge Enhanced More Image

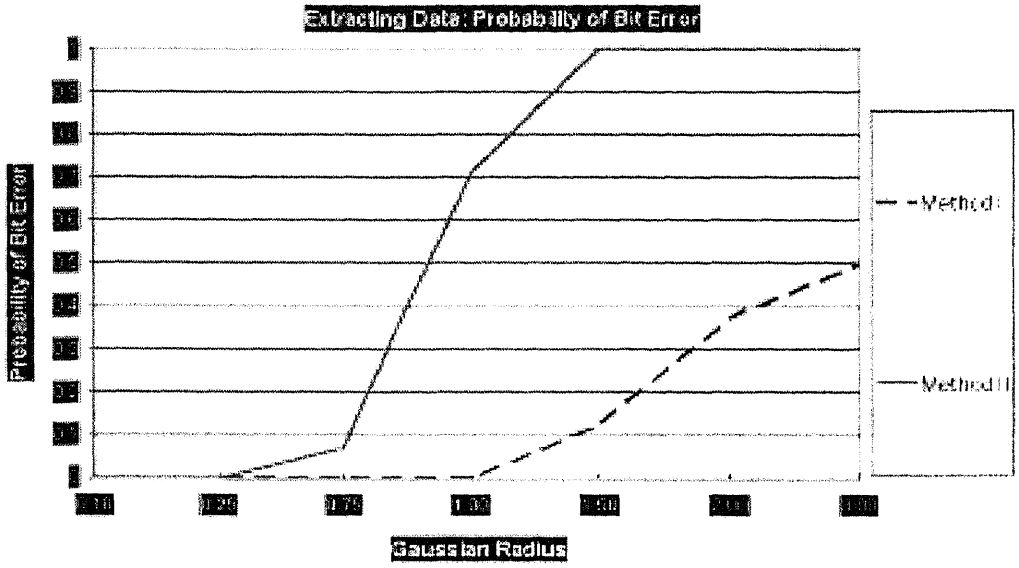

Figure 5.77 Effect of Changing Radius in Gaussian Blur on Bit Error Rate

bit error rates for the two methods. Yet again, the first method displays superiority of data preservation over the DFT technique.

5.3.8 Soften, Median Cut, Erode and Emboss

LENA images with Scheme I and II embedded data were softened, softened more, subjected to a median filter, eroded (Figure 5.79) and embossed (Figure 5.80). Then, the data was attempted to be extracted. All the results were tabulated in Table 5.6. Interestingly, Method II is totally immune to embossing, while Method I gives all errors. As expected, data hidden with the first method survives the other operations much better than the DFT technique's data.

Table 5.5 Blur and More Blur: Probability of a Bit Error

Type of Blur	Method I	Method II
blur	0.125	0.143
blur more	0.500	1.000

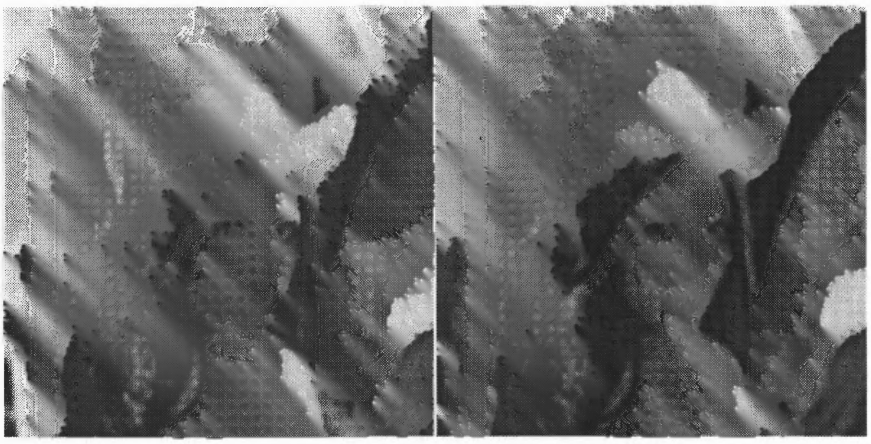

Figure 5.78 Method I (left) and Method II (right) Gaussian Blur with Radius = 1.5

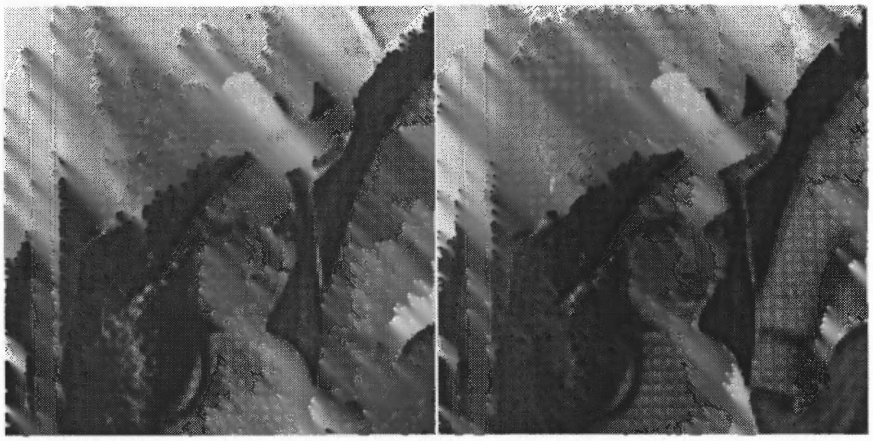

Figure 5.79 Method I (left) and Method II (right) Images Subject to Eroding

Figure 5.80 Method I (left) and Method II (right) Images Subject to Embossing

Table 5.6 Other Filters: Probability of a Bit Error

Type of filter	Method I	Method II
soften	0.000	0.143
soften more	0.000	1.000
median cut	0.000	0.786
erode	0.000	1.000
emboss	1.000	0.000

5.3.9 Histogram Operations and Gamma Correction

Histogram equalization (Figure 5.81), histogram stretching and gamma corrections:

- red = 3.70, green = 3.70, blue = 3.70 (Figure 5.82),
- red = 2.20, green = 2.20, blue = 2.20,
- red = 0.60, green = 0.60, blue = 0.60,

do not distort the hidden data at all.

Figure 5.81 Method I (left) and Method II (right) Images after Histogram Equalization

5.3.10 Manipulation of Bit Number per Channel

Starting with 8 bits per channel we decrease the bit allocation to 1 bit. The effect of this can be seen in Figures 5.83, 5.84 . There are no errors until only 1 bit is

allocated per channel in which case the DFT scheme performs much better than the DCT one (Figure 5.85).

5.3.11 Brightness

Method II is much more resistant to brightness manipulation than Method I, see Figures 5.86, 5.87, 5.88.

5.3.12 Contrast

For contrast changes the DFT scheme is totally robust, no errors, Figures 5.89, 5.90, 5.91.

Figure 5.82 Method I (left) and Method II (right) Images after Gamma Correction (red = 3.70, green = 3.70, blue = 3.70)

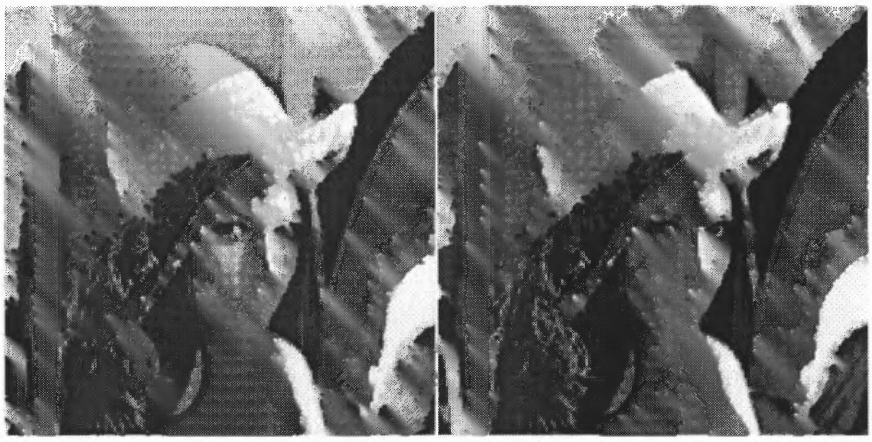

Figure 5.83 Method I (left) and Method II (right) Images with 2 Bits per Channel Allocation

Figure 5.84 Method I (left) and Method II (right) Images with 1 Bit per Channel Allocation

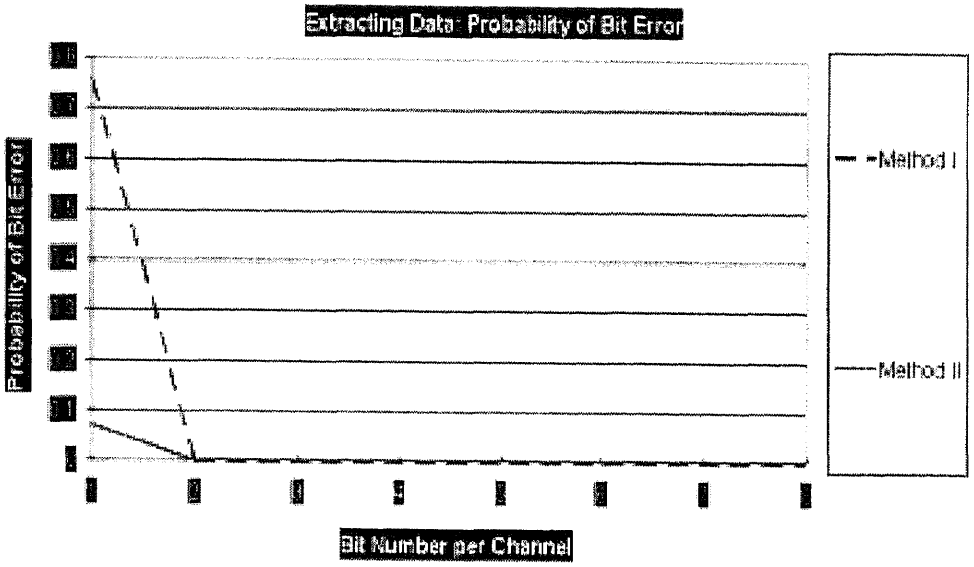

Figure 5.85 Effect of Channel Bit Number on Probability of Error

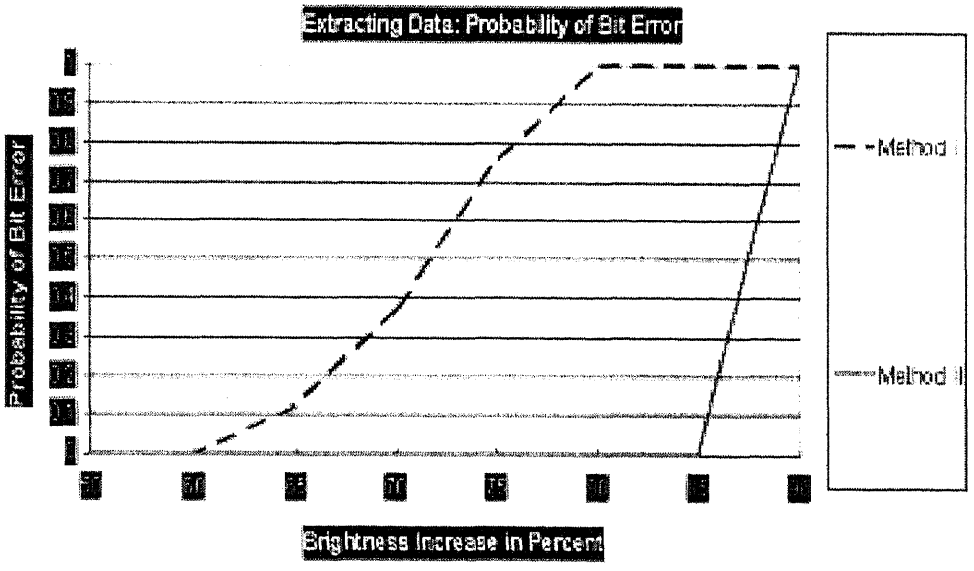

Figure 5.86 Effect of Brightness Increase on Probability of Error

Figure 5.87 Method I (left) and Method II (right) Images at Brightness +50%

Figure 5.88 Method I (left) and Method II (right) Images at Brightness +75%

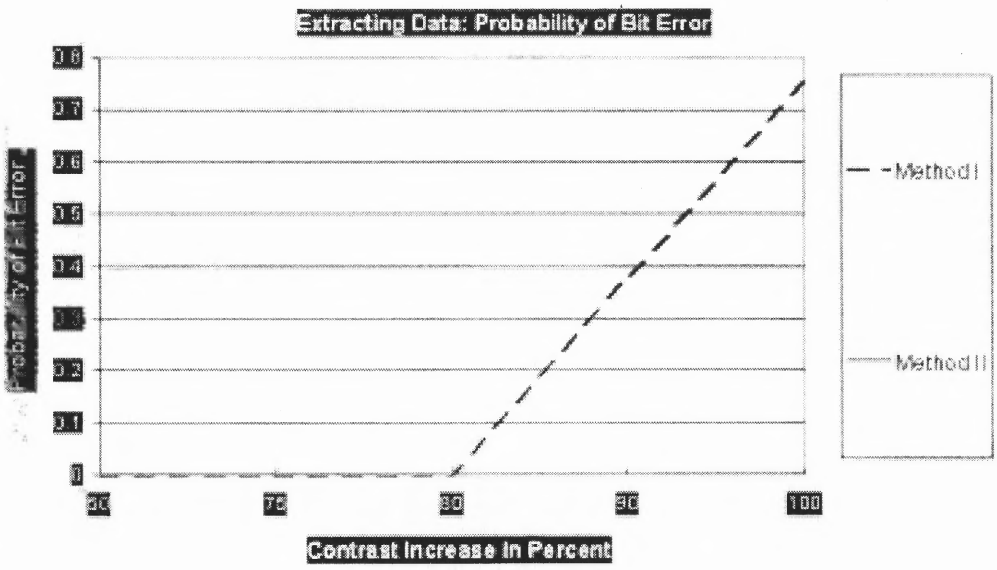

Figure 5.89 Effect of Contrast Increase on Probability of Error

Figure 5.90 Method I (left) and Method II (right) Images at Contrast +75%

Figure 5.91 Method I (left) and Method II (right) Images at Contrast +100%

CHAPTER 6

CONCLUSION

In today's world more and more concern is being raised over intellectual property protection. The companies are investigating means of effective enforcement of copyrights that would stand in the court of law. Like in other areas, digital picture and movie owners must somehow guard themselves against stolen and unauthorized distribution of their works. This thesis studies the performance of two methods suitable for this task. In addition, a full software implementation of the presented algorithms is performed as a part of this study. The listings of the programs can be found in the appendices. The watermarking schemes were tested and their performance interpreted to provide a good understanding of their capabilities.

Both watermarking techniques preserve the original image quality with minor exceptions. Their algorithms are complex enough to be virtually unbreakable by an uninformed party. And most importantly, they offer a good degree of tamper resistance for someone attempting to modify the image with intent to destroy the watermark. Here, an assumption is made that during the picture modification, the hacker does not want to destroy the image's commercial value. Since, it's trivial to destroy the whole image, by which operation the watermark will be erased, too.

6.1 Capacity Estimates

Method I is based on a blockwise DCT coefficient manipulation. Into a 256x256 image only 64 bits of data constituting a watermark can be hidden. This means that 8 characters, 1 byte each, can be hidden in such a picture. Method II embeds data using DFT magnitudes. 14 data-words can be hidden in a picture of 256x256 size. Throughout chapter 5 these data-words were called *bits*. This is only true if we elect for them to take on values of '0' or '1', which was done for the purpose of comparison

between two data hiding schemes. However, each data-word can in fact store an integer of value between 0 and 4095 (2^{12}). This means that when in chapter 5 we spoke of an *error bit* we meant that the whole data-word is incorrect. Data-words can be used as bits, bytes, or other measures due to their bases in integers. On the other hand, data hidden by the first method are true bits, that is, they can *only* be zeros or ones. Summarizing, in the first method 64 bits or 8 bytes or 8 characters (8 bits each) can be stored inside a 256x256 picture. In Method II, 14 bits or 14 bytes or 14 characters (8 bits each) or 14 characters (12 bits each) can be hidden. Therefore, if we are smart and combine single bits (zeros and ones) into decimal numbers representing 12 bits at a time, we can in fact hide 168 bits of data in a single 256x256 image (Table 6.1). This means that the second scheme has 2.6 times more capacity than the first one. However, it should be remembered that if one data-word is incorrectly detected, we will loose all 12 bits of information. Hence, there is a trade off between capacity and *amount* of error when the detection fails for one data-word..

Table 6.1 True Capacity of Both Schemes

Unit	Method I	Method II
bit	64	14
byte	8	14
character (8 bits each)	8	14
character (12 bits each)	5.3	14

6.2 Watermark Survival

It's been shown that the first scheme survives JPEG and SPIHT much better than the second one. In fact, for JPEG 40 and up the watermarks survive errorless! For SPIHT the performance is worse, however, at bitrate 1.0 the probability of an

error is still below 0.05. For MPEG-2 the data hiding performance is superior. Method II only at JPEG 90 achieves 100% reliable data detection. Under SPIHT at bitrates 1.25 and 1.5 both techniques perform equally well. Summarizing, Scheme I is outstanding for surviving JPEG and MPEG-2 compression and slightly worse under SPIHT. Scheme II should not be used with JPEG below quality factor of 80, however it's equally effective as Method I for SPIHT at 1.25 and 1.5 bitrates.

Data embedded by Method I showed remarkable immunity to image resizing. The picture size had to be decreased by 70% for errors to appear. Only addition of more than 40% of uniform noise would introduce errors in the watermark. Method II watermarks are not as resistant to resizing as the first method's. However, under Method II embossing and image negation has zero effect on hidden data while Method I data is completely destroyed. Both schemes are equally resistant to rotations, which result in errors if they exceed 0.1 degree. Superposition of text over the images, sharpening, edge enhancement, histogram equalization, histogram stretching and gamma correction do not introduce any errors in hidden data whether the first or second data hiding scheme is used. It should be noted that Scheme I is more resistant to noise addition be it a median filter, softening or Gaussian blur. However, the second technique is more immune to contrast and brightness changes. At 80% brightness and 100% contrast the watermark was perfectly reconstructable.

6.3 Conclusion and Future Work

Careful analysis of all the collected results leads to an obvious conclusion that Method I DCT data hiding scheme would be much more suitable for watermarking, while Method II DFT scheme for data hiding. DCT technique showed noteworthy resistance to JPEG and MPEG-2 compressions, as well as its remarkable immunity to resizing, noise introduction and other common signal processing operations. DFT scheme is less resistant to JPEG and SPIHT, especially, at low bitrates, however

its data survives intact many image editing operations and its capacity is 2.6 times bigger than that of Method I.

In the future, an introduction of an error correcting scheme into the software should greatly improve data survival rates. In addition, a Web-based software package for the DCT method and a Windows-based package for the DFT method can be developed. Further capacity experiments conducted on these schemes might also increase their data hiding capacity. Finally, based on the results presented in this work new data hiding / watermarking schemes can be developed, whose performance surpasses that displayed by these two methods.

APPENDIX A

DHS1 C/C++ PROGRAM LISTINGS

C/C++ programs for DCT-based watermarking algorithm are presented here.

