IMAGE CODING USING DERIVATIVE GAUSSIAN TRANSFORM WITH EMBEDDED ZEROTREES QUANTIZATION

A THESIS SUBMITTED TO THE GRADUATE DIVISION OF THE UNIVERSITY OF HAWAI'I IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

AUGUST 2004

By
Xiangang Li

Thesis Committee
Todd R. Reed, Chairperson
N. Thomas Gaarder
Anders Host-Madsen
Acknowledgement

I would like to express my appreciation to my advisor Dr. Todd R Reed for his guidance, patience, and encouragement throughout my research and study at Electrical Engineering department. I would also like to express my sincere gratitude to my committee members Dr. N. Thomas Gaarder and Dr. Anders Host-Madsen for their valuable advice and feedback.
Abstract

The Embedded Zerotree Wavelet (EZW) quantization method, which was originally designed for the wavelet transform, is a very effective technique for image compression. The Derivative Gaussian Transform (DGT), which represents image data as a linear combination of shifted Gaussian derivative, has shown its good behaviors for the image processing without the segmentation of the image into blocks. This research first applies the EZW method as part of an algorithm for image compression using the DGT. For employing the EZW quantizer, the DGT coefficients are reorganized into a hierarchical subband structure. In order to study the performance difference of the DGT, the Wavelet Transform and the Discrete Cosine Transform (DCT) by applying the Embedded Zerotree quantizer, several codecs are built. A quantization matrix and quality factor are used in the EZDGT codec to trade the image quality in terms of the Peak Signal-to-Noise Ratio (PSNR)) and the cost in terms of bits per pixel (bpp). Testing of these codecs has shown that the EZDGT outperforms the EZDCT and sometimes even better than the EZW.
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 DCT Quantization Table</td>
<td>27</td>
</tr>
<tr>
<td>5.1 Reconstructed 'Lena' image (512x512) qualities at given bpp</td>
<td>98</td>
</tr>
<tr>
<td>5.2 Reconstructed 'Barbara' image (512x512) qualities at given bpp</td>
<td>99</td>
</tr>
<tr>
<td>5.3 Reconstructed 'Girl' image (256x256) qualities at given bpp</td>
<td>99</td>
</tr>
<tr>
<td>5.4 Reconstructed 'Camman' image (256x256) qualities at given bpp</td>
<td>100</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1: A Typical transform Encoder/Decoder</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2.1: Gaussian basis function</td>
<td>9</td>
</tr>
<tr>
<td>Figure 2.2: Reconstructed DCT image at about 21 dB</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2.3: Reconstructed DGT image at about 21 dB</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2.4: DGT Basis functions in space</td>
<td>13</td>
</tr>
<tr>
<td>Figure 2.5: Basis functions in frequency</td>
<td>14</td>
</tr>
<tr>
<td>Figure 2.6: Girl image (256 × 256) and its DGT coefficients map</td>
<td>14</td>
</tr>
<tr>
<td>Figure 2.7: Lena image (512 × 512) and its DGT coefficients map</td>
<td>15</td>
</tr>
<tr>
<td>Figure 2.8: DGT transform coding example</td>
<td>16</td>
</tr>
<tr>
<td>Figure 2.9: DGT vs. Black-based DCT</td>
<td>16</td>
</tr>
<tr>
<td>Figure 2.10: Reconstructed Lena images (a) bpp=1.00; (b) bpp=0.5</td>
<td>17</td>
</tr>
<tr>
<td>Figure 3.1: Eight DCT cosine basis functions</td>
<td>21</td>
</tr>
<tr>
<td>Figure 3.2: 64 (8 × 8) DCT basis functions</td>
<td>22</td>
</tr>
<tr>
<td>Figure 3.3: JPEG Encoder Block Diagram</td>
<td>26</td>
</tr>
<tr>
<td>Figure 3.4: JPEG Decoder Block Diagram</td>
<td>26</td>
</tr>
<tr>
<td>Figure 4.1: A generic transform coder</td>
<td>28</td>
</tr>
<tr>
<td>Figure 4.2: (a). Original Lena Image (b). Reconstructed Lena with DC component, show-blocking artifacts</td>
<td>29</td>
</tr>
<tr>
<td>Figure 4.3: Touching wavelet spectra resulting from scaling of the mother wavelet in the time domain</td>
<td>31</td>
</tr>
<tr>
<td>Figure 4.4: Two-dimensional, four-band filter bank for subband image coding</td>
<td>32</td>
</tr>
<tr>
<td>Figure 4.5: Pyramidal Decomposition of an Image</td>
<td>33</td>
</tr>
</tbody>
</table>
Figure 4.6: Original 'girl' image and its wavelet transform coefficients map ........................................34
Figure 4.7: A three-level wavelet decomposition of the Lena image ........................................37
Figure 4.8: An example of Zerotree structure ........................................................................37
Figure 4.9: Block diagram of EZW algorithm ........................................................................40
Figure 4.10: The Morton scan order .......................................................................................41
Figure 4.11: Flow chart of the dominant pass ........................................................................42
Figure 4.12: Flow chart for Subordinate pass .........................................................................43
Figure 4.13: EZW decoder's dominant-pass flow chart .............................................................45
Figure 4.14: EZW decoder's subordinate-pass flow chart .............................................................46
Figure 5.1: 8×8 DCT coefficients block treated as 3-scale level tree .........................................51
Figure 5.2: The illustration of DCT coefficients reorganization ................................................51
Figure 5.3: Example of the rearrangement of four 4×4 DCT blocks to form a hierarchical subband structure .................................................................52
Figure 5.4: Reorganized DCT 10-subband image ..................................................................53
Figure 5.5: The Wavelet coefficients map of Girl image (3-scale) ............................................54
Figure 5.6: Block Diagram of EZDCT Encoder ....................................................................55
Figure 5.7: Lena image and its DCT coefficients map ...............................................................56
Figure 5.8: Bpp vs. PSNR Curve of reconstructed image Lena ..................................................60
Figure 5.9: Reorganized DCT coefficients (wavelet-like) tree structure ..................................60
Figure 5.10: Reconstructed image at bpp=0.0176, PSNR=7.61 dB, threshold=4096(First pass) .61
Figure 5.11: Reconstructed DCT image at bpp=0.0669; PSNR=17.41 dB, threshold=2048...........61
Figure 5.12: Reconstructed image at bpp=0.1122; PSNR=24.29 dB, threshold=1024.................62
Figure 5.13: Reconstructed image at bpp=0.1662; PSNR=26.86 dB, threshold=512..................62
Figure 5.14: Reconstructed image at bpp=0.2606; PSNR=29.36dB, threshold=256...................63
Figure 5.15: Reconstructed image at bpp=0.4220; PSNR=31.78 dB, threshold=128..................63
Figure 5.16: Reconstructed image at bpp=0.6785; PSNR=33.99dB, threshold=64
Figure 5.17: Reconstructed image at bpp=1.0733; PSNR=35.96 dB, threshold=32
Figure 5.18: Reconstructed image at bpp=1.6841; PSNR=38.00 dB, threshold=16
Figure 5.19: Reconstructed image at bpp=2.6030; PSNR=40.23 dB, threshold=8
Figure 5.20: Reconstructed image at bpp=3.9384; PSNR=43.23 dB, threshold=4
Figure 5.21: Reconstructed image at bpp=5.8860; PSNR=48.91 dB, threshold=2
Figure 5.22: Reconstructed DCT image at bpp=7.3645, PSNR=55.28 dB, threshold=1
Figure 5.23: Calculating the field of view of an arbitrary display device
Figure 5.24: Contrast sensitivity as predicted by Mannos and Sakrison model
Figure 5.25: Original DGT Lena coefficients map
Figure 5.26: Reorganized DCT-liked DGT coefficients map
Figure 5.27: Reorganized coefficients map after applying qmat & quality factor
Figure 5.28: Wavelet-like DGT coefficients map
Figure 5.29: The results of different algorithms bpp vs. PSNR chart
Figure 5.30: Reconstructed Lena image bpp vs. PSNR at low-bit-rates
Figure 5.31: Reconstructed DGT image at bpp=0.0055, PSNR=6.24dB, TH=2048
Figure 5.32: Reconstructed DGT image at bpp=0.0512, PSNR=15.02dB, TH=1024
Figure 5.33: Reconstructed DGT image at bpp=0.0989, PSNR=23.27dB, TH=512
Figure 5.34: Reconstructed DGT image at bpp=0.1731, PSNR=27.72dB, TH=256
Figure 5.35: Reconstructed DGT image at bpp=0.3119, PSNR=31.25dB, TH=128
Figure 5.36: Reconstructed DGT image at bpp=0.5338, PSNR=34.33dB, TH=64
Figure 5.37: Reconstructed DGT image at bpp=0.9742, PSNR=37.03dB, TH=32
Figure 5.38: Reconstructed DGT image at bpp=1.7699, PSNR=39.79dB, TH=16
Figure 5.39: Reconstructed DGT image at bpp=1.00, PSNR=37.03dB
Figure 5.40: Reconstructed DCT image at bpp=1.00, PSNR=35.01dB
Figure 5.41: Reconstructed EZW image at bpp=1.00, PSNR=36.33dB .......................... 81
Figure 5.42: Reconstructed DGT image at bpp=0.5, PSNR=32.75dB ............................. 82
Figure 5.43: Reconstructed DCT image at bpp=0.5, PSNR=31.75dB ............................. 82
Figure 5.44: EZW lean reconstructed image at bpp=0.5, PSNR=32.88dB ....................... 83
Figure 5.45: Reconstructed EZDGT image at bpp=0.25, PSNR=28.82dB ....................... 84
Figure 5.46: Reconstructed EZDCT image at bpp=0.25, PSNR=28.14dB ....................... 84
Figure 5.47: Reconstructed EZW image at bpp=0.25, PSNR=29.71dB .......................... 85
Figure 5.48: Reconstructed EZDGT image at bpp=0.125, PSNR=23.98dB ....................... 85
Figure 5.49: Reconstructed DCT image at bpp=0.125, PSNR=24.33dB ......................... 86
Figure 5.50: Reconstructed EZW image at bpp=0.125, PSNR=25.03dB ......................... 86
Figure 5.51: Reconstructed EZDGT image at bpp=0.0625, PSNR=16.76dB ..................... 87
Figure 5.52: Reconstructed EZDCT image at bpp=0.0625, PSNR=14.49dB ..................... 87
Figure 5.53: Reconstructed EZW image at bpp=0.0625, PSNR=14.49dB ....................... 88
Figure 5.54: Bpp vs. PSNR Curve of the reconstructed Lena images ............................ 89
Figure 5.55: Reconstructed EZDGT image at bpp=0.50, PSNR=29.06dB ......................... 90
Figure 5.56: Reconstructed EZDCT image at bpp=0.50, PSNR=26.96dB ......................... 90
Figure 5.57: Reconstructed EZW image at bpp=0.50, PSNR=26.56dB .......................... 91
Figure 5.58: Reconstructed EZDGT image at bpp=0.25, PSNR=25.64dB ....................... 91
Figure 5.59: Reconstructed EZDCT image at bpp=0.25, PSNR=23.69dB ....................... 92
Figure 5.60: Reconstructed EZW image at bpp=0.2500, PSNR=23.52dB ....................... 92
Figure 5.61: Reconstructed EZDGT image at bpp=0.125, PSNR=21.83dB ....................... 93
Figure 5.62: Reconstructed EZDCT image at bpp=0.125, PSNR=21.49dB ....................... 93
Figure 5.63: Reconstructed EZW image at bpp=0.125/PSNR=21.89dB ......................... 94
Figure 5.64: Reconstructed EZDGT image at bpp=0.0625/PSNR=14.81dB ..................... 94
Figure 5.65: Reconstructed EZDCT image at bpp=0.0625, PSNR=15.83dB ..................... 95
Figure 5.66: Reconstructed EZW image at bpp=0.0625, PSNR=13.12dB

Figure 5.67: Bpp vs. PSNR curves of reconstructed images using different transform.
List of Abbreviations

DGT: Derivative Gaussian Transform.
DCT: Discrete Cosine Transform.
DFT: Discrete Fourier Transform.
EZDGT: Embedded Zerotree Derivative Gaussian Transform.
EZDCT: Embedded Zerotree Discrete Cosine Transform.
EZW: Embedded Zerotree Wavelet Transform.
HVS: Human Visual System.
JPEG: Joint Photographic Experts Group.
MMSE: Minimum Mean Square Error.
PSNR: Peak Signal-to-Noise Ratio.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgement</td>
<td>iii</td>
</tr>
<tr>
<td>Abstract</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td>List of Abbreviations</td>
<td>xi</td>
</tr>
<tr>
<td>Chapter 1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Chapter 2. Derivative Gaussian Transform (DGT)</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Discrete 1 D DGT</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Non-orthogonality</td>
<td>6</td>
</tr>
<tr>
<td>2.3 MMSE Decomposition</td>
<td>7</td>
</tr>
<tr>
<td>2.4 Separable 2D Transform</td>
<td>8</td>
</tr>
<tr>
<td>2.5 Basis Selection - Spatial</td>
<td>9</td>
</tr>
<tr>
<td>2.6 Basis Selection - Derivative Order</td>
<td>10</td>
</tr>
<tr>
<td>2.7 Spectral Distribution</td>
<td>10</td>
</tr>
<tr>
<td>2.8 Transform Coding using the DGT</td>
<td>14</td>
</tr>
<tr>
<td>2.9 Related Applications</td>
<td>18</td>
</tr>
<tr>
<td>Chapter 3. JPEG: DCT-Based Image Coding</td>
<td>19</td>
</tr>
<tr>
<td>3.1 One-dimensional DCT</td>
<td>19</td>
</tr>
<tr>
<td>3.2 Two-dimensional DCT</td>
<td>22</td>
</tr>
<tr>
<td>3.3 DCT quantization</td>
<td>24</td>
</tr>
<tr>
<td>3.4 DCT blocks</td>
<td>25</td>
</tr>
<tr>
<td>3.5 DCT-based Coding Models</td>
<td>26</td>
</tr>
<tr>
<td>Chapter 4. Wavelet Transform and Embedded Zerotree coding</td>
<td>28</td>
</tr>
<tr>
<td>4.1 Wavelet transform</td>
<td>28</td>
</tr>
</tbody>
</table>

xii
Chapter 1. Introduction

Image coding has played an important role in modern society, for example, people often use the Internet to send pictures and use cell phones to transmit image information. Uncompressed multimedia such as graphics, audio and video data requires considerable storage capacity and transmission bandwidth. Although new technologies are progressing rapidly in mass-storage density, processor speeds, and digital communication system performance, the demand for data storage capacity and data-transmission bandwidth continues to be far beyond the capabilities of available technologies. As a result, image compression is very important and applicable in many fields. There exist a lot of methods to the processing of image data for storage, transformation and representation; the most widely used is JPEG.

The Joint Photographic Experts Group or JPEG [15] standard has been established by ISO (International Standards Organization) and IEC (International Electro-Technical Commission) for still image compression. The performance of these coders generally degrades at low bit-rates mainly because of the underlying block-based Discrete Cosine Transform (DCT) scheme. Presently, the wavelet transform based coding [10] provides substantial improvements in picture quality at higher compression ratio or low bit rate.

A common characteristic of most images is that the neighboring pixels are correlated and therefore contain redundant information [15]. There is suggested that natural images have the common characteristic, in which most of the energy is concentrated in low frequency information, and of the remaining high frequency components of the image, most energy is spatially concentrated around the edges [4].

Image compression is used to compress the visual images. In order to be able to store or transmit image date in an efficient form, the redundancy of the image data should be reduced. Image compression is the process of encoding image data so that it takes less storage space or less
transmission time or bandwidth than it would if it were not compressed. The possibility of it lies in the common characteristics of most natural images, and the bandlimited nature of the human visual system.

Image compression can be lossy or lossless. Lossless data compression is a data compression algorithm which allows the original data to be reconstructed exactly from the compressed data. Artificial images, such as technical drawings, icons are preferred to use lossless compression. Lossy data compression is a method where compressing a file and then decompressing it retrieves a file that may well be different to the original, but is "close enough" to be useful somehow. Lossy methods are suitable for natural still images such as photos.

There are two basic image compression components: redundancy reduction and irrelevancy reduction. Redundancy reduction removes the duplication from the signal source (image/video). Irrelevancy reduction omits parts of the signal that will not be noticed by the signal receiver, namely the Human Visual System (HVS). In general, three types of redundancy can be identified:

-----**Spatial Redundancy** or correlation between neighboring pixel values.

-----**Spectral Redundancy** or correlation between different color planes or spectral bands.

-----**Temporal Redundancy** or correlation between adjacent frames in a sequence of images (in video applications).

In image coding, most of the image area represents spatial “trends”, or the areas of high statistical spatial correlation [2]. Transforms are at the heart of the compression engine; transform coding has been widely used in many practical image/video compression, because the quantization coefficients after linear transform can get better compression results than that of direct coding the image intensity in the spatial domain. The goal of the transformation is to decrease the correlation of the coefficients. In an ideal situation, a good transform will remove all dependencies between samples, and concentrates the important signal in a few data values.
In general, the purpose of image compression is to reduce the number of bits needed to represent a digital image by removing the spatial and spectral redundancies as much as possible. From a mathematic viewpoint, this amounts to transforming a 2-D pixel array into a statistically uncorrelated data set [17]. The Figure 1.1 is a typical transform coding system, including encoder and decoder. Both in encoder and decoder, there are three closely connected components. In encoding, they consist of applying a linear transform to decorrelate the input image data, quantizing the resulting transform coefficients, and entropy coding the quantized symbols. The compressed image usually called a bit stream. In decoding, the process is reversed; the compressed image is decompressed to reconstruct the original image or an approximation of it.

