Image compression using second generation lifting scheme

Aman Verma (071137)
Puneet Taneja (071349)

Under the supervision of
Mrs. Meenakshi S. Arya

May- 2011

Submitted in partial fulfillment
Of the requirements for the degree of
BACHELOR OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
WAKNAGHAT
SOLAN, HIMACHAL PRADESH

TALBLE OF CONTENTS

Chapter No.	Topics	Page No.
	Certificate from the Supervisor	V
	Acknowledgement	VI
	Summary (Not more than 250 words)	VII
	List of Figures	VIII
Chapter-1	**Image -An introduction**	
	1.1 What is an Image?	page 1
	1.2 Compression overlook	page 1
	1.3 What is compression?	page 2
	1.4 Lossy	page 3
	1.5 Lossless	page 4
	1.6 Types Of encoding	page 4
	1.6.1 Huffman	page 4
	1.6.2 Arithmetic	page 6
	1.6.3 Limpel -Ziv -Welch	page 7
	1.6.4 JPEG	page 10
	1.6.5 JPEG 2000	page 11
Chapter-2	**Literature Review**	
	[1] Wavelets for computer graphics	page 14
	[2] Construction Of Second Generation Wavelets	page 14
	[3] Image Compression by Linear Splines over Adaptive Triangulations	page 14
	[4] Lifting Filter Using Second Generation Wavelets	page 15

[5] A New Robust Data Compressor page 15

Chapter- 3 **Wavelet**
Overview page 16

3.1 Wavelets in one dimension page 16
3.1.1 One-dimensional Haar wavelet transform page 16

3.2 Image Approximation page 18

3.3 Wavelets Transform page 19

3.4 Second Generation Wavelets page 20

3.5 Lifting Scheme page 21
3.5.1 Basic Lifting Scheme Wavelets page 21
3.5.2 Predict Wavelets page 22

3.5.3 The update step page 22

Chapter- 4 **Implementation and coding**

4.1 The concept of the algorithm
4.1.1 Module 1 page 24
4.1.2 Module 2 page 25

4.2 The Terminology of The Algorithm Explained

4.2.1 Module 1 page 26
4.2.2 Module 2 page 28

4.1 Normal application
4.1.1 Main program: page 30
4.1.2 Results and comparisons page 30

4.2 Modified application
4.2.1 Main program: page 31
4.2.2 Functions used page 31
4.2.3 Results and comparisons page 34

4.3 Modified application-2
4.3.1 Main program: page 35

4.3.2 Functions used page 35
4.3.3 Results and comparisons page 39

4.4 Modified application-3
4.4.1 Main program: page 40
4.4.2 Functions used page 41
4.4.3 Results and comparisons page 44

4.5 Conclusion page 45

Bio data page 46

References page 47

CERTIFICATE

This is to certify that the work titled **Image Compression Using Second Generation Lifting Scheme** submitted by **Aman verma (071137) and Puneet Taneja (071349)** in partial fulfillment for the award of degree of. **B. Tech**, of Jaypee University of Information Technology, Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor	[illegible]
Name of Supervisor	Mrs. Meenakshi Arya
Designation	Lecturer
Date	25.05.2011

ACKNOWLEDGMENT

We would like to thank our project guide Mrs. Meenakshi S. Arya who showed us the way to get the job done, not providing the exact way to do it, but the concept behind the complexities so that we can make best of our knowledge and build up higher skills to meet the IT industry needs. However, at some places, where we could not find the solution or were unable to adopt the better solution our self she taught us the work done in efficient and best manner and given up explored details about the problem and the solution.

Her technical help and goal oriented approach has been unique and a stepping stone towards the successful understanding of our project.

Signature of the Students	[illegible]	Puneet
Name of Students	Aman verma (071137)	Puneet Taneja (071349)
Date	23.05.2011	23.05.2011

SUMMARY

The approach to our project is simple and straightforward. We did a thorough survey of the existing techniques in the field of image compression and tried to find out the scope for improvement in the same. The existing techniques have been implemented and their results analyzed. The shortcomings in the existing techniques have been removed by developing a new algorithm which is found to give better results, thus providing novelty.

The platform that we have chosen to do this is MATLAB as it provides us with the basic functions to operate on the image by converting it into matrices. We have implemented trivial compression techniques, done meticulous study of the latest technology available, tried different approach towards the problem and finally have come up with approach that are proven to be much better than what was being achieved.

The present algorithm derived it's inspiration from JPEG 2000 which is implemented using the wavelet transform. We have modified the currently existing compression functions and merged them in a way to obtain results which are superior to the ones already existing. We tested our results with different mathematical functions that gave us different result and better compression.

[illegible], Puneet

Signature of Students

Names 23.05.2011

Date Aman Verma, Puneet Taneja

[illegible]

Signature of Supervisor

Name 23·05·2011

Date Meenakshi S Arya

List of figures

Figure no.	Figure name	Page no.
Figure 1.1	General Compression System Model	1
Figure 1.2	Block Representation of Compression Techniques	2
Figure 1.3	Lossy Compression	3
Figure 1.4	A Bottom-Up Approach	5
Figure 1.5	JPEG 2000	12
Figure 1.6	Wavelet Transforms Using Lifting Scheme	12
Figure 3.1	Filter Analysis	17
Figure 3.2	A 3 Level Filter Bank	18
Figure 3.3	Wavelet Transform	19
Figure 3.4	Second Generation Wavelets	20
Figure 3.5	Lifting Scheme Inverse Transform	21
Figure 3.6	Lifting scheme forward transform	23

Chapter 1

Image An introduction

1.1 What is an Image?

An image is a rectangular grid of pixels. It has a definite height and a definite width counted in pixels. Each pixel is square and has a fixed size on a given display. Each pixel has a color. The color is a 32-bit integer. The first eight bits determine the redness of the pixel, the next eight bits the greenness, the next eight bits the blueness, and the remaining eight bits the transparency of the pixel.

1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1
Transparency	Red	Green	Blue

Table 1.1 – Pixel representation of an image

Each of these values can be interpreted as an unsigned byte between 0 and 255. Within the color higher numbers are brighter. Thus a red of 0 is no red at all while a red of 255 is a very bright red. Different colors are made by mixing different levels of the three primary colors. For example, medium gray is 127 red, 127 green and 127 blue.

1.2 Compression overlook

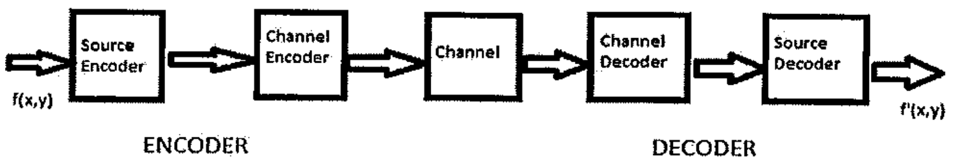


Figure 1.1 – General compression system model

1.3 What is compression?

Image compression is the application of data compression. The objective is to reduce redundancy of the image data in order to be able to store or transmit data in an efficient form. Image compression addresses the problem of reducing the amount of data required to represent a digital image. The underlying basis of the reduction process is the removal of redundant data. From a mathematical viewpoint, this amounts to transforming a 2-D pixel array into a statistically uncorrelated data set.

The transformation is applied prior to storage or transmission of the image. At some later time, the compressed image is decompressed to reconstruct the original image or an approximation of it. Interest in image compression dates back more than 35 years. The initial focus of research efforts in this field was on the development of analog methods for reducing video transmission bandwidth, a process called bandwidth compression. The advent of the digital computer and subsequent development of advanced integrated circuits, however, caused interest to shift from analog to digital compression approaches. The field has undergone significant growth through the practical application of the theoretic work that began in the 1940s, when C. E. Shannon and others first formulated the probabilistic view of information and its representation, transmission, and compression.

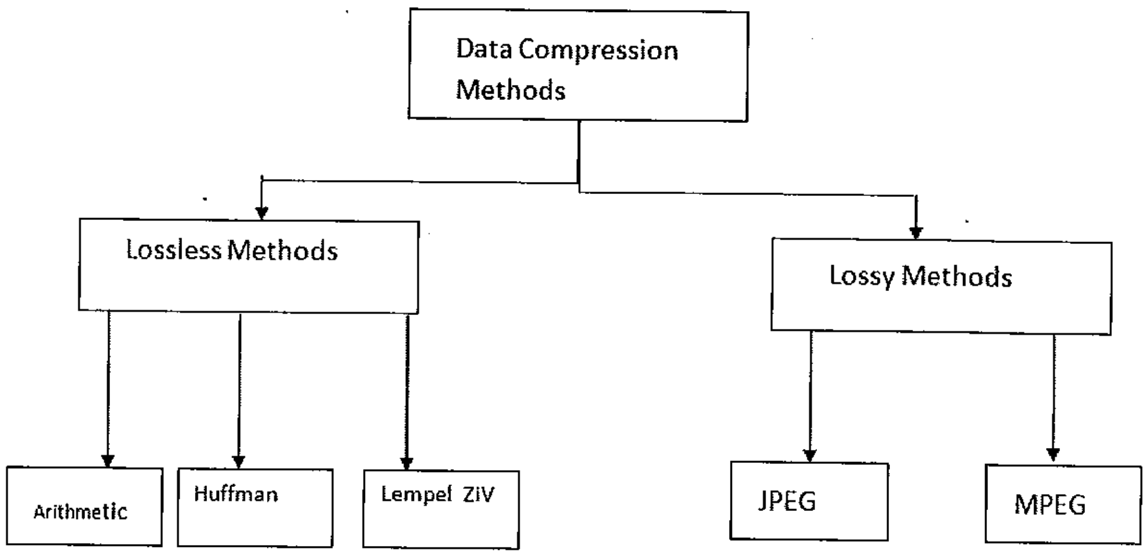
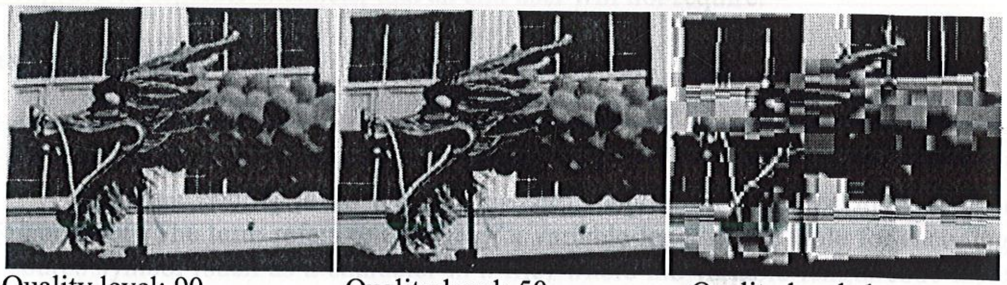


Figure 1.2 – Graphical representation of compression methods

Compression techniques fall into two broad categories: information preserving and lossy. Methods in the first category, which are particularly useful in image archiving (as in the storage of legal or medical records). These methods allow an image to be compressed and decompressed without losing information. Lossy image compression is useful in applications such as broadcast television, videoconferencing, and facsimile transmission, in which a certain amount of error is an acceptable trade-off for increased compression performance.