DHS1.cpp - main program:

```
// DHS1.cpp : Defines the class behaviors for the application.
//
#include "stdafx.h"
#include "DHS1.h"
#include "MainFrm.h"
#include "DHS1Doc.h"
#include "DHS1View.h"
#include "GDlgH8.h"
#include "GDlgD8.h"
#include "VDlgH8.h"
#include "VDlgD8.h"
#include "GDlgH64.h"
#include "GDlgD64.h"
#include "VDlgH64.h"
#include "VDlgD64.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// CDHS1App
BEGIN_MESSAGE_MAP(CDHS1App, CWinApp)
//{{AFX_MSG_MAP(CDHS1App)
ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
ON_COMMAND(ID_GRAY_8X8, OnGray8x8)
ON_COMMAND(ID_GRAY_8x8D, OnGRAY8x8D)
ON_COMMAND(ID_VIDEO_8X8, OnVideo8x8)
ON_COMMAND(ID_VIDEO_8X8D, OnVideo8x8D)
ON_COMMAND(ID_GRAY_64X64, OnGray64x64)
ON_COMMAND(ID_GRAY_64X64D, OnGray64x64D)
ON_COMMAND(ID_VIDEO_64X64, OnVideo64x64)
ON_COMMAND(ID_VIDEO_64X64D, OnVideo64x64D)
//}}AFX_MSG_MAP
// Standard file based document commands
```

```
ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
// Standard print setup command
ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
/////////////////////////////////////////////////////////////////////////////
// CDHS1App construction
CDHS1App::CDHS1App()
{
// TODO: add construction code here,
// Place all significant initialization in InitInstance
}
/////////////////////////////////////////////////////////////////////////////
// The one and only CDHS1App object
CDHS1App theApp;
/////////////////////////////////////////////////////////////////////////////
// CDHS1App initialization
BOOL CDHS1App::InitInstance()
{
AfxEnableControlContainer();
// Standard initialization
// If you are not using these features and wish to reduce the size
// of your final executable, you should remove from the following
// the specific initialization routines you do not need.
#ifdef _AFXDLL
Enable3dControls(); // Call this when using MFC in a shared DLL
#else
Enable3dControlsStatic(); // Call this when linking to MFC statically
#endif
// Change the registry key under which our settings are stored.
// TODO: You should modify this string to be something appropriate
// such as the name of your company or organization.
SetRegistryKey(_T("Local AppWizard-Generated Applications"));
LoadStdProfileSettings(); // Load standard INI file options (including MRU)
// Register the application's document templates. Document templates
// serve as the connection between documents, frame windows and views.
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
IDR_MAINFRAME,
RUNTIME_CLASS(CDHS1Doc),
RUNTIME_CLASS(CMainFrame), // main SDI frame window
```

```
RUNTIME_CLASS(CDHS1View));
AddDocTemplate(pDocTemplate);
// Parse command line for standard shell commands, DDE, file open
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);
// Dispatch commands specified on the command line
if (!ProcessShellCommand(cmdInfo))
return FALSE;
// The one and only window has been initialized, so show and update it.
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();
return TRUE;
}
////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About
class CAboutDlg : public CDialog
{
public:
CAboutDlg();
// Dialog Data
//{{AFX_DATA(CAboutDlg)
enum { IDD = IDD_ABOUTBOX };
//}}AFX_DATA
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CAboutDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
//}}AFX_VIRTUAL
// Implementation
protected:
//{{AFX_MSG(CAboutDlg)
// No message handlers
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
//{{AFX_DATA_INIT(CAboutDlg)
//}}AFX_DATA_INIT
}
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
```

```
{
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CAboutDlg)
//}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
//{{AFX_MSG_MAP(CAboutDlg)
// No message handlers
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
// App command to run the dialog
void CDHS1App::OnAppAbout()
{
CAboutDlg aboutDlg;
aboutDlg.DoModal();
}
/////////////////////////////////////////////////////////////////////////////
// CDHS1App message handlers
void CDHS1App::OnGray8x8()
{
// TODO: Add your command handler code here
CGDlgH8 grayH8Dlg;
grayH8Dlg.DoModal();

}
void CDHS1App::OnGRAY8x8D()
{
// TODO: Add your command handler code here
CGDlgD8 grayD8Dlg;
grayD8Dlg.DoModal();

}
void CDHS1App::OnVideo8x8()
{
// TODO: Add your command handler code here
CVDlgH8 videoH8Dlg;
videoH8Dlg.DoModal();

}
void CDHS1App::OnVideo8x8D()
{
```

```
// TODO: Add your command handler code here
CVDlgD8 videoD8Dlg;
videoD8Dlg.DoModal();

}
void CDHS1App::OnGray64x64()
{
// TODO: Add your command handler code here
CGDlgH64 grayH64Dlg;
grayH64Dlg.DoModal();
}
void CDHS1App::OnGray64x64D()
{
// TODO: Add your command handler code here
CGDlgD64 grayD64Dlg;
grayD64Dlg.DoModal();
}
void CDHS1App::OnVideo64x64()
{
// TODO: Add your command handler code here
CVDlgH64 videoH64Dlg;
videoH64Dlg.DoModal();
}
void CDHS1App::OnVideo64x64D()
{
// TODO: Add your command handler code here
CVDlgD64 videoD64Dlg;
videoD64Dlg.DoModal();
}
```

GDlgH8.cpp - data hiding in images subroutine: // GDlgH8.cpp : implementation file

```
//
#include "stdafx.h"
#include "DHS1.h"
#include "GDlgH8.h"
#define ISIZE 1000 // max image size: m_horiz x ISIZE
#define TSIZE 32 // max transform size: TSIZExTSIZE
#define RSIZE (TSIZE*TSIZE) // max row vector from a block size: 1xRSIZE
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
#ifdef _DEBUG
#define new DEBUG_NEW
```

```
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
////////////////////////////////////////////////////////////////////////////
// CGDlgH8 dialog
CGDlgH8::CGDlgH8(CWnd* pParent /*=NULL*/)
: CDialog(CGDlgH8::IDD, pParent)
{
//{{AFX_DATA_INIT(CGDlgH8)
m_din = _T("");
m_fin = _T("");
m_fout = _T("");
m_horiz = 0;
m_msg = _T("");
m_vert = 0;
N = 0;
c_mode = 0;
//}}AFX_DATA_INIT
}
void CGDlgH8::DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CGDlgH8)
DDX_Control(pDX, IDC_PROGRESS, m_progress);
DDX_Text(pDX, IDC_DIN, m_din);
DDX_Text(pDX, IDC_FIN, m_fin);
DDV_MaxChars(pDX, m_fin, 20);
DDX_Text(pDX, IDC_FOUT, m_fout);
DDV_MaxChars(pDX, m_fout, 20);
DDX_Text(pDX, IDC_HORIZ, m_horiz);
DDX_Text(pDX, IDC_MSG, m_msg);
DDX_Text(pDX, IDC_VERT, m_vert);
DDV_MinMaxInt(pDX, m_vert, 0, 1000);
//}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CGDlgH8, CDialog)
//{{AFX_MSG_MAP(CGDlgH8)
ON_BN_CLICKED(IDGO, OnGo)
ON_BN_CLICKED(IDC_CLEAR, OnClear)
ON_BN_CLICKED(IDC_RADIO1, OnRadio1)
ON_BN_CLICKED(IDC_RADIO2, OnRadio2)
```

```
ON_BN_CLICKED(IDC_RADIO3, OnRadio3)
ON_BN_CLICKED(IDC_RADIO4, OnRadio4)
ON_BN_CLICKED(IDC_RADIO5, OnRadio5)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////////
// CGDlgH8 message handlers
void CGDlgH8::OnCancel()
{
// TODO: Add extra cleanup here

CDialog::OnCancel();
}
void CGDlgH8::OnClear()
{
// TODO: Add your control notification handler code here
SetDlgItemText(IDC_MSG,"");
SetDlgItemText(IDC_FIN,"");
SetDlgItemText(IDC_FOUT,"");
SetDlgItemText(IDC_DIN,"");
SetDlgItemInt(IDC_HORIZ,0);
SetDlgItemInt(IDC_VERT,0);
m_progress.SetPos(0); // PROG
}
void CGDlgH8::OnRadio1()
{
// TODO: Add your control notification handler code here
N = 8;
SetDlgItemText(IDC_MSG,"");
}
void CGDlgH8::OnRadio2()
{
// TODO: Add your control notification handler code here
N = 16;
SetDlgItemText(IDC_MSG,"");
}
void CGDlgH8::OnRadio3()
{
// TODO: Add your control notification handler code here
N = 32;
SetDlgItemText(IDC_MSG,"");
```

```
}
void CGDlgH8::OnRadio4()
{
// TODO: Add your control notification handler code here
c_mode = 1;
SetDlgItemText(IDC_MSG,"");
}
void CGDlgH8::OnRadio5()
{
// TODO: Add your control notification handler code here
c_mode = 2;
SetDlgItemText(IDC_MSG,"");
}
void CGDlgH8::OnGo()
{
/* Declaration of variables */
int l, k;
int k1, k2, cnt, cnt1;
int ll, kk, row, col;
int len_band, din_len;
int chowac, hcoef, thr, i_lastcoef, fac;
float lastcoef, vthrC, s1;
unsigned char num;
float red, gre, blu;
unsigned int sd;
float rblock[RSIZE];
char line1[102], bitt, bitts[65];
char *token;
const float pi = float(3.1416);
double norm_vthr, s1a;
int fft_len;
/* Function Prototypes */
int ch_blank(const char* );
void dct(float [][TSIZE], int);
void tnsps(float [][TSIZE], float [][TSIZE], int);
void vec_mult(float [][TSIZE], float [][TSIZE], float [][TSIZE], int);
void zigzag(float [], float [][TSIZE], int);
void sortArray(int [], int, int []);
void four1(double [], unsigned long, int);
void imbed(double [], char, int, int);
void dataproc(const char [], int, char []);
```

```
void izigzag(float [][TSIZE], float [], int);
int round(float);
int bound(int, int, int);
int chk_pow2(int);
void FreeMem();
m_progress.SetPos(0); // PROG
hcoef = 128; /* number of significant coefficients used for hiding
1 bit of data */
sd = 12345; /* seed for scattering and scrambling */
thr = 30; /* dominance threshold used for imbedding the signature */
vthrC = 0.5; /* coefficient of vthr used to indicate max visual distortion */
/* Input and validation of variables from the dialog box */
GetDlgItemText(IDC_FIN,m_fin);
if (ch_blank(m_fin)==1) {
SetDlgItemText(IDC_MSG,"Error - Enter input file name");
return;
}

GetDlgItemText(IDC_FOUT,m_fout);
if (ch_blank(m_fout)==1) {
SetDlgItemText(IDC_MSG,"Error - Enter output file name");
return;
}

GetDlgItemText(IDC_DIN,m_din);
if (*m_din==0) {
SetDlgItemText(IDC_MSG,"Error - Enter data for hiding");
return;
}

m_horiz=GetDlgItemInt(IDC_HORIZ);
if (m_horiz==0) {
SetDlgItemText(IDC_MSG,"Error - Enter width");
return;
}
m_vert=GetDlgItemInt(IDC_VERT);
if (m_vert==0) {
SetDlgItemText(IDC_MSG,"Error - Enter height");
return;
}
if (c_mode==0) {
```

```
SetDlgItemText(IDC_MSG,"Error - Choose color mode");
return;
}
if (N==0) {
SetDlgItemText(IDC_MSG,"Error - Choose block size");
return;
}
chowac = int(floor(m_vert*m_horiz/hcoef/8/8)); /* number of letters to be hidden */
chowac = chowac * 8; /* number of bits to be hidden */
m_progress.SetPos(10); // PROG

/* Dynamic memory allocation on the heap */
/* yell, cb, cr */
/* imig, trans, transT, block, blockT, res, band, scoef */
/* oseq, order, scoefo, scoef2, ang, scoefoA, SCR, scoefos */
/* scoefosA, scoefo2, coefseg, scoefos2, vthr */
float (*yell)[ISIZE] = new float[m_horiz][ISIZE];
if (yell == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
float (*cb)[ISIZE] = new float[m_horiz][ISIZE];
if (cb == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
float (*cr)[ISIZE] = new float[m_horiz][ISIZE];
if (cr == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
unsigned char (*imig)[ISIZE] = new unsigned char[m_horiz][ISIZE];
if (imig == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
float (*trans)[TSIZE] = new float[TSIZE][TSIZE];
if (trans == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
```

```
float (*transT)[TSIZE] = new float[TSIZE][TSIZE];
if (transT == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
float (*block)[TSIZE] = new float[TSIZE][TSIZE];
if (block == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
float (*blockT)[TSIZE] = new float[TSIZE][TSIZE];
if (blockT == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
float (*res)[TSIZE] = new float[TSIZE][TSIZE];
if (res == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
len_band = int(floor(m_vert/float(N))*floor(m_horiz/float(N)));
float (*band)[RSIZE] = new float[len_band][RSIZE];
if (band == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
lastcoef = (float(hcoef)*chowac/len_band); /* last significant coefficient
in a block */
if (floor(lastcoef)!=lastcoef) {
lastcoef=float(floor(lastcoef)) + 1;
}

i_lastcoef = int(lastcoef);
float (*scoef) = new float[len_band*i_lastcoef];
if (scoef == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
fft_len = len_band*i_lastcoef;
while (chk_pow2(fft_len)!=0) {
fft_len++;
```

```
}
int (*oseq) = new int[len_band*i_lastcoef];
if (oseq == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}

int (*order) = new int[len_band*i_lastcoef];
if (order == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
float (*scoefo) = new float[len_band*i_lastcoef];
if (scoefo == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
float (*scoef2) = new float[len_band*i_lastcoef];
if (scoef2 == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
double (*ang) = new double[fft_len];
if (ang == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
double (*scoefoA) = new double[2*fft_len];
if (scoefoA == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
double (*SCR) = new double[2*fft_len];
if (SCR == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
double (*scoefos) = new double[2*fft_len];
if (scoefos == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
```

```
}
double (*scoefosA) = new double[len_band*i_lastcoef];
if (scoefosA == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
double (*scoefo2) = new double[2*fft_len];
if (scoefo2 == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
double (*coefseg) = new double[hcoef];
if (coefseg == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
double (*scoefos2) = new double[2*fft_len];
if (scoefos2 == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
double (*vthr) = new double[len_band*i_lastcoef];
if (vthr == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
/* End of dynamic memory allocation on the heap */
m_progress.SetPos(20); // PROG
/* Reading original image file */
FILE *f1Ptr; /* f1Ptr = input file pointer */

if ((f1Ptr = fopen(m_fin, "rb")) == NULL) {
SetDlgItemText(IDC_MSG,"Error - Input file could not be found");
delete [] imig;
delete [] yell;
delete [] cb;
delete [] cr;
delete [] trans;
delete [] transT;
delete [] block;
delete [] blockT;
```

```
delete [] res;
delete [] band;
delete [] scoef;
delete [] oseq;
delete [] order;
delete [] scoefo;
delete [] scoef2;
delete [] ang;
delete [] scoefoA;
delete [] SCR;
delete [] scoefos;
delete [] scoefosA;
delete [] scoefo2;
delete [] coefseg;
delete [] scoefos2;
delete [] vthr;
return;
}
else {
if (c_mode==2) {
/* get line with P6 */
fgets(line1, 101, f1Ptr);
if (line1[1]!='6') {
SetDlgItemText(IDC_MSG,"Error - Input file of wrong format");
fclose(f1Ptr);
delete [] imig;
delete [] yell;
delete [] cb;
delete [] cr;
delete [] trans;
delete [] transT;
delete [] block;
delete [] blockT;
delete [] res;
delete [] band;
delete [] scoef;
delete [] oseq;
delete [] order;
delete [] scoefo;
delete [] scoef2;
delete [] ang;
```

```
delete [] scoefoA;
delete [] SCR;
delete [] scoefos;
delete [] scoefosA;
delete [] scoefo2;
delete [] coefseg;
delete [] scoefos2;
delete [] vthr;
return;
}
/* get next line */
fgets(line1, 101, f1Ptr);

/* get remaining lines starting with # and one line after them */
while (line1[0] == '#') {
fgets(line1, 101, f1Ptr);
}
/* getting rid of width, height and no. of colors */
token = strtok (line1," ");
token = strtok (NULL," ");
token = strtok (NULL," ");
/* supporting 2 styles of P6 files */
if (token == 0) {
fgets(line1, 101, f1Ptr);
}
/* getting data */
for (l = 0; l <=m_vert-1; l++) {
for (k = 0; k <=m_horiz-1; k++) {
num = fgetc(f1Ptr);
red = float(num);
num = fgetc(f1Ptr);
gre = float(num);
num = fgetc(f1Ptr);
blu = float(num);
/* Convert to other color domain */
yel[k][l] = float(0.3*red+0.6*gre+0.1*blu);
cr[k][l] = float(0.45*blu-0.15*red-0.3*gre+0.5);
cb[k][l] = float(0.4375*red-0.375*gre-0.0625*blu+0.5);
}
}
}
```

```
else {
/* get line with P5 */
fgets(line1, 101, f1Ptr);
if (line1[1]!='5') {
SetDlgItemText(IDC_MSG,"Error - Input file of wrong format");
fclose(f1Ptr);
delete [] imig;
delete [] yell;
delete [] cb;
delete [] cr;
delete [] trans;
delete [] transT;
delete [] block;
delete [] blockT;
delete [] res;
delete [] band;
delete [] scoef;
delete [] oseq;
delete [] order;
delete [] scoefo;
delete [] scoef2;
delete [] ang;
delete [] scoefoA;
delete [] SCR;
delete [] scoefos;
delete [] scoefosA;
delete [] scoefo2;
delete [] coefseg;
delete [] scoefos2;
delete [] vthr;
return;
}
/* get next line */
fgets(line1, 101, f1Ptr);

/* get remaining lines starting with # and one line after them */
while (line1[0] == '#') {
fgets(line1, 101, f1Ptr);
}
/* getting rid of width, height and no. of colors */
token = strtok (line1," ");
```

```
token = strtok (NULL," ");
token = strtok (NULL," ");
/* supporting 2 styles of P5 files */
if (token == 0) {
fgets(line1, 101, f1Ptr);
}
/* getting data */
for (l = 0; l <=m_vert-1; l++) {
for (k = 0; k <=m_horiz-1; k++) {
num = fgetc(f1Ptr);
imig[k][l] = num;
}
}
}
fclose(f1Ptr);
dct(trans,N); /* creation of DCT matrix */
tnsps(transT, trans, N); /* transpose of DCT matrix */
/* Creating bands */
cnt = 0;
for (ll = 0; ll<=int(floor(m_vert/float(N))-1); ll++) {
for (kk = 0; kk<=int(floor(m_horiz/float(N))-1); kk++) {
col = 0;
for (l = (ll*N); l<=(N*(ll+1))-1; l++) {
row = 0;
for (k = (kk*N); k<=(N*(kk+1))-1; k++) {
if (c_mode==2) {
block[row][col] = yell[k][l];
}
else {
block[row][col] = float(imig[k][l]);
}
row++;
}
col++;
}
vec_mult(res, trans, block, N); /* trans*block */
vec_mult(block, res, transT, N); /* vector multiplication */
tnsps(blockT, block, N); /* transpose of block */
zigzag(rblock, blockT, N); /* zigzag single block */
for (k1 = 0; k1<=N*N-1; k1++) {
band[cnt][k1]=rblock[k1];
```

```
}
cnt++;
}
}

/* Preparation of signature for processing */
din_len = m_din.GetLength();
dataproc(m_din, din_len, bitts);
/* Extracting significant coefficients from the subbands */
cnt1 = 0;
for (k = 0; k<=len_band-1; k++) {
for (l = 1; l<=i_lastcoef; l++) {
scoef[cnt1] = band[k][l];
cnt1++;
}
}
/* Random scattering of significant coefficients ************************/
/* Random sequence generation */
srand(sd);
for (k = 0; k<=len_band*i_lastcoef-1; k++) {
oseq[k] = rand();
order[k] = k; /* initializing order array */
}
m_progress.SetPos(30); // PROG
/* Sorting of random sequence in ascending order */
sortArray(oseq, len_band*i_lastcoef, order);

/* Scattering of significant coefficients */
for (k = 0; k<=len_band*i_lastcoef-1; k++) {
scoefo[k] = scoef[order[k]];
}
/* Preparing them for FFT */
l = 0;
for (k = 0; k<=2*len_band*i_lastcoef-1; k=k+2) {
scoefoA[k] = scoefo[l];
scoefoA[k+1] = 0;
l++;
}
/* making scoefoA an array of length that is a power of 2 */
k = 2*len_band*i_lastcoef;
while (k<2*fft_len) {
```

```
scoefoA[k] = 0;
scoefoA[k+1] = 0;
k+=2;
}
/* FFT(scoefoA) */
four1(scoefoA-1,fft_len,1);
m_progress.SetPos(40); // PROG
/* All-pass scrambling ************************************************/
/* Generation of random angles */
srand(sd);
k1 = 2;
for (k = 0; k<=fft_len-1; k++) {
if ((k==0) —— (k==fft_len/2)) {
ang[k] = 0;
}
else {
if (k<fft_len/2) {
ang[k] = (rand() % 1001)/1000*2*pi;
}
else {
ang[k] = ang[k-k1];
k1=k1+2;
}
}
}
/* SCR=cos(ang)+i*sin(ang) */
l = 0;
for (k = 0; k<=2*fft_len-1; k=k+2) {
SCR[k] = cos(ang[l]); // real part
SCR[k+1] = sin(ang[l]); // imaginary part
l++;
}
/* fft(scoefo).*conj(SCR) */
for (k = 0; k<=2*fft_len-1; k=k+2) {
scoefos[k]=scoefoA[k]*SCR[k]-scoefoA[k+1]*(-1)*SCR[k+1];
scoefos[k+1]=scoefoA[k+1]*SCR[k]+scoefoA[k]*(-1)*SCR[k+1];
}
/* ifft(fft(scoefo).*conj(SCR)) */
four1(scoefos-1,fft_len,-1);
for (k = 0; k<=2*fft_len-1; k++) {
scoefos[k]=scoefos[k]/fft_len;
```

```
}
/* scoefosA = real(scoefos) */
l = 0;
for (k = 0; k<=len_band*i_lastcoef-1; k++) {
scoefosA[k] = scoefos[l]; // scoefos real parts only
l=l+2;
}
/*****************************************************/

m_progress.SetPos(50); // PROG
/* Adding a signature ************************************************/
for (k = 0; k<=chowac-1; k++) {
bitt = bitts[k];
ll = 0;
for (kk = k*hcoef; kk<=(k+1)*hcoef-1; kk++) {
coefseg[ll] = scoefosA[kk];
ll++;
}
imbed(coefseg, bitt, thr, hcoef);
ll = 0;
for (kk = k*hcoef; kk<=(k+1)*hcoef-1; kk++) {
scoefosA[kk] = coefseg[ll];
ll++;
}
}
/* All-pass unscrambling *********************************************/
/* Preparing scoefos2 for FFT */
l = 0;
for (k = 0; k<=2*len_band*i_lastcoef-1; k=k+2) {
scoefos2[k] = scoefosA[l];
scoefos2[k+1] = 0;
l++;
}
/* making scoefos2 an array of length that is a power of 2 */
k = 2*len_band*i_lastcoef;
while (k<2*fft_len) {
scoefos2[k] = 0;
scoefos2[k+1] = 0;
k+=2;
}
```

```
/* FFT(scoefos2) */
four1(scoefos2-1,fft_len,1);
m_progress.SetPos(60); // PROG
/* fft(scoefos2).*conj(SCR) */
for (k = 0; k<=2*fft_len-1; k=k+2) {
scoefo2[k]=scoefos2[k]*SCR[k]-scoefos2[k+1]*SCR[k+1];
scoefo2[k+1]=scoefos2[k+1]*SCR[k]+scoefos2[k]*SCR[k+1];
}
/* ifft(fft(scoefos2).*conj(SCR)) */
four1(scoefo2-1,fft_len,-1);
for (k = 0; k<=2*fft_len-1; k++) {
scoefo2[k]=scoefo2[k]/fft_len;
}
/* Uncattering of significant coefficients */
l = 0;
for (k = 0; k<=len_band*i_lastcoef-1; k++) {
scoef2[order[k]] = float(scoefo2[l]); // scoefo2 real parts only
l=l+2;
}
m_progress.SetPos(70); // PROG
/* Inserting coefficients back into the subband array *****************/

cnt1 = 0;
for (k = 0; k<=len_band-1; k++) {
for (l = 1; l<=i_lastcoef; l++) {
band[k][l] = scoef2[cnt1];
cnt1++;
}
}

/*********************************************************************/
/* Retrieve the picture with signature into viewable form ************/
ll = 0;
kk = 0;
for (k2=0;k2<=len_band-1;k2++) {
for (k1 = 0; k1<=N*N-1; k1++) {
rblock[k1] = band[k2][k1];
}
izigzag(blockT, rblock, N);
tnsps(block, blockT, N); /* transpose of block */
vec_mult(res, transT, block, N); /* transT*block */
```

```
vec_mult(block, res, trans, N); /* vector multiplication */

l = ll*N;
for (col=0; col<=N-1; col++) {
k = kk*N;
for (row=0; row<=N-1; row++) {
if (c_mode==2) {
yell[k][l] = block[row][col];
}
else {
imig[k][l] = bound(round(block[row][col]), 0, 255);
}
k++;
}
l++;
}
kk++;
if (kk==int(floor(m_horiz/float(N)))) {
kk = 0;
ll++;
}
}
m_progress.SetPos(80); // PROG

/* Writing image to the output file */
FILE *flPtr; /* flPtr = input file pointer */

if ((flPtr = fopen(m_fout, "wb")) == NULL) {
SetDlgItemText(IDC_MSG,"Error - Output file could not be opened");
delete [] imig;
delete [] yell;
delete [] cb;
delete [] cr;
delete [] trans;
delete [] transT;
delete [] block;
delete [] blockT;
delete [] res;
delete [] band;
delete [] scoef;
delete [] oseq;
```

```
delete [] order;
delete [] scoefo;
delete [] scoef2;
delete [] ang;
delete [] scoefoA;
delete [] SCR;
delete [] scoefos;
delete [] scoefosA;
delete [] scoefo2;
delete [] coefseg;
delete [] scoefos2;
delete [] vthr;
return;
}
else {
if (c_mode==2) {
fprintf(f1Ptr, "%s", "P6\n");
fprintf(f1Ptr, "%s", "# Created using: DHS Version: 1.1\n");
fprintf(f1Ptr, "%s", "# Copyright: Arkadiusz Komenda\n");
fprintf(f1Ptr, "%d%s%d%s", m_horiz, " ", m_vert, " 255\n");
for (l = 0; l <=m_vert-1; l++) {
for (k = 0; k <=m_horiz-1; k++) {
/* red */
num=bound(round(float(yell[k][l]+1.6*cb[k][l]-0.8)), 0, 255);
fputc(num,f1Ptr);
/* green */
num=bound(round(float(yell[k][l]-(float(1)/3)*cr[k][l]+(float(1)/6)-0.8*cb[k][l]+0.4)), 0, 255);
fputc(num,f1Ptr);
/* blue */
num=bound(round(float(yell[k][l]+2*cr[k][l]-1.0)), 0, 255);
fputc(num,f1Ptr);
}
}
}
else {
fprintf(f1Ptr, "%s", "P5\n");
fprintf(f1Ptr, "%s", "# Created using: DHS Version: 1.1\n");
fprintf(f1Ptr, "%s", "# Copyright: Arkadiusz Komenda\n");
fprintf(f1Ptr, "%d%s%d%s", m_horiz, " ", m_vert, " 255\n");
for (l = 0; l <=m_vert-1; l++) {
for (k = 0; k <=m_horiz-1; k++) {
```

```
num = imig[k][l];
fputc(num,f1Ptr);
}
}
}
fclose(f1Ptr);
}

}

m_progress.SetPos(90); // PROG
/* Dynamic memory deallocation on the heap */
/* imig, trans, transT, block, blockT, res, band */
delete [] imig;
delete [] yell;
delete [] cb;
delete [] cr;
delete [] trans;
delete [] transT;
delete [] block;
delete [] blockT;
delete [] res;
delete [] band;
delete [] scoef;
delete [] oseq;
delete [] order;
delete [] scoefo;
delete [] scoef2;
delete [] ang;
delete [] scoefoA;
delete [] SCR;
delete [] scoefos;
delete [] scoefosA;
delete [] scoefo2;
delete [] coefseg;
delete [] scoefos2;
delete [] vthr;

m_progress.SetPos(100); // PROG
SetDlgItemText(IDC_MSG,"");
}
```

```
/* Function Definitions */
int ch_blank(const char* data)
/* Check if a string consists of spaces only */
{
if (strlen(data)==strspn(data," "))
return 1;
else
return 0;
}
void dct(float arr[][TSIZE], int ntemp)
/* Generate ntempXntemp DCT matrix */
{
const float pi = float(3.1416);
int l, k;
float alpha, n;
n = float(ntemp);
for (l = 0; l<=ntemp-1; l++) {
for (k = 0; k<=ntemp-1; k++) {
if (k==0) {
alpha = float(1/sqrt(2));
}
else {
alpha = 1;
}
arr[k][l] = float(sqrt(2/n)*alpha*cos(((2*l+1)*k*pi)/(2*n)));
}
}
return;
}
void tnsps(float arrT[][TSIZE], float arr[][TSIZE], int size)
/* Transpose of a square matrix */
{

int l, k;
for (l = 0; l<=size-1; l++) {
for (k = 0; k<=size-1; k++) {
arrT[k][l] = arr[l][k];
}
}
return;
}
```

```
void vec_mult(float prod[][TSIZE], float arr1[][TSIZE], float arr2[][TSIZE], int size)
/* Vector product of two same size square matrices */
{
int l1, k1, k2;
float sum;
for (l1 = 0; l1<=size-1; l1++) {
for (k2 = 0; k2<=size-1; k2++) {
sum = 0;
for (k1 = 0; k1<=size-1; k1++) {
sum = sum + (arr1[l1][k1] * arr2[k1][k2]);
}
prod[l1][k2] = sum;
}
}
return;
}
void zigzag(float vect[], float arr[][TSIZE], int size)
/* zigzag read the block of data */
{
int k, l, cnt;
cnt=0;
for (k = 0; k<=size-1; k++) {
for (l = 0; l<=k; l++) {
vect[cnt] = arr[k-l][l];
cnt++;
}
}
for (k = 1; k<=size-1; k++) {
for (l = size-1; l>=k; l--) {
vect[cnt] = arr[l][k+(size-1)-l];
cnt++;
}
}
return;
}
void DecToBin(int dec, char binar[])
/* converts a decimal number to 8-bit binary */
{
int k,l;
k = 8;
while (dec!=0) {
```

```
k--;
if ((dec % 2)==0) {
binar[k]='0';
}
else {
binar[k]='1';
}
dec = dec/2;
}
if (k>0) {
for (l=k-1; l>=0; l--) {
binar[l] = '0';
}
}
return;
}
void sortArray(int orig[], int size, int ind[])
/* sort array in ascending order and memorize old positions of elements
prior to sorting */
{
int k, l;
int temp, itemp;
for (k = size-2; k>=0; k--) {
for (l = 0; l<=k; l++) {
if (orig[l]>orig[l+1]) {
temp = orig[l];
orig[l] = orig[l+1];
orig[l+1] = temp;
itemp = ind[l];
ind[l] = ind[l+1];
ind[l+1] = itemp;
}
}
}

return;
}
void four1(double data[], unsigned long nn, int isign)
/* FFT and IFFT routine */
{
unsigned long n,mmax,m,j,istep,i;
```

```
double wtemp,wr,wpr,wpi,wi,theta;
double tempr,tempi;
n=nn << 1;
j=1;
for (i=1;i<n;i+=2) {
if (j > i) {
SWAP(data[j],data[i]);
SWAP(data[j+1],data[i+1]);
}
m=n >> 1;
while (m >= 2 && j > m) {
j -= m;
m >>= 1;
}
j += m;
}
mmax=2;
while (n > mmax) {
istep=mmax << 1;
theta=isign*(6.28318530717959/mmax);
wtemp=sin(0.5*theta);
wpr = -2.0*wtemp*wtemp;
wpi=sin(theta);
wr=1.0;
wi=0.0;
for (m=1;m<mmax;m+=2) {
for (i=m;i<=n;i+=istep) {
j=i+mmax;
tempr=wr*data[j]-wi*data[j+1];
tempi=wr*data[j+1]+wi*data[j];
data[j]=data[i]-tempr;
data[j+1]=data[i+1]-tempi;
data[i] += tempr;
data[i+1] += tempi;
}
wr=(wtemp=wr)*wpr-wi*wpi+wr;
wi=wi*wpr+wtemp*wpi+wi;
}
mmax=istep;
}
}
```

```
#undef SWAP
void imbed(double seg[], char bitt, int dom, int len_seg)
/* Embedding data */
{
int k, chk, dthr;
const int thr = 48; /* standard threshold */
double (*segs) = new double[len_seg];

for(k=0; k<=len_seg-1; k++) {
segs[k] = seg[k];
}

if (bitt=='1') {
for (k=0; k<=len_seg-1; k++) {
if ((seg[k]<0)&&(seg[k]>((-1)*thr))) {
segs[k] = thr*9/4;
}
else {
if ((seg[k]>0)&&(seg[k]<thr/2)) {
segs[k] = seg[k] + thr/9;
}
}
}
}
else {
for (k=0; k<=len_seg-1; k++) {
if ((seg[k]>0)&&(seg[k]<thr)) {
segs[k] = (-1)*thr*9/4;
}
else {
if ((seg[k]<0)&&(seg[k]>(-1)*thr/2)) {
segs[k] = seg[k] - thr/9;
}
}
}
}
/* Checking signs */
chk = 0;
for(k=0; k<=len_seg-1; k++) {
if (segs[k]<0) {
chk = chk - 1;
```

```
}
else {
if (segs[k]>0) {
chk = chk + 1;
}
}
}
dthr = thr;
while (((bitt=='1')&&(chk<=dom+8)) —— ((bitt=='0')&&(chk>=-dom+8))) {
for(k=0; k<=len_seg-1; k++) {
segs[k] = seg[k];
}
dthr += 1;
if (bitt=='1') {
for (k=0; k<=len_seg-1; k++) {
if ((seg[k]<0)&&(seg[k]>((-1)*dthr))) {
segs[k] = dthr*9/4;
}
else {
if ((seg[k]>0)&&(seg[k]<dthr/2)) {
segs[k] = seg[k] + dthr/9;
}
}
}
}
else {
for (k=0; k<=len_seg-1; k++) {
if ((seg[k]>0)&&(seg[k]<dthr)) {
segs[k] = (-1)*dthr*9/4;
}
else {
if ((seg[k]<0)&&(seg[k]>(-1)*dthr/2)) {
segs[k] = seg[k] - dthr/9;
}
}
}
}
/* Checking signs */
chk = 0;
for(k=0; k<=len_seg-1; k++) {
if (segs[k]<0) {
```

```
chk = chk - 1;
}
else {
if (segs[k]>0) {
chk = chk + 1;
}
}
}
}
for(k=0; k<=len_seg-1; k++) {
seg[k] = segs[k];
}
delete [] segs;
return;
}
void dataproc(const char dat[], int dat_len, char bitts[])
/* data pre-processing */
{
int k, dec_num;
char binary[8];
void DecToBin(int, char []);
bitts[0]='\0';
for (k = 0; k <= dat_len-1; k++) {
dec_num = __toascii(dat[k]);
DecToBin(dec_num,binary);
if (k==0) {
strncat(bitts, binary, 8);
}
else {
strncat(bitts,binary, 8);
}
}
return;
}
void izigzag(float arr[][TSIZE], float vect[], int size)
/* zigzag write the block of data */
{
int k, l, cnt;
cnt=0;
for (k = 0; k<=size-1; k++) {
for (l = 0; l<=k; l++) {
```

```
arr[k-l][l] = vect[cnt];
cnt++;
}
}
for (k = 1; k<=size-1; k++) {
for (l = size-1; l>=k; l--) {
arr[l][k+(size-1)-l] = vect[cnt];
cnt++;
}
}
return;
}
int round(float num)
/* rounding float numbers to integers */
{
int whole, out;
float test;
whole = int(floor(num));
test = float(whole) + float(0.5);
if (num >= test) {
out = whole + 1;
}
else {
out = whole;
}
return out;
}
int bound(int a, int lowerbound, int upperbound)
/* bound the integer */
{
int b;

b=__min(a,upperbound);
b=__max(a,lowerbound);
return b;
}
int chk_pow2(int num)
/* check whether an integer is a power of 2 or not */
/* Return values: */
/* flag = 0 : integer is a power of 2 */
/* flag = 1 : integer is NOT a power of 2 */
```

```
{
int flag = 0;
while ((num>1)&&(flag==0)) {
if ((num % 2)==0) {
flag = 0;
}
else {
flag = 1;
}
num = num/2;
}
return flag;
}
/* End of Function Definitions */
```