(a). The key components of a transform encoder are **Transform**. A variety of linear discrete 2-D transforms have been developed which include Discrete Fourier Transform (DFT), Discrete Cosine Transform (DCT), Discrete Wavelet Transform (DWT), Derivative Gaussian Transform (DGT) [1] and many more; each with its own advantages and disadvantages. The choice of a particular transform in a given application depends on how much reconstruction error that can be tolerated and the computational resources available [17].

(b) **Quantizer**, A quantizer simply reduces the number of bits needed to store the transformed coefficients by reducing the precision of those values. Almost all information loss occurs here...
because this is a lossy process. Quantization can be performed on each individual coefficient, which is known as Scalar Quantization (SQ). Quantization can also be performed on a group of coefficients together, and this is known as Vector Quantization (VQ) [15]. Image compression is achieved by quantizing the transformed coefficients.

(c) Entropy Encoder, An entropy encoder further compresses the quantized symbol stream losslessly to give better overall compression. Assuming that the resulting symbols are all independent, the entropy of the symbols \( H_s \) can be expressed as

\[
H_s = - \sum_{i=1}^{n} p_i \log_2 p_i
\]  

(1.1)

where "−" means minus, \( p_i \) is the probability of occurrence of a quantized transform coefficient, and \( H_s \) is the entropy of the set of quantized coefficients. It uses a model to accurately determine the probabilities for each quantized value and produces an appropriate code based on these probabilities, so that the resultant output bit stream will be smaller than that of the input stream.

The Huffman encoder and the Arithmetic encoder are the most commonly used in practice. A properly designed quantizer and an entropy encoder are absolutely necessary along with signal/image transformation to get the best possible compression.

The Embedded Zerotree Quantization (EZQ) method, which was originally designed for the wavelet transform, is a very effective technique for image compression. This paper will apply EZQ method as part of an algorithm for compression using the Derivative Gaussian transform (DGT) [1] that was initially introduced by Mr. Jeffrey Adam Bloom and Dr. Todd R. Reed. A quantization matrix and a quality factor are used in the codec to balance image quality (e.g., in terms of the Peak Signal-to-Noise Ratio (PSNR)) and cost in terms of bits per pixel (bpp).
Chapter 2. Derivative Gaussian Transform (DGT)

According to the most accepted definition, a transform has the following two important properties:

1. A transform is invertible, and
2. A transform does not result in an expansion of data.

The traditional presentation of a linear transform, which involves the representation of a signal as a weighted sum of the basis functions as follows:

\[ x(n) = \sum_{m=0}^{M-1} C_m g_m(n) \]  

(2.1)

Where the \( g_m(n) \) are the basis functions, and \( C_m \) are the corresponding coefficients.

How to calculate the decomposition coefficients is a big challenge for all kinds of transforms. If all of the basis functions in the set are orthogonal, the coefficients can be achieved via one of many inner product-based techniques. In this case, the corresponding coefficients for the linear combination reconstruction will be provided by the inner product of each basis function and the original signal. This is possible because the inner product of any two-basis functions will be zero because of their mutual orthogonality.

The Derivative Gaussian Transform (DGT) is a non-orthogonal linear transform, which was initially introduced by Jeffrey Adam Bloom in his PHD thesis "The Derivative of Gaussian Transform."[1] in 1999. He introduced a new image transform that decomposes an image using a set of Gaussian functions and their derivatives [5]. The DGT, which presents image data as a linear combination of shifted Gaussian derivatives, has the potential to allow local processing or analysis without the segmentation of the image into blocks. As with all linear transforms, the input is represented as a linear combination of basis function. The transform is characterized by the basis functions and the method of determining the coefficients [1]. The following material will briefly describes the DGT [17].
2.1 Discrete 1 D DGT

In one dimension, an arbitrary discrete signal, \( f(n) \) can be expressed as a linear combination of basis functions \( g_m(n) \),

\[
f(n) = \sum_{m=0}^{M-1} F_m g_m(n); \quad 0 \leq n < N \tag{2.2}
\]

Where \( f(n) \) is an \( N \)-point signal being represented by \( M \) basis functions. \( F_m \) are the coefficients. The index \( m \) covers all basis functions, and is a combination of two indices tracking location and derivative order (since the decomposition is a local decomposition).

The continuous basis functions are defined as the Gaussian and its derivatives with respect to \( x \).

\[
g_0(x) = e^{-\frac{x^2}{2\sigma^2}} \tag{2.3}
\]

\[
g_n(x) = \frac{d^n}{dx^n} g_0(x) = P_{n,\sigma}(x)g_0(x) \tag{2.4}
\]

The \( n^{th} \) derivative of a Gaussian can be written as the product of a polynomial \( P_{n,\sigma}(x) \) (Called Hermite polynomial) and the original Gaussian \( g_0(x) \). Since these functions are local in \( x \), they are translated to various locations in order to cover the \( x \)-axis.

In the frequency (Fourier) domain, the functions defined are given by:

\[
G_0(\omega) = \sigma e^{-\frac{\omega^2\sigma^2}{2}} \tag{2.5}
\]

\[
G_n(\omega) = (j\omega)^n G_0(\omega) \tag{2.6}
\]

2.2 Non-orthogonality:

When the basis set \( (g_m(x)) \) is mutually orthogonal, it can be shown that the coefficients of the representation \( (F_m) \) can be found with inner product techniques.
The subset of Gaussian derivatives at one location is mutually orthogonal. However, the full set of basis functions, which includes Gaussian derivatives at multiple locations, is non-orthogonal. The coefficients of this transform cannot be found by computing the inner product of the signal of interest and the basis functions.

2.3 MMSE Decomposition

We can write the expression (2.1) for \( f(n) \) in matrix form where we consider \( f \) to be a column vector of length \( N \), \( \bar{F} \) a column vector of length \( M \), and \( G \) an \( N \times M \) matrix with each of the \( M \) basis functions making up one column of \( G \).

\[
f = G \bar{F}
\]  
(2.7)

In the special case where \( M=N \) (same number of basis functions as original data points), \( G \) will be square. If we also require the basis functions (columns of \( G \)) be linearly independent, then \( G \) will be non-singular and the coefficients, \( F_m \), are easily found:

\[
\bar{F} = G^{-1} \bar{f}
\]  
(2.8)

The basis is then complete. This basis can represent any such signal \( f(n) \) or in other words, a set of coefficients exists for all \( N \)-point signals.

In general we can find the best estimate of \( \bar{F} \) with respect to the squared error, \( Error \),

\[
Error = G^{-1} \bar{f} - \bar{F}
\]  
(2.9)

\[
(Error)^2 = (G^{-1} \bar{f} - \bar{F})^T (G^{-1} \bar{f} - \bar{F})
\]  
(2.10)

Let \( \frac{d(Error)^2}{d \bar{F}} = 0 \), we can get the best estimation of \( \bar{F} \)

\[
\bar{F}_{MSE} = (G^T G)^{-1} G^T \bar{f} = A^T \bar{f}
\]  
(2.11)

Where \( A^T = (G^T G)^{-1} G^T \)  
(2.12)
\( A^T \) is the pseudo inverse of \( G \). When \( G \) is a square matrix, this \( A^T \) will equal \( G^{-1} \) and the estimate of \( \bar{F} \) is exact. Notice that both \( G \) and \( A^T \) are independent of the actual signal and can be computed off-line. The processes of analysis and synthesis (inverse and forward transforms) are simply matrix multiplications.

2.4 Separable 2D Transform

The 2-D synthesis inverse transform is given by the equation

\[
f(n_1, n_2) = \sum_{m_1} \sum_{m_2} F(m_1, m_2) g_{m_1m_2}(n_1, n_2)
\]

(2.13)

Where the two indices, \( m_1 \) and \( m_2 \), track both spatial location and derivative order in their respective directions.

A common approach to the multidimensional case is to define the transform to be separable, so that the basis functions can be written

\[
g_{m_1m_2}(n_1, n_2) = g_{m_1}(n_1) g_{m_2}(n_2)
\]

(2.14)

Then consider a separable two-dimensional transform by representing the 2-D basis function as the product of two 1-D basis functions.

\[
f(n_1, n_2) = \sum_{m_1=0}^{M_1-1} \sum_{m_2=0}^{M_2-1} F_{m_1m_2} g_{m_1}(n_1) g_{m_2}(n_2)
\]

(2.15)

\[0 \leq n_1 \leq N_1\]
\[0 \leq n_2 \leq N_2\]

Here \( f(n_1, n_2) \) is a 2-D signal, \( F_{m_1m_2} \) is a 2-D array of coefficients, and the \( g_m \) are the same basis functions as in the 1-D case.

This expression for representing a signal, \( f(n_1, n_2) \), as a linear combination of separable basis functions can be written in matrix form and the MMSE estimate of the coefficient array can be found.
\[ f = GFG^T \quad (2.16) \]

\[ F = (G^T G)^{-1} G^T \bar{f} G (G^T G)^{-1} = A^T \bar{f} A \quad (2.17) \]

Where \( A \) is as defined in equation 2.12, then equation 2.16 and 2.17 form the DGT transform pair. When the number of basis functions is the same as the number of original data points, the estimation of \( F \) is exact and the basis is complete. The forward and inverse transforms are simply matrix multiplications.

2.5 Basis Selection - Spatial

Uniform Spacing: In the spatial domain, the basis functions will be repeated at various locations. The simplest approach is to space the functions uniformly across the \( x \)-axis \( D \) pixels (or samples) apart.

Uniform \( \sigma \): Along with uniform spacing, it is most straightforward to require the variance of the Gaussian to be invariant to the spatial location of the basis functions.

The actual values of \( D \) and \( \sigma \) are chosen for a particular implementation. For this work, we chose that \( D \) is set to 8 pixels and \( \sigma \) to 3.4 pixels as shown below.

\[ \text{Figure 2.1: Gaussian basis function} \]
2.6 Basis Selection - Derivative Order

In order to choose the derivative orders to be used in this basis set, we first note that all Gaussian derivative spectra are bandpass with the sole exception of the original Gaussian, which is low-pass.

We can define the spectral location as the peak of the curve in the frequency domain \{formula 2.28 in [1]\}.

\[
\omega_n^2 = \frac{n}{\sigma^2}
\]

\[
\omega_n = \pm \frac{\sqrt{n}}{\sigma}
\]

\[
n = \sigma^2 \omega_n^2
\]

Here we can see that the spectral location is dependent on both the Gaussian variance and the derivative order. Solving for \(n\) yields a real number representing the derivative order necessary to locate a bandpass peak at a desired frequency \(\omega_n\). This value is rounded to obtain an integral derivative order.

2.7 Spectral Distribution

The simplest spectral distribution is to require that spectra of the basis functions at each spectral location be distributed uniformly between \(-\pi\) and \(\pi\).

We will denote by \(m_i\) that is the number of basis functions at each location. In the spatial domain, the basis functions are placed every \(D\) pixels. Clearly, with this sub-sampling of the N point signal by \(D\), we will require \(m_i = D = 8\) basis functions at each location in order for the basis to be complete.

Dr. Jeffrey Adam Bloom's experiment, shows that the quantization errors introduced by the DGT will be less offensive to the human eyes than those introduced by a block-based one such as block DCT. A good choice of the invariant variance \(\sigma\) for all basis functions is 3.4.
Since the DGT does not have a true DC basis function, the lowest frequency basis function is the Gaussian, which is not sufficient to represent a flat signal. In order to represent such a flat signal, contribution from other basis functions is employed. This process is called DC leakage, because the energy at DC appears to leak into higher frequency coefficients and into adjacent locations. Due to this kind of DC leakage, the so-called Screen Door Artifacts appear in the reconstructed image. It manifests as a regular pattern of Gaussian-like peaks in the intensity of the reconstructed image. A solution that was suggested for this problem is modifying the basis by increasing the variance of the lowest order basis function (which is Gaussian). In this work, the recommended variance of 4.8 was used. The other seven basis functions are the derivatives of a Gaussian with standard deviation 3.45. Figure 2.2 and 2.3 show that reconstructed image using DGT and DCT with the same quantization method. We can see that in flat areas, the screen door artifact is still visible in Figure 2.3, but the flat regions are much smoother in Figure 2.3 than those of Figure 2.2.

The following expression, with \( m \) counting 0 to 7, finds the 8 uniformly distributed and centered frequencies between 0 and \( \pi \) and the expression for \( n \) from the previous page is used to find the derivative orders necessary to place spectral peaks at these frequencies.

\[
\omega_n = \frac{\pi}{m_i} \left(n + \frac{1}{2}\right) \tag{2.21}
\]

\[
n = \{0, 3, 10, 21, 36, 53, 74, 99\}
\]

The resulting basis functions are shown in Figure 2.4, with their spectra shown in Figure 2.5.
Figure 2.2: Reconstructed DCT image at about 21 dB

Figure 2.3: Reconstructed DGT image at about 21 dB
Figure 2.4: DGT Basis functions in space
2.8 Transform Coding using the DGT

The following experiment is intended to evaluate the DGT for potential use in transform image coding. The original images are 8-bits, grayscale, one size is 256×256, another one is 512×512; their DGT representation (Coefficients matrix) on the right. The basis set’s variances are 3.45 except the lowest Gaussian basis function which is 4.8.
We can see that the DGT coefficients representation is four-dimensional. Each coefficient is indexed by 2 spatial indices and 2 derivative order indices [17]. Subsampling of the 256×256 size image by 8 results 32×32 different spatial locations; whereas subsampling a 512×512 size image by 8 gets in 64×64 different spatial locations. All the DGT coefficients have been collected by the same derivative order. For each derivative order (total 64 combinations, 8 in the \( x \) and 8 in the \( y \) dimensions), the coefficients are collected at the 32×32 locations for 256×256 size images, at the 64×64 locations for 512×512 size images. The upper left block represents the coefficients of the normal Gaussian basis function in both dimensions. The next block to the right represents the coefficients of the basis functions which are the third derivative in the \( x \) dimension and Gaussian in the \( y \) dimension and so on.

The DGT coefficients are real and thus any finite precision representation will necessarily introduce quantization error in the reconstruction. Obviously we can see that at the lower right of the coefficient map, the most of the coefficients are very close to zero. Because of this characteristic, we may apply the uniform quantization method to the DGT coefficients.

We next use the DGT to test the quantization artifacts and compare with the block-based DCT. The transform coefficients of an image are quantized and then used to reconstruct an
approximation to the original image. Different transforms will introduce different types of artifacts into the reconstruction.

Figure 2.8: DGT transform coding example

The JPEG [10] image compression standard is a transform coding technique, which uses a block-based DCT. This experiment will apply the same coding technique (quantization of transform coefficients) to a test image using the DGT and a block-based DCT (JPEG). The DCT is implemented with 8x8 blocks to match the 8 pixel centers of the DGT basis functions.

Figure 2.9: DGT vs. Block-based DCT

The image on the left is the DGT based reconstruction while that on the right is the block-based DCT reconstruction. Both reconstructed images have approximately 2.0 bit per pixel, only the transform distribution differently. Errors introduced in the DCT domain are spread uniformly
over an 8x8 block in the reconstruction. We can see that when adjacent blocks have different quantization errors, the block boundaries become visible in DCT. Since the DGT does not employ blocking, no blocking artifacts are introduced.

![Reconstructed DGT and DCT image at bpp=1.0]

![Reconstructed DGT and DCT image at bpp=0.5]

**Figure 2.10: Reconstructed Lena images (a) bpp=1.00; (b) bpp=0.5**

Figure 2.10 shows the facial details of a girl's reconstructed images at 1.0 and 0.5 bits per pixel. We can see how the artifacts affect the reconstructed image quality. The DCT reconstructed images exhibit obvious blocking. The PSNR of the DGT images is also better than with the DCT. Also the quantization errors instructed by the DGT are much less offensive to human eyes. In the original sized reconstructions (top), we do see an artifact of the DGT, which does not present in the block-based DCT version. The artifact has been described as the screen door effect since the smooth portions of the reconstruction appear to be viewed through a screen door. In the
implementation presented, this effect is controlled by the ratio of the spacing between basis functions (D) and the standard deviation of the Gaussian (σ).

2.9 Related Applications

In [9], the authors claimed that the DGT could be used in a transform-coding scheme for still image compression, and in their experiments, good compression results vs. quality tradeoff was achieved. The authors predicted that with the further refinement of the quantization mask and the spectral distribution of the basis function, better results were expected to be accomplished. P. Morgan, L.Watson and R.Young demonstrated a similar application of Gaussian derivatives as an image representation scheme in their paper “A Gaussian Derivative Based Version of JPEG for Image Compression and Decomposition,” [7]. Depending on the computer arithmetic hardware used, this scheme might yield a compression/decompression technique twice as fast as the DCT and have equal quality. The author in [1] also claimed that the DGT held promise in many other image processing potential applications such as watermarking, texture analysis, segmentation, and pattern recognition etc.
Chapter 3. JPEG: DCT-Based Image Coding

JPEG is an acronym for "Joint Photographic Experts Group", the committee's original name that wrote the standard. JPEG is designed to compress either full-color or gray-scale images. It has shown good performance on photographs, naturalistic artwork, and similar images; but when JPEG is applied to lettering, simple cartoons and line drawings, the results are not as good.

JPEG is a lossy compression method, meaning the reconstructed image is not exactly the same as the original image. It’s well known that small color changes are perceived less accurately than small changes in brightness, so JPEG is focused on compressing images that will be good for visual observation.

The discrete cosine transform (DCT) is an important transform in 2-D signal processing. This method is close to optimal in terms of its energy compaction capabilities and can be computed via a fast algorithm. The DCT is currently used in two international image/video compression standards, JPEG and Motion Picture Experts Group (MPEG).

3.1 One-dimensional DCT

Based on William B. Pennebaker and Joan L. Mitchell’s book, "JPEG still image data compression standard"[19], the DCT can be regarded as a discrete-time version of the Fourier-Cosine series. It is a close relative of the Discrete Fourier Transform (DFT), a technique for converting a signal into elementary frequency components. A set of eight different cosine waveforms (Which are called cosine basis functions) of uniform amplitude are used in the DCT, each sampled at eight points (Figure 3.1). The top-left waveform is simply a constant, whereas the other seven waveforms show an alternating behavior at progressively high frequencies.

