

Portland State University

PDXScholar

Dissertations and Theses

Dissertations and Theses

1-1-2010

Removing Textured Artifacts from Digital Photos Using Spatial Frequency Filtering

Ben Huang

Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Let us know how access to this document benefits you.

Recommended Citation

Huang, Ben, "Removing Textured Artifacts from Digital Photos Using Spatial Frequency Filtering" (2010). *Dissertations and Theses*. Paper 148.

<https://doi.org/10.15760/etd.148>

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

Removing Textured Artifacts from Digital Photos Using Spatial
Frequency Filtering

by

Ben Huang

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electric and Computer Engineering

Thesis Committee -
Fu Li, Chair
Xiaoyu Song
James E. Morris

Portland State University
©2010

Abstract

Virtually all image processing is now done with digital images. These images, captured with digital cameras, can be readily processed with various types of editing software to serve a multitude of personal and commercial purposes. But not all images are directly captured and even of those that are directly captured many are not of sufficiently high quality.

Digital images are also acquired by scanning old paper images. The result is often a digital image of poor quality. Textured artifacts on some old paper pictures were designed to help protect pictures from discoloration. However, after scanning, these textured artifacts exhibit annoying textured noise in the digital image, highly degrading the visual definition of images on electronic screens. This kind of image noise is academically called global periodic noise. It is in a spurious and repetitive pattern that exists consistently throughout the image. There does not appear to be any commercial graphic software with a tool box to directly resolve this global periodic noise. Even Photoshop, considered to be the most powerful and authoritative graphic software, does not have an effective function to reduce textured noise. This thesis addresses this problem by proposing the use of an alternative graphic filter to what is currently available.

To achieve the best image quality in photographic editing, spatial frequency domain filtering is utilized instead of spatial domain filtering. In frequency domain images, the consistent periodicity of the textured noise leads to well defined spikes

in the frequency transform of the noisy image. When the noise spikes are at a sufficient distance from the image spectrum, they can be removed by reducing their frequency amplitudes. The filtered spectrum may then yield a noise reduced image through inverse frequency transforming.

This thesis proposes a method to reduce periodic noise in the spatial frequency domain; summarizes the difference between DFT and DCT, FFT and fast DCT in image processing applications; uses fast DCT as the frequency transform to solve the problem in order to improve both computational load and filtered image quality; and develops software that can be implemented as a plug in for large graphic software to remove textured artifacts from digital images.

Table of Contents

| | |
|------------------------------------------------------------------------------|------------|
| ABSTRACT | I |
| TABLE OF CONTENTS | III |
| LIST OF FIGURES | IV |
| LIST OF TABLES | V |
| CHAPTER 1 | |
| INTRODUCTION | 1 |
| 1.1 RESEARCH BACKGROUND..... | 1 |
| 1.2 PROBLEM STATEMENT..... | 2 |
| CHAPTER 2 | |
| CURRENT SOLUTIONS | 6 |
| 2.1 SPATIAL DOMAIN FILTERING | 7 |
| 2.1.1 Reducing Textured Noise Using Gaussian Blur..... | 7 |
| 2.1.2 Reducing Textured Noise Using Median Filter | 8 |
| 2.2 FREQUENCY DOMAIN FILTERING | 10 |
| 2.2.1 Reducing Textured Noise by Low Pass Filter in Frequency Domain..... | 10 |
| 2.2.2 Reducing Textured Noise by Removing Frequency Coefficient Spikes..... | 12 |
| CHAPTER 3 | |
| PROPOSED SOLUTION - DCT SPATIAL FREQUENCY FILTERING | 13 |
| 3.1 PROPOSED SOLUTION STATEMENT | 14 |
| 3.2 REVIEW OF THE DEFINITION OF DCT..... | 14 |
| 3.3 PROPERTIES OF DCT BENEFICIAL TO THE PROPOSED SOLUTION | 15 |
| 3.3.1 Properties of DCT Beneficial for Improving Filtered Image Quality..... | 15 |
| 3.3.2 Properties Beneficial for Improving Filter Programming Speed | 18 |
| 3.4 REVIEW OF FAST DISCRETE COSINE TRANSFORM | 20 |
| 3.5 THE APPEARANCE OF TEXTURED NOISE IN FREQUENCY DOMAIN | 21 |
| 3.6 ALGORITHM..... | 24 |
| 3.6.1 Detecting the Textured Noise in DCT Image..... | 25 |
| 3.6.2 Removing the Textured Noise Frequency Representative - Spikes | 26 |
| 3.6.3 Block Filtering Implementation | 29 |
| CHAPTER 4 | |
| SOFTWARE IMPLEMENTATION | 31 |
| 4.1 STRUCTURE..... | 31 |
| 4.2 PRIMARY FUNCTIONS..... | 32 |
| 4.2.1 Interface | 32 |
| 4.2.2 Variable Type Design..... | 35 |
| 4.2.3 Program design | 36 |
| CHAPTER 5 | |
| RESULT AND COMPARISON | 41 |
| 5.1 VISUAL RESULT AND COMPARISON | 41 |
| 5.2 NUMERICAL RESULTS AND COMPARISON | 48 |
| CHAPTER 6 | |
| CONCLUDING REMARKS | 53 |
| 6.1 SUMMARY AND CONCLUSION | 53 |
| 6.2 FUTURE RESEARCH | 53 |
| REFERENCE | 55 |
| APPENDICE | 56 |
| A. PROOF OF DIVING DCT FROM DFT | 56 |
| B. APPLICATION CODE | 58 |

List of Figures

| | |
|-------------------------------------------------------------------------------------------------|----|
| Figure 1 - Textured noise caused by acquisition | 3 |
| Figure 2 - Textured noise caused by transmission..... | 4 |
| Figure 3 - Textured noise caused by photo print paper | 5 |
| Figure 4 - Gaussian blur filtered image..... | 8 |
| Figure 5 - Median filter filtered image..... | 10 |
| Figure 6 - Filtering textured noise by MATLAB image tool box..... | 11 |
| Figure 7 - Low pass filter cut off high frequency component details..... | 12 |
| Figure 8 - Noise images and their DCT images | 17 |
| Figure 9 - 64×64 size textured noise image | 22 |
| Figure 10 - Textured noise DCT image..... | 24 |
| Figure 11 - Natural exponential equation wave..... | 27 |
| Figure 12 - Natural exponential equation wave analyze | 27 |
| Figure 13 - 2 -D taper filter | 28 |
| Figure 14 - Proposed band reject filter 3 -D image | 29 |
| Figure 15 - Software application structure | 31 |
| Figure 16 - Software user interface | 32 |
| Figure 17 - Software setting Interface..... | 34 |
| Figure 18 - Software thread process structure..... | 38 |
| Figure 19 - Filter setting flow graphic..... | 40 |
| Figure 20 - Clown face noisy image | 41 |
| Figure 21 - Removing textured noise from clown face noisy image..... | 42 |
| Figure 22 - Removing textured noise from clown face noisy image in the frequency domain | 43 |
| Figure 23 - Satellite mountain noisy image..... | 44 |
| Figure 24 - Removing textured noise from satellite mountain noisy image in spatial domain | 44 |
| Figure 25 - Removing textured noise from satellite noisy image in the frequency domain..... | 45 |
| Figure 26 - Removing textured noise from noisy old photograph in spatial domain | 46 |
| Figure 27 - Removing textured noise from noisy old photograph in the frequency domain..... | 48 |
| Figure 28 - PSNR testing image 1 | 49 |
| Figure 29 - PSNR testing image 2..... | 51 |

List of Tables

| | |
|--------------------------------------------------------------------------------------|----|
| Table 1 - Main User Interface Button Manual..... | 33 |
| Table 2 - Button Function of Setting Interface..... | 35 |
| Table 3 - Data Input Window Function of Setting Interface | 35 |
| Table 4 - Matrix Data Variables..... | 36 |
| Table 5 - Thread Parameters..... | 36 |
| Table 6 - Mathematical Results of Image Contains Only Low Frequency Information..... | 50 |
| Table 7- Mathematical Results of Image Contains High Frequency Information..... | 52 |

Chapter 1

Introduction

1.1 Research Background

Image representation plays an important role in image processing applications, and usually involves large quantities of data. The digital image is usually a two dimensional real matrix, with each point containing usually color information. This image information usually contains both value and noise data. Filtering the noise from the digital image data is akin to doing cosmetic surgery on the image. The main idea is to make the picture as close to a real life representation as possible.

There are various types of image noise. Most are photo-electronic types of noise. This includes impulse types, such as salt and pepper noise and line drop, and structured types, such as periodic, aperiodic, detector striping noise, etc. This thesis, will deal with textured noise, which is also classified as global periodic noise in the field of image processing. This textured noise is usually caused by image sensors, electronic interference, photo print paper, etc. This type of noise exists in almost every pixel of the image and highly degrades the image presentation and visibility.

The elimination of global periodic noise has been studied and discussed in many articles. The main solutions that have been proposed can be divided into two categories, spatial domain filtering and spatial frequency domain filtering. Spatial domain filtering provides low calculation load, but most of the time results in a

poor quality filtered image. Spatial domain filtering includes the use of the Gaussian filter and the Median filter. In spatial frequency domain filtering, frequency transform is implemented in order to collect all of the noise into a few frequency coefficient spikes to then be reduced or removed. Spatial frequency domain filtering achieves better filtered image quality, but signal conversion between the spatial and frequency domains is time consuming.

The project in this thesis focuses on solving an application problem, that of removing textured noise from photographs. With the development of computer technology, calculation capability of personal computers is now sufficient to the task making filtered image quality the most important measure to be considered. To provide better quality and best image information preservation, spatial frequency domain filtering is the preferred choice. Fast DCT (discrete cosine transform) will be shown to reduce the computational load of frequency domain filtering.

The main experimental environment for this project is MATLAB, along with a software implementation developed in Visual C++ that could be further implemented in large graphic software, such as Photoshop.

1.2 Problem Statement

Textured noise can generally result from three sources. It can result from the digital image acquisition process, such as a CCD (charged coupled device), a camera,

or a scanner. Electronic signal fluctuations in the detector can cause textured noise.

A noisy image caused by the acquisition process is shown in Figure 1.



Figure 1 - Textured noise caused by acquisition

A second source of noise can occur during image transmission such as from a wireless network. Lighting or other magnetic field disturbance may also cause some corruption. A noisy image created during transmission is shown in Figure 2.

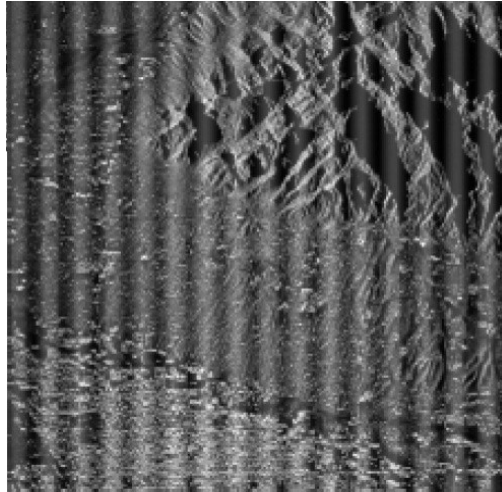


Figure 2 - Textured noise caused by transmission

A third source of textured noise can be caused by the image itself. Examples would be some characteristic in the paper on which the image was printed such as an old matte finish procedure for photographic protection or from some type of Copyright protection using a periodic pattern, such as in the photo in Figure 3.



Figure 3 - Textured noise caused by photo print paper

Textured noise also has three characteristics:

1. It is global. The noise covers almost every pixel in the image, which makes it hard to remove in the spatial domain without applying distortion.
2. It is repetitive. Textured noise usually has a constant period and a fixed shape.
3. It has energy centralized in a specific frequency. The consistent periodicity of the noise leads to well-defined energy spikes in the frequency transform of the noisy image.

Chapter 2

Current Solutions

Currently, several kinds of filters are available to reduce textured noise. In spatial domain filtering they are Gaussian blur and Median filtering. In the frequency domain, low pass filtering and band reject filtering are available. Reducing textured noise by using the frequency domain band reject filter gives much better filtered image quality, but spatial domain filtering provides better processing speed. Since for a photographic editing project, filtered image quality is more important than filtering speed, a frequency domain band reject filter is preferable. Currently, a frequency domain band reject filter has not yet been implemented in any public commercial graphic editing software. Only several big institutions, such as AVHRR^[1], AVIRIS^[2], Fuyo -1^[3], etc., are using frequency domain filtering and this is for some large image processing programs. The main frequency transform utilized in those programs is FFT (fast Fourier transform), which features a high redundancy when the input signal is always a real finite matrix, such as a digital image. High redundancy not only results in higher computational load but also increases the processing memory requirement.