1.4 Lossy

In information technology, **"lossy" compression** is a data encoding method which compresses data by discarding (losing) some of it. The procedure aims to minimize the amount of data that need to be held, handled, and/or transmitted by a computer. The different versions of the photo of the dog at the right demonstrate how much data can be dispensed with, and how the pictures become progressively coarser as the data that made up the original one is discarded (lost). Typically, a substantial amount of data can be discarded before the result is sufficiently degraded to be noticed by the user.



Quality level: 90	Quality level: 50	Quality level: 1
File size: 10,582 bytes	File size: 5,154 bytes	File size: 923 bytes

Figure 1.3- Representation of lossy compression techniques

Lossy compression is most commonly used to compress multimedia data (audio, video, still images), especially in applications such as streaming media and internet telephony. By contrast, lossless compression is required for text and data files, such as bank records, text articles, etc. In many cases it is advantageous to make a master lossless file which can then be used to produce compressed files for different purposes; for example a multi-megabyte file can be used at full size to produce a full-page advertisement in a glossy magazine, and a 10 kilobyte lossy copy made for a small image on a web page.

Lossy compression formats suffer from generation loss: repeatedly compressing and decompressing the file will cause it to progressively lose quality. This is in contrast with lossless data compression, where data will not be lost via the use of such a procedure.

Information-theoretical foundations for lossy data compression are provided by rate-distortion theory. Much like the use of probability in optimal coding theory, rate-distortion theory heavily draws on Bayesian estimation and decision theory in order to model perceptual distortion and even aesthetic judgment.

1.5 Lossless

Lossless compression is a form of compression in which data files are split up into different chunks and reorganized to optimize them. This sort of compression very rarely saves much space, but it is ideal for transporting enormous files by breaking them into easier-to-handle pieces. Lossless compression is used when every bit of data is needed in the end product, often when transmitting a file to a designer. In the case of images, a lossless compression allows the designer to be sure that any data they may want to alter will be there, letting them create a final product before compressing the file further using a lossy compression. This is also true of sound files, where a sound mixer may need additional information, such as separate channels, that an end user will not require.

1.6 Types Of encoding

1.6.1 Huffman

Huffman coding is an entropy encoding algorithm used for lossless data compression. The term refers to the use of a variable-length code table for encoding a source symbol (such as a character in a file) where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol. It was developed by David A. Huffman while he was a Ph.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes".

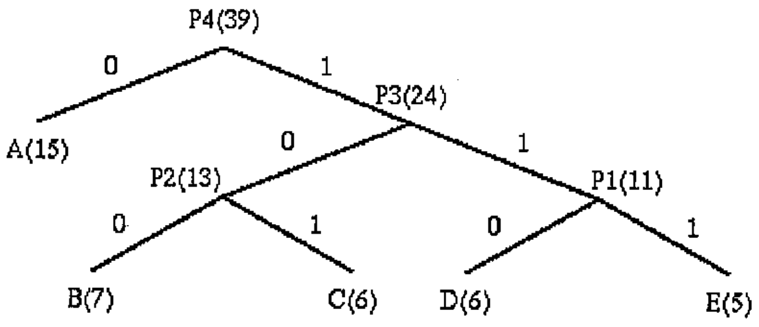
Huffman coding uses a specific method for choosing the representation for each symbol, resulting in a prefix code (sometimes called "prefix-free codes", that is, the bit string representing some particular symbol is never a prefix of the bit string representing any other symbol) that expresses the most common source symbols using shorter strings of bits than are used for less common source symbols. Huffman coding is such a widespread method for creating prefix codes that the term "Huffman code" is widely used as a synonym for "prefix code" even when such a code is not produced by Huffman's algorithm.

Encoding for Huffman Algorithm:

- A bottom-up approach

1. Initialization: Put all nodes in an OPEN list, keep it sorted at all times (e.g., ABCDE).
2. Repeat until the OPEN list has only one node left:

- From OPEN pick two nodes having the lowest frequencies/probabilities, create a parent node of them.
- Assign the sum of the children's frequencies/probabilities to the parent node and insert it into OPEN.
- Assign code 0, 1 to the two branches of the tree, and delete the children from OPEN.



Symbol	Count	$\log_2(1/p_i)$	Code	Subtotal (# of bits)
A	15	1.38	0	15
B	7	2.48	100	21
C	6	2.70	101	18
D	6	2.70	110	18
E	5	2.96	111	15

TOTAL (No. of bits): 87

Figure 1.4- Huffman encoding

Decoding:

• Decoding for the above two algorithms is trivial as long as the coding table (the statistics) is sent before the data. (There is a bit overhead for sending this, negligible if the data file is big.)
• **Unique Prefix Property**: no code is a prefix to any other code (all symbols are at the leaf nodes) --> great for decoder, unambiguous.
• If prior statistics are available and accurate, then Huffman coding is very good.

In the above example:

Entropy = (15 x 1.38 + 7 x 2.48 + 6 x 2.7 + 6 x 2.7 + 5 x 2.96) / 39)
= 85.26 / 39 = 2.19

Number of bits needed for Human Coding is: 87 / 39 = 2.23

Although Huffman's original algorithm is optimal for a symbol-by-symbol coding (i.e. a stream of unrelated symbols) with a known input probability distribution, it is not optimal when the symbol-by-symbol restriction is dropped, or when the probability mass functions are unknown, not identically distributed, or not independent (e.g., "cat" is more common than "cta"). Other methods such as arithmetic coding and LZW coding often have better compression capability.

1.6.2 Arithmetic

Arithmetic coding is a form of variable-length entropy encoding used in lossless data compression. Normally, a string of characters such as the words "hello there" is represented using a fixed number of bits per character, as in the ASCII code. When a string is converted to arithmetic encoding, frequently used characters will be stored with fewer bits and not-so-frequently occurring characters will be stored with more bits, resulting in fewer bits used in total. Arithmetic coding differs from other forms of entropy encoding such as Huffman coding in that rather than separating the input into component symbols and replacing each with a code, arithmetic coding encodes the entire message into a single number, a fraction *n* where ($0.0 \leq n < 1.0$).

Encoding and decoding

In general, each step of the encoding process, except for the very last, is the same; the encoder has basically just three pieces of data to consider:

- The next symbol that needs to be encoded
- The current interval (at the very start of the encoding process, the interval is set to [0,1), but that will change)
- The probabilities the model assigns to each of the various symbols that are possible at this stage (as mentioned earlier, higher-order or adaptive models mean that these probabilities are not necessarily the same in each step.)

The encoder divides the current interval into sub-intervals, each representing a fraction of the current interval proportional to the probability of that symbol in the current context. Whichever interval corresponds to the actual symbol that is next to be encoded becomes the interval used in the next step.

1.6.3 Limpel -Ziv -Welch

Suppose we want to encode the Webster's English dictionary which contains about 159,000 entries. Why not just transmit each word as an 18 bit number?

Problems: (a) Too many bits, (b) everyone needs a dictionary, (c) only works for English text.

- Solution: Find a way to build the dictionary adaptively.
- Original methods due to Ziv and Lempel in 1977 and 1978. Terry Welch improved the scheme in 1984 (called LZW compression). It is used in e.g., UNIX *compress*, GIF, V.42 bis.

LZW Compression Algorithm:

```
  w = NIL;
while ( read a character k )
   {
    if wk exists in the dictionary
     w = wk;
    else
      add wk to the dictionary;
      output the code for w;
      w = k;
   }
```

• Original LZW used dictionary with 4K entries, first 256 (0-255) are ASCII codes.
Example: Input string is "^WED^WE^WEE^WEB^WET".

w	*k*	*Output*	*Index*	*Symbol*
NIL	^			
^	W	^	256	^W
W	E	W	257	WE
E	D	E	258	ED
D	^	D	259	D^
^	W			
^W	E	256	260	^WE
E	^	E	261	E^
^	W			
^W	E			
^WE	E	260	262	^WEE
E	^			
E^	W	261	263	E^W
W	E			
WE	B	257	264	WEB
B	^	B	265	B^
^	W			
^W	E			
^WE	T	260	266	^WET
T	EOF	T		

Table 1.2- LZW Encoding table.

• A 19-symbol input has been reduced to 7-symbol plus 5-code output. Each code/symbol will need more than 8 bits, say 9 bits. Usually, compression doesn't start until a large number of bytes (e.g., > 100) are read in.