GDlgD8.cpp - data retrieval from images subroutine: // GDlgD8.cpp : implementation file

```
//
#include "stdafx.h"
#include "DHS1.h"
#include "GDlgD8.h"
#define ISIZE 1000 // max image size: m_horiz x ISIZE
#define TSIZE 32 // max transform size: TSIZExTSIZE
#define RSIZE (TSIZE*TSIZE) // max row vector from a block size: 1xRSIZE
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// CGDlgD8 dialog
CGDlgD8::CGDlgD8(CWnd* pParent /*=NULL*/)
: CDialog(CGDlgD8::IDD, pParent)
{
//{{AFX_DATA_INIT(CGDlgD8)
m_dout = _T("");
m_fin = _T("");
m_msg = _T("");
m_horiz = 0;
m_vert = 0;
N = 0;
```

```
c_mode = 0;
//}}AFX_DATA_INIT
}
void CGDlgD8::DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CGDlgD8)
DDX_Control(pDX, IDC_PROGRESS, m_progress);
DDX_Text(pDX, IDC_DOUT, m_dout);
DDX_Text(pDX, IDC_FIN, m_fin);
DDV_MaxChars(pDX, m_fin, 20);
DDX_Text(pDX, IDC_MSG, m_msg);
DDX_Text(pDX, IDC_HORIZ, m_horiz);
DDX_Text(pDX, IDC_VERT, m_vert);
DDV_MinMaxInt(pDX, m_vert, 0, 1000);
//}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CGDlgD8, CDialog)
//{{AFX_MSG_MAP(CGDlgD8)
ON_BN_CLICKED(IDGO, OnGo)
ON_BN_CLICKED(IDC_CLEAR, OnClear)
ON_BN_CLICKED(IDC_RADIO1, OnRadio1)
ON_BN_CLICKED(IDC_RADIO2, OnRadio2)
ON_BN_CLICKED(IDC_RADIO3, OnRadio3)
ON_BN_CLICKED(IDC_RADIO4, OnRadio4)
ON_BN_CLICKED(IDC_RADIO5, OnRadio5)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////////
// CGDlgD8 message handlers
void CGDlgD8::OnCancel()
{
// TODO: Add extra cleanup here

CDialog::OnCancel();
}
void CGDlgD8::OnClear()
{
// TODO: Add your control notification handler code here
SetDlgItemText(IDC_MSG,"");
SetDlgItemText(IDC_FIN,"");
```

```
SetDlgItemText(IDC_DOUT,"");
SetDlgItemInt(IDC_HORIZ,0);
SetDlgItemInt(IDC_VERT,0);
m_progress.SetPos(0); // PROG
}
void CGDlgD8::OnRadio1()
{
// TODO: Add your control notification handler code here
N = 8;
SetDlgItemText(IDC_MSG,"");
}
void CGDlgD8::OnRadio2()
{
// TODO: Add your control notification handler code here
N = 16;
SetDlgItemText(IDC_MSG,"");
}
void CGDlgD8::OnRadio3()
{
// TODO: Add your control notification handler code here
N = 32;
SetDlgItemText(IDC_MSG,"");
}
void CGDlgD8::OnRadio4()
{
// TODO: Add your control notification handler code here
c_mode = 1;
SetDlgItemText(IDC_MSG,"");
}
void CGDlgD8::OnRadio5()
{
// TODO: Add your control notification handler code here
c_mode = 2;
SetDlgItemText(IDC_MSG,"");
}
void CGDlgD8::OnGo()
{
// TODO: Add your control notification handler code here
int l, k, k1, cnt, cnt1;
int ll, kk, row, col;
int len_band;
```

```
int chowac, hcoef, i_lastcoef;
float lastcoef, red, gre, blu;
unsigned char num, datout[8], numR1, numB1, numG1;
unsigned int sd;
float rblock[RSIZE];
char line1[102];
char bitt[2], bitts[65], letter[8];
int i_bitt;
char *token;
const float pi = float(3.1416);
int fft_len;
/* Function Prototypes */
int ch_blank(const char* );
void dct(float [][TSIZE], int);
void tnsps(float [][TSIZE], float [][TSIZE], int);
void vec_mult(float [][TSIZE], float [][TSIZE], float [][TSIZE], int);
void zigzag(float [], float [][TSIZE], int);
void sortArray(int [], int, int []);
void four1(double [], unsigned long, int);
int chk_pow2(int);
int DetBit(double [], int);
int BinToDec(char []);
m_progress.SetPos(0); // PROG
hcoef = 128; /* number of significant coefficients used for hiding
1 bit of data */
sd = 12345; /* seed for scattering and scrambling */
GetDlgItemText(IDC_FIN,m_fin);
if (ch_blank(m_fin)==1) {
SetDlgItemText(IDC_MSG,"Error - Enter input file name");
return;
}

m_horiz=GetDlgItemInt(IDC_HORIZ);
if (m_horiz==0) {
SetDlgItemText(IDC_MSG,"Error - Enter width");
return;
}
m_vert=GetDlgItemInt(IDC_VERT);
if (m_vert==0) {
SetDlgItemText(IDC_MSG,"Error - Enter height");
return;
```

```
}
if (c_mode==0) {
SetDlgItemText(IDC_MSG,"Error - Choose color mode");
return;
}
if (N==0) {
SetDlgItemText(IDC_MSG,"Error - Choose block size");
return;
}
chowac = int(floor(m_vert*m_horiz/hcoef/8/8)); /* number of letters to be hidden */
chowac = chowac * 8; /* number of bits to be hidden */
/* Dynamic memory allocation on the heap */
/* yell, imig, trans, transT, block, blockT, res, band, scoef */
/* oseq, order, scoefo, ang, scoefoA, SCR, scoefos */
/* scoefosA, coefseg */
float (*yell)[ISIZE] = new float[m_horiz][ISIZE];
if (yell == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
unsigned char (*imig)[ISIZE] = new unsigned char[m_horiz][ISIZE];
if (imig == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
float (*trans)[TSIZE] = new float[TSIZE][TSIZE];
if (trans == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
float (*transT)[TSIZE] = new float[TSIZE][TSIZE];
if (transT == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
float (*block)[TSIZE] = new float[TSIZE][TSIZE];
if (block == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
float (*blockT)[TSIZE] = new float[TSIZE][TSIZE];
```

```
if (blockT == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
float (*res)[TSIZE] = new float[TSIZE][TSIZE];
if (res == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
len_band = int(floor(m_vert/float(N))*floor(m_horiz/float(N)));
float (*band)[RSIZE] = new float[len_band][RSIZE];
if (band == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
lastcoef = (float(hcoef)*chowac/len_band); /* last significant coefficient
in a block */
if (floor(lastcoef)!=lastcoef) {
lastcoef=float(floor(lastcoef)) + 1;
}

i_lastcoef = int(lastcoef);
float (*scoef) = new float[len_band*i_lastcoef];
if (scoef == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
fft_len = len_band*i_lastcoef;
while (chk_pow2(fft_len)!=0) {
fft_len++;
}
int (*oseq) = new int[len_band*i_lastcoef];
if (oseq == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}

int (*order) = new int[len_band*i_lastcoef];
if (order == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
```

```
}
float (*scoefo) = new float[len_band*i_lastcoef];
if (scoefo == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
double (*ang) = new double[fft_len];
if (ang == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
double (*scoefoA) = new double[2*fft_len];
if (scoefoA == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
double (*SCR) = new double[2*fft_len];
if (SCR == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
double (*scoefos) = new double[2*fft_len];
if (scoefos == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
double (*scoefosA) = new double[len_band*i_lastcoef];
if (scoefosA == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
double (*coefseg) = new double[hcoef];
if (coefseg == NULL) {
SetDlgItemText(IDC_MSG,"Error - Not enough memory");
return;
}
/* End of dynamic memory allocation on the heap */
/* Reading original image file */
FILE *flPtr; /* flPtr = input file pointer */

if ((flPtr = fopen(m_fin, "rb")) == NULL) {
```

```
SetDlgItemText(IDC_MSG,"Error - Input file could not be found");
delete [] yell;
delete [] imig;
delete [] trans;
delete [] transT;
delete [] block;
delete [] blockT;
delete [] res;
delete [] band;
delete [] scoef;
delete [] oseq;
delete [] order;
delete [] scoefo;
delete [] ang;
delete [] scoefoA;
delete [] SCR;
delete [] scoefos;
delete [] scoefosA;
delete [] coefseg;
return;
}
else {
if (c_mode==2) {
/* get line with P6 */
fgets(line1, 101, f1Ptr);
if (line1[1]!='6') {
SetDlgItemText(IDC_MSG,"Error - Input file of wrong format");
fclose(f1Ptr);
delete [] yell;
delete [] imig;
delete [] trans;
delete [] transT;
delete [] block;
delete [] blockT;
delete [] res;
delete [] band;
delete [] scoef;
delete [] oseq;
delete [] order;
delete [] scoefo;
delete [] ang;
```

```
delete [] scoefoA;
delete [] SCR;
delete [] scoefos;
delete [] scoefosA;
delete [] coefseg;
return;
}
/* get next line */
fgets(line1, 101, f1Ptr);

/* get remaining lines starting with # and one line after them */
while (line1[0] == '#') {
fgets(line1, 101, f1Ptr);
}
/* getting rid of width, height and no. of colors */
token = strtok (line1," ");
token = strtok (NULL," ");
token = strtok (NULL," ");
/* supporting 2 styles of P6 files */
if (token == 0) {
fgets(line1, 101, f1Ptr);
}
/* getting data */
for (l = 0; l <=m_vert-1; l++) {
for (k = 0; k <=m_horiz-1; k++) {
num = fgetc(f1Ptr);
if ((k==23)&&(l==54)) {
numR1=num;
}
red = float(num);
num = fgetc(f1Ptr);
if ((k==23)&&(l==54)) {
numG1=num;
}
gre = float(num);
num = fgetc(f1Ptr);
if ((k==23)&&(l==54)) {
numB1=num;
}
blu = float(num);
/* Convert to other color domain */
```

```
yell[k][l] = float(0.3*red+0.6*gre+0.1*blu);
}
}
}
else {
/* get line with P5 */
fgets(line1, 101, f1Ptr);
if (line1[1]!='5') {
SetDlgItemText(IDC_MSG,"Error - Input file of wrong format");
fclose(f1Ptr);
delete [] yell;
delete [] imig;
delete [] trans;
delete [] transT;
delete [] block;
delete [] blockT;
delete [] res;
delete [] band;
delete [] scoef;
delete [] oseq;
delete [] order;
delete [] scoefo;
delete [] ang;
delete [] scoefoA;
delete [] SCR;
delete [] scoefos;
delete [] scoefosA;
delete [] coefseg;
return;
}
/* get next line */
fgets(line1, 101, f1Ptr);

/* get remaining lines starting with # and one line after them */
while (line1[0] == '#') {
fgets(line1, 101, f1Ptr);
}
/* getting rid of width, height and no. of colors */
token = strtok (line1," ");
token = strtok (NULL," ");
token = strtok (NULL," ");
```

```
/* supporting 2 styles of P5 files */
if (token == 0) {
fgets(line1, 101, f1Ptr);
}
/* getting data */
for (l = 0; l <=m_vert-1; l++) {
for (k = 0; k <=m_horiz-1; k++) {
num = fgetc(f1Ptr);
imig[k][l] = num;
}
}
fclose(f1Ptr);
}
}
dct(trans,N); /* creation of DCT matrix */
tnsps(transT, trans, N); /* transpose of DCT matrix */
/* Creating bands */
cnt = 0;
for (ll = 0; ll<=int(floor(m_vert/float(N))-1); ll++) {
for (kk = 0; kk<=int(floor(m_horiz/float(N))-1); kk++) {
col = 0;
for (l = (ll*N); l<=(N*(ll+1))-1; l++) {
row = 0;
for (k = (kk*N); k<=(N*(kk+1))-1; k++) {
if (c_mode==2) {
block[row][col] = yell[k][l];
}
else {
block[row][col] = float(imig[k][l]);
}
row++;
}
col++;
}
vec_mult(res, trans, block, N); /* trans*block */
vec_mult(block, res, transT, N); /* vector multiplication */
tnsps(blockT, block, N); /* transpose of block */
zigzag(rblock, blockT, N); /* zigzag single block */
for (k1 = 0; k1<=N*N-1; k1++) {
band[cnt][k1]=rblock[k1];
}
```

```
cnt++;
}
}

/* Extracting significant coefficients from the subbands */
cnt1 = 0;
for (k = 0; k<=len_band-1; k++) {
for (l = 1; l<=i_lastcoef; l++) {
scoef[cnt1] = band[k][l];
cnt1++;
}
}
/* Random scattering of significant coefficients ************************/
/* Random sequence generation */
srand(sd);
for (k = 0; k<=len_band*i_lastcoef-1; k++) {
oseq[k] = rand();
order[k] = k; /* initializing order array */
}
m_progress.SetPos(30); // PROG
/* Sorting of random sequence in ascending order */
sortArray(oseq, len_band*i_lastcoef, order);

/* Scattering of significant coefficients */
for (k = 0; k<=len_band*i_lastcoef-1; k++) {
scoefo[k] = scoef[order[k]];
}
/* Preparing them for FFT */
l = 0;
for (k = 0; k<=2*len_band*i_lastcoef-1; k=k+2) {
scoefoA[k] = scoefo[l];
scoefoA[k+1] = 0;
l++;
}
/* making scoefoA an array of length that is a power of 2 */
k = 2*len_band*i_lastcoef;
while (k<2*fft_len) {
scoefoA[k] = 0;
scoefoA[k+1] = 0;
k+=2;
}
```

```
/* FFT(scoefoA) */
four1(scoefoA-1,fft_len,1);
m_progress.SetPos(40); // PROG
/* All-pass scrambling ***********************************************/
/* Generation of random angles */
srand(sd);
k1 = 2;
for (k = 0; k<=fft_len-1; k++) {
if ((k==0) —— (k==fft_len/2)) {
ang[k] = 0;
}
else {
if (k<fft_len/2) {
ang[k] = (rand() % 1001)/1000*2*pi;
}
else {
ang[k] = ang[k-k1];
k1=k1+2;
}
}
}
/* SCR=cos(ang)+i*sin(ang) */
l = 0;
for (k = 0; k<=2*fft_len-1; k=k+2) {
SCR[k] = cos(ang[l]); // real part
SCR[k+1] = sin(ang[l]); // imaginary part
l++;
}
/* fft(scoefo).*conj(SCR) */
for (k = 0; k<=2*fft_len-1; k=k+2) {
scoefos[k]=scoefoA[k]*SCR[k]-scoefoA[k+1]*(-1)*SCR[k+1];
scoefos[k+1]=scoefoA[k+1]*SCR[k]+scoefoA[k]*(-1)*SCR[k+1];
}
m_progress.SetPos(50); // PROG
/* ifft(fft(scoefo).*conj(SCR)) */
four1(scoefos-1,fft_len,-1);
for (k = 0; k<=2*fft_len-1; k++) {
scoefos[k]=scoefos[k]/fft_len;
}
/* scoefosA = real(scoefos) */
l = 0;
```

```
for (k = 0; k<=len_band*i_lastcoef-1; k++) {
scoefosA[k] = scoefos[l]; // scoefos real parts only
l=l+2;
}
m_progress.SetPos(60); // PROG
/* Extracting bits from the significant coefficients */
for (k = 0; k<=chowac-1; k++) {
ll = 0;
for (kk = k*hcoef; kk<=(k+1)*hcoef-1; kk++) {
coefseg[ll] = scoefosA[kk];
ll++;
}
i_bitt = DetBit(coefseg, hcoef);
_itoa(i_bitt, bitt, 10);
bitts[k] = bitt[0];
}
m_progress.SetPos(70); // PROG
/* Conversion of bits to ASCII characters */
for (k = 0; k<=chowac/8-1; k++) {
ll = 0;
for (kk = k*8; kk<=(k+1)*8-1; kk++) {
letter[ll] = bitts[kk];
ll++;
}
num = BinToDec(letter);
datout[k] = num;
}
m_dout = datout;

m_progress.SetPos(80); // PROG
SetDlgItemText(IDC_DOUT,m_dout.Left(chowac/8));

m_progress.SetPos(90); // PROG

delete [] yell;
delete [] imig;
delete [] trans;
delete [] transT;
delete [] block;
delete [] blockT;
delete [] res;
```

```
delete [] band;
delete [] scoef;
delete [] oseq;
delete [] order;
delete [] scoefo;
delete [] ang;
delete [] scoefoA;
delete [] SCR;
delete [] scoefos;
delete [] scoefosA;
delete [] coefseg;
m_progress.SetPos(100); // PROG
SetDlgItemText(IDC_MSG,"");
}
/* Function Definitions */
int DetBit(double seg[], int size)
/* Determines the bit value */
{
int k, chk;
/* Checking signs */
chk = 0;
for(k=0; k<=size-1; k++) {
if (seg[k]<0) {
chk = chk - 1;
}
else {
if (seg[k]>0) {
chk = chk + 1;
}
}
}
if (chk>0) {
return 1;
}
else {
return 0;
}
}
int BinToDec(char binar[])
/* Convert an 8-bit binary number to integer */
{
```

```
int sum = 0;
int k;
for(k = 0; k<=7; k++) {
if (binar[k]=='1') {
sum = sum + int(pow(2,7-k));
}
}
if (sum<=31) {
sum = 126;
}
return sum;
}
/* End of Function Definitions */
```