Considering that there is a need to preserve image quality and to reduce filtering computational load, fast DCT (Discrete Cosine Transform) may be the best option. It is the most widely used transform method in image compression and, with a better energy compression ratio and a faster calculation speed, it is gradually being considered to be the best transform method in image processing.

2.1 Spatial Domain Filtering

The first consideration is to examine currently available solutions in spatial domain filtering. These include the Gaussian blur filter and the Median filter. Both can mask textured noise, but both also reduce edge sharpness, which results in a fuzzy filtered image.

2.1.1 Reducing Textured Noise Using Gaussian Blur

Gaussian blur is a type of image blurring filter that calculates every pixel as a weighted sum of its neighboring pixels. The weighted sum of every neighboring pixel is determined by the Gaussian function. Mathematically, applying a Gaussian blur to an image is convolving the image with a Gaussian function. The Fourier transform of a Gaussian function is another Gaussian function, and convolution in the spatial domain results in multiplication in the frequency domain. Therefore, in terms of the image in the frequency domain, applying a Gaussian blur has the same effect as multiple Gaussian waves, which is a Gaussian low pass filter.



Figure 4 - Gaussian blur filtered image

The result of filtering by Gaussian blur is shown in Figure 4. It can remove high frequency periodic noise, but it also reduces all the high frequency information. Using this method to reduce textured noise will either make the resulting image too fuzzy or will not remove the textured noise clearly.

2.1.2 Reducing Textured Noise Using Median Filter

The main idea of the Median filter is to run through the signal pixel by pixel, replacing each pixel with the Median element within the neighboring pixels. To demonstrate, using a window size of three with one pixel immediately preceding and following each pixel, a Median filter will be applied to the following simple

sequence x.

$$x = [2 \ 80 \ 6 \ 3]$$

The Median filtered output for sequence y will be:

$$y[1] = \text{Median}[2 \ 2 \ 80] = 2$$

$$y[2] = \text{Median}[2 \ 80 \ 6] = \text{Median}[2 \ 6 \ 80] = 6$$

$$y[3] = \text{Median}[80 \ 6 \ 3] = \text{Median}[3 \ 6 \ 80] = 6$$

$$y[4] = \text{Median}[6 \ 3 \ 3] = \text{Median}[3 \ 3 \ 6] = 3$$

$$y = [2 \ 6 \ 6 \ 3].$$

This method is widely utilized in image decontamination to remove the unusual or sharp color change in the image. But it has some of the same weaknesses as the Gaussian blur. It reduces edge sharpness resulting in the filtered image being fuzzy.



Figure 5 - Median filter filtered image

The result of applying a Median filter to a textured noise image is shown in Figure 5. While the image looks somewhat better than the Gaussian blur because certain image details, such as Chinese characters and hair, become blurred, the Median filter is not acceptable if those details are important.

2.2 Frequency Domain Filtering

2.2.1 Reducing Textured Noise by Low Pass Filter in the Frequency Domain

Currently, the most commonly used frequency domain filtering that can reduce textured noise is achieved by applying a low pass filter in the FFT matrix of the noise image, such as with the software MATLAB and mathematical algorithms

such as AVHRR^[1], AVIRIS^[2], Fuyo -1^[3], etc. The idea is based on the frequency transform centralizing image information in every frequency and being reordered by its frequency. If the textured noise is in a relatively high frequency, it may be removed by simply deleting all the high frequency components.



Figure 6 - Filtering textured noise by MATLAB image tool box

Figure 6 shows the operation of using MATLAB image tool box. One can see that the low pass filter works just fine when the image involves almost no high frequency information, but when the image contains valuable high frequency information the filtered image as in Figure 7 has very poor visual quality.



Figure 7 - Low pass filter cut off high frequency component details

2.2.2 Reducing Textured Noise by Removing Frequency Coefficient Spikes

The purpose of using a frequency domain band reject filter for reducing textured noise is to build a filter that rejects the noise frequency energy. This approach has been mentioned and discussed in some articles, but has never been used in public commercial software.

There four challenges:

- Noise frequency detection
- Noise energy reduction level
- High programming complexity
- High computational load

The following chapters address finding the best method(s) to overcome these challenges and to implement them in public commercial graphic software.

Chapter 3

Proposed Solution - DCT Spatial Frequency Filtering

3.1 Proposed Solution Statement

Frequency transform relies on the premise that pixels in an image exhibit a certain level of correlation with their neighboring pixels. A transformation is, therefore, defined to map the spatial correlated data into transformed uncorrelated coefficients. The transformation utilizes the fact that the information content of an individual pixel is relatively small. When discussing visual contribution of pixels, information is reordered by their frequencies. The coefficients on these frequencies represent the energy at those frequencies.

Textured noise, which has a certain frequency in the spatial domain, may be easily removed in the frequency domain by reducing the total energy at that frequency. Therefore, a DCT band reject filter will reduce this energy total more effectively than most existing graphic editing software. Fast DCT is utilized because there are many advantages of using fast DCT versus FFT in reducing textured noise. After the basic frequency transform is selected, a frequency domain filter can be designed to detect and remove the spikes (high energy coefficient peak), and a noise reduced image can be produced after an inverse frequency transform.

3.2 Review of the Definition of DCT

Over the previous decade, Discrete Cosine Transform (DCT) has emerged as the de-facto image transformation in most visual systems. DCT has been widely deployed by modern video coding standards (such as MPEG, JVT, etc.) but has

rarely been utilized in graphic processing.

The most common DCT definition of a 1 -D sequence of length N is :

$$C[u] \equiv a(u) \sum_{x=0}^{N-1} f[x] \cos\left(\frac{(2x+1)\pi u}{2N}\right), \quad u = 0, 1, 2, \dots, N-1 \quad (3.1)$$

where $a(u)$ is defined as"

$$a(u) \equiv \begin{cases} \sqrt{\frac{1}{N}} & \text{for } u = 0 \\ \sqrt{\frac{2}{N}} & \text{for } u = 1, 2, \dots, N-1 \end{cases} \quad (3.2)$$

In graphic processing, images are usually treated as matrices, and the matrix

form of this DCT equation is defined as:

$$c[u, x] \equiv a(u) \cos\left(\frac{(2x+1)u\pi}{2N}\right), \quad (u, x = 0, 1, \dots, N-1) \quad (3.3)$$

Being that digital images are 2 -D matrices, 2 -D DCT transform is implemented.

The 2 -D DCT's matrix form is defined as:

$$\mathbf{X} = \mathbf{c}(\mathbf{c}\mathbf{X}^T)^T = \mathbf{c}\mathbf{X}\mathbf{c}^T \quad (3.4)$$

where its inverse is:

$$\mathbf{x} = \mathbf{c}^{-1}(\mathbf{c}^{-1}\mathbf{X}^T)^T = \mathbf{c}^T\mathbf{X}\mathbf{c} \quad (3.5)$$

3.3 Properties of DCT Beneficial to the Proposed Solution

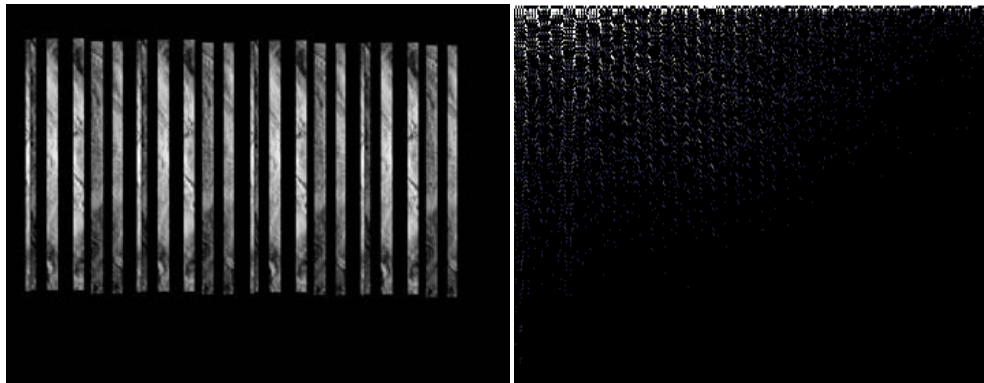
3.3.1 Properties of DCT Beneficial for Improving Filtered Image Quality

1. Energy compression

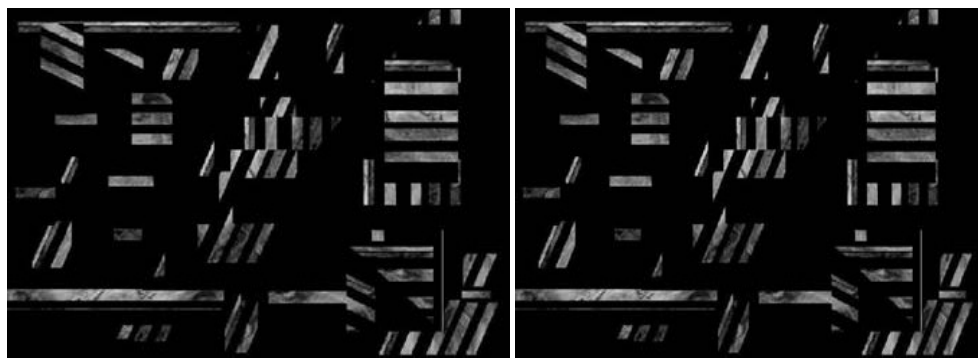
DCT has been proven to have better energy compression efficacy than DFT.

Efficacy of a transformation scheme can be directly gauged by its ability to pack input data into few coefficients. DCT exhibits excellent energy compaction for highly correlated images, which provides less frequency coefficient modification in the filtering procedure.

The periodic information that has a fixed frequency is gathered in the frequency domain. As a result, the better the energy compression property, the less the frequency coefficients need to be reduced. Block artifacts can be directly reduced thereby preserving more image information.



(a)



(b)

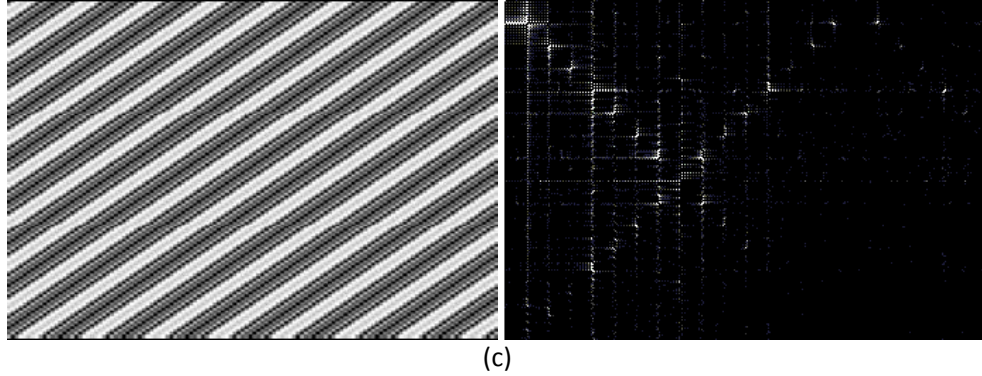


Figure 8 - Noise images and their DCT images

Figure 8 clearly shows the energy compaction property of DCT. Low frequency image information is compacted to the left top corner in a frequency domain image, and the special periodic pattern in Figure 8 (c) is compacted into coefficient patterns.

2. Even assumption in deriving DCT from DFT

DCT is derived from Fourier Transform by creating an even sequence according to the original signal sequence in order to achieve a zero imaginary part matrix. Then the symmetry property of the sequences results in the DCT definition ^[Appendix A].

With this assumption, when imposing periodicity in a spatial domain signal for frequency transform, DCT does not introduce discontinuity. As a rule, the original image is segmented into smaller blocks for computation convenience. Because of the underlying cyclical assumption, DFT will introduce discontinuity in the time domain, when partial coefficients are used to reconstruct the original image. Coefficients are processed through a filter in the frequency domain. This

discontinuity will introduce some corresponding artifacts, known as the Gibbs phenomenon, and these types of artifacts are called block artifacts. In DCT, even symmetry needs to be assumed while truncating the time signal. No discontinuity or related artifacts are introduced, therefore, blocking artifacts are significantly reduced.

