Sheffield Hallam University

Novel methods of image compression for 3D reconstruction

SIDDEQ, Mohammed Mustafa

Available from Sheffield Hallam University Research Archive (SHURA) at:

http://shura.shu.ac.uk/18148/

This document is the author deposited version. You are advised to consult the publisher's version if you wish to cite from it.

Published version

SIDDEQ, Mohammed Mustafa (2017). Novel methods of image compression for 3D reconstruction. Doctoral, Sheffield Hallam University.

Repository use policy

Copyright © and Moral Rights for the papers on this site are retained by the individual authors and/or other copyright owners. Users may download and/or print one copy of any article(s) in SHURA to facilitate their private study or for non-commercial research. You may not engage in further distribution of the material or use it for any profit-making activities or any commercial gain.

Sheffield Hallam University Research Archive
http://shura.shu.ac.uk

Novel Methodsof Image Compression for 3D Reconstruction

Mohammed Mustafa Siddeq

Supervisor: Professor Marcos A. Rodriguez

A thesis submitted in partial fulfilment of the requirements of Sheffield Hallam University for the degree of
Doctor of Philosophy

October 2017

Declaration

I hereby declare that the work described in this thesis is my own work, done by me and has not been submitted for any other degree anywhere.

Acknowledgements

First and foremost I would like to express my deepest appreciation to my Director of Studies, Prof. Marcos A. Rodrigues. It has been a pleasure to work under his supervision for the past 4 years. His advice, enthusiasm, encouragement and dedication have truly been inspirational. Also I would like to thank the Iraqi-Cultural-Attaché/Embassy of the Republic of Iraq in London for their financial support through a PhD studentship.

I would like to express my gratitude to my sister Israa and her husband Ahmed Saki for their support and encouragement. Also, I would like to express my thanks tomy friend Stephanie from Sheffield University for her encouragement. I would like also to thank my friends in Unit 12, members of Cultural, Communication and Computing Research Institute, Sheffield Hallam University.

Thank you to my supportive wife Damat and my father Mustafa Siddeq without them, I wouldn't have been successful. Although this journey is quite challenging, their patience, encouragement and continuous support have motivated me to complete this PhD. Last but not least, I would like to thank my best friend Asst. Prof. AbdulrhmanIkrahmSiddiq for his encouragement.

List of Publications

Patent publication

UK patent application GB 1503433.3, 28 Feb 2016.

Sheffield Hallam University, Mohammed M Siddeq, and Marcos A Rodrigues (2016). *Image Data Compression and Decompression Using Minimise Size Matrix Algorithm.* WO 2016/135510 A1.

Journal publications

A. Siddeq and Rodrigues (2017). A novel high-frequency encoding algorithm for image compression, *EURASIP Journal on Advances in Signal Processing 2017:26.* DOI 10.1186/s13634-017-0461-4

B. SIDDEQ, M. M. and RODRIGUES, Marcos (2017). **DCT and DST based Image Compression for Structured Light 3D Reconstruction,** 3D Research-Springer, Vol. 8, No. 1. DOI: 10.1007/s13319-017-0116-0

C. SIDDEQ, M. M. and RODRIGUES, A. Marcos (2016) Novel 3D Compression Methods for Geometry, Connectivity and Texture, 3D Research-Springer, Vol. 7 No.2. DOI: 10.1007/s13319-016-0091-x

D. RODRIGUES, A. Marcos and SIDDEQ, M. Mohammed (2016). Information Systems: Secure Access and Storage in the Age of Cloud Computing. *Athens Journal of Sciences*, **3** (4), 267-284.

E. SIDDEQ, M. M. and RODRIGUES, A. Marcos (2015). A novel 2D image compression algorithm based on two levels DWT and DCT transforms with enhanced Minimize-Matrix-Size algorithm for high resolution structured light 3D surface reconstruction. 3D Research-Springer, 6 (3), p. 26.DOI: 10.1007/s13319-015-0055-6

F. SIDDEQ, M. M. and RODRIGUES, A. Marcos (2014). A novel image compression algorithm for high resolution 3D reconstruction. 3D Research-Springer, 5 (7), 17 pages. DOI: 10.1007/s13319-014-0007-6

Conference proceedings

A. SIDDEQ, M.M. and RODRIGUES, Marcos (2016). 3D Point Cloud Data and Triangle Face Compression by a Novel Geometry Minimization Algorithm and Comparison with other 3D Formats. In: The 7th International Conference on Computational Methods (ICCM2016), Berkeley California, USA, August 1-4, 2016.

B. SIDDEQ, M and RODRIGUES, Marcos (2015). Applied sequential-search algorithm for compression-encryption of high-resolution structured light 3D data. In: BLASHKI, Katherine and XIAO, Yingcai, (eds.) MCCSIS:Multiconference on Computer Science and Information Systems 2015. IADIS Press, 195-202.

C. SIDDEQ, M.M. and RODRIGUES, Marcos (2014). A new 2D image compression technique for 3D surface reconstruction. In: MASTORAKIS, Nikos, PSARRIS, Kleanthis, VACHTSEVANOS, George, DONDON, Philippe, MLADENOV, Valeri, BULUCEA, Aida, RUDA, Imre and MARTIN, Olga, (eds.) Advances in information sciences and applications : Proceedings of 18th International Conference on Computers (part of CSCC'14). Recent advances in computer engineering series, 1 (22). World Scientific and Engineering Academy and Society (WSEAS), 379-386

Abstract

Data compression techniques are widely used in the transmission and storage of 2D image, video and 3D data structures. The thesis addresses two aspects of data compression: 2D images and 3D structures by focusing research on solving the problem of compressing structured light images for 3D reconstruction. It is useful then to describe the research by separating the compression of 2D images from the compression of 3D data. Concerning image compression, there are many types of techniques and among the most popular are JPEG and JPEG2000. The thesis addresses different types of discrete transformations (DWT, DCT and DST) thatcombined in particular ways followed by Matrix Minimization algorithm,which is achieved high compression ratio by converting groups of data into a single value. This is an essential step to achieve higher compression ratios reaches to 99%. It is demonstrated that the approach is superior to both JPEG and JPEG2000 for compressing 2D images used in 3D reconstruction. The approach has also been tested oncompressing natural or generic 2D images mainly through DCT followed by Matrix Minimization and arithmetic coding.Results show that the method is superior to JPEG in terms of compression ratios and image quality, and equivalent to JPEG2000 in terms of image quality.

Concerning the compression of 3D data structures, the Matrix Minimization algorithm is used to compress geometry and connectivity represented by a list of vertices and a list of triangulated faces. It is demonstrated that the method can compress vertices very efficiently compared with other 3D formats. Here the Matrix Minimization algorithm converts each vertex (X, Y and Z) into a single value without the use of any prior discrete transformation (as used in 2D images) and without using any coding algorithm. Concerningconnectivity,the triangulated face data are also compressed with the Matrix Minimizationalgorithm followed by arithmetic coding yielding a stream of compressed data. Results show compression ratiosclose to 95% which are far superior to compression with other 3D techniques.

The compression methods presented in this thesis are defined as per-file compression. The methods to generate compression keys depend on the data to be compressed. Thus, each file generates their own set of compression keys and their own set of unique data. This feature enables application in the security domain for safe transmission and storage of data. The generated keys together with the set of unique data can be defined as an encryption key for the file as, without this information, the file cannot be decompressed.

Table of contents

Chapter 1
Introduction

1.1. Introduction

In the last decade, we have seena large expansion in the way we communicate through digital media, and still the process is in progress. Theenablers include a growing Internet infrastructure with increased bandwidth, storage space and user base; the explosive development of mobile communications; and the increasing importance of image andvideo data in communication. Data compression is one of the technologies used to squeeze data size (i.e. image, video, and audio) to reduce communication delay times. Websites are not restricted to images, they also contain video and audio and for this reason data compression algorithms are required **[1]**. For instance, smart phones would not be able to provide fast communication without data compression. Digital TV cannot be realized without data compression. Data compression is used in all domains, for example: make a long-distance call through digital media requires data compression **[2]**. Other examples include fax, and listening to music on your player or watch a DVD require data compression **[3]**.

Data compressions algorithms are used to reduce the amount of data required to save an image or a video or music. **[3]**. In brief, data compression is the art and science for representing data in a compact form, the structures that exist in the data, the samples of audio or image waveforms or numbers that are generated by processing **[4]**.Data compression encodes information used or generated in digital form (numbers represented by bytes), and these numbers are used to represent multimedia files. For example, to represent 1 second of movie without compression (using the CCIR 601 format) it is required approximately 20 MB, or 160 MB, for this reason we need to compress each frame in the movie.To represent 2 minutes of uncompressed music (44,100 samples per second, 16 bits per sample) requires approximately 84 Mb. To downloading this uncompressed music from a website, or streaming the same file, longtime spans arerequired **[3]**.

As any human activity has an influence on the environment, there is an increasing need for information about the environment to reduce harmful impacts. Various space agencies from around the world, including the European Space Agency (ESA), the National Aeronautics and Space Agency (NASA), the Canadian Space Agency (CSA), and the Japanese Space Agency (STA), are co-operating on a program to observe changes in the global environment that generate many terabytes of data per day; compare this with 130 terabytes of data currently stored at the EROS data center in South Dakota, which is the largest archive in the world **[5]**.

According to growth of data that needs to be transmitted and saved, there is the question of why there is not a more concerted effort on developing sophisticated transmission and storage technologies? This is happening, but still it is not enough. There have been significant developments that allow transmitting and storing large amounts of information without using compression, including CD-ROMs, optical fibres, Asymmetric Digital Subscriber Lines (ADSL), and cable modems **[6]**. However, as both storage and transmission capacities increase with new technological innovations, as a outcome to Parkinson' s FirstLaw: "Work expands so as to fill the time available," in Parkinson' s Law and Other Studies in Administration, by Cyril

Northcote Parkinson, Ballantine Books, New York, 1957**[7]**.It seems that the need for mass storage and transmission increases at least twice as fast than storage and transmission capacities. There are obvious physical limitations, such as the amount of information we can transmit over the airwaves will always be influenced by the characteristics of the atmosphere **[5]** and the limitations of bandwidth capacity.

The main motivation for the research stems from the large size of 3D face meshes generated by the GMPR 3D scan technologies [69] (around 20MB for geometry and connectivity plus 4MB for texture mapping) which limited their intended application in a security context. The GMPR technologies were tried in an airport scenario in a confidential trial with the main outcome being the realization that the size of the data files made the system unworkable. The idea was that each passenger was to be scanned at check-in and again at the gates before boarding the plane. All data were to be sent to the Police who had exactly 24 hours to perform background checking. If needed, the Police could apply for judicial order to keep the data for longer, otherwise all data had to deleted within 24 hours. If the plane were say, from London to New York, all data would be sent to the Police in New York before taking off. The law enforcement abroad would have the same limitations of keeping the data as the UK Police.

Around 70 million passengers per year go through London Heathrow Airport and a simple multiplication of two 3D scans per passenger by the required disk storage clearly indicates the need forhuge bandwidth requirements to transfer data. Having identified the bottleneck, the solution would require data compression of at least 2 orders of magnitude, say a file of 24 MB be reduced to around 240 KB. From these considerations, thefollowing research questionsare posed:

- *Can a method be developed to efficiently compress geometry, connectivity and texture by 2 orders of magnitude*?
- *Can the method be generic enough to be applied to any kind of data such as text, video and audio*?

To investigate the issue, the starting point must be 2D image compression as the GMPR 3D technologies are based on generating 3D meshes from 2D images. Even if 3D data are not generated by structured light techniques, 2D images are used for texture mapping of 3D models, thus, there is a clear requirement for 2D image compression. In the following sections, emphasis is placed on two popular and standard image representation techniques namely JPEG and JPEG2000 that keep the data in compressed format. Many other representations exist such as BMP, PNG, TIFF, GIF, and so on. Some are more efficient than others in terms of image size but these will not be considered in this thesis for comparative analyses. The thesis focuses on JPEG2000, which is the best and the standard method of image compression so any proposed compression technique should stake its case against JPEG2000, and on JPEG, which is a popular standard technique for image and video coding.

1.2. Compression Techniques

When we talk about compression techniques or compression algorithms [8] we are actually referring to two algorithms. There is the compression algorithm that takes an input X and generates a new stream of data x_c that requires fewer bits, and the

decompression algorithm operates on x_c to reconstruction Y. These operations are shown in Figure 1.1.

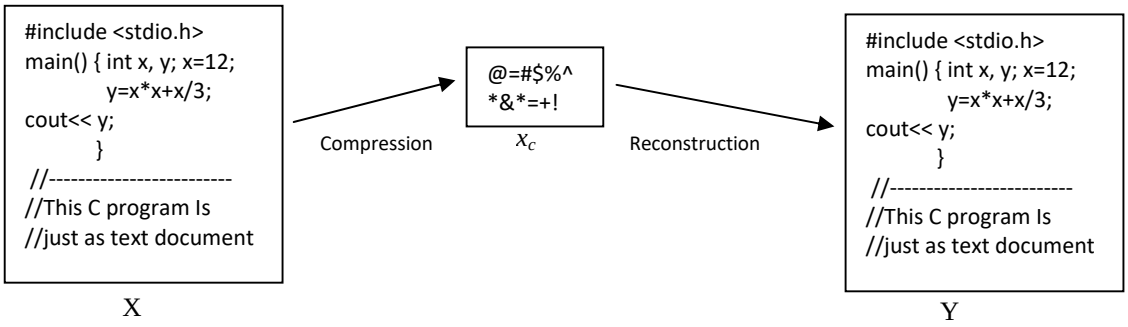


Figure 1.1: Compression and Reconstruction

We will follow current understanding and refer to both the compression and decompression algorithms to mean the compression algorithm based on the requirements of reconstruction.Data compression programs can be divided into two broad classes: lossless compression algorithms, in which Y is identical to X, and lossy compression algorithms, which provide higher compression than lossless compression, and the result Y is not strictly equal to X but not much different**[6, 10]**.

1.2.1. Lossless Data Compression

Lossless compression techniquesimply no loss of information. If data have been lossless compressed, the original data can be recovered exactly as the original data. Lossless compression is generally used for applications like: texts, word documents, executable files, and library documents used by computer programming languages (i.e. C++, java or any machine languages)**[11]**.

Text compression is an important area for lossless compression. It is very important that the reconstruction is identical to the original text, even very small differences between original and recovered data can result in a big change in meaning. Consider the original text: "*Do not send money*" and received sentences: "*Do now send money*". If any kind of data are enhanced (after compression algorithm) they may yield additional information [**12]**.For example, assume we compressed a radiological image as lossy type, and the difference between the decompressed image and the original image was visually negligible. If this image were later enhanced, it may cause the appearance of artifacts that could delude the radiologist. Because the cost could be a human life, it is important to be very careful about using a compression method that, after reconstruction, shows significant differences (large and small) from the original image **[13]**.

Data obtained from satellites are often processed later to obtain numerical indicators about our environment. If the reconstructed data are not approximately identical to the original data, the result is enhanced data. In this caseit may be impossible to recover the original data. Therefore, in processing satellite data it is normally not allowed for any differences or degradation to happen in the compression process **[14, 15, 16]**.

There are many applications that require data compression where the reconstructed data must be identical to the original. However, there are also a myriad of applications in which some data loss is acceptable and indeed, it is requirementtoobtain higher compression ratios.

1.2.2. Lossy Data Compression

Lossy compression techniques lead to loss of some information, and data that have been compressed generally cannot be recovered or reconstructed identically to the original. By accepting lossy in the reconstructed image, we can generally obtain much higher compression ratios than is possible with lossless compression **[17]**.

In many applications, exact reconstruction is not necessary. For example, when transmittingspeech or an image or video. Depending on the quality required of the reconstructed data, some reduction in data accuracy can be tolerated **[18]**. However, if the reconstructed data needs to be of high quality, the amount of information loss must be controlled to low levels **[19]**.

Once we have specified data compression requirements, we need to measure the data compression-decompression performance. Performance is elusive as it can refer to the perceived accuracy or some quantifiable parameter. This is sobecause in different fields of application, different terms have been used to describe performance measurements **[20]**. In this thesis we focus on the perceived quality, the compression ratios, and the root mean square errors between reconstructed and original data.

1.3. 2D Image Compression

A digital image is a matrix of dots, arranged in *n* rows and *m* columns. The expression [m n] represents the resolution of the image and the dots refer to pixels. The term “resolution” is used to show the number of pixels per unit length of the image. Therefore, dpi stands for dots per inch [21]. We can classify the images in the following: **[22, 23, 24]**:

A. *A monochromatic image.* This simplest type of images contains two values referring to black and white, each pixel represented by one bit.
B. *A grayscale image*. In this type of image pixel values are between 0 and 255, and each pixel is represented by a byte.
C. *Colour image/natural image*. This type of image is similar to grayscale type, but consist of three layers RGB (Red, Green and Blue), pixel values in each layer between 0 and 255. A pixel in three layers can be represented by 24-bit [25]. When adjacent pixels are slightly similar, it may be impossible for the eye to recognize their colour. This type of image can obtained for example, from digital cameras. Figures 1.2 shows a typical example of colour images [3].
D. *A graphical image.* This type of image is normally an artificial image: cartoon or a graph that contains texts (obtained from Photoshop programs, or any paint programs). It may have a few colours (not natural), free from noise or blurring as appears in a natural image.

It is axiomatic that each type of image has redundancy in colour, but they are redundant in different ways. Therefore, image compression algorithms cannot

perform well acrossall images and different algorithms are required to compress different types of images [26]. There are also methods that break an image into parts, to render compression easier [27].

Figure 1.2: (left) Lena and details, this image is part of the Play- Boy centerfold for November, 1972, (right) Mandrill and details, used as sample in MATLAB language for image processing test

The idea of lossy compression becomes more agreeable with digital images; this is because the images are created by the following: 1) an image may be scanned from a photograph and digitized (i.e. converted to pixels); 2) An image may be captured by a digital camera that creates pixels and save them directly in memory; 3) An image may be painted on the screen (i.e. paint software) [28].In all these cases, some data is lost when the image is digitized. Normally the viewer accepts this loss of information if done properly. Digitizing an image can be defined in two steps: sampling and quantization. Sampling an image is the process dividing the original image into small regions of pixels. Quantization is the process of assigning an integer value to each pixel (i.e. thresholding)[29], for example, if a pixel value is greater than a threshold, the pixel value changes, otherwise no action.

We present a simple process that can be used as a measurement to determine the amount of data loss in a compressed image. For example, an original image M is compressed to T, and then decompress T to S, finally subtract V = S−M. if the image S is identical to original image M, then V should be uniformly black (V=0). Ifsomedetails are lost during compression,V would be approximately black [30] and may be acceptable, depending on the application.

1.3.1. The JPEG Technique

The name JPEG is a shortcut that stands for Joint Photographic Experts Group. This was a joint effort by the CCITT and the ISO (the International Standards Organization) that started in 1987 and produced the first JPEG draft proposal in 1991. The JPEG standard has proved successful and has become widely used for image compression, especially in Web pages **[21, 27,30]**.

JPEG is a lossy/lossless compression method for colour or grayscale still images. An important feature of JPEG is its use ofparameters, allowing the user to adjust the amount of data quality or compression ratio. Often, the eye cannot see any image degradation even at compression ratio more than 80%.For this reason, most implementations support JPEG method lossy mode [31,32,33].The main JPEG compression steps are described as follows:

A- Colour images are transformed from RGB into a luminance and chrominance colour space (YCbCr format). The eye is sensitive to small changes in luminance (Y) but not in chrominance (CrCb),so the chrominance part can

later be compressed at high compression ratio, without losing much visual quality [31]. Figure 1.3 shows the RGB conversion, the main reason for this conversion is to achievehigher compression ratios.

B- Each layer in an image (Y or Cb or Cr) is divided into non-interleaved blocks (8x8) pixels, and each block is compressed separately. If the image size is not compatible with 8×8 block, the bottom row and the rightmost column are duplicated.To be compatible, the JPEG encoder holds all the blocks of the first image layer, then the encoder operates on the second layer, and finally the encoder is applied on the third layer. If the user chooses maximum compression ratio, block artefacts appear in the decompressed image [31] as shown in figure 1.4.

C- The Discrete Cosine Transform (DCT) is applied to each block to create new 8×8 datablock of frequency components. It contains low-frequency at the top corner, while other values represent higher-frequency. The main advantage ofDCT is de-correlate the data, and some data in the right bottom are negligible. This process increases the compression ratio, while another advantage the reconstructed data is approximately similar to original data as the human eye cannot recognize these differences [29].

D- Each of the 64 frequency components in a block is divided by a separate number called quantization coefficient (QC), and then keeping an integer part. This is where original information becomes irretrievable. Large QCs values cause more losses. The JPEG compression algorithm implements a QC table for luminance and a different QC table for chrominance components [28].

E- The 64 quantized frequency coefficients (integer values) of each block are scanned to one-dimensional array then encoded by a combination of RLE and Huffman coding. An arithmetic coding different from Huffman coding known as the QM coder [32, 33] can optionally be used instead of Huffman coding [32]. Figure 1.5 shows JPEG steps to compress an image.