APPENDIX B

DHS5 C/C++ AND HTML PROGRAM LISTINGS

HTML and C/C++ programs for DFT-based watermarking algorithm are given here.

index.html - main screen:

```
<HTML>
<HEAD>
<TITLE>Data Hiding Engine</TITLE>
</HEAD>
<BODY BGCOLOR=008040 TEXT=FFFFFF LINK=00FF00 VLINK=FF0080>
<TABLE>
<TR><TD><h1>Data Hiding Engine</h1></TD></tr>
<TR><TD><small> <I>Arkadiusz Edward Komenda</I> &#183;
email: <a
href="mailto:aek4692@megahertz.njit.edu">aek4692@megahertz.njit.edu</a>
&#183 web: <a
href="http://www-ec.njit.edu/~aek3247/"
>http://www-ec.njit.edu/~aek3247/</a><br>
</small>
</TD></TR></TABLE>
<hr>
<CENTER>
<TABLE>
<TR><TD><FONT SIZE = +1><B>Image:</B></FONT></TD></TR>
<TR><TD><B><A HREF="image_h.html">Hide Data</A></B></TD></TR>
<TR><TD><B><A HREF="image_d.html">Detect Data</A></B></TD></TR>
</TABLE>
</CENTER>
<P><P><P>
<CENTER>
<A HREF=http://njcmr.org/>
<IMG BORDER=0 WIDTH=150 HEIGHT=75 SRC="njcmrembed2.gif"></CENTER>
</A>
<BR><BR>
<P>
<CENTER><FONT SIZE = +1><B>Available images for
testing:</B></FONT></CENTER>
<p>
<CENTER>
<TABLE border = 5>
```

```
<TR>
<TD><B>Grayscale 256x256:</B></TD>
<TD><B>Grayscale 512x512:</B></TD>
<TD><B>Color 256x256:</B></TD>
</TR>
<TR>
<TD>
<p>airplane1.pgm
<p>baboon1.pgm
<p>couple1.pgm
<p>girl.pgm<p>
girl2.pgm<p>
girl3.pgm<p>
house1.pgm<p>
lena1.pgm<p>
peppers1.pgm<p>
sailboat1.pgm<p>
splash1.pgm<p>
tiffany1.pgm<p>
tree1.pgm<p>
</TD>
<TD>
airplane.pgm<p>
baboon.pgm<p>
lena.pgm<p>
peppers.pgm<p>
sailboat.pgm<p>
splash.pgm<p>
</TD>
<TD>
air4.ppm<p>
bab4.ppm<p>
coup4.ppm<p>
gi4.ppm<p>
girl24.ppm<p>
girl34.ppm<p>
hous4.ppm<p>
lena4.ppm<p>
pep4.ppm<p>
sail4.ppm<p>
spl4.ppm<p>
```

```
tiff4.ppm<p>
tree4.ppm <p>
</TD>
</TR>
</TABLE>
</CENTER>
</body>
</html>
```