3.3.2 Properties Beneficial for Improving Filter Programming Speed

1. Orthogonality

With this property, a bit reversal algorithm can be utilized to reduce the programming computational load. In frequency transform, the definition of every transform is based on the property of orthogonality. DCT matrix

$$\mathbf{c}(\mathbf{u}, \mathbf{x}) = a(n) \cos\left(\frac{(2x+1)u\pi}{2N}\right), \mathbf{c}(\mathbf{u}, \mathbf{x}) \text{ is a real orthogonal matrix :}$$

$$C = \sqrt{\frac{2}{n}} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \cdots & \frac{1}{\sqrt{2}} \\ \cos \frac{\pi}{2n} & \cos \frac{3\pi}{2n} & \cdots & \cos \frac{(2n-1)\pi}{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \cos \frac{(n-1)\pi}{2n} & \cos \frac{(n-1)3\pi}{2n} & \cdots & \cos \frac{(n-1)(2n-1)\pi}{2n} \end{bmatrix} \quad (3.6)$$

$$C^{-1} = C^T = \sqrt{\frac{2}{n}} \begin{bmatrix} \frac{1}{\sqrt{2}} & \cos \frac{\pi}{2n} & \cdots & \cos \frac{(n-1)\pi}{2n} \\ \frac{1}{\sqrt{2}} & \cos \frac{3\pi}{2n} & \cdots & \cos \frac{(n-1)3\pi}{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{\sqrt{2}} & \cos \frac{(2n-1)\pi}{2n} & \cdots & \cos \frac{(n-1)(2n-1)\pi}{2n} \end{bmatrix} \quad (3.7)$$

The equation above (3.7) shows that, not only is the DCT matrix C itself

orthogonal, but also every row in matrix C is orthogonal with each other. This means that when doing matrix computation, the row order of the DCT matrix can be changed to achieve better computation speed. This property is utilized in the recursive fast DCT algorithm, which makes it faster than FFT.

2. Symmetry

A DCT matrix can be reordered and partitioned into four quadrants. For the even numbered rows, the left and right quadrants are the same. For the odd numbered rows, the left and right quadrants are opposite in sign. The orthogonal and the symmetry properties are the main reasons why the recursive fast DCT algorithm is faster than FFT.

3. Pure real matrix computation

DCT is a real transform while DFT is complex. DFT transforms a complex signal into its complex spectrum. However, when the signal is real, which is always true in a digital image application, half of the data is redundant. In the time domain, the imaginary part of the signal is all zero. In the frequency domain, the real part of the spectrum is even symmetry and the imaginary part is odd symmetry. On the other hand, DCT is a real transform that transforms a sequence of real data points into its real spectrum and therefore avoids the problem of redundancy. Therefore, when implementing DCT into the proposed solution, there is no imaginary part, unlike DFT, where both the real and imaginary parts must be dealt with. This will improve the programming time and reduce complexity.

3.4 Review of Fast Discrete Cosine Transform

To achieve better computation speed, Fast DCT is preferred. Fast DCTs are fast algorithms. They simplify the computational structure and lower the computational complexity for the computation of the discrete cosine transform (DCT). Fast DCTs have been adopted in most image processing in general, and in data compression coding, filtering, and feature extraction in particular.

Many fast algorithms for computing the DCT have been published. These algorithms can be classified into the following categories based on their methods of approach:

1. Indirect computation. Fast Fourier transform (FFT) and Walsh Hadamard transform are used to obtain the DCT, but unnecessary operations are often involved in the computation steps.
2. Direct factorization. Using sparse matrix factorization offers speed advantages over the other methods because a unitary matrix, theoretically, can always be factorized into products of relatively sparse matrices.
3. Recursive Fast DCT. Recursive computation is implemented into Fast DCT and is intended to generate higher order DCT from lower order DCT. Hou^[5] presented a numerically stable, fast, and recursive algorithm for computing DCT in 1987. It is much faster than the above two fast DCT algorithms. Similar to the Cooley Tukey FFT algorithm, this algorithm allows generating the next higher order DCT from two identical lower order DCTs.

Fast DCTs have better computation speed than FFT. Chen and Smith^[6] proved this in 1977. The fast DCT algorithm they presented takes only one third as many calculation steps as the FFT.

The fast DCT C++ program utilized in the proposed solution is based on the recursive decomposition algorithm for q^m length DCT, suited for implementation in hardware and parallel processing, as proposed by DuanXiang Yin from South China University of Technology in 2000^[7].

3.5 The Appearance of Textured Noise in Frequency Domain

Having shown that DCT is the preferred choice for frequency transform, what is the appearance of the textured noise in the frequency domain? The periodic textured information in a spatial domain image leads to well defined coefficient spikes in a frequency domain image. To explain the relationship between textured artifact period and the location of the coefficient spikes, the following example is provided.

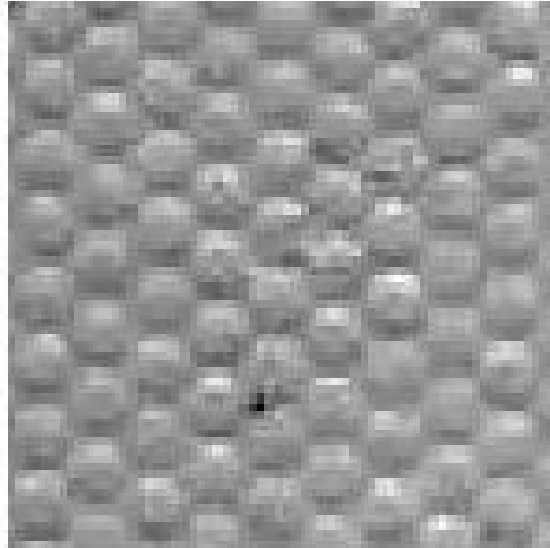
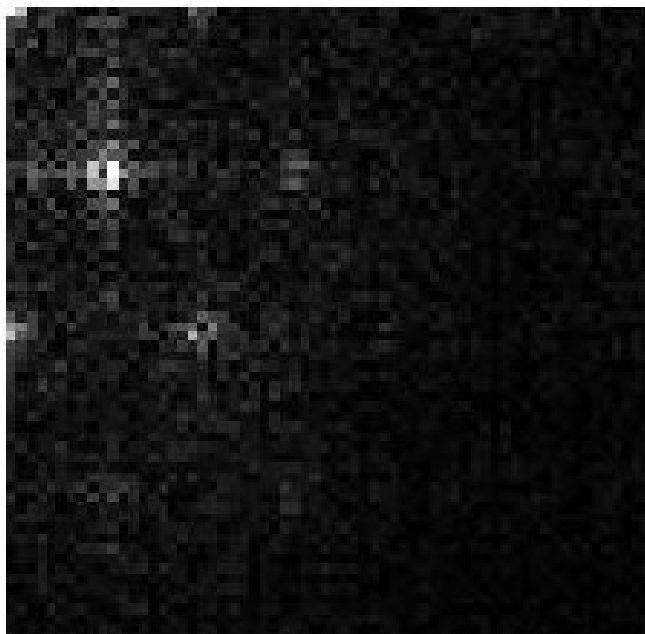
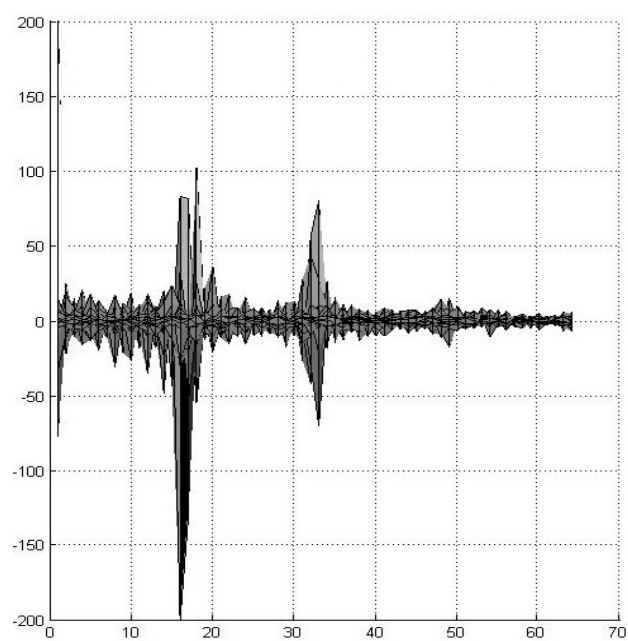


Figure 9 - 64×64 size textured noise image

A photo is sampled by a scanner and normalized so that the sampled photo length is 1. Then for a size 64 noise image (shown in Figure 9), the highest frequency is 64, and the textured mattes frequency is around 4 to 5 (by row) and 8 to 9 (by column). After the DCT operation, frequency spikes in frequency domain images at these frequencies would be expected.



(a) x-y view



(b) y-z view

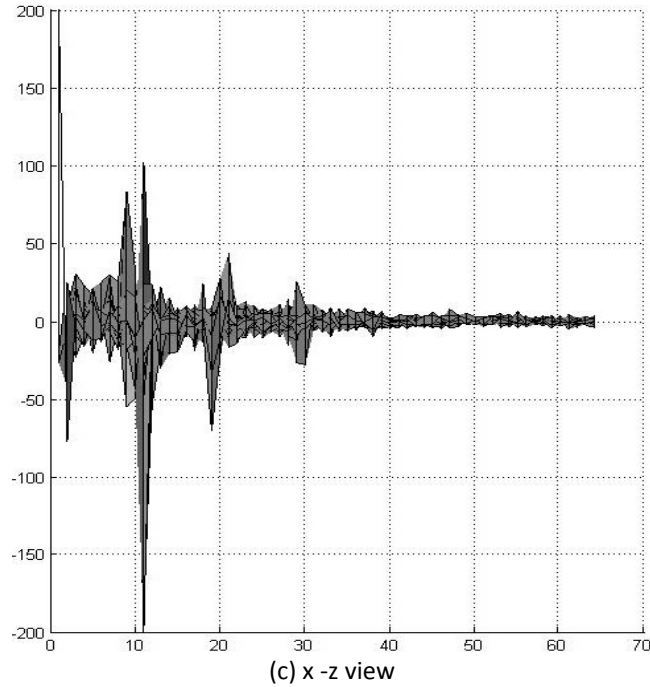


Figure 10 - Textured noise DCT image

The biggest frequency spike shows up at around row 10 (by x -axis) and column 17 (by y -axis), which means the textured noise's frequency in a DCT matrix is 10 by row and 17 by column. The number is twice what would be realized by counting from a spatial domain. This is caused by the even symmetry assumption in deriving DCT from DFT ^[Appendix A]. The spike is exactly at the frequency counted from Figure 9.

As demonstrated, the textured artifacts in a spatial domain image leads to the coefficient spikes at twice its frequency in a DCT frequency domain. All other smaller coefficient spikes shown in Figure 10 are the ultra harmonics.

3.6 Algorithm

From the previous discussion, DCT is shown to be a fundamental frequency

transform, and the appearance of the textured noise is proven to be a frequency spectrum spike at twice its frequency in the frequency domain image. So the global periodic noise removal problem becomes a frequency spectrum spike detection and reduction problem. Therefore, a DCT band reject filter is used to remove textured noise. Fast DCT is implemented as the frequency transform method instead of FFT for better computation efficiency and less blocking artifact effect, and DCT content (only a real number matrix in frequency domain) will again greatly reduce the programming complexity of the filter. After selecting DCT as the fundamental frequency transform, the next concern is to locate and remove the spikes, in the most straight forward manner so as to provide the best quality.

3.6.1 Detecting the Textured Noise in a DCT Image

As addressed in the previous section, the textured noise has repetitive and consistent characteristics throughout an image, and the textured artifacts lead to frequency coefficient spikes in the frequency domain. They are global in a spatial domain, and compressed in a frequency domain. These properties provide two options for detecting the spikes.

The first option is to count the period directly from the spatial domain image. As block filtering is being implemented, the original image is divided into small blocks before filtering in order to achieve better calculation speed. The period of the textured artifact in a single block is countable. The frequency of the textured noise can determine the coordinates of the spikes. This method requires carefully

counting out the period but this is not an efficient method of detection.

A second option is to detect the periodic noise in a frequency domain image. The textured noise leads to spikes in a frequency domain. The spikes can simply be detected and circled out manually from the DCT frequency domain image. This is a straightforward method, simple to program, and easy to use. Therefore, it is the method to be implemented in the software implementation of this project.