Decoding

The decoding algorithm works by reading a value from the encoded input and outputting the corresponding string from the initialized dictionary. At the same time it obtains the next value from the input, and adds to the dictionary the concatenation of the string just output and the first character of the string obtained by decoding the next input value. The decoder then proceeds to the next input value (which was already read in as the "next value" in the previous pass) and repeats the process until there is no more input, at which point the final input value is decoded without any more additions to the dictionary.

```
read a character k;
output k;
w = k;
while ( read a character k )   /* k could be a character or a code. */
  {
   entry = dictionary entry for k;
   output entry;
   add w + entry[0] to dictionary;
   w = entry;
  }
```

Example (continued): Input string is "^WED<256>E<260><261><257>B<260>T".

w	*k*	*Output*	*Index*	*Symbol*
	^	^		
^	W	W	256	^W
W	E	E	257	WE
E	D	D	258	ED
D	<256>	^W	259	D^
<256>	E	E	260	^WE
E	<260>	^WE	261	E^
<260>	<261>	E^	262	^WEE
<261>	<257>	WE	263	E^W
<257>	B	B	264	WEB
B	<260>	^WE	265	B^
<260>	T	T	266	^WET

Table 1.3- LZW decoding table

• Problem: What if we run out of dictionary space?

- Solution 1: Keep track of unused entries and use LRU (Least Recently Used)
- Solution 2: Monitor compression performance and flush dictionary when performance is poor.

• Implementation Note: LZW can be made *really* fast; it grabs a fixed number of bits from input stream, so bit parsing is very easy. Table lookup is automatic.

Summary

- Huffman maps fixed length symbols to variable length codes. Optimal only when symbol probabilities are powers of 2.
- Lempel-Ziv-Welch is a dictionary-based compression method. It maps a variable number of symbols to a fixed length code.
- Adaptive algorithms do not need a priori estimation of probabilities, they are more useful in real applications.

1.6.4 JPEG

One of the most popular and comprehensive continuous tone, still frame compression standards is the JPEG () standard. In the JPEG baseline coding system, which is based on the discrete cosine transform and is the adequate for most compression application, the input and the output images are limited to 8 bits while the quantized DCT coefficient values are restricted to 11 bits. The compression itself is performed in four sequential steps ; 8x8 sub image extraction , DCT computation, quantization, and variable length code assignment.

The first step in the JPEG compression is to sub divides the input image into non overlapping pixel block of size 8x8. They are then subsequently processed left to right top to bottom. As each 8x8 block or sub image is processed, its 64 pixels are level shift by subtracting 2^(m-1), where 2^m is the number of gray level in the image and its 2d discrete cosine transform is computed .

Discrete cosine transform

The 8×8 sub-image shown in 8-bit grayscaleEach 8×8 block of each component (Y, Cb, Cr) is converted to a frequency-domain representation, using a normalized, two-dimensional type-II discrete cosine transform (DCT).

As an example, one such 8×8 8-bit subimage might be:

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}.$$

Quantization matrix used in JPEG

The resulting coefficient are then simultaneously normalized and quantized in accordance with

$$\hat{T}(u,v) = \text{round}\left[\frac{T(u,v)}{Z(u,v)}\right]$$

Where T(u,v) for u,v=0,1……,7 are the resulting normalized and quantized coefficients, T(u,v) is the DCT of 8x8 block of image f(x,y), and Z(u,v) is a transform normalization. By scaling Z(u,v), a varity of compression rates and reconstructed image qualities can be achieved.

1.6.5 JPEG 2000

JPEG 2000 is a new image coding system that uses state-of-the-art compression techniques based on wavelet technology. Its architecture should lend itself to a wide range of uses from portable digital cameras through to advanced pre-press, medical imaging and other key sectors.

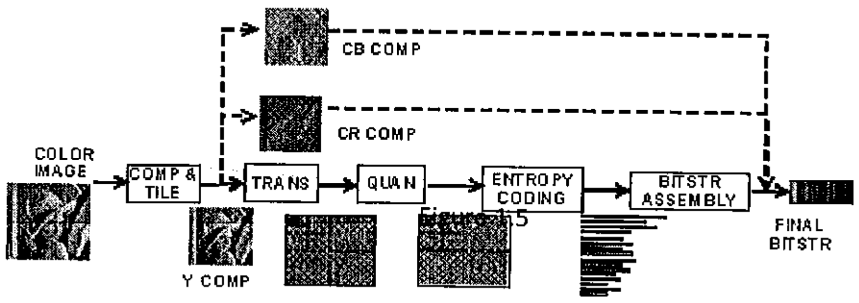


Figure 2.5 – Flow chart of JPEG 2000

Wavelet transform

JPEG 2000 uses two different wavelet transforms:

- *irreversible*: the CDF 9/7 wavelet transform. It is said to be "irreversible" because it introduces quantization noise that depends on the precision of the decoder.
- *reversible*: a rounded version of the bi orthogonal CDF 5/3 wavelet transform. It uses only integer coefficients, so the output does not require rounding (quantization) and so it does not introduce any quantization noise. It is used in lossless coding.

The wavelet transforms are implemented by the lifting scheme or by convolution.

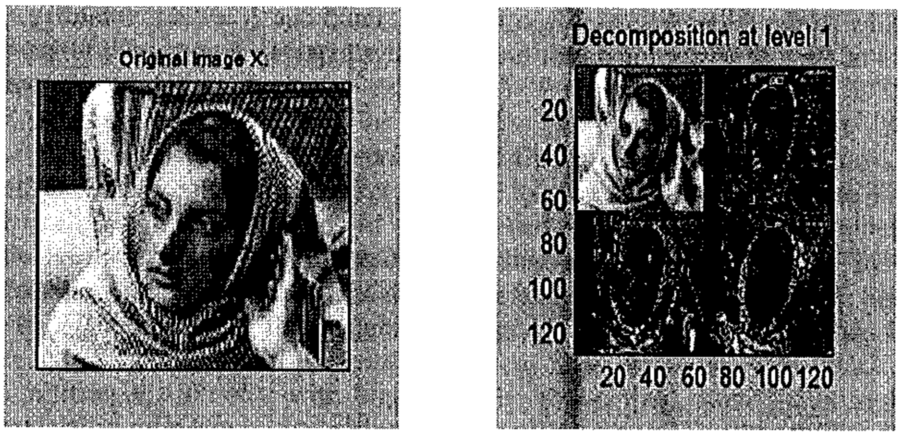


Figure 2.6- Wavelet transform

Quantization

After the wavelet transform, the coefficients are scalar-quantized to reduce the amount of bits to represent them, at the expense of a loss of quality. The output is a set of integer numbers which have to be encoded bit-by-bit. The parameter that can be changed to set the final quality is the quantization step: the greater the step, the greater is the compression and the loss of quality.

Coding

The result of the previous process is a collection of *sub-bands* which represent several approximation scales. A sub-band is a set of *coefficients*—real numbers which represent aspects of the image associated with a certain frequency range as well as a spatial area of the image.

Precincts are split further into *code blocks*. Code blocks are located in a single sub-band and have equal sizes—except those located at the edges of the image. The encoder has to encode the bits of all quantized coefficients of a code block, starting with the most significant bits and progressing to less significant bits by a process called the *EBCOT* scheme. *EBCOT* here stands for *Embedded Block Coding with Optimal Truncation*. In this encoding process, each bit plane of the code block gets encoded in three so-called *coding passes*, first encoding bits (and signs) of insignificant coefficients with significant neighbors (i.e., with 1-bits in higher bit planes), then refinement bits of significant coefficients and finally coefficients without significant neighbors. The three passes are called *Significance Propagation*, *Magnitude Refinement* and *Cleanup* pass, respectively.

The result is a bit-stream that is split into *packets* where a *packet* groups selected passes of all code blocks from a precinct into one indivisible unit. Packets are the key to quality scalability (i.e., packets containing less significant bits can be discarded to achieve lower bit rates and higher distortion).Packets from all sub-bands are then collected in so-called *layers*.

Chapter 2
Literature review

Wavelets are a mathematical tool for hierarchically decomposing functions. They allow a function to be described in terms of a coarse overall shape, plus details that range from broad to narrow. Regardless of whether the function of interest is an image, a curve, or a surface, wavelets offer an elegant technique for representing the levels of detail present. This primer is intended to provide people working in computer graphics with some intuition for what wavelets are, as well as to present the mathematical foundations necessary for studying and using them. In Part 1, we discuss the simple case of Haar wavelets in one and two dimensions, and show how they can be used for image compression.

The mathematical theory of multiresolution analysis, then develop spline wavelets and describe their use in multiresolution curve and surface editing. Although wavelets have their roots in approximation theory and signal processing, they have recently been applied to many problems in computer graphics. These graphics applications include image editing, image compression, and image querying, automatic level-of-detail control for editing and rendering curves and surfaces surface reconstruction from contours; and fast methods for solving simulation problems in animation and global illumination [1]

.

It is presented in this paper, a simple construction of second generation wavelets, wavelets that are not necessarily translates and dilates of one fixed function. Such wavelets can be adapted to intervals, domains, surfaces, weights, and irregular samples. We show how the lifting scheme leads to a faster, in-place calculation of the wavelet transform. Several examples are included. [2]

This proposes a new method for image compression. The method is based on the approximation of an image, regarded as a function, by a linear spline over an adapted triangulation, D(Y),which is the Delaunay triangulation of a small set Y of significant pixels. The linear spline minimizes the distance to the image, measured by the mean square error, among all linear splines over D(Y). The significant pixels in Y are selected by an adaptive thinning algorithm, which recursively removes less significant pixels in a greedy way, using a sophisticated criterion for measuring the significance of a pixel. The proposed compression method combines the approximation scheme with a customized scattered data coding scheme.
We compare our compression method with JPEG2000 on two geometric images and on three popular test cases of real images.[2]

In this paper the mathematical methods applied in the most recent image formats are presented. First of all, the application of the wavelet transform in JPEG2000 is gone through. JPEG2000 is a standard established by the same group which created the widely used JPEG standard, and it was established to solve some of the shortcomings of JPEG. Also presented are other recently established image formats having wavelet transforms as part of the codec.