img_h.c.cgi - data hiding subprogram: /* Filename: img_h.c.cgi

```
Written by: Arkadiusz Edward Komenda
/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <malloc.h>
#define ISIZE 1000 /* max image size: m_horiz x ISIZE */
#define _max(a,b) (((a) > (b)) ? (a) : (b))
#define _min(a,b) (((a) < (b)) ? (a) : (b))
#define _toascii(_c) ( (_c) & 0x7f )
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
/* this is the structure we use for the CGI variables */
struct {
char name[20];
char val[7000];
} elements[7];
/* forward CGI data processing declarations */
void splitword(char *out, char *in, char stop);
char x2c(char *x);
void unescape_url(char *url);
/* Function Prototypes (main program) */
void err_input(char* );
int ch_blank(char* );
int chk_pow2(int);
int round(float);
int bound(int, int, int);
void ddft(float [][ISIZE], int, int, double [][34], double [][34]);
void tnsps34(float [][34], float [][34]);
void vec_mult34(double [][34], double [][34], float [][34], int);
void chk_zero(double [][34], int);
```

```
void iddft(float [][ISIZE], int, int, double [][34], double [][34]);
void fsub(double [][34], double [][34], double [][34], int);
void ifsub(double [][34], double [][34], double [][34], int);
void sortArray(int [], int, int []);
void addsig(double [], int, int, int);
void out_scr(char*);

/* BEGINNING OF PROGRAM ***********************************/
main(int argc, char ** argv)
{
/* for CGI data processing */
char * ct; /* for content-type */
char * cl; /* for content-length */
int icl; /* content-length */
char qs[1000]; /* query string */
int rc;
int i;
/* program data declatations (input variables from webpage) */
int c_mode, m_horiz, m_vert;
char * m_fin, * m_fout, * m_din;
char chky[22], foutpath[50];
/* program data declatations */
int k,l;
unsigned char num;
float red, gre, blu;
char line1[102], line2[350];
char *token;
FILE *f1Ptr; /* f1Ptr = input file pointer */
int band_num;
double nb;
int n_size, nbI;
unsigned int sd;
int din_len, len, seg_len, no_seg;
/* send the mime-type header */
printf("Content-type: text/html\n\n");
/* CGI BIN DATA PROCESSING ************************************/
/* grab the content-type and content-length
and check them for validity */
ct = getenv("CONTENT_TYPE");
cl = getenv("CONTENT_LENGTH");
if(cl == NULL)
```

```
{
err_input("content-length is undefined!");
exit(1);
}
icl = atoi(cl);
if(strcmp(ct, "application/x-www-form-urlencoded"))
{
err_input("I don't understand the content-type");
exit(1);
}
else if (icl == 0)
{
err_input("content-length is zero");
exit(1);
}
if((rc = fread(qs, icl, 1, stdin)) != 1)
{
err_input("cannot read the input stream! Contact the webmaster");
exit(1);
}
qs[icl] = '\0';
/* split out each of the parameters from the
query stream */
for(i = 0; qs[0] != '\0'; i++)
{
/* first divide by '&' for each parameter */
splitword(elements[i].val, qs, '&');
/* convert the string for hex characters and pluses */
unescape_url(elements[i].val);
/* now split out the name and value */
splitword(elements[i].name, elements[i].val, '=');
}
/* END OF CGI BIN DATA PROCESSING ****************************/
/* REASSIGNMENT OF INPUT DATA ********************************/
c_mode = atoi(elements[0].val);
// N = (int) pow(2,atoi(elements[1].val));
m_fin = elements[1].val;
m_horiz = atoi(elements[2].val);
m_vert = atoi(elements[3].val);
m_fout = elements[4].val;
m_din = elements[5].val;
```

```
sd = 12345; /* seed for scattering */
seg_len = 2048; /* length of a segment of bits hidden at one time */
/* END OF INPUT DATA REASSIGNMENT ***************************/
/* DATA VALIDATION ****************************************/

/* Input and validation of variables from the dialog box */

if (ch_blank(m_fin)==1) {
err_input("Error - Enter input file name");
exit(1);
}

if (ch_blank(m_fout)==1) {
err_input("Error - Enter output file name");
exit(1);
}

if (*m_din==0) {
err_input("Error - Enter data for hiding");
exit(1);
}

if (m_horiz<=0) {
err_input("Error - Enter width");
exit(1);
}
if (m_vert<=0) {
err_input("Error - Enter height");
exit(1);
}

/* END OF DATA VALIDATION *********************************/
/* MEMORY ALLOCATION **************************************/
/* Dynamic memory allocation on the heap */
float (*yell)[ISIZE] = new float[m_horiz][ISIZE];
if (yell == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
float (*cb)[ISIZE] = new float[m_horiz][ISIZE];
if (cb == NULL) {
```

```
err_input("Error - Not enough memory");
exit(1);
}
float (*cr)[ISIZE] = new float[m_horiz][ISIZE];
if (cr == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
/* unsigned char (*imig)[ISIZE] = new unsigned char[m_horiz][ISIZE];
if (imig == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
/
float (*t34)[34] = new float[34][34];
if (t34 == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
float (*t34T)[34] = new float[34][34];
if (t34T == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
/* number of bands */
band_num = int(floor(m_horiz/float(8)))*(int(floor(m_vert/float(8))));

double (*mag_dft)[34] = new double[band_num][34];
if (mag_dft == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*ang_dft)[34] = new double[band_num][34];
if (ang_dft == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*mm)[34] = new double[band_num][34];
if (mm == NULL) {
err_input("Error - Not enough memory");
exit(1);
```

```
}
/* number of useful bands of subband coefficients */
n_size = band_num;
nb = sqrt(n_size);
while ((nb!=floor(nb))——(chk_pow2(int(nb))!=0)) {
n_size--;
nb = sqrt(n_size);
}
nbI = int(nb);
double (*mus)[34] = new double[16][34];
if (mus == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*ms)[34] = new double[nbI*nbI-16][34];
if (ms == NULL) {
err_input("Error - Not enough memory");

exit(1);
}
double (*vec) = new double[30*(nbI*nbI-16)];
if (vec == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
int (*oseq) = new int[30*(nbI*nbI-16)];
if (oseq == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
int (*order) = new int[30*(nbI*nbI-16)];
if (order == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*vecro) = new double[30*(nbI*nbI-16)];
if (vecro == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*vecS) = new double[30*(nbI*nbI-16)];
```

```
if (vecS == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
/* length of data to be hidden */
din_len = strlen(m_din);
len = 30*(nbI*nbI-16);
no_seg = int(floor(len/seg_len));
int (*bitseq) = new int[no_seg];
if (bitseq == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*seg) = new double[seg_len];
if (seg == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*mag_dftS)[34] = new double[band_num][34];
if (mag_dftS == NULL) {
err_input("Error - Not enough memory");
exit(1);
}

/* END OF MEMORY ALLOCATION ******************************/
/* READING IMAGE INPUT FILE ******************************/
/* if c_mode = 3 check for .Y or .y */
strcpy( chky, "");
strcat( chky, m_fin);
token = strtok(chky,".");
token = strtok(NULL,".");
if ((c_mode==3)&&((strcmp(token,"y\0")!=0)&&(strcmp(token,"Y\0")!=0))) {
err_input("Error - Input file of wrong format");
exit(1);
}
if ((f1Ptr = fopen(m_fin, "rb")) == NULL) {
err_input("Error - Input file could not be found");
exit(1);
}
else {
if (c_mode==3) {
```

```
/* .Y or .y */
/* getting data */
for (l = 0; l <=m_vert-1; l++) {
for (k = 0; k <=m_horiz-1; k++) {
num = fgetc(f1Ptr);
// imig[k][l] = num;
yell[k][l] = float(num);
}
}
}
else {
if (c_mode==2) {
/* get line with P6 */
fgets(line1, 101, f1Ptr);
if (line1[1]!='6') {
err_input("Error - Input file of wrong format");
fclose(f1Ptr);
exit(1);
}
/* get next line */
fgets(line1, 101, f1Ptr);

/* get remaining lines starting with # and one line after them */
while (line1[0] == '#') {
fgets(line1, 101, f1Ptr);
}
/* getting rid of width, height and no. of colors */
token = strtok (line1," ");
token = strtok (NULL," ");
token = strtok (NULL," ");
/* supporting 2 styles of P6 files */
if (token == 0) {
fgets(line1, 101, f1Ptr);
}
/* getting data */
for (l = 0; l <=m_vert-1; l++) {
for (k = 0; k <=m_horiz-1; k++) {
num = fgetc(f1Ptr);
red = float(num);
num = fgetc(f1Ptr);
gre = float(num);
```

```
num = fgetc(f1Ptr);
blu = float(num);
/* Convert to other color domain */
yell[k][l] = float(0.3*red+0.6*gre+0.1*blu);
cr[k][l] = float(0.45*blu-0.15*red-0.3*gre+0.5);
cb[k][l] = float(0.475*red-0.35*gre-0.065*blu+0.5);
}
}
}
else {
/* get line with P5 */
fgets(line1, 101, f1Ptr);
if (line1[1]!='5') {
err_input("Error - Input file of wrong format");
fclose(f1Ptr);
exit(1);
}
/* get next line */
fgets(line1, 101, f1Ptr);

/* get remaining lines starting with # and one line after them */
while (line1[0] == '#') {
fgets(line1, 101, f1Ptr);
}
/* getting rid of width, height and no. of colors */
token = strtok (line1," ");
token = strtok (NULL," ");
token = strtok (NULL," ");
/* supporting 2 styles of P5 files */
if (token == 0) {
fgets(line1, 101, f1Ptr);
}
/* getting data */
for (l = 0; l <=m_vert-1; l++) {
for (k = 0; k <=m_horiz-1; k++) {
num = fgetc(f1Ptr);
// imig[k][l] = num;
yell[k][l] = float(num);
}
}
}
```

```
}
}
fclose(f1Ptr);

/* END OF READING IMAGE INPUT FILE **********************/
/* LOAD TABLE T34 ***************************************/
if ((f1Ptr = fopen("t34mat.txt", "r")) == NULL) {
err_input("Error - Input file for table T34 could not be found");
exit(1);
}
else {
for (k=0;k<=33;k++) {
/* get line */
fgets(line2, 349, f1Ptr);
token = strtok(line2,",");
t34[k][0] = float(atof(token));
for (l=1;l<=33;l++) {
token = strtok(NULL,",");
t34[k][l] = float(atof(token));
}
}
}
fclose(f1Ptr);
/* END OF LOAD TABLE T34 *********************************/
ddft(yell, m_horiz, m_vert, mag_dft, ang_dft);
tnsps34(t34T, t34);
vec_mult34(mm, mag_dft, t34T, band_num);
fsub(mus, ms, mm, nbI);
i = 0;
for(l=0;l<=nbI*nbI-16-1;l++) {
for(k=4;k<=33;k++) {
vec[i] = ms[l][k];
i++;
}
}
/* RANDOM SEQUENCE GENERATION AND SCATTERING **************/
srand(sd);
for (k = 0; k<=30*(nbI*nbI-16)-1; k++) {
oseq[k] = rand();
order[k] = k; /* initializing order array */
}
```

```
/* Sorting of random sequence in ascending order */
sortArray(oseq, 30*(nbI*nbI-16), order);

/* Scattering of coefficients */
for (k = 0; k<=30*(nbI*nbI-16)-1; k++) {
vecro[k] = vec[order[k]];
}
/* END OF RANDOM SEQUENCE GENERATION AND SCATTERING *******/
/* HIDING DATA ******************************************/
for (k = 0; k <= no_seg-1; k++) {
if (k<=din_len-1) {
bitseq[k] = __toascii(m_din[k]);
}
else {
bitseq[k] = 32; /* space */
}
}
for(i=0;i<=no_seg-1;i++) {
l = 0;
for (k = (i*seg_len); k<=(seg_len*(i+1))-1; k++) {
seg[l] = vecro[k];
l++;
}
addsig(seg, seg_len, sd, bitseq[j]);

l = 0;
for (k = (i*seg_len); k<=(seg_len*(i+1))-1; k++) {
vecro[k] = seg[l];
l++;
}
}

/* END OF HIDING DATA ***********************************/
/* UNSCATTERING OF COEFFICIENTS *************************/
for (k = 0; k<=30*(nbI*nbI-16)-1; k++) {
vecS[order[k]] = vecro[k];
}
/* END OF UNSCATTERING OF COEFFICIENTS ******************/

i = 0;
for(l=0;l<=nbI*nbI-16-1;l++) {
```

```
for(k=4;k<=33;k++) {
ms[l][k] = vecS[i] ;
i++;
}
}
ifsub(mm, mus, ms, nbl);
vec_mult34(mag_dftS, mm, t34, band_num);
chk_zero(mag_dftS, band_num);
iddft(yell, m_horiz, m_vert, mag_dftS, ang_dft);

/* WRITING IMAGE TO THE OUTPUT FILE ************************/
/* path where to put the output file */
strcpy( foutpath, "./output/"); /* path */
strcat( foutpath, m_fout);
if ((f1Ptr = fopen(foutpath, "wb")) == NULL) {
err_input("Error - Output file could not be opened");
exit(1);
}
else {
if (c_mode==3) {
/* .Y or .y */
/* writing data */
for (l = 0; l <=m_vert-1; l++) {
for (k = 0; k <=m_horiz-1; k++) {
// num = imig[k][l];
num=bound(round(float(yell[k][l])), 0, 255);
fputc(num,f1Ptr);
}
}
}
else {
if (c_mode==2) {
fprintf(f1Ptr, "%s", "P6\n");
fprintf(f1Ptr, "%s", "# Created using: DHS Version: 2.1\n");
fprintf(f1Ptr, "%s", "# Copyright: Arkadiusz Komenda\n");
fprintf(f1Ptr, "%d%s%d%s", m_horiz, " ", m_vert, " 255\n");
for (l = 0; l <=m_vert-1; l++) {
for (k = 0; k <=m_horiz-1; k++) {
/* red */
num=bound(round(float(yell[k][l]+1.6*cb[k][l]-0.8)), 0, 255);
fputc(num,f1Ptr);
```

```
/* green */
num=bound(round(float(yell[k][l]-(float(1)/3)*cr[k][l]+(float(1)/6)-0.8*cb[k][l]+0.4)), 0, 255);
fputc(num,f1Ptr);
/* blue */
num=bound(round(float(yell[k][l]+2*cr[k][l]-1.0)), 0, 255);
fputc(num,f1Ptr);
}
}
}
else {
fprintf(f1Ptr, "%s", "P5\n");
fprintf(f1Ptr, "%s", "# Created using: DHS Version: 2.1\n");
fprintf(f1Ptr, "%s", "# Copyright: Arkadiusz Komenda\n");
fprintf(f1Ptr, "%d%s%d%s", m_horiz, " ", m_vert, " 255\n");
for (l = 0; l <=m_vert-1; l++) {
for (k = 0; k <=m_horiz-1; k++) {
num=bound(round(float(yell[k][l])), 0, 255);
// num = imig[k][l];
fputc(num,f1Ptr);
}
}
}
}
}
fclose(f1Ptr);
/* END OF WRITING IMAGE TO THE OUTPUT FILE ***************/
/* MEMORY DEALLOCATION ************************************/
/* Dynamic memory deallocation */
delete [] yell;
delete [] cb;
delete [] cr;
// delete [] imig;
delete [] t34;
delete [] t34T;
delete [] mag_dft;
delete [] ang_dft;
delete [] mm;
delete [] mus;
delete [] ms;
delete [] vec;
delete [] oseq;
```

```
delete [] order;
delete [] vecro;
delete [] vecS;
delete [] bitseq;
delete [] seg;
delete [] mag_dftS;

/* END OF MEMORY DEALLOCATION ****************************/
out_scr(m_fout); /* final screen */
return 0;
}
/* END OF PROGRAM ***************************************/
/* SUBROUTINES ******************************************/
/* CGI function definitions *****************************/
void splitword(char *out, char *in, char stop)
{
int i, j;
for(i = 0; in[i] && (in[i] != stop); i++)
out[i] = in[i];
out[i] = '\0'; /* terminate it */
if(in[i])
++i;
for(j = 0; in[j]; ) /* shift the rest of the in */
in[j++] = in[i++];
}
char x2c(char *x)
{
register char c;
/* note: (x & 0xdf) makes x upper case */
c = (x[0] >= 'A' ? ((x[0] & 0xdf) - 'A') + 10 : (x[0] - '0'));
c *= 16;
c += (x[1] >= 'A' ? ((x[1] & 0xdf) - 'A') + 10 : (x[1] - '0'));
return(c);
}

/* this function goes through the URL char-by-char
and converts all the "escaped" (hex-encoded)
sequences to characters
this version also converts pluses to spaces. I've
seen this done in a separate step, but it seems
```

```
to me more efficient to do it this way. -wew
/
void unescape_url(char *url)
{
register int i, j;
for(i = 0, j = 0; url[j]; ++i, ++j)
{
if((url[i] = url[j]) == '%')
{
url[i] = x2c(&url[j + 1]);
j += 2;
}
else if (url[i] == '+')
url[i] = ' ';
}
url[i] = '\0'; /* terminate it at the new length */
}
/* End of CGI function definitions ************************/
/* Input Error Screen function definition ******************/
void err_input(char* msg)
/* Multipurpose error screen for displaying error mesages
msg - error message
/
{
puts(
"<HTML><HEAD><TITLE>Data Hiding Engine - Error</TITLE></HEAD>"
"<BODY BGCOLOR=008040 TEXT=FFFFFF LINK=00FF00 VLINK=FF0080>"
"<TABLE>"
"<TR><TD><h1>Data Hiding Engine - Error</h1></TD></tr>"
"<TR><TD><small> <I>Arkadiusz Edward Komenda</I> &#183;"
"email: <a href=mailto:aek4692@megahertz.njit.edu>aek4692@megahertz.njit.edu</a> &#183;"
"web: <a href=http://megahertz.njit.edu/~aek4692>http://megahertz.njit.edu/~aek4692</a><br>"
"</small>"
"</TD></TR></TABLE>"
"<hr>"
);
printf("<font color=E6EB1F><center><h2><b>%s</b></h2><p></font>", msg);
puts("<h3>You can use Back button in your browser to return to your form"
"</h3></center>");
puts("</BODY></HTML>");
return;
```

```
}
/* End of Input Error Screen function definition **********/


/* Function Definitions (main program) *********************/
int ch_blank(char* data)
/* Check if a string consists of spaces only
data - string to be checked
Return: 0 = not white spaces only
1 = white spaces only
/
{
if (strlen(data)==strspn(data," "))
return 1;
else
return 0;
}
int round(float num)
/* Rounding floating numbers into integers
num - floating number to be rounded
Return: integer after rounding operation
/
{
int whole, out;
float test;
whole = int(floor(num));
test = float(whole) + float(0.5);
if (num >= test) {
out = whole + 1;
}
else {
out = whole;
}
return out;
}
int bound(int a, int lowerbound, int upperbound)
/* Bounding the integer between the lower and upper limits
a - integer to be bounded
lowerbound - lower limit of the range
upperbound - upper limit of the range
Return: an integer that is bigger or equal to the lower limit and
```

```
at the same time lower or equal to the upper limit of the range
/
{
int b;

b=__min(a,upperbound);
b=__max(a,lowerbound);
return b;
}
void ddft(float arr[][ISIZE], int horiz, int vert, double mag[][34], double ang[][34])
/* This function obtains the magnitude and angle of the DFT
coefficients of arr of size horizXvert. Block size 8x8.

Number of bands of magnitude DFT is 8^2/2 + 2
Inputs:
arr - floating data array
horiz - width of arr array (integer)
vert - height of arr array (integer)
Outputs:
mag - array containing DFT magnitudes of arr elements
and - array containing DFT angles of arr elements
/
{
int row, col, k, l, kk, ll, k1, l1, cntR, cntC;
double block_r[8][8], block_i[8][8];
void fft2(double [][8], double [][8], int, int, int, int, int);
cntR = 0;
for (kk = 0; kk<=int(floor(horiz/float(8))-1); kk++) {
for (ll = 0; ll<=int(floor(vert/float(8))-1); ll++) {
row = 0;
for (k = (kk*8); k<=(8*(kk+1))-1; k++) {
col = 0;
for (l = (ll*8); l<=(8*(ll+1))-1; l++) {
block_r[row][col] = arr[k][l];
block_i[row][col] = 0;
col++;
}
row++;
}
fft2(block_r,block_i,8,8,3,3,0); /* forward 2D Fourier transform */
cntC = 0;
```

```
for (l1=0;l1<=4;l1++) {
for (k1=0;k1<=4;k1++) {
mag[cntR][cntC] = sqrt(pow(block_r[k1][l1],2.0)+pow(block_i[k1][l1],2.0))/8;
ang[cntR][cntC] = atan2(block_i[k1][l1],block_r[k1][l1]);
cntC++;
}
}
for (l1=1;l1<=3;l1++) {
for (k1=5;k1<=7;k1++) {
mag[cntR][cntC] = sqrt(pow(block_r[k1][l1],2.0)+pow(block_i[k1][l1],2.0))/8;
ang[cntR][cntC] = atan2(block_i[k1][l1],block_r[k1][l1]);
cntC++;
}
}
cntR++;
}
}
return;
}
void fft2(double x2[][8], double y2[][8], int n1, int n2, int m1, int m2, int idir)
/* x2 - real part (2D array of size n1Xn2)
y2 - imaginary part (2D array of size n1Xn2)
m1 = log2(n1)
m2 = log2(n2)
idir - direction of the transformation (0 - forward , 1 - reverse)
/
{
double *x,*y;
int i,j;

void srfft(double [], double [], int, int);
void isrfft(double [], double [], int, int);

x = (double *) calloc(n2,sizeof(double)); /* allocate buffers */
y = (double *) calloc(n2,sizeof(double));

/* ——computing the DFT of the rows of the picture matrix—— */
for (i = 0; i < n1 ; i++)
{

for(j = 0 ; j < n2 ; j++)
```

```
{
x[j] = x2[i][j];
y[j] = y2[i][j];
}

if (idir == 0 ) srfft(x,y,n2,m2);
if (idir == 1 ) isrfft(x,y,n2,m2);
/* the intermediate matrix */

for(j = 0 ; j < n2 ; j++)
{
x2[i][j] = x[j];
y2[i][j] = y[j];
}
}

free(x);free(y);
x = (double *) calloc(n1,sizeof(double)); /* allocate buffers */
y = (double *) calloc(n1,sizeof(double));
/* —computing the DFT of the columns of the intermediate matrix— */

for (j = 0; j < n2 ; j++)
{
for(i = 0 ; i < n1 ; i++)
{
x[i] = x2[i][j];
y[i] = y2[i][j];
}

if (idir == 0) srfft(x,y,n1,m1);
if (idir == 1) isrfft(x,y,n1,m1);

for(i = 0 ; i < n1 ; i++)
{
x2[i][j] = x[i];
y2[i][j] = y[i];
}
}
// free(*x);free(*y);
free(x); free(y);
}
```

```
void srfft(double x[], double y[], int n, int m)
{

int i,j,k,n2,n4,is,id,i0,i1,i2,i3;
double a,a3,e,r1,r2,s1,s2,s3,cc1,ss1,cc3,ss3,xt;


n2 = 2 * n;
for (k = 1;k <= m-1; k++)
{
n2 = n2/2;
n4 = n2/4;
e = 6.283185307179586/n2;
a = 0.0;
for (j = 1; j <= n4 ; j++)
{
a3 = 3.0 * a;

cc1 = cos(a);
ss1 = sin(a);
cc3 = cos(a3);
ss3 = sin(a3);

a = j * e;
is = j;
id = 2 * n2;
do
{
for (i0 = is-1 ; i0 <= n-1 ; i0 += id)
{
i1 = i0 + n4;
i2 = i1 + n4;
i3 = i2 + n4;

r1 = x[i0] - x[i2] ;
x[i0] += x[i2];
r2 = x[i1] - x[i3] ;
x[i1] += x[i3];

s1 = y[i0] - y[i2] ;
y[i0] += y[i2];
```

```
s2 = y[i1] - y[i3] ;
y[i1] += y[i3];

s3 = r1 - s2;
r1 += s2;
s2 = r2 - s1;
r2 += s1;

x[i2] = r1 * cc1 - s2 * ss1;
y[i2] = -s2 * cc1 - r1 * ss1;
x[i3] = s3 * cc3 + r2 * ss3;
y[i3] = r2 * cc3 - s3 * ss3;
}

is = 2 * id - n2 + j;
id *= 4;
} while (is < n);
}
}
/* */
/* ----------last stage, length-2 butterfly---------- */
/* */
is = 1;
id = 4;
do
{
for (i0 = is ; i0 <= n ; i0 += id)
{
i0--;
i1 = i0 + 1;

r1 = x[i0];
x[i0] = r1 + x[i1];