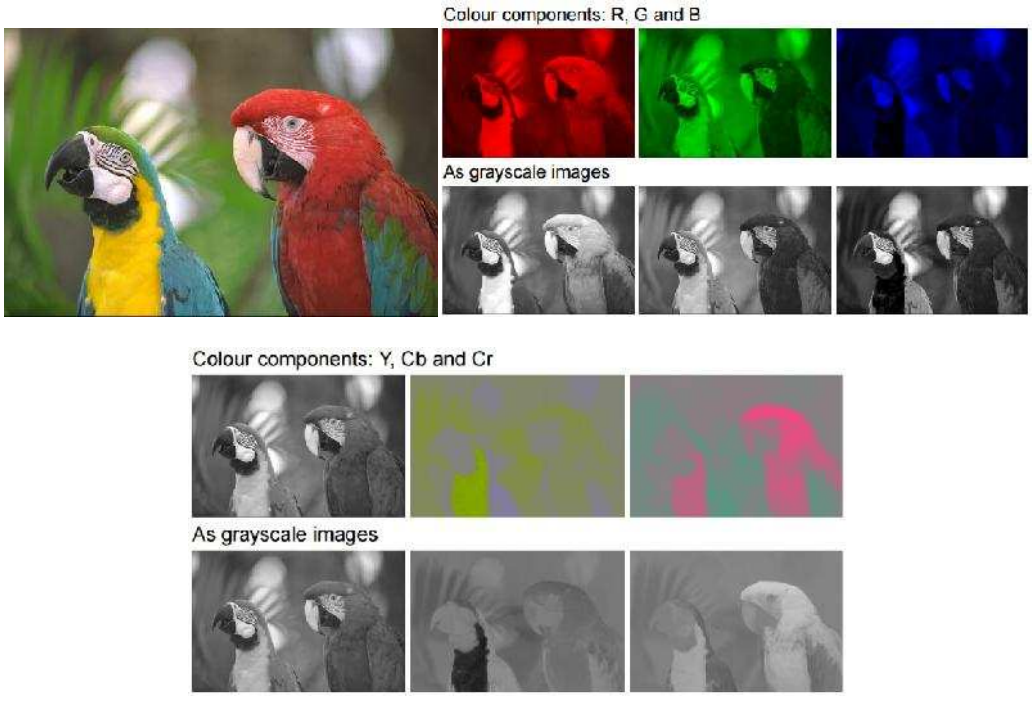


Figure 1.3: (Top) original image RGB, (bottom) RGB converted to three different layers YCbCr

Figure 1.4: Lena's picture compressed at bitrate 0.25, the artifact block appeared on the decompressed image (picture zoomed-in 3 times)

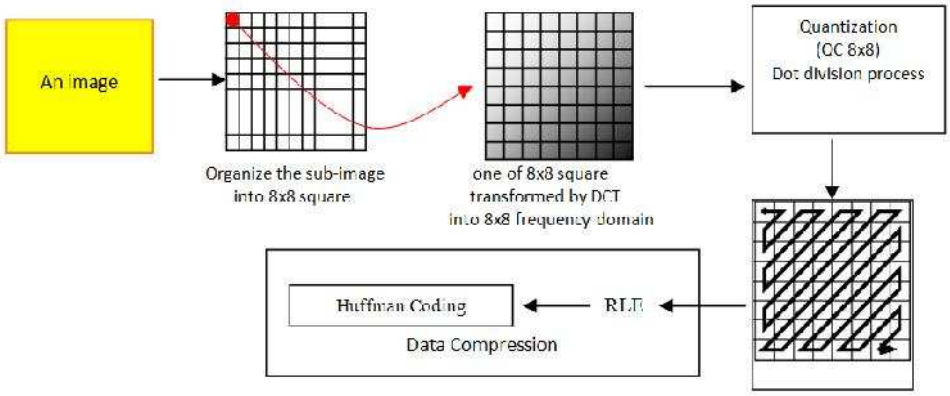


Figure 1.5: Layout JPEG compression steps for one layer or grayscale image

1.3.2. The JPEG 2000 Technique

The current JPEG standard provides excellent compression performance at rates above 0.25 bits per pixel. However, at lower rates there is a sharp degradation in the quality of the reconstructed image. To correct this and other shortcomings, the JPEG committee initiated work on another standard, commonly known as JPEG2000. The JPEG2000 is the standard based on wavelet decomposition **[34]**.The main sources of information on JPEG2000 are ISO/IEC (2000), International Standard IS 15444-1**[34, 36]** the final committee draft (FCD) released in March 2000. This document defines the compressed stream (referred to as the bit stream) and the operations of the decoder. It contains informative sections about the encoder, but any encoder that produces a valid bit stream is considered a valid JPEG2000 encoder. Following is a list of features JPEG2000 is expected to improve upon existing methods **[35]**:

A. High compression efficiency, bitrates less than 0.25 bpp are expected with high quality images.
B. The ability of DWT being appliedto large blocks of images or over the complete image (The JPEG maximum block size 8x8).
C. Progressive image transmission, the proposed standard can decompress an image progressively.
D. The decoder can decompress part of an image inside a ROI (Region Of Interest).

To illustrate JPEG2000 encoding algorithm, assume a colourimagethatis divided into three components by converting RGB into YCbCr format. Each component is partitioned into rectangular, non-overlapping regions called *tiles*, and each are compressed individually as illustrated in the following four steps[37]:

1- Compute a Digital Wavelet Transforms (DWT) the results issub-bands of coefficients LL (low frequency sub-band), LH, HL and HH are high-frequencies sub-bands.
2- The wavelet coefficients are quantized. This is done if the user specifies a target bitrate. The lower the bitrate, most of high-frequencies becomezeros (LH, HL and HH), roughly similar to quantizing the wavelet coefficients **[39]**.
3- It uses arithmetic coding tocompress each sub-band coefficients individually **[37]**.

The EBCOT algorithm (Embedded Block Coding with Optimized Truncation) **[38]** has been adopted for the encoding step. The principle of EBCOT is to divide each sub-band into blocks (i.e. code-blocks) that are coded individually. The bits resulting from coding several code-blocks become a packet and the packets are the components of the bit stream. The packets are used later by the decoder to decode specified area and skip other areas, thereby displaying the ROI. The packet bit stream is organized in layers. Each layer contains image information; therefore, decoding the image layer by layer is a natural way to achieve progressive data decompression **[37, 40]**. Figure 1.6 depicts the layout of the JPEG2000 compression technique.

In summary,current experiments indicate that JPEG2000 performs better than the previous JPEG, in still images at higher compression ratios or whenvery high image quality isrequired**[41]**.On the other hand, a decompressed image by JPEG2000 contains blurring at higher compression ratios, this is because multi-level DWT is applied **[43].**Figure 1.7 shows Lena's image compressed by JPEG2000.

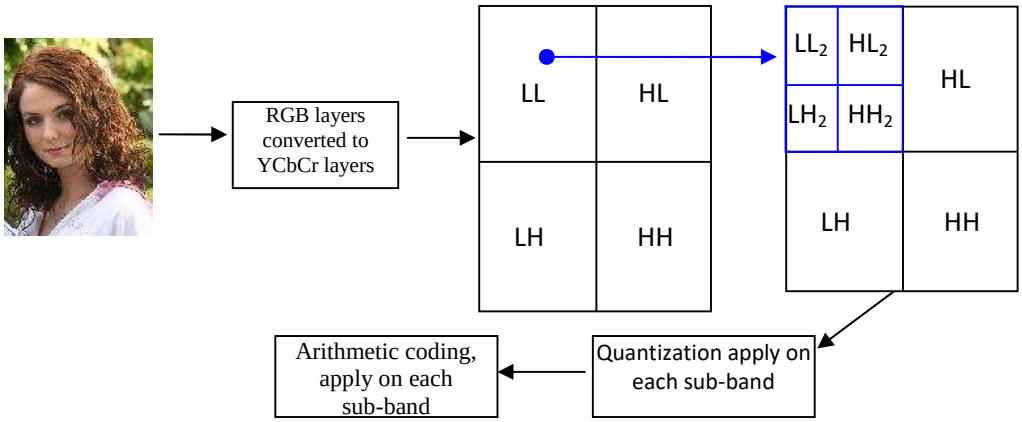


Figure1.6: Two levels DWT decomposition applied on each layer in acolour image to create six high-frequency coefficients matrices and one low-frequency coefficient matrix, each sub-band compress independently.

Figure 1.7: Lena's picture compressed at bitrates 0.25, the decompressed image zoomed-in 3 times.

1.4. 3D Surface Data

3D meshes may be the most popular discrete virtual surface and volume representation. Its simplicity makes it so popular today that electronic chips, called GPUs (Graphical Processing Units), partially specialized in the rendering of images from 3D meshes are integrated in all personal computers, and some tablets. However, their data representation takes a large amount of space **[42]**.

3D meshes are used in many application areas including computational simulation, entertainment, medical imaging, digital heritage, computer-aided design, e-commerce, etc.The need for precision is unstoppable, this leads to the generation of meshes composed of many elements, demanding processing, and complex visualization and storage.3D mesh compression has been an active research topic since the mid 90' s. Because 3D meshes are normally large data files, it is important that good compression methods are available for efficient storage and transmission [**44]**.

Two different methods exist for mesh compression: single-rate and progressive approaches. The advantage of the single-rate methods is that generally they output higher compression ratios. However, the reconstructed mesh is only available when all data are decoded at the decompression stage [45,46]. The progressive approaches are relevant; to cooperative visualization that requires fast data transmission, also a progressive compression approach allows achieving high compression ratio and produces different levels of detail. They provide the chance to obtain an unpolished pattern (version of 3D mesh that needs some enhancement) of the original object and to polish it progressively until the levels of details are the most suitable for the terminal client [47, 48].

A polygon mesh is defined by a set of vertices and by its triangle-face (incidence graph). The vertex description involves three coordinates per vertex: x, y and z [49]. Incidence (sometimes referred to as “topology”) defines each triangle-face by three integer values that refer to its vertices,as shown in figure 1.8[50,51]

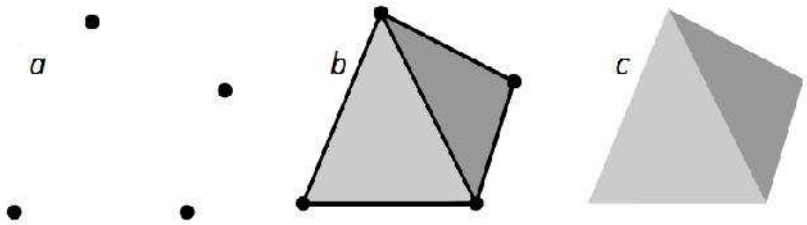


Figure 1.8: (a) vertices (b) triangle faces, (c) surface topology

Some mesh connectivity is said to be "regular" because they are collected together of the repetition of the same pattern. In case of few/some elements of the mesh do not have a regular connectivity, then the mesh is called "semi-regular". When there is no regular structure connectivity in the mesh, in this case it is called "irregular"[51, 52].

There are several 3D mesh structures, such as OBJ, VRML, X3D, COLLADA, etc. These methods are based on an indexed data structure. The first part of this structure consists of a list of all the vertices: X, Y and Z (coordinates) that also introduces an ordering of the mesh vertices. The second part describes the connectivity of the mesh consisting of a list of triangles faces. Each triangle is composed of three indices (i.e. 3-indices refer to 3 vertices in the 3D mesh file).Such data structure formats have the advantage of being simple. Additionally, most 3D mesh compression algorithms depend on this structure as input[53, 54]. Figure 1.9 shows the Wavefront’ s object file structure as an example. In this section, we will divide 3D mesh surface compression into two parts: (1) 3D mesh compression based on 3D data (2) 3D surface compression based on 2D Bitmap image.

1.4.1. 3D Mesh Compression Based on 3D Data

3D mesh compression techniques are different from compression methods forother multimedia (e.g. images, video). The common point between images and videos is that their structure is known (pixels value in an image are limited) by the compression and the decompression algorithm. While in 3D meshes the connectivity is completely unknown to the encoder before compression. So, besides compressing the geometry (vertex positions), a connectivity structure must also be encoded [55, 56].

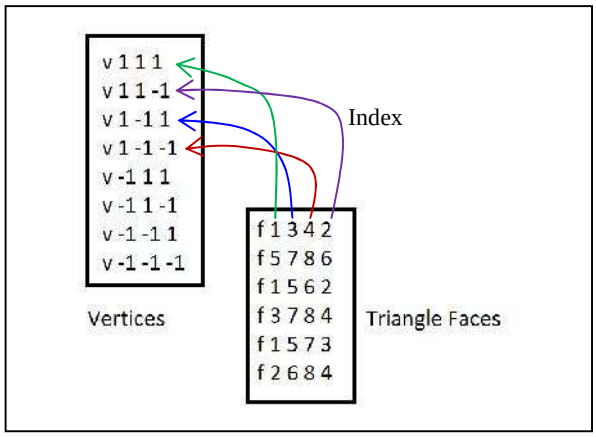


Figure 1.9:3D object triangle-face refers to indices in vertex

A. **Geometry Compression:**Mesh geometry compression represents compression the vertex coordinates, in most cases it is larger than the connectivity information (triangle faces). Usually the compression of the geometry of a mesh begins with the *Quantization* of all the coordinates. Next, in each compression algorithm iteration the position of anencoded vertex is predicted.Ifthe prediction is accurate, the prediction error is small so, it can be later efficiently coded by Huffman or Arithmetic coding [57, 58].

A vertex coordinates are represented by 96-bit floating-point numbers. This accuracy is not important for most 3D applications, in this case the quantization can significantly reduce the amount of data to encode without any distinguishable quality loss[59].The number of quantization bits usually ranges between {8 and 16 bits}, as accordingly, the mesh geometry softly changed [60].

B. **Connectivity Compression:** The general principle for all connectivity compression techniques is implementing a traversal to the mesh (vertices connectivity) and output a table of symbols depending on the configurations[61]. The main point in the traversal, defines a new set of numberings for the mesh, different from the one used in the input indexed data structure [62]. The generated symbols are encoded by Huffman coding or Arithmetic coding [63].

3D mesh connectivity compression requires high computational capabilities. The GPUs (Graphic Processor unit) can be usedto process a 3D mesh in parallel[64]. An optimized method to transfer mesh data can significantly decrease processing time such as the use of triangle strips.These strips are transferred from main memory to GPU memory. Triangle strips can be defined by a sequence of vertices, where a new vertex is added to atriangle strip created with two previous vertices. This method is much more efficient than the indexed representation that requires three vertices to encode each triangle [65,66, 67].

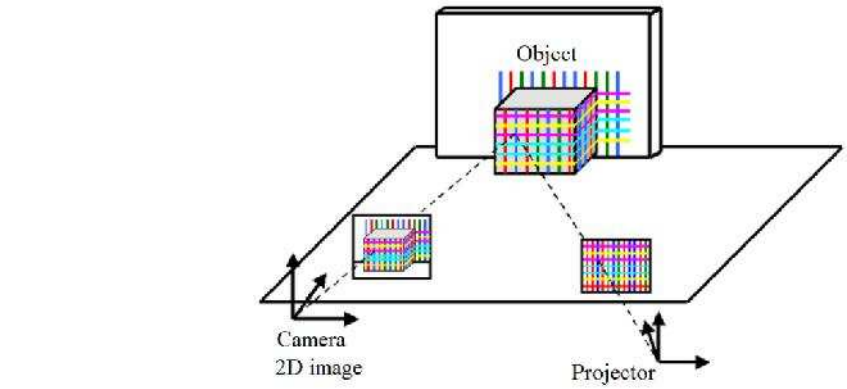
1.4.2 3D Mesh Compression Based on 2D Image

Still high-quality cameras and imaging sensors are able to obtain good 2D images, but these images do not contain full 3D object representation, because depth information is absent. This restriction limits our ability to extrapolate from a 2D image to a 3D surface. The last two decades characterized significant progress in research related to 3D surfaces through developed and commercialized 3D imaging techniques. This encouraged 3D applications that, in turn, demand high resolution and high speed electronic imaging systems [69].

3D images refer to techniques that are capable to obtaining 3D data (i.e. such as distribution of density for 3D object).Surface imaging measures the coordinates of an object' ssurface (i.e. measurement points x,y,z). Since the surface is normally non-planar the result of the measurement is a 3D surface image that can be described as a three dimensional matrix (i.e. first layer contain coordinates: x,y. while second layer contains depth z)

A general 3D surface imaging system based on structured lightor laser scanning canestimate a scalar value representing the depth from surface reflectance. Each point in the non-planar surface is represented as point cloud Pi (i.e. P_i= x_i, y_i, z_i and f_i)where$f_i$represents a colour at the i-th point in the surface as shown in Figure 1.10.

The GMPR group has developed and patented new 3D scanning methods at Sheffield Hallam University [70, 71] based on structured light. The methods convert a single image into a 3D surface by processing the light patterns in the image. The scanner processes a pattern of projected stripes on the target object. The shape of the captured pattern is combined with light source and the camera, to determine the 3D position of the surface along the pattern [72] as showed in the Figure 1.11. The system can work in real time enabling the concept of 3D CCTV to be implemented [73]. The issue is the massive amount of generated data which is addressed in this thesis.



Original object 2D image with structure-lights 3D data created

Figure 1.10: (Top) shows projector used spotlight on an object and camera used to capture 2D image for an object with patterns lights (structured lights) as a 2D image, (bottom) shows an example the 2D image converted to 3D object data.

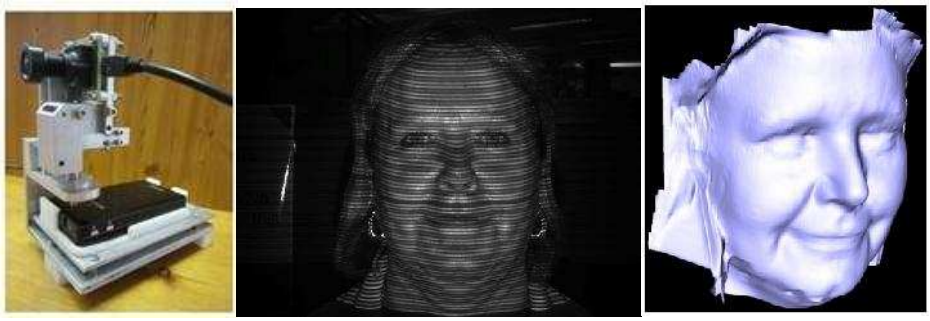


Fig. 1.11: (left) 3D scanner convert any object to 2D bitmap pattern image, (middle) 2D image captured by 2D camera on the top of the 3D scanner, (right) 3D mesh created by software developed by GMPR[70].

1.5. Aims and Objectives

In this section, we will mention how our proposed method is used in 2D images, 3D from structured light data and general data. Our proposed method focuses on reducing the amount of data to 2 orders of magnitude compared to standard uncompressed data.

Because there are many kinds of data (images, text, audio, video, 3D, etc.), the steps described by our approach differdepending on the type of data. For example, 2D images need discrete transformationsbefore coding, while in 3D meshes we do not necessarily apply discrete transformations to reduce the number of vertices. And for

other types of the data the method depends on redundant data and the relationship between data neighbours.

The purpose of the proposed method is to combine groups of data to a single value. Before we apply our proposed method, we must analyse the data (i.e. image or 3D data or text, or other type). Because some data cannot compressed without some pre-processing, as an example an image must be transformed before applying our proposed approach. In lossy compression, redundant vertices from 3D data are reduced before applying our proposed approach. Lossy compression allows for more flexibility on the number and type of transformations we can use before applying the main steps of the proposed compression method.

In lossless text data compression, our approach can be applied directly to the data.Inthat case, the compression ratio for a text file will not be very high, so it is not possible to reduce this type of data by 2 orders of magnitude. For this reason we can apply some lossless pre-process (i.e. RLE) **[3]**to reduce the amount of data followed by our proposed method to increase the compression ratio.

The aims of this thesis are to investigate and improve compression ratios and image quality in the context of 3D reconstruction placing in the context of current technologies. The quantitative target is reduction in file size by 2 orders of magnitude with acceptable quality parameters. The quality of decompressed image is computed by the Root-Mean-Square-Error (RMSE) technique for both 2D images and 3D surfaces. Also the Human Visual System (HVS) is used as a measure of image quality (i.e. it depends on human visual perception, as pure arithmetic is not a sufficient measure of quality between decompressed and original image data). The objectives of PhD research are highlighted in the following steps:

A) **Investigate discrete transformations and their combination for digital image processing, assessing their combined effects on compression and decompression**. The principle of operation we are pursuing is that in data compression, the discrete transformation should divide an image into low and high frequency bands. In case the number of high frequencies is increased, the compression ratio will increase. Additionally, some of the less significant high frequency coefficients can be neglected by quantization. The aim of the transformation is thus, to separate low frequencies from high frequencies. Similarly, low frequencies are subject to a possibly different or same type of transformation. By increasing the number of high frequency coefficients, higher compression ratios are obtained. Thereafter, a single type of discrete transformation is applied to the image and compared with a multi-level discrete transformation applied to the same image. Moreover, it is an objective of the dissertation to adapt the same transformations or sequence of operations to directly compress 3D geometry data.

B) **Investigate and develop novel ways to exploit wavelet decomposition to various areas of an image for efficient compression**. For example, by applying a two-level DWT decomposition: LL2, LH2, HL2 and HH2 on a 2D image, and

then using quantization process in LL2 to generate new low-frequency coefficients ($LL2_{quantized}$) sub-band. Furthermore, exploit the Matrix Minimization algorithm applied to high frequency matrices. The advantage of the Matrix Minimization algorithm method is that it decreases the matrix size to 1/3 of its original size and increases the compression ratio. The main reason for using the Matrix Minimization algorithm is because we have many matrices that must be reduced before using a coding algorithm. Additionally, Matrix Minimization algorithm applied to vertices in 3D geometry data to show the ability our method to compress 3D mesh surface. Firstly, the differences computed between each two vertices, and then our method applies to convert each 3 data to single value.

C) **Develop novel methods and algorithms for accurate decompression of high frequency matrices**. For instance, the Limited-Sequential Search Algorithm (LSS Algorithm) is used to decompress high frequency matrices. The LSS algorithm is a searching method used to find the original data matrix through space search (i.e. using a table that contains minimum and maximum values of the original matrix before compression).The LSS algorithm searches for the original data sequentially using the table (i.e. decode the original high-frequency matrix). So, this algorithm represents the inverse of the Matrix Minimization algorithm. If no search algorithm is used, it is not possible to decode the matrices and thus, not possible to decompress the 2D image/3D surface. Furthermore, we propose to develop this algorithm into a Block-Sequential-Search Algorithm, which represents two or more LSS-Algorithm working together and synchronize their results to speed up the searching method.

D) **To perform a comparative analysis of performance with standard compression algorithms for image and data structures.**JPEG and JPEG2000 represent the most popular image and video compression techniques. These two techniques will be compared with the proposed algorithms. The comparison will be based on compression ratios and image quality through RMS error and HVS.

1.6. Contributions to Knowledge

The expected contributions to knowledge from this thesis can be summarized as follows:

- A Matrix Minimization algorithm encoding/compressing each group of data items into a single value.
- A method to generate a data-dependent set of compression keys. The keys are used both for compression and decompression of data.
- A method to define the space domain of search (unique data) that is also data dependent. The set of unique data can be seen as a compression-

encryption key. Without the set of unique data, the file cannot be decompressed.

- An iterative method to recover the original data making use of the compression keys and the unique data set. Iteration is required as there is one equation for 3 variables in the demonstrated examples, which is a mathematically underdetermined problem.

Concerning the compression step, the thesis is focused on the Matrix Minimization algorithm. The algorithm is based on merging a group of data (normally three data items used in this thesis) to a single value. The algorithm is used to reduce the number of high-frequency components. In other words, as the high-frequency matrices' size are reduced there is a corresponding increase in the compression ratio.