Other components in modern image compression systems are also gone through, together with the mathematical and statistical methods used. [4]

This paper Proposes a realistic representation of a terrain Light Detection and Ranging data (LiDAR) requires trillion numbers of points. These points connected in triangles that represent the surface of the terrain ultimately increase the data size. For online GIS interactive programs it has become highly essential to reduce the number of triangles in order to save more storing space. In this paper, it is extended to the LiDAR data compression. A newly developed data compression approach to approximate the LiDAR surface with a series of non-overlapping triangles has been presented. Generally a Triangulated Irregular Networks (TIN) are the most common form of digital surface model that consists of elevation values with x, y coordinates that make up triangles. Compression of TIN is needed for efficient management of large data and good surface visualization. This approach covers following steps: First, by using a Delaunay triangulation, an efficient algorithm is developed to generate TIN, which forms the terrain from an arbitrary set of data.

A new interpolation wavelet filter for TIN has been applied in two steps, namely splitting and elevation. In the splitting step, a triangle has been divided into several sub-triangles and the elevation step has been used to 'modify' the point values (point coordinates for geometry) after the splitting. Then, this data set is compressed at the desired locations by using second generation wavelets. The quality of geographical surface representation after using proposed technique is compared with the original LIDAR data. The results show that this method can be used for significant reduction of data set. [13]

Chapter 3

Wavelets (New approach)

Wavelets are a mathematical tool for hierarchically decomposing functions. They allow a function to be described in terms of a coarse overall shape, plus details that range from broad to narrow. Regardless of whether the function of interest is an image, a curve, or a surface, wavelets offer an elegant technique for representing the levels of detail present. This primer is intended to provide people working in computer graphics with some intuition for what wavelets are, as well as to present the mathematical foundations necessary for studying and using them.

Although wavelets have their roots in approximation theory and signal processing, they have recently been applied to many problems in computer graphics. These graphics applications include image editing, image compression , and image querying; automatic level-of-detail control for editing and rendering curves and surfaces surface reconstruction from contours and fast methods for solving simulation problems in animation and global illumination. For a discussion of wavelets that goes beyond the scope of this primer, we refer readers to our forthcoming monograph.

3.1 Wavelets in one dimension

The Haar basis is the simplest wavelet basis. We will first discuss how a one-dimensional function can be decomposed using Haar wavelets, and then describe the actual basis functions. Finally, we show how sing the Haar wavelet decomposition leads to a straightforward technique for compressing a one-dimensional function.

3.1.1 One-dimensional Haar wavelet transform

To get a sense for how wavelets work, let's start with a simple example. Suppose we are given a one-dimensional "image" with a resolution of four pixels, having values .We can represent this image in the *Haar basis* by computing a wavelet transform. To do this, we first average the pixels together, pairwise, to get the new lower resolution image with pixel values .Clearly, some information has been lost in this averaging process. To recover the original four pixel values from the two averaged values,we need to store some *detail coefficients*, which capture the missing information Finally, we will define the *wavelet transform* (also called the *wavelet decomposition*) of the original four-pixel image to be the single coefficient representing the overall average of the original image, followed by the detail coefficients in order of increasing resolution. Thus, for the one-dimensional Haar basis, the wavelet transform of our original four-pixel image. The way we computed the wavelet transform, by recursively averaging and differencing coefficients, is called *afilter bank*—a process we will generalize to other types of wavelets in Part 2. Note that no information has been gained or lost by this process.

One level of the transform

The DWT of a signal x is calculated by passing it through a series of filters. First the samples are passed through a low pass filter with impulse response g resulting in a convolution of the two:

$$y[n] = (x * g)[n] = \sum_{k=-\infty}^{\infty} x[k]g[n-k].$$

The signal is also decomposed simultaneously using a high-pass filter h. The outputs giving the detail coefficients (from the high-pass filter) and approximation coefficients (from the low-pass). It is important that the two filters are related to each other and they are known as a quadrature mirror filter.

However, since half the frequencies of the signal have now been removed, half the samples can be discarded according to Nyquist's rule. The filter outputs are then sub sampled by 2 (Mallat's and the common notation is the opposite, g- high pass and h- low pass):

$$y_{\mathrm{low}}[n] = \sum_{k=-\infty}^{\infty} x[k]g[2n-k]$$

$$y_{\mathrm{high}}[n] = \sum_{k=-\infty}^{\infty} x[k]h[2n-k]$$

This decomposition has halved the time resolution since only half of each filter output characterises the signal. However, each output has half the frequency band of the input so the frequency resolution has been doubled.

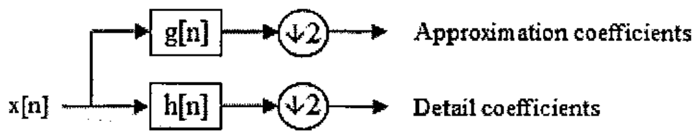


Figure 3.1 Block diagram of filter analysis

With the sub sampling operator $\downarrow$

$$(y \downarrow k)[n] = y[kn]$$

The above summation can be written more concisely.

$$y_{\text{low}} = (x * g) \downarrow 2$$

$$y_{\text{high}} = (x * h) \downarrow 2$$

However computing a complete convolution $x * g$ with subsequent down sampling would waste computation time.

The Lifting scheme is an optimization where these two computations are interleaved.

Cascading and Filter banks

This decomposition is repeated to further increase the frequency resolution and the approximation coefficients decomposed with high and low pass filters and then down-sampled. This is represented as a binary tree with nodes representing a sub-space with a different time-frequency localization. The tree is known as a filter bank.

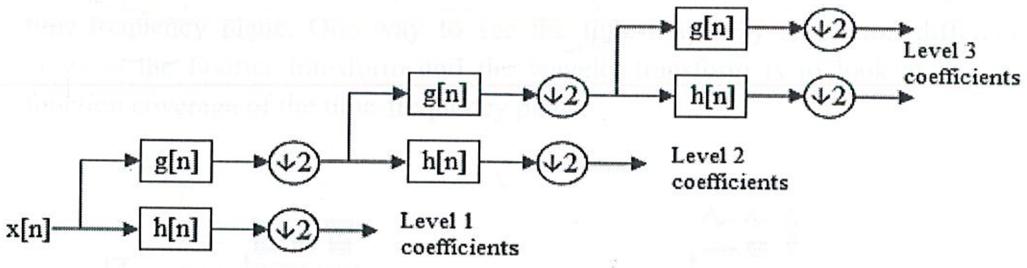


Figure 3.2 A 3 level filter bank

3.2 Image Approximation

This section provides a detailed description of our image approximation scheme. We introduce the adaptive thinning algorithm for the selection of a set of significant pixels Y . This includes a discussion of its significance measure and of linear splines over Delaunay triangulations. Moreover, we describe the final step of the image approximation scheme, where we construct the best approximation to the image, minimizing the mean square error among all linear splines over the Delaunay triangulation of Y . Finally, we show how to control the mean square error of the image approximation.

3.3 Wavelets Transform

There are many advantages of Wavelet Transforms over the Fourier Transforms because of which Wavelets are currently most widely used. Some of the major differences are discussed below:

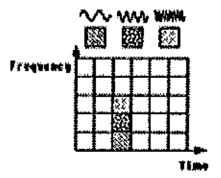
a. The most interesting dissimilarity between these two kinds of transforms is that individual wavelet functions are ***localized in space***. Fourier sine and cosine functions are not.

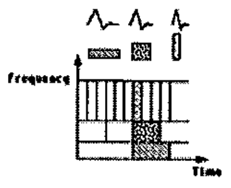
b. This localization feature, along with wavelets' localization of frequency, makes many functions and operators using wavelets "sparse" when transformed into the wavelet domain. This sparseness, in turn, results in a number of useful applications such as ***data compression, detecting features in images, and removing noise*** from time series.

c. The output signal in a Wavelet Transform is a signal in both ***time and frequency domain*** whereas a Fourier Transform converts the time-domain to frequency domain.

d. One way to see the time-frequency resolution differences between the Fourier transform and the wavelet transform is to look at the ***basis function*** coverage of the time-frequency plane. One way to see the time-frequency resolution differences between the Fourier transform and the wavelet transform is to look at the basis function coverage of the time-frequency plane.

FT





WT

1. Window is simply a square wave.
2. The square wave window truncates the sine or cosine function to fit a window of a particular width

1. Windows *vary*.
2. In order to isolate signal discontinuities, one would like to have some very short basis functions. At the same time, in order to obtain detailed frequency analysis, one would like to have some very long basis functions

Figure 3.3 Wavelet Transform

Wavelet transforms do not have a single set of basis functions like the Fourier transform, which utilizes just the sine and cosine functions. Instead, wavelet transforms have an infinite set of possible basis functions. Thus wavelet analysis provides immediate access to information that can be obscured by other time-frequency methods such as Fourier analysis.

3.4 SECOND GENERATION WAVELETS

The Second Generation Wavelets are easier to understand and implement than the First Generation Wavelets. The first generation wavelets could be easily used for periodic and infinite domain signals. But there was no clear cut way of using it for bounded domain signals. Furthermore, even 1-D signals are often not sampled regularly. In higher dimension, domains are often have boundaries, and often the metric is not flat, i.e., we need to analyze functions on manifolds or surfaces. Thus the Second Generation Wavelets came into picture. The Second Generation Wavelets have all the useful properties like time-frequency localization and fast implementation of the First Generation Wavelets in addition to being able to represent the signals which are bounded. This has been achieved by removing the translation and dilation of the mother wavelet. Instead we use a Lifting Scheme. Hence we do not use any Fourier Analysis.