x[i1] = r1 - x[i1];

r1 = y[i0];
y[i0] = r1 + y[i1];
y[i1] = r1 - y[i1];
i0++;
}
is = 2 * id - 1;
```

```
id *= 4;
} while (is < n);
/* */
/* ————————bit reverse counter———————— */
/* */
j = 1;
for (i = 1; i <= n-1 ; i ++)
{
if (i < j)
{
j--; i--;
xt = x[j]; x[j] = x[i]; x[i] = xt;
xt = y[j]; y[j] = y[i]; y[i] = xt;
j++; i++;
}
k = n/2;
while (k < j)
{
j -= k;
k = k/2;
}
j += k;
}
}
void isrfft(double x[], double y[], int n, int m)
{
int i;
double n1;
void srfft(double [], double [], int, int);
n1 = 1. / ((double) n); /* scale factor */

srfft(y,x,n,m); /* perform a foward DFT */
for (i = 0; i < n ; i++)
{
x[i] = x[i] * n1;
y[i] = y[i] * n1;
}
}
void tnsps34(float arrT[][34], float arr[][34])
```

```
/* Transpose of a square matrix of size: 34X34
Input: arr - matrix to be transposed (float)
Output: arrT - transposed matrix arr (float)
/
{
int l, k;
for (l = 0; l<=33; l++) {
for (k = 0; k<=33; k++) {
arrT[k][l] = arr[l][k];
}
}
return;
}
void vec_mult34(double prod[][34], double arr1[][34], float arr2[][34], int size)
/* Vector product of two matrices
Inputs: arr1 - leftmost array of size: sizeX34 (double)
arr2 - rightmost array of size: 34x34 (float)
size - number of rows in arr1 (integer)
Output: prod - vector product of arr1 and arr2 of size: sizeX34 (double)
/
{
int l1, k1, k2;
double sum;
for (l1 = 0; l1<=size-1; l1++) {
for (k2 = 0; k2<=33; k2++) {
sum = 0;
for (k1 = 0; k1<=33; k1++) {
sum = sum + (arr1[l1][k1] * arr2[k1][k2]);
}
prod[l1][k2] = sum;
}
}
return;
}
void chk_zero( double arr[][34], int size)
/* If the element of the array is less than zero then make it zero
Inputs: arr - array to be checked of size: sizeX34 (double)
size - number of rows in arr (integer)
Output: arr - updated array
/
{
```

```
int k, l;
for (k=0; k<=size-1; k++) {
for (l=0; l<=33; l++) {
if (arr[k][l]<0) {
arr[k][l] = 0;
}
}
}
return;
}
void iddft(float arr[][ISIZE], int horiz, int vert, double mag[][34], double ang[][34])
/* Inverse of ddft function.
This function takes the magnitude (mag) and angle (ang) and
converts them into the DFT coefficients' real and imaginary parts.
Block size 8x8. Number of bands of magnitude DFT is 8^2/2 + 2= 34.
Inputs:
mag - array containing DFT magnitudes (double)
and - array containing DFT angles (double)
horiz - width of arr array (integer)
vert - height of arr array (integer)
Output:
arr - data array containing DFT coefficients (float)
/
{
int row, col, k, l, kk, ll, k1, l1, cntR, cntC;
double block_abs[8][8], block_ang[8][8];
double re_part, im_part;
void fft2(double [][8], double [][8], int, int, int, int, int);
cntR = 0;
for (kk = 0; kk<=int(floor(horiz/float(8))-1); kk++) {
for (ll = 0; ll<=int(floor(vert/float(8))-1); ll++) {

cntC = 0;
for (i1=0;l1<=4;l1++) {
for (k1=0;k1<=4;k1++) {
block_abs[k1][l1] = mag[cntR][cntC]*8;
block_ang[k1][l1] = ang[cntR][cntC];
cntC++;
}
}
for (l1=1;l1<=3;l1++) {
```

```
for (k1=5;k1<=7;k1++) {
block_abs[k1][l1] = mag[cntR][cntC]*8;
block_ang[k1][l1] = ang[cntR][cntC];
cntC++;
}
}
block_abs[5][0] = block_abs[3][0];

block_ang[5][0] = -1*block_ang[3][0];
block_abs[6][0] = block_abs[2][0];
block_ang[6][0] = -1*block_ang[2][0];
block_abs[7][0] = block_abs[1][0];
block_ang[7][0] = -1*block_ang[1][0];
block_abs[5][4] = block_abs[3][4];
block_ang[5][4] = -1*block_ang[3][4];
block_abs[6][4] = block_abs[2][4];
block_ang[6][4] = -1*block_ang[2][4];
block_abs[7][4] = block_abs[1][4];
block_ang[7][4] = -1*block_ang[1][4];
block_abs[0][5] = block_abs[0][3];
block_ang[0][5] = -1*block_ang[0][3];
block_abs[0][6] = block_abs[0][2];
block_ang[0][6] = -1*block_ang[0][2];
block_abs[0][7] = block_abs[0][1];
block_ang[0][7] = -1*block_ang[0][1];
block_abs[4][5] = block_abs[4][3];
block_ang[4][5] = -1*block_ang[4][3];
block_abs[4][6] = block_abs[4][2];
block_ang[4][6] = -1*block_ang[4][2];
block_abs[4][7] = block_abs[4][1];
block_ang[4][7] = -1*block_ang[4][1];
block_abs[5][5] = block_abs[3][3];
block_ang[5][5] = -1*block_ang[3][3];
block_abs[5][6] = block_abs[3][2];
block_ang[5][6] = -1*block_ang[3][2];
block_abs[5][7] = block_abs[3][1];
block_ang[5][7] = -1*block_ang[3][1];
block_abs[6][5] = block_abs[2][3];
block_ang[6][5] = -1*block_ang[2][3];
block_abs[6][6] = block_abs[2][2];
block_ang[6][6] = -1*block_ang[2][2];
```

```
block_abs[6][7] = block_abs[2][1];
block_ang[6][7] = -1*block_ang[2][1];
block_abs[7][5] = block_abs[1][3];
block_ang[7][5] = -1*block_ang[1][3];
block_abs[7][6] = block_abs[1][2];
block_ang[7][6] = -1*block_ang[1][2];
block_abs[7][7] = block_abs[1][1];
block_ang[7][7] = -1*block_ang[1][1];
block_abs[1][5] = block_abs[7][3];
block_ang[1][5] = -1*block_ang[7][3];
block_abs[1][6] = block_abs[7][2];
block_ang[1][6] = -1*block_ang[7][2];
block_abs[1][7] = block_abs[7][1];
block_ang[1][7] = -1*block_ang[7][1];
block_abs[2][5] = block_abs[6][3];
block_ang[2][5] = -1*block_ang[6][3];
block_abs[2][6] = block_abs[6][2];
block_ang[2][6] = -1*block_ang[6][2];
block_abs[2][7] = block_abs[6][1];
block_ang[2][7] = -1*block_ang[6][1];
block_abs[3][5] = block_abs[5][3];
block_ang[3][5] = -1*block_ang[5][3];
block_abs[3][6] = block_abs[5][2];
block_ang[3][6] = -1*block_ang[5][2];
block_abs[3][7] = block_abs[5][1];
block_ang[3][7] = -1*block_ang[5][1];
for (l1=0;l1<=7;l1++) {
for (k1=0;k1<=7;k1++) {
/* real part of block_abs and block_ang */
re_part = block_abs[k1][l1]*cos(block_ang[k1][l1]);
/* imaginary part of block_abs and block_ang */
im_part = block_abs[k1][l1]*sin(block_ang[k1][l1]);
/* real part of block_abs and block_ang */
block_abs[k1][l1] = re_part;
/* imaginary part of block_abs and block_ang */
block_ang[k1][l1] = im_part;
}
}

fft2(block_abs,block_ang,8,8,3,3,1); /* inverse 2D Fourier transform */
row = 0;
```

```
for (k = (kk*8); k<=(8*(kk+1))-1; k++) {
col = 0;
for (l = (ll*8); l<=(8*(ll+1))-1; l++) {
/* round and bound the real part */
arr[k][l] = float(block_abs[row][col]);
col++;
}
row++;
}

cntR++;
}
}
return;
}
void fsub(double vecfus[][34], double vecfs[][34], double picf[][34], int nbI)
/* This function obtains the subband coefficients of the
block transform coefficients.
vecfus is not used for data hiding, vecfs is.
Inputs:
picf - array of block transform coefficients (double)
nbI - helps to determine the size of vecfs (integer)
Outputs:
vecfus - array not used for data hiding, of size: 16x34 (double)
vecfs - array used later for data hiding, of size: (nbI*nbI-16)x34 (double)
/
{
int k, l, row, col;
void err_input(char* );
void rsdec(double [][100], int);
double (*segf)[100] = new double[nbI][100];
if (segf == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
for(l=0;l<=33;l++) {
k = 0;
for(col=0;col<=nbI-1;col++) {
for(row=0;row<=nbI-1;row++) {
segf[row][col] = picf[k][l];
k++;
```

```
}
}
rsdec(segf, nbI);
k = 0;
for(col=4;col<=nbI-1;col++) {
for(row=0;row<=3;row++) {
vecfs[k][l] = segf[row][col];
k++;
}
}
for(col=4;col<=nbI-1;col++) {
for(row=4;row<=nbI-1;row++) {
vecfs[k][l] = segf[row][col];
k++;
}
}

for(col=0;col<=3;col++) {
for(row=4;row<=nbI-1;row++) {
vecfs[k][l] = segf[row][col];
k++;
}
}
k = 0;
for(col=0;col<=3;col++) {
for(row=0;row<=3;row++) {
vecfus[k][l] = segf[row][col];
k++;
}
}
}
delete [] segf;
return;
}
void rsdec(double I[][100], int size)
/* Inputs: I - image of size: sizeXsize (double)
size - number of rows and columns in I (integer)
Output: I - modified image of size: sizeXsize (double)
/
{
int m, k, l, kk, ll, row, col;
```

```
int vhs, nd;
double y[64][64];
void sdecrect(double [][64], int, int);
for(m=1;m<=3;m++) {
nd = int(pow(2,m-1));
vhs=size/nd;
for(ll=0;ll<=nd-1;ll++) {
for(kk=0;kk<=nd-1;kk++) {
col = 0;
for (l = (ll*vhs); l<=(vhs*(ll+1))-1; l++) {
row = 0;
for (k = (kk*vhs); k<=(vhs*(kk+1))-1; k++) {
y[row][col] = I[k][l];
row++;
}
col++;
}
sdecrect(y, col, 0);
col = 0;
for (l = (ll*vhs); l<=(vhs*(ll+1))-1; l++) {
row = 0;
for (k = (kk*vhs); k<=(vhs*(kk+1))-1; k++) {
I[k][l] = y[row][col];
row++;
}
col++;
}
}
}
}
return;
}
void sdecrect(double y[][64], int size, int mode)
/* Inputs: y - image array of size: sizeXsize (double)
size - number of columns and rows in y (integer)
mode - switch
mode = 0 forward
mode = 1 inverse of function
Output: y - resulting array (double)
/
{
```

```
int k, l;
void err_input(char* );
void cirshift(double [], double [], int, int);
void vec_mult64(double [][64], double [][64], double [][64], int);
void tnsps64(double [][64], double [][64], int);
double (*h) = new double[size];
if (h == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*hmod) = new double[size];
if (hmod == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*g) = new double[size];
if (g == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*gmod) = new double[size];
if (gmod == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*T)[64] = new double[size][64];
if (T == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*TT)[64] = new double[size][64];
if (TT == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*res)[64] = new double[size][64];
if (res == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
/* Get filter coefficients */
```

```
h[0]=(1+sqrt(3))/4/sqrt(2);
h[1]=(3+sqrt(3))/4/sqrt(2);
h[2]=(3-sqrt(3))/4/sqrt(2);
h[3]=(1-sqrt(3))/4/sqrt(2);
for(k=4;k<=size-1;k++) {
h[k] = 0;
}
g[0]=h[3];
g[1]=-1*h[2];
g[2]=h[1];
g[3]=-1*h[0];
for(k=4;k<=size-1;k++) {
g[k] = 0;
}
for(k=0;k<=size/2-1;k++) {
cirshift(hmod, h, size, 2*k);
cirshift(gmod, g, size, 2*k);
for(l=0;l<=size-1;l++) {
T[k][l] = hmod[l];
T[size/2+k][l] = gmod[l];
}
}
if (mode==0) {
vec_mult64(res, T, y, size);
tnsps64(TT, T, size);
vec_mult64(y, res, TT, size);
}
else {
tnsps64(TT, T, size);
vec_mult64(res, TT, y, size);
vec_mult64(y, res, T, size);
}
delete [] h;
delete [] hmod;
delete [] g;
delete [] gmod;
delete [] T;
delete [] TT;
delete [] res;
return;
}
```

```
void tnsps64(double arrT[][64], double arr[][64], int size)
/* Transpose of a square matrix of size: sizeXsize
Input: arr - matrix to be transposed (double)
Output: arrT - transposed matrix arr (double)
/
{
int l, k;
for (l = 0; l<=size-1; l++) {
for (k = 0; k<=size-1; k++) {
arrT[k][l] = arr[l][k];
}
}
return;
}
void vec_mult64(double prod[][64], double arr1[][64], double arr2[][64], int size)
/* Vector product of two matrices

Inputs: arr1 - leftmost array of size: sizeXsize (double)
arr2 - rightmost array of size: sizeXsize (double)
size - number of rows and columns in arr1 and arr2 (integer)
Output: prod - vector product of arr1 and arr2 of size: sizeXsize (double)
/
{
int l1, k1, k2;
double sum;
for (l1 = 0; l1<=size-1; l1++) {
for (k2 = 0; k2<=size-1; k2++) {
sum = 0;
for (k1 = 0; k1<=size-1; k1++) {
sum = sum + (arr1[l1][k1] * arr2[k1][k2]);
}
prod[l1][k2] = sum;
}
}
return;
}
void cirshift(double n_vect[], double vect[], int size, int shft)
/* Shift left to right elements of a vector in a circular manner
Inputs:
vect - vector to be shifted of length: size (double)
size - length of a vector (integer)
```

```
shft - number of spaces the vector should be shifted by (integer)
Output:
n_vect - new vector of length: size, after the shift operation (double)
/
{
int k, l;
double dummy;
for(l=0;l<=size-1;l++) {
n_vect[l] = vect[l];
}
if (shft!=0) {
for(l=0;l<=shft-1;l++) {
dummy = n_vect[size-1];
for(k=size-1;k>=1;k--) {
n_vect[k] = n_vect[k-1];
}
n_vect[0] = dummy;
}
}

return;
}
int chk_pow2(int num)
/* check whether an integer is a power of 2 or not */
/* Return values: */
/* flag = 0 : integer is a power of 2 */
/* flag = 1 : integer is NOT a power of 2 */
{
int flag = 0;
while ((num>1)&&(flag==0)) {
if ((num % 2)==0) {
flag = 0;
}
else {
flag = 1;
}
num = num/2;
}
return flag;
}
void ifsub(double picf[][34], double vecfus[][34], double vecfs[][34], int nbI)
```

```
/* Inverse of fsub() subroutine
Inputs:
vecfus - array not used for data hiding, of size: 16x34 (double)
vecfs - array used for data hiding, of size: (nbI*nbI-16)x34 (double)
nbI - helps to determine the size of vecfs (integer)
Output:
/
{
int k, l, row, col;
void err_input(char* );
void irsdec(double [][100], int);
double (*segf)[100] = new double[nbI][100];
if (segf == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
for(l=0;l<=33;l++) {
k = 0;
for(col=4;col<=nbI-1;col++) {
for(row=0;row<=3;row++) {
segf[row][col] = vecfs[k][l];
k++;
}
}
for(col=4;col<=nbI-1;col++) {
for(row=4;row<=nbI-1;row++) {
segf[row][col] = vecfs[k][l];
k++;
}
}
for(col=0;col<=3;col++) {
for(row=4;row<=nbI-1;row++) {
segf[row][col] = vecfs[k][l];
k++;
}
}
k = 0;
for(col=0;col<=3;col++) {
for(row=0;row<=3;row++) {
segf[row][col] = vecfus[k][l];
k++;
```

```
}
}
irsdec(segf, nbI);
k = 0;
for(col=0;col<=nbI-1;col++) {
for(row=0;row<=nbI-1;row++) {
picf[k][l] = segf[row][col];
k++;
}
}
}
delete [] segf;
return;
}
void irsdec(double I[][100], int size)
/* Inverse of rsdec() subroutine
Inputs: I - image of size: sizeXsize (double)
size - number of rows and columns in I (integer)

Output: I - modified image of size: sizeXsize (double)
/
{
int m, k, l, kk, ll, row, col;
int vhs, nd;
double y[64][64];
void sdecrect(double [][64], int, int);
for(m=3;m>=1;m--) {
nd = int(pow(2,m-1));
vhs=size/nd;
for(ll=0;ll<=nd-1;ll++) {
for(kk=0;kk<=nd-1;kk++) {
col = 0;
for (l = (ll*vhs); l<=(vhs*(ll+1))-1; l++) {
row = 0;
for (k = (kk*vhs); k<=(vhs*(kk+1))-1; k++) {
y[row][col] = I[k][l];
row++;
}
col++;
}
sdecrect(y, col, 1);
```

```
col = 0;
for (l = (ll*vhs); l<=(vhs*(ll+1))-1; l++) {
row = 0;
for (k = (kk*vhs); k<=(vhs*(kk+1))-1; k++) {
I[k][l] = y[row][col];
row++;
}
col++;
}
}
}
}
return;
}
void sortArray(int orig[], int size, int ind[])
/* Sort array in ascending order and memorize old positions of elements
prior to sorting
Inputs: orig - data vector of length: size (integer)
ind - vector of length: size holding the index of data
vector, orig (integer)
size - length of vectors orig and ind (integer)
Outputs: orig - sorted vector
ind - vector holding initial orig index before sorting
/
{
int k, l;
int temp, itemp;
for (k = size-2; k>=0; k--) {
for (l = 0; l<=k; l++) {
if (orig[l]>orig[l+1]) {
temp = orig[l];
orig[l] = orig[l+1];
orig[l+1] = temp;
itemp = ind[l];
ind[l] = ind[l+1];
ind[l+1] = itemp;
}
}
}

return;
```

```
}
void addsig(double segA[], int segA_len, int sdd, int daat)
/* Hiding data
Inputs:
segA - data vector of length: segA_len, inside which data will be hidden (double)
segA_len - length of the data vector (integer)
sdd - seed for gaussian random number generator (integer)
daat - number to hide (integer)
Output:
segA - segment containing hidden information (double)
/
{
int k, l;
double segA_norm, ip;
void err_input(char* );
void gaussian(float [][1], float , int, int, int);
void four1(double [], unsigned long, int);
void getcyc(double [], double [], int, int);
double (*segB) = new double[segA_len*2];
if (segB == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*segB_ang) = new double[segA_len];
if (segB_ang == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*segC) = new double[segA_len];
if (segC == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*y) = new double[segA_len];
if (y == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*v) = new double[segA_len];
if (v == NULL) {
err_input("Error - Not enough memory");
```

```
exit(1);
}
float (*h)[1] = new float[segA_len][1];
if (h == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
/* generating signature */
gaussian(h, 1, sdd, segA_len, 1);

/* Preparing h for FFT */
l = 0;
for (k = 0; k<=2*segA_len-1; k+=2) {
if (h[l][1]<0) {
segB[k] = -1;
}
else {
segB[k] = 1;
}
segB[k+1] = 0;
l++;
}
/* FFT(segB) */
four1(segB-1,segA_len,1);
/* angle of segB */
for (k = 0; k<=segA_len-1; k++) {
segB_ang[k] = atan2(segB[2*k+1],segB[2*k]);
}
/* segB=cos(ang)+i*sin(ang) */
l = 0;
for (k = 0; k<=2*segA_len-1; k+=2) {
segB[k] = cos(segB_ang[l]); // real part
segB[k+1] = sin(segB_ang[l]); // imaginary part
l++;
}
/* IFFT(segB) */
four1(segB-1,segA_len,-1);
/* Real part of segB */
for (k = 0; k<=segA_len-1; k++) {
segC[k] = segB[2*k]/segA_len;
}
```

```
getcyc(y, segC, segA_len, daat*16);
segA_norm = 0;
for (k = 0; k<=segA_len-1; k++) {
segA_norm = segA_norm + pow(segA[k], 2);
}
segA_norm = sqrt(segA_norm);
for (k = 0; k<=segA_len-1; k++) {
v[k] = segA[k]/segA_norm;
}
ip = 0;
for (k = 0; k<=segA_len-1; k++) {
ip = ip + y[k]*v[k];
}
for (k = 0; k<=segA_len-1; k++) {
segA[k] = segA[k] + y[k] * (7-2*ip) * pow(segA_len,0.5);
}
delete [] segB;
delete [] segB_ang;
delete [] segC;
delete [] y;
delete [] v;
delete [] h;
return;
}
void gaussian(float gauss[][1], float sigma2, int iseed, int nr, int nc)
/* This routine generates an approximately Gaussian random
sequence with zero mean and variance sigma2.
Inputs:
sigma2 - variance (float)
iseed - seed to random number generator (integer)
nr - number of rows of gauss array (integer)
nc - number of columns of gauss array (integer)
Output:
gauss - matrix of randomly generated data of size nrXnc (float)
/
{
int i,j,k;
float x,sum;

for (i = 0; i < nr; i++)
{
```

```
for (j = 0; j < nc; j++)
{
for (sum = 0,k = 0; k < 12; k++)
{
iseed = 2045 * (iseed) + 1;
iseed = iseed - (iseed/1048576) * 1048576;
x = float((iseed + 1) / 1048577.0) ;
sum = sum + float(sqrt(sigma2) * (x - 0.5));
} /* gaussian sequence with m= 0, sigma2 */
gauss[i][j] = sum;
}
}

}
void getcyc(double y[], double x[], int size, int b)

/* Input: x - vector of data that will be hidden of length: size (double)
size - length of data vector (integer)
b - shift parameter (integer)
Output: y - vector of data that will be hidden after preprocessing
of length: size (double)
/
{
int k;
void cirshift(double [], double [], int, int);
if (b <= size-1) {
cirshift(y, x, size, b);
}
else {
for(k=0;k<=size-1;k++) {
x[k] = -2*x[k];
}
cirshift(y, x, size, b - (size-1));
}
return;
}
void four1(double data[], unsigned long nn, int isign)
/* FFT and IFFT routine
Inputs:
data - vector of data of length 2*nn (double)
Note: Even elements are real parts of data
```

```
Odd elements are imaginary parts of data
nn - number of complex data numbers (has to be a power of 2) (unsigned long)
isign - indicator: isign = 1 forward FFT
isign = -1 IFFT
Output:
data - vector of data after FFT or IFFT of length 2*nn (double)
Note: Even elements are real parts of data
Odd elements are imaginary parts of data
/
{
unsigned long n,mmax,m,j,istep,i;
double wtemp,wr,wpr,wpi,wi,theta;
double tempr,tempi;
n=nn << 1;
j=1;
for (i=1;i<n;i+=2) {
if (j > i) {
SWAP(data[j],data[i]);
SWAP(data[j+1],data[i+1]);
}
m=n >> 1;
while (m >= 2 && j > m) {
j -= m;
m >>= 1;
}
j += m;
}
mmax=2;
while (n > mmax) {
istep=mmax << 1;
theta=isign*(6.28318530717959/mmax);
wtemp=sin(0.5*theta);
wpr = -2.0*wtemp*wtemp;
wpi=sin(theta);
wr=1.0;
wi=0.0;
for (m=1;m<mmax;m+=2) {
for (i=m;i<=n;i+=istep) {
j=i+mmax;
tempr=wr*data[j]-wi*data[j+1];
tempi=wr*data[j+1]+wi*data[j];
```

```
data[j]=data[i]-tempr;
data[j+1]=data[i+1]-tempi;
data[i] += tempr;
data[i+1] += tempi;
}
wr=(wtemp=wr)*wpr-wi*wpi+wr;
wi=wi*wpr+wtemp*wpi+wi;
}
mmax=istep;
}
}
#undef SWAP
void out_scr(char* msg)
/* End of processing screen

msg - message
/
{
puts(
"<HTML><HEAD><TITLE>Data Hiding Engine - Data Successfully Hidden</TITLE></HEAD>"
"<BODY BGCOLOR=008040 TEXT=FFFFFF LINK=00FF00 VLINK=FF0080>"
"<TABLE>"
"<TR><TD><h1>Data Hiding Engine - Data Successfully Hidden</h1></TD></tr>"
"<TR><TD><small> <I>Arkadiusz Edward Komenda</I> &#183;"
"email: <a href=mailto:aek4692@megahertz.njit.edu>aek4692@megahertz.njit.edu</a> &#183;"
"web: <a href=http://megahertz.njit.edu/~aek4692>http://megahertz.njit.edu/~aek4692</a><br>"
"</small>"
"</TD></TR></TABLE>"
"<hr>"
);
printf("<font color=E6EB1F><center><h2><b>%s%s</b></h2><p></font>", "Name of the file with hidden
data is: ", msg );
puts(
"<P><H4><CENTER>"
"<A HREF=http://njcmr.org/dhs/image_h.html>Hide data in the next image</A>"
"</CENTER></H4>"
);
puts(
"<P><H4><CENTER>"
"<A HREF=http://njcmr.org/dhs/image_d.html>Detect data from a new image</A>"
"</CENTER></H4>"
```

```
);
puts(
"<P><H4><CENTER>"
"<A HREF=http://njcmr.org/dhs/>Back to the main page</A>"
"</CENTER></H4>"
);
puts(
"<P><H4><CENTER>"
"<A HREF=http://njcmr.org/>NJCMR Home Page</A>"
"</CENTER></H4>"
);
puts("</BODY></HTML>");
return;
}
/* End of Function Definitions (main program) **************/

/* END OF SUBROUTINES ************************************/
```