3.6.2 Removing the Textured Noise Frequency Representative Spikes

After circling out the noise frequency spectrum spike area, several options can be used to remove the spikes, which include setting all the values in that circle to 0, or multiplying each of them by a small constant, such as 0.1. But these methods are not sufficient. As shown in Figure 9, the shapes of the spikes look like a taper. The objective is to reduce the center of the circle more than the edge.

In order to reduce the taper spikes, a taper filter is created with a value range from 1 to 0. In order to adapt to various spike shapes and frequencies, the center coordinates and the radius of the bottom circle of this taper filter must be adjustable.

Equation $f(x) = e^{-\frac{(x-b)^2}{c^2}}$ fully fits the requirement, and a plot is shown in

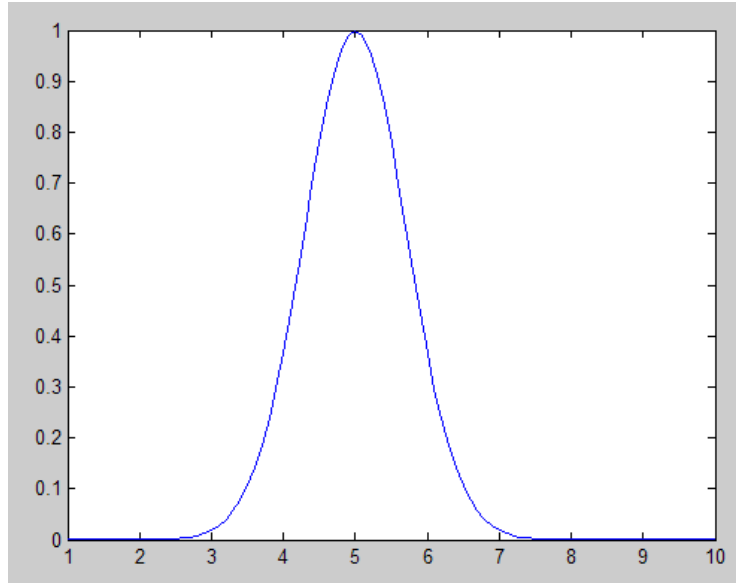


Figure 11 - Natural exponential equation wave

Figure 11, where $0 < x < 10$, b is the x-coordinate, and c is half of the width of the base. More specifically, c is increasing from 1 to 4 ($c=1$ -1 -4). The plot of the equation is shown in Figure 12.

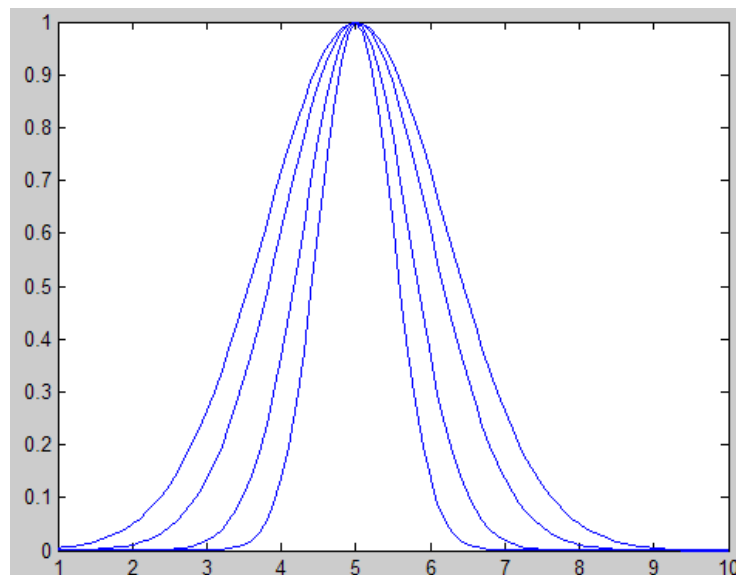


Figure 12 - Natural exponential equation wave analyze

For the image processing problem, which is a 2-D matrix, a 2-D version of this equation is created, as shown in equation 3.10,

$$f(x, y) = e^{-\frac{x(x-x_0)}{r^2} - \frac{y(y-y_0)}{r^2}} \quad (3.10)$$

and the plot of it is shown in Figure 13.

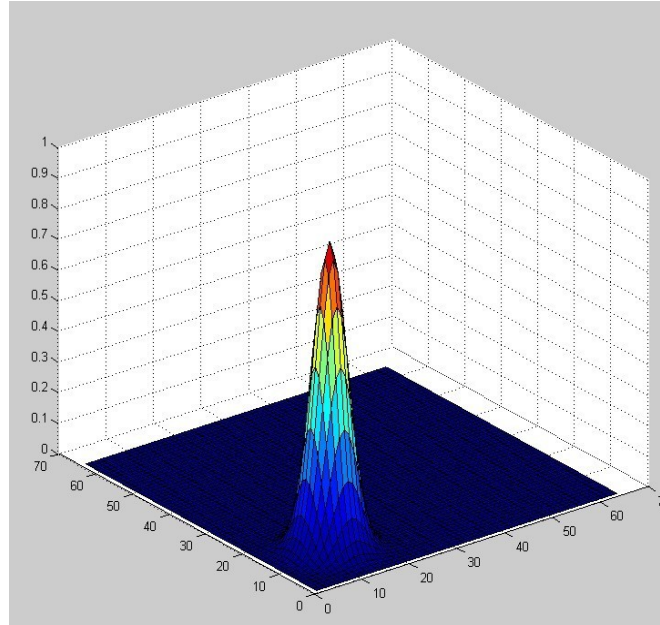


Figure 13 - 2 -D taper filter

(x_0, y_0) is the coordinate of the center top point and r is the radius of the bottom circle. The inverse taper filter can be calculated by $W = 1 - f(x, y)$. After the fundamental spike and its ultra harmonics are treated by a sized taper filter, a 3 -D view of the proposed band reject filter is shown in Figure 14.

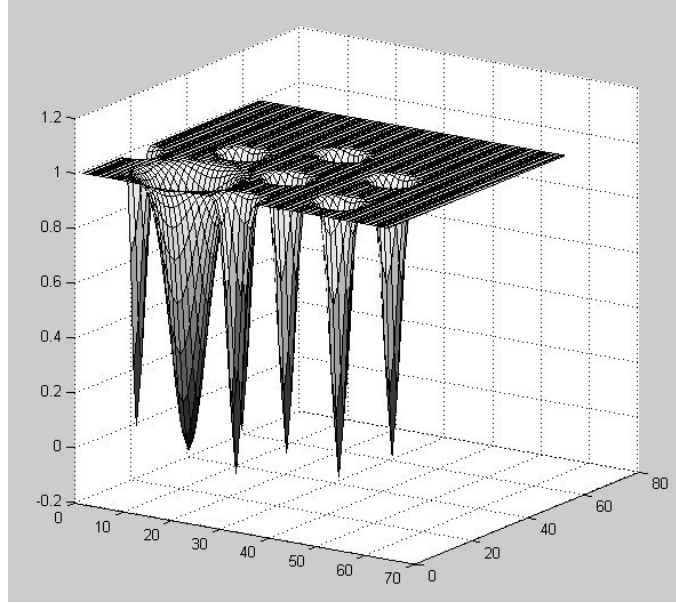


Figure 14 - Proposed band reject filter 3 -D image

3.6.3 Block Filtering Implementation

Again, reducing computational load is always a concern. Processing the filter block by block will improve programming speed. Given the fact that the textured noise is globally repetitive in a spatial domain, the frequency and amplitude of the frequency coefficient spikes in every block are the same. Only one or two blocks need to be tested for creating the taper filter. It can then it can be applied to every block. But dividing an image into blocks introduces another type of noise, block artifacts. In any reconstruction of the block from only a partial number of Fourier coefficients, the block artifacts will emerge.

The proposed solution provides two ways of minimizing block artifacts. First, given the fact that DCT is a fully reversible procedure, it can be concluded that the lower the frequency coefficient, the lower the level of block artifacts. In the

proposed solution, only the frequency coefficients of the textured noise are reduced, and with the greater compression property of DCT, block artifacts are minimized. Second, compared to FFT, DCT with its even assumption property does not introduce discontinuity, and this greatly reduces block artifacts. At this point, the digital image can be divided into smaller square matrix blocks for better calculation speed without creating many block artifacts.

Chapter 4 Software Implementation

4.1 Structure

The software created for the proposed solution was written in a Visual C++ environment. Its function is to calculate the DCT matrix of a digital image, display it from three points of view, provide a user interface for the filter setting base and then apply the filter to the image block by block.

The flow graphic of the software is shown in Figure 15.

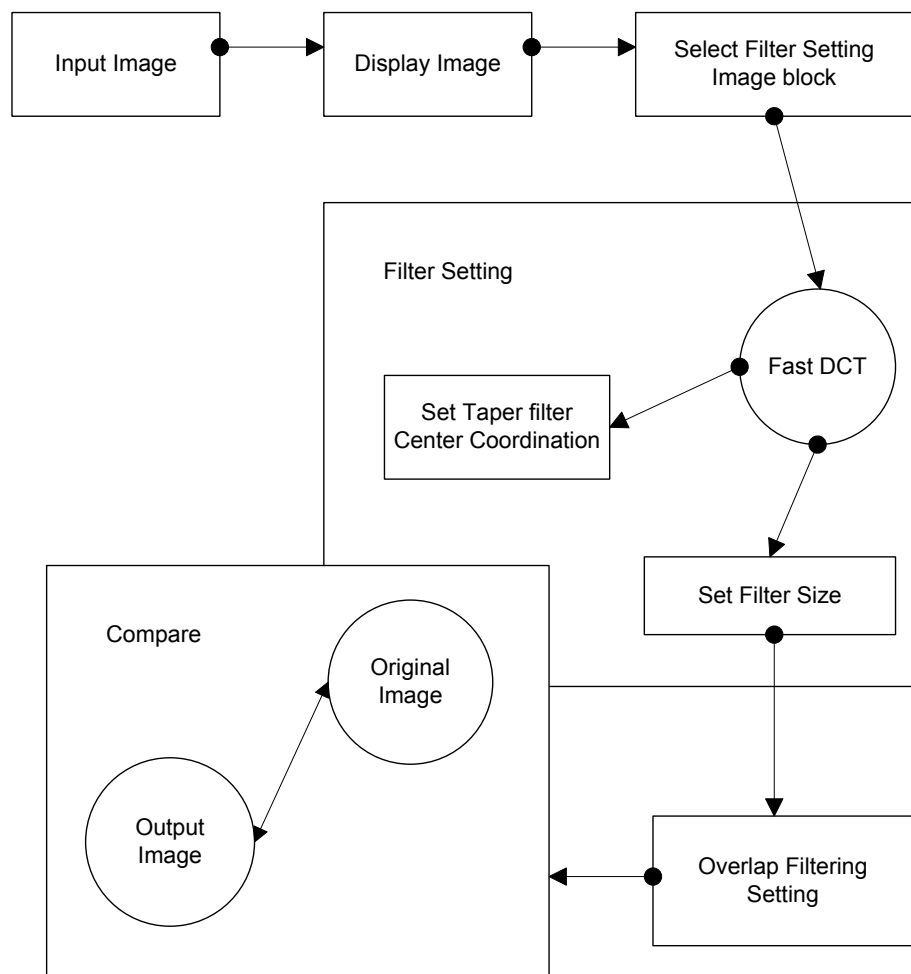


Figure 15 - Software application structure

4.2 Primary Functions

4.2.1 Interface

1. Main User Interface

The user interface is shown in Figure 16. It includes three image display windows, one data input window, and nine functional buttons.