The Matrix Minimization algorithm is based on keys (three different keys are suggested in this thesis). These keys are used for encoding/compression (See Section 1.7.2), and will be use later in the decompression algorithm. Furthermore, during encoding by the Matrix Minimization algorithm, the space search is created to determine the minimum and maximum boundaries of the coded data. Our proposed decompression algorithm uses this space search for fast decoding (See Figure 1.15)

Concerning the decompression step, the thesis is focused on iterative decoding algorithmsthat are applied to recover the original matrices. These are normally matrices of high-frequency coefficients but the method is valid for any matrix or vector that has been coded by the Matrix Minimization algorithm. Because the decompression algorithms rely on iterative search, there are various optimization techniques that can be used. Some are explored in this thesis, such as the Limited Sequential Search algorithm (LSS-Algorithm), which is used to recover the original data as illustrated in section 1.7.4.

1.7. General Research Methodology

This section describes the general methodological approach and main ideas to be developed into a PhD dissertation in image and data compression with specific examples. Many different algorithms were developed and are demonstrated in Chapters 2—8 with the general methodology described here. The method depends on the use of discrete transformations to generate two types of matrices namely a low and a high frequency matrix. Subsequently, the Matrix Minimization algorithm codes the high frequency matrix reducing it to 1/3 of its original size. This reduced matrix is then subject to arithmetic coding. Also, the Limited-Sequential-Search algorithm is illustrated as a search method for decoding high-frequency matrices. Figure 1.12 shows a dataflow of the proposed compression methods.

1.7.1. The Use of Digital Signal Transformations

The starting point in this research is the use of one or more types of transformations (i.e. DCT/DWT/DFT). The main objective from the digital transformation is to decompose the data into low and high frequency matrices. It is important to note that

the digital transformation can also be applied to the previously obtained low-frequency matrix, this will increase the number of high-frequencies that can be subsequently eliminated. This process will be investigated by using two different types of digital transformations. As discussed above, a DWT divides an image into four sub-bands, three of them high-frequency sub-bands labelled "LH", "HL" and "HH", and the last one is the low-frequency sub-band labelled "LL". In this stage the "LL" sub-band data are quantized. The value of the quantization could be selected by a user/programmer. The dimensions of "LL" and data size are not appropriate for compression without excessive deterioration in quality. For this reason, another transformation is used for reducing the size of "LL" and increase the number of high frequencies that can then be safely quantized, compressed or discarded **[42, 43].**

The second type of transformation is applied to the low-frequency sub-band by dividing it into blocks (i.e. block size 2x2 / 4x4 / 8x8…etc.), and then one of the digital transformations are applied (i.e. DWT/DCT/DFT) on each block. Each block consists of a DC component and high frequency coefficients. The DC component from each block is stored in a new matrix called DC-Matrix. The dimensions of this matrix are much smaller than "LL", while, high-frequency data are stored in a new matrix called AC-Matrix. After this process, the AC-Matrix is ready for coding by the Matrix Minimization algorithm. The DC-Matrix can be transformed again (i.e. apply DWT/DCT/DFT) to increase the number of high-frequency coefficients. Following the transformation, the DC-Matrix consist of: 1) high frequency coefficients, which can be coded by the Matrix Minimization algorithm, and 2) all DC-values, which are approximately similar, and thus the differences between DC-values are small and amenable to compression by arithmetic coding [42].

1.7.2. The Matrix Minimization algorithm Applied to High Frequency Matrices

Each high frequency matrix (i.e. LH, HL, HH, and AC-coefficients) are compressed by our Matrix Minimization algorithm. This method converts three items of data or more into a single value. This is achieved by using a random key. For example:

Original_data = [1 0 0 0 -2 0]

Key = [0.8147, 0.9058, 0.1270]

Which results in the minimized data illustrated as follows:

Data1=[1*0.8147 + 0*0.9058 + 0*0.1270] = 0.8147,

Data2=[0*0.8147 + (-2)*0.9058 + 0*0.1270]=-1.8116.

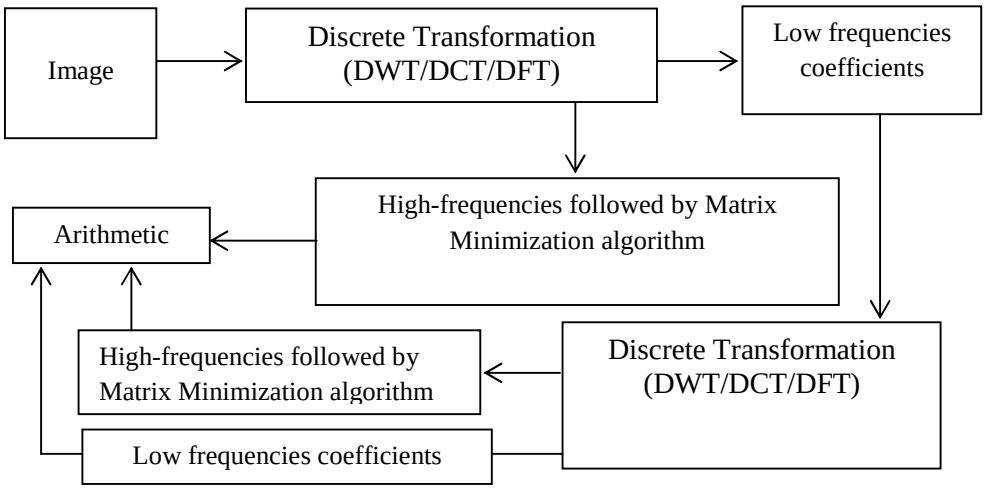


Figure 1.12: Proposed steps for 2D image compression

The final result will be **Minimized_array=[0.8147, -1.8116]**. The advantage of this method is that it eliminates some zeros and keeps the high-frequency data without the need for additional quantization. The used key in this method is user defined **[42]**.The key values are generated randomly (i.e. random numbers in the range =[0…1]), which are then multiplied by each row of matrix to produce the minimized array.

The Matrix Minimization algorithm is applied to each sub-band independently; this means each minimized sub-band is independently compressed by arithmetic coding. Figure 1.13 illustrates the Matrix Minimization algorithm applied to a matrix. Likewise, for larger reductions in data size, it is possible to compress the resulting minimized array by the Matrix Minimization algorithm. However, care must be taken if this further compression is used, as the side effect is less reliable probability or unique data**[42, 43]**.

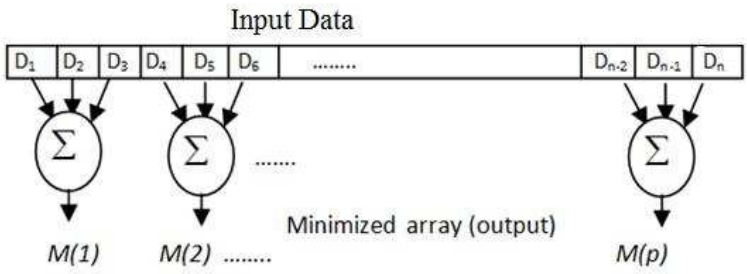


Figure 1.13:The original array size *n* is minimized to another array *M*.

Before applying the Matrix Minimization algorithm, our compression algorithm computes the probability of the data for each high frequency matrix. These probabilities are called *Limited-Data or Unique data*, which is used later in the

decompression stage. The Limited-Data are stored as a header in the compressed file and are not subject to compression. Figure 1.14 illustrates the probability of data in a matrix **[42]**.

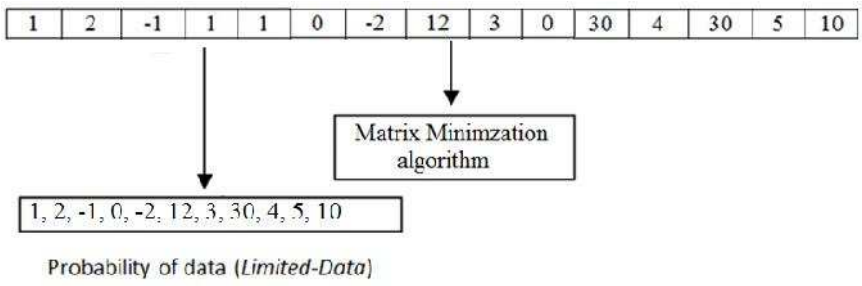


Figure 1.14: The Limited-Data for anarray size 15 is illustrated as a list of unique data.

1.7.3. Coding Stage

After compression of high frequency coefficients by the Matrix Minimization algorithm, the final compressed array contains a lot of zeros and nonzero-data. In this case all nonzero-data will be stored in the new array followed by Arithmetic coding. On the other hand, the number of zeros counted and stored in another new array. For example: minimized-array=[1.7 0 00 -1.7 0 0 9.2 0 000], nonzero=[1.7, -1.7, 9.2], zeroscount=[**2**,3,2,4]. The first number "**2**" at "zeroscount" refers to the position of first zero, while the other numbers in array refer to the number of zeros between data, which are then compressed by Arithmetic coding [42].

The main reason for using Arithmetic coding rather than Huffman coding is because Huffman coding uses a binary tree to generate a stream of bits, and this needsa huge amount of memory. For this reason, Huffman coding is useful for small matrices, for example, in JPEG Huffman coding is implemented on each 8x8 block, this means each block is compressed independently. On the other hand, arithmetic coding generates just one table for an image. The table needs less memory space than a binary tree.

1.7.4. Decompression Algorithm

The decompression algorithm reverses the compression steps. This proposal introduces new algorithms for searching methods for decompressing the reduced matrices. These algorithms will be tested on different types of 2D and 3D images. The main parameters to judge the success of the searching methods are the efficiency and accuracy of decompression. The following steps illustrate the proposed novel methods:

A) The DC-Matrix together with nonzero and zerocount arrays (as defined above) are decompressed by using Arithmetic Decoding. Moreover, the nonzero-array is merged with the zerocount-array producing the minimized-array.

B) As described above the minimized-array is obtained from the Matrix Minimization algorithm. The Decompression algorithm uses a new method called Limited-Sequential-Search Algorithm (LSS-Algorithm), the idea being to restore the original data from the Limited-Data. This process depends on the random key. Other algorithms than the LSS will be explored in subsequent chapters of this thesis. Initially, the LSS-Algorithm assigns three pointers (P1,P2 and P3) where each pointer is increment by one at each iteration loop. This process looks like a digital clock [sec, min, hour]. For example: assume we have the following information about the compressed data:

Limited_Data=[3 -1 0 1],Key=[0.8147, 0.9058, 0.1270].

The compressed data was: **Minimized_array=[0.8147, -1.8116]**. The decompression by the LSS-Algorithm is illustrated in Table-1.

Table-1: TheLSS-Algorithm iterative search

P1	P2	P3	Computation with key to find "**0.8147**"
1 → [3]	1 → [3]	1 → [3]	Result= 5.5425
2 → [-1]	1 → [3]	1 → [3]	Result=2.2837
3 → [0]	1 → [3]	1 → [3]	Result=3.0984
4 → [1]	1 → [3]	1 → [3]	Result=4.0697
1 → [3]	2 → [-1]	1 → [3]	Result=1.9193
2 → [-1]	2 → [-1]	1 → [3]	Result=-1.3395
3 → [0]	2 → [-1]	1 → [3]	Result=-0.5248
4 → [1]	2 → [-1]	1 → [3]	Result=0.2899
And so on…			

C) P1, P2 and P3 refers to locations in Limited-Data, which represents the search space. In the above Table-1, P1=1, P2=1 and P3=1 mean the first value: Limited-Data(1)=3. If "Result" in the above table is matched with Minimized_array(1)=[0.8147], this means P1,P2 and P3 found the data in Limited-Data (i.e. P1=4, P2=3 and P3=3). Similarly, the second data picked from minimized-array (2)=[-1.8116] for processing. In other words, the LSS-Algorithm works until finding all the original high-frequency data [42].

D) Finally, the decoded AC-Matrix is combined with the DC-Matrix to obtain the "LL" sub-band, and then applied inverse digital transformation (i.e. DCT/DWT/DFT) on each block (i.e. block size 2x2/4x4/8x8) followed by inverse second stage digital transformation for obtains 2D/3Ddecompressed image[42].

1.8. Overview of the Thesis

1. Chapter 1: general introduction about data and 2D image compression methods, then discusses the way that 3D mesh data represented and introduces the main ideas for data compression explored in subsequent chapters.

2. Chapter 2: introduces previous works for 3D mesh data compression.
3. Chapter 3: demonstrates compression of 2D structured light images used in 3D applications.These 2D images are transformed by using discrete DWT and DCT then the final transformed image coded by using the Matrix Minimization algorithm, which is used to reduce the number of high-frequency coefficients.
4. Chapter 4: the DWT is combined with the JPEG technique and used to reduce the number of transformation steps and increase the number of high-frequency coefficients.Additionally a new search algorithm is designed to speed up the decompression stage.
5. Chapter 5: the same Matrix Minimization algorithm used in the previous chapter extended by a novel search algorithm called FMS-Fast Matching Algorithm that is much faster than the previous decompression algorithm. Additionally, we enhanced the compression method by deleting zeros from each sub-band created by the DWT.
6. Chapter 6: in this chapter, we transformed a 2D image by DCT followed by DST applied over the entire image with Matrix Minimization algorithm.Results show the effectiveness of the method in compressing images up to 99% with high accuracy.
7. Chapter 7: we applied DCT (uni-transformation) on each block of an image followed by removing the number of zeros from the transformed matrix, then applied the Matrix Minimization algorithm for coding.Impressive results were obtained compared with JPEG and JPEG2000. The decompression algorithm was based on a new Concurrent-Binary-Search algorithm which is our fastest decoding algorithm to recover the original data.
8. Chapter 8: we applied the methods developed in the thesis to 3D data for coding both geometry and connectivity. Firstly, we convert the geometry vertices (X, Y, Z) to integer values by a shift to the left. Thenthe differences for all X' s, Y' s and Z' s respectively are computed. The MatrixMinimization method is then appliedto convert each set of X, Y and Z to a single value. Also, the same compression steps are applied to the connectivity (triangle faces). Results of the comparison with other 3D formats areimpressive, representing the state-of-the-art technology for 3D data compression of geometry and connectivity. The algorithm yielded compression ratios up to 97% with highly accurate 3D data reconstruction.
9. Chapter 9: presents the conclusion and future work.The chapter discusses advantages and disadvantages of the proposed methodology. A number of possible transformations to be use with our method have not been tried and these are discussed as further work. Additionally, we propose further work on applying the Matrix Minimization algorithm to more than three data items to a single value and approaches to optimizing decoding speeds.

Chapter 2
Overview of 3D Mesh Compression

2.1. Introduction

Graphics data are widelyused in several applicationssuch as video games, CAD/CAM design, virtual reality and visualization among others. Within 3D environments,triangular meshes represent the means to display and visualize 3D models. Normally, the geometry (vertices) and connectivity (triangulated faces) are used to create 3D polygonalmeshes and other properties such as illumination are used in 3D design. Connectivity describes the geometry relationships with data attributes such as colour, normal and texture coordinates (2D image coordinates).3D triangle faces handlesvertices and attributes data in the same way. Therefore, we concentrate on the geometry and connectivity compression methods in this chapter.

To obtain a high definition 3D model, normally a large amount of data are required. The dataareacquired from3D softwaremapping real world points to 3D model representations (for example, a software that converts a series of 2D images to a single 3D model) or a 3D scanner system that converts a single image to a 3D object. 3D models require large space (memory/hard disk) and bandwidth fortransmission as 1D-array. Thus, the expected level ofrealism from 3D representations calls for storage space, parallel-CPUs, efficient computing algorithms, and high speed bandwidths. Network bandwidth representsa bottleneck to the use and transmission of 3D data. Thus, it is essentialthat methods be develop to compress 3D data at higher ratios than currently available. Research on this topichas received attention and there has beenprogress in this direction over the last two decades.

3D mesh compression it has been stored in several standard formats. VRML is one of standard formats used for transmitting 3D models through the Internet [75]. Initially, 3D mesh data were represented as ASCII without compressing it into VRML format. Taubin[76] with his colleagues developed the topological surgery algorithm, a compression method for VRML for efficient transmission. MPEG (Moving Picture Experts Group) developed by multimedia standard ISO/IEC included encoded algorithm for 3D mesh data, which is based on the topological surgery algorithm, implementing a single-rate coder for manifold triangle faces [65]. Subsequently, MPEG incorporated progressive 3D mesh compression for non-manifold meshes.

In this chapter, we review different types of 3D mesh compression methods focusing on vertices and triangular faces compression. Many surveyed papers pointed out to this interesting subject. Rossignac [77] summarized a schema for

vertex and triangle faces data compression.Taubin[54] reviewed progressive3D mesh compression using topological surgery. Shikhare [78] classified 3D mesh compression schemesbut this type of schema did not work with progressive compression. Gotsman et al. [79] introduced a tutorial about 3D mesh compression and 3D geometry compression. However, it is focused on single-rate region-growing schemes. Alliez and Gotsman[45] reviewed single-rate and progressive compression of 3D meshes. Their proposed algorithm is classified asa high-level algorithm, however, it focuses on static 3D polygon compression.

2.2. Basic concepts of 3D mesh compression

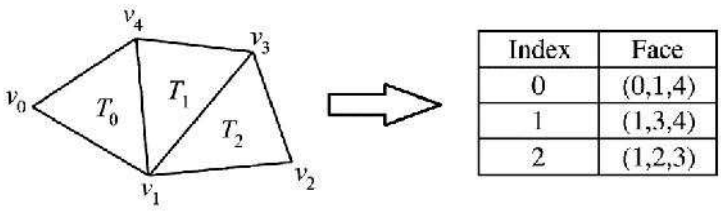
3D mesh compression algorithms encode each of connectivity data and geometry data individually.Earlierresearchconcentrate on connectivity compression with geometry coding determined from theconnectivity coding. However, geometry data required more bits than triangle faces, and several methods were developed to compress geometry data.

2.2.1. Triangle faces (Connectivity) Compression

In this Section, we will describe previous work on connectivity compression methods; these can be classified into different categories:

- **Index Triangle Face Compression**

The triangular mesh in the VRML format [75] represented with an index set, which is consisting of: array locations (vertices locations) and triangle faces location by its vertices. Figure 2.1 shows indexed face representation. To index each vertex, it is required at least $\log_2 v$ bits approximately. For this reason, each triangle face (connectivity information) needs $\log_2 v$ bits. This method provides triangular mesh representation. In other words, in this method no compression is involved. In this approach, each vertex is indexed multiple times. The repeated vertex thus, degrades compression performance. To solve this problem, we should reduce the number of repeated vertices reference.



Index	Face
0	(0,1,4)
1	(1,3,4)
2	(1,2,3)

Figure 2.1: Triangle face indexed representation

- **Triangle Strips Compression**

The 3D mesh can be divided into long triangle strips, and then encode each strip. The main advantage ofthis method is to reduce the amount of data transmitted

between the CPU and video graphic adapter (VGA). Although this method requires less memory space and transmission bandwidth than the previous one (indexed Triangle Face method), it still does not represent a very efficient compression method.Figure2.2 shows a triangle strip, where each vertex is combined with the previous two vertices in a vertex sequence to form a new triangle [79].However, many vertices are repeated in the generalized triangle strip of the mesh, which is a waste of storage. To addressthis type of problem, many schemes weredeveloped.

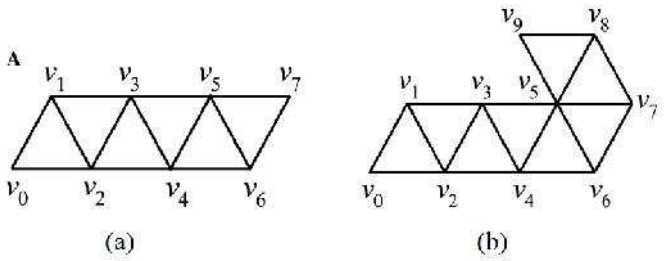


Figure 2.2: (a) The triangle strip, (b) the generalized triangle strip[79]

The concept of triangle strip mesh was introduced by Deering [79]. Itis formed by combining a vertex with groups of strips. Deering used FIFO (Fist-In-First-Out) buffer to save indices of 16 visited vertices.The vertices are saved in the buffer required fewerbits than a global vertices index.Each vertex is used once by the FIFO index, and Taubin and Rossignac[67] showed the generalized triangle mesh requires approximately 11bpv to encode connectivity data.

Chow [64]proposed a mesh compression method for real-time rendering as showed in Figure 2.3. The methods start with finding the boundary edges, next step searching for triangle fans around a vertex fallen on two successive boundary edges. The triangles in this strip are marked as “discovered triangles”. Undiscovered boundary triangles are similarly formed from new sets of boundary edges. The vertices in the vertex buffer can be reused for the next triangle strip, finally this process stop if all the triangles are visited in a mesh[80]. This method is efficient if the mesh is decomposed into long triangle strips, but it is achallenging computational mesh problem to obtain triangle strip decomposition [81]. Many heuristic methods are suggested for the triangular decomposition withaverage computational cost [82,83].

- **Triangle Spanning Tree Compression**

3D mesh can be represented as a graph, which means vertex nodes linked through edges look like triangle form. This conversion from 3D mesh to graph representation can be used in 3D mesh compression.Tutte[66] first proposed an algorithm to enumerate triangulation, a technique that compressed approximately 3.25

bpv.Thetechnique proved the feasibility of mesh connectivity coding[47].Turan [84]showed thata3D mesh can be compressed with a fixed number of bits using Spanning Tree for both geometry and connectivity as shown in Figure 2.4.

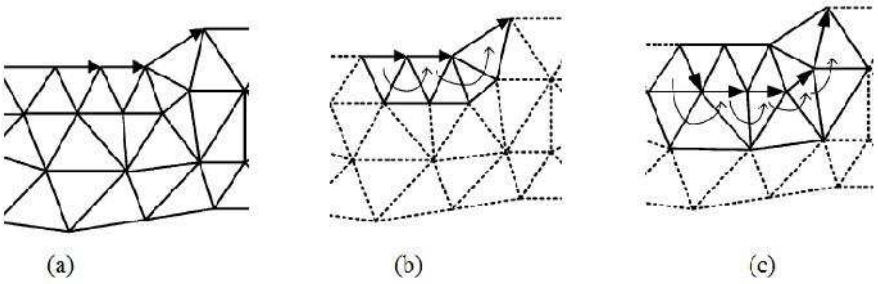


Figure 2.3: (a) Finding boundary edges, (b) and (c) shows trianglesat first strip and second strip. The arrows show selected boundary edges, while bots lines show the triangles associated with each inner boundary vertex[64].