There are two advantages of second generation wavelets. The first advantage is that fast computation and multi resolution capabilities of the first generation wavelets, are retained in the second generation wavelets. The second advantage is that the forward and inverse wavelet transform are invertible to each other as can be seen from the figure below:

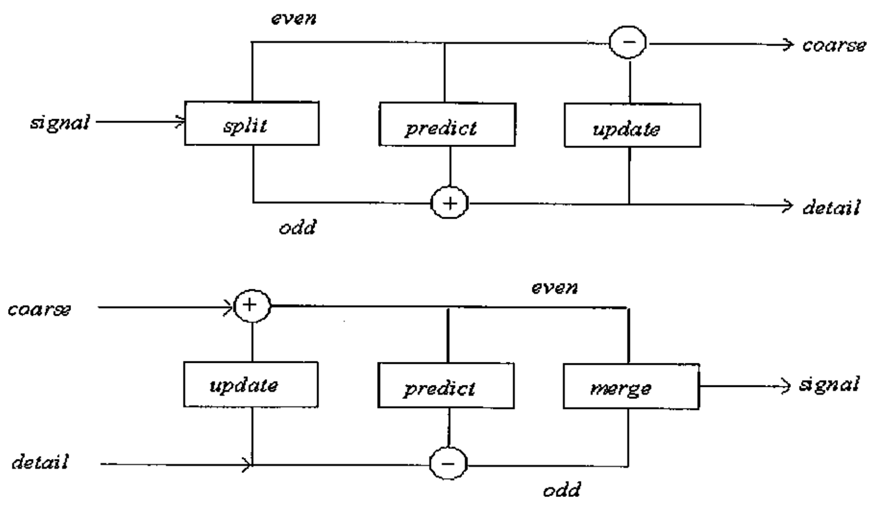


Figure 3.4 Second Generation Wavelets

3.5 Lifting Scheme

The **lifting scheme** is a technique for both designing wavelets and performing the discrete wavelet transform. Actually it is worthwhile to merge these steps and design the wavelet filters *while* performing the wavelet transform. This is then called the second generation wavelet transform.

The discrete wavelet transform applies several filters separately to the same signal. In contrast to that, for the lifting scheme the signal is divided like a zipper. Then a series of convolution-accumulate operations across the divided signals is applied.

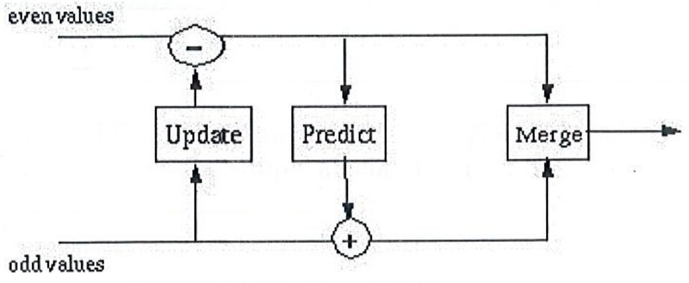


Figure 3.5 Lifting Scheme Inverse Transform

3.5.1 Basic Lifting Scheme Wavelets

Wavelet algorithms are recursive. The output of one step of the algorithm becomes the input for the next step. The initial input data set consists of 2^n elements. Each successive step operates on 2^{n-i} elements, were $i = 1 \ldots n-1$.

On this web page $step_{j+1}$ follows $step_j$. If element i in step j is being updated, the notation is $step_{j,i}$. The forward lifting scheme wavelet transform divides the data set being processed into an even half and an odd half. In the notation below $even_i$ is the index of the i^{th} element in the even half and odd_i is the i^{th} element in the odd half (I'm pretending that the even and odd halves are both indexed from 0). Viewed as a continuous array (which is what is done in the software) the even element would be a[i] and the odd element would be a[i+(n/2)].

Another way to refer to the recursive steps is by their power of two. This notation is used in Ripples in Mathematics. Here $step_{j-1}$ follows $step_j$, since each wavelet step operates on a decreasing power of two. This is a nice notation, since the references to the recursive step in a summation also correspond to the power of two being calculated.

3.5.2 Predict Wavelets

The prediction step predicts that the odd element will be equal to the even element. The difference between the predicted value (the even element) and the actual value of the odd element replaces the odd element. For the forward transform iteration *j* and element *i*, the new odd element, *j+1,i* would be

$$odd_{j+1,i} = odd_{j,i} - even_{j,i}$$

In the lifting scheme version of the Haar transform the update step replaces an even element with the average of the even/odd pair (e.g., the even element s_i and its odd successor, s_{i+1}):

$$even_{j+1,i} = \frac{even_{j,i} + odd_{j,i}}{2}$$

The original value of the $odd_{j,i}$ element has been replaced by the difference between this element and its even predecessor. Simple algebra lets us recover the original value:

$$odd_{j,i} = even_{j,i} + odd_{j+1,i}$$

Substituting this into the average, we get

$$even_{j+1,i} = \frac{even_{j,i} + even_{j,i} + odd_{j+1,i}}{2}$$

$$even_{j+1,i} = even_{j,i} + \frac{odd_{j+1,i}}{2}$$

3.5.3 The Update Step

The update step replaces the even elements with an average. These results in a smoother input for the next step of the next step of the wavelet transform. The odd elements also represent an approximation of the original data set, which allows filters to be constructed. A simple lifting cheme forward transform is diagrammed.

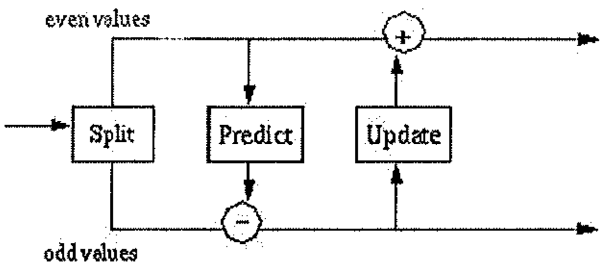


Figure 3.6 Lifting Scheme Forward Transform

The update phase follows the predict phase. The original value of the odd elements has been overwritten by the difference between the odd element and its even "predictor". So in calculating an average the update phase must operate on the differences that are stored in the odd elements:

$$\text{even}_{j+1,i} = \text{even}_{j,i} + U(\ \text{odd}_{j+1,i}\ )$$

Chapter 4
Implementation and coding

4.1 The concept of the algorithm

In our algorithm we have implemented compression by using wavelets decomposition

4.1.1 Module 1

The first step of our algorithm takes in the input in three forms

1. The original black and white image.
   Or
2. The decomposed image's approximation part
   This decomposition is performed by lifting wavelet transform by the function *lwt2*. We take the approximation part as the input.
   Or
3. The decomposed image's detail parts
   This decomposition is performed by lifting wavelet transform by the function *lwt2*.We take detail parts as the inputs.

After passing any of these as an input to our function **ddencmp_mod(),** the function thus generated a threshold which is passed on to the next step. The threshold is calculated as such

1. The image provided as the input is decomposed using the function wavedec2. – a 2nd level decomposition function
2. The a normalize value is calculated
   a. It is first initialized as max value of the absolute of the decomposed image.
3. Then the normalized value is then multiplied with a constant of our choice.
4. The threshold thus found is then directed into a matrix of 3x1 with addition of variables.
5. This matrix is then fed as an input to the next module.

4.1.2 <u>Module 2</u>

The first step of our second module are the inputs to our second function **Wdencmp_mod_mod()**

1. the threshold provided by the first module,
2. the wavelet name we wish to decompose it by,
3. the image again
4. the level by which we want to decompose is
5. Whether the *hard* of the *soft* threshold has to be taken

After passing all these as in inputs to the second module the calculation happens as such

1. The image is again decomposed using the same *wavedec2* function
2. A coefficient is calculated for all the three different details of the image provided
3. The coefficient thus provided and the threshold passed into the module provides the tools by which the image is reconstructed by using the function *waverec2.*
4. The last phase of the module two is the calculation of both normal recovery of the image and the compression ratio.
5. These are then passed as outputs of the algorithm

These two parameters lets us compare the original image to the reconstructed one, so that we could decipher the improvement thus occurred.

Note : If we have used the lifting wavelet decomposition function to provide inputs in the module 1 then we have to reconstruct the image back by using *ilwt2* function .

4.2 The Terminology of The Algorithm Explained

4.2.1 Module 1

Ddencmp_mod()

- [THR,SORH,KEEPAPP] = ddencmp_mod(IN1,IN2,X) returns default values for compression, using wavelets or wavelet packets, of an input vector or matrix X, which can be a one- or two-dimensional signal. THR is the threshold, SORH is for soft or hard thresholding, KEEPAPP allows you to keep approximation coefficients.

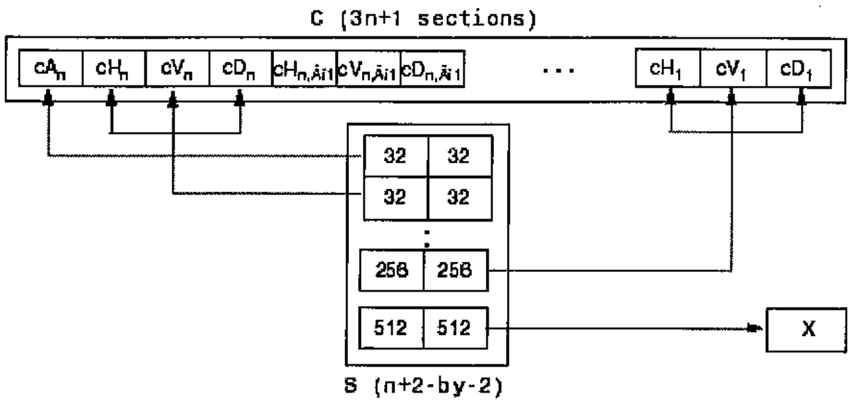
wavedec2()

- wavedec2 is a two-dimensional wavelet analysis function.
- [C,S] = wavedec2(X,N,*'wname'*) returns the wavelet decomposition of the matrix X at level N, using the wavelet named in string *'wname'* (see wfilters for more information).
- Outputs are the decomposition vector C and the corresponding bookkeeping matrix S.
- N must be a strictly positive integer (see wmaxlev for more information). Instead of giving the wavelet name, you can give the filters. For [C,S] = wavedec2(X,N,Lo_D,Hi_D), Lo_D is the decomposition low-pass filter and Hi_D is the decomposition high-pass filter.
- Vector C is organized as
  C = [A(N) | H(N) | V(N) | D(N) | ...
  H(N-1) | V(N-1) | D(N-1) | ... | H(1) | V(1) | D(1)].
  where A, H, V, D, are row vectors such that
  A = approximation coefficients
  H = horizontal detail coefficients
  V = vertical detail coefficients
  D = diagonal detail coefficients

  Each vector is the vector column-wise storage of a matrix.