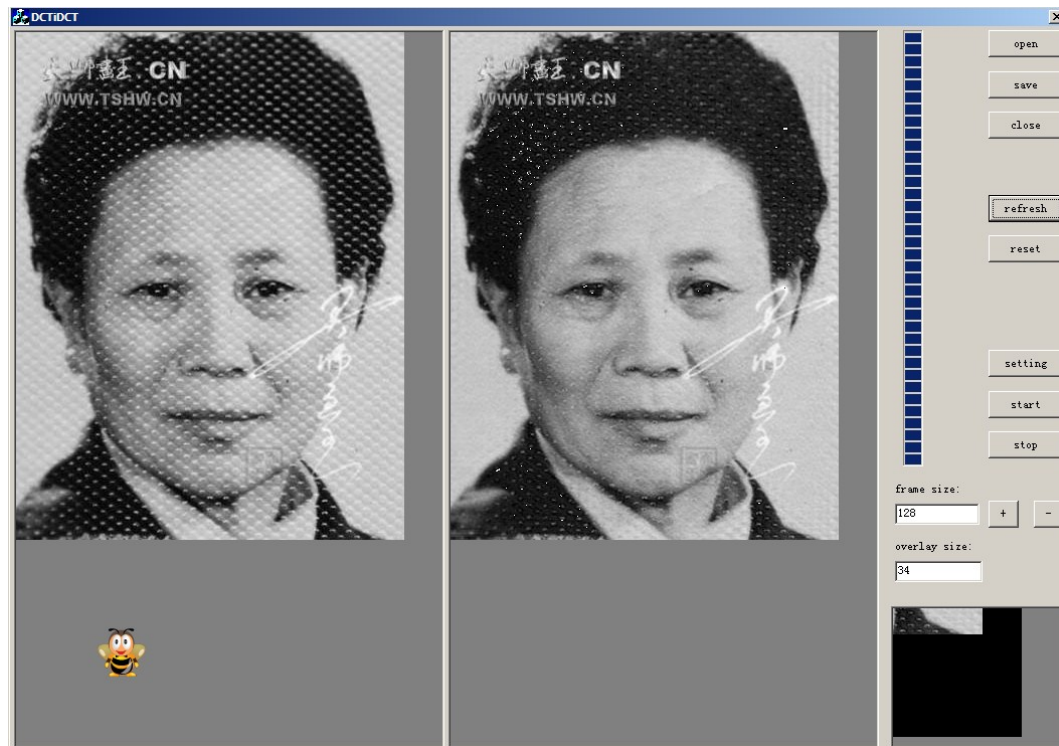


Figure 16 - Software user interface

Three image display windows display the original image, a filtered image, and the selected block for filter setting. There are nine function buttons in this interface, and are shown in Table 1. One data input frame customizes the step level of the block filtering procedure.

| Button | Function |
|------------|------------------------------------------------|
| 'open' | Load in the noise image |
| 'save' | Save the image in result filtered image window |
| 'close' | Exit program |
| 'refresh' | Refresh screen |
| 'reset' | Reset the filtered image to original image |
| 'setting' | Pop 'setting' interface |
| 'start' | Star filtering |
| 'stop' | Stop filtering |
| '+' or '-' | Adjust the selected block size, 64/step |

Table 1 - Main User Interface Button Manual

2. Filter setting interface

A pop out window (as shown in Figure 17) is displayed when you click the setting button. It has five image display windows, six function buttons and four data input windows.

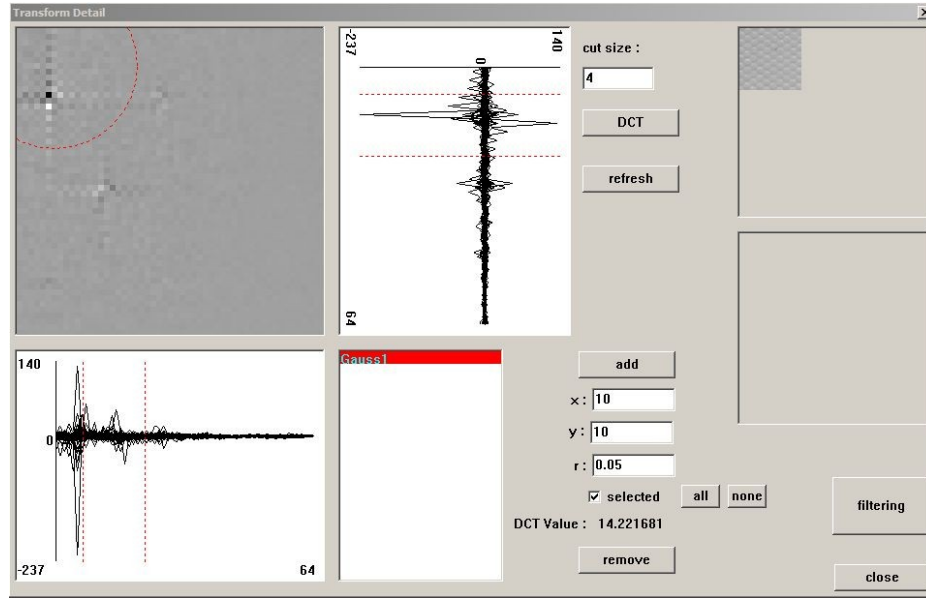


Figure 17 - Software setting Interface

Of the five image display windows in this interface, three are on the left side. These are the 3-D DCT images for filter parameter settings. The other two are on the right side, and are the image blocks selected from a previous interface and its filtered image.

The functions of the six buttons are shown in Table 2. Four data input buttons are shown in Table 3.

| | |
|-------------|----------------------------------------|
| 'DCT' | Operate DCT of the select image block. |
| 'refresh' | Refresh the whole screen. |
| 'add' | Add a taper filter. |
| 'remove' | Remove a taper filter. |
| 'filtering' | Apply the filter to the image. |

| | |
|---------|------------------------------------|
| 'close' | Return to the main user interface. |
|---------|------------------------------------|

Table 2 - Button Function of Setting Interface

| | |
|------------------------------|---------------------------------------------------------------------------------------|
| 'Cut Size' | Set the cutoff size of the low frequency amplitudes to get a better display on spikes |
| Taper filter select window | Let user to select each taper filter for relocating or resizing. |
| Spike coordinate 'x' and 'y' | Set the x_0 and y_0 coordinate of spikes. |
| Taper filter size | Set the size of each Taper filter. |

Table 3 - Data Input Window Function of Setting Interface

4.2.2 Variable Type Design

1. Image class

The image class code was obtained from an Internet website. The main functions provide for opening any type of image, saving the image object as any type of image, and the creation of a standard description of the image object, such as length, width, and height.

2. Data matrix and taper filter data matrix

The matrix is saved as a 2-D form in memory. In the filter setting process, several taper filters may be required to reduce the coefficient spikes if the periodic noise contains harmonics. The Taper Filter Equation (3.10) shows that the filter needs three variables, along with the matrix variables listed in Table 4.

| | |
|----------------------|-------------------------------------------------------------------------------|
| 'm_dataMatrix[i][j]' | Variable for the image data at i th row and j th column |
| 'm_dctMatrix[i]' | Variable for DCT matrix data |
| 'x0' & 'y0' | Contains the coordinate of the Taper filter |
| 'pr' | Taper filter bottom circle radius |
| 'rgb' | A color variable for using different colors to distinguish the Taper filters |

Table 4 - Matrix Data Variables

4. Thread parameter structure

The whole image process will be handled as a thread. Parameters are transmitted between functions. Utilized parameters are shown in Table 5.

| | |
|---------------|-------------------------------------------------------------------------------------------------------------------|
| 'processLoop' | Sign program process stop or continue, evaluate by 'stop' button in main interface. |
| 'frameSize' | Represent block size, evaluate by 'frame size' button. |
| 'progressCtr' | Pointer variable represents processing tempo. Use for time meter and for image refresh when processing filtering. |

Table 5 - Thread Parameters

4.2.3 Program design

Only main the program functions will be discussed in this section.

1. Thread process

The processing may be very slow if the image size is very large. In order to terminate the process at any time, a thread process (shown in Figure 18) is used to carry the whole process.

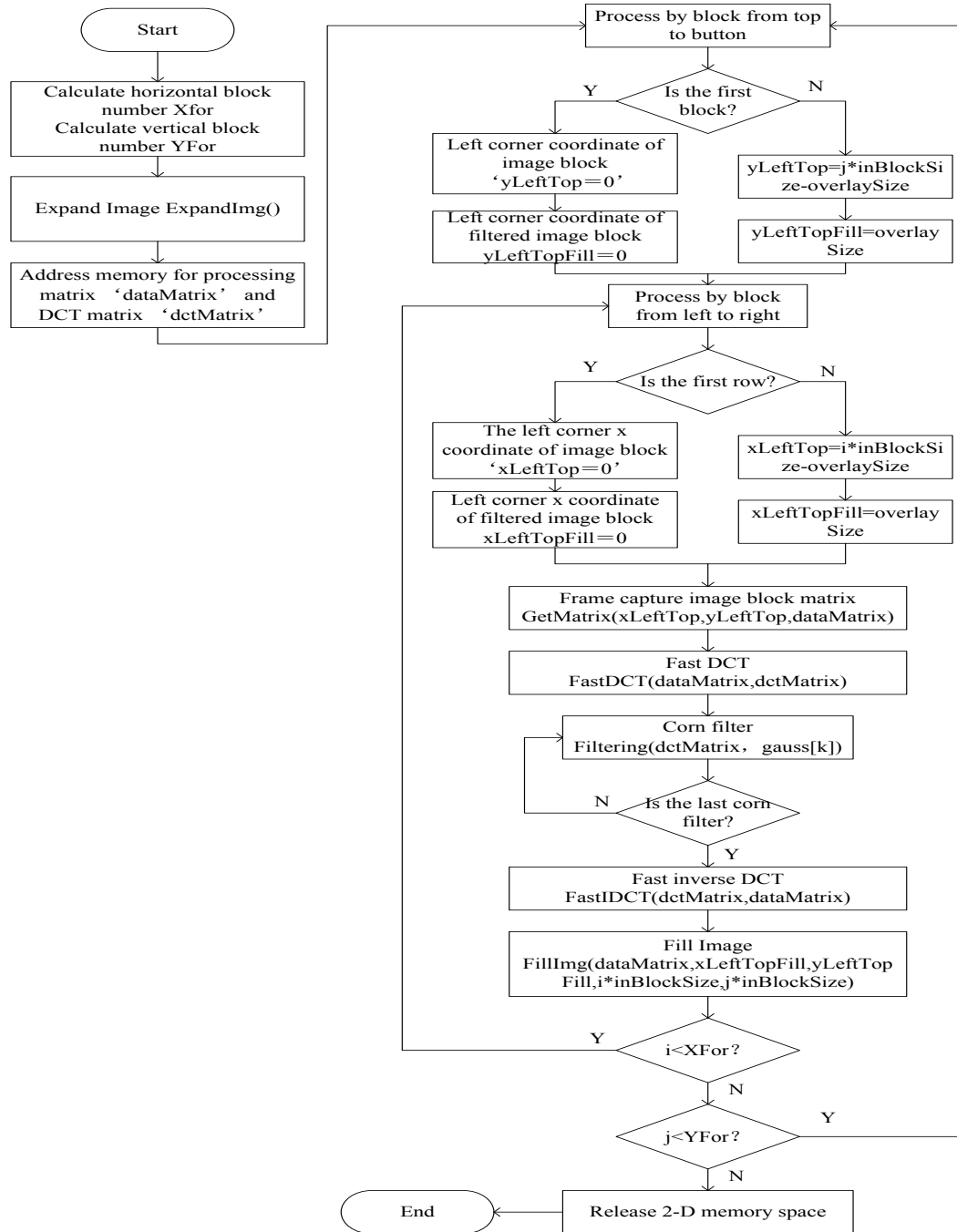


Figure 18 - Software thread process structure

2. Image Expanding - 'ExpandImg'

In order to make the image size match integral times of the image block size,

the 'ExpandImg' function is created for expanding the image to a given image size.

3. Red frame captured image block - GetMatrix

By capturing an image block from the original image, it can be saved to '*dataMatrix*' for the filter setting application. The frame size can be adjusted by 32 pixels per step. The captured image block will be displayed in the main user interface, in the right bottom corner window.

4. Fast DCT and Fast Inverse DCT

Fast DCT is utilized in this software to improve the speed of the calculation. A recursive fast DCT algorithm provided by Rui -Xiang Yin in 2000^[7] is utilized.

Because many coefficients in DCT calculations are iteratively calculating, an initial DCT matrix is pre-calculated. The program will search the coefficients instead of calculating them every time needed. This will highly reduce the computational steps. The whole DCT process could be divided into 4 paths:

- 1) Initial pre-calculation of DCT matrix
- 2) Forward calculation to the end
- 3) Backward calculation to the end
- 4) Calculating 2-D DCT from 1-D DCT

After implementing this Fast DCT code into the proposed application, the calculation speed is significantly improved. For a 256 X 256 matrix, it only takes 94 ms, compared to DCT, which would take 575172 ms.

5. Filter matrix

The filter matrix design flow is shown in Figure 19. This procedure is based on the algorithm in Section 3.4.