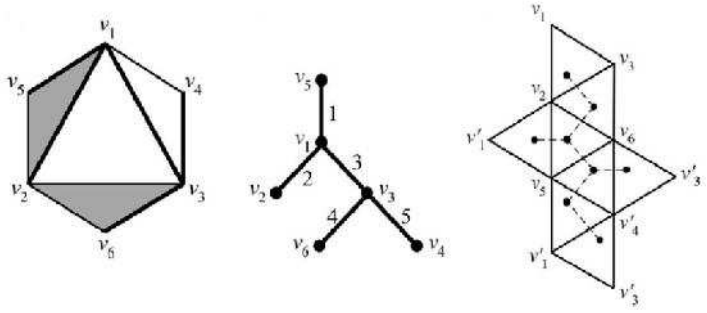


Figure2.4: (A) 3D mesh, (left) its vertex spanning tree, (right) the cut and flattened mesh with its triangle spanning tree shown by dashed lines[47].

The encoding theory based on the Taubin and Rossignac[85] approach (topological surgery algorithm)together with the impact of graph encoding can be felt on connectivity mesh compression technique, compressing about 2.5bpv to 6bpv (as we know,connectivity data is twice as large as geometry data).The Hand-and-Glove encoding algorithm from Diaz-Gutierrez *etal*. [68]compresses a mesh by using two types of spanning tree (the Hand and Glove trees). These two trees are applied in order on triangles strips loop traversing the entire mesh. The trees are encoded with 2bpv and an additional bit per triangle (allow to re-buildthe triangle strip), this means the total number of bits needed to compress a vertexis 4bits.Li and Kuo[86] suggested an algorithm to compress connectivity of a triangle with dual graph. Each node in this graph refers to three edges. Breadth-First Traversal algorithm was applied to dual graph produced a binary data for edges, if the edge was already visited or not.Triangle spanning tree generates such a tree by using breadth first traversal algorithm, during compression step some faces probably are visited while others not. This probably makes one or more closed border edges. The advantage of the spanning tree simplicity makes the algorithms appropriate to represent mesh data [87].

The Cut-Border machine [88] technique is used to extend the border by adding a new vertex to triangle iteratively. In case the border is separated or joined together, the scheme is able to compress manifold triangle mesh about 4bpv.The method isonly applied to regular meshes.

- **Edge-Breaker Compression**

The Edge-Breaker technique suggested by Rossignac[90, 91]generates a symbol for each triangle, this guarantee 4bpv. While, in real-world (in practice) a mesh is coded between 1.8 - 2.4bpv. The idea behind the algorithm is to encode a mesh by iteratively nibbling its faces, and each time a new face is traversed. Figure 2.5 shows the configuration for five patches [89 ,90, 91].

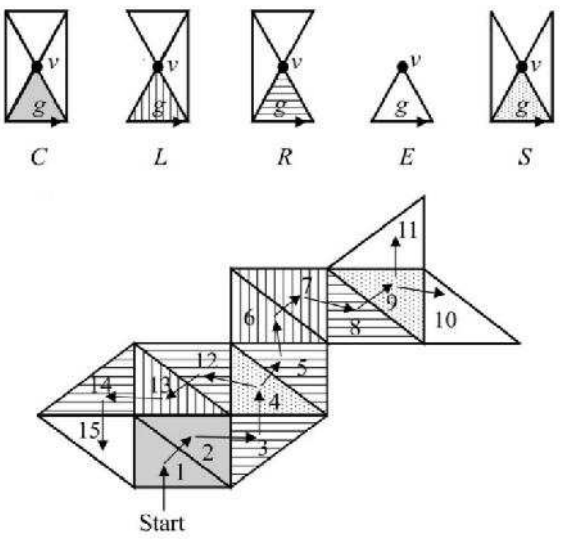


Figure2.5:(Top) Five encodes C, L, R, E, and S, while the gate g is the input to the triangle, and (Bottom) shows traversal triangles to encode part of mesh [44].

In Figure2.5 (bottom) shows the encoding process, by using the edge-breaker algorithm. The triangles are filled with op-code matching with op-code in Figure2.5(top), in this case the encoded mesh is "CCRSRLLRSEERLRE"

The edge-breaker method can encode data of manifold meshes with multiple boundary and the worst-case compression is approximately 4 bpv. However,the method is not suitable for streaming applications, it takes $O(v^2)$(i.e. execution time) of decompression time. Additionally, for regular meshes and non-regular meshes the same bitrate is required [87].

The original Edge-Breaker techniquewas optimized to encode 3D mesh with maximum regularity, the worst case for this method for a large regular mesh is

1.62bpv [92]. Coors and Rossignac[94]added connectivity prediction based on mesh geometry[93]. The Edge-breaker configuration used forpredicting the distance between position of parallelogram and the positions of active gate vertices, had an average connectivity compression ratio approximately less than 0.7bpv. Gumhold in 2005[88] proposed an enhanced Markov model coding converging to arithmetic coding. While Ying in 2010 [74]design an algorithm to select edge gate vertex toreduce the number of "s" symbols. Additionally, it creates a table depending on thenumber of traversed faces connected to gate vertices; this depends on a connectivity prediction method.

Kinget*et al*. [90] introduced an algorithm of generalized edge-breaker to compress quad triangle mesh connectivity. This algorithm based on splitting each quadrangle into two triangles that are on the same traversal sequence, this leads to efficient compression,because some combinations of edge-breaker cannot encode quadrangles [91].

Lee in 2002[95]proposed angle-analyzer scheme to encode the connectivity for manifold triangular at 1.5 bpv. The next face chosen to ensure the border between regions of the mesh the most convex as possible, Lee et al. [95]in 2005 proposed acompression ratio that can be reduced by using arithmetic coding.

- **Valence Encoding**

In valence encoding a manifold triangle mesh contains half the number of vertices than triangles. For this reason, an algorithm focusing on inserting a new vertex into the mesh and generating a symbol for the vertex, produces less symbols than a triangle traversal algorithm. So, this method leads to better connectivity compression performance [44].The valence approach is driven from Touma and Gotsman in 1998[46]. The algorithm detects the edge boundary formed by an initial triangle and expands the boundary by adding adjacent vertices iteratively. Therefore, the generated list of vertex valences can be efficiently compressed at 2.3bpv. This algorithm is still one of most efficient connectivity compression methods.

The Face Fixer by Isenburg and Snoeyink[50] compresses connectivity of manifold mesh with face degree by using face traversal. The algorithm generates symbols for each edge and experimental results showed compression ratios between 1.7bpv and 2.9bpv.Thealgorithm is more efficient than Kronrod and Gotsman[96],however, their encoder algorithm is easier to implement.

Isenburg and khodakovsky in 2002[97,98] worked independently onvalence approaches toencode connectivity of manifold meshes. Their work is based on Touma and Gotsman approach [46]. Khodakovsky*et. al*.[98]demonstratedschemes for entropy matches through Tutte's entropy for planar graphs. For both methods,experimental

results show compression ratio between 0.8bpv and 2.6bpv, improving over the face fixer [50].

Table 2.1 summarizes the state-of-the-art technique for 3D mesh compression. The bitrate of various connectivity coding methods reviewed was obtained theoretically throughworst-case analysis.

Table 2.1:3D mesh compression techniques

Algorithm	Proposed by	bitrate in worst-case (bpv)
Deering	Deering in 1995	11
Topological surgery	Taubin and Rossignac in 1998	6
Cut border machine	Gumhold and StraBer 1998	4.4
Valence coder	Touma and Gotsman 1998	2.3
Edge-Breaker	Rossignac in 1999 and Gumhold in 2000	4
Proved optimality for Valence coder	Alliez and Desbrun 2001	2.1
Valence manifold	Khodakovsky et al. 2002 and Isenburg 2002	2.6

2.2.2. Geometry Compression

Geometry compression is concerned with compressing vertex coordinates (x, y,z). Normally the compression of geometry begins byquantizingthe coordinates of vertices (i.e. geometry encoded in lossy mode). Subsequently, encode vertex through acoding algorithm[44, 87].

- **Quantization**

Vertices in computer memory are often represented by 32-bit floating point number for each coordinate (x, y, z). Such accuracy is not needed in some applications; for this reason aquantization process isused to reduce the amountof data without adversely affecting its quality. However, for some high-accurate 3D data the degradation appears or it is noticeable on the 3D surface [99].One of the mostused quantizationtechniquesin 3D mesh is called *Scalar Quantization,* whichconsists oftransforming the floating-point number vertices into integer vertices. The quantization is based on the maximum integer that can be coded with the number of quantization bits.Generally,thegeometry compression methods that go along with well-known connectivity compression schemes proposed by Deering, Rossignac, Gotsman*et. al*.[46, 63,67] use uniform scalar quantization.The number of bits range between 8-bit to 16-bit, thus connectivity in 3D mesh softly conflicts with geometry. Bajaj *et. at*. [61] and Lee *et. at*. [60]proposed to encode a vertex with three angles, by using an angle-analyser encoder. The computation is based on two internal angles and one dihedral angle by applying different quantization to these local angles. Lee and

Park [62]proposed to locate vertices in 4 different range sizes; however, there are some vertices located in bigger ranges. Thus, the vertex is encoded according to the type of the range [100].

- **Prediction**

Prediction means encoding vertex positions predictively. The prediction uses correlation between adjacent coordinates, and the most important aspect of prediction is to reduce the amount of geometry, and then encoding with one of entropy coding methods (Huffman or Arithmetic coding).There are different prediction methods that have been proposed in theliterature: delta prediction [78], linear prediction [64, 65], and parallelogramprediction [101]. Overall, the predictions schema is considered as linear prediction and their coefficients are chosen carefully.

Delta Prediction means the differences between coordinates are usually small. For this reason, adjacent vertices have slightly different coordinates. Deering and Chowused delta coordinates rather than the original coordinates followed by encoding with variable length codes. In each work quantized coordinates range between was 10—16bpv and 13—18bpv respectively[62,78].

*Linear Prediction*suggested by Taubin and Rossignac[65] predicatesvertex position from a group of positions of previous vertices in the vertex spanning tree. The number of previous vertices are selected from the root to the current vertex in the spanning tree. The estimated geometry bitrate is reported by Tuma and Gotsmanat around 13bpv[46].

Parallelogram prediction suggested by Touma and Gotsman[46] encodes new vertex "r" within a triangle vertices "u and v" as show in in Figure 2.6, where the triangle "u, v, w" is already encoded. Parallelogram predicted a new vertex "r" position by using the form r^p_1=v+u-w. The rule for this prediction is thatthe four verticesmust beexactlycoplanar. The parallelogram improved the prediction accuracy by using the angle between two adjacent trianglestoestimate vertex position r^p_2 as shown in Figure 2.6 This type of prediction achieved 9bpv[46, 101].

Additionally, Isenburg and Alliez[102]used parallelogram prediction for geometry compression of polygon meshes. The position of amissing vertex of a polygon is predicted with weights computed from different degrees and computed from different known vertices.

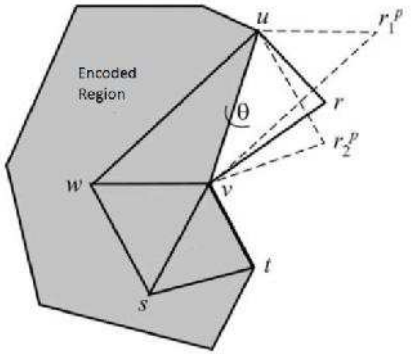


Figure 2.6: Parallelogram prediction[102]

- **Geometry-driven compression**

In 2002 Kronrod and Gotsman[103]designed a geometry-driven compression scheme. The approach covers the traversal tree that contains all vertices. The reasons for using the tree is to minimize parallelogram prediction errors between vertex positions. Geometry compression yielded ratios 50% more efficient than connectivity compression. This approach is used for CAD models, which often have non-smooth geometry.

Shikhare with his colleague in 2001 [104] developed a geometry-driven compression method that tries to find repeated geometry patterns in the 3D mesh. There are many options to recognize patterns, these can be in components or regions inside components. The approach is useful for large 3D CAD models or digital heritage models.Cai*et al.*in 2009 [105]added scaling and transformation for finding repeated patterns, an approach that achievedslightly better performance.

Asconnectivity and geometry are compressed at same time in the above methods, their data are interleaved in the compressed stream. Lewinere*tet al*. [106]in 2006 proposed an alternative geometry-driven algorithm. This algorithm compresses geometry independently from the connectivity encoding. Then the surface is reconstructed iteratively and each new triangle connected to the border of the mesh is built by selecting a new vertex among the candidates. This algorithm can compress any type of triangulatedmesh. Additionally, Chaine*et al.*[107] in 2009 proposed a mesh connectivity compression scheme that assumes that the geometry is already decoded. While the connectivity between vertices are generated by the Delaunay triangulation based on point set, in this method the mesh is encoded at low cost.

- **Compressing floating-point vertices**

The methods described above are based on lossy mesh compression (i.e. the compression algorithm quantized the vertices). But some application required encodingthe exact floating point coordinates. Isenburg[108]in 2005 investigatedthat the floating point can be broken to exponent and mantissa. Meanwhile, the predicted error between decompressed and original mesh are negligible, this method is able to compress geometry about 35bpv.

Siddeq and Rodrigues [109]in 2016suggested compressing 3D point cloud by converting floating-point vertices to integer then encoding by the Matrix Minimization algorithm. This algorithm is designed to encode-encrypt a vertex to a single integer value, then the result can be subject to arithmetic coding. Compression ratios about 3.7bpv were obtained, while the connectivity (triangle meshes) was compressed independently by computing the differences between vertex references (i.e. in OBJ and COLLADA formats the vertex references definetriangle faces). The compression ratio for the connectivity in theapproach is about 0.69bit for each triangle in the best case. However,the worst case reaches to 20 bitsfor each triangle. In same year, they developed connectivity compression by applying the Matrix Minimizationalgorithmon the stream of triangle faces to reduce from 20bit/triangle to 13bits /triangle inthe worst case.

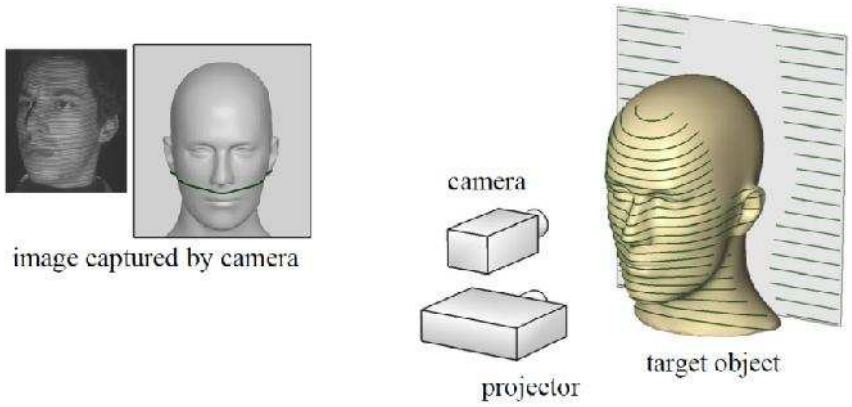
2.3. 3D mesh Compression based on 2D images

In 2012 Siew and Rahman [110] pointed out that XML is a candidate to represents the meaning of a 3D data structure, which is distributed in many applications. On theotherhand, XML generates large files that, concatenated with geometry information,renders the scheme impracticable. The issue lies on finding a way to compress the semantic of data that represents calibration of geometry and connectivity of a 3D surface.

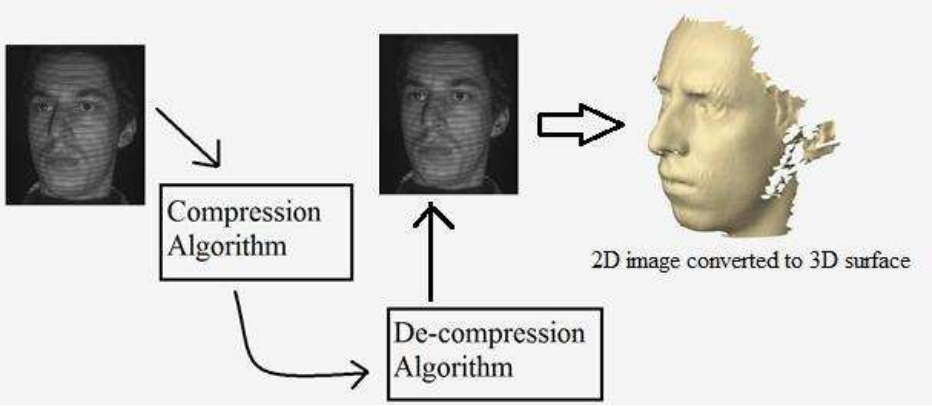
Rodrigues and Robinson[73] earlier in 2008 reconstructed 3D data from BMP images using structured light techniques.A projector and camera setting was developedtogether with image processing functions to project, capture, and process and image of an object with a projected pattern of stripes. The image processing functions detected the projected patterns and reconstruct the object in 3D as shown in Figure 2.7[73, 111,112, 113]. The captured BMP images with structured light information can be saved for later use in many applications such as 3D face recognition and sharing of 3Ddata.

Rodrigues and Osman[114] in 2010 proposed a polynomial interpolation method tocompress 3D data files.The source data model uses vertices which are the standard data in all 3D geometry models, such as 3D Wavefront OBJ, VRML and COLLADA data files [55, 74, 115]. Only geometry data was compressed with compression ratiosover97%. The method did not compress connectivity as this could be directly

inferred from the projected patterns, so the solution was specific to a scanning method, which limited its application. The result was only applicable to surface patches and, if structured pattern information were not available, then Delaunay triangulation could be applied to recover the surface from the point cloud[116].



(a) (Right) The projector spot a pattern of light on an object, (Left) A camera records the reflected patterns with the object.



(b) The 3D surface generated from stripes available in the 2D image, compression and decompression algorithm works just on 2D images.

Figure 2.7: (a) and (b) shows the high quality 2D images are used to reconstruct 3D surfaces model [114].

2.4. Summary

There are many 3D mesh compression techniques as reviewed in this Chapter, and each of these techniques may be suitable to a particular application only such as AutoCAD or a specific structured light scanner.Some compression techniques are slow duringcompression, but fast duringdecompression; this is normally desired in 3D graphics for games. The main disadvantage of some 3D compression techniques is that they fail tocompress some 3D objects. While other techniques can onlycompress the geometry but not the connectivity of the vertices.

For these reasons,the work in this thesis concerning 3D mesh compression is divided into two parts:

1) Compress 2D imagesthat contain structured lightpatterns (such as stripes or dots);this kind of image contains information enabling3D reconstruction of the object. After compression, reconstruct the object and compare with other compression methods namely JPEG and JPEG2000.
2) Compress 3D object data files which contain information on geometry and connectivity – i.e. vertices and triangle faces. It is proposed to compress vertices and triangle faces independently.

A comparative analysis will be performed between 2D image compression followed by 3D reconstruction and direct 3D object compression. The analysis is based on speed of compression and decompression, compression ratios, RMSE (image quality), and perceptual assessment of the image and 3D model quality.

Recently, some applications were developed and are available to convert a series of images to a 3D object, one of these applications is 123D AutoDesk. The big challenge is to compress these 3D objects by our proposed algorithms and compare the results with other 3D mesh compression techniques. Additionally, our proposed algorithm will compress a stream of 2D images thatare used by the 123D Autodesk software. A comparative analysis of results will be performed, i.e., of2D images and 3D object which are described in the following Chapters.

Chapter 3

DWT-DCT based Compression with Sequential Search Decompression

3.1. Introduction

Despite recentfast progress in storage density and increased processor performance,there is still demand for algorithms that work on images and these algorithms continue to exceed the abilities of available technologies [117]. Recent growth of multimedia based applications have encouraged the need for very efficient ways to compress signals and images, the main reasons beingcompression for storage and digital communication [118].Compressing an image is significantly different from compressing a stream of data (i.e. lossless stream of data). It is certainly the case that general purpose compression algorithms can be used to compress images, but the result is non-optimal. This is because images have statistical properties that can be used by encoders specifically designed for them [118, 119]. Also, some of the details in the image can be omitted for saving bandwidth or storage space. Lossless compression mean thata, when decompressed, the exact replica of the original image is obtained [120]. On the other hand, lossy image compression does not need be decoded exactly as the original. An approximation of the original image is acceptable for most purposes (e.g. large images with high-resolution or video compression), as much as the differences between the original and the compressed image is deemed adequate [121].

In most cases, the pixels in an image are correlated (i.e. a pixel has mutual connection with another neighbour pixel), therefore we use this idea for reducing the number of redundant information. Furthermore, amajor task is to find out uncorrelated pixels (i.e. the other pixels where there is no connection between them). The main task inimage compression is to reduce the amount of redundant data to an acceptable level without degrading the quality of the image [122].

We can divide image compression into: 1) redundancy reduction and 2) insignificance reduction. The redundancy reduction aims to remove duplication from the signal source image, while the insignificance reduction delete parts of the image that is not noticed by the receiver (i.e. cannot discover the distortion by Human Visual System (HVS), and this depends on image details and image size). Consequently, repeated pixels are eliminated according to statistical properties and the HVS will not detect the difference between original and reproduced images [117].

The standard JPEG for compression of still imagesuses theDiscrete Cosine Transform (DCT), which represents an image as a superposition of cosine functions with different discrete frequencies. The DCT can be regarded as a discrete time version of

the Fourier Cosine series. It is a close relative of Discrete Fourier Transform (DFT), a technique for converting a signal into elementary frequency components. Thus, DCT can be computed with a Fast Fourier Transform (FFT) an algorithm of complexity $O(n \log 2\ n)$[123]. More recently, the wavelet transform has emerged as a cutting-edge technology within the field of image analysis. The wavelet transform has a wide variety of different applications in computer graphics including radiosity, multi-resolution painting, curve design, mesh optimization, volume visualization, image searching and one of the first applications in computer graphics, image compression [22]. The Discrete Wavelet Transform (DWT) provides adaptive spatial frequency resolution (better spatial resolution at high frequencies and better frequency resolution at low frequencies) that is well matched to the properties of a Human Visual System (HVS)[118, 124].