Matrix S is such that

- S(1,:) = size of approximation coefficients(N).
- S(i,:) = size of detail coefficients(N-i+2) for i = 2, ...N+1 and S(N+2,:) = size(X).



lwt2()

- lwt2 performs a 2-D lifting wavelet decomposition with respect to a particular lifted wavelet that you specify.

- [CA,CH,CV,CD] = lwt2(*X,W*) computes the approximation coefficients matrix CA and detail coefficients matrices CH, CV, and CD, obtained by a lifting wavelet decomposition, of the matrix *X*. *W* is a lifted wavelet name

- X_InPlace = lwt2(X,LS) computes the approximation and detail coefficients. These coefficients are stored in place:

  - CA = X_InPlace(1:2:end,1:2:end)
  - CH = X_InPlace(2:2:end,1:2:end)
  - CV = X_InPlace(1:2:end,2:2:end)
  - CD = X_InPlace(2:2:end,2:2:end)

- lwt2(*X,W*,LEVEL) computes the lifting wavelet decomposition at level LEVEL.

- X_InPlace = lwt2(*X,W*,LEVEL,'typeDEC',typeDEC) or [CA,CH,CV,CD] = LWT2(*X,W*,LEVEL,'typeDEC',typeDEC) with typeDEC = 'w' or 'wp' computes the wavelet or the wavelet packet decomposition using lifting, at level LEVEL.

4.2.2 Module 2

Wdencmp_mod_mod()

- [XC,CXC,LXC,PERF0,PERFL2] = wdencmp_mod('gbl',X,'wname',N,THR,SORH, KEEPAPP) returns a de-noised or compressed version XC of input signal X (one- or two-dimensional) obtained by wavelet coefficients thresholding using global positive threshold THR.

- Additional output arguments [CXC,LXC] are the wavelet decomposition structure of XC (see wavedec or wavedec2 for more information). PERF0 and PERFL2 are L2-norm recovery and compression score in percentage.

- PERFL2 = 100 * (vector-norm of CXC / vector-norm of C)2 if [C,L] denotes the wavelet decomposition structure of X.

- Wavelet decomposition is performed at level N and 'wname' is a string containing wavelet name .SORH ('s' or 'h') is for soft or hard thresholding .If KEEPAPP = 1, approximation coefficients cannot be thresholded, otherwise it is possible.

waverec2()

• X = waverec2(C,S,*'wname'*) performs a multilevel wavelet reconstruction of the matrix X based on the wavelet decomposition structure [C,S]. For detailed storage information, see wavedec2. *'wname'* is a string containing the name of the wavelet..

• Instead of specifying the wavelet name, you can specify the filters.

  - X = waverec2(C,S,Lo_R,Hi_R), Lo_R is the reconstruction low-pass filter
  - Hi_R is the reconstruction high-pass filter.

• waverec2 is the inverse function of wavedec2 in the sense that the abstract statement waverec2(wavedec2(X,N,*'wname'*),*'wname'*) returns X.

• X = waverec2(C,S,*'wname'*) is equivalent to X = appcoef2(C,S,*'wname'*,0).

ilwt2()

- ilwt2 performs a 2-D lifting wavelet reconstruction with respect to a particular lifted wavelet that you specify.
- X = ilwt2(AD_In_Place,W) computes the reconstructed matrix X using the approximation and detail coefficients matrix AD_In_Place, obtained by a lifting wavelet decomposition. W is a lifted wavelet name (see liftwave).
- X = ilwt2(CA,CH,CV,CD,W) computes the reconstructed matrix X using the approximation coefficients vector CA and detail coefficients vectors CH, CV, and CD obtained by a lifting wavelet decomposition.
- X = ilwt2(AD_In_Place,W,LEVEL) or X = ILWT2(CA,CH,CV,CD,W,LEVEL) computes the lifting wavelet reconstruction, at level LEVEL.
- X = ilwt2(AD_In_Place,W,LEVEL,'typeDEC',typeDEC) or X = ilwt2(CA,CH,CV,CD,W,LEVEL,'typeDEC',typeDEC) with typeDEC = 'w' or 'wp' computes the wavelet or the wavelet packet decomposition using lifting, at level LEVEL.

4.3 Normal application

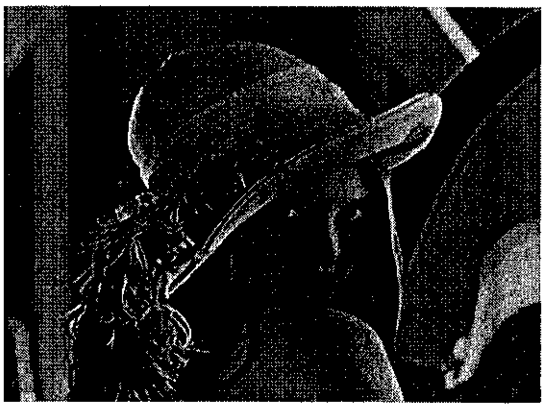
4.3.1 Main program:-

```
close all;
clear all;
summ=0;
[X,map] = imread('lena512color.tiff');
I= rgb2gray(X);
I1=double(I);
figure (1);
imshow(I);
[thr,sorh,keepapp]=ddencmp('cmp','wv',I1);
thr_h = [17 18];     % Horizontal thresholds.
thr_d = [19 20];     % Diagonal thresholds.
thr_v = [21 22];     % Vertical thresholds.

thr = [thr_h ; thr_d ; thr_v]

[xd,cxd,lxd,perf0,perfl2] = wdencmp('lvd',I1,'db3',2,thr,sorh);
figure (3);
xd=uint8(xd);
imshow(xd);
perf0
perfl2
```

4.3.2 Results and comparisons:-

Original

Result



Normal recovery= 90.6412
Compression score = 99.8999

4.4 Modified application

4.4.1 Main program:-

```
close all;
clear all;
summ=0;
[X,map] = imread('lena512color.tiff');
I= rgb2gray(X);
I1=double(I);
figure (1);
imshow(I);
[thr,sorh,keepapp]=ddencmp_mod('cmp','wv',I1);

[xd,cxd,lxd,perf0,perfl2] = wdencmp_mod('lvd',I1,'db3',2,thr,sorh);

figure (3);
xd=uint8(xd);
imshow(xd);
perf0
perfl2
```

4.4.2 Functions used:-

ddencmp_mod ()*

```
function [thr,sorh,keepapp,crit] = ddencmp_mod(dorc,worwp,x)
% Check arguments.
nbIn = nargin;
if nbIn<3
    error('Not enough input arguments.');
elseif isequal(worwp,'wv')
    if (nargout>3)
        error('Too many output arguments.');
    end
elseif isequal(worwp,'wp')
    if (nargout>4)
        error('Too many output arguments.');
    end
else
    error('Invalid argument value');
end


% Set problem dimension.
if min(size(x))~=1, dim = 2; else , dim = 1; end

% Set keepapp default value.
keepapp = 1;
```

```
% Set sorh default value.
if isequal(dorc,'den') & isequal(worwp,'wv') , sorh = 's'; else ,sorh =
'h'; end

% Set threshold default value.
n = prod(size(x));

% nominal threshold.
switch dorc
  case 'cmp' ,  thr = 1;
end

% rescaled threshold.
if dim == 1
    [c,l] = wavedec(x,1,'db1');
    c = c(l(1)+1:end);
else
    [c,l] = wavedec2(x,1,'db1');
    c = c(prod(l(1,:))+1:end);
end

normaliz = var(abs(c))+6;

% if normaliz=0 in compression, kill the lowest coefs.
if dorc == 'cmp' & normaliz == 0
    normaliz = 0.05*max(abs(c));
end

if dorc == 'cmp'
     thr = thr*normaliz;
    if worwp == 'wp', crit = 'threshold'; end
end

thr_h = [thr*normaliz   thr*normaliz+2];     % Horizontal thresholds.
thr_d = [thr*normaliz+1 thr*normaliz+3];     % Diagonal thresholds.
thr_v = [thr*normaliz+3 thr*normaliz+1];     % Vertical thresholds.

thr = [thr_h ; thr_d ; thr_v] ;
sorh
keepapp
% crit
```

wdencmp_mod()*

```
function [xc,cxc,lxc,perf0,perfl2] = wdencmp(o,varargin)
% Check arguments and set problem dimension.
klo=10;
dim = 1;         % initialize dimension to 1D.
nbIn  = nargin;
nbOut = nargout;
switch o
    case 'gbl' , minIn = 7; maxIn = 8;
    case 'lvd' , minIn = 6; maxIn = 7;
    otherwise  , error('Invalid argument value.')
```