img_d.c.cgi - data extracting subprogram: /* Filename: img_d.c.cgi

Written by: Arkadiusz Edward Komenda

```
/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <malloc.h>
#define ISIZE 1000 /* max image size: m_horiz x ISIZE */
#define _max(a,b) (((a) > (b)) ? (a) : (b))
#define _min(a,b) (((a) < (b)) ? (a) : (b))
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
/* this is the structure we use for the CGI variables */
struct {
char name[20];
char val[7000];
} elements[5];
/* forward CGI data processing declarations */
void splitword(char *out, char *in, char stop);
char x2c(char *x);
void unescape_url(char *url);
/* Function Prototypes (main program) */
void err_input(char* );
```

```
int ch_blank(char* );
int chk_pow2(int);
int round(float);
int bound(int, int, int);
void ddft(float [][ISIZE], int, int, double [][34], double [][34]);
void tnsps34(float [][34], float [][34]);
void vec_mult34(double [][34], double [][34], float [][34], int);
void chk_zero(double [][34], int);
void fsub(double [][34], double [][34], double [][34], int);
void sortArray(int [], int, int []);
void clsb(double [], int);
int detsig(double [], int, int);
void out_scr(unsigned char*);
/* BEGINNING OF PROGRAM ************************************/
main(int argc, char ** argv)
{
/* for CGI data processing */
char * ct; /* for content-type */
char * cl; /* for content-length */
int icl; /* content-length */
char qs[1000]; /* query string */
int rc;
int i;
/* program data declatations (input variables from webpage) */
int c_mode, m_horiz, m_vert;
char * m_fin;
char chky[22], foutpath[50];
/* program data declatations */
int k,l;
unsigned char num;
float red, gre, blu;
char line1[102], line2[350];
char *token;
FILE *f1Ptr; /* f1Ptr = input file pointer */
int band_num;
double nb;
int n_size, nbI;
unsigned int sd;
int len, seg_len, no_seg;

/* send the mime-type header */
```

```
printf("Content-type: text/html\n\n");

/* CGI BIN DATA PROCESSING ************************************/
/* grab the content-type and content-length
and check them for validity */
ct = getenv("CONTENT_TYPE");
cl = getenv("CONTENT_LENGTH");
if(cl == NULL)
{
err_input("content-length is undefined!");
exit(1);
}
icl = atoi(cl);
if(strcmp(ct, "application/x-www-form-urlencoded"))
{
err_input("I don't understand the content-type");
exit(1);
}
else if (icl == 0)
{
err_input("content-length is zero");
exit(1);
}
if((rc = fread(qs, icl, 1, stdin)) != 1)
{
err_input("cannot read the input stream! Contact the webmaster");
exit(1);
}
qs[icl] = '\0';
/* split out each of the parameters from the
query stream */
for(i = 0; qs[0] != '\0'; i++)
{
/* first divide by '&' for each parameter */
splitword(elements[i].val, qs, '&');
/* convert the string for hex characters and pluses */
unescape_url(elements[i].val);
/* now split out the name and value */
splitword(elements[i].name, elements[i].val, '=');
}
/* END OF CGI BIN DATA PROCESSING ***************************/
```

```
/* REASSIGNMENT OF INPUT DATA ********************************/
c_mode = atoi(elements[0].val);
// N = (int) pow(2,atoi(elements[1].val));
m_fin = elements[1].val;
m_horiz = atoi(elements[2].val);
m_vert = atoi(elements[3].val);
sd = 12345; /* seed for scattering */
seg_len = 2048; /* length of a segment of bits hidden at one time */
/* END OF INPUT DATA REASSIGNMENT ****************************/
/* DATA VALIDATION ***************************************/

/* Input and validation of variables from the dialog box */

if (ch_blank(m_fin)==1) {
err_input("Error - Enter input file name");
exit(1);
}

if (m_horiz<=0) {
err_input("Error - Enter width");
exit(1);
}
if (m_vert<=0) {
err_input("Error - Enter height");
exit(1);
}
/* END OF DATA VALIDATION *********************************/
/* MEMORY ALLOCATION **************************************/

/* Dynamic memory allocation on the heap */
float (*yell)[ISIZE] = new float[m_horiz][ISIZE];
if (yell == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
float (*cb)[ISIZE] = new float[m_horiz][ISIZE];
if (cb == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
float (*cr)[ISIZE] = new float[m_horiz][ISIZE];
```

```
if (cr == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
/* unsigned char (*imig)[ISIZE] = new unsigned char[m_horiz][ISIZE];
if (imig == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
/
float (*t34)[34] = new float[34][34];
if (t34 == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
float (*t34T)[34] = new float[34][34];
if (t34T == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
/* number of bands */
band_num = int(floor(m_horiz/float(8)))*(int(floor(m_vert/float(8))));

double (*mag_dft)[34] = new double[band_num][34];
if (mag_dft == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*ang_dft)[34] = new double[band_num][34];
if (ang_dft == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*mm)[34] = new double[band_num][34];
if (mm == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
/* number of useful bands of subband coefficients */
n_size = band_num;
nb = sqrt(n_size);
```

```
while ((nb!=floor(nb))——(chk_pow2(int(nb))!=0)) {
n_size--;
nb = sqrt(n_size);
}
nbI = int(nb);
double (*mus)[34] = new double[16][34];
if (mus == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*ms)[34] = new double[nbI*nbI-16][34];
if (ms == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*vec) = new double[30*(nbI*nbI-16)];
if (vec == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
int (*oseq) = new int[30*(nbI*nbI-16)];
if (oseq == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
int (*order) = new int[30*(nbI*nbI-16)];
if (order == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*vecro) = new double[30*(nbI*nbI-16)];
if (vecro == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*vecS) = new double[30*(nbI*nbI-16)];
if (vecS == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
/* number of blocks used to hide data, no_seg */
```

```
len = 30*(nbI*nbI-16);
no_seg = int(floor(len/seg_len));
/* int (*bitseq) = new int[no_seg];
if (bitseq == NULL) {
err_input("Error - Not enough memory");
exit(1);
}*/
double (*seg) = new double[seg_len];
if (seg == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
unsigned char (*dataout) = new unsigned char[no_seg+1];
if (dataout == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
/* END OF MEMORY ALLOCATION ******************************/

/* READING IMAGE INPUT FILE ******************************/
/* if c_mode = 3 check for .Y or .y */
strcpy( chky, "");
strcat( chky, m_fin);
token = strtok(chky,".");
token = strtok(NULL,".");
if ((c_mode==3)&&((strcmp(token,"y\0")!=0)&&(strcmp(token,"Y\0")!=0))) {
err_input("Error - Input file of wrong format");
exit(1);
}
/* path where the input file is */
strcpy( foutpath, "./output/"); /* path */
strcat( foutpath, m_fin);
if ((f1Ptr = fopen(foutpath, "rb")) == NULL) {
err_input("Error - Input file could not be found");
exit(1);
}
else {
if (c_mode==3) {
/* .Y or .y */
/* getting data */
for (l = 0; l <=m_vert-1; l++) {
```

```
for (k = 0; k <=m_horiz-1; k++) {
num = fgetc(f1Ptr);
// imig[k][l] = num;
yell[k][l] = float(num);
}
}
}
else {
if (c_mode==2) {
/* get line with P6 */
fgets(line1, 101, f1Ptr);
if (line1[1]!='6') {
err_input("Error - Input file of wrong format");
fclose(f1Ptr);
exit(1);
}
/* get next line */
fgets(line1, 101, f1Ptr);

/* get remaining lines starting with # and one line after them */
while (line1[0] == '#') {
fgets(line1, 101, f1Ptr);
}
/* getting rid of width, height and no. of colors */
token = strtok (line1," ");
token = strtok (NULL," ");
token = strtok (NULL," ");
/* supporting 2 styles of P6 files */
if (token == 0) {
fgets(line1, 101, f1Ptr);
}
/* getting data */
for (l = 0; l <=m_vert-1; l++) {
for (k = 0; k <=m_horiz-1; k++) {
num = fgetc(f1Ptr);
red = float(num);
num = fgetc(f1Ptr);
gre = float(num);
num = fgetc(f1Ptr);
blu = float(num);
/* Convert to other color domain */
```

```
yell[k][l] = float(0.3*red+0.6*gre+0.1*blu);
cr[k][l] = float(0.45*blu-0.15*red-0.3*gre+0.5);
cb[k][l] = float(0.4375*red-0.375*gre-0.0625*blu+0.5);
}
}
}
else {
/* get line with P5 */
fgets(line1, 101, f1Ptr);
if (line1[1]!='5') {
err_input("Error - Input file of wrong format");
fclose(f1Ptr);
exit(1);
}
/* get next line */
fgets(line1, 101, f1Ptr);

/* get remaining lines starting with # and one line after them */
while (line1[0] == '#') {
fgets(line1, 101, f1Ptr);
}
/* getting rid of width, height and no. of colors */
token = strtok (line1," ");
token = strtok (NULL," ");
token = strtok (NULL," ");
/* supporting 2 styles of P5 files */
if (token == 0) {
fgets(line1, 101, f1Ptr);
}
/* getting data */
for (l = 0; l <=m_vert-1; l++) {
for (k = 0; k <=m_horiz-1; k++) {
num = fgetc(f1Ptr);
// imig[k][l] = num;
yell[k][l] = float(num);
}
}
}
}
}
fclose(f1Ptr);
```

```
/* END OF READING IMAGE INPUT FILE ************************/

/* LOAD TABLE T34 ***************************************/
if ((f1Ptr = fopen("t34mat.txt", "r")) == NULL) {
err_input("Error - Input file for table T34 could not be found");
exit(1);
}
else {
for (k=0;k<=33;k++) {
/* get line */
fgets(line2, 349, f1Ptr);
token = strtok(line2,",");
t34[k][0] = float(atof(token));
for (l=1;l<=33;l++) {
token = strtok(NULL,",");
t34[k][l] = float(atof(token));
}
}
}
fclose(f1Ptr);
/* END OF LOAD TABLE T34 *********************************/
ddft(yell, m_horiz, m_vert, mag_dft, ang_dft);
tnsps34(t34T, t34);
vec_mult34(mm, mag_dft, t34T, band_num);
fsub(mus, ms, mm, nbI);
i = 0;
for(l=0;l<=nbI*nbI-16-1;l++) {
for(k=4;k<=33;k++) {
vec[i] = ms[l][k];
i++;
}
}
/* RANDOM SEQUENCE GENERATION AND SCATTERING **************/
srand(sd);
for (k = 0; k<=30*(nbI*nbI-16)-1; k++) {
oseq[k] = rand();
order[k] = k; /* initializing order array */
}
/* Sorting of random sequence in ascending order */
sortArray(oseq, 30*(nbI*nbI-16), order);
```

```
/* Scattering of coefficients */
for (k = 0; k<=30*(nbI*nbI-16)-1; k++) {
vecro[k] = vec[order[k]];
}
/* END OF RANDOM SEQUENCE GENERATION AND SCATTERING *******/
/* Quantizer */
clsb(vecro, len);
/* DETECTING DATA ****************************************/
for(i=0;i<=no_seg-1;i++) {
l = 0;
for (k = (i*seg_len); k<=(seg_len*(i+1))-1; k++) {
seg[l] = vecro[k];
l++;
}
dataout[i] = detsig(seg, seg_len, sd);
}
dataout[i] = '\0';
/* END OF DETECTING DATA *********************************/
/* MEMORY DEALLOCATION ***********************************/
/* Dynamic memory deallocation */
delete [] yell;
delete [] cb;
delete [] cr;
// delete [] imig;
delete [] t34;
delete [] t34T;
delete [] mag_dft;
delete [] ang_dft;
delete [] mm;
delete [] mus;
delete [] ms;
delete [] vec;
delete [] oseq;
delete [] order;
delete [] vecro;
delete [] vecS;
// delete [] bitseq;
delete [] seg;
/* END OF MEMORY DEALLOCATION ****************************/
out_scr(dataout); /* final screen */
delete [] dataout;
```

```
return 0;
}
/* END OF PROGRAM ***************************************/
/* SUBROUTINES ******************************************/

/* CGI function definitions *****************************/
void splitword(char *out, char *in, char stop)
{
int i, j;
for(i = 0; in[i] && (in[i] != stop); i++)
out[i] = in[i];
out[i] = '\0'; /* terminate it */
if(in[i])
++i;
for(j = 0; in[j]; ) /* shift the rest of the in */
in[j++] = in[i++];
}
char x2c(char *x)
{
register char c;
/* note: (x & 0xdf) makes x upper case */
c = (x[0] >= 'A' ? ((x[0] & 0xdf) - 'A') + 10 : (x[0] - '0'));
c *= 16;
c += (x[1] >= 'A' ? ((x[1] & 0xdf) - 'A') + 10 : (x[1] - '0'));
return(c);
}

/* this function goes through the URL char-by-char
and converts all the "escaped" (hex-encoded)
sequences to characters
this version also converts pluses to spaces. I've
seen this done in a separate step, but it seems
to me more efficient to do it this way. -wew
/
void unescape_url(char *url)
{
register int i, j;
for(i = 0, j = 0; url[j]; ++i, ++j)
{
if((url[i] = url[j]) == '%')
{
```

```
url[i] = x2c(&url[j + 1]);
j += 2;
}
else if (url[i] == '+')
url[i] = ' ';
}
url[i] = '\0'; /* terminate it at the new length */
}
/* End of CGI function definitions ************************/
/* Input Error Screen function definition ******************/
void err_input(char* msg)
/* Multipurpose error screen for displaying error mesages
msg - error message
/
{
puts(
"<HTML><HEAD><TITLE>Data Hiding Engine - Error</TITLE></HEAD>"
"<BODY BGCOLOR=008040 TEXT=FFFFFF LINK=00FF00 VLINK=FF0080>"
"<TABLE>"
"<TR><TD><h1>Data Hiding Engine - Error</h1></TD></tr>"
"<TR><TD><small> <I>Arkadiusz Edward Komenda</I> &#183;"
"email: <a href=mailto:aek4692@megahertz.njit.edu>aek4692@megahertz.njit.edu</a> &#183;"
"web: <a href=http://megahertz.njit.edu/~aek4692>http://megahertz.njit.edu/~aek4692</a><br>"
"</small>"
"</TD></TR></TABLE>"
"<hr>"
);
printf("<font color=E6EB1F><center><h2><b>%s</b></h2><p></font>", msg);
puts("<h3>You can use Back button in your browser to return to your form"
"</h3></center>");
puts("</BODY></HTML>");
return;
}
/* End of Input Error Screen function definition ***********/

/* Function Definitions (main program) *********************/
int ch_blank(char* data)
/* Check if a string consists of spaces only
data - string to be checked
Return: 0 = not white spaces only
```

```
1 = white spaces only
/
{
if (strlen(data)==strspn(data," "))
return 1;
else
return 0;
}
int round(float num)
/* Rounding floating numbers into integers
num - floating number to be rounded
Return: integer after rounding operation
/
{
int whole, out;
float test;
whole = int(floor(num));
test = float(whole) + float(0.5);
if (num >= test) {
out = whole + 1;
}
else {
out = whole;
}
return out;
}
int bound(int a, int lowerbound, int upperbound)
/* Bounding the integer between the lower and upper limits
a - integer to be bounded
lowerbound - lower limit of the range
upperbound - upper limit of the range
Return: an integer that is bigger or equal to the lower limit and
at the same time lower or equal to the upper limit of the range
/
{
int b;

b=__min(a,upperbound);
b=__max(a,lowerbound);
return b;
}
```

```
void ddft(float arr[][ISIZE], int horiz, int vert, double mag[][34], double ang[][34])
/* This function obtains the magnitude and angle of the DFT
coefficients of arr of size horizXvert. Block size 8x8.
Number of bands of magnitude DFT is 8^2/2 + 2
Inputs:
arr - floating data array
horiz - width of arr array (integer)
vert - height of arr array (integer)
Outputs:
mag - array containing DFT magnitudes of arr elements
and - array containing DFT angles of arr elements
/
{
int row, col, k, l, kk, ll, k1, l1, cntR, cntC;
double block_r[8][8], block_i[8][8];
void fft2(double [][8], double [][8], int, int, int, int, int);
cntR = 0;
for (kk = 0; kk<=int(floor(horiz/float(8))-1); kk++) {
for (ll = 0; ll<=int(floor(vert/float(8))-1); ll++) {
row = 0;
for (k = (kk*8); k<=(8*(kk+1))-1; k++) {
col = 0;
for (l = (ll*8); l<=(8*(ll+1))-1; l++) {
block_r[row][col] = arr[k][l];
block_i[row][col] = 0;
col++;
}
row++;
}
fft2(block_r,block_i,8,8,3,3,0); /* forward 2D Fourier transform */
cntC = 0;
for (l1=0;l1<=4;l1++) {
for (k1=0;k1<=4;k1++) {
mag[cntR][cntC] = sqrt(pow(block_r[k1][l1],2.0)+pow(block_i[k1][l1],2.0))/8;
ang[cntR][cntC] = atan2(block_i[k1][l1],block_r[k1][l1]);
cntC++;
}
}
for (l1=1;l1<=3;l1++) {
for (k1=5;k1<=7;k1++) {
mag[cntR][cntC] = sqrt(pow(block_r[k1][l1],2.0)+pow(block_i[k1][l1],2.0))/8;
```

```
ang[cntR][cntC] = atan2(block_i[k1][l1],block_r[k1][l1]);
cntC++;
}
}
cntR++;
}
}
return;
}
void fft2(double x2[][8], double y2[][8], int n1, int n2, int m1, int m2, int idir)
/* x2 - real part (2D array of size n1Xn2)
y2 - imaginary part (2D array of size n1Xn2)
m1 = log2(n1)
m2 = log2(n2)
idir - direction of the transformation (0 - forward , 1 - reverse)
/
{
double *x,*y;
int i,j;


void srfft(double [], double [], int, int);
void isrfft(double [], double [], int, int);

x = (double *) calloc(n2,sizeof(double)); /* allocate buffers */
y = (double *) calloc(n2,sizeof(double));
/* ——computing the DFT of the rows of the picture matrix—— */
for (i = 0; i < n1 ; i++)
{

for(j = 0 ; j < n2 ; j++)
{
x[j] = x2[i][j];
y[j] = y2[i][j];
}

if (idir == 0 ) srfft(x,y,n2,m2);
if (idir == 1 ) isrfft(x,y,n2,m2);
/* the intermediate matrix */

for(j = 0 ; j < n2 ; j++)
```

```
{
x2[i][j] = x[j];
y2[i][j] = y[j];
}
}

free(x);free(y);
x = (double *) calloc(n1,sizeof(double)); /* allocate buffers */
y = (double *) calloc(n1,sizeof(double));
/* —computing the DFT of the columns of the intermediate matrix— */

for (j = 0; j < n2 ; j++)
{
for(i = 0 ; i < n1 ; i++)
{
x[i] = x2[i][j];
y[i] = y2[i][j];
}

if (idir == 0) srfft(x,y,n1,m1);
if (idir == 1) isrfft(x,y,n1,m1);

for(i = 0 ; i < n1 ; i++)
{
x2[i][j] = x[i];
y2[i][j] = y[i];
}
}
// free(*x);free(*y);
free(x); free(y);
}
void srfft(double x[], double y[], int n, int m)
{

int i,j,k,n2,n4,is,id,i0,i1,i2,i3;
double a,a3,e,r1,r2,s1,s2,s3,cc1,ss1,cc3,ss3,xt;


n2 = 2 * n;
for (k = 1;k <= m-1; k++)
{
```

```
n2 = n2/2;
n4 = n2/4;
e = 6.283185307179586/n2;

a = 0.0;
for (j = 1; j <= n4 ; j++)
{
a3 = 3.0 * a;

cc1 = cos(a);
ss1 = sin(a);
cc3 = cos(a3);
ss3 = sin(a3);

a = j * e;
is = j;
id = 2 * n2;
do
{
for (i0 = is-1 ; i0 <= n-1 ; i0 += id)
{
i1 = i0 + n4;
i2 = i1 + n4;
i3 = i2 + n4;

r1 = x[i0] - x[i2] ;
x[i0] += x[i2];
r2 = x[i1] - x[i3] ;
x[i1] += x[i3];

s1 = y[i0] - y[i2] ;
y[i0] += y[i2];
s2 = y[i1] - y[i3] ;
y[i1] += y[i3];

s3 = r1 - s2;
r1 += s2;
s2 = r2 - s1;
r2 += s1;

x[i2] = r1 * cc1 - s2 * ss1;
```

```
y[i2] = -s2 * cc1 - r1 * ss1;
x[i3] = s3 * cc3 + r2 * ss3;
y[i3] = r2 * cc3 - s3 * ss3;
}
is = 2 * id - n2 + j;
id *= 4;
} while (is < n);
}
}
/* */
/* ————————-last stage, length-2 butterfly————————— */
/* */
is = 1;
id = 4;
do
{
for (i0 = is ; i0 <= n ; i0 += id)
{
i0–;
i1 = i0 + 1;

r1 = x[i0];

x[i0] = r1 + x[i1];
x[i1] = r1 - x[i1];

r1 = y[i0];
y[i0] = r1 + y[i1];
y[i1] = r1 - y[i1];
i0++;
}
is = 2 * id - 1;
id *= 4;
} while (is < n);
/* */
/* ————————————bit reverse counter————————————— */
/* */
j = 1;
for (i = 1; i <= n-1 ; i ++)
{
if (i < j)
```

```
{
j--; i--;
xt = x[j]; x[j] = x[i]; x[i] = xt;
xt = y[j]; y[j] = y[i]; y[i] = xt;
j++; i++;
}
k = n/2;
while (k < j)
{
j -= k;
k = k/2;
}
j += k;
}
}
void isrfft(double x[], double y[], int n, int m)
{
int i;
double n1;
void srfft(double [], double [], int, int);
n1 = 1. / ((double) n); /* scale factor */

srfft(y,x,n,m); /* perform a foward DFT */
for (i = 0; i < n ; i++)
{
x[i] = x[i] * n1;
y[i] = y[i] * n1;
}
}
void tnsps34(float arrT[][34], float arr[][34])
/* Transpose of a square matrix of size: 34X34
Input: arr - matrix to be transposed (float)
Output: arrT - transposed matrix arr (float)
/
{
int l, k;
for (l = 0; l<=33; l++) {
for (k = 0; k<=33; k++) {
arrT[k][l] = arr[l][k];
}
}
```

```
return;
}
void vec_mult34(double prod[][34], double arr1[][34], float arr2[][34], int size)
/* Vector product of two matrices
Inputs: arr1 - leftmost array of size: sizeX34 (double)
arr2 - rightmost array of size: 34x34 (float)
size - number of rows in arr1 (integer)
Output: prod - vector product of arr1 and arr2 of size: sizeX34 (double)
/
{
int l1, k1, k2;
double sum;
for (l1 = 0; l1<=size-1; l1++) {
for (k2 = 0; k2<=33; k2++) {
sum = 0;
for (k1 = 0; k1<=33; k1++) {
sum = sum + (arr1[l1][k1] * arr2[k1][k2]);
}
prod[l1][k2] = sum;
}
}
return;
}
void chk_zero( double arr[][34], int size)
/* If the element of the array is less than zero then make it zero
Inputs: arr - array to be checked of size: sizeX34 (double)
size - number of rows in arr (integer)
Output: arr - updated array
/
{
int k, l;
for (k=0; k<=size-1; k++) {
for (l=0; l<=33; l++) {
if (arr[k][l]<0) {
arr[k][l] = 0;
}
}
}
return;
}
void fsub(double vecfus[][34], double vecfs[][34], double picf[][34],
```

```
int nbI)
/* This function obtains the subband coefficients of the
block transform coefficients.
vecfus is not used for data hiding, vecfs is.
Inputs:
picf - array of block transform coefficients (double)
nbI - helps to determine the size of vecfs (integer)
Outputs:
vecfus - array not used for data hiding, of size: 16x34 (double)
vecfs - array used later for data hiding, of size: (nbI*nbI-16)x34 (double)
/
{
int k, l, row, col;
void err_input(char* );
void rsdec(double [][100], int);
double (*segf)[100] = new double[nbI][100];
if (segf == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
for(l=0;l<=33;l++) {
k = 0;
for(col=0;col<=nbI-1;col++) {
for(row=0;row<=nbI-1;row++) {
segf[row][col] = picf[k][l];
k++;
}
}
rsdec(segf, nbI);
k = 0;
for(col=4;col<=nbI-1;col++) {
for(row=0;row<=3;row++) {
vecfs[k][l] = segf[row][col];
k++;
}
}
for(col=4;col<=nbI-1;col++) {
for(row=4;row<=nbI-1;row++) {
vecfs[k][l] = segf[row][col];
k++;
}
```

```
}
for(col=0;col<=3;col++) {
for(row=4;row<=nbI-1;row++) {
vecfs[k][l] = segf[row][col];
k++;
}
}
k = 0;
for(col=0;col<=3;col++) {
for(row=0;row<=3;row++) {
vecfus[k][l] = segf[row][col];
k++;
}
}
}
delete [] segf;
return;
}
void rsdec(double I[][100], int size)
/* Inputs: I - image of size: sizeXsize (double)
size - number of rows and columns in I (integer)
Output: I - modified image of size: sizeXsize (double)
/
{
int m, k, l, kk, ll, row, col;
int vhs, nd;
double y[64][64];
void sdecrect(double [][64], int, int);
for(m=1;m<=3;m++) {
nd = int(pow(2,m-1));
vhs=size/nd;
for(ll=0;ll<=nd-1;ll++) {
for(kk=0;kk<=nd-1;kk++) {
col = 0;
for (l = (ll*vhs); l<=(vhs*(ll+1))-1; l++) {
row = 0;
for (k = (kk*vhs); k<=(vhs*(kk+1))-1; k++) {
y[row][col] = I[k][l];
row++;
}
col++;
```

```
}
sdecrect(y, col, 0);
col = 0;
for (l = (ll*vhs); l<=(vhs*(ll+1))-1; l++) {
row = 0;
for (k = (kk*vhs); k<=(vhs*(kk+1))-1; k++) {
I[k][l] = y[row][col];
row++;
}
col++;
}
}
}
}
return;
}
void sdecrect(double y[][64], int size, int mode)
/* Inputs: y - image array of size: sizeXsize (double)
size - number of columns and rows in y (integer)
mode - switch
mode = 0 forward
mode = 1 inverse of function
Output: y - resulting array (double)
/
{
int k, l;
void err_input(char* );
void cirshift(double [], double [], int, int);
void vec_mult64(double [][64], double [][64], double [][64], int);

void tnsps64(double [][64], double [][64], int);
double (*h) = new double[size];
if (h == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*hmod) = new double[size];
if (hmod == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
```

```
double (*g) = new double[size];
if (g == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*gmod) = new double[size];
if (gmod == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*T)[64] = new double[size][64];
if (T == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*TT)[64] = new double[size][64];
if (TT == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*res)[64] = new double[size][64];
if (res == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
/* Get filter coefficients */
h[0]=(1+sqrt(3))/4/sqrt(2);
h[1]=(3+sqrt(3))/4/sqrt(2);
h[2]=(3-sqrt(3))/4/sqrt(2);
h[3]=(1-sqrt(3))/4/sqrt(2);
for(k=4;k<=size-1;k++) {
h[k] = 0;
}
g[0]=h[3];
g[1]=-1*h[2];
g[2]=h[1];
g[3]=-1*h[0];
for(k=4;k<=size-1;k++) {
g[k] = 0;
}
for(k=0;k<=size/2-1;k++) {
```

```
cirshift(hmod, h, size, 2*k);
cirshift(gmod, g, size, 2*k);
for(l=0;l<=size-1;l++) {
T[k][l] = hmod[l];
T[size/2+k][l] = gmod[l];
}
}
if (mode==0) {
vec_mult64(res, T, y, size);
tnsps64(TT, T, size);
vec_mult64(y, res, TT, size);
}
else {
tnsps64(TT, T, size);
vec_mult64(res, TT, y, size);
vec_mult64(y, res, T, size);
}
delete [] h;
delete [] hmod;
delete [] g;
delete [] gmod;
delete [] T;
delete [] TT;
delete [] res;
return;
}
void tnsps64(double arrT[][64], double arr[][64], int size)
/* Transpose of a square matrix of size: sizeXsize
Input: arr - matrix to be transposed (double)
Output: arrT - transposed matrix arr (double)
/
{
int l, k;
for (l = 0; l<=size-1; l++) {
for (k = 0; k<=size-1; k++) {
arrT[k][l] = arr[l][k];
}
}
return;
}
void vec_mult64(double prod[][64], double arr1[][64], double arr2[][64], int size)
```

```
/* Vector product of two matrices
Inputs: arr1 - leftmost array of size: sizeXsize (double)
arr2 - rightmost array of size: sizeXsize (double)
size - number of rows and columns in arr1 and arr2 (integer)
Output: prod - vector product of arr1 and arr2 of size: sizeXsize (double)
/
{
int l1, k1, k2;
double sum;
for (l1 = 0; l1<=size-1; l1+-) {
for (k2 = 0; k2<=size-1; k2++) {
sum = 0;
for (k1 = 0; k1<=size-1; k1++) {
sum = sum + (arr1[l1][k1] * arr2[k1][k2]);
}
prod[l1][k2] = sum;
}
}
return;
}
void cirshift(double n_vect[], double vect[], int size, int shft)

/* Shift left to right elements of a vector in a circular manner
Inputs:
vect - vector to be shifted of length: size (double)
size - length of a vector (integer)
shft - number of spaces the vector should be shifted by (integer)
Output:
n_vect - new vector of length: size, after the shift operation (double)
/
{
int k, l;
double dummy;
for(l=0;l<=size-1;l++) {
n_vect[l] = vect[l];
}
if (shft!=0) {
for(l=0;l<=shft-1;l++) {
dummy = n_vect[size-1];
for(k=size-1;k>=1;k-) {
n_vect[k] = n_vect[k-1];
```

```
}
n_vect[0] = dummy;
}
}

return;
}
int chk_pow2(int num)
/* check whether an integer is a power of 2 or not */
/* Return values: */
/* flag = 0 : integer is a power of 2 */
/* flag = 1 : integer is NOT a power of 2 */
{
int flag = 0;
while ((num>1)&&(flag==0)) {
if ((num % 2)==0) {
flag = 0;
}
else {
flag = 1;
}
num = num/2;
}
return flag;
}
void sortArray(int orig[], int size, int ind[])
/* Sort array in ascending order and memorize old positions of elements
prior to sorting
Inputs: orig - data vector of length: size (integer)
ind - vector of length: size holding the index of data
vector, orig (integer)
size - length of vectors orig and ind (integer)
Outputs: orig - sorted vector
ind - vector holding initial orig index before sorting
/
{
int k, l;
int temp, itemp;
for (k = size-2; k>=0; k--) {
for (l = 0; l<=k; l++) {
if (orig[l]>orig[l+1]) {
```

```
temp = orig[l];
orig[l] = orig[l+1];
orig[l+1] = temp;
itemp = ind[l];
ind[l] = ind[l+1];
ind[l+1] = itemp;
}
}
}

return;
}
int detsig(double segA[], int segA_len, int sdd)
/* Detecting data
Inputs:
segA - data vector of length: segA_len, inside which data is hidden (double)
segA_len - length of the data vector (integer)
sdd - seed for gaussian random number generator (integer)
Output:
daat - detected data (integer)
/
{
int k, l;
double slice_abs, slice_max;
int slice_ind;
void err_input(char* );
void gaussian(float [][1], float , int, int, int);
void four1(double [], unsigned long, int);
double (*segB) = new double[segA_len*2];
if (segB == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*segB_ang) = new double[segA_len];
if (segB_ang == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*segA1) = new double[segA_len*2];
if (segA1 == NULL) {
err_input("Error - Not enough memory");
```

```
exit(1);
}
float (*h)[1] = new float[segA_len][1];
if (h == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*segA2) = new double[segA_len*2];
if (segA2 == NULL) {
err_input("Error - Not enough memory");
exit(1);
}
double (*slice) = new double[2*segA_len/16];
if (slice == NULL) {
err_input("Error - Not enough memory");
exit(1);
}

/* generating signature */
gaussian(h, 1, sdd, segA_len, 1);
/* Preparing h for FFT */
l = 0;
for (k = 0; k<=2*segA_len-1; k+=2) {
if (h[l][1]<0) {
segB[k] = -1;
}
else {
segB[k] = 1;
}
segB[k+1] = 0;
l++;
}
/* FFT(segB) */
four1(segB-1,segA_len,1);
/* angle of segB */
for (k = 0; k<=segA_len-1; k++) {
segB_ang[k] = atan2(segB[2*k+1],segB[2*k]);
}
/* segB=cos(ang)+i*sin(ang) */
l = 0;
for (k = 0; k<=2*segA_len-1; k+=2) {
```

```
segB[k] = cos(segB_ang[l]); // real part
segB[k+1] = sin(segB_ang[l]); // imaginary part
l++;
}
/* Preparing segA for FFT */
l = 0;
for (k = 0; k<=2*segA_len-1; k+=2) {
segA1[k] = segA[l];
segA1[k+1] = 0;
l++;
}
/* FFT(segA1) */
four1(segA1-1,segA_len,1);
/* fft(segA1).*conj(segB) */
for (k = 0; k<=2*segA_len-1; k+=2) {
segA2[k]=segA1[k]*segB[k]-segA1[k+1]*(-1)*segB[k+1];
segA2[k+1]=segA1[k+1]*segB[k]+segA1[k]*(-1)*segB[k+1];
}
for (k = 0; k<=2*segA_len/16-1; k+=2) {
slice[k] = 0;
}

l = 0;
for (k = 0; k<=2*segA_len-1; k++) {
slice[l] = slice[l] + segA2[k];
l++;
if (l>2*segA_len/16-1) {
l = 0;
}
}
/* IFFT(slice) */
four1(slice-1,segA_len/16,-1);
slice_max = sqrt(pow(slice[0]/segA_len/16,2.0)+pow(slice[1]/segA_len/16,2.0));
slice_ind = 0;
/* ABS(slice) */
for (k = 2; k<=2*segA_len/16-1; k+=2) {
slice_abs = sqrt(pow(slice[k]/segA_len/16,2.0)+pow(slice[k+1]/segA_len/16,2.0));
if (slice_max < slice_abs) {
slice_max = slice_abs;
slice_ind = k/2;
}
```

```
}
delete [] segB;
delete [] segB_ang;
delete [] segA1;
delete [ h;
delete [] segA2;
delete [] slice;
if (slice_ind<=31) {
slice_ind = 126;
}
else {
if (slice_ind==60) {
slice_ind = 126;
}
else {
if (slice_ind==62) {
slice_ind = 126;
}
}
}
return slice_ind;
}
void gaussian(float gauss[][1], float sigma2, int iseed, int nr, int nc)
/* This routine generates an approximately Gaussian random
sequence with zero mean and variance sigma2.
Inputs:
sigma2 - variance (float)
iseed - seed to random number generator (integer)
nr - number of rows of gauss array (integer)
nc - number of columns of gauss array (integer)
Output:
gauss - matrix of randomly generated data of size nrXnc (float)
/
{
int i,j,k;
float x,sum;

for (i = 0; i < nr; i++)
{
for (j = 0; j < nc; j+−)
{
```

```
for (sum = 0,k = 0; k < 12; k++)
{

iseed = 2045 * (iseed) + 1;
iseed = iseed - (iseed/1048576) * 1048576;
x = float((iseed + 1) / 1048577.0) ;
sum = sum + float(sqrt(sigma2) * (x - 0.5));
} /* gaussian sequence with m= 0, sigma2 */
gauss[i][j] = sum;
}
}

}
void four1(double data[], unsigned long nn, int isign)
/* FFT and IFFT routine
Inputs:
data - vector of data of length 2*nn (double)
Note: Even elements are real parts of data
Odd elements are imaginary parts of data
nn - number of complex data numbers (has to be a power of 2) (unsigned long)
isign - indicator: isign = 1 forward FFT
isign = -1 IFFT
Output:
data - vector of data after FFT or IFFT of length 2*nn (double)
Note: Even elements are real parts of data
Odd elements are imaginary parts of data
/
{
unsigned long n,mmax,m,j,istep,i;
double wtemp,wr,wpr,wpi,wi,theta;
double tempr,tempi;
n=nn << 1;
j=1;
for (i=1;i<n;i+=2) {
if (j > i) {
SWAP(data[j],data[i]);
SWAP(data[j+1],data[i+1]);
}
m=n >> 1;
while (m >= 2 && j > m) {
j -= m;
```

```
m >>= 1;
}
j += m;
}
mmax=2;
while (n > mmax) {
istep=mmax << 1;
theta=isign*(6.28318530717959/mmax);
wtemp=sin(0.5*theta);
wpr = -2.0*wtemp*wtemp;
wpi=sin(theta);
wr=1.0;
wi=0.0;
for (m=1;m<mmax;m+=2) {
for (i=m;i<=n;i+=istep) {
j=i+mmax;
tempr=wr*data[j]-wi*data[j+1];
tempi=wr*data[j+1]+wi*data[j];
data[j]=data[i]-tempr;
data[j+1]=data[i+1]-tempi;
data[i] += tempr;
data[i+1] += tempi;
}
wr=(wtemp=wr)*wpr-wi*wpi+wr;
wi=wi*wpr+wtemp*wpi+wi;
}
mmax=istep;
}
}
#undef SWAP
void clsb(double arr[], int size)

/* Choose 6 least significant bits of arr
Inputs:
arr - vector of data of length: size (double)
size - length of vector arr (integer)
Output:
arr - vector of data of 6 least significant bits of length: size (double)
/
{
int k, znak, arr_int;
```

```
double arr_max;
int n;
void err_input(char* );
int round(float);
void DecToBinN(int, char *, int);
int BinToDecN(char *, int);
n = 6; /* 6 least significant bits */
arr_max = fabs(arr[0]);
for (k = 1; k<= size-1; k++) {
arr_max = _max(arr_max, fabs(arr[k]));
}
arr_max = ceil(log(arr_max)/log(2));
if (n > arr_max) {
n = int(arr_max);
}
char (*binary) = new char[int(arr_max)];
if (binary == NULL) {
err_input("Error - Not enough memory");
exit(1);
}

for (k = 0; k<=size-1; k++) {
if (arr[k]>0) {
znak = 1;
}
else {
znak = -1;
}
arr_int = round(float(fabs(arr[k])));
DecToBinN(arr_int, binary, int(arr_max));
arr[k] = double(BinToDecN(binary, n));
arr[k] = arr[k] * znak;
}
delete [] binary;
return;
}
void DecToBinN(int dec, char * binar, int N)
/* Converts a decimal number to N-bit binary representation
Inputs:
dec - decimal number (integer)
N - length of the binary representation (integer)
```

```
Output:
binar - N bit long binary number (character)
/
{
int k,l;
k = N;
while (dec!=0) {
k--;
if ((dec % 2)==0) {
binar[k]='0';
}
else {
binar[k]='1';
}
dec = dec/2;
}
if (k>0) {
for (l=k-1; l>=0; l--) {
binar[l] = '0';
}
}
return;
}
int BinToDecN(char * binar, int N)
/* Convert last N bits of a binary number to an integer
Inputs:
binar - binary number (character)
N - how many last digits of a binary number convert to an integer (integer)
Return an integer
/
{
int sum = 0;
int k, l, len;
len = strlen(binar)-1;
l = 0;
for(k = len-1; k>=len-N; k--) {
if (binar[k]=='1') {
sum = sum + int(pow(2,l));
}
l++;
}
```

```
return sum;
}
void out_scr(unsigned char* msg)
/* End of processing screen
msg - message
/
{

puts(
"<HTML><HEAD><TITLE>Data Hiding Engine - Data Successfully Detected</TITLE></HEAD>"
"<BODY BGCOLOR=008040 TEXT=FFFFFF LINK=00FF00 VLINK=FF0080>"
"<TABLE>"
"<TR><TD><h1>Data Hiding Engine - Data Successfully Detected</h1></TD></tr>"
"<TR><TD><small> <I>Arkadiusz Edward Komenda</I> &#183;"
"email: <a href=mailto:aek4692@megahertz.njit.edu>aek4692@megahertz.njit.edu</a> &#183;"
"web: <a href=http://megahertz.njit.edu/~aek4692>http://megahertz.njit.edu/~aek4692</a><br>"
"</small>"
"</TD></TR></TABLE>"
"<hr>"
);
printf("<font color=E6EB1F><center><h2><b>%s%s</b></h2><p></font>", "Detected data is: ", msg );
puts(
"<P><H4><CENTER>"
"<A HREF=http://njcmr.org/dhs/image_d.html>Detect data from the next image</A>"
"</CENTER></H4>"
);
puts(
"<P><H4><CENTER>"
"<A HREF=http://njcmr.org/dhs/image_h.html>Hide data in a new image</A>"
"</CENTER></H4>"
);
puts(
"<P><H4><CENTER>"
"<A HREF=http://njcmr.org/dhs/>Back to the main page</A>"
"</CENTER></H4>"
);
puts(
"<P><H4><CENTER>"
"<A HREF=http://njcmr.org/>NJCMR Home Page</A>"
"</CENTER></H4>"
);
```

```
puts("</BODY></HTML>");
return;
}
/* End of Function Definitions (main program) **************/

/* END OF SUBROUTINES ************************************/
```