Here a further requirement is introduced concerning the compression of 3D data. We demonstrated that while geometry and connectivity of a 3D mesh can be tackled by several techniques such as high degree polynomial interpolation [114] or partial differential equations [125], the issue of efficient compression of 2D images both for 3D reconstruction and texture mapping for structured light 3D applications has not yet been addressed. Moreover, in many applications, it is necessary to transmit 3D models over the Internet to share CAD/-CAM models with e-commerce customers, to update content for entertainment applications, or to support collaborative design, analysis, and display of engineering, medical, and scientific datasets. Bandwidth imposes hard limits on the amount of data transmission and, together with storage costs, limit the complexity of the 3D models that can be transmitted over the Internet and other networked environments [125].

In this Chapter,it isinvestigated that surface patches can be compressed as a 2D image together with 3D calibration parameters(i.e. lossy compression), transmitted over a network and remotely reconstructed (geometry, connectivity and texture map) at the receiving end with the same resolution as the original data.The widespread integration of 3D models in different fields motivates the need to be able to store, index, classify, and retrieve 3D objects automatically and efficiently. In the following sections, we describe a novel algorithm that can robustly achieve the aims of efficient compression and accurate 3D reconstruction.

3.2. The Proposed Compression Algorithm

The lossyimage compression method proposed here is based on DWT and DCT which are used to increase the number of high-frequency sub-bands with few significant data. The first stage DWT is applied to decompose an image into four sub-bands (LL,LH,HL and HH). The LL is approximately similar to the original image (i.e.LL represntes average value of the 2D images – For this reason all the values in this subband are postive), while the other sub-bands represent image details and contain few data with huge number of zeros (i.e. the main reason most of values are zeros,

this is because neighbours in a part of original image are similar to each other). Furthermore, the LL sub-band is divided into non-overlapped block of matrices (4 x 4 pixels). These blocks are transformed by DCT producing a DC value (i.e. this value is positive coefficients with is represents average value of 4 x 4) and a set of AC coefficients data (i.e. reprinted the approximation coefficients, these values are vary from top-left to button-right, which is normally be around zero). Additionally, the DC-values of each block are saved in a matrix called DC-Matrix whichrepresents a new low frequency sub-band. Similarly, the AC coefficients are collected in a new matrix called AC-Matrix which is equivalent to the high frequency sub-band. Finally, the AC-Matrix with other high-frequency sub-bands are coded by the Matrix Minimization algorithm, while the DC-Matrix transformed again by DWT to increasethe number of high frequency sub-bands. Also, this chapter describes the Limited-Sequential Search Algorithm (LSS-Algorithm) used to decode the DC-Matrix and AC-Matrix to reconstructapproximately the originalimage [42]. Figure 3.1 shows the main steps of the proposed compression method in a flowchart style.

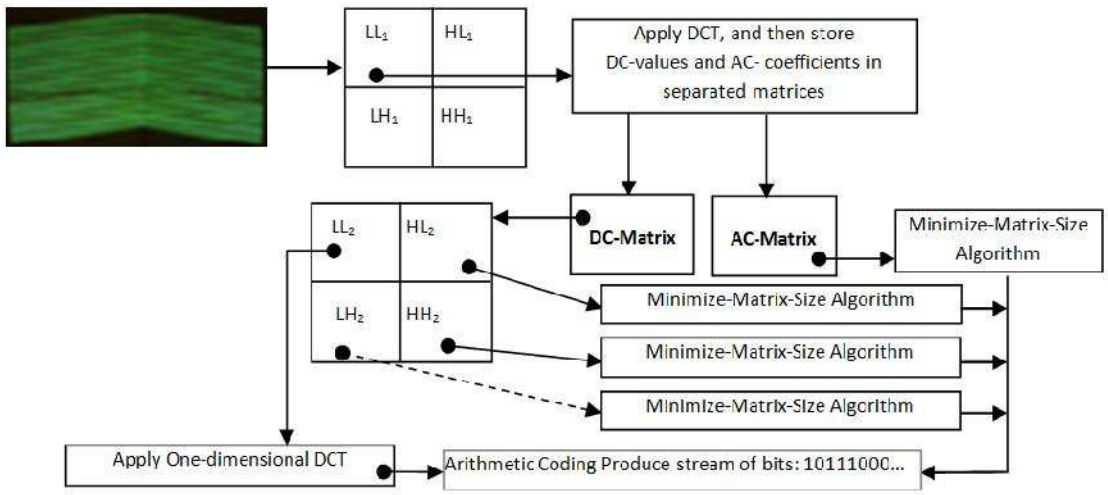


Figure 3.1: proposed image compression method flowchart

3.2.1. The Discrete Wavelet Transform (DWT)

The discrete wavelet transform (DWT) used to transform an image into four sub-bands from low to high frequency coefficients. The DWT is used for multi resolution data analysis (i.e. it can be used in progressive transmission and image zooming without requesting additional storage) [42,126]. Additionally, the inverse DWT has thesame complexity,which means the forward and inverse transformation are symmetric; this feature of DWT is suited for fast image compression and decompression. Furthermore, it has very good energy compaction capabilities, robustness under transmission and high compression ratios [127].

The DWT is applied toimage compression by using a Daubechies (db3) filter that decomposesthe original image. The output of the filter bank is down-sampled sub-bands. The inverse DWT decodes by up-sampling and recomposes the original image [37]. The DWT divide an image into four sub-bands (LL, LH, HL and HH). LL is represented down-sampled original image (Low-frequency coefficient), which is represented important information about original image. While other sub-bands represent vertical, horizontal and diagonal details of an image (high-frequency coefficients). In case these details are small,their values can be set to zero without change in image details.For this reason,the high-frequency sub-bands can be compressed to a few bits/bytes [128]. In this Chapter,we will use DWT twice toincrease the number of high-frequency coefficients. This process will reduce image size to a fraction of the original image size enabling high compression ratios [42].

3.2.2. The Discrete Cosine Transform (DCT)

After DWT applied on an image, the LL sub-band is transformed again by applying DCT on each 4x4 sub-matrix (block) as shown in Figure 3.2.

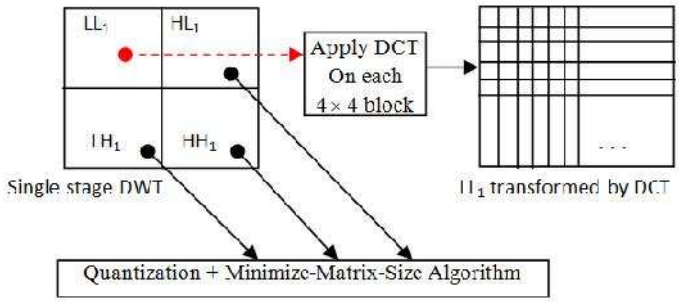


Figure 3.2: LL_1 sub-band transformed by DCT for each 4x4 block set.

The data available in the LL sub-band are still correlated. For this reason, DCT is applied to transform LL to de-correlated coefficients. First, the LL sub-band is divided into small blocks (4 x 4). Second, DCT is implemented on each block, the top corner of the block of positive value represents peak of energy and the other coefficients are called de-correlated (high-frequency domain) until thebottom right of the block. The coefficients with small values can be discarded without affecting image quality. The transformed block by DCT can compress more efficiently than a correlated block. The following equations illustrated DCT and Inverse DCT functions for two-dimensional matrices [42]:

$$C(u,v) = a(u)a(v)\sum_{x=0}^{n-1}\sum_{y=0}^{n-1} f(x,y)c\left[\frac{(2x+1)u}{2n}\right]\left[\frac{(2y+1)v}{2n}\right] \quad (3.1)$$

$$whe \quad \backslash a(u) = \sqrt{\frac{1}{N}}, f \quad u = 0$$

$$a(u) = \sqrt{\frac{2}{N}}, f \quad u \neq 0$$

$$f(x,y) = \sum_{u-0}^{n-1} \sum_{v-0}^{n-1} a(u)a(v)C(u,v)c \quad \left[\frac{(2x+1)u}{2n}\right]\left[\frac{(2y+1)v}{2n}\right] \tag{3.2}$$

Where *f(x,y), C(u,v)* are represented original and transformed image respectively, and the main differences between DWT and DCT is that DCT is applied on small block or rectangular regions of images, this is because DCT is progressively complex to calculate on lager blocks, for this reason we used small blocks of 4x4 pixels. Whereas, the DWT can be applied on an image as a large block, it works more efficiently yielding better compression ratio [42, 122].

After DCT is applied on each 4x4 block of LL_1, these blocks are ready for quantization. The quantization processesdivides each value in the block by a factor Q which removes trivial coefficients keeping fewer data from the block. The factor Q can be computed as follows [42]:

$$L = Q \qquad \times \max(L_{\ 1}) \tag{3.3}$$

$$Q(i,j) = \begin{cases} 10, & i,j = 1 \\ L + i + j, & i,j > 1 \end{cases} \tag{3.4}$$

Note\i, j=1,2,3,4

The maximum value in LL_1 sub-band is used to compute the parameter L in Eq. 3.4 (i.e. used as quantization value "L"). Additionally, the quality value is used to control the maximum value, for example if maximum value is 51 and Quality=0.01, this means all the values in a sub-band will be divided by 0.51. If this value is increased, it leads to the removal of large number of coefficients (i.e. forced data to zero), and this leads to lower image quality. The DC-Matrix is created by the DC values from each block (4x4) of LL_1, while other coefficients (4x4-1) are stored as column in a new matrix called AC-Matrix. HL_1,LH_1 and HH_1 sub-bands are quantized by Eq. 3.3 followed by coding by Matrix Minimization algorithm [42].

DWT is used again to transform the DC-Matrix yielding new sub-bands: LL_2, LH_2, HL_2 and HH_2. The size of LL_2 small can be encoded into a few Bytes/Kbytes (according to image size). While other high-frequencies sub-bands LH_2, HL_2 and HH_2 are quantized by dividing the coefficient matrices by "2", for normalization and increase the number of zeros to be easier encoded by the Matrix Minimization algorithm [42].

To compress the sub-band LL_2, first transform each 4 data (one-dimensional array size 4) of the LL_2 by using one-dimensional DCT (i.e. using the same Eq. 3.1 with

u=0 and v=0) and then truncate each value to integer value, in this stage we should not use scalar quantization.The difference between two adjacent values in the same column is computed and stored in the same position matrix LL_2 this difference computation is shown in Figure 3.3. This process assumes that neighbour's coefficients are correlated. The correlated values are generally similar so the differences will be small and more data are repeated. This process will increase the compression ratio. Eq. (3.5) represents the difference for each column in LL_2[42]. Third, apply an encoding method to convert the final transformed matrix into a stream of bits.

$$D(i) = D(i) - D(i+1) \tag{3.5}$$

Where\ i= 1,2,3,…, m-*1* and *m* is the column size of LL_2.

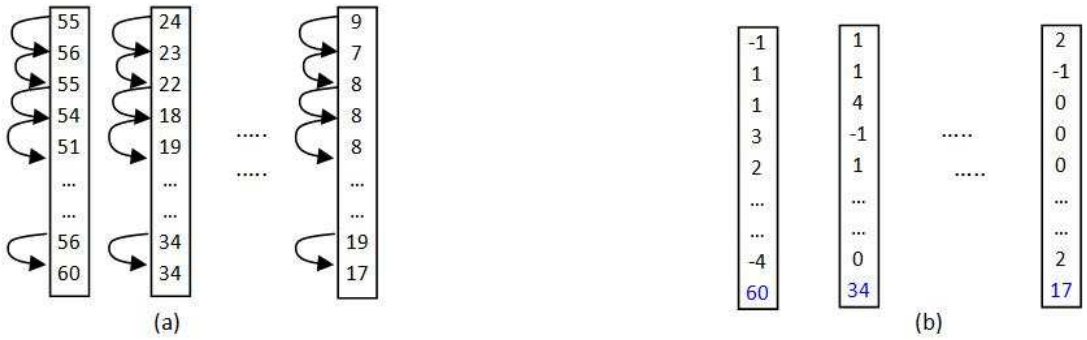


Figure 3.3: (a) A matrix before DBV, (b) Apply DVB between two neighbours in each column.

3.2.3. Compress Data by Matrix Minimization Algorithm

This algorithm is designed to reduce high-frequency coefficients. This algorithm is applied onthe AC-Matrix and other high-frequency sub-bands independently. The main idea for the algorithm is to convert three adjacent coefficients to one encoded value.The calculation depends on Random-Weight-Values and three adjacent coefficients and the results restored in a new encoded array.The following List 3.1 describes the steps of the algorithm [42]:

List 3.1 Matrix Minimization Algorithm

```
Let K=3                              %% take each three coefficients from a matrix
W=Generate-Ransom-Weights (K)        %% generate three random weights values according
                                     %% to the number of coefficients
Let p=1
Fori=1 to column size
        For  j=1 to row size
                Intermediate [p]=Matrix[i,j]   %% Scan row-by-row
                p++
        End
End

Let j=1; p=1
 While (j<row size*column size)
        Arr=Read_K_coefficients (Intermediate [j])
```

$$M(p) = \sum_{i=1}^{K} W(i) * Arr(i)$$

j=j+k

$p++$

End

In the above List 3.1 weight values are generated by using a random function (random number range ={0...1}). These weights are multiplied by Arr(i), Arr(i+1) and Arr(i+2) representing three coefficients within the high-frequency domain to produce a minimized array "M". Since the Matrix Minimization algorithm is applied to each sub-band independently, each sub-band has its own encoded minimized array. Figure 3.4 illustrates the Matrix Minimization algorithm implementation.

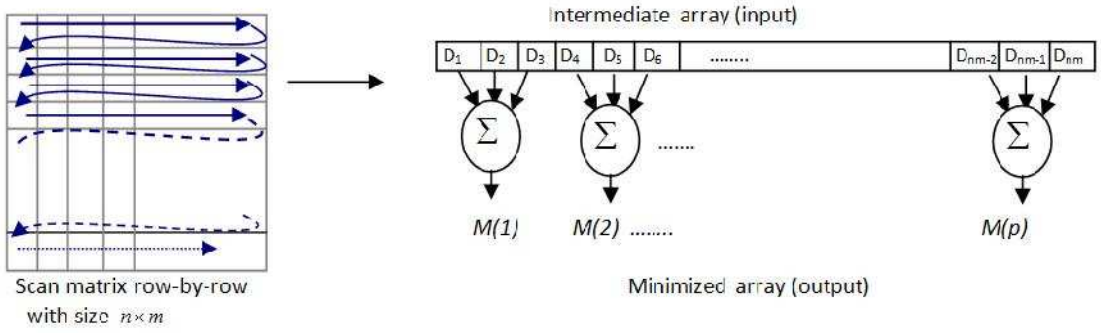


Figure 3.4: An *nxm* matrix is minimized into an array *M*.

Before applying the Matrix Minimization algorithm, we need to choose a valuebetween duplicate numbers of high-frequency domain; these valuesare called Limited-Data, this new stream of data is used at decompression stage. These data limit the search space ofdecompression [42]. Figure 3.5 illustrates the computation of the Limited-Data.

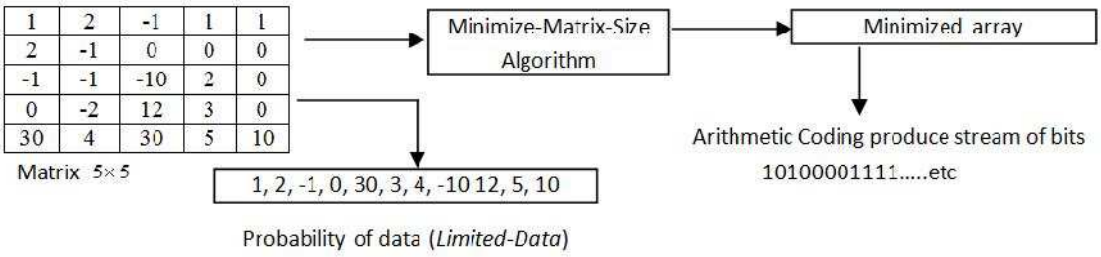


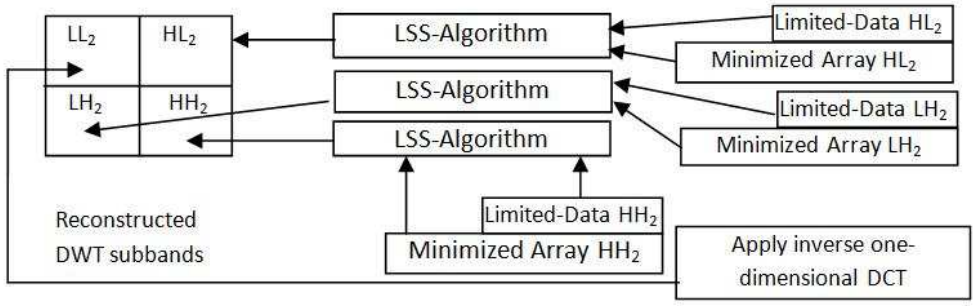
Figure 3.5: The Limited-Data for a 5x5 matrix is illustrated as a list of probabilities and the minimized array is subject to arithmetic coding.

Arithmetic Coding is the final step in the compression algorithm describedthis Chapter. This lossless coding algorithm applied to astream of data convertsthe stream to a singlefloating point value; this output in the range between zero and one. When

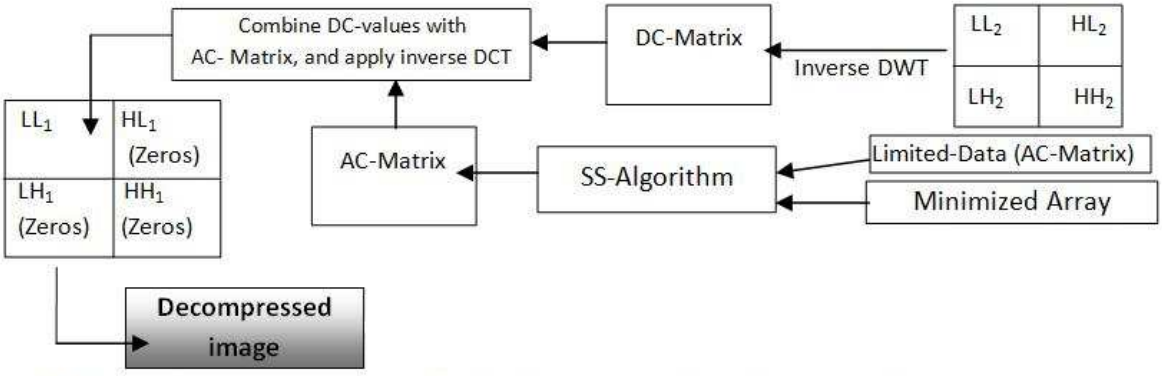
decoded, the exact original data are recovered. To summarize this lossless algorithm, we need to compute the probability of all the data (before compression), to assign a low and high values for each data (each coefficient).

3.3. The Decompression Algorithm

This section describes details thedecompression algorithm, which is the inverse of the compression algorithm. The first stage uses arithmetic decoding for decoding the minimized-array. Next, we have designed aLimited Sequential Search Algorithm (LSS-Algorithm) to decode each sub-band. The LSS-Algorithm used Limited-Data array with Random-Weights to reconstruct the original coefficients. If Limited-Data is missed or destroyed, the image cannot be recovered. Figure 3.6 shows the decompression method in a flowchart style [42].



(a) First Stage Decompression algorithm for reconstruct DWT subbands (LL_2, LH_2, HL_2 and HH_2)



(b) Second stage Decompression algorithm for reconstruct LL_1 (decompressed image)

Figure 3.6: A two-stage decompression algorithm is depicted in (a) and (b).

The main reason to design this decoding algorithm (LSS-Algorithm) is tosearchfor the original coefficients inside the Limited-Data array; this operation is done by using three pointers (S1, S2 and S3), which refer to positions in the Limited-Data array.These pointers work in sequence like a clock: where S1, S2 and S3 represent hour, minutes and seconds respectively. Initially these values are set to "1", (i.e. S1=S2=S3=1). To illustrate theLSS-Algorithm assume that we have the following 2x3 matrix:

30	1	0

19	1	1

First, the above matrix will be compressed by theMatrix Minimization algorithm to produce a minimized array M={3.65, 2.973} with Limited-Data={30,19,1,0}.Additionally, threerandom weight values or key={0.1, 0.65, and 0.423}. Second, the LSS-Algorithm is used to decode the minimized array returning the original 2x3 matrix. The decoding algorithm startsbypicking up the first data item from Limited-Data (S1=S2=S3=1) and then compute the estimated value by using the following equation [42]:

$$Est = W(1)Limited(S1) + W(2)Limited(S2) + W(3)Limited(S3) \quad (3.6)$$

Where "W" is Random-Weight-Values and "Limited" is the Limited-Data matrix. The LSS-Algorithm computes "Est" at each iteration and compares with M(i). The iteration means S3 will increment by 1 (i.e. it works like a second hand in a clock), after all the positions in Limited-Data, the S2 start work (like a minute hand in a clock) followed by S1 (like an hour hand in a clock). If M(i)=Est, this means the original coefficients data are in locations {S1, S2 and S3} according to Limited-Data. Otherwise, the decoding algorithm will continue searching to find the original coefficients. This process continues until the end of the minimize array M(i). The List 3.2 illustrates the LSS-Algorithm [42].

List 3.2*LSS-Algorithm*

```
Let Limited[1…m]          %% represents Limited Data
Let M [1…p]               %% represents minimized array with size p
Let K=3    %% number of estimated coefficients
For i=1 to p
        □1=1; S2□1; S3□1      %% initial location
  □     1□□□□□□□□□□□
        Est=W(1)*Limited[S1] + W(2)*Limited[S2]+W(3)*Limited[S3]

While ( (□□)   □  □□□□)  0)    %% check if Error =0 or not
        S3++                  %% increment pointer represents "Seconds"
        IF (S3>m) S2++; S3=1end□ %% check if S3 is over the limit, return back to "1", and increment S2
        IF (S2>m) S1++; S2=1end□
        IF (S1>m) S1=1;end□
        Est=W(1)*Limited[S1] + W(2)*Limited[S2]+W(3)*Limited[S3];%% compute Est after increments
        Iterations++          %% compute number of iterations
  End                         %% end of while
End
```

After the LSS-Algorithm has decompressed all high-frequencies matrices, the next step is to reverse the difference operation of Eq. 3.5 by addition as defined in equation 3.7on the decoded LL2 to recover the original coefficient values. This is applied to each column by taking the last value at position m, adding it to the position m-1, and then the total adds to the next position m-2 and so on. The following Figure 3.7 illustrates the addition decoder[42].

$$D(i-1) = D(i-1) + D(i) \tag{3.7}$$

Where \i= *m, (m-1), (m-2), (m*-3),…,2

Before the final steps, the inverse one-dimensional DCT is applied to each 4 coefficients for LL_2, followed by the composition of all sub-bands (LL_2, LH_2, HL_2 and HH_2) by applying the inverse DWT resulting in the decoded DC-Matrix.Finally, combine the DC-Matrix with AC-Matrix (same way they decomposed from LL1 see section 3.2) to generate LL_1, and then apply inverse DCT. The inverse DWT will recompose all decoded $LL_1$to recover the original 2D image [42].