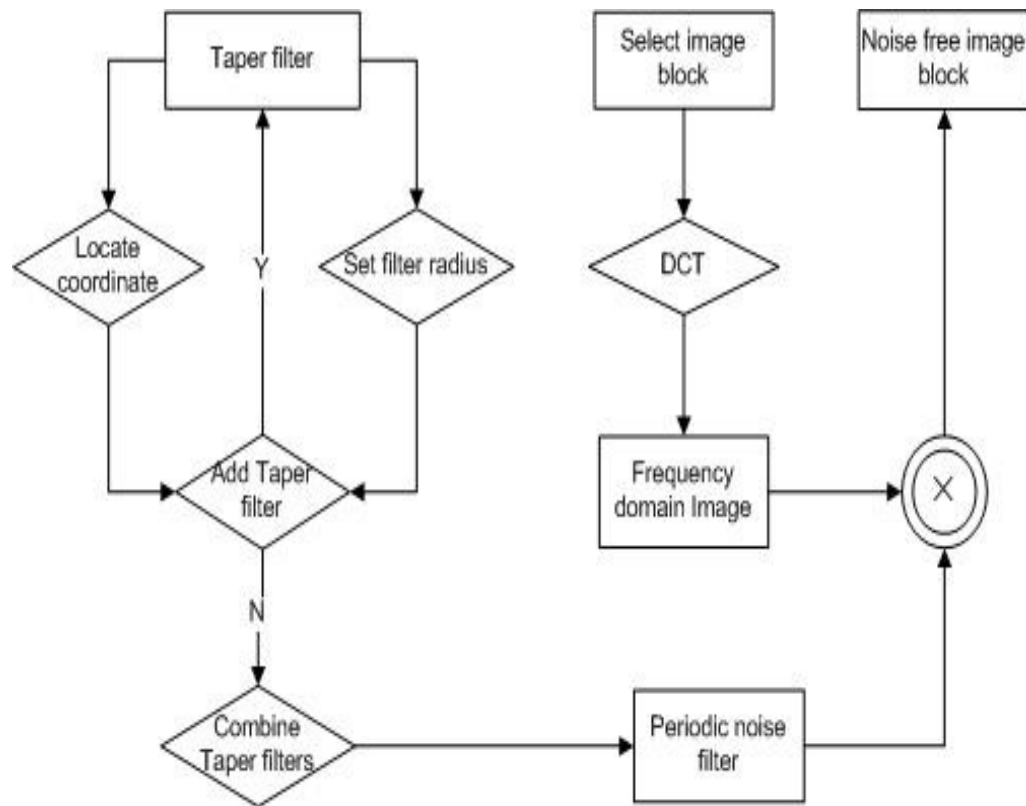


Figure 19 - Filter setting flow graphic

Chapter 5

Result and Comparison

5.1 Visual Result and Comparison

What follows is a visual comparison of applying three currently available filters to a noisy image. The Gaussian blur and Median filters are applied in the spatial domain and the low pass filter is applied in the frequency domain. The results of these three filters are then compared to the application of a fast DCT filter (the proposed solution) also in the frequency domain. Three example situations follow. The first is applying the filters to textured noise generated in the acquisition process as shown in Figure 20.

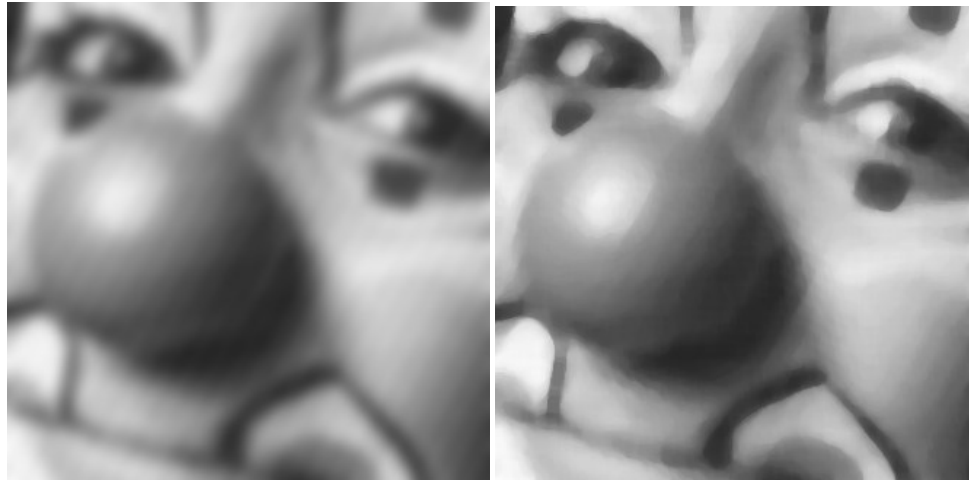


Figure 20 - Clown face noisy image

The results of applying the Gaussian blur filter and of applying the Median filter in the spatial domain are shown in figures 21a and 21b respectively. As can be

seen, a considerable amount of noise remains.

Figure 21 - Removing textured noise from clown face noisy image
in spatial domain



21 (a) Gaussian blur filtered image

21 (b) Median filter filtered image

The results of applying the low pass filter and of applying the fast DCT, the proposed filter, in the frequency domain are shown in figures 22a and 22b respectively. While a considerable amount of noise remains from the low pass filter, the use of the fast DCT results in a markedly improved visual image.

Figure 22 - Removing textured noise from clown face noisy image in the frequency domain



22(a) Low pass filter filtered image

22 (b) Proposed filter filtered image

The second example is that of applying the filters to textured noise generated in the transmission process. Figure 23 is the acquired noisy image. Figures 24a and 24b show the results of applying the Gaussian blur filter and of applying the Median filter in the spatial domain.

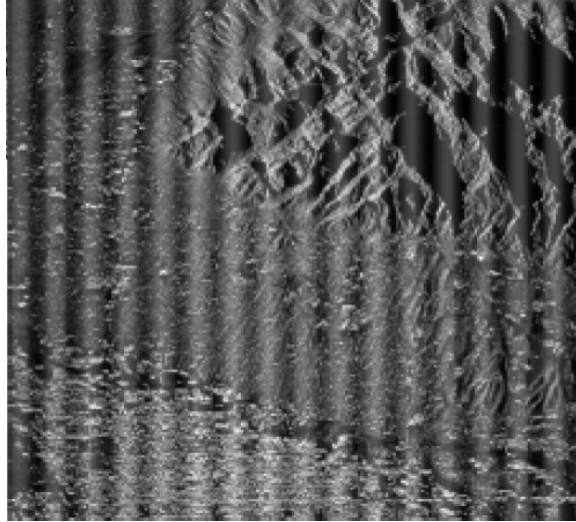
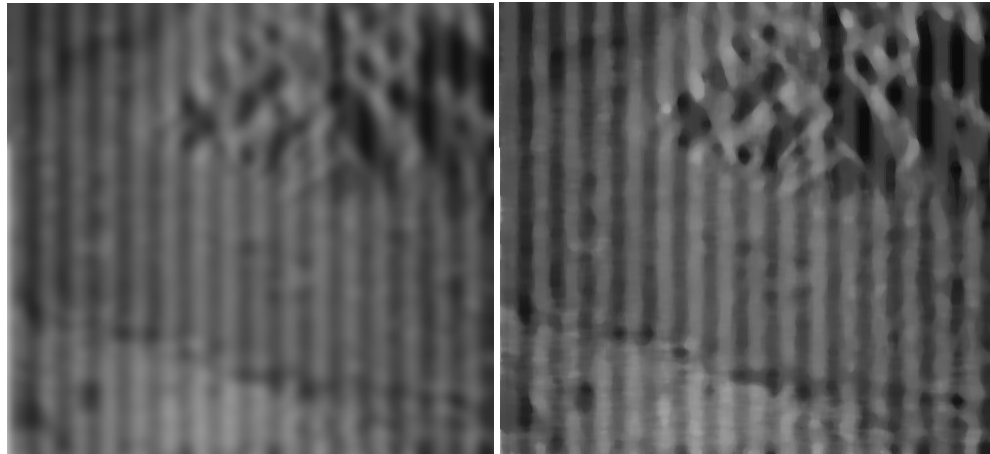


Figure 23 - Satellite mountain noisy image

Figure 24 - Removing textured noise from satellite mountain noisy image in spatial domain

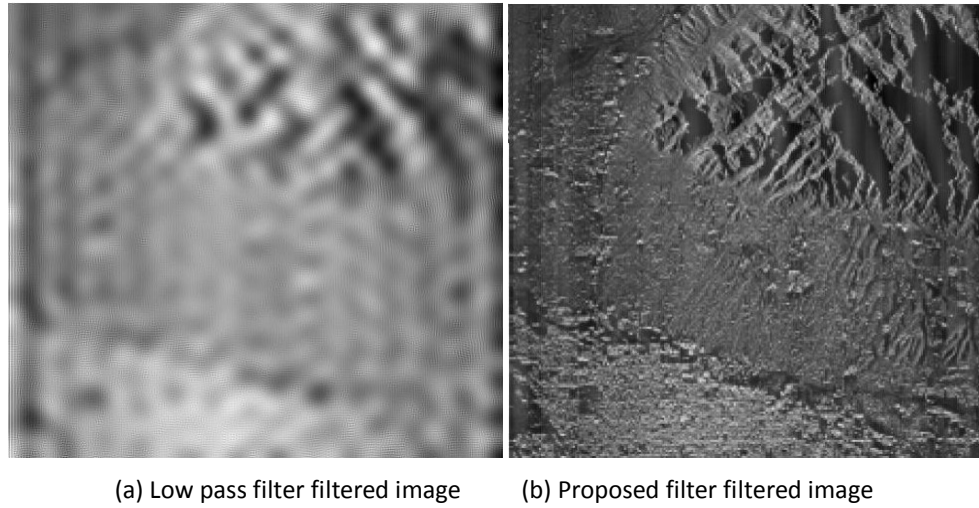


24(a) Gaussian blur filtered image

24 (b) Median filter filtered image

Figures 25a and 25b show the results of applying the low pass filter and of applying the fast DCT, the proposed filter, in the frequency domain.

Figure 25 - Removing textured noise from satellite noisy image in the frequency domain



The third example situation is to apply the filters to textured noise that results from the photo paper. Figure 26 is the noisy old photo. Figures 27a and 27b show the results of applying the Gaussian blur filter and of applying the Median filter in the spatial domain.

Fig.26 Noisy old photograph



Figure 26 - Removing textured noise from noisy old photograph in spatial domain



27(a) Gaussian blur filtered image



27(b) Median filter filtered image

Figures 28a and 28b show the results of applying the low pass filter and of applying the fast DCT, the proposed filter, in the frequency domain.

Figure 27 - Removing textured noise from noisy old photograph in the frequency domain



28(a) Low pass filter filtered image

28 (b) Proposed filter filtered image

In all the three of the above comparisons, the proposed filter, fast DCT, provides the best visual quality. This is especially seen in Figure 28b. The textured noise is fully removed, with legible Chinese characters preserved.

5.2 Numerical Results and Comparison

Images were also compared numerically by examining the PSNR (peak signal to noise ratio) of each filtering method. The definition of PSNR is shown in equation:

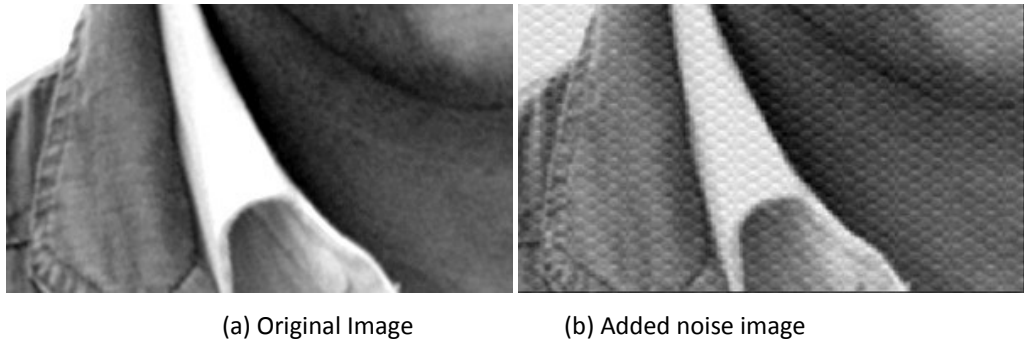
$$PSNR = 20 \times \log_{10}(\frac{MAX_I}{\sqrt{MSE}})$$

MAX is the maximum pixel value of the image and MSE is the mean square error, which for two $m \times n$ monochrome images I and K, one of the images is considered a noisy approximation of the other and is defined as:

$$MSE = \frac{\sum_{i=1}^m \sum_{j=1}^n |I(i,j) - K(i,j)|^2}{mn}$$

In order to compare the filtered image results of the different methods, textured noise is added into two noise free images. The first image, Figure 29a, contains almost no high frequency information. In Figure 29b, a considerable amount of textured noise has been added. Figures 29 c – f show the results of applying the filters, as labeled to the noise added image (figure 29b). The numerical results of the application of these filters are shown in Table 6.