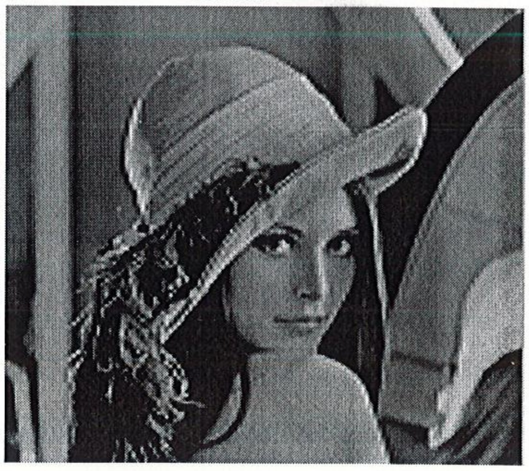
```
end
if nbIn < minIn
    error('Not enough input arguments.');
elseif nbIn > maxIn
    error('Too many input arguments.');
end
okOut = [0:1 3:5];
if ~any(okOut==nbOut)
    error('Invalid number of output arguments.');
end
 if nbIn == minIn
    x = varargin{1}; indarg = 2;
    if min(size(x))~=1, dim = 2;
            end
else
    c = varargin{1}; l = varargin{2}; indarg = 3;
    if min(size(l))~=1, dim = 2;
     end
end
 % Get Inputs
w    = varargin{indarg};
n    = varargin{indarg+1};
thr  = varargin{indarg+2};
sorh = varargin{indarg+3};

if (o=='gbl' & nbIn==7)  | (o=='lvd' & nbIn==6)

    if dim == 1, [c,l] = wavedec(x,n,w);
    else,        [c,l] = wavedec2(x,n,w);

    end
end

% Wavelet coefficients thresholding.
if o=='gbl'
    if keepapp
        % keep approximation.
             cxc = c;
                    if dim == 1, inddet = l(1)+1:length(c);
        else, inddet = prod(l(1,:))+1:length(c); end
        % threshold detail coefficients.
        cxc(inddet) = wthresh(c(inddet),sorh,thr);
    else
        % threshold all coefficients.
        cxc = wthresh(c,sorh,thr);
    end
else

    if dim == 1, cxc = wthcoef('t',c,l,1:n,thr,sorh);
    else

        cxc = wthcoef2('h',c,l,1:n,thr(1,:),sorh);
        cxc = wthcoef2('d',cxc,l,1:n,thr(2,:),sorh);
        cxc = wthcoef2('v',cxc,l,1:n,thr(3,:),sorh);
    end
end
```

```
lxc = l;

% Wavelet reconstruction of xd.
if dim == 1,xc = waverec(cxc,lxc,w);
else
    xc = waverec2(cxc,lxc,w);
end

if nbOut<4 , return; end

% Compute compression score.
perf0 = 100*(length(find(cxc==0))/length(cxc));
if nbOut<5 , return; end

% Compute L^2 recovery score.
nc = norm(c);

if nc<eps
    perfl2 = 100;
else
    perfl2 = 100*((norm(cxc)/nc)^2);
end
% size(xc)
% CR= size(c)/size(cxc)
```

4.4.3 Results and comparisons:-

Original

Result



Normal recovery=   93.6047
Compression score = 99.5704

4.5 Modified application 2

4.5.1 Main program:-

```
close all;
clear all;
summ=0;
[X,map] = imread('lena512color.tiff');
I= rgb2gray(X);
I1=double(I);
figure (1);
imshow(I);
[cA,cH,cV,cD]=lwt2(I1,'db3');

figure (2);
cA=uint8(cA);
imshow(cA)
[thr,sorh,keepapp]=ddencmp_mod('cmp','wv',cA);

[xd,cxd,lxd,perf0,perfl2] = wdencmp_mod('lvd',cA,'db3',2,thr,sorh);

figure (3);
xd=uint8(xd);
xd=double(xd);
imshow(xd);

perf0
perfl2

size(xd)
size(cA)
p=ilwt2(xd,cH,cV,cD,'db3');
figure (4);
p=uint8(p);
imshow(p);
```

4.5.2 Functions used:-

ddencmp_mod ()*

```
function [thr,sorh,keepapp,crit] = ddencmp_mod(dorc,worwp,x)
% Check arguments.
nbIn = nargin;
if nbIn<3
    error('Not enough input arguments.');
elseif isequal(worwp,'wv')
    if (nargout>3)
        error('Too many output arguments.');
    end
elseif isequal(worwp,'wp')
    if (nargout>4)
```

```
        error('Too many output arguments.');
    end
else
    error('Invalid argument value');
end


% Set problem dimension.
if min(size(x))~=1, dim = 2; else , dim = 1; end

% Set keepapp default value.
keepapp = 1;

% Set sorh default value.
if isequal(dorc,'den') & isequal(worwp,'wv') , sorh = 's'; else ,sorh =
'h'; end

% Set threshold default value.
n = prod(size(x));

% nominal threshold.
switch dorc
  case 'cmp' ,  thr = 1;
end

% rescaled threshold.
if dim == 1
    [c,l] = wavedec(x,1,'db1');
    c = c(l(1)+1:end);
else
    [c,l] = wavedec2(x,1,'db1');
    c = c(prod(l(1,:))+1:end);
end

normaliz = var(abs(c))+6;

% if normaliz=0 in compression, kill the lowest coefs.
if dorc == 'cmp' & normaliz == 0
    normaliz = 0.05*max(abs(c));
end

if dorc == 'cmp'
     thr = thr*normaliz;
    if worwp == 'wp', crit = 'threshold'; end
end

thr_h = [thr*normaliz   thr*normaliz+2];     % Horizontal thresholds.
thr_d = [thr*normaliz+1 thr*normaliz+3];     % Diagonal thresholds.
thr_v = [thr*normaliz+3 thr*normaliz+1];     % Vertical thresholds.

thr = [thr_h ; thr_d ; thr_v] ;
sorh
keepapp
% crit
```

wdencmp_mod()*

```
function [xc,cxc,lxc,perf0,perfl2] = wdencmp(o,varargin)
% Check arguments and set problem dimension.
klo=10;
dim = 1;         % initialize dimension to 1D.
nbIn  = nargin;
nbOut = nargout;
switch o
    case 'gbl' , minIn = 7; maxIn = 8;
    case 'lvd' , minIn = 6; maxIn = 7;
    otherwise  , error('Invalid argument value.')
end
if nbIn < minIn
    error('Not enough input arguments.');
elseif nbIn > maxIn
    error('Too many input arguments.');
end
okOut = [0:1 3:5];
if ~any(okOut==nbOut)
    error('Invalid number of output arguments.');
end
 if nbIn == minIn
    x = varargin{1}; indarg = 2;
    if min(size(x))~=1, dim = 2;
            end
else
    c = varargin{1}; l = varargin{2}; indarg = 3;
    if min(size(l))~=1, dim = 2;
     end
end
 % Get Inputs
w    = varargin{indarg};
n    = varargin{indarg+1};
thr  = varargin{indarg+2};
sorh = varargin{indarg+3};

if (o=='gbl' & nbIn==7)  | (o=='lvd' & nbIn==6)

    if dim == 1, [c,l] = wavedec(x,n,w);
    else,        [c,l] = wavedec2(x,n,w);

    end
end

% Wavelet coefficients thresholding.
if o=='gbl'
    if keepapp
        % keep approximation.
             cxc = c;
                    if dim == 1, inddet = l(1)+1:length(c);
        else, inddet = prod(l(1,:))+1:length(c); end
        % threshold detail coefficients.
        cxc(inddet) = wthresh(c(inddet),sorh,thr);
        else
            % threshold all coefficients.
            cxc = wthresh(c,sorh,thr);
        end
    else

        if dim == 1, cxc = wthcoef('t',c,l,1:n,thr,sorh);
        else

            cxc = wthcoef2('h',c,l,1:n,thr(1,:),sorh);
            cxc = wthcoef2('d',cxc,l,1:n,thr(2,:),sorh);
            cxc = wthcoef2('v',cxc,l,1:n,thr(3,:),sorh);
        end
    end
    lxc = l;

    % Wavelet reconstruction of xd.
    if dim == 1,xc = waverec(cxc,lxc,w);
    else
        xc = waverec2(cxc,lxc,w);
    end

    if nbOut<4 , return; end

    % Compute compression score.
    perf0 = 100*(length(find(cxc==0))/length(cxc));
    if nbOut<5 , return; end

    % Compute L^2 recovery score.
    nc = norm(c);

    if nc<eps
        perfl2 = 100;
    else
        perfl2 = 100*((norm(cxc)/nc)^2);
    end
    % size(xc)
    % CR= size(c)/size(cxc)
```

4.5.3 Results and comparisons:-

Original

Result

Normal recovery= 93.4616
Compression score =99.2711

4.6 Modified application 3

4.6.1 Main program:-

```
close all;
clear all;
summ=0;
[X,map] = imread('lena512color.tiff');
I= rgb2gray(X);
I1=double(I);
figure (1);
imshow(I);
[cA,cH,cV,cD]=lwt2(I1,'db3');

figure (2);
cA=uint8(cA);
cA=double(cA);
% imshow(cA)
[thr,sorh,keepapp]=ddencmp_mod('cmp','wv',cH);

[xd1,cxd,lxd,perf0,perfl2] = wdencmp_mod('lvd',cH,'db3',2,thr,sorh);

[thr,sorh,keepapp]=ddencmp_mod('cmp','wv',cV);

[xd2,cxd,lxd,perf0,perfl2] = wdencmp_mod('lvd',cV,'db3',2,thr,sorh);

[thr,sorh,keepapp]=ddencmp_mod('cmp','wv',cD);

[xd3,cxd,lxd,perf0,perfl2] = wdencmp_mod('lvd',cD,'db3',2,thr,sorh);

figure (3);
axis('square');
xd1=uint8(xd1);
xd11=double(xd1);

figure (4);
axis('square');
xd2=uint8(xd2);
size(xd2)
xd22=double(xd2);

figure (5);
axis('square');
xd3=uint8(xd3);
xd33=double(xd3);


perf0
perfl2
```

```
p=ilwt2(cA,xd11,xd22,xd33,'db3');
figure (4);
p=double(p);
p=uint8(p);
imshow(p);
```