These waveform are orthogonal, thus are independent. The coefficient that scale the constant basis function is called the DC coefficient, the others are called AC coefficients. The process of decomposing a set of sample into a scaled set of cosine basis functions is called the forward discrete cosine transform (FDCT). The process of reconstructing the set of samples from the
scaled set of cosine basis functions is called inverse discrete cosine transform (IDCT). If the sample sequence is longer than eight samples, it can be divided into eight-sample groups and the DCT can be computed independently for each group.

The 1-D DCT is defined as follow.

FDCT:

\[
S(u) = \frac{C(u)}{2} \sum_{x=0}^{7} s(x) \cos\left(\frac{(2x+1)u\pi}{16}\right)
\]  
(3.1)

IDCT:

\[
s(x) = \sum_{u=0}^{7} \frac{C(u)}{2} S(u) \cos\left(\frac{(2x+1)u\pi}{16}\right)
\]  
(3.2)

Where \(C(u) = 1/\sqrt{2}\) for \(u=0\)

\(C(u) = 0\) for \(u>0\)

\(s(x) = 1\)-D sample value

\(S(u) = 1\)-D DCT coefficient

This transform is orthonormal, meaning that the following relationship involving the sum exists:

\[
\sum_{x=0}^{7} \frac{C(u)}{2} \cos\left(\frac{(2x+1)u\pi}{16}\right) \frac{C(u')}{2} \cos\left(\frac{(2x+1)u'\pi}{16}\right) = \delta(u,u')
\]  
(3.3)

Where

\(\delta(u,u') = 1\) if \(u = u'\)

\(\delta(u,u') = 0\) if \(u \neq u'\)

It is this property of orthonormality that allows us to decompose any sequence of eight sample values into the set of weighted cosine functions, by computing inner products.
Figure 3.1: Eight DCT cosine basis functions
3.2 Two-dimensional DCT

The 2-D DCT can be extended from the 1-D DCT. When scaled by an appropriate set of 64 coefficients, these 64 basis functions can be used to represent any 64 sample values, arranged as 8×8 blocks of samples. The way the two-dimensional DCT is obtained is by performing a one-dimensional DCT on the columns and then, a one-dimensional DCT on the resulting rows. The transformed coefficients from the two-dimensional DCT are ordered so that the mean value (the DC coefficient) is in the upper left corner of the 8 × 8 coefficient block, which is illustrated in Figure 3.2, and the higher frequency coefficients progressed by distance from the DC coefficient. Higher row numbers represents higher vertical frequencies, and higher column numbers represent higher horizontal frequencies.

![Figure 3.2: 64 (8x8) DCT basis functions](image)
and $y$ represents vertical displacements.

**FDCT:**

$$S(v,u) = \frac{C(v)}{2} \frac{C(u)}{2} \sum_{y=0}^{16} \sum_{x=0}^{16} s(y,x) \cos[(2x+1)u\pi/16] \cos[(2y+1)v\pi/16]$$  \hspace{1cm} (3.4)

**IDCT:**

$$s(y,x) = \sum_{v=0}^{16} \sum_{u=0}^{16} S(v,u) \cos[(2x+1)u\pi/16] \cos[(2y+1)v\pi/16]$$  \hspace{1cm} (3.5)

Where

- $C(u) = 1/\sqrt{2}$ for $u = 0$
- $C(u) = 1$ for $u > 0$
- $C(v) = 1/\sqrt{2}$ for $v = 0$
- $C(v) = 1$ for $v > 0$

$s(y,x)$ = 2-D sample value

$S(v,u)$ = 2-D DCT coefficient

The discrete cosine transform (DCT) partitions the image into parts (or spectral sub-bands) of differing importance with respect to the image's visual quality. We may say that the DCT and the discrete Fourier transform (DFT) have similar characteristic, in that they both transform a signal or image from the spatial domain to a frequency domain. The DCT has some advantages for image processing. It is real-valued and provides a better approximation of a signal with fewer coefficients than other transforms like the DFT.

Even when divided into one-dimensional transforms, the DCT computation involves a large number of multiplications and additions/subtractions. Therefore, quite a few algorithms have been developed to reduce the computational complexity of the 8-point DCT. Because round-off and truncation effects depend on the way in which the calculations are done, different FDCT and
IDCT implementations will probably give slightly different results [19].

3.3 DCT quantization

The aim of quantization is to discard information which is not visually significant. Since quantization is a many-to-one mapping, it is fundamentally lossy. Quantization allows us to reduce the accuracy with which the DCT coefficients are represented when converting the DCT to an integer representation. This can be very important in image compression, as it tends to make many coefficients zero—especially those for high spatial frequencies.

In one approach to quantization, the division of each DCT coefficient by its corresponding quantizer step size is followed by rounding to the nearest integer. Each coefficient is quantized independently. Due to their relatively low visual importance, larger quantize step size is used with higher frequency coefficient.

The process of scaling the DCT coefficients and truncating them to integer values is called quantization, and the rescaling to rebuild approximately the original DCT coefficient magnitude is called dequantization. In 1992, JPEG established the first international standard for still frame image compression where the encoders and decoders are DCT-based. In the JPEG system the quantized DCT coefficients are always integers. Applied as a block transform, input images are subdivided so that local correlation can be exploited. Usually the dimension of subimage is an integer power of 2. The most popular subimage sizes are $8 \times 8$ and $16 \times 16$. JPEG uses an $8 \times 8$ block size with each $8 \times 8$ block transformed independently. As we have seen, this segmentation into blocks can cause some problems.

The image is first subdivided into pixel blocks of size $8 \times 8$, which are processed left to right, top to bottom. Each $8 \times 8$ block makes its way through each processing step, and yields output in compressed form into the data stream. Because adjacent image pixels are highly correlated, the 'forward' DCT (FDCT) processing step lays the foundation for achieving data compression by concentrating most of the signal in the lower spatial frequencies. For a typical $8 \times 8$ sample block
from a typical source image, most of the spatial frequencies have zero or near-zero amplitude and need not to be encoded.

The transformed coefficients (8×8 block) may require external tables (quantization matrix), which is carefully designed 64-elements tables to uniformly quantized each block coefficients. The Tables 3.1 is such a quantization mask used in JPEG. The quantization process is controlled by two parameters: quantization table and quality factor, which used to scale the table.

Table 3.1: DCT quantization table

3.4 DCT blocks

Because the input image is subdivided into 8×8 block, i.e. the spatial frequencies in the image and the spatial frequencies of the cosine basis functions are not precisely equivalent. Segmentation into 8×8 blocks also can cause “blocking artifacts”, visible boundaries between adjacent blocks. This is because the boundary pixels of the blocks assume the mean values of discontinuities formed at the boundary points. These blocking artifacts appear as edges in the image, and abrupt edges imply high spatial frequencies. Since these high spatial frequencies at the transitions between blocks are caused by the absence of AC coefficients, non-zero AC
coefficients are required to reduce the blocking. Another method of reducing greatly these blocking artifacts is introduced by [19] in chapter 16, that is, predicting the low-frequency AC coefficients from DC coefficient changes within a $3 \times 3$ array of blocks centered on the block of interest.

3.5 DCT-based Coding Models

The JPEG standard specifies three different coding system modes for lossy encoding: the DCT-based sequential mode (baseline), the DCT-based progressive mode, and the hierarchical mode which uses extensions of either DCT-based or predictive coding. A form of DPCM (differential pulse code modulation) predicting coding system is used for sequential lossless encoding.

![Figure 3.3: JPEG Encoder Block Diagram](image)

The 'baseline JPEG coder', which is the DCT-based sequential baseline encoding in its simplest form, will be briefly discussed here. Figure 3.3 and 3.4 show the key processing steps in such an encoder and decoder for grayscale images.
After output from the FDCT, each of the 64 DCT coefficients (blocks) is uniformly quantized in conjunction with a carefully designed 64-element Quantization Table, which is shown in Table 3.1. In decoder side, the quantized coefficients are multiplied by the corresponding QT elements to get an approximation to the original unquantized coefficients. After quantization, all of the quantized coefficients are ordered into the "zig-zag" sequence as shown in Figure.3.5. This ordering helps to facilitate entropy encoding by placing low-frequency non-zero coefficients before high-frequency coefficients. The DC coefficient, which contains a significant fraction of the total image energy, is differentially encoded.

![Zig-zag sequence](image)

**Figure 3.5: Zigzag sequence for DCT**

Entropy Coding (EC) achieves additional compression losslessly by encoding the quantized DCT coefficients more compactly. The JPEG standard specifies both Huffman coding and arithmetic coding. The baseline sequential codec uses Huffman coding, but codecs with both methods are specified for all modes of operation. Arithmetic coding, though more complex, normally achieves 5-10% better compression than Huffman coding [15].
Chapter 4. Wavelet Transform and Embedded Zerotree coding

4.1 Wavelet transform

A signal can be represented as a sum of a series of sine and cosine functions in Fourier theory. This sum is also called a Fourier expansion. However, a big disadvantage of the Fourier expansion is that it has only frequency resolution but no time resolution. This means that we are able to determine the frequency components in a signal, but we do not know when these frequencies will be presented.

Among several solutions that have been developed to overcome this problem, wavelet analysis is probably the best-known solution. Unlike the Fourier transform, whose basis functions are sinusoids, wavelet transforms are based on small waves, called wavelets, with varying frequency and different duration.

Wavelet bases are known to efficiently approximate piecewise regular functions with a small number of nonzero wavelet coefficients [10]. The wavelet bases are good candidates for building efficient image transform coders. Wavelet–based analysis of images is an interesting, and relatively recent new tool, and its most popular application is being used in transform coding as a linear transform. As the general transform coding, shown in Figure 4.1, it includes three components, namely transformation, quantizer and entropy coder.

![Figure 4.1: A generic transform coder](image)

Wavelet functions are defined over a finite interval [15] and have zero mean. The basic idea of the wavelet transform is to represent any arbitrary function $f(t)$ as a superposition of a set of such
wavelet functions (basis). All the basis functions namely baby wavelets can be obtained by scaling and shifting from a single prototype wavelet namely mother wavelet. There are different kinds of basic wavelets like Haar, Daubechies, and Mexican Hat etc. In many applications of image coding, wavelet-based compression performs better than other coding schemes such as block-based DCT. Figure 4.2 is an example to show the blocking artifacts. We can see that the reconstructed image by using only the DC components of the DCT shows pronounced block-artifacts. These artifacts are also visible when using the DCT in applications which require high compression ratios.

![Original Lena Image](image1.png) ![DC components image](image2.png)

**Figure 4.2:** (a). Original Lena Image (b). Reconstructed Lena with DC component, show-blocking artifacts

The continuous wavelet transform or CWT can be written as:

\[
\gamma(s, \tau) = \int f(t) \psi^*_s(t) \, dt
\]

(4.1)

\[
\psi_s(t) = \frac{1}{\sqrt{s}} \psi\left(\frac{t-\tau}{s}\right)
\]

(4.2)

The arbitrary function \( f(t) \) is decomposed into a set of basis functions \( \psi^*_s(t) \), called the wavelets, where * indicates complex conjugate operation. The variables \( s \) and \( \tau \), are scale and shift factors, respectively. The signal can be reconstructed as follows:
\[ f(t) = \frac{1}{C_\psi} \int \int \gamma(s, \tau) \frac{\psi_{s,\tau}(t)}{s^2} d\tau ds \quad (4.3) \]

Where
\[ C_\psi = \int \left| \frac{\psi(u)}{|u|} \right|^2 du \quad (4.4) \]

And \( \psi(u) \) is the Fourier transform of \( \psi(t) \).

In wavelets analysis, the purpose of a scaling function is to create series of approximations of image [17]. The scaling function \( \varphi(t) \), that is, the set \( \{ \varphi_{j,k}(t) \} \), where \( \varphi_{j,k}(t) \) is defined by
\[
\varphi_{j,k}(t) = 2^{j/2} \varphi(2^j t - k) \quad (4.5)
\]

Here \( j, k \) are integers, and \( \varphi(t) \in L^2(R) \). Parameter \( k \) is the translation, determines the position of \( \varphi_{j,k}(t) \) along the \( t \)-axis; and \( j \) is the dilation or compression parameter, determines \( \varphi_{j,k}(t) \) 's width along the \( t \)-axis; and \( 2^{j/2} \) controls its height or amplitude. Since the shape of \( \varphi_{j,k}(t) \) changes with the \( j \), \( \varphi(t) \) is called scaling function. The scaling function can be seen as a low-pass filter and be used like a cork to cover the spectrum all the way down to zero.

Discrete wavelets are used to encode the difference in information between adjacent approximations. Given a scaling function, a set \( \{ \psi_{j,k}(t) \} \) of wavelets is defined by
\[
\psi_{j,k}(t) = 2^{j/2} \psi(2^j t - k) \quad (4.6)
\]

Where \( j, k \) are integers.

The discrete wavelet transform is a representation of a signal \( f(t) \) using an orthogonal family of basis functions.
\[
f(t) = \frac{1}{\sqrt{M}} \sum_k W_\psi(j_0, k) \varphi_{j_0,k}(t) + \frac{1}{\sqrt{M}} \sum_{j=j_0}^{\infty} \sum_k W_\psi(j, k) \psi_{j,k}(t) \quad (4.7)
\]

Where the discrete wavelet transform pair is given by:
Here \( f(t), \varphi_{j_0,k}(t), \) and \( \psi_{j,k}(t) \) are functions of the discrete variable \( t = 0, 1, 2, \ldots, M-1. \) \( j, j_0, k \) are integers and \( j \geq j_0, \) usually let \( j_0 = 0 \) and \( M \) be selected a power of 2.

Therefore, wavelet basis functions are obtained from a single mother wavelet by translation and scaling. The mother wavelet must simply satisfy a small set of conditions and is typically selected based on the signal processing problem domain.

If we treat one wavelet as a band-pass filter, then the set of dilated wavelets can be seen as the impulse responds of a band-pass filter bank. To get a good coverage of the signal spectrum, the stretched wavelet spectra should overlap, as shown in Figure 4.3.

![Figure 4.3: Touching wavelet spectra resulting from scaling of the mother wavelet in the time domain.](image)

### 4.2 Subband Coding of Image

The basic concept of the Subband Coding is to split up the frequency band of a signal and then to code each subband. The Wavelet Transform can be regarded as a filter bank, that means the wavelet transforms a signal as passing the signal through this filter bank. The outputs of the different bandpass filter stages are the wavelet and scaling function transform coefficients. In [17], the authors conclude that, in subband coding, each subband can be created by decomposing an image into a set of bandlimited components. These subbands can be reassembled to rebuild the
original image without error. The Figure 4.4 shows a two dimensional separable filters for the processing of images.

As can be seen in Figure 4.4, separable filters are first split to two parts, and then each part is split again. Filter $h_0(m)$, $h_0(n)$ are low-pass filters, whose outputs are an approximation corresponding to their input images; whereas $h_1(m)$, $h_1(n)$ are high-pass filters, their outputs are the details or high frequency part of their input signals. To reduce the overall number of computations, downsampling ($2 \downarrow$) is performed twice, once before the second filtering operation. The resulting decomposed outcomes, $a(m,n)$, $d^v(m,n)$, $d^h(m,n)$, $d^d(m,n)$ in Figure 4.4, are called approximation, vertical details, horizontal details and diagonal details subbands of image, respectively. This can be seen in Figure 4.5 (a), the LL, HL, LH, HH represent these four subbands of image. Each coefficient represents a spatial area corresponding to approximately a $2 \times 2$ area of the original image. To obtain the next coarser scale of wavelet coefficients, the lowest frequency represented subband LL is further decomposed and sampled as illustrated.
in Figure 4.5(b), two scale-level decomposition; and Figure 4.5 (c), three scale-level decomposition.

One way to build the filter bank needed in subband coding is to split the signal spectrum in two equal parts, named low-pass part and high-pass part. Because most of the energy of natural image is concentrated in low frequency [4], the high-pass part contains smallest detail. The low pass part still contains some details and then we can split at again. This process can iterate again until it satisfies a “desire state”. The “desire state” is either when a given bit-rate is satisfied or when the threshold reaches a minimum value (usually the minimum threshold is equal to 1). For reconstructing the original image, the subband images are upsampled, filtered, and then combined additively.

There are several ways in which wavelet transforms can decompose an image into various subbands, including uniform decomposition, octave-band decomposition, and adaptive or wavelet-packet decomposition [11]. Out of all these methods, octave-band decomposition is most widely used. This is a non-uniform band splitting method that decomposes the lower frequency part into narrower bands and the high-pass output at each level is left without any further decomposition.
Figure 4.6: Original ‘girl’ image and its wavelet transform coefficients map

The Figure 4.6 shows a sample 3-scale decomposition of discrete wavelet transform, it’s related to the Figure 4.5 (c), ‘three level decomposition’. We can see that the up-left corner (1/8x1/8) is the approximation area, and the others are details in different level.

The main difference between currently JPEG standard and the new JPEG 2000 standard is that the latter uses the DWT instead of the DCT. The image needs not to be divided into 8 x8 blocks, so there are no blocking artifact errors introduced into the reconstructed image. Certainly, many enhancements have been made to the standard quantization and encoding techniques to take advantage of how the wavelet transform works on an image and the properties and statistics of the transform coefficients.

The Wavelet transform exhibits the above-mentioned advantages especially at low bit rates (high compression) achieved without the blocking artifacts. Because of the interaction among the three image encoder/decoder components, the wavelet transform alone still cannot be credited with all of the improvements seen in JPEG 2000. To get the best compression, a properly designed quantizer and a suitable entropy encoder/decoder are absolutely necessary. Many improvements have recently been made to the standard quantizers and entropy encoders to take advantage of the
properties of the wavelet transform as applied to images. The properties of the HVS, and the behaviors of different transformed coefficients are also important factors that we should consider in the design of an encoder/decoder.