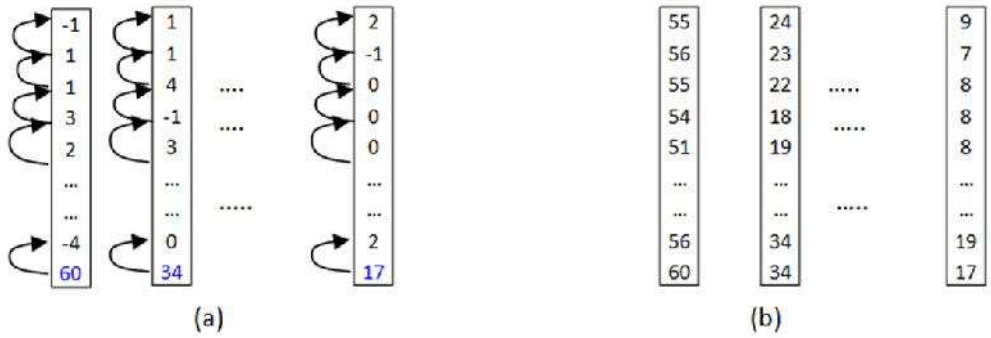


Figure 3.7: A Matrix before Apply ABV, (b) Apply ABV between two neighbours in each column

3.4. Experimental Results in 2D and 3D

The algorithms described in this Chapter were tested with three types of 2D images type RGB.Firstly, the RGB colour convert to YCbCr format (i.e. this format is useful by any compression method, which is converts true colour to another formula, all the image data will be available in single layer called “Y”. While other layers are contains less information capable to compress at higher compression without affects on image details) [131]. Second, the algorithmswere implemented in MATLAB running on a Laptop AMD quad-core: 2.4GHz with SDRAM: 6GBytes. Also, the decompressed 2D images showed by 3D visualization software running on the same laptop. Figure 3.8 shows 2D images used for testing our approach, and Table 3.1 shows the compressed size for each image.

The 3D visualization software read an image with structured light patterns on, and determines the position of the 3D vertices in space from each stripe in the image. The software was develop within the GMPR group [135]at Sheffield Hallam University. The patterns of stripes are projected on the surface on an object and captured by a high resolution camera. The GMPR 3D reconstruction software creates a 3D representation of the object in a few milliseconds. The main advantages of the developed 3D scannerare speed and accuracy [136].

(a) 2D BMP “Wall”, dimension 1280x1024 pixels, size=3.75 Mbytes

(b) 2D BMP “Girl”, dimension 1392x1040 pixels,size=1.38 Mbytes

(c) 2D BMP “Woman”, 1392x1040 pixels, size=1.38 Mbytes

Figure 3.8: (a) 2D colour MP image, (b-c) 2D grey scale images

Table 3.1: Compressed image sizes using high frequencies on first level DWT

Image name	**Original size**	**Compressed size**	**Compression Ratio**	**Quantization Values**	
				Low-frequency	**High-frequency**
Wall	3.75MB	74 KB	98%	0.02	0.02
Wall	3.75MB	47.6 KB	98.7%	0.04	0.04
Wall	3.75MB	33.7 KB	99.1%	0.08	0.08
Girl	1.38MB	78 KB	94.4%	0.02	0.02
Girl	1.38MB	48 KB	96.6%	0.04	0.04
Girl	1.38MB	29.1 KB	97.9%	0.08	0.08
Woman	1.38MB	62.1 KB	95.6%	0.02	0.02
Woman	1.38MB	38.1 KB	97.3%	0.04	0.04
Woman	1.38MB	24.5 KB	98.2%	0.08	0.08

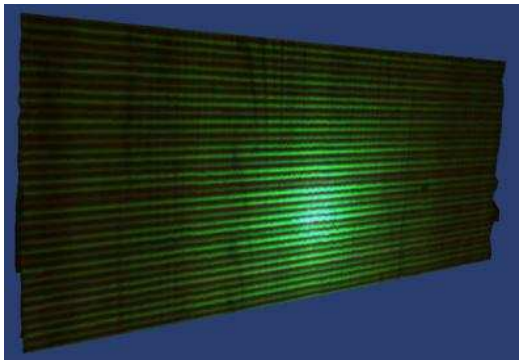
The quantization values: "0.02", "0.04" and "0.08" in above in Table 3.1 refer to image quality: high, median and low respectively (i.e. the Quality value in Eq. 3.3 responsible for keeping 2D image details: LH_1, HL_1 and HH_1 in DWT at first level). For example: if Quality=0.02, this refers almost all coefficient data remain, otherwise, if Quality value is greater than 0.02 this means partially the coefficient are set to zero in a sub-band. The LL_1 sub-band depends on the DCT coefficients. Additionally, Table 3.2 shows that high frequencies are ignored from the first level of DWT decomposition (i.e. all high-frequency coefficients are setto zero) [42].

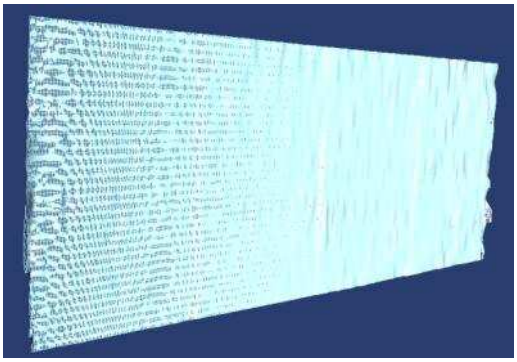
Table 3.2: Compressed image size without using high-frequencies in first level DWT

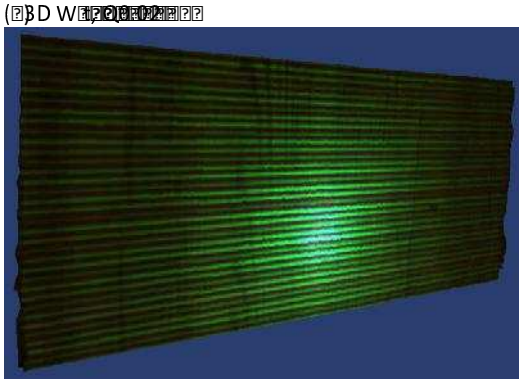
Image name	**Original size**	**Compressed size**	**Compression Ratio**	**Quantization Values**	
				Low-Frequency	**High-frequency**
Wall	3.75MB	62 KB	98.3%	0.02	ignored
Wall	3.75MB	45 KB	98.8%	0.04	ignored

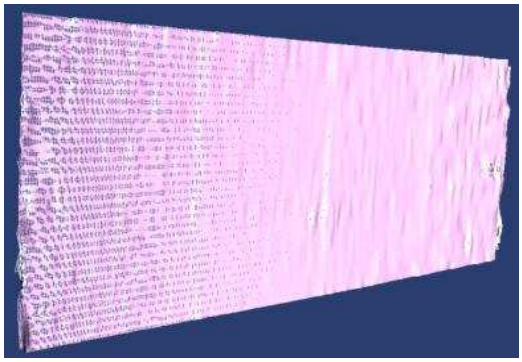
Wall	3.75MB	33.5 KB	99.1%	0.08	ignored
Girl	1.38MB	61.2 KB	95.6%	0.02	ignored
Girl	1.38MB	42.6 KB	96.9%	0.04	ignored
Girl	1.38MB	28.3 KB	97.9%	0.08	ignored
Woman	1.38MB	53.4 KB	96.2%	0.02	ignored
Woman	1.38MB	35.4 KB	97.4%	0.04	ignored
Woman	1.38MB	24.3 KB	98.2%	0.08	ignored

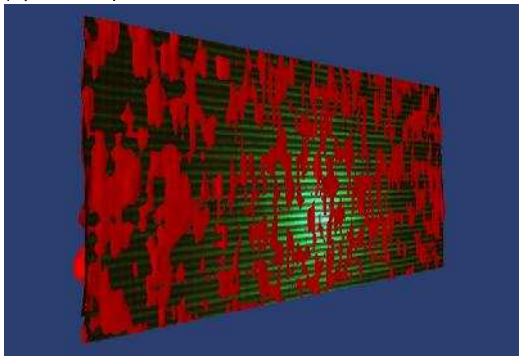
The proposed image compression algorithm are applied by using a range of Quality factors toeach image and the decompressed images are used by our GMPR software to reconstructthe 3D mesh, which is then compared with original 3D mesh model. Figures 3.9, 3.10 and 3.11 show the 3D reconstructed Wall, Girl and Woman respectively. Table 3.3 shows the 2D RMSE and 3D RMSE for each 2D decompressed image and 3D reconstructed surface. The Root-Mean Square Error (RMSE) is used to calculate 2D/3D image quality mathematically. RMSE is a very popular measure to compute the differences between decoded image and original image [42,129].

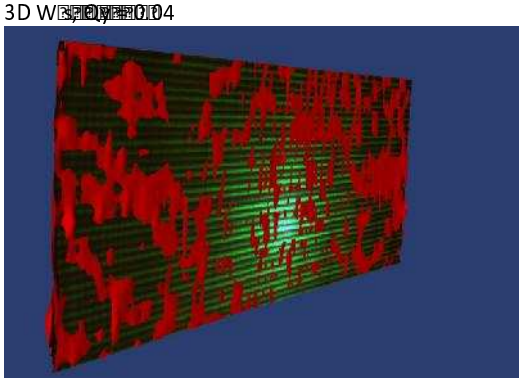




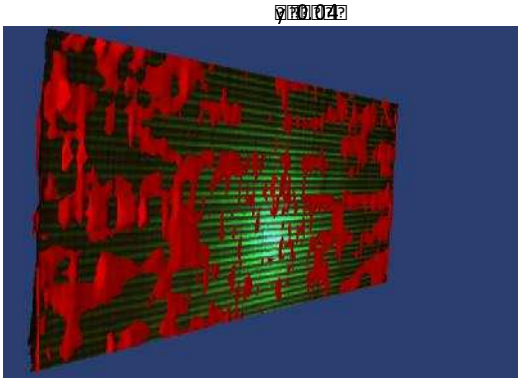






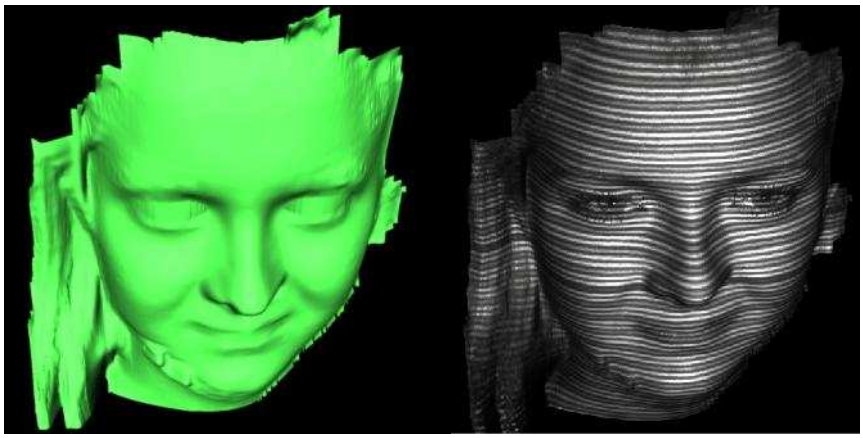


(?) 3D ????(????)?????????? ?? ????????? 3D W????????????)
(?????????2 ?????

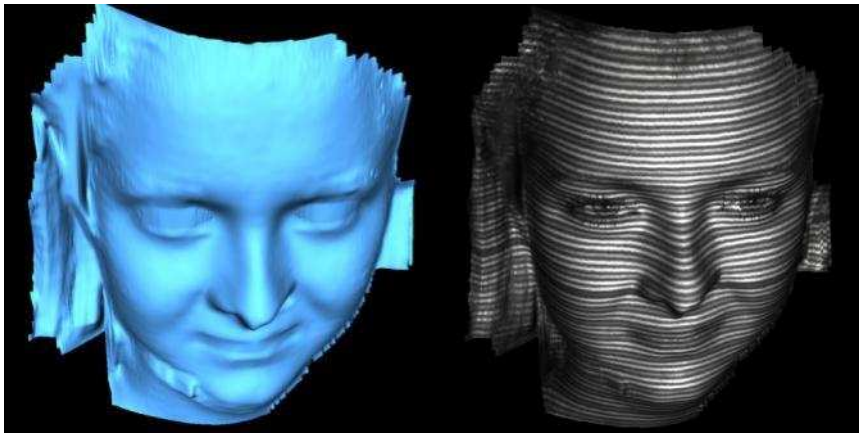


(?3 D W????????? (????????)

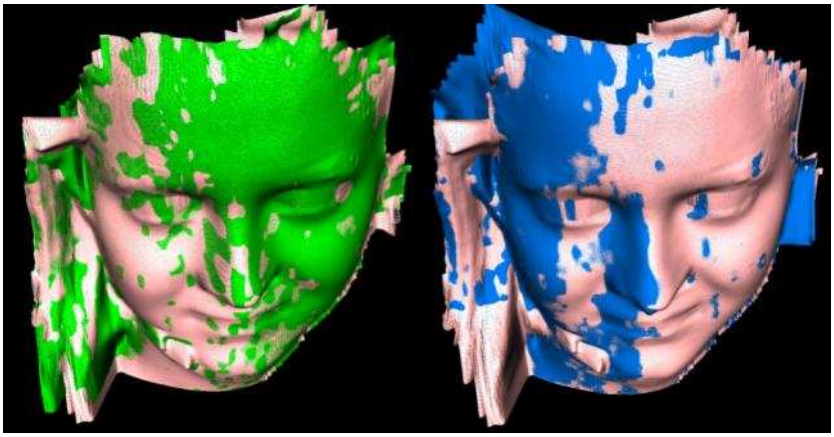
Figure 3.9: (a) and (b) 3D decompressed image of Wall with different quality values, (c), (d) and (e) Differences between original 3D Wall image and Decompressed 3D Wall image according to quality parameter. Red regions represent the 3D Wall decompressed image matched with the background original 3D Wall image in three cases, i.e., High, Median and Low quality parameters.



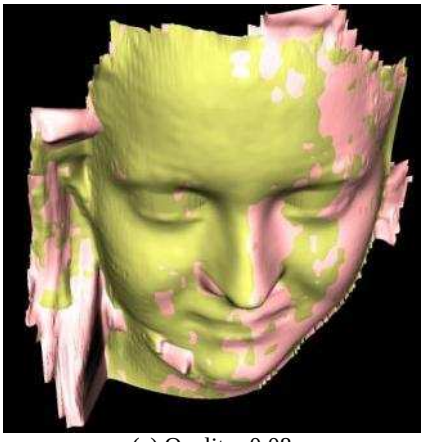
(a) 3D Girl image texture and shaded, Quality=0.02



(b) 3D Girl image shaded and texture, Quality=0.04

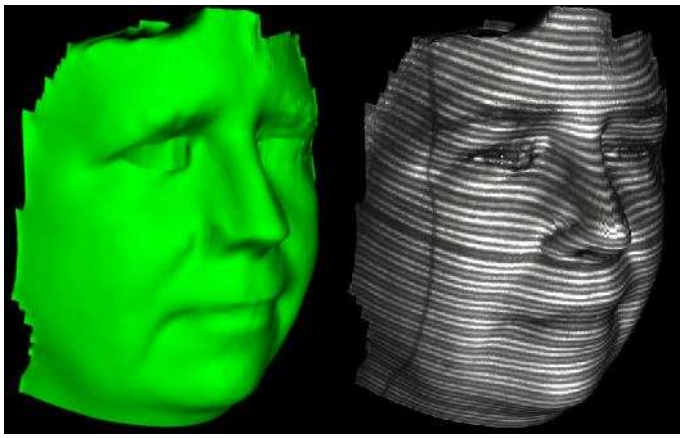


(c) Quality=0.02 (d) Quality=0.04

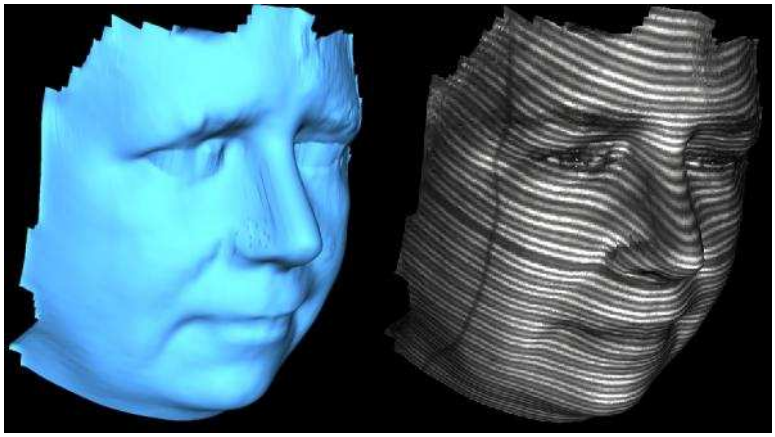


(e) Quality=0.08

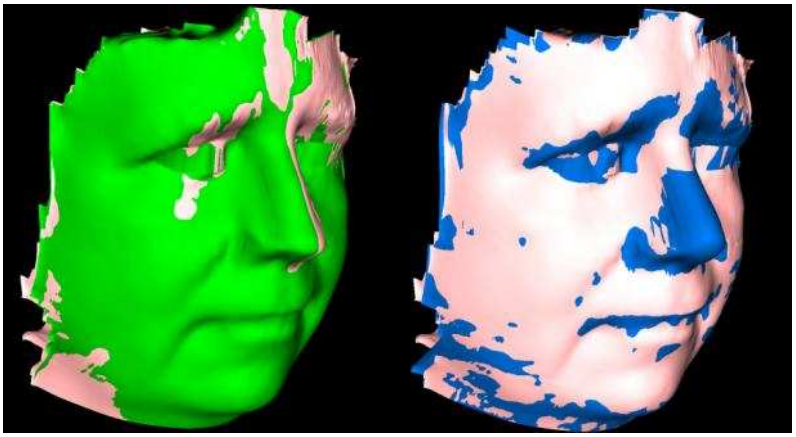
Figure 3.10: (a) and (b) 3D decompressed Girl image with different quality values, (c), (d) and (e) Differences between original 3D Girl image and Decompressed 3D Girl image according to quality parameters.The pink model represents the original background 3D image, while other colours represent the 3D decompressed image with various quality parameters.



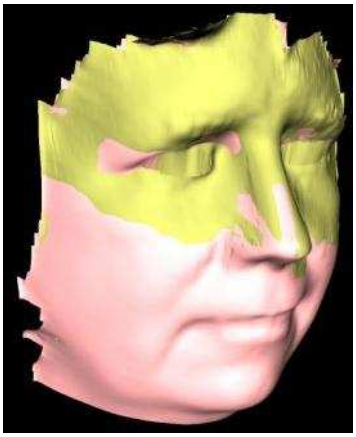
(a) 3D Woman image shaded and texture, Quality=0.02



(b) 3D Woman image shaded and texture, Quality=0.04



(c) Quality=0.02 (d) Quality=0.04



(e) Quality=0.08

Figure 3.11: (a) and (b) 3D decompressed Woman image with different quality values, (c), (d) and (e) Differences between original 3D Woman image and Decompressed 3D Woman image according to quality parameters. The pink model is the original 3D Woman model while blue, green, and golden models refer to high, median and low image quality respectively.

Table 3.3:PSNR and MSE between original and decompressed 2D images

Image name	RMSE	3D RMSE	Quantization Values	
			Low-frequency	High-frequency
Wall	2.49	2.09	0.02	0.02
Wall	2.82	3.95	0.04	0.04
Wall	3.25	4.72	0.08	0.08
Girl	3.09	3.78	0.02	0.02
Girl	4.08	3.94	0.04	0.04
Girl	5.25	3.66	0.08	0.08
Woman	2.88	3.37	0.02	0.02
Woman	3.53	3.09	0.04	0.04
Woman	4.35	2.61	0.08	0.08
Image name	**RMSE**	**3D RMSE**	**Low-frequency**	**High-frequency**
Wall	2.66	2.09	0.02	Ignored
Wall	2.86	3.95	0.04	Ignored
Wall	3.24	4.72	0.08	Ignored
Girl	4.39	3.41	0.02	Ignored
Girl	4.71	3.83	0.04	Ignored
Girl	5.34	3.74	0.08	Ignored
Woman	3.38	3.12	0.02	Ignored
Woman	3.73	3.07	0.04	Ignored
Woman	4.38	2.71	0.08	Ignored

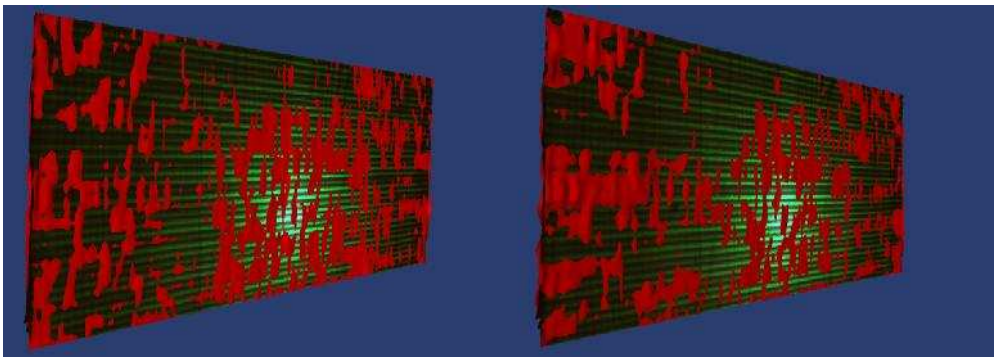
3.5. Comparison with JPEG2000 and JPEG Compression Techniques

The proposed compression algorithm iscompared with two techniques widely used in image, video compression and image transmission: JPEG and JPEG2000. As mentioned din Chapter 1 (See section 1.2.1) the JPEG technique is based on two dimensional DCT applied toan image, and previously the image is divided into 8x8 blocks. Additionally, each block is encoded separately[42, 127]. While JPEG2000 is based on DWT, which is applied to a partitioned image into non-overlapped blocks (i.e. block size variable specified by the use/programmer), then the transformed blocked addressed t coding algorithm for compression (See section 1.2.2) [42]. Most image compression applications allow the use/programmer to determine image quality by using specific parameters for balance between image quality and compression ratio [22]. The comparison between these two methods and our approach is based on Root-Mean-Square-Error (RMSE).