4.6.2 Functions used:-

ddencmp_mod ()*

```
function [thr,sorh,keepapp,crit] = ddencmp_mod(dorc,worwp,x)
% Check arguments.
nbIn = nargin;
if nbIn<3
    error('Not enough input arguments.');
elseif isequal(worwp,'wv')
    if (nargout>3)
        error('Too many output arguments.');
    end
elseif isequal(worwp,'wp')
    if (nargout>4)
        error('Too many output arguments.');
    end
else
    error('Invalid argument value');
end


% Set problem dimension.
if min(size(x))~=1, dim = 2; else , dim = 1; end

% Set keepapp default value.
keepapp = 1;

% Set sorh default value.
if isequal(dorc,'den') & isequal(worwp,'wv') , sorh = 's'; else ,sorh =
'h'; end

% Set threshold default value.
n = prod(size(x));

% nominal threshold.
switch dorc
  case 'cmp' ,  thr = 1;
end

% rescaled threshold.
if dim == 1
    [c,l] = wavedec(x,1,'db1');
    c = c(l(1)+1:end);
else
    [c,l] = wavedec2(x,1,'db1');
    c = c(prod(l(1,:))+1:end);
end

normaliz = var(abs(c))+6;

% if normaliz=0 in compression, kill the lowest coefs.
if dorc == 'cmp' & normaliz == 0
    normaliz = 0.05*max(abs(c));
end

if dorc == 'cmp'
     thr = thr*normaliz;
    if worwp == 'wp', crit = 'threshold'; end
end

thr_h = [thr*normaliz   thr*normaliz+2];     % Horizontal thresholds.
thr_d = [thr*normaliz+1 thr*normaliz+3];     % Diagonal thresholds.
thr_v = [thr*normaliz+3 thr*normaliz+1];     % Vertical thresholds.

thr = [thr_h ; thr_d ; thr_v] ;
sorh
keepapp
% crit
```

wdencmp_mod()*

```
function [xc,cxc,lxc,perf0,perfl2] = wdencmp(o,varargin)
% Check arguments and set problem dimension.
klo=10;
dim = 1;         % initialize dimension to 1D.
nbIn  = nargin;
nbOut = nargout;
switch o
    case 'gbl' , minIn = 7; maxIn = 8;
    case 'lvd' , minIn = 6; maxIn = 7;
    otherwise  , error('Invalid argument value.')
end
if nbIn < minIn
    error('Not enough input arguments.');
elseif nbIn > maxIn
    error('Too many input arguments.');
end
okOut = [0:1 3:5];
if ~any(okOut==nbOut)
    error('Invalid number of output arguments.');
end
 if nbIn == minIn
    x = varargin{1}; indarg = 2;
    if min(size(x))~=1, dim = 2;
            end
else
    c = varargin{1}; l = varargin{2}; indarg = 3;
    if min(size(l))~=1, dim = 2;
     end
end
```

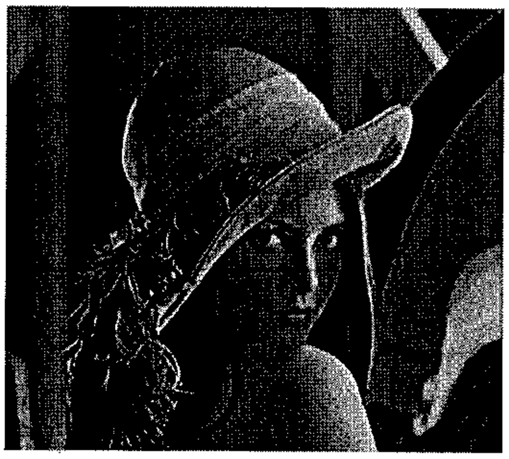
```
 % Get Inputs
w    = varargin{indarg};
n    = varargin{indarg+1};
thr  = varargin{indarg+2};
sorh = varargin{indarg+3};

if (o=='gbl' & nbIn==7)  | (o=='lvd' & nbIn==6)

    if dim == 1, [c,l] = wavedec(x,n,w);
    else,        [c,l] = wavedec2(x,n,w);

    end
end

% Wavelet coefficients thresholding.
if o=='gbl'
    if keepapp
        % keep approximation.
             cxc = c;
                    if dim == 1, inddet = l(1)+1:length(c);
        else, inddet = prod(l(1,:))+1:length(c); end
        % threshold detail coefficients.
        cxc(inddet) = wthresh(c(inddet),sorh,thr);
    else
        % threshold all coefficients.
        cxc = wthresh(c,sorh,thr);
    end
else

    if dim == 1, cxc = wthcoef('t',c,l,1:n,thr,sorh);
    else


        cxc = wthcoef2('h',c,l,1:n,thr(1,:),sorh);
        cxc = wthcoef2('d',cxc,l,1:n,thr(2,:),sorh);
        cxc = wthcoef2('v',cxc,l,1:n,thr(3,:),sorh);
    end
end
lxc = l;

% Wavelet reconstruction of xd.
if dim == 1,xc = waverec(cxc,lxc,w);
else
    xc = waverec2(cxc,lxc,w);
end

if nbOut<4 , return; end

% Compute compression score.
perf0 = 100*(length(find(cxc==0))/length(cxc));
if nbOut<5 , return; end

% Compute L^2 recovery score.
nc = norm(c);

if nc<eps
```

```
    perfl2 = 100;
else
    perfl2 = 100*((norm(cxc)/nc)^2);
end
% size(xc)
% CR= size(c)/size(cxc)
```

4.6.3 Results and comparisons:-

Original

Result



Normal recovery= 93.2613
Compression score = 99.4317

Conclusion

The conclusion that we would like to draw from this project is that we have achieved a better compression of images than that was existing. This was done by careful examination of the text that we consulted as well as the books that we referred to. Forming a basis of our approach. This lead to the new algorithm that we have developed which gives us better results than that of previous one.

Another point that should be taken into consideration is that we found that there still are few areas that can be improved upon, namely:-

1. The threshold can be selected more precisely
2. The input given to the newer algorithm can be altered to achieve better outputs

Thus we can say that these areas can be of imminent importance for the improvement of image compression.

Brief Bio data

AMAN VERMA- Department of Computer Science & Engineering
Jaypee University Of Information Technology,
Waknaghat, Solan (H.P)

Pursuing	**Name Of School/University**	**Year**	**CGPA**
B.Tech CSE	Jaypee University Of Information Technology, Solan (H.P.)	2011	6.7 (73.03%) Up Till 7th sem

Currently working on the **Image compression using second generation lifting scheme.**
Team size: Two

PUNEET TANEJA- Department of Computer Science & Engineering
Jaypee University Of Information Technology,
Waknaghat, Solan (H.P)

Pursuing	**Name Of School/University**	**Year**	**CGPA**
B.Tech CSE	Jaypee University Of Information Technology, Solan (H.P.)	2011	6.3(70.0%) Up Till 7th sem

Currently working on the **Image Compression Using Second Generation Lifting Scheme.**
Team size: Two

References

I. yEric J. Stollnitz, Tony D. DeRose, and David H. Salesin. Wavelets for computer graphics: A primer, part 1. *IEEE Computer Graphics and Applications*,15(3):76–84, May 1995.

II. Wim Sweldens .The Lifting Scheme: A Construction Of Second Generation Wavelets May 1995, Revised November 1996 To appear in SIAM Journal on Mathematical Analysis

III. Laurent Demaret (GSF-IBB Neuherberg, Germany),Nira Dyn (Tel-Aviv University, Israel), and Armin Iske (University of Leicester, UK) "Image Compression by Linear Splines over Adaptive Triangulations"

IV. Rajesh Siddavatam.2009 International Conference on Advances in Recent Technologies in Communication and Computing Image Noise Cancellation by Lifting Filter Using Second Generation Wavelets (LFSGW)Kottayam, Kerala, India October 27-October 28 ISBN: 978-0-7695-3845-7

V. Rajesh Siddavatam international conference on advances in recent technology in communication and computing,2009.ARTCom '09,27-28 Oct. 2009,pages on 509-513,kottayam,Kerela,978-1-4244-5104-

VI. B. Pradhan, Shattri Mansor. Institute for Advanced Technologies (ITMA)Faculty of Engineering, University Putra Malaysia, 43400, UPM, Serdang Selangor Darul Ehsan, Malaysia

VII. Book on image processing by Rafael C. Gonzales and Richard E. Woods

VIII. N. Dyn, M. S. Floater, and A. Iske, "Adaptive Thinning for Bivariate Scattered Data", J. Comput. Appl. Math. 145(2), 2002, pp. 505-517.

IX. Y. Eldar, M. Lindenbaum, M. Porat, and Y.Y. Zeevi, "The Farthest Point Strategy for Progressive Image Sampling", IEEE Trans. Image Processing 6(9), Sep. 1997, pp. 1305-1315.

X. Iske, Multiresolution Methods in Scattered Data Modelling,Springer, Heidelberg, 2004.

XI. M. Jansen, R. Baraniuk, and S. Lavu, "Multiscale Approximation of Piecewise Smooth Two-Dimensional Functions using Normal Triangulated Meshes", Appl. Comp. Harm. Anal. 19(1), 2005, pp. 92-130.

XII. E. LePennec and S. Mallat, "Sparse Geometric Image Representation with Bandelets", to appear in IEEE Transactions on Image Processing.S. Mallat, "A Theory for Multiresolution Signal Decomposition: The "Wavelet Representation", IEEE Trans.

XIII. Pradhan, Shattri Mansor, Abdul Rahman Ramli and Abdul Rashid B. Mohamed Sharif, K. Sandeep,"A New Robust Data Comprssor For Lidar Data ",Institute for Advanced Technologies (ITMA) Faculty of Engineering, University Putra Malaysia, 43400, UPM, Serdang