Over the past decade, a variety of novel and sophisticated wavelet-based image coding schemes have been developed. These include EZW [2], SPIHT [3], SFQ [4], Classified Zerotree Wavelet Image Coding and Adaptive Packetization for Low-Bit-rates Transport [11], Adaptive Wavelet Threshold for Image Denoising and Compression [12], Wavelet Image Coding Using Trellis Coded Space-Frequency Quantization [14], etc. Such innovative techniques continue to be developed as this thesis is written. These efforts give the improved results in terms of lower bit rates for a required image quality and better image quality for a given bit rate. Out of all these schemes, the Embedded Zerotrees Wavelet (EZW) compression and Set Partitioning in Hierarchical Trees (SPHIT) algorithm [3] are the most popular in practice.

Traditional transform encoders, such as those using the block-based DCT, decompose images into a representation in which each coefficient corresponds to a fixed size spatial area, and spatial areas are exactly the same for all coefficients [2]. Because many non-zero coefficients are required when representing the edge with good fidelity, the representation edge information of an image trends to disperse. However, because the edges represent relatively insignificant energy with respect to the whole image, traditional transforms have been successful for use at medium and high-bit rates. At extremely low bit rates, the traditional transform coding schemes, such as JPEG [10] that employs the block-based DCT, will allocate many bits to the “trends”, with very few bits left to represent “anomalies”. As a result, blocking artifacts often appear in the reconstructed image.

4.3 Embedded Zerotree encoding:

In 1993 Shapiro introduced a quantization scheme called “Zerotree” of wavelet coefficients in his paper “Embedded Image Coding Using Zerotree of Wavelet Coefficients” [2]. The Embedded
Zerotree quantization method and his elegant algorithm for entropy coding with the wavelet transform is referred to as the Embedded Zerotree Wavelet (EZW) algorithm. The EZW algorithm is based on two important observations:

Large wavelet coefficients are more important visually than small wavelet coefficients. Therefore, the EZW encodes the larger wavelet coefficients first.

The energy of the wavelet coefficients in the lower frequency subbands (higher scales) is higher than that in the higher frequency subbands (lower scales), because of natural images’ low pass spectrum.

According to these two observations, the wavelet decomposition is a natural basis for progressive encoding schemes, since the high frequency subbands only add details. Following this idea, the progressive encoding based EZW encoder compresses an image into a bit stream with increasing accuracy. When more bits are added to the stream, the EZW decoded image will contain more details. JPEG encoded images can also be encoded progressively. Progressive encoding is also known as embedded encoding, which explains the E in the EZW algorithm.

An EZW quantization method was originally designed to use with wavelet transform. Using an embedded coding algorithm, an encoder can terminate the encoding at any point thereby allowing a target rate or target distortion to be met exactly. A significance map is a binary function whose value determines whether each coefficient is significant or not. If not significant, a coefficient is quantized to zero. Hence a decoder that knows the significance map needs no further information about the coefficient. Otherwise, the coefficient is quantized to a non-zero value.

4.4 The Zerotree Structure

A three-level octave subband wavelet decomposition, shown in Figure 4.7. Each coefficient (except in the lowest subband) in the lower bands of the wavelet transform has four coefficients corresponding to its spatial position in the next higher subband. These four descendants also have four descendants in the next higher subbands, shown in Figure 4.8. This leads to a quad-tree,
every root has four leaves. The very special structure of the decomposition can be exploited in
encoding its coefficients to achieve better compression results.

Figure 4.7: A three-level wavelet decomposition of the Lena image

Figure 4.8: An example of Zerotree structure
The Zerotree is based on the hypothesis: all the wavelet coefficients in a quad tree will be smaller than a given threshold if the root is smaller than this threshold. Empirical and practical evidence shows that this hypothesis is often true even though sometimes it is violated. Under this condition, the whole tree including root and all its leaves can be coded with a single Zerotree symbol. Then if an image is scanned using a pre-defined order, which is moved from the coarse scale to finer, there will be a high possibility that many positions are coded through the use of zerotree symbols. Some terminologies used in the EZW algorithm can be defined as follows:

**Parent:** a coefficient at the coarse scale.

**Children:** all coefficients corresponding to the same spatial location at the next finer scale of similar orientation.

**Zerotree:** a quad-tree of which all nodes are equal to or smaller than the root;

The tree is coded with a single symbol and reconstructed by the decoder as a quad-tree filled with zeros. Many insignificant coefficients at higher frequency subbands (finer resolutions) can be discarded, because the tree grows as powers of four. The EZW algorithm encodes the tree structure so obtained. The main advantage of this encoding is that embedded wavelet coding not only provides very good compression performance, but also has the important property that we can truncate the bit-stream at any point and still decode to rebuild a reasonably good quality image form the number of bits transmitted.

### 4.5 Encoding algorithm of the EZW

Given an image, the first step is to apply a 2-D discrete wavelet transform to the input image, resulting in a set of wavelet coefficients. The next step is to determine the initial threshold. A commonly used initial threshold is:

$$T_0 = 2^{\left\lceil \log_2 \left( \text{MAX}(|C(x,y)|) \right) \right\rceil}$$

(4.10)
Where the Max (.) means the maximum coefficient and \( \lfloor \cdot \rfloor \) means floor function. The initial threshold is the largest power of 2 that is less than the maximum absolute value of the coefficients. Using this initial threshold, a dominant pass procedure (to be described in details in section 4.5.1 below) is applied to all the wavelet coefficients. After the dominant pass procedure is completed, the current threshold is halved, and then a subordinate pass procedure (described in section 4.5.2 below) is applied to all the significant coefficients coded in the dominant pass. The Figure 4.9 illustrates the block diagram of the EZW compression algorithm.

The EZW coefficients are 2-dimensional, so we have to code not only the coefficients values, but also their position in space. After computing the wavelet transform of the image, we can represent these image coefficients by using trees. The EZW encoder uses a predefined scan order to transmit the coefficients to the decoder, hence all the coefficient positions are known at decoder side. A number of scan orders are possible such as Raster, Morton etc. The fundamental rule for the scan order is that the lower subbands (coarse scale) should be completely scanned before going on to higher subbands (finer scale). The purpose of this kind of ordering is to code the insignificant coefficients (most of the coefficients in higher subbands are coded to zero). Such that the coding gain earned from using Zerotree structure can be maximized. Figure 4.10 illustrates the Morton scan order that is used in the experiments, which follow.
Apply 2-D DWT of image

Decide initial threshold:
\[ T_0 = 2^{\log_2 \left( \frac{\text{MAX} |C(x,y)|}{J} \right)} \]

DOMINANT PASS

\[ T = T / 2 \]

SUBORDINATE PASS

T = 1 or reached the desired bit rates?

Y

STOP

N

Figure 4.9: Block diagram of EZW algorithm
4.5.1 The Dominant Pass

The dominant pass is a type of significance map coding (binary map indicating the positions of the significant coefficients) for a particular bit plane. The coefficients are divided into significant (those that have magnitudes larger than the current threshold of the pass), and insignificant (those that have magnitudes smaller than the threshold). The EZW encoder codes the coefficients in successive passes referred to as the dominant (first) pass and the subordinate (following) pass. For each pass a threshold should be chosen to which all the coefficients are compared. If a coefficient is larger than the threshold then it is encoded and removed from the image, if it’s smaller than threshold it is left for the next pass. When a significant coefficient is encountered, its significance is announced with a special symbol. The dominant pass considers only the coefficients, which were not significant in previous dominant passes.

When the all wavelet coefficients have been visited, the threshold is reduced by halving the previous value, and the image is scanned again to add more detail to the already encoded image. This procedure is repeated until all the coefficients have been completely encoded.
Figure 4.11 illustrates the flow chart of the dominant pass procedure. The first step is to check whether the current coefficient was found to be previously significant or not. If it was, the current coefficient is skipped and there is no output. If was not, the coefficient (the absolute value) is compared with the current threshold. If the value is larger than the current threshold, then the absolute value of the coefficient is appended to the Subordinate List (which collects all the significant coefficients and will be used in the subordinate pass). Either a positive significant ('P') or a negative significant ('N') symbol is output; depending on the coefficient sign. In case the coefficient is insignificant, the next step is to check whether it is a descendant from a Zerotree or not. If the answer is yes, there is no output. If the coefficient does not belong to a discovered Zerotree, it must either be a new Zerotree root ('T') or an isolated zero ('Z'). In conclusion, the
output is comprised of four symbols:

1. **P**: if the coefficient is larger than the threshold, a **P** (positive) is coded;
2. **N**: if the coefficient is smaller than minus the threshold, an **N** (negative) is coded;
3. **T**: if the coefficient is the root of a **Zerotree** then a **T** (**Zerotree**) is coded;
4. **Z**: and finally, if the coefficient is smaller than the threshold but it is not the root of a **Zerotree** (which means that this coefficient has at least one significant descendant with respect to the current threshold), then a **Z** (isolated zero) is coded.

### 4.5.2 The Subordinate Pass

The following is the subordinate pass, which sometimes is also called the refinement pass. The subordinate pass will refine all significant coefficients (Subordinate List) determined by the previously dominant pass, through identifying them as being in different threshold intervals. Figure 4.12 illustrates the flow chart of the subordinate pass in the encoder side.

![Flow chart for Subordinate pass](image)

**Figure 4.12: Flow chart for Subordinate pass**

The encoder codes the significant wavelet coefficients map by stepping through in a predefined order (fully reproduced by the decoder) and outputting information accordingly. The subordinate (refinement) pass works on the coefficients found significant in previous dominant passes. It increases their precision by sending one more bit from the binary representation of their values. For each coefficient in the Subordinate List (all significant coefficients are added to the
Subordinate List by the dominant pass), the subordinate pass checks whether their current absolutely value is larger or small than the currently threshold. If it is larger, a ‘1’ is output to the entropy encoder and the current coefficient value is subtracted by current threshold. If it is smaller than the current threshold, a ‘0’ is sent to the entropy encoder. Usually entropy coding is used to further compress the bit stream produced in the subordinate pass. Many current research efforts are directed to finding efficient ways to code the significance map.

A summary of the EZW algorithm:

1. Set a threshold to \( T = 2^{\log_2 \left( \text{MAX} (|C(x,y)|) \right)} \), where MAX is the maximum magnitude of the coefficients.
2. Sorting (dominant pass) scans all the coefficients in a predefined order and output a symbol when the magnitude is larger than the threshold. The coefficient value is set to \( \pm 1.5T \).
3. Refinement (subordinate pass), refines significant coefficients, the coefficient value is changed by \( \pm 0.25T \).
4. \( T = 0.5T \). Go to step 2.

4.6 Decoding the EZW data stream

Decoding the EZW data stream also uses sets of dominant-passes and a subordinate-passes. The most important factor about the decoder is that we must use the same scanning order as the encoder used to ensure the correct position of reconstructed coefficients.

The dominant-pass loads the dominant-pass symbols from the data stream in order to retrieve significant coefficients. At each pass, there is another reference matrix the same size as the original coefficients matrix, with all elements of the matrix initially set to zero. Once a coefficient is scanned, its corresponding reference matrix coefficient is set to ‘1’. After each dominant-pass is ended, all of reference matrix’s coefficient should be ‘1’. This means that every coefficient is scanned in this dominant-pass, otherwise, something must be wrong. When a positive significant
'P' is detected, a value of 1.5 times the current threshold is added (the initial threshold at decoder side is the same as in encoder) to the reconstructed coefficient; when a negative significant 'N' is detected, a value -1.5 times the current threshold is added. When a Zerotree 'T' is detected, there is no contribution to the reconstructed coefficients, just set the current position and all its descendants' positions are simply to 1 in the reference matrix. When an isolated zero is detected,

![Diagram](image)

Figure 4.13: EZW decoder's dominant-pass flow chart

nothing will be done to the reconstructed coefficient, except setting the current position in reference matrix to '1'. The dominant-pass ends after all the coefficients have been scanned.

Figure 4.13 and 4.14 illustrate the flow chart of dominant pass and subordinate pass at the decoder side, respectively. The subordinate passes load the pass symbol data stream for each pass. Also at the subordinate pass, there is another array the same size as the subordinate array, recording all the significant coefficients. This array is used to record every significant coefficients position in the reconstructed coefficients map. If a '1' is encountered, a value of half threshold (1 over 4 threshold of dominant-pass used) is added to or subtracted from the current coefficient depending on whether its value is positive or negative. If a '0' is encountered, a value of half
threshold is subtracted from or added to the current coefficient depending on whether its value is positive or negative (the situation is reversed).

Decoding will stop either when a desired bit rate has been satisfied or when the threshold reaches a minimum value (usually the minimum threshold is set to 1).

![Flow chart](image)

**Figure 4.14: EZW decoder's subordinate-pass flow chart**

After the reconstructed coefficients map is obtained, the inverse discrete wavelet transform is used to recover the image from reconstructed coefficients. Assume the original image is defined by a set of pixel values \( p_{i,j} \), where \((i, j)\) is the pixel coordinate. The decoder of the EZW establishes a reconstructed image from the received data stream, named \( \tilde{p}_{i,j} \), the mean square error (MSE) between original and reconstructed image is given by

\[
MSE(p - \tilde{p}) = \frac{\|p - \tilde{p}\|^2}{MN} = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} (p_{i,j} - \tilde{p}_{i,j})^2
\]  

(4.11)
Where $M$ and $N$ are the row and column number of the image respectively, $M \times N$ is the number of image pixels. Then the distortion between the original and reconstructed image is measured in terms of the Peak Signal to Noise Ratio

$$PSNR = 10 \log_{10} \left( \frac{255^2}{MSE \left( p - \hat{p} \right)} \right) \text{ dB}$$ (4.12)
Chapter 5. Implementation of the EZDGT

Since Embedded Zerotree Quantization was initially designed for use with wavelet transform coefficients, if we want to apply this kind of quantization method to the Derivative Gaussian Transform (DGT) coefficients, we must satisfy all the EZW’s requirements. The most important factor is to form a significant tree structure as done for the wavelet coefficient map so that the Embedded Zerotree quantization method can be applied. This requires rearranging the DGT coefficients to form a wavelet-like tree structure. Because the DCT and DGT can use the same organization for their coefficients, a reorganization strategy suitable for the DCT could also be used for the DGT. In this work, we adapt the method developed by Xiong et al’s [16] for the DCT to the DGT.

5.1 Significance Tree Quantization

In [16], Zixiang Xiong, O.G. Guleryuz and M.T. Orchard first claimed that since the wavelet transform is just one member of a large family of linear transforms, the DCT could also be coupled with an Embedded Zerotree quantizer. Their DCT-based embedded image coder performs better than other DCT-based coders like JPEG, and also achieves higher peak signal-to-noise ratios (PSNR) than the quoted results of Shapiro’s EZW coder. This method that employs the Embedded Zerotree quantizer to the DCT is called EZDCT.

In [22], D. Monro and G. Dickson showed that an efficient Embedded Zerotree quantizer originally developed for wavelet compression algorithm could be used with the DCT instead of the DWT with excellent results. Combining the near optimum decorrelation of the DCT with the efficient significance map coding of the Zerotree data structure, their algorithm achieved very competitive compression. They used different block sizes such as 8×8, 16×16 and 32×32 etc, to evaluate the performance of the method. In their results, the 16×16 blocks yielded the best compression results, whereas the 8×8 blocks produced the best visual quality exhibiting reduced ringing artifacts.
The schemes like EZW, SPHIT and EZDCT employ a common algorithm called significance tree quantization (STQ). In [24] and [25], the authors G. Davis and S. Chawla talked about significance tree quantization, that is a scheme for producing an n-vector \( X=(X_1, X_2, \ldots, X_n) \) of coefficients using a tree-structured significance map. The basic data structure in a STQ scheme is a tree or set of trees in which each leaf corresponds to one of the scalars \( X_k \). A coefficient \( X_k \) is called significant with respect to a given threshold \( T \) if \( |X_k| \geq T \), otherwise it is called insignificant.

The excellent performance of EZW and SPHIT wavelet coders can be explained in part by their ability to effectively exploit coefficient interdependencies. The performance of significance tree quantization is strongly dependent on the particular manner in which coefficients are grouped. The cost of the significant tree depends on the tree structure used. Therefore, to minimize the total encoding cost, finding a reasonable tree structure for the significant map is very important.

Similarly with the DGT, one of the most important things is how to decide the method used to group coefficients together to form a tree structure.

DCT coefficients decay rapidly in frequency for most 8×8 blocks. As a result, high frequency coefficients will tend to be insignificant simultaneously [24] at the same frequencies in each block. By grouping these coefficients together in a significance tree, the result is a large collection of high frequency coefficients that are relatively insignificant with respect to a given threshold.

5.2 The EZDCT Sub-band implementation

After Shapiro introduced his famous Embedded Zerotree wavelet coder (EZW) and Said and Pearlman developed an algorithm by set partitioning in hierarchical trees (SPHIT), much of the research in image coding has focused on application of wavelet transforms. Their methods provide very high compression efficiency in terms of Peak Signal-to-Noise Ration (PSNR) versus
required bits-per-pixel (bpp). In paper [6], the authors Debin Zhao, Dapeng Zhang and Wen Gao illustrated that the Zerotree quantizer developed originally for wavelet compression could be effectively applied to Discrete Cosine Transform (DCT) in a hierarchical way. When matched with a Zerotree quantizer instead of the traditional methods which were used in JPEG, the DCT also can provide very high compression efficiency, although wavelets are capable of providing more flexible space-frequency resolution.

As discussed previously, in DCT-based compression, the input image \((M \times N)\) is partitioned into a number of \(8 \times 8\) blocks, and each block is then transformed by a 2-D DCT. Each \(8 \times 8\) DCT block can be treated as a depth-3 tree of coefficients labeled from 1 to 64 as shown in Figure 5.1(a). We can see the parent-children relationships between DCT coefficients as follows:

1. the coefficient labeled \(j\)'s parent is \(\lfloor \frac{j}{4} \rfloor\) for \(1 \leq j \leq 64\),

   where the \(\lfloor \cdot \rfloor\) is the floor function;

2. the set of its four children is \(\{4j-3, 4j-2, 4j-1, 4j\}\) for \(1 \leq j \leq 16\).