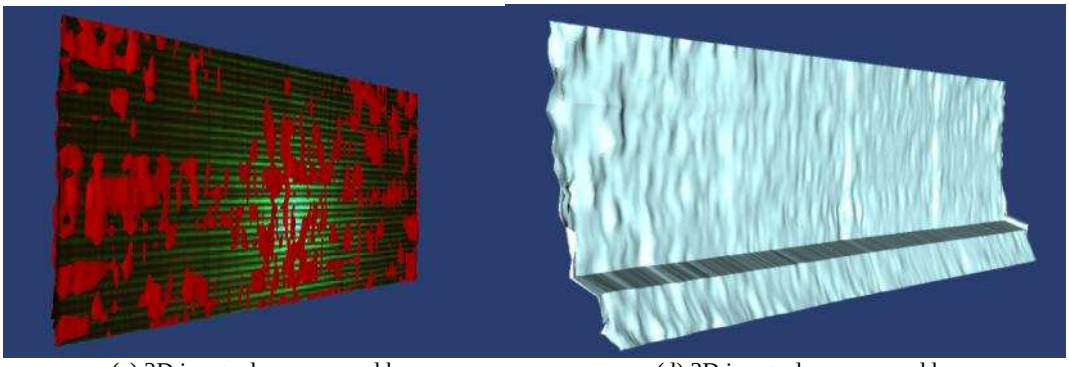
Table 3.4: "High", "Median" and "Low" represent image quality used by each method. Moreover, "FAIL" means that the method (JPEG or JPEG2000) cannot compress images when the quality value is "Low" as reached by our approach and unable to reconstruct the 3D model. Figures 3.12, 3.13 and 3.14 show the 3D reconstructed images by JPEG and JPEG2000.

Table 3.4: Comparison between the proposed algorithm and JPEG2000 and JPEG techniques

Image Name	Quality	Proposal Method		JPEG2000		JPEG		Compression Ratio
		RMSE	3D RMSE	RMSE	3D RMSE	RMSE	3D RMSE	
Wall	High	2.49	2.09	1.92	4.28	3.14	2.8	98%
	Median	2.82	3.95	2.14	5.01	3.87	4.5	98.7%
	Low	3.25	4.72	2.42	3.52	5.34	6.9	99.1%
Girl	High	3.09	3.78	2.14	3.94	3.28	3.94	94.4%
	Median	4.08	3.94	2.88	4.02	4.72	3.72	96.6%
	Low	5.25	3.66	4.1	1.51	**FAIL**	**FAIL**	**97.9%**
Woman	High	2.88	3.37	2.14	3.14	2.6	2.55	95.6%
	Median	3.53	3.09	2.7	3.2	4.58	2.75	97.3%
	Low	4.35	2.61	**FAIL**	**FAIL**	**FAIL**	**FAIL**	**98.2%**

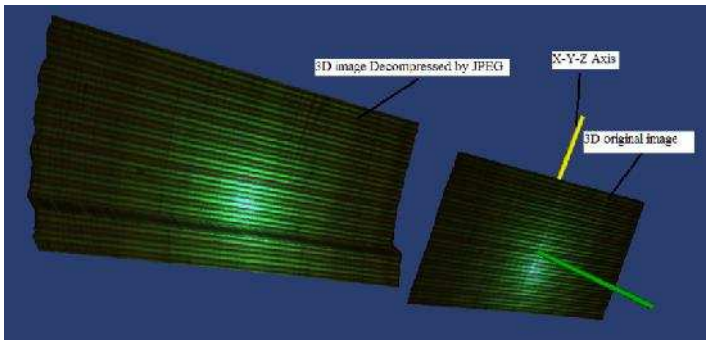


(a) Decompressed by JPEG2000 3D Flat image(b) Decompressed by JPEG2000 3D Flat image, 3D RMSE =4.283D RMSE=5.01



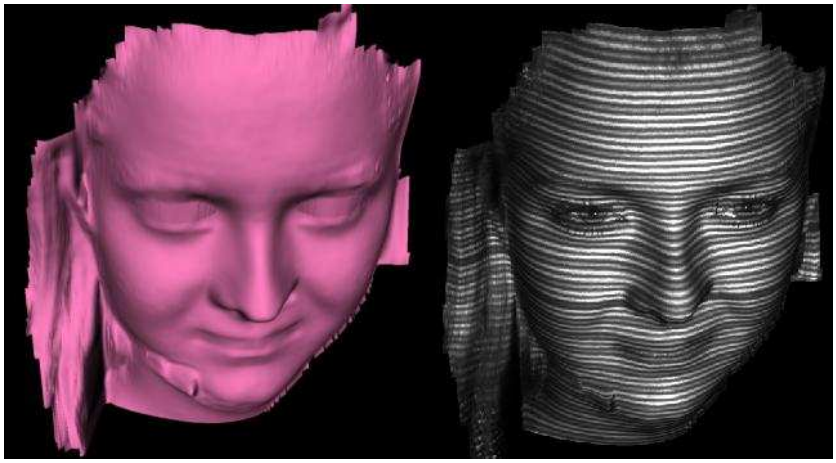
(c) 3D image decompressed by JPEG2000 3D RMSE=3.52

(d) 3D image decompressed by JPEG Quality=56% (degraded 3D)

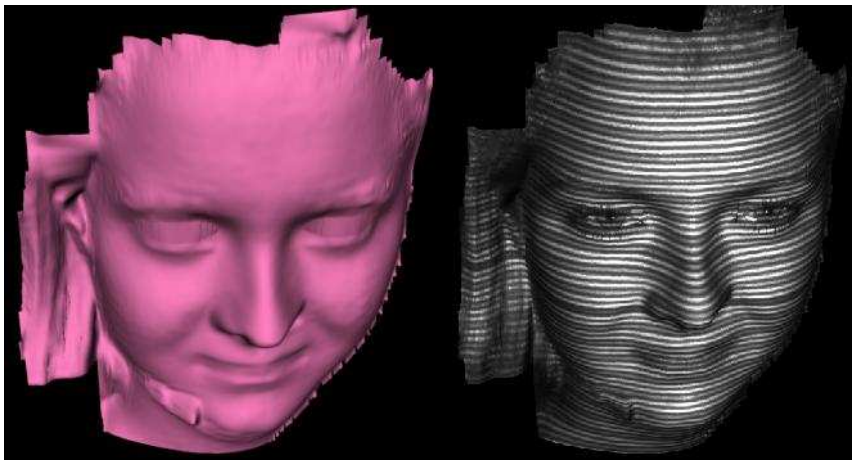


(e) Decompressed 3D Wall image by JPEG, Quality=26%, 3D RMSE =2.8 [Degraded 3D image]

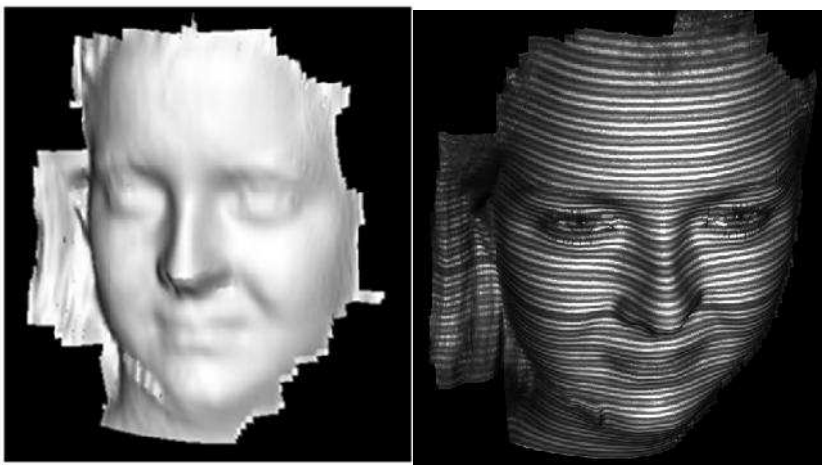
Figure 3.12: (a), (b) and (c) Decompressed 3D Wall image by JPEG2000, Decompressed image with quality=40% most of regions are matched with the original image,similarly with quality=26% and quality=10%approximately matched with the original image, (d,e) Decompressed 3D Flat image by JPEG (degraded) un-recognized with original image. Median quality 2D decompressed image by JPEG at quality=51%, quality=23% non-capable of generating 3D model.



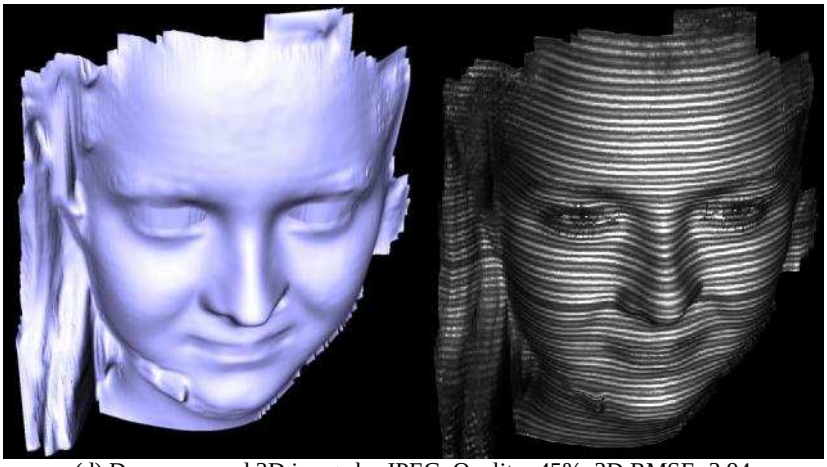
(a) Decompressed 3D image by JPEG2000, 3D RMSE=3.94



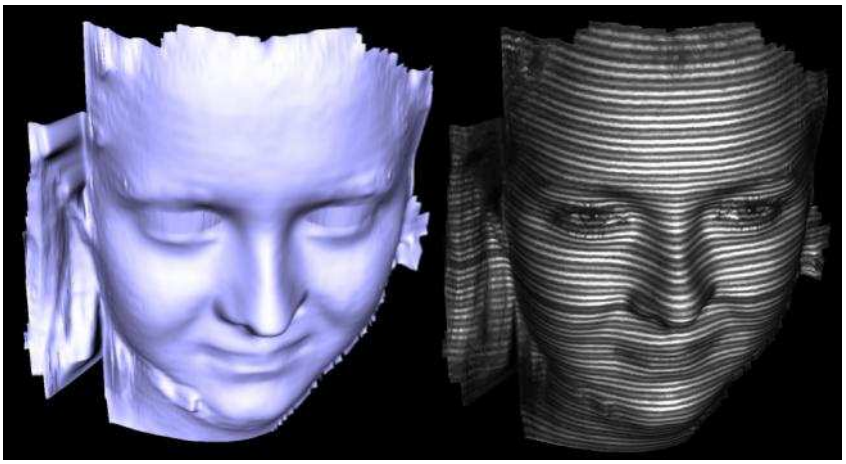
(b) Decompressed 3D image by JPEG2000, 3D RMSE=4.02



(c) Decompressed 3D image by JPEG2000, 3D RMSE=1.51

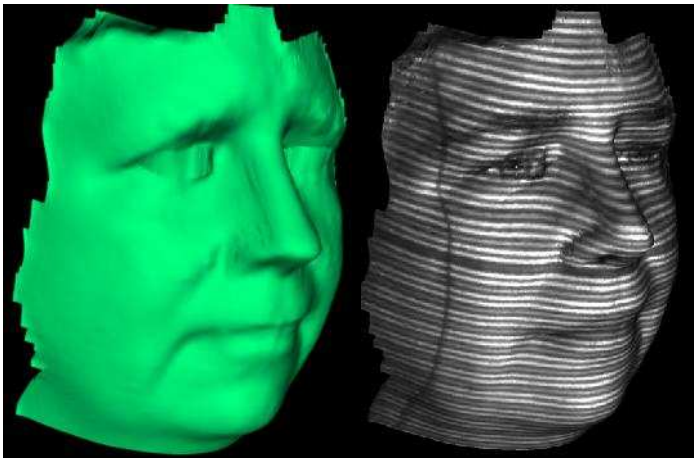


(d) Decompressed 3D image by JPEG, Quality=45%, 3D RMSE=3.94

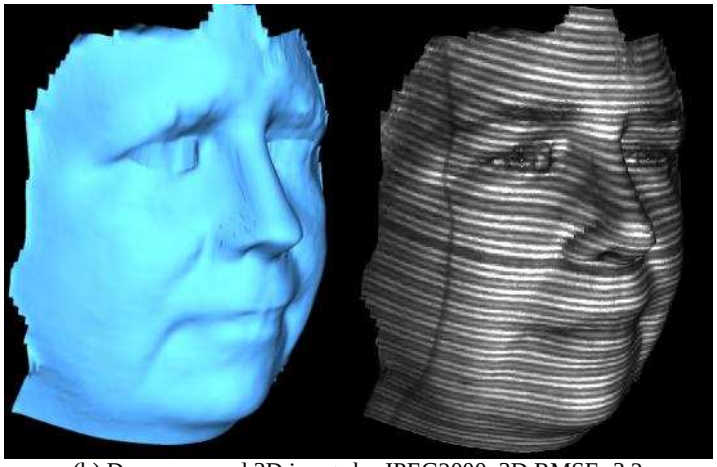


(e) Decompressed 3D image by JPEG, Quality=17%, 3D RMSE=3.72

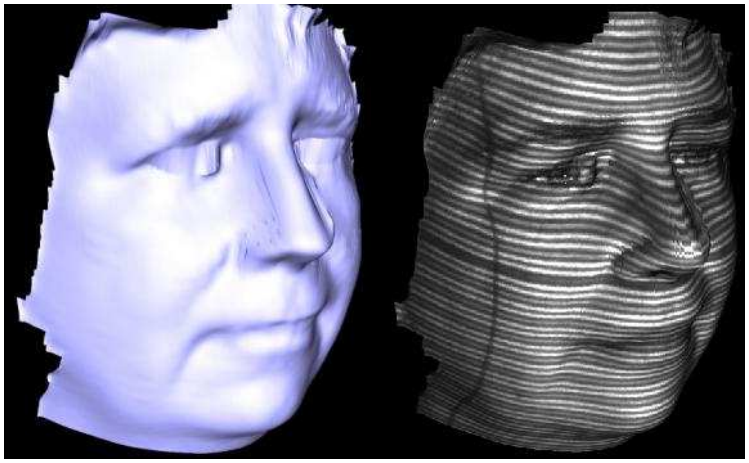
Figure 3.13: (a), (b) and (c) Decompressed 3D Girl image by JPEG2000, (d), (e) Decompressed 3D Girl image by JPEG.For low quality, JPEG cannot compress to29.1 KB.



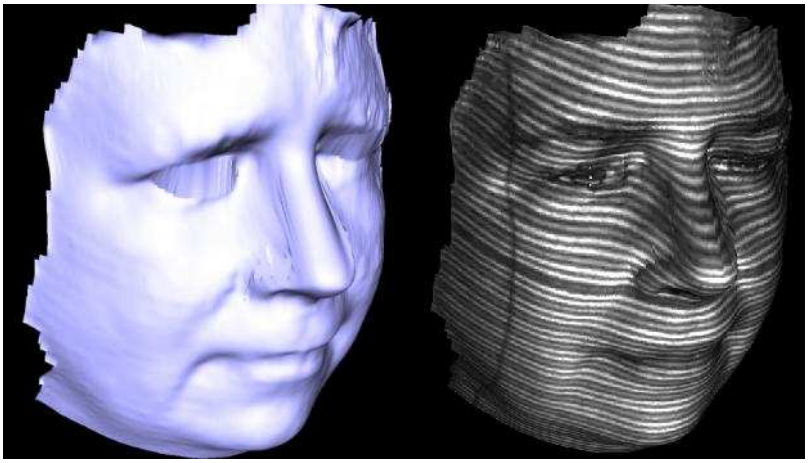
(a) Decompressed 3D image by JPEG2000, 3D RMSE=3.14



(b) Decompressed 3D image by JPEG2000, 3D RMSE=3.2



(c) Decompressed 3D image by JPEG, Quality=56%, 3D RMSE=2.55,



(d) Decompressed 3D image by JPEG, Quality=13%, 3D RMSE=2.75

Figure 3.14: (a), (b) Decompressed Women image by JPEG2000, (c), (d) Decompressed 3D Women image by JPEG. For low quality JPEG cannot compress to24.5 KB.

3.6. Conclusion

In this Chapter, we demonstrated a new image compression methodwhich is used in 3D applications. Our method is based on two different types of transformations: DWT and DCT respectively followed by the Matrix Minimization algorithm which is proposed in this thesis. The results show that our approach produced image quality at higher compression ratios that are capable to reconstruct the3D object. Another advantage, our approach has better image quality than JPEG and JPEG2000. On the otherhand, the steps in the algorithm are more complex than JPEG and JPEG2000. The most important feature ofthis method is its ability to provide high quality image athigh compression ratios.The main features of the techniques proposed and demonstrated in this Chapter are highlighted as follows [42].

1- The two transformations are proposed by the method to increase the high-frequency coefficients, this is one of the reasons to achieve higher compression ratios,
2- The proposed Matrix Minimization algorithm is used to collect each three adjacent coefficients from the high-frequency matrices converting to a single floating point value. This is a lossless step that reduces the size of the data and at the same time preserves image quality,
3- The main reason touse Daubechieswavelet (db3) is that it helps our approach to get higher compression ratios, because the Daubechies (db3) DWT family can zoom-in an image and all high-frequency sub-bands are set to zeroatthe first level: HL1, LH1 and HH1 (i.e. the high-frequency sub-matrices are ignored - See Table 3.3),
4- The new decoding algorithm is proposed in this chapter called LSS-Algorithm, whichrepresents the core of the proposed decompression algorithm. This algorithm retrieve a matrix from a one-dimensional array depending on RandomWeightValues (i.e. which is themain key responsible for coding/decoding).In addition, the LSS-Algorithm represents lossless decompression byrecovering the exact original coefficients,
5- The RandomWeightValues with Limited-Data are the keys used for coding and decoding an image, without these two keys an image cannot be recovered,
6- Another feature oftheproposed approach has better visual image quality at higher compression ratios compared with JPEG and JPEG2000. This is becausetheapproach removes most of the block artefacts caused by the 8x8 two-dimensional DCT of the JPEG technique. Also, our approach removes some blurring caused by quantization used in multi-level DWT of the JPEG2000 [22].

The disadvantages of the methods are illustrated as follows.

1- The compression/decompression steps are more complex than JPEG and JPEG2000, leading to increased execution times compared with JPEG and JPEG2000. Furthermore, the LSS-Algorithm iterative method is particularly complex.
2- Because theMatrix Minimization algorithm converts each integer coefficients to floating-point number, thereby causing increasing header-compressed file size.

Chapter 4

DWT-JPEGbased Compression with Block Sequential Search Decompression

4.1. Introduction

The JPEG image compression method is based on the Discrete Cosine Transform (DCT) (See section 1.2.1)[117]. The image is divided into segments and each segment is then a subject of the transform, creating a series of frequency components that correspond with detail levels of the image[127, 133]. A step beyond JPEG is the JPEG2000 technique that is based on DWT (See section 1.2.2) which is one of the mathematical tools for hierarchically decomposing functions. Image compression using Wavelet Transforms is a powerful method that is preferred by scientists to get compressed images at higher compression ratios with higher PSNR values [3][5]. Its superiority in achieving high compression ratio, error resilience, and other features promotes it to become today's compression standard leading to the JPEG2000 ISO.

As referred to the JPEG abbreviation, which stands for Joint Photographic Expert Group, JPEG2000 codec is more efficient than its predecessor JPEG and overcomes its drawbacks [12]. It also offers higher flexibility compared to even many other codec such as region of interest, high dynamic range of intensity values, multi component, lossy and lossless compression, efficient computation, compression rate control, etc. The robustness of JPEG2000 stems from its utilization of the Discrete Wavelet Transform (DWT) in encoding the image data. DWT exhibits high effectiveness in image compression due to its support to multi-resolution representation in both spatial and frequency domains. In addition, DWT supports progressive image transmission and region of interest coding [118,119].

We described in the previous Chapter a two level DWT and two level DCT transforms applied to 2D structured light images. The drawbacksof that method motivated us to reduce the number of transformation steps and increase the search algorithm’ s speed to reduce coding and decoding time.In this Chapter, we introduce a new method of applying the JPEG technique with the Discrete Wavelet Transform (DWT) for high-resolution compression. This image compression algorithm starts with transforming an image by a single level DWT,followed by the JPEG technique applied to the "LL" sub-band (Low-frequency coefficients) this process is called here the JPEG Transform. Next, we separate the final transformed matrix into a DC-Array and an AC-Matrix containing the DC values and the AC coefficients respectively. Finally, the Matrix Minimization algorithm is applied to the AC-Matrix followed by arithmetic coding[133].

The novel decompression algorithm proposed in this chapter is a Block Sequential Search Algorithm, which is represents the inverse of theMatrix Minimization algorithm. This search algorithm consist of pointers (*P*) searching as a block to find the original AC-coefficients.

Thereafter, it combines all decoded DC-values with the decoded AC-coefficients in one matrix followed by inverse JPEG transformed and inverse DWT. Thisnew technique is tested by compression and reconstruction of 2D structured light images. Additionally, this technique is compared with JPEG and JPEG2000 algorithm by using 2D and 3D RMSE[133].

4.2. Proposed 2D Image Compression Algorithm

The JPEG technique is one of the techniques used in image compression,an important feature of the JPEG method is the "Quality" parameter, which allows the user to adjust the amount of the data lost over a very wide range. In this Section, we explain in detail theJPEG transformation applied on the outputs of a discrete wavelet transform. The JPEG transformation consists of: 1) Apply DCT on each 8x8 block followed by a quantization process. 2) Zigzag scan converting each block into 64 coefficients, and store the 64-coefficients in two different matrices [130]. Figure 4.1 describes the proposed DWT-JPEG algorithm steps.