REFERENCES

1. R.J. Anderson and F.A. Petitcolas, "On the Limits of Steganography," *IEEE Journal on Selected Areas of Communications*, vol. 16, no 4, pp. 474-481, May 1998.

2. I.J. Cox, J. Kilian, F.T. Leighton, and T.G. Shamoon, "Secure Spread Spectrum Watermarking for Multimedia," *IEEE Transactions on Image Processing*, vol. 6, no.12, pp. 1673-1687, 1997.

3. E. Koch, J. Bindfrey and J. Zhao, "Copyright Protection for Multimedia Data," *Proc. of the Int. Conf. on Digital Media and Electronic Publishing*, Leeds, UK, December 1994.

4. K. Mantusi and K. Tanaka, "Video Steganography: How to secretly embed a signature in a picture?," *Proceedings of the IMA Intellectual Property Project*, Interactive Multimedia Association, Annapolis, MD, 1994.

5. M.Ramkumar and A.N. Akansu, "A Robust Scheme for Oblivious Detection of Watermarks/Data Hiding in Still Images," *Proceedings of SPIE, Multimedia Systems and Applications*, Boston, MA, pp. 474-481, Nov. 1998.

6. M.Ramkumar A.N. Akansu and A.A Alatan, "An FFT Based Signaling Scheme for Multimedia Steganography," Submitted to Globecom-99.

7. M.Ramkumar A.N. Akansu and A.A Alatan, "A Robust Data Hiding Scheme for Digital Images Using DFT," Submitted to ICIP-99.

8. J. O'Rnanaidh, W. Dowling and F. Boland, "Watermarking digital images for copyright protection," *IEE Proceedings on Vision, Image and Signal Processing*, vol. 143, no 4, pp. 250-256, August 1996.

9. D. Kundur and D. Hatzinakos, "A Robust Digital Image Watermarking Method using Wavelet-Based Fusion," *Proceedings of ICIP'97*, Santa Barbara, CA, vol. I, pp. 544-547, October, 1997.

10. G. Voyatzis and I. Pitas, "Applications of Toral automorphisms in image watermarking," *Proceedings of ICIP'96*, vol. II, pp. 237-240, 1996.

11. G. Voyatzis and I. Pitas, "Embedding Robust Logo Watermarks in Digital Images," *Proceedings of DSP'97*, vol. 1, pp. 213-216, 1997.

12. R.G. van Schyndel, A.Z. Tirkel and C.F. Osborne, "A Digital Watermark," *IEEE International Conference on Image Processing*, vol. 2, pp. 86-90, 1994.

13. W. Bender, D. Gruhl and N. Morimoto, "Techniques for Data Hiding," *Proceedings of SPIE*, vol. 2420, pp. 40-50, Feb 1995.

14. I. Pitas, "A method for signature casting on digital images," *International Conference on Image Processing*, vol. 3, pp. 215-218, September 1996.

15. M.D. Swanson, B. Zhu and A.H. Tewfik, "Transparent Robust Image Watermarking," *IEEE International Conference on Image Processing*, pp. 211-214, 1996.

16. J.J.K.Ó Ruanaidh, W.J.Dowling and F.M. Boland, "Watermarking Digital Images for Copyright Protection," *IEE Proceedings on Signal and Image Processing*, vol. 143, no 4, pp. 250-256, Aug 1996.

17. M. Barni, F. Bartolini, V. Cappellini and A. Piva, "A DCT-domain system for robust image watermarking," *Signal Processing*, vol. 66, no 3, pp. 357-372, May 1998.

18. H. Inoue, A. Miyazaki and T. Katsura, "A Digital Watermark for Image Signals Using a Controlled Quantization Method of the Wavelet Coefficients," submitted for publication.

19. H. Inoue, A. Miyazaki, A. Yamamoto and T. Katsura, "A Digital Watermark Based on the Wavelet Transform and its Robustness on Image Compression," *Proceedings of IEEE International Conference on Image Processing*, Chicago, Illinois, vol. 3, pp. 391-395, Oct 1998.

20. M.D. Swanson, B. Zhu and A.H. Tewfik, "Multiresolution Scene-Based Video Watermarking Using Perceptual Models," *IEEE Journal on Special Areas in Communications*, vol. 16, no 4, pp. 540-550, May 1998.

21. J.J.K.O. Ruanaidh, W.J.Dowling and F.M. Borland, "Phase Watermarking of Digital Images," *Proceedings of the IEEE Intl. Conf. on Image Processing ICIP-96*, Lausanne, Switzerland, pp. 239-242, Sep.1996.

22. G. Voyatzis and I. Pitas, "Chaotic Mixing of Digital Images and Applications to Watermarking," *ECMAST 1996*, Belgium, vol. 2, pp. 687-695, May 1996.

23. G. Voyatzis and I. Pitas, "Applications of Toral Automorphisms in image Watermarking," *IEEE International Conference on Image Processing*, vol. 2, pp. 237-240, Sep 1996.

24. P. Davern and M. Scott, "Fractal Based Image Steganography," *Info Hiding 96*, pp. 279-294, 1996.