The basic idea to form a significant tree-structure for \(8 \times 8\) DCT coefficients is to view it as a 10-subband decomposition as shown in Figure 5.1(b). This procedure is repeated on all the DCT coefficients \((M/8) \times (N/8)\) with DCT block size of \(8 \times 8\) until the last step is reached. For example, \(256 \times 256\) image is divided into \((256/8) \times (256/8) = 1024, 8 \times 8\) blocks.

When an image is treated as a single entity, the DCT coefficients within each block can be reorganized to obtain similar to that used in EZW and SPHIT. We can take each \(8 \times 8\) DCT blocks as a 10-subband decomposition, as shown in Figure 5.1(a) and the 5.1(b).
The next step is grouping subbands with the same index together for all DCT blocks. Figure 5.2 illustrates this strategy for reorganizing the DCT coefficients.

In order to use Embedded Zerotrees Quantization with a DCT transformed image, it is necessary to rearrange the coefficients from individual localized blocks to a global hierarchical subband structure [22]. Grouping together DCT coefficients of similar spatial orientation and frequency in
the hierarchical subband structure, we obtain the larger blocks required for a significant subband
tree structure.

![Diagram of DCT blocks](image)

Figure 5.3: Example of the rearrangement of four 4×4 DCT blocks to form a hierarchical subband structure.

Figure 5.3 illustrates an example to show how to rearrange four 4×4 DCT blocks to form a
hierarchical sub-band structure. In a 4×4 DCT block, the coefficients A11, B11, C11 and D11
make up the lowest frequency subband; the A12, B12, C12 and D12, the A21, B21, C21 and D21,
the A22, B22, C22 and D22 coefficients are assembled into horizontal, vertical and diagonal
subbands at the same scale respectively. Interleaving the higher frequency DCT coefficients,
forms larger, higher frequency sub-bands.
For the different $N \times N$ block size, the number of decomposition scales in the sub-band image $M$, is related to the $N$ by

$$M = \log_2 N$$  \hspace{1cm} (5.1)

For reorganization of all $8 \times 8$ DCT blocks into a single DCT entity, an important step of this method involves the grouping of the same subbands for all DCT blocks. This step is illustrated as Figure 5.4. $Go1$ is a group of subbands 1 as shown in Figure 5.1 (b), and so on, with $Go10$ a group of subbands 10. To form a wavelet-like coefficient map such as Figure 5.5 shown below, reorganize each $8 \times 8$ DCT block into 10 subbands. The result is a hierarchical sub-band structure to which the embedded quantization method can be applied. Figure 5.6 is the block diagram of the EZDCT algorithm. More details about how to implement the EZDCT algorithm are described in Section 5.2.1.

Figure 5.4: Reorganized DCT 10-subband image
DOCUMENTS
CAPTURED AS
RECEIVED
Figure 5.5 is a 3-scale wavelet coefficients map in which the Embedded Zerotree Quantization method can be applied; the Figure 5.7 is the original Lena image and its DCT coefficients map. To apply the EZW method to the DCT coefficients, it is necessary to reorganize the DCT coefficients to form a wavelet-like coefficients tree structure, as discussed above.

![Wavelet coefficients using "haar" wavelet](image)

**Figure 5.5: The Wavelet coefficients map of Girl image (3-scale)**
START

Divide Image into 8x8 blocks, Take 2-D DCT for each block

Re-organize DCT coefficients to form a wavelet-like structure

Determine the initial threshold
\[ t_0 = 2^\left\lfloor \log_2 \{\text{MAX}\ (|C(x,y)|)\} \right\rfloor \]

DOMINANT PASS

\[ T = T / 2 \]

SUBORDINATE PASS

N

T = 1 or reached the desired bit rates?

Y

STOP

Figure 5.6: Block Diagram of EZDCT Encoder
5.2.1 Procedures of the EZDCT

In [21], the authors presented a progressive DCT image compression scheme, which generates an embedded bit stream for DCT coefficients according to their importance. The embedding property is essential for progressive image transmission; it greatly simplifies the rate-control problem when the bit rate is low. Getting inspiration from the several references cited above, the encoder procedure using the DCT with Embedded Zerotree quantizer, are as follows:

**Step 1: Divide the image into N x N blocks**

The input image is partitioned into blocks. In the following, we will assume 8 x 8 blocks.

For example, if the image size is 256 x 256, the total block number is 32(256/8) x 32(256/8) = 32 x 32 = 1024; if the image size is 512 x 512, the total 8 x 8 block number is 64 x 64, etc.

**Step 2: Apply the DCT for each 8 x 8 blocks**

The block DCT transform is applied to each block. The DCT coefficients \( C' \) are scaled with the standard JPEG quantization table \( Q \) (illustrated in Table 3-1):
\[ C_j = F \times \frac{C_j'}{Q_j}, \quad j = 1, \ldots, 64 \]  \hfill (5.2)

The scaling is performed to recognize the visual importance of low frequency components; the quality factor \( F \) is usually chosen to be between 30 to 100.

**Step 3: Rearrangement of DCT blocks to form a wavelet-like hierarchical sub-band tree structure**

After the DCT transform and scaling, JPEG applies a uniform quantization, which maps each DCT coefficient to a value in a finite index set \([21]\). The value is then converted to an intermediate symbol and encoded by an entropy coder. In order to use Embedded Zerotrees with DCT transformed image coefficients, DCT coefficients of similar spatial orientation and frequency are grouped together to create the larger blocks required in the hierarchical sub-band structure as mentioned above. Larger and higher frequency DCT coefficient sub-bands are formed by interleaving the higher frequency DCT coefficients.

This kind of structure has properties similar to those of the Wavelet tree structure.

**Step 4: Determine the initial threshold**

For all coefficients, \( C_{i,j} \) in EZDCT, STQ, EZW and SPIHT, an initial threshold is chosen as a power of 2. The power \( n \) can be computed in an initialization step:

\[ n = \left\lfloor \log_2(\max_{i,j}(|C_{i,j}|)) \right\rfloor \]  \hfill (5.3)

Where max is the maximum magnitude of the coefficients, and \( \left\lfloor \right\rfloor \) is the floor function.

Then the initial threshold is determined by

\[ T_0 = 2^n \]  \hfill (5.4)

Because the maximum absolute value of coefficients in each block may be very different, it may not be reasonable to only use one initial threshold \( T_0 \) for all blocks. Some papers suggest using different threshold factors, \( n_k \) for each block \( k \).
\[ n_k = \left\lfloor \log_2 \left( \max_{(m,n)} \{|b_{m,n}|\} \right) \right\rfloor \quad (5.5) \]

Where \( b_{m,n} \) are the coefficients in block \( k \). Since much redundancy exists between \( n_k \) and \( n_{k-1} \), it may be possible to decorrelate them.

In this work, we use the same initial threshold for all 8x8 blocks.

**Step 5: Embedded Zerotree quantization**

After we identify the parent-children relationship between DCT coefficients, the Embedded Zerotree quantizer is applied to the wavelet-like tree structure DCT coefficients.

Embedded Zerotree quantization is a type of bit plane coding. It consists of two steps, sorting (dominant pass) and refinement (subordinate pass), iterated for each bit plane until a desired image quality or bit rate is achieved. Each iteration, called a pass, is associated with a threshold. The threshold is halved in each iteration. See chapter 4 for more details in our discussions of the EZW.

**Step 6: Entropy Coding**

One approach to entropy coding the quantized symbols is by predicting the location of the significant coefficients with the context adaptive arithmetic coder used in the JPEG extended system. The refinement symbols and signs can be encoded with specific context. In order to improve compression performance and maintain low complexity, a binary adaptive arithmetic coder may also be used. In the EZDCT implementation used here, the compression achieved by entropy coding is estimated by computing the entropy of the symbol stream converted to bits per pixel (bpp).

In decoding, all the above-mentioned procedures are reversed.

**5.2.2 An example of the EZDCT**

In this section, we show an example of DCT based encoding using Embedded Zerotree quantization. Reconstructed images are shown at each pass. A quantization matrix (Q) and
quality factor (QF) are used in the encoder/decoder to trade image quality in terms of the Peak Signal-to-Noise Ratio (PSNR) and cost in terms of bits per pixel (bpp). The reconstructed images’ bpp and PSNR are listed as follows, for example, a reconstructed image at 0.0176 bit-rates, its PSNR is 7.6078 dBs. Figures 5.10 to Figure 5.22 show all these 13 reconstructed images.

\[
\begin{align*}
\text{Bpp} &= 0.0176 \ 0.0669 \ 0.1122 \ 0.1662 \ 0.2606 \ 0.4220 \ 0.6785 \ 1.0733 \\
& \quad 1.6841 \ 2.6030 \ 3.9384 \ 5.8860 \ 7.3645 \\
\text{PSNR (dB)} &= 7.6078 \ 17.4102 \ 24.2897 \ 26.8606 \ 29.3565 \ 31.7761 \ 33.9856 \\
& \quad 35.9641 \ 37.9961 \ 40.2274 \ 43.2322 \ 48.9083 \ 55.2849
\end{align*}
\]

Figure 5.8. shows the bpp vs. PSNR curve of the reconstructed Lena images; Figure 5.9 is the reorganized DCT coefficients (Wavelet-like) tree structure to which Embedded Zerotree quantization can be applied. The whole process includes 13 dominant passes and subordinate passes. The initial threshold \( T_0 \) was chosen based on formula 5.4, which is 4096. When a pass is finished, the thresholds is halved. At the finally pass the threshold will be 1. The reconstructed image quality in terms of PSNR is progressively increased from about 7.6dB (first pass) to 55.3dB (13th pass), its relative cost in terms of bits per pixel (bpp) gradually increased from 0.0176 (first pass) to 7.3645(13th pass). From Figures 5.10 to Figure 5.22, we can see that the reconstructed images’ quality becomes better and better as the pass number increases. At lower bit rates, the coding artifacts usually appear on the boundaries between adjacent blocks because of the DCT dividing the input image into blocks.

In encoding, we use quantization matrix Q (see table 3.1) and the quality factor QF to quantize the coefficients as follows:

\[
\text{AA} = \text{round} (\text{QF} \times (\text{DCT2(AA)}/Q)) \tag{5.6}
\]

where ‘round’ means round towards nearest integer; DCT2 means 2 dimensional DCT.

And in the inverse DCT use the formula

\[
\text{BB} = \text{round} (\text{IDCT2(BB \times Q/QF)}) \tag{5.7}
\]

59
DOCUMENTS
CAPTURED AS RECEIVED
where quality factor QF is set equal to 50.

**Figure 5.8: Bpp vs. PSNR Curve of reconstructed image Lena**

**Figure 5.9: Reorganized DCT coefficients (wavelet-like) tree structure**
Figure 5.10: Reconstructed image at bpp=0.0176, PSNR=7.61 dB, threshold=4096 (First pass)

Figure 5.11: Reconstructed DCT image at bpp=0.0669; PSNR=17.41 dB, threshold=2048
Figure 5.12: Reconstructed image at bpp=0.1122; PSNR=24.29 dB, threshold=1024

Figure 5.13: Reconstructed image at bpp=0.1662; PSNR=26.86 dB, threshold=512
Figure 5.14: Reconstructed image at bpp=0.2606; PSNR=29.36 dB, threshold=256

Figure 5.15: Reconstructed image at bpp=0.4220; PSNR=31.78 dB, threshold=128
Figure 5.16: Reconstructed image at bpp=0.6785; PSNR=33.99 dB, threshold=64

Figure 5.17: Reconstructed image at bpp=1.0733; PSNR=35.96 dB, threshold=32
Figure 5.18: Reconstructed image at bpp=1.6841; PSNR=38.00 dB, threshold=16

Figure 5.19: Reconstructed image at bpp=2.6030; PSNR=40.23 dB, threshold=8
Figure 5.20: Reconstructed image at bpp=3.9384; PSNR=43.23 dB, threshold=4

Figure 5.21: Reconstructed image at bpp=5.8860; PSNR=48.91 dB, threshold=2
5.3 Implementation of the EZDGT

5.3.1 Quantization matrix design for the EZDGT

To allow an effective tradeoff between reconstructed image quality (e.g., in terms of the Peak Signal-to-Noise Ratio (PSNR)) and cost in terms of bits per pixel (bpp), we now designed a quantization matrix to be used in the same way as that used in JPEG.

The sensitivity of the HVS to error introduced into an image is dependent on the frequency bands into which those errors are introduced [1]. A simple model capturing this effect is the contrast sensitivity function (CSF), which describes the sensitivity of the HVS to contrast changes at different spatial frequencies. Quantization is accomplished via a 2 dimensional quantization matrix, based on the reciprocal of the 1 dimensional contrast sensitivity model by Mannos and Sakrison in their paper “The effects of a visual fidelity criterion on the encoding of image”[26]. This model is given below as a function of radial frequency, \( \rho \), in cycles per degree subtended at the human eye.

Figure 5.22: Reconstructed DCT image at bpp=7.3645, PSNR=55.28 dB, threshold=1
The following values of the constants have been suggested, and are used in this work:

\[ H(\rho) = A \left[ \alpha + \frac{\rho}{\rho_0} \right] e^{\left( \frac{\rho}{\rho_0} \right)^\beta} \]  
(5.6)

A dot pitch of 0.31mm is assumed (the resolution of the monitor is assumed to 0.31mm), with a viewing distance of different multiples of the picture height. For example, a 512×512 size of image, we choose a distance as 5 times the image height; a 256×256 size of image, we use a distance of 7 times the image height (see Figure 5.23).

The function is plotted against the radial frequency up to 50 cycles/degree in Figure 5.24.

\[ A=2.6, \quad \alpha =0.0192, \quad \rho_0 =8.772, \quad \text{and} \quad \beta =1.1. \]
Contrast sensitivity as predicted by the Mannos and Sakrison model

Figure 5.24: Contrast sensitivity as predicted by Mannos and Sakrison model

In this implementation, the DGT quantization matrices for 512x512 and 256x256 images are given as follows:

Qmat(512x512) =
1.0766 1.2965 1.0348 1.0425 1.1711 1.3767 1.6911 2.1492
1.2965 1.1076 1.0204 1.0621 1.2862 1.5185 1.8632 2.3595
1.0348 1.0204 1.0369 1.1210 1.2380 1.4316 1.6911 2.0696
1.0425 1.0621 1.1210 1.2380 1.4316 1.6911 2.0696 2.6101
1.1711 1.2037 1.2862 1.4316 1.6585 1.9547 2.3815 2.9862
1.3767 1.4177 1.5185 1.6911 1.9547 2.2947 2.7805 3.4647
1.6911 1.7412 1.8632 2.0696 2.3815 2.7805 3.3469 4.1403
2.1492 2.2106 2.3595 2.6101 2.9862 3.4647 4.1403 5.0814

Qmat (256x256) =
1.2469 1.6243 1.1600 1.0351 1.0245 1.0706 1.1617 1.2982
1.6243 1.3152 1.1023 1.0256 1.0298 1.0818 1.1767 1.3162
1.1600 1.1023 1.0396 1.0197 1.0474 1.1106 1.2132 1.3594
1.0351 1.0256 1.0197 1.0365 1.0858 1.1617 1.2747 1.4306
1.0245 1.0298 1.0474 1.0858 1.1520 1.2406 1.3657 1.5341
1.0706 1.0818 1.1106 1.1617 1.2406 1.3406 1.4780 1.6601
1.1617 1.1767 1.2132 1.2747 1.3657 1.4780 1.6296 1.8287
1.2982 1.3162 1.3594 1.4306 1.5341 1.6601 1.8287 2.0486
DOCUMENTS CAPTURED AS RECEIVED
Each coefficient is divided by the respective matrix value; multiplied by a quality factor, and rounded to the nearest integer. The Embedded Zerotree Quantization method is applied to the result. Note that these quantization matrices do not seek to take advantage of the oblique effect (the reduction of visual sensitivity to diagonal structure). Refinements to this matrix may be explored in future work.

5.3.2 Implementation of the EZDGT

$m_1$ (the number of basis functions at each location) and the $D$ (in the spatial domain the basis functions are $D$ pixels apart) were chosen to be $m_1 = D = 8$. The DGT coefficients map will appear $8 \times 8$ areas (blocks). It should be noted, however, that the image itself is not partitioned into blocks, Figure 5.25 shows a typical DGT coefficients map. To apply the Embedded Zerotree method to the Quantized DGT coefficients, it is necessary to reorganize the DGT coefficients to form a wavelet-like coefficient structure. Since the Quantization matrix we specially designed for DGT coefficients is used in the same way as that used in JPEG, so before we apply the quantization matrix to the DGT coefficients, reorganizing the DGT coefficients to form a DCT liked coefficients map is necessary. The procedure is as follows:

![Original DGT Lena coefficients map.](image)

Figure 5.25: Original DGT Lena coefficients map.
1. Apply the DGT to the input image. A typical DGT coefficients map is shown in Figure 5.25.

2. Reorganize the DGT coefficients to get a DCT-like coefficient map as shown in Figure 5.26.

**Figure 5.26**: Reorganized DCT-like DGT coefficients map

**Figure 5.27**: Reorganized coefficients map after applying qmat & quality factor
3. Apply the specially designed quantization matrix and the quality factor to the DCT-like reorganized DGT coefficients. Since the absolute values of the DGT coefficients are very small, the quality factor we use is much larger than that used in DCT coefficients. In this work, the quality factor is set to equal 2000. The coefficients map is shown as Figure 5.27.

4. These DCT-liked coefficients are reorganized into a wavelet-like coefficient structure, as implemented in the EZDCT (Figure 5.28).