Figure 28 - PSNR testing image 1



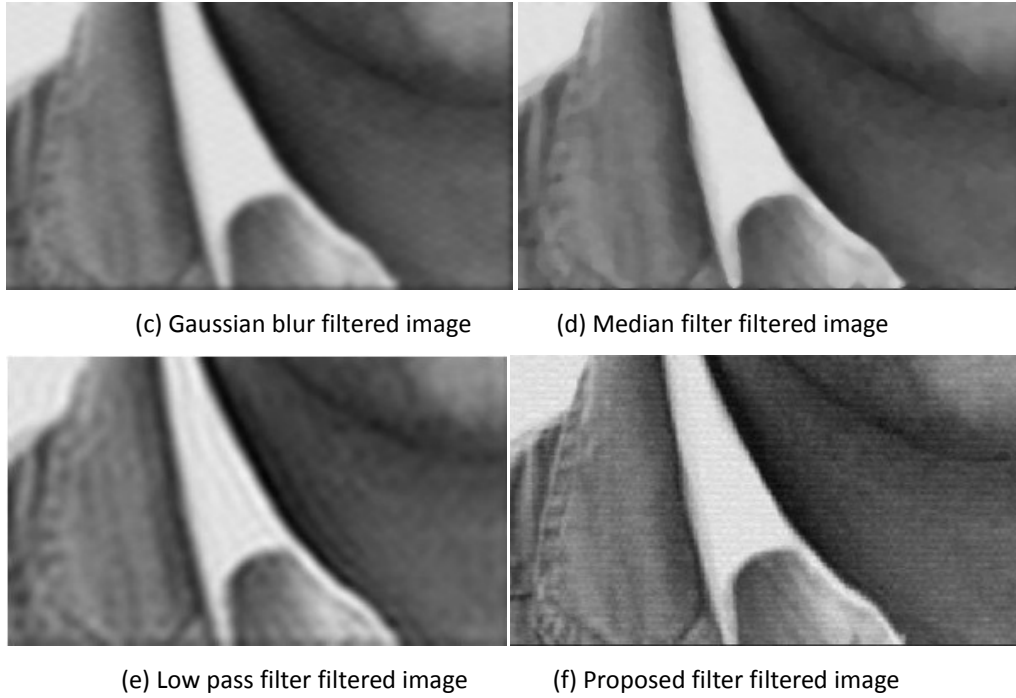


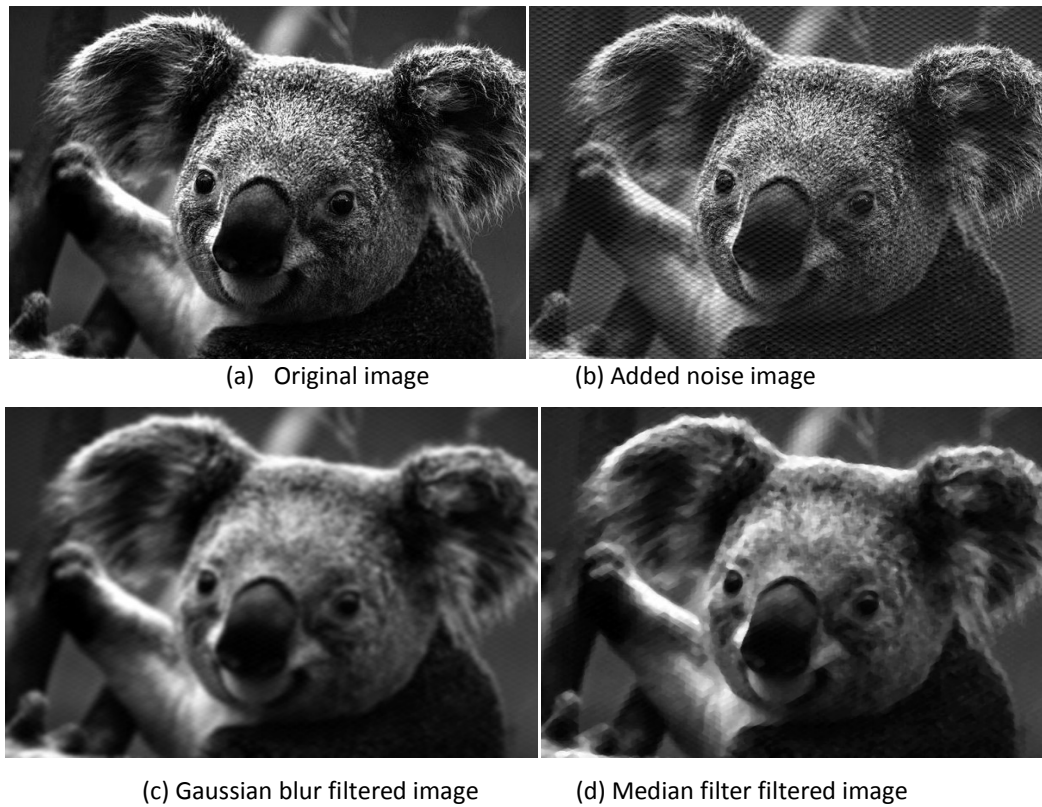
Table 6 - Mathematical Results of Image Contains Only Low Frequency Information

| Filtering methods | PSNR (db) |
|-------------------|-----------|
| Added noisy image | 22.73 |
| Median filter | 23.40 |
| Gaussian filter | 23.15 |
| Low pass filter | 23.58 |
| Proposed method | 23.55 |

As can be seen in Table 6 the results do not show the proposed solution providing significant improvement when applied an image which does not contain a great deal of high frequency details.

A second analysis is provided with an image that contains a good deal more high frequency details, as show in figure 30. Again, a considerable amount of textured noise was added to the image in Figure 30b, and figures 30 c – f display the effects of applying the filters, as labeled, to the noise added image (figure 30b). The numerical results of the application of these filters are shown in Table 7.

Figure 29 - PSNR testing image 2





(e) Low pass filter filtered image

(f) Proposed filter filtered image

Table 7- Mathematical Results of Image Contains High Frequency Information

| Filtering methods | PSNR (db) |
|-------------------|-----------|
| Added noisy image | 21.20 |
| Median filter | 19.20 |
| Gaussian filter | 19.04 |
| Low pass filter | 19.41 |
| Proposed method | 22.22 |

The numerical results clearly show that for image content of almost no high frequency components, all filters work fine. But for image content with a lot of high frequency components, the proposed filter shows significant improvement over the current solution.

Chapter 6

Concluding Remarks

6.1 Summary and Conclusion

This paper has proposed a practical solution for textured noise reduction problems based on the benefit of DCT's improving both computational speed and filtered image quality as compared to DFT. For the proposed solution DCT is chosen to be the fundamental frequency transform.

Software based on the algorithm introduced in Section 3.4 has been written to provide for textured noise detection and removal that has low programming complexity and provides exceptional results. This software implementation provides dramatic results and efficiency as compared to currently available noise reduction filters. This software can also be programmed into some of the currently available major graphic editing software such as Photoshop.

The results and comparison of the proposed solution with current solutions was discussed in Chapter 5. Both the visual and the numerical improvements document the validity of the solution put forward in this thesis.

6.2 Future Research

First, a good auto spikes detection method in the frequency domain matrix that could be implemented is a viable concept for future work. Auto spikes detection and removal would lead to an auto textured noise filter which can be

applied in large image data noise reductions, such as noisy videos.

A second area would be to apply the proposed application to Photoshop. This plug in is written in Microsoft Visual Studio C++. However to fully integrate it with Photoshop there is a fair amount of debugged that needs to be completed.

Another useful aspect to this work would be to add textured artifacts into images from the frequency domain. To control the shape and the intensity of the textured artifacts spikes in specific locations and of specific amplitudes in the frequency domain would be added.

REFERENCE

- [1] Stephen E. Reichenbach, Daniel E. Koehler, and Dennis W. Strelow Restoration of Reconstruction of AVHRR images IEEE TRANCTIONS ON GEOSCINCE AND REMOTE SENSING, VOL. 33, NO.4, JULY 1995
- [2] Paul J. Curran and Jennifer L. Dungan Estimation of Signal -to -Noise: A New Procedure Applied to AVIRIS Data IEEE TRANSATIONS ON GEOSCIENCE AND REMOTE SENSING, VOL. 27, NO. 5, SEPTEMBER 1989
- [3] C.R. de Souza Filho, S.A. Drury, A.M. Denniess, R.W.T. Carlton, and D.A. Rothery. Restoration of Corrupted Optical Fuyo -1 (JERS -1) Data Using Frequency Domain Techniques Photogrammetric Engineering & Remote Sensing, Vol. 62, No.9 September 1996, pp. 1037 -1047
- [4] Syed Ali Khayam The Discrete Cosine Transform (DCT): Theory and Application Department of Electrical & Computer Engineering Michigan State University March 2003
- [5] Hsien S. Hou A Fast Recursive Algorithm For Computing the Discrete Cosine Transform IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, VOL. ASSP -35, NO. 10, Octorber 1987
- [6] Wen -Hsiung Chen, C. Harrison Smith, And S. C. Fralick A Fast Computational Algorithm for the Discrete Cosine Transform IEEE TRANSACTIONS ON COMMUNICATIONS. VOL, COM -25, NO.9, SEPTEMBER 1977
- [7] Rui -Xiang Yin The Research for Fast Algorithms and Filter Implementation Structure of Discrete Cosine Transform South China University of technology, PHD thesis, 2000
- [8] Robert A. Schowengerdt Remote Sensing: Models and Methods for Image Processing 2nd ed. pp. 287 -356 Department of Electrical and Computer Engineering University of Arizona 1997
- [9] James W. Cooley and John W. Tukey An Algorithm for the Machine Calculation of complex Fourier Series Princeton University August 1964
- [10] Wen -Hsiung Chen, C. Harrison Smith, and S. C. Fralick A Fast computational Algorithm for the Discrete Cosine Transform IEEE Transactions on Communications, VOL. COM -25, No.9 September 1977

APPENDICE

A. Proof of **diving** DCT from DFT

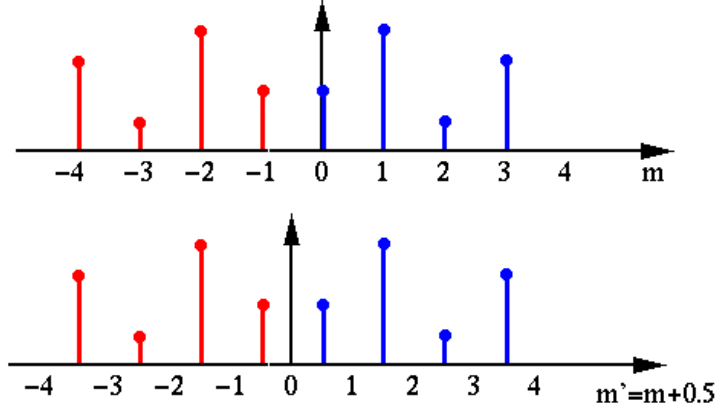
We first construct a new sequence of $2N$ points:

$$x'[m] \equiv \begin{cases} x[m] & (0 \leq m \leq N-1) \\ x[-m-1] & (-N \leq m \leq -1) \end{cases}$$

This $2N$ -point sequence $x'[m]$ is assumed to repeat itself outside the range $-N \leq m \leq N-1$, i.e., it is periodic with period $2N$, and it is even symmetric with respect to the point $m = -1/2$:

$$x'[m] = x'[-m-1] = x'[2N-m-1]$$

If we shift the signals $x'[m]$ to the right by $1/2$, or, equivalently, shift m to the left by $1/2$ by defining another index $m' = m + 1/2$, then $x'[m] = x'[m' - 1/2]$ is even symmetric with respect to $m' = 0$.