![Wavelet-like DCT coefficients tree-structure image before EZW](image)

**Figure 5.28: Wavelet-like DGT coefficients map**

Now we can employ the Embedded Zerotree quantization to the reorganized and quantized DGT coefficients. In the decoder side, every step is reversed.

The following results are an example of EZDGT using quality factor with special designed quantization matrix (qmat). Based on the discussion in Chapter 2, the variance of the lowest DGT order basis function is 4.8, the others are 3.45 (See sigma'). The derivative orders are 0, 3, 10, 21, 36, 53, 74, and 99 (the derivatives). Figures 5.29 and 5.30 are the bpp vs. PSNR curves of the reconstructed images at bit-rates (up to 8 bpp) and at low bit-rates (up to 1 bpp) using different
transforms (DCT, Wavelet and DGT). The followed list shows the reconstructed images’ bit-rates and PSNRs, the entire process is total 13 passes (including 13 dominant passes and 12 subordinate passes). Because at the first pass, the reconstructed image’s bit-rates and PSNR are so low that almost can’t distinguish the image, so we do not show this image. Figures 5.31 to 5.38 illustrate the reconstructed images from second pass to 9th pass, as can be seen, the bit-rates/PSNR of the reconstructed images starts at 0.0055/6.24dB (second pass), and progressively increase to 1.7699/39.79dB (9th pass).

\[ Bpp = \begin{array}{cccccccccccc}
0.0001 & 0.0055 & 0.0512 & 0.0989 & 0.1731 & 0.3119 & 0.5538 & 0.9742 & 1.7699 \\
3.1611 & 5.1175 & 7.2670 & 8.3125 \\
\end{array} \]

\[ \text{PSNR (dB)} = \begin{array}{cccccccccccc}
5.6621 & 6.2402 & 15.0217 & 23.2696 & 27.7220 & 31.2475 & 34.3277 & 37.0257 \\
39.7927 & 43.3590 & 48.6880 & 57.3614 & 62.7583 \\
\end{array} \]

\[ \sigma' =\begin{array}{cccccccccccc}
\end{array} \]

\[ \text{derivatives} = \begin{array}{cccccccccccc}
0 & 3 & 10 & 21 & 36 & 53 & 74 & 99 \\
\end{array} \]

Figure 5.29: The results of different algorithms bpp vs. PSNR chart
The Embedded Zerotree quantization includes 13 passes; the reconstructed images from pass 2 to pass 9 for the EZDGT are illustrated in Figures 5.31 to 5.38. Even though at very low bit-rates 0.0055 (Figure 5.31), the visual quality of the reconstructed image is still very good.

Figure 5.31: Reconstructed DGT image at bpp=0.0055, PSNR=6.24dB, TH=2048
Figure 5.32: Reconstructed DGT image at bpp=0.0512, PSNR=15.02dB, TH=1024

Figure 5.33: Reconstructed DGT image at bpp=0.0989, PSNR=23.27dB, TH=512
Figure 5.34: Reconstructed DGT image at bpp=0.1731, PSNR=27.72dB, TH=256

Figure 5.35: Reconstructed DGT image at bpp=0.3119, PSNR=31.25dB, TH=128
Figure 5.36: Reconstructed DGT image at bpp=0.5338, PSNR=34.33dB, TH=64

Figure 5.37: Reconstructed DGT image at bpp=0.9742, PSNR=37.03dB, TH=32
Figure 5.38: Reconstructed DGT image at bpp=1.7699, PSNR=39.79dB, TH=16

From the bpp vs. PSNR curves and the reconstructed images, we can see that the quality of the reconstructed images using the DGT is better than those using the DCT, but a little bit worse than those using Wavelet Transform at higher bit-rates. At lower bit-rates range (from 0 bpp to 1 bpp), the DGT obtains the same or better results as achieved by the Wavelet transform, in terms of PSNR. At higher bit-rates, the PSNRs of the reconstructed images are usually more than 30 dB. To human eye, these images appear nearly identical to the original. The progressive quantization process is also shown from the reconstructed images. It can be seen that in the initial passes, the reconstructed image is a fairly rough approximation. After additional passes, more details are added to the reconstructed image. This is very useful for network transmission, as we do not need wait to see the whole image. Once we see the approximation, we can decide whether or not to continue with the transmission. This saves substantial time and bandwidth.
5.4 Comparing with the different transformation algorithms using Embedded Quantization method at given bpp

For comparing the performances of different transform algorithms using the same quantization method (Embedded Zerotree) at lower bit-rates, we have implemented several codecs. We chose:

1. the Discrete Cosine Transform (DCT), which is the most common used in the current standard JPEG;

2. the Wavelet transform, which will be the standard transform in JPEG-2000;

3. the Derivative of Gaussian Transform which is discussed in this paper.

The bits per pixel (bpp) we achieve is estimated by computing the entropy of the symbol stream. The bit-rates selected were 1.00, 0.50, 0.25, 0.125 and 0.0625. The test images are ‘Lena’, ‘Barbara’, ‘Girl’, and ‘Camman’.

In each EZW quantization pass, the first pass is a dominant pass, so we calculate the symbol stream’s entropy and bits per pixel (‘bpp’), once the bpp satisfies the target bpp, then encoder stops. If the bpp does not satisfy the target bpp, then continues the subordinate pass, and compute the bpp again, until the target bit-rates are satisfied. At decoder side, the decoder reconstructs the image based on the encoder’s bit stream.

5.4.1 Example of reconstructed ‘Lena’ image:

We first show the reconstructed Lena image (512×512) at bpp equal to 1.0, 0.5, 0.25, 0.125, and 0.0625 applying different transforms and using Embedded Zerotree quantization method. These algorithms are called EZDGT, EZDCT and EZW. The wavelet used in the EZW is Daubechies #5 wavelets (‘db5’).

1. At 1.0 bpp:

Figures 5.39, 5.40 and 5.41 are the reconstructed images at 1.00 bpp using EZDGT, EZDCT and EZW, respectively. Because the PSNR is more than 35 dB, we almost cannot distinguish between
them. However, the EZDGT scheme performs best in terms of PSNR, followed by the EZW and EZDCT.

![Reconstructed DGT image at bpp=1.00, PSNR=37.03dB](image1)

Figure 5.39: Reconstructed DGT image at bpp=1.00, PSNR=37.03dB

![Reconstructed DCT image at bpp=1.00, PSNR=35.01dB](image2)

Figure 5.40: Reconstructed DCT image at bpp=1.00, PSNR=35.01dB
2. At 0.5 bpp:

Figures 5.42, 5.43, and 5.44 show the reconstructed images using the EZDGT, EZDCT and EZW, respectively at 0.5 bpp. The EZDCT reconstructed image shows block artifacts in the arm and face area, but the EZDGT and EZW look very good.
Figure 5.42: Reconstructed DGT image at bpp=0.5, PSNR=32.75dB

Figure 5.43: Reconstructed DCT image at bpp=0.5, PSNR=31.75dB
3.1 25 bpp:

Figures 5.45 5.46, and 5.47 are the reconstructed images using the EZDGT, EZDCT and EZW at 0.25 bpp. Almost in the entire image in the EZDCT case exhibits block artifacts. The PSNR of EZDGT image has a PSNR of only 28.82 dB, but it looks better than from the EZDCT (28.14 dB) and appears essentially the same as that resulting from EZW (29.71 dB).
Figure 5.45: Reconstructed EZDGT image at bpp=0.25, PSNR=28.82dB

Figure 5.46: Reconstructed EZDCT image at bpp=0.25, PSNR=28.14dB
4. At 0.125 bpp:

Figure 5.47: Reconstructed EZW image at bpp=0.25, PSNR=29.71dB

Figure 5.48: Reconstructed EZDG image at bpp=0.125, PSNR=23.98dB
Figure 5.49: Reconstructed DCT image at bpp=0.125, PSNR=24.33dB

Figure 5.50: Reconstructed EZW image at bpp=0.125, PSNR=25.03dB

5. At 0.0625 bpp:
Figure 5.51: Reconstructed EZDGT image at bpp=0.0625, PSNR=16.76dB

Figure 5.52: Reconstructed EZDCT image at bpp=0.0625, PSNR=14.49dB
Figures 5.48, 5.49, and 5.50 are those reconstructed images using EZDGT, EZDCT and EZW at 0.125 bpp. Figure 5.51, 5.52, and 5.53 are at 0.0625 bpp. The EZDCT image severe blocking, such that image contours are badly distorted. The EZDGT and EZW images show much more detail, and the contours are clear, even at very low bit-rates 0.0625. The EZDGT yields the best results both in terms of PSNR and in visual quality. Clearly, the DGT and Wavelet transform are better than the DCT at low bit-rates (high compression ratio). We can conclude from these results that the quantization errors introduced by the DGT are less offensive than those resulting from the DCT, and even those caused by the EZW.

Figure 5.54 illustrates that the bpp vs. PSNR curve of the reconstructed images. We can see that the EZDGT performs better in terms of PSNR than the EZDCT, but a little bit worse than EZW, at 0.1 bpp to 0.5 bpp. On the other hand, even though the EZDGT reconstructed image’s PSNR is lower than that using EZW at 0.125 bpp (Figure 5.48, Figure 5.50), the visual quality of the
reconstructed image is very good. The quantization errors with the DGT are less objectionable than resulting from the DCT and Wavelet transforms.

![Bpp vs. PSNR Curve of the reconstructed Lena images](image_url)

**Figure 5.54: Bpp vs. PSNR Curve of the reconstructed Lena images**

### 5.4.2 Example of reconstructed ‘Barbara’ image:

We next consider the “Barbara” image. The image size is $512 \times 512$, the target bit-rates are 0.5, 0.25, 0.125 and 0.0625. Since this image is not as smooth as the Lena image, the EZDGT algorithm achieves much better results than the EZDCT and EZW. Figures 5.55 to 5.66 are the reconstructed images at 0.5, 0.25, 0.125 and 0.0625 bits per pixel using EZDGT, EZDCT and EZW. We can clearly see that the EZDGT yields the best results. Although at 0.0625 bpp, the EZDGT’s PSNR (14.81 dB) is a little worse than EZDCT’s (15.83 dB), the reconstructed images (Figure 5.64 and 5.65) make it clear that the EZDGT’s visual quality is much better than the EZDCT’s. Most of the time, EZW is better than EZDCT.

1. At 0.50 bpp:
Figure 5.55: Reconstructed EZDGT image at bpp=0.5, PSNR=29.06dB

Figure 5.56: Reconstructed EZDCT image at bpp=0.5, PSNR=26.96dB
2. At 0.25 bpp:

Figure 5.58: Reconstructed EVDGT image at bpp=0.25, PSNR=25.64dB

91
3. At 0.125 bpp:

Figure 5.59: Reconstructed EZDCT image at bpp=0.25, PSNR=23.69dB

Figure 5.60: Reconstructed EZW image at bpp=0.2500, PSNR=23.52dB
Figure 5.61: Reconstructed EZDGT image at bpp=0.125, PSNR=21.83dB

Figure 5.62: Reconstructed EZDCT image at bpp=0.125, PSNR=21.49dB
4. At 0.0625 bpp:

Figure 5.64: Reconstructed EZDGT image at bpp=0.0625/PSNR=14.81dB
The Figure 5.67 illustrates the bpp vs. PSNR curve of the reconstructed Barbara image between 0 to 1 bpp.
The Tables 5.1, 5.2, 5.3 and 5.4 show performance comparisons of the EZDGT, EZDCT and EZW for the “Lena”, “Barbara”, “Girl” and “Camman” images. The reconstructed images' PSNRs at 1.00, 0.50, 0.25, 0.125 and 0.0625 (for “Lena” and “Barbara” only) bits per pixel are shown in these tables. Two images are 512 × 512, the two are 256 × 256 pixels in size. From these performance comparisons, we can see that at low bit-rates, the EZDGT algorithm yields better results than those achieved by the EZDCT. For most of the images (“Lena”, “Barbara”, “Girl”), the EZDGT performances better than the EZW. Only for the “Camman” image did the EZDGT performance worse than EZW.
**Lena 512×512**

<table>
<thead>
<tr>
<th>Bit-rate</th>
<th>EZW (Shapiro using 'db5')</th>
<th>EZDCT (QF70 with Qmat)</th>
<th>EZDGT (quality 2000 with Qmat)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000</td>
<td>36.33</td>
<td>35.01</td>
<td>37.03</td>
</tr>
<tr>
<td>0.500</td>
<td>32.88</td>
<td>31.75</td>
<td>32.75</td>
</tr>
<tr>
<td>0.250</td>
<td>29.71</td>
<td>28.14</td>
<td>28.82</td>
</tr>
<tr>
<td>0.125</td>
<td>25.03</td>
<td>24.33</td>
<td>23.98</td>
</tr>
<tr>
<td>0.0625</td>
<td>13.88</td>
<td>14.49</td>
<td>16.76</td>
</tr>
</tbody>
</table>

Table 5.1 Performance comparisons of the EZW, EZDCT, and EZW of the image Lena.

**Barbara 512×512**

<table>
<thead>
<tr>
<th>Bit-rate</th>
<th>EZW (Shapiro using 'db5')</th>
<th>EZDCT (QF70 with Qmat)</th>
<th>EZDGT (quality 2000 with Qmat)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000</td>
<td>30.80</td>
<td>31.23</td>
<td>32.95</td>
</tr>
<tr>
<td>0.500</td>
<td>26.56</td>
<td>26.96</td>
<td>29.06</td>
</tr>
<tr>
<td>0.250</td>
<td>23.52</td>
<td>23.45</td>
<td>25.64</td>
</tr>
<tr>
<td>0.125</td>
<td>21.89</td>
<td>21.49</td>
<td>21.83</td>
</tr>
<tr>
<td>0.0625</td>
<td>13.12</td>
<td>15.83</td>
<td>14.81</td>
</tr>
</tbody>
</table>

Table 5.2 Performance comparisons of the EZW, EZDCT, and EZW of the image Barbara.
## Girl 256 × 256

<table>
<thead>
<tr>
<th>Bit-rate</th>
<th>EZW (Shapiro using ‘db5’)</th>
<th>EZDCT (QF70 with Qmat)</th>
<th>EZDGT (quality 2000 with Qmat)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000</td>
<td>30.86</td>
<td>30.16</td>
<td>31.89</td>
</tr>
<tr>
<td>0.500</td>
<td>27.35</td>
<td>27.78</td>
<td>27.84</td>
</tr>
<tr>
<td>0.250</td>
<td>24.25</td>
<td>24.16</td>
<td>24.33</td>
</tr>
<tr>
<td>0.125</td>
<td>21.41</td>
<td>20.52</td>
<td>21.42</td>
</tr>
</tbody>
</table>

Table 5.3 Performance comparisons of the EZW, EZDCT, and EZW of the image Girl.

## Camman 256 × 256

<table>
<thead>
<tr>
<th>Bit-rate</th>
<th>EZW (Shapiro using ‘db5’)</th>
<th>EZDCT (QF70 with Qmat)</th>
<th>EZDGT (quality 2000 with Qmat)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000</td>
<td>29.57</td>
<td>25.68</td>
<td>27.44</td>
</tr>
<tr>
<td>0.500</td>
<td>25.49</td>
<td>23.03</td>
<td>23.36</td>
</tr>
<tr>
<td>0.250</td>
<td>21.57</td>
<td>20.24</td>
<td>20.07</td>
</tr>
<tr>
<td>0.125</td>
<td>18.61</td>
<td>17.81</td>
<td>17.53</td>
</tr>
</tbody>
</table>

Table 5.4 Performance comparisons of the EZW, EZDCT, and EZW of the image Camman.
5.5 Summary

In this Chapter, we first introduce the implementation of the EZDCT according to Xiong's algorithm using Matlab program, and shown an example of the entire process of the EZDCT. Then we built a quantization matrix for the DOT used in the same way as for the block-based DCT; the quantization matrix is varied based on the size of the image. Using this quantization matrix to implement the EZDOT, we also showed an example of the whole process of the EZDOT. Then we built different codecs for different transform (Wavelet, DCT and DOT), illustrated the reconstructed images at bit-rates equal to 1.00, 0.5, 0.25, 0.125 and 0.0625. The Tables 5.1 to 5.4 shown these results.
Chapter 6. Conclusion and future work

In this work, we successfully formulated the EZDGT and implemented it using the Matlab program. From the results shown in Chapter 5, we can see that most of the time, the EZDGT is better than the EZDCT and sometimes is better than the EZW. This work verifies that the Derivative of Gaussian Transform is one of the members of the family of transforms to which Embedded Zerotree techniques can be successfully applied. We also get a conclusion that the quantization errors introduced by the DGT are less objective than those introduced by the block-based DCT and by the Wavelet Transform. Especially at very low bit rates, sometimes the PSNRs of the reconstructed images of the DGT are lower than that of the DCT and of the Wavelet, but the visual quality is better than them.

In this work, the PSNRs of our EZW are generally 2-3 dB lower than those Shapiro claimed for his EZW algorithm; PSNRs of our ECDCT are even 3-4 dB lower than Xiong et al claimed for their EZDCT algorithm. Since we use same quantization method for different transform (including DGT, DCT and Wavelet Transform); it is reasonable to compare their results. If we improve somewhere in the quantization and entropy coder, the results should be entire shifted to a higher level for every transform.

All of the programs in this work are written using Matlab (See Appendix A). While Matlab is convenient to use, it is often inefficient computationally. To use the EZDGT in practice, an implementation in a more efficient language is needed. This is an area for future work. Another area of investigation is improving the quantization matrix for DGT, to take advantages of the oblique effect.
Appendix A: Matlab Programs of EVDGT

%Embedded Zerotree DGT image encoding/decoding with quantization matrix
% Written by xiangang li
% This function calls the following M-file functions:
% dominantpass.m
% subordinatepass.m
% checkdescents.m
% checkchildren.m
% mapping.m
% GenBasis.m
% Hermite.m
% DG.m
% makeg.m
% svdinv.m
% cfactor.m
% csf.m

[I,Imap]=imread('lena.bmp'); % get initial image
Figure;
image(I);
colormap(Imap);
title('original image');

II=double(I); % convert vector I from uint8 to double;
[dimrow,dimcol]=size(II);
step=dimrow/8; % step size.

ml=8;
derivatives=[0 3 10 21 36 53 74 99]; % DGT derivative orders
sigma=[4.8 3.45 3.45 3.45 3.45 3.45 3.45 3.45]; % variance of the basis functions.
% Choose derivatives and sigma for the DGT basis functions.