The DFT of this $2N$ -point even symmetric sequence can be found as:

$$X[n] = \frac{1}{\sqrt{2N}} \sum_{m'=-N+\frac{1}{2}}^{N-\frac{1}{2}} x[m' - \frac{1}{2}] e^{-\frac{j2\pi m'n}{2N}}$$

According to the Euler's formula $e^{ix} = \cos(x) + j\sin(x)$, $X[n]$ rewrite as:

$$X[n] = \frac{1}{\sqrt{2N}} \sum_{m'=-N+\frac{1}{2}}^{N-\frac{1}{2}} x'[m' - \frac{1}{2}] \cos\left(\frac{2\pi m'n}{2N}\right) - \frac{j}{\sqrt{2N}} \sum_{m'=-N+\frac{1}{2}}^{N-\frac{1}{2}} x'[m' - \frac{1}{2}] \sin\left(\frac{2\pi m'n}{2N}\right)$$

Note that the second summation above is zero. This is because $x'[m]$ is even and $\sin\left(\frac{(2m+1)n\pi}{2N}\right)$ is odd with respect to $m = -\frac{1}{2}$, all terms in the second summation are odd and the summation is zero (while all terms in the first summation are even).

$$X[n] = \frac{1}{\sqrt{2N}} \sum_{m'=-N+\frac{1}{2}}^{N-\frac{1}{2}} x' \left[m' - \frac{1}{2} \right] \cos \left(\frac{2\pi m' n}{2N} \right), \quad (n = 0, \dots, 2N-1)$$

It can also be seen that all $X[n]$ is real and even $X[n] = X[-n]$. Next, we replace m' by $m + \frac{1}{2}$ and get,

$$X[n] = \frac{1}{\sqrt{2N}} \sum_{m'=-N+\frac{1}{2}}^{N-\frac{1}{2}} x' \left[m' - \frac{1}{2} \right] \cos \left(\frac{2\pi m' n}{2N} \right) = X[n] = \frac{1}{\sqrt{2N}} \sum_{m=-N}^{N-1} x'[m] \cos \left(\frac{(2m+1)\pi n}{2N} \right)$$

From the definition of $x'[m]$ we notice that the summation of $x'[m] \cos \left(\frac{(2m+1)\pi n}{2N} \right)$ in the interval $0 \leq m \leq N-1$ is equal to the summation in the interval $-N \leq m \leq -1$, Next we can replace $x'[m]$ by $x[m]$ and have,

$$X[n] = \sqrt{\frac{2}{N}} \sum_{m=0}^{N-1} x[m] \cos \left(\frac{(2m+1)\pi n}{2N} \right), \quad (n = 0, \dots, 2N-1)$$

Note that since all terms in the summation are even, only the first half of the data points need to be used. Moreover, as cosine function is even, $X[n] = X[-n]$ is also even and periodic with period $2N$, we have $X[N+n] = X[N+n-2N] = X[-N+n] = X[N-n]$, indicating that the summation of point $X[N+n]$, ($n = 0, 1, \dots, N-1$) in the second half is the same as its corresponding points $X[N-n]$ in the first half, i.e., the second half is redundant and therefore can be dropped. Now we have the discrete cosine transform (DCT):

$$X[n] = \sqrt{\frac{2}{N}} \sum_{m=0}^{N-1} x[m] \cos \left(\frac{(2m+1)\pi n}{2N} \right) = \sum_{m=0}^{N-1} c[n, m] x[m], \quad (n = 0, \dots, N-1)$$

Here, $c[n, m]$ is the n th row and m th column of the DCT matrix:

$$c[n, m] = \sqrt{\frac{2}{N}} \cos \left(\frac{(2m+1)\pi n}{2N} \right), \quad (m, n = 0, 1, \dots, N-1)$$

All row vectors of this DCT matrix are orthogonal and normalized except the first one ($n=0$), to make DCT an orthonormal transform, $a[n]$ was defined:

$$a[n] = \begin{cases} \sqrt{\frac{1}{N}} & \text{for } n = 0 \\ \sqrt{\frac{2}{N}} & \text{for } n = 1, 2, \dots, N-1 \end{cases}$$

Now compare to the definition of DCT, it is exactly the same.

$$X[n] = a[n] \sum_{m=0}^{N-1} x[m] \cos \left(\frac{(2m+1)\pi n}{2N} \right) = \sum_{m=0}^{N-1} c[n, m] x[m], \quad (n = 0, \dots, N-1)$$

B. Application Code

1. Data matrix

```
m_dataMatrix=new int*[size];
    for(i=0;i<size;i++)
        m_dataMatrix[i]=new int[size];
```

2. Taper filter

```
    struct CGaussParaneter
    {
        int x0;
        int y0;
        double pr;
        COLORREF rgb;
    }

    double XYGaussian(int x,int y,int x0, int y0, double sigma_squ)
    {
return (1 -1/sqrt(2*3.1415926*sigma_squ)*exp( -((x -x0)*(x -x0)+(y -y0)*(y -
y0))/(2*sigma_squ)));
    }
    double XYGaussian2(int x,int y,int x0, int y0, double pr)
    {
        return (1 -exp( -((x -x0)*(x -x0)+(y -y0)*(y -y0))*pr/2));
    }
    float SetRadius(float pr)
    {
        float sigma_squ;
        sigma_squ=1/pr;
        return(3*sqrt(sigma_squ)+1);
    }

    void Filtering(double **dctMatrix, int size, int x0, int y0, float pr)//sigma_squ)
    {
//        double sigma, sigma_squ;
        int r,yup,ydown,xright,xleft;
        int i,j,x1,x2;
        int templnt;
        double tempDouble, rate;
```

```

r=(int)(SetRadius(pr)+1);

yup=y0 -r;
ydown=y0+r;
xright=x0 -r;
xleft=x0+r;

for(j=yup;j<=y0;j++)
{
    if(j>=0 && j<size)
    {
        templnt=sqrt(r*r -(j -y0)*(j -y0));
        x1=x0 -templnt;
        x2=x0+templnt;
        for(i=x1;i<=x2;i++)
        {
            if(i>=0 && i<size)
            {
                tempDouble=dctMatrix[i][j];
                //rate=XYGaussian(i,j,x0,y0,sigma_squ);
                rate=XYGaussian2(i,j,x0,y0,pr);
                dctMatrix[i][j]=tempDouble*rate;
            }
        }
    }
}
for(j=y0+1;j<=ydown;j++)
{
    if(j>=0 && j<size)
    {
        templnt=sqrt(r*r -(j -y0)*(j -y0));
        x1=x0 -templnt;
        x2=x0+templnt;
        for(i=x1;i<=x2;i++)
        {
            if(i>=0 && i<size)
            {
                tempDouble=dctMatrix[i][j];
                //rate=XYGaussian(i,j,x0,y0,sigma_squ);
                rate=XYGaussian2(i,j,x0,y0,pr);
                dctMatrix[i][j]=tempDouble*rate;
            }
        }
    }
}

```

```

    }
    }
}

```

3. Image block capture

```

void GetMatrix(CImage *grayImg, int x, int y, int size, int** dataMatrix)
{
    int i, j;
    int widthImg, heightImg;
    BYTE* cbuf;
    widthImg=grayImg ->GetWidth();
    heightImg=grayImg ->GetHeight();
    for(j=0; j<size; j++)
    {
        if(j+y<heightImg) cbuf=grayImg ->GetPixelAddress(x, y+j);
        for(i=0; i<size; i++)
        {
            if(i+x<widthImg && j+y<heightImg)
dataMatrix[i][j]=cbuf[i];
            else dataMatrix[i][j]=0;
        }
    }
}

```

4. Fast DCT

```

void initDCTParam(int deg) //pre -calculation of DCT matrix
{
    // deg is the power of DCT data
    if(bHasInit)
    {
        return;
    }
    int total, halftotal, i, group, endstart, factor;
    total = 1 << deg;
    if(C != NULL) delete []C;
    C = (double *)new double[total];
    halftotal = total >> 1;
    for(i=0; i < halftotal; i++)
        C[total - i - 1] = (double)(2*i+1);
    for(group=0; group < deg - 1; group++)
    {
        endstart = 1 << (deg - 1 - group);

```

```

        int len = endstart >> 1;
        factor=1 << (group+1);
        for(int j = 0;j < len; j++)
            C[endstart -j -1] = factor*C[total -j -1];
    }
    for(i=1; i < total; i++)
        C[i] = 2.0*cos(C[i]*PI/(total << 1)); ///C[0] is left empty
    bHasInit=true;
}

void dct_forward(double *f,int deg)// forward calculation
{
    // DCT data is saved in 'f'
    int i_deg, i_halfwing, total, wing, wings, winglen, halfwing;
    double temp1,temp2;
    total = 1 << deg;
    for(i_deg = 0; i_deg < deg; i_deg++)
    {
        wings = 1 << i_deg;
        winglen = total >> i_deg;
        halfwing = winglen >> 1;
        for(wing = 0; wing < wings; wing++)
        {
            for(i_halfwing = 0; i_halfwing < halfwing; i_halfwing++)
            {
                temp1 = f[wing*winglen+i_halfwing];
                temp2 = f[(wing+1)*winglen -1 -i_halfwing];
                if(wing%2)
                    swap(temp1,temp2); // exchange temp1 and temp2
                f[wing*winglen+i_halfwing] = temp1+temp2;
                f[(wing+1)*winglen -1 -i_halfwing] =
                    (temp1 -temp2)*C[winglen -1 -i_halfwing];
            }
        }
    }
}

void dct_backward(double *f,int deg) //backward calculation
{
    int total,i_deg,wing,wings,halfwing,winglen,i_halfwing,temp1,temp2;
    total = 1 << deg;
    for(i_deg = deg -1; i_deg >= 0; i_deg - -)
    {

```

```

wings = 1 << i_deg;
winglen = 1 << (deg - i_deg);
halfwing = winglen >> 1;
for(wing = 0; wing < wings; wing++)
{
    for(i_halfwing = 0; i_halfwing < halfwing; i_halfwing++)
    {
        //f[i_halfwing+wing*winglen] = f[i_halfwing+wing*winglen];
        if(i_halfwing == 0)
        {
f[halfwing+wing*winglen+i_halfwing]=0.5*f[halfwing+wing*winglen+i_halfwing];
        }
        else
        {
            temp1=bitrev(i_halfwing,deg - i_deg -1);// bitrev is inverse function
            temp2=bitrev(i_halfwing -1,deg - i_deg -1);// the first para modified
            f[halfwing+wing*winglen+temp1]=f[halfwing+wing*winglen+temp1] - f[halfwing+wing*winglen+temp2];
        }
    }
}
}

```

```

int bitrev(int bi,int deg)//Antition function
{
    int j = 1, temp = 0, degnum, halfnum;
    degnum = deg;
    //if(deg<0) return 0;
    if(deg == 0) return bi;
    halfnum = 1 << (deg -1);
    while(halfnum)
    {
        if(halfnum&bi)
            temp += j;
        j<<=1;
        halfnum >>= 1;
    }
    return temp;
}

```

```

void fdct_1D_no_param(double *f,int deg)//1 -D DCT

```

```

{
    initDCTParam(deg);
    dct_forward(f,deg);
    dct_backward(f,deg);
    fbitrev(f,deg);    // Antitone order
    f[0] = 1/(sqrt(2.0))*f[0];
}
void fdct_1D(double *f,int deg)
{
    fdct_1D_no_param(f,deg);
    int total = 1 << deg;
    double param = sqrt(2.0/total);
    for(int i = 0; i < total; i++)
        f[i] = param*f[i];
}

void fdct_2D(double *f,int deg_row,int deg_col)//2 -D DCT
{
    int rows,cols,i_row,i_col;
    double two_div_sqrtcolrow;
    rows=1 << deg_row;
    cols=1 << deg_col;
    init2D_Param(rows,cols);
    two_div_sqrtcolrow = 2.0/(sqrt(double(rows*cols)));
    for(i_row = 0; i_row < rows; i_row++)
    {
        fdct_1D_no_param(f+i_row*cols,deg_col);
    }
    for(i_col = 0; i_col < cols; i_col++)
    {
        for(i_row = 0; i_row < rows; i_row++)
        {
            temp_2D[i_row] = f[i_row*cols+i_col];
        }
        fdct_1D_no_param(temp_2D, deg_row);
        for(i_row = 0; i_row < rows; i_row++)
        {
            f[i_row*cols+i_col] = temp_2D[i_row]*two_div_sqrtcolrow;
        }
    }
    bHasInit = false; }

```