[Gaussian,Amatrix]=GenBasis(dimrow,ml,derivatives,sigma);
% Take Derivative Gaussian Transform;
coeffs=Amatrix*II*Amatrix; % get DGT coefficients.
Figure;
image(1000*coeffs);
colormap(Imap);
title('original DGT coefficients image');

% Next step is to form a DCT liked DGT coefficients map.
k=1;
for j=1:step:dimrow,
   for i=1:step:dimcol,
      S{k}=coeffs(j:j+step-1,i:i+step-1);
      k=k+1;
   end
end

101
CC=zeros(step^2,64);

for level=1:64,
    [a,b]=size(S{level});
    k=1;
    for i=1:a,
        for j=1:b,
            CC(k,level)=S{level}(i,j);
            k=k+1;
        end
    end
end

EE=[]; BB=[];
for r=1:step:step^2,
    GG=[];
    for m=r:(r+step-1),
        k=1;
        for i=1:8,
            for j=1:8,
                BB(i,j)=CC(m,k);
                k=k+1;
            end
        end
        GG=[GG,BB];
    end
    EE=[EE;GG]; % EE is DCT-liked DGT coefficients matrix.
end

Figure;
image(200*EE);
colormap(Imap);
title('re-organized DGT coefficients');

% ************************************************%
% Next is to create a quantization matrix for DGT coefficients%
% ************************************************%
end
qmat=ones(8,8);
for m=1:8,
    for n=1:8,
        qmat(m,n)=1/csft(sqrt(rho(m)^2+rho(n)^2));
    end
end
qmat(1,1)=1/csft(cfactor(pixels,distance)*2*pi/8);
fullqmat=repmat(qmat,[dimrow/8 dimcol/8]);
% fullqmat is the DGT quantization matrix

quality=2000; % quality factor
Qcoeffs=round(quality*EE/fullqmat); % DGT coefficients
Figure;
image(10*Qcoeffs);
colormap(Imap);
title('DGT coefficients image after applying fullqmat/quality factor');

% Now we form a wavelet-like tree structure that Embedded
% Zero-tree Quantization method can be applied.**********%

C=[];
for i=1:8:dimrow,
    for j=1:8:dimcol,
        AA=Qcoeffs(i:i+7,j:j+7);
        B=[];
        for l=1:8,
            B=[B, AA(l,1:8)]; % B is a row vector;
        end
        C=[C;B];
    end
end

[m,n]=size(C);
step=sqrt(m);
F0=[];
for i=1:step:m,
    D=C(i:i+step-1,1);
    E=D';
    F0=[F0;E]; % F0 is DC coefficients;
end

F1=[];
for i=1:step:m,
    D=C(i:i+step-1,2);
    E=D';
    F1=[F1;E]; % F1 is AC1 coefficients;
end

F2=[];

103
for i=1:step:m,
    D=C(i:i+step-1,9);
    E=D';
    F2=[F2;E];% F2 is AC2 coefficients;
end

F3=[];
for i=1:step:m,
    D=C(i:i+step-1,10);
    E=D';
    F3=[F3;E];% F3 is AC3 coefficients;
end

F4=[];
for i=1:step:m,
    F=[]; G=[];
    for k=i:i+step-1,
        D=C(k,3:4);
        F=[F;D];
        E=C(k,11:12);
        F=[F;E];
        G=[G,F];
        F=[];
    end
    F4=[F4;G];% F4 is AC4 coefficients;
end

F5=[];
for i=1:step:m,
    F=[]; G=[];
    for k=i:i+step-1,
        D=C(k,17:18);
        F=[F;D];
        E=C(k,25:26);
        F=[F;E];
        G=[G,F];
        F=[];
    end
    F5=[F5;G];% F5 is AC5 coefficients;
end

F6=[];
for i=1:step:m,
    F=[]; G=[];
    for k=i:i+step-1,
        D=C(k,19:20);
        E=D';
        F2=[F2;E];% F2 is AC2 coefficients;
    end

F = [F; D];
E = C(k, 27:28);
F = [F; E];
G = [G, F];
F = [];
end

F6 = [F6; G]; % F6 is AC6 coefficients;
end

F7 = [];
for i = 1:step:m,
F = []; X = []; G = []; H = []; M = []; NN = [];
for k = i:i + step - 1,
D = C(k, 5:6);
E = C(k, 13:14);
F = [D; E]; % coefficients number 17, 18, 19, 20 formed a 2x2 matrix
D = C(k, 7:8);
E = C(k, 15:16);
G = [D; E]; % coefficients number 20, 21, 22, 23 formed a 2x2 matrix
G0 = [F, G]; % form a 2x4 matrix
F = []; G = []; D = C(k, 21:22);
E = C(k, 29:30);
F = [D; E];
D = C(k, 23:24);
E = C(k, 31:32);
G = [D; E];
G1 = [F, G];
NN = [G0; G1]; % N is a 4x4 matrix
X = [X, NN]; % X be a 4x128 matrix
F = []; G = []; G0 = []; G1 = [];
end
F7 = [F7; X]; % F7 is AC7 coefficients;
end

F8 = [];
for i = 1:step:m,
F = []; X = []; G = []; H = []; M = []; NN = [];
for k = i:i + step - 1,
D = C(k, 33:34);
E = C(k, 41:42);
F = [D; E];
D = C(k, 35:36);
E=C(k,43:44);
G=[D;E];
G0=[F,G];

F=[]; G=[];
D=C(k,49:50);
E=C(k,57:58);
F=[D;E];

D=C(k,51:52);
E=C(k,59:60);
G=[D;E];
G1=[F,G];
NN=[G0;G1];
X=[X,NN];
F=[]; G=[]; G0=[]; G1=[];

F8=[F8;X]; % F8 is AC8 coefficients;

end

F9=[];
for i=1:step:m,
F=[]; X=[]; G=[]; H=[]; M=[]; NN=[];
for k=i:i+step-1,
D=C(k,37:38);
E=C(k,45:46);
F=[D;E];

D=C(k,39:40);
E=C(k,47:48);
G=[D;E];
G0=[F,G];
F=[]; G=[];
D=C(k,53:54);
E=C(k,61:62);
F=[D;E];

D=C(k,55:56);
E=C(k,63:64);
G=[D;E];
G1=[F,G];
NN=[G0;G1];
X=[X,NN];
F=[]; G=[]; G0=[]; G1=[];
end
F9=[F9;X]; % F9 is AC9 coefficients;

end
now we use the subband images F0–F9 to form a new matrix,
this matrix will be a 3-scale level wavelet-like coefficients matrix.

AA = [ ]; B = [ ]; B1 = [ ]; D = [ ]; E = [ ]; F = [ ]; G = [ ]; H = [ ]; X = [ ];
AA = [ F0, F1 ];
B = [ F2, F3 ];
B1 = [ AA; B ];
D = [ B1, F4 ];
E = [ F5, F6 ];
F = [ D; E ];
G = [ F, F7 ];
H = [ F8, F9 ];
X = [ G; H ];
initialcoeffs = X; % X is wavelet-like coefficients matrix, same size of original image

Figure; % display wavelet-like image
image(X);
colormap(lmap);
title('wavelet-like DGT coefficients tree-structure image before EZW');

% Now we apply the Embedded Zerotree quantization method to quantize
% the tree-structured coefficients.

Y1 = max(max(abs(X)));
for i=0:20,
    if 2^i <= Y1 & 2^i > Y1/2
        threshold = 2^i;
        initialthreshold = threshold; % get initial threshold T0;
        laststeplevel = i+1; % last step level
        break;
    end
end

sublist = [ ];
sub_list = [ ]; % sub_list is a new position matrix n x 2
    % for all significant coefficients 'p' and 'n';

% This section is to form a Morton scan order vector. The first column is the
% row number, the second column is the column number.
A = mapping(dimrow); % Form a Morton scan order matrix
global N; % Let Morton scan order vector as a global variable
N = zeros(dimrow*dimcol, 2);
for i=1:dimrow,
    for j=1:dimcol,
        N(A(i,j), 1) = i;
        N(A(i,j), 2) = j;
    end
end
Use the EZW method to quantize the coefficients matrix. Each dominant pass and subordinate pass are saved as DD(order), SS(order).

order=1; Bit=0; givenbpp=1.00; % Given bit-rates at your desired.

while threshold ~= 0.5, % if threshold~1, do dominantpass and subordinatepass.

threshold, % display threshold

% Dominant Pass

[D,X,sublist,sub_list] = dominantpass(X,threshold,sublist,sub_list);

DD(order)=D % display symbols of dominant-pass.

significantlist(order)=sub_list;

% Now calculate the entropy, bits number, and the bits rate for each dominant pass to check the bit rate satisfies given bit rate or not%

[m,n]=size(DD(order));

pp=0;nn=0;zz=0;tt=0;

pentropy=0;nentropy=0;

zentropy=0; tentropy=0;

bitrate=Bit;

for i=1:n,

if DD(order)(i)=='p'
    pp=pp+1;
    if pp~=0
        pentropy=(pp/n)*log2(pp/n);
    end

end

if DD(order)(i)=='n'
    nn=nn+1;
    if nn~=0
        nentropy=(nn/n)*log2(nn/n);
    end

end

if DD(order)(i)=='z'
    zz=zz+1;
    if zz~=0
        zentropy=(zz/n)*log2(zz/n);
    end

end

if DD(order)(i)=='t'

end

if DD(order)(i)=='t'

end

end
tt = tt + 1;
if tt = 0
    tentropy = -(tt/n)*log2(tt/n);
end
end

D_entropy(order) = pentropy + nentropy + zentropy + tentropy;

D_bitsnum(order) = D_entropy(order)*n;

% Now determine whether the bpp reaches the given value.
bpp = D_bitsnum(order)/(dimrow*dimcol);

bitrate = Bit + bpp;

if bitrate >= givenbpp
    DD(order) = DD(order)(1:i);
    SS(order) = 0;
    B_pp(order) = bitrate; % if bit rate reaches the given bit rate, stop quantization.
    break;
end
end

Bit = bitrate;
if bitrate >= given bpp % if bit rate reaches the given bit rate, stop quantization
    break;
end

% Subordinate pass

threshold = threshold/2;
if threshold == 0.5 % when threshold reached 1, there is no subordinate pass
    B_pp(order) = bitrate;
    break;
end

S = subordinatepass(sublist, threshold);
SS(order) = S

% Subordinate pass

if order == laststeplevel % if threshold == 1, there is no subordinate pass
    bitsnum(order) = D_bitsnum(order);
    bpp = bitsnum(order)/(dimrow*dimcol);
    B_pp(order) = Bit % display the bpp
    break;
end
% Now calculate the entropy, bits number, and the bits rate for each subordinate
% pass to check the bit rate satisfies given bit rate or not
%
% [a,b]=size(SS{order});
one=0;
zero=0;
for i=1:b,
    if SS{order}(i)==1,
        one=one+1;
    else zero=zero+1;
    end

    if one==0;
        S_entropy{order} = -(zero/b)*log2(zero/b);
    elseif zero==0;
        S_entropy{order} = -(one/b)*log2(one/b);
    else
        S_entropy{order} = -(zero/b)*log2(zero/b)-(one/b)*log2(one/b);
    end

    S_bitsnum{order}= S_entropy{order}*b;

    bitsnum{order}= S_bitsnum{order};
bpp=bitsnum{order}/(dimrow*dimcol);

    bitrate=Bit+bpp;

    if bitrate>=givenbpp
        SS{order}=SS{order}(1:i);
        B_pp(order)=bitrate;
        break;
    end
end

if bitrate>=givenbpp % if bit rate equals the given bit rate, stop quantization
    break;
end

Bit=bitrate;
B_pp(order)=Bit

order=order+1;

end

if order<laststeplevel
    B_pp(order)=Bit;
    laststeplevel=order;
end
XX=zeros(size(X)); % initialize the reconstructed image to zero; threshold=initialthreshold; % get initial threshold

for level=1:laststeplevel,
    RR=zeros(size(X)); % reference matrix RR;
    [a,b]=size(DD{level});
    % dominant pass decoder
    i=1; j=1;
    while j<=b,
        if RR(N(i,1),N(i,2)==0
            if DD{level}(j)=='p'
                if threshold==1
                    XX(N(i,1),N(i,2))=threshold;
                else
                    XX(N(i,1),N(i,2))=1.5*threshold;
                end
            end
            if DD{level}(j)=='n'
                if threshold==1
                    XX(N(i,1),N(i,2))=-threshold;
                else
                    XX(N(i,1),N(i,2))=-1.5*threshold;
                end
            end
            if DD{level}(j)=='t' & A(N(i,1),N(i,2)<=dimrow*dimcol/4
                RR=checkchildren(i,RR); % all Zerotree's descendants are set to 1.
            end
        RR(N(i,1),N(i,2))=1; % reference matrix =1;
        i=i+1;
        j=j+1;
    else i=i+1;
% subordinate pass

[xx,yy]=size(significantlist{level});
threshold=threshold/2;

for i=1:xx,
    if level==laststeplevel|threshold==0.5
        break;
    end
    if SS{level}(i)==1
        if XX(sub_list(i,1),sub_list(i,2))>0;
            XX(sub_list(i,1),sub_list(i,2))= fix(XX(sub_list(i,1),sub_list(i,2))+ threshold/2);
        else
            XX(sub_list(i,1),sub_list(i,2))= fix(XX(sub_list(i,1),sub_list(i,2))-threshold/2);
        end
    end
    if SS{level}(i)==0
        if XX(sub_list(i,1),sub_list(i,2))>0;
            XX(sub_list(i,1),sub_list(i,2))= fix(XX(sub_list(i,1),sub_list(i,2))-threshold/2);
        else
            XX(sub_list(i,1),sub_list(i,2))= fix(XX(sub_list(i,1),sub_list(i,2))+threshold/2);
        end
    end
end

LL=XX-initialcoeffs;
level
threshold
Maxerror=max(max(LL))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Next program will transform the wavelet-like coefficients to initial coefficients;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Go0=XX(1:dimrow/8,1:dimrow/8);
Go1=XX(1:dimrow/8,(dimrow/8+1):dimrow/4);
Go2=XX((dimrow/8+1):dimrow/4,1:dimrow/8);
Go3=XX((dimrow/8+1):dimrow/4,(dimrow/8+1):dimrow/4);
Go4=XX(1:dimrow/4,(dimrow/4+1):dimrow/2);
Go5=XX((dimrow/4+1):dimrow/2,1:dimrow/4);
Go6=XX((dimrow/4+1):dimrow/2,(dimrow/4+1):dimrow/2);
Go7=XX(1:dimrow/2,(dimrow/2+1):dimrow);
Go8=XX((dimrow/2+1):dimrow,1:dimrow/2);
Go9=XX((dimrow/2+1):dimrow,(dimrow/2+1):dimrow);

CC=zeros(stepA2,64);

[a,b]=size(Go0); % Go0 is 64x64 matrix
k=1;
for i=1:a,
    for j=1:b,
        CC(k,1)=Go0(i,j); % DC, column 1
        k=k+1;
    end
end

[a,b]=size(Go1);
k=1;
for i=1:a,
    for j=1:b,
        CC(k,2)=Go1(i,j); % AC1, column 2
        k=k+1;
    end
end

[a,b]=size(Go2);
k=1;
for i=1:a,
    for j=1:b,
        CC(k,9)=Go2(i,j); % AC2, column 3
        k=k+1;
    end
end

[a,b]=size(Go3);
k=1;
for i=1:a,
    for j=1:b,
        CC(k,10)=Go3(i,j); % AC3, column 4
        k=k+1;
    end
end
end
end

[a,b]=size(G04);% Go4 is 128x128 matrix
k=1;
for i=1:2:a,
    for j=1:2:b,
        CC(k,3)=G04(i,j); % column 5,6,7,8
        CC(k,4)=G04(i,j+1);
        CC(k,11)=G04(i+1,j);
        CC(k,12)=G04(i+1,j+1);
        k=k+1;
    end
end
end

[a,b]=size(G05);% Go5 is 128x128 matrix
k=1;
for i=1:2:a,
    for j=1:2:b,
        CC(k,17)=G05(i,j); % column 9,10,11,12
        CC(k,18)=G05(i,j+1);
        CC(k,25)=G05(i+1,j);
        CC(k,26)=G05(i+1,j+1);
        k=k+1;
    end
end
end

[a,b]=size(G06);% Go6 is 128x128 matrix
k=1;
for i=1:2:a,
    for j=1:2:b,
        CC(k,19)=G06(i,j); % column 13,14,15,16
        CC(k,20)=G06(i,j+1);
        CC(k,27)=G06(i+1,j);
        CC(k,28)=G06(i+1,j+1);
        k=k+1;
    end
end
end

[a,b]=size(G07);% Go7 is 256x256 matrix
k=1;
for i=1:4:a,
for j=1:4:b,
    CC(k,5)=G07(i,j); % column 17,...,32
    CC(k,6)=G07(i,j+1);
    CC(k,13)=G07(i+1,j);
    CC(k,14)=G07(i+1,j+1);
    CC(k,7)=G07(i,j+2);
    CC(k,8)=G07(i,j+3);
    CC(k,15)=G07(i+1,j+2);
    CC(k,16)=G07(i+1,j+3);
    CC(k,21)=G07(i+2,j);
    CC(k,22)=G07(i+2,j+1);
    CC(k,23)=G07(i+2,j+2);
    CC(k,24)=G07(i+2,j+3);
    CC(k,29)=G07(i+3,j);
    CC(k,30)=G07(i+3,j+1);
    CC(k,25)=G07(i+3,j+2);
    CC(k,26)=G07(i+3,j+3);
    CC(k,29)=G07(i+3,j+2);
    CC(k,30)=G07(i+3,j+3);
    CC(k,31)=G07(i+3,j+2);
    CC(k,32)=G07(i+3,j+3);
    k=k+1;
end
end

[a,b]=size(G08); % G08 is 256x256 matrix
k=1;
for i=1:4:a,
    for j=1:4:b,
        CC(k,33)=G08(i,j); % column 33,...,48
        CC(k,34)=G08(i,j+1);
        CC(k,41)=G08(i+1,j);
        CC(k,42)=G08(i+1,j+1);
        CC(k,35)=G08(i,j+2);
        CC(k,36)=G08(i,j+3);
        CC(k,43)=G08(i+1,j+2);
        CC(k,44)=G08(i+1,j+3);
        CC(k,49)=G08(i+2,j);
        CC(k,50)=G08(i+2,j+1);
        CC(k,57)=G08(i+3,j);
        CC(k,58)=G08(i+3,j+1);
        CC(k,51)=G08(i+2,j+2);
        CC(k,52)=G08(i+2,j+3);
        CC(k,59)=G08(i+3,j+2);
        CC(k,60)=G08(i+3,j+3);
        k=k+1;
    end
end

115
\[a,b]=\text{size}(Go9);\text{ }% \text{ Go9 is 256x256 matrix}
\]
\[k=1;\]
\[\text{for } i=1:4:a,\]
\[\text{ for } j=1:4:b,\]
\[\text{ CC}(k,37)=Go9(i,j); \text{ }% \text{ column 49, \ldots, 64}\]
\[\text{ CC}(k,38)=Go9(i,j+1);\]
\[\text{ CC}(k,45)=Go9(i+1,j);\]
\[\text{ CC}(k,46)=Go9(i+1,j+1);\]
\[\text{ CC}(k,39)=Go9(i,j+2);\]
\[\text{ CC}(k,40)=Go9(i,j+3);\]
\[\text{ CC}(k,47)=Go9(i+1,j+2);\]
\[\text{ CC}(k,48)=Go9(i+1,j+3);\]
\[\text{ CC}(k,53)=Go9(i+2,j);\]
\[\text{ CC}(k,54)=Go9(i+2,j+1);\]
\[\text{ CC}(k,61)=Go9(i+3,j);\]
\[\text{ CC}(k,62)=Go9(i+3,j+1);\]
\[\text{ CC}(k,55)=Go9(i+2,j+2);\]
\[\text{ CC}(k,56)=Go9(i+2,j+3);\]
\[\text{ CC}(k,63)=Go9(i+3,j+2);\]
\[\text{ CC}(k,64)=Go9(i+3,j+3);\]
\[k=k+1;\]
\[\text{ end}\]
\[\text{ end}\]

\%Next step is recovered initial image from a 1024x64 vector CC
\% Inverse DCT transform

\[YY= [];\text{ Beta} = [];\]
\[\text{ for } r=1:\text{ step} : \text{ step} \times 2,\]
\[\text{ Game} = [];\]
\[\text{ for } m=r:(r+\text{ step}-1),\]
\[\text{ k}=1;\]
\[\text{ for } i=1:8,\]
\[\text{ Beta}(i,j)=\text{ CC}(m,k);\]
\[\text{ k}=k+1;\]
\[\text{ end}\]
\[\text{ end}\]
\[\text{ Game} = [\text{ Game}, \text{ Beta}]; \text{ }% \text{ GG is a 8xdimrow matrix}\]
\[\text{ end}\]
\[\text{ YY} = [\text{ YY}, \text{ Game}]; \text{ }% \text{ EE is the reconstructed image dimrow*dimcol.}\]
\[\text{ end}\]

116
YYY = (YY/quality) * fullqmat;

Cell = [];
for i = 1:8:dimrow,
  for j = 1:8:dimcol,
    AA = YYY(i:i+7:j:j+7);
    B = [];
    for l = 1:8,
      B = [B, AA(1, 1:8)]; % B is a row vector size 1x64;
    end
    Cell = [Cell; B];
    % C is a new formed coefficients matrix, size 4096x64;
  end
end

for j = 1:64,
  V(j) = [];
  for i = 1:step:m,
    D = Cell(i:i+step-1, j);
    E = D';
    V(j) = [V(j); E]; % F(j) is DC coefficients, size 64x64;
  end
end

Rcoeffs = [];
for i = 1:8:64,
  aa = [];
  for j = i:i+7,
    aa = [aa, V(j)];
  end
  Rcoeffs = [Rcoeffs; aa];
end

VV = Gaussian * Rcoeffs * Gaussian'; % Inverse DGT transform;
Rimage{level} = VV; % store reconstructed image of each level

Figure; % display the recovered image.
image(VV);
colormap(Imap);
title('reconstructed DGT image');
image_max_error = max(max(abs(VV - II)));

error = VV - II;
decibels = 20*log10(255/(sqrt(mean(mean(error.^2)))));
disp(sprintf('PSNR = + %5.2f dB', decibels))
dB(level) = decibels
B_pp  %display bit rate
Figure;
hold on;
xlabel('bits per pixel (bpp)');
ylabel('PSNR (dB)');
title('bpp vs PSNR of reconstructed image barbara.bmp');
plot(B_pp,dB,'*'); % display the bit per pixel vs. PSNR of reconstructed images.
grid on;

%*********************************************************
%**********Now calculate the significant coefficients*****%
%*********************************************************
[m,n]=size(DD{level});
pp=0;nn=0;
for i=1:n,
    if DD{level}(i)=='p'
        pp=pp+1;
    end
    if DD{level}(i)=='n'
        nn=nn+1;
    end
end
signcoeff=2*b+pp+nn; % significant coefficients.
disp(sprintf('Significant coefficients = %5.0f',signcoeff))
function [D,X,sublist,sub_list] = dominantpass(X,threshold,sublist,sub_list)
% This is Dominant-pass function
% Usage: [D,X,sublist,sub_list] = dominantpass(X,threshold,sublist,sub_list)
% X is the coefficients matrix.
% This function will scan all the coefficients using a predefined scan
% order. The significant coefficients' values will append to the Subordinate
% List array sublist, their positions will put in array sub_list, with
% respect to current threshold. The coefficients' output sign
% 'p','n','z' and 't'
% written by xiangnag Li.
% This function calls the following M-file functions:
% checkdescendants1.m
% checkchildren.m

D=[];
global N;
[m,n]=size(X);
% X is the coefficients matrix
R=zeros(m); % matrix R is a reference matrix, same size as X; '0' means
% this coefficient is not a descendant from Zerotree root;
[a,b]=size(N);

if abs(X(1,1))>=threshold % X(1,1) is DC coefficient
    sublist=[sublist, abs(X(1,1))]; % put significant coefficients's value to sublist
    sub_list=[sub_list;N(1,1),N(1,2)]; % put the significant coefficients' position in sub_list
    if X(1,1)>0
        D=[D,'p'];
    else D=[D,'n'];
    end
    X(1,1)=0;
else D=[D,'z'];
end

for k=2:4,
    if abs(X(N(k,1),N(k,2)))>=threshold,
        sublist=[sublist, abs(X(N(k,1),N(k,2)))); % append this significant coefficient to the subordinate list;
        sub_list=[sub_list;N(k,1),N(k,2)];
        if X(N(k,1),N(k,2))>0 % determine the sign
            D=[D,'p']; % >0, assign a "p"
        else D=[D,'n'];% <0, assign a "n"
        end
    end
end
\[ X(N(k, 1), N(k, 2)) = 0; \]

% the significant coefficients is replaced by a '0' in the coefficients matrix
else
  % 2,3,4 has no parents, just check its descendants.
  result = checkdescendants1(k, X, threshold, 0);
  if result == 1
    D = [D, 'z'];
  else
    D = [D, 't'];
    R(N(k, 1), N(k, 2)) = 1; % Zerotree, make all its descendants
    R = checkchildren(k, R); % reference matrix component to 1.
  end
end
end

for k = 5:a,
  if abs(X(N(k, 1), N(k, 2))) >= threshold,
    sublist = [sublist, abs(X(N(k, 1), N(k, 2)))];
    sub_list = [sub_list; N(k, 1), N(k, 2)];
    if X(N(k, 1), N(k, 2)) > 0, % determine the sign
      D = [D, 'p']; % >0, assign a 'p'
    else
      D = [D, 'n']; % <0, assign a 'n'
    end
    X(N(k, 1), N(k, 2)) = 0;
  elseif R(N(k, 1), N(k, 2)) == 0
    result = checkdescendants1(k, X, threshold, 0);
    % Check it has significant descendants?
    if result == 1,
      D = [D, 'z']; % isolated zero
    else
      D = [D, 't']; % Zerotree
      R(N(k, 1), N(k, 2)) = 1;
      R = checkchildren(k, R);
    end
    % if Zerotree, reference matrix coefficients=1
  end
end
function S = subordinatepass(sublist, threshold)
% This is subordinatepass function.
% Usage: S = subordinatepass(sublist, threshold)
% Subordinate function will create the subordinate array
% S with respect to the current threshold.
% Written by xiangang Li.

S=[];

[m,n]=size(sublist);
for i=1:n,
    if bitand(sublist(1,i), threshold) == threshold
        S=[S, 1];
    else
        S=[S, 0];
    end
end
function result = checkdescendants (j,X,threshold,result)
% This is function checkdescendant1.
%
% Usage: result = checkdescendant1( k,X,threshold,0)
% X is the coefficients matrix, parameter k is scan order sequence.
%
% This function will check whether a coefficient has a significant
% descendant or not. The initial result is set to 0, if the result=1,
% means that kth coefficient has at least one significant descendant.
%
% Written by xiangang Li.
%
% This function calls:
% chenkdescendant.m

global N
% Morton scan order vector N is a global variable
[m,n]=size(N);

for i=(4*j-3):4*j;
    if result=1 | i>m,
        break;
    end

    if abs(X(N(i,1),N(i,2)))>=threshold
        result=1;
        break;
    else
        result=checkdescendants1(i,X,threshold,result);
    end
end
function RR=checkchildren(j,RR)
% This is checkchildren function.
%
% Usage: R=checkchildren(k,R)
%
% When a symbol 't' (Zerotree) is encountered, this function will make
% all the values of this kth coefficient's descendants in the reference
% matrix R to 1. Then these coefficients do not need scan again.
%
% Written by xiangang Li.
%
% This function calls:
% checkchildren.m

global N
[m,n]=size(N);

for i=(4*j-3):4*j,
    if i<=m
        RR(N(i,1),N(i,2))=1;
        RR=checkchildren(i,RR);
    end
end

function A = mapping(n)
% This is mapping function
%
% Usage: A=mapping(n)
%
% To create a n*n Morton Scan order matrix, the number
% n should equal to the value of the power of 2.
%
% Written by xiangang Li.
%
% This function calls:
% mapping.m

if n == 2
    A = [1 2; 3 4];
else
    B = mapping(n/2);
    A = [B B+(n/2)^2; B+(n/2)^2*2 B+(n/2)^2*3];
end
function [G, A] = GenBasis(dim, m1, derivatives, sigma)
% This is function GenBasis
% 
% Usage: [G, A] = GenBasis(dim, m1, derivatives, sigma)
% 
% This function will generate a synthesis basis (G) and an
% analysis basis (A). The basis functions of the synthesis
% basis are described by the variables dim and m1 and by the
% vectors derivatives and sigma. See 'help makeg' for details
% of these variables.
% 
% Written by Jeffrey A Bloom
% 
% This function calls:
% makeg.m
% svdinvrt.m

% Calculate the G matrix
G = makeg(dim, m1, derivatives, sigma);
% Calculate the A matrix
disp('Inverting G to create A')
[gtgi,CN] = svdinvrt(G'*G);
CNstring=[' Condition number is ',num2str(CN)];
disp(CNstring);
A = ( gtgi'G' )';
function [g]=DG(param)
% DG - Derivative of Gaussian function
%
% Usage: g = DG(param)
%
% param = [n,dim,sigma,loc]
% This function creates a function which is the nth derivative
% of a Gaussian of width sigma. The function is dim long and is
% centered at loc. Default values are n=0, dim=256, sigma=3,
% and loc=(dim+1)/2
%
% Written by Jeffrey A Bloom
%
% This function calls the following M-file functions:
% Hermite.m

default=[0,256,3,128.5];
if (nargin<1)
    param=default;
elseif (size(param,2)<3)
    param(size(param,2)+1:3) = default(size(param,2)+1:3);
end
if (size(param,2)<4)
    param(4) = (param(2)+1)/2;
end

n = param(1);
dim = param(2);
sigma = param(3);
loc = param(4);

globalx = 1:dim;
x = globalx-loc;

g = polyval(Hermite(param),x).* exp(- x.^2/(2*sigma^2));
% plot(x,g);
% ff=fft(g)
% Figure;
% omiga=0:0.01:pi;
% plot(omiga,ff);

return

%end function
function [g]=Hermite(param)  
% Hermite - Hermite Polynomial  
%  
% Usage: g = Hermite(param)  
%  
% param = [n,dim,sigma,loc]  
% This function returns the coefficients of the nth order Hermite  
% polynomial generated from a Gaussian of width sigma.  
% dim is not used.  
%  
% Written by Jeffrey A Bloom  

n   = param(1);  
sigma = param(3);  
p = 1;  

for k=1:n  
    q = polyder(p);  
    q = [zeros(1,(k+1)-size(q,2)),q];  
    r = [p,0];  
    p = q - (1/sigma^2) * r;  
end % for  

  g = p;  
return  
%end function
function [G]=makeg(dim, m1, derivatives, sigma)

% Usage: G = makeg(dim, m1, derivatives, sigma)
%
% makeg will create a (dim x dim) DGT matrix with basis functions in the columns. 
% There are m1 basis functions at each location. 'derivatives' and 'sigma' are
% length m1 vectors specifying the derivative orders of each basis function at a
% location and the standard deviations of the Gaussians from which each derivative
% is generated. A good basis set, with m1 = 8, is:
% derivatives = [ 0 3 10 21 36 53 74 99];
% sigma = [3.4 3.4 3.4 3.4 3.4 3.4 3.4 3.4]' 
%
% This set of m1 basis functions is then shifted to dim/m1 uniformly distributed
% locations.
%
% Written by Jeffrey A Bloom.
%
% This function calls the following M-file functions:
% DG.m

%---------------------
% General variables
%---------------------

disp(' makeg called with sigma ');disp(sigma);
disp(' and orders ');disp(derivatives);

offset=(m1-l)/2;
longdim=2*dim-l;
param= [0,longdim,0,(longdim+1)/2+offset]; % initial value 
% param = [n,dim,sigma,loc]
locations = (0:(dim/m1-l»*m1 + 1 + offset;

%---------------------
% Generate the G matrix
%---------------------
%
% There are m1 basis functions which are then shifted to get the rest.
% LongG are the m1 functions, doubly long to allow proper shifting

LongG = zeros(longdim,m1);
for j=1:m1,
  param(3) = sigma(j);
  param(1) = derivatives(j);
  LongG(:,j) = DG(param)';
end

for j=1:m1,
  for i=1:dim/m1
    start = (longdim+1)/2 - (i-1)*m1;
    G(:,(j-l)*dim+1+i) = LongG(start:start+dim-1,j);
  end
end
% Normalize the columns of G

disp(' Normalizing G')
Gmax = max(G);   % This returns a row vector with each element the max of
    % the corresponding column in G
G = G./Gmax(ones(dim,1),:)*10;    % Now all columns of G have a max of 10. 

return
% end function
function [A,CN]=svdinvert(B)

% Usage: [A] = svdinvert(B)
% This function performs an SVD decomposition on the matrix B. The
% inverse is returned in A.
% [A,CN] = svdinvert(B)
% The condition number is also calculated and returned in CN.
%
% Written by Jeffrey A Bloom

disp('svdinvert')

MaxCondNum = 10e10; % largest allowed condition number

[U,S,V]=svd(B); % calculate the SVD

% B = U*S*V' Where S is a diagonal matrix with diagonal elements s.
% A = B^(-1) = V*T*U' Where T is a diagonal matrix with diagonal elements 1/s.

maxs = max(diag(S)); % largest eigenvalue
mins = min(diag(S)); % smallest eigenvalue
minw = maxs/MaxCondNum; % smallest allowed eigenvalue
smallw = find(diag(S)<minw); % find any zero eigenvalues

if (~isempty(smallw))
    disp('SVDINVERT WARNING: This matrix is too poorly conditioned for accurate results!')
    newvec = diag(S); % get diag
    newvec(smallw) = inf*ones(size(smallw)); % replace zeros with infinity
    S = diag(newvec); % replace S with new diag matrix
end

T = diag(1./diag(S));
A = V*T*U';

if nargout==2
    CN = maxs/mins;
end

return

% end function
function H=csf(rho)
% csf-contrast sensitivity function.
%
% Usage: H=csf(rho)
% rho is cfactor.
%
% Written by Dr.Todd.R.Reed

A=2.6;
alpha=.0192;
rho0=8.772;
beta=1.1;
H=A*(alpha + ( rho/rho0))\*exp(-(rho/rho0)^beta);

function CF=cfactor(pixels,distance)
% CF-contrast frequency.
%
% Usage: CF=cfactor(pixels, distance)
%
% This function will calculate the radical frequency rho
% in formula (5.6) of the thesis.
%
% Written by Dr.Todd.R.Reed

pitch=.00031; % dot pitch=0.31mm
h=pixels*pitch; % pixel=512 in special case
theta=2*atan(h/(2*distance));
deg=theta*360/(2*pi);
ppd=pixels/deg;
CF=ppd/(2*pi);
References