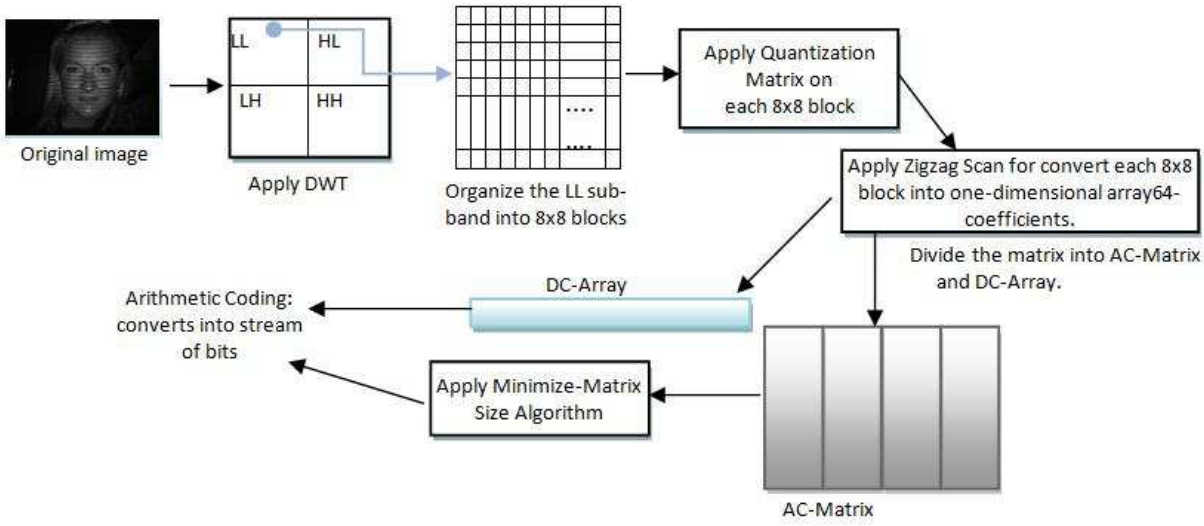


Figure 4.1: Proposed DWT-JPEG Compression Techniques

4.2.1. The Discrete Wavelet Transform

DWT is the first phase in the proposed image compression algorithm, to produce four sub-bands (See Section 3.2.1) [131]. Most values in the high-frequency domains (i.e HL, LH and HH) are insignificant coefficients without affecting on the reconstructed image. For this reason all the high frequency domains are discarded in this research (i.e. set all values to zero), and this does not mean that the image will lose much information, this depends on the

image dimensions. Figure 4.2 shows the decomposition image by Daubechies filter, and then recomposes sub-bands without high-frequencies[134].

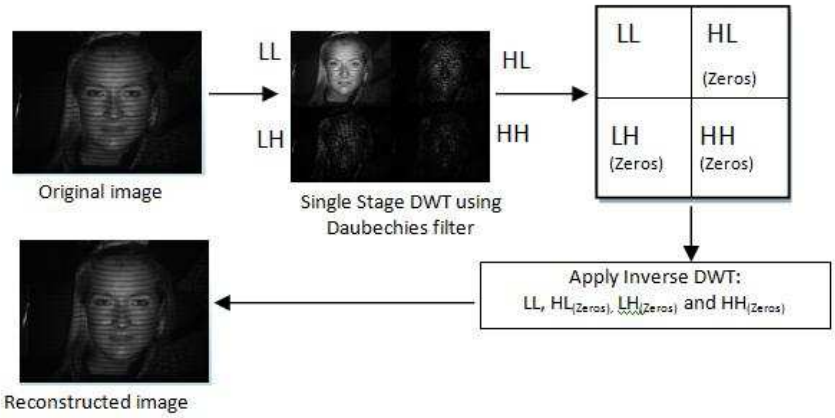


Figure 4.2: reconstructed image by using Daubechies single stage DWT

4.2.2. The JPEG Transform

The "LL" sub-band partitioned into non-overlapping 8x8 blocks, each block is transformed by using a two-dimensional DCT to produce de-correlated coefficients. Each 8x8 frequency domain consists of DC-value at the first location, while the other coefficients are called AC coefficients (See Section 3.2.2)[5]. After applying the two-dimensional DCT on each 8x8 block, each block is quantized by the Quantization Matrix "QM" using dot-division-matrix with truncates the results. This process removes insignificant coefficients and increases the number of zeros in the each block. QM computes as follows[132, 133]:

$$QM(i,j) = \begin{cases} Block + (i+j), & if\ (QM(i,j)) = odd \\ Block + (i+j) + 1\ if\ (QM(i,j)) = even \end{cases} \quad (4.1)$$

Where \ Block: is represented block size i ,j=1,2,3…,Block

$$Q\quad(i,j) = Q\quad(i,j)\,S_i \quad (4.2)$$

In the above Eq. (4.2), the factor "Scale" is used to increase/decrease the values of the "QM". Thus, image details are reduced in case factor Scale >1. There is no limit range for this factor, because this depends on the DCT coefficients[133].

Each quantized 8x8 block is converted into one-dimensional array (i.e. the array contains 64 coefficients) by zigzag scan [118].Whereas, the first value transferred into new array called DC-Array, while the other63 coefficients are stored into a new matrix "LL_{AC}". Finally, the DC-Array is compressed by Arithmetic coding. The Arithmetic coding is one of the important methods used in data compression method, especially used in JPEG2000. Arithmetic coding depends on "Low" and "High" equations to generate streams of bits [135].

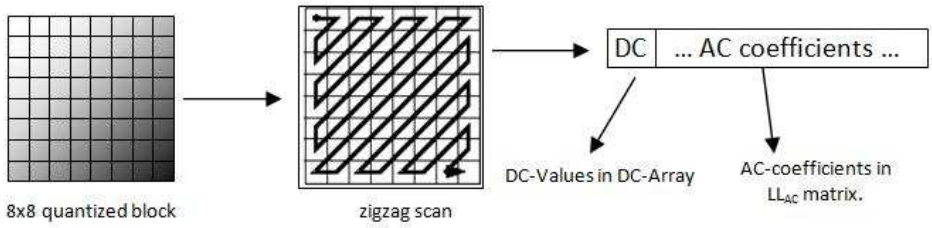


Figure 4.3:Operation and separation of DC-value from the 64 coefficients

4.2.3. The Matrix Minimization Algorithm

The LL_{AC} matrix ready for coding by the Matrix Minimization algorithm, which is applied to each three coefficients (See section 3.2.3), to produce a single data item.

In this Chapter we apply the Matrix Minimization algorithm to each three columns of an image, this means reducing each three columns to a single coded column. However, the bit size for each data in the minimized array is increased. Figure 4.4 illustrates the conversion of three columns into one dimensional array [42,133].

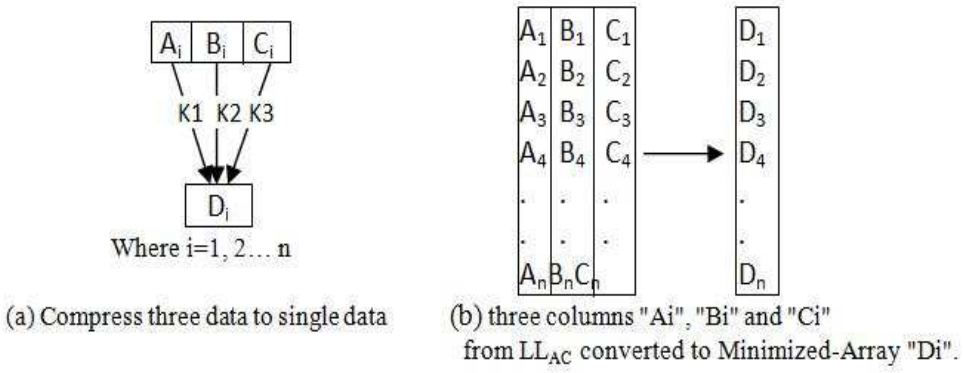


Figure 4.4: MatrixMinimization Algorithm

In above Figure 4.4 (a) K1, K2 and K3 represent the conversion keys of the Matrix Minimization algorithm. The following equation illustrates converting three data to a single data item(See List 3.1)[42,133].

$$Di=(K1\ Ai)+ (K2\ Bi) + (K3\ Ci) \quad (4.3)$$

Where\ i=1, 2, 3,…n

If the key values are lost, the data cannot be retrieved, because the keys are used for both coding and decoding. The key values are generated through a random number generator. For example, Key1= 0.8147, Key2=0.9058, and Key3=0.1270. The keys are generated once for all matrix data and, after calculation, all coded data "Di" are arranged together into a one-dimensional array[42,133].

Before applying the Matrix Minimization algorithm, the probability of the data for AC-matrix is computed. These probabilities are called Limited-Data, which are used later in the decompression stage (See Figure 3.5). The Limited-Data set is stored in the header file as additional information of compressed data [42,133].

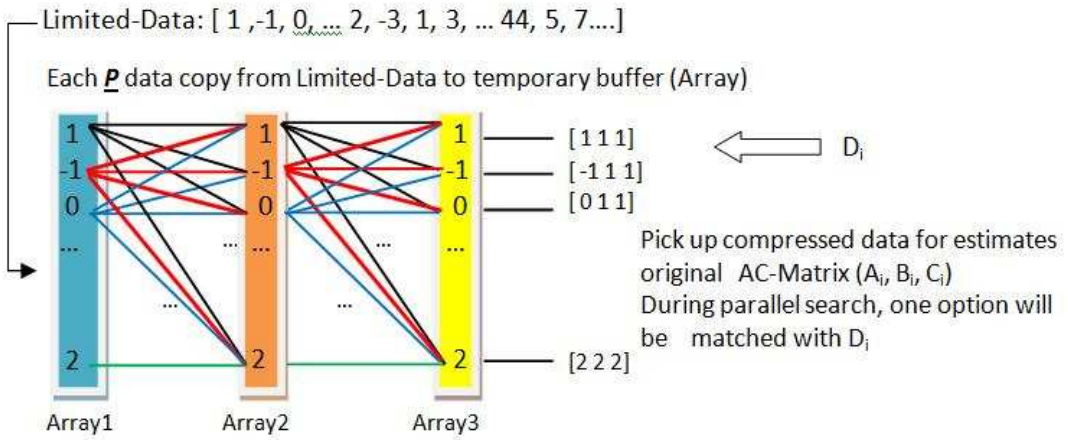
4.3. Proposed Decompression Algorithm

The decompression algorithm represents the reverse steps of the proposed image compression method. First, apply arithmetic decoding to decompress the DC-array. Thereafter, decode theminimized-array. Second, usethe novel Block Sequential Search Algorithm (BSS-Algorithm), which is the inverse of the Matrix Minimization Algorithm to reconstruct theAC-Matrix. The BSS-Algorithmestimates (Ai, Bi and Ci) by using "Di" with the compression keys. Whereas, Ai, Bi and Ci representstimed columns of the AC-Matrix. The BSS-Algorithm can be described by the following steps[133].

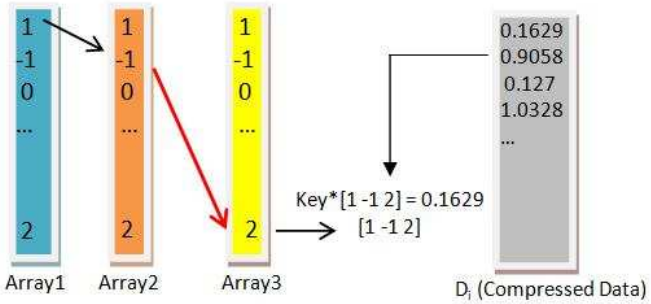
Step 1: BSS-Algorithm starts by picking the first block of data from the Limited-Data the size of this block P. Each value of block relates to each other like a network as shown in Figure 4.5, where "Column-1" is connected with "Column-2", which is connected with "Column-3". In other words, the search algorithm computes all options in parallel. For example: A=[1 -1 0] , B=[1 -2 0] and C=[3 -1 5], and P=3.According to Eq. (4.3), "A","B" and "C" are computed 27 times. This means, all options are computed in parallel and one option will be matched with "Di", and "Ai", "Bi" and "Ci" in "Column-1", "Column-2" and "Column-3" represented the decompressed data[133].

Initially, the BSS algorithm starts with P=10 from "Limited-Data(1…10.)" which is used by the algorithm to estimate three columns (A, B and C), as mentioned in Figure4.5(a). Thereafter, the algorithm starts searching for the original data (Ai, Bi and Ci) which depends on compressed column "Di" and the values of the keys. The first iteration starts with matching the selected "Di" with 10 outputs. In other words,Eq.(4.3)is executed 1000 times in parallel to find the original values for columns (A,B and C) as mentioned in Figure 4.5(b). If the result is unmatched, in this case the second option will be taken form "Limited-Data(11…20.)" (i.e. selecting another 10 data from Limited-Data transferred to Array1, while "Array2" and "Array3" remain in same old options.If the processing still cannot find the result, in this case"Array2=Array1" (i.e. transferred data from Array1 to Array2), then a new processing starts.Through this explanation, "Array1", "Array2" and "Array3" are working like a digital clock: sec, min. and hour respectively, this process will continue until find all original columns (Ai, Bi and Ci) in the AC-Matrix[133].

Step 2: In this step, the decompressed AC-Matrix is composed with each DC-value (i.e. DC-values from the DC-array), then followed by inverse zigzag scan to convert each 64-coefficients to 8x8 blocks. These blocks are combined with each other to build the LL sub-bands. Subsequently, apply inverse quantization (i.e. dot-multiplication), followed by inverse DCT on each 8x8 block. Finally, apply the inverse DWT for obtaining the 2D image. The decompression algorithm steps are showed in Figure 4.6[133].



(a) copy ***P*** data from Limited-Data to temporary "Array1" for BSS-Algorithm



(b) data matched through BSS-Algorithm

Figure 4.5: (a, b) strategy for the BSS-Algorithm

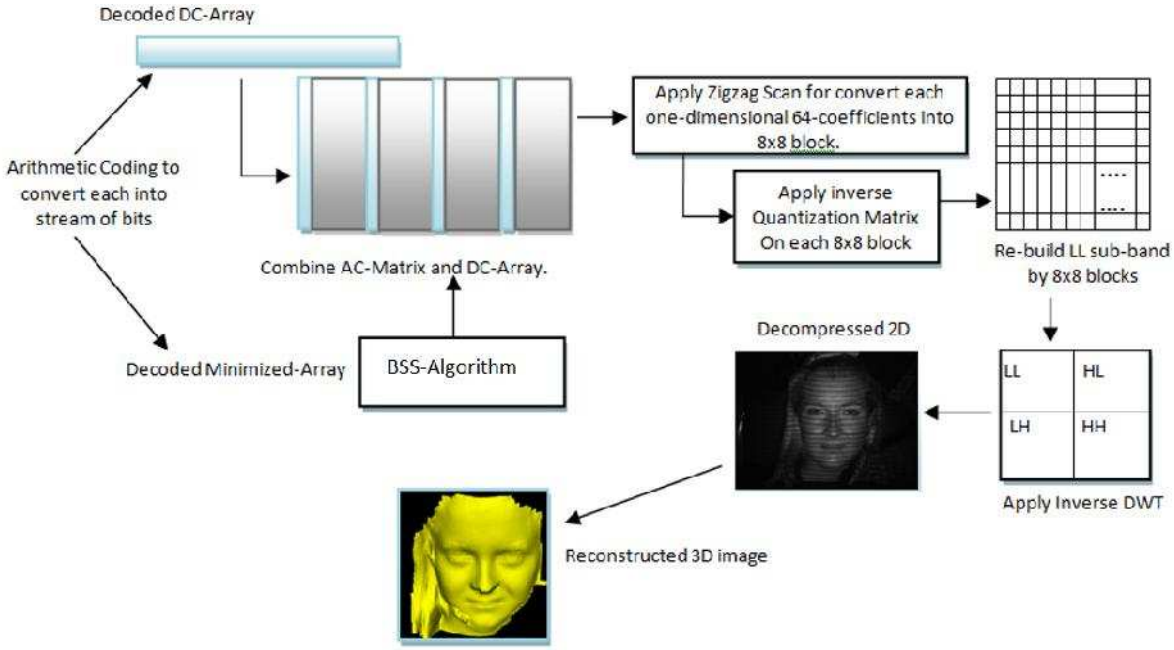


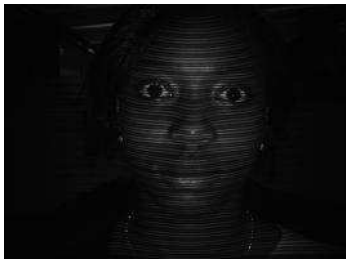
Figure 4.6: Flowchart of the proposed Decompression Algorithm

4.4. Experimental Results

The proposed compression method was applied on five types of images, as shown in Figure 4.7. The tests have been performed using Daubechies DWT (db3), the block sizes used by DCT (4×4 and 8×8). The results described below used MATLABfor 2D image compression in connection with a 3D visualization software (See Section 3.4) running on an AMD Quad-Core microprocessor. Tables: 4.1, 4.2, 4.3, 4.4 and 4.5 show the compressed sizes for each image.

(a) 2D BMP “Wall”, dimension1280x1024 pixels (b) 2D BMP “Corner”, dimension1280x1024 pixels

(c) 2D BMP “Face1”, dimension 1392x1040 (d) 2D BMP “Face2”, dimension 1392x1040



(e) 2D BMP “Face3”, dimension 1392x1040

Figure 4.7: (a, b) Colour 2D BMP image, size=3.75MB, (c,d,e) grayscale 2D BMP image, size=1.38 MB

Table 4.1: 2D image"Wall.bmp" of 3.75 MB compressed by the proposed image compression algorithm

Scale – parameter used by quantization		**Block size used by JPEG-Transformation**	**Compressed Size (KB)**	**Compression Ratio**
Luminance Y / single layer	**Chrominance [Cb , Cr]**			
2	[2,2]	8x8	27.2	99.2%
2	[4,4]	8x8	21.7	99.4%
4	[4,4]	8x8	17.5	99.5%
4	[8,8]	8x8	13.5	99.6%

Table 4.2: 2D image"Corner.bmp" of 3.75 MBcompressed by the proposed image compression algorithm

Scale – parameter used by quantization		**Block size used by JPEG-Transformation**	**Compressed Size (KB)**	**Compression Ratio**
Luminance Y / single layer	**Chrominance [Cb , Cr]**			
2	[2,2]	8x8	52.6	98.6%
2	[4,4]	8x8	39.9	98.9%
4	[4,4]	8x8	33.4	99.1%
4	[8,8]	8x8	25.1	99.3%
8	[8,8]	8x8	20.1	99.4%

Table 4.3: 2D image"FACE1.bmp" of 1.38 MB compressed by the proposed image compression algorithm

Scale – parameter used by quantization (single layer)	Block size used by JPEG-Transformation	Compressed Size (KB)	Compression Ratio
2	4x4	51.6	96.3%
2	8x8	28.4	97.9%
4	8x8	16.3	98.8%
5	8x8	13.5	99%
6	8x8	11.6	99.1%
8	8x8	9	99.3%

Table 4.4: 2D image"FACE2.bmp" of 1.38 MB compressed by the proposed image compression algorithm

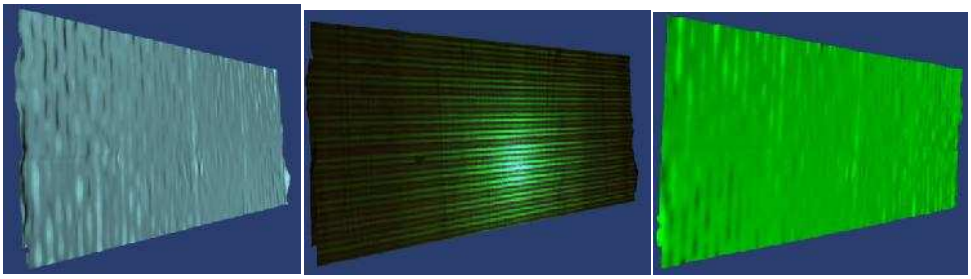
Scale – parameter used by quantization (single layer)	Block size used by JPEG-Transformation	Compressed Size (KB)	Compression Ratio
2	4x4	39	97.2%
2	8x8	20	98.5%
4	8x8	11.6	99.1%
5	8x8	8.4	99.4%
6	8x8	9.6	99.3%
8	8x8	6.7	99.5%

Table 4.5: 2D image"FACE3.bmp" of 1.38 MB compressed by the proposed image compression algorithm

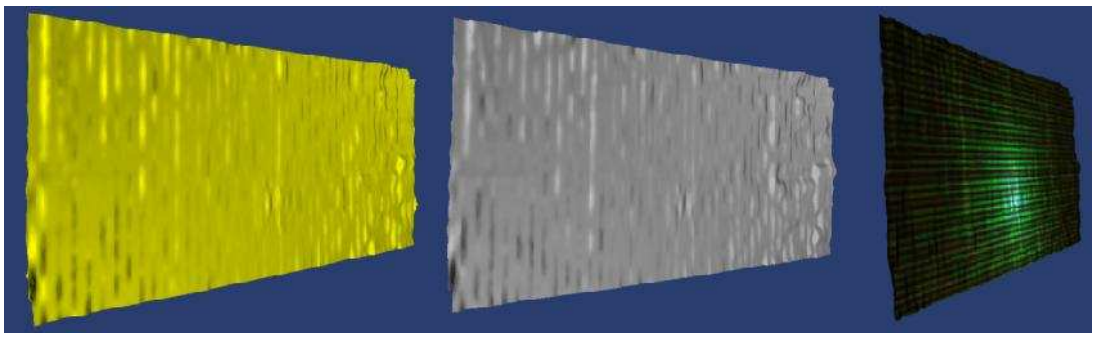
Scale – parameter used by quantization (single layer)	Block size used by JPEG-Transformation	Compressed Size (KB)	Compression Ratio
2	4x4	33	97.6%
2	8x8	16.8	98.8%
4	8x8	9.4	99.3%
5	8x8	7.7	99.4%

The proposed decompression algorithm was applied to each compressed image, as mentioned before in section 4.2. The decompressed algorithm shows a range of image quality according to "Scale" parameter and block size used in the JPEG-Transformation (See Eq.(2)). Figure

4.8, Figure 4.9, Figure 4.10, Figure 4.11 and Figure 4.12 showsequence of decompressed 3D images: Wall, Corner, Face1, Face2 and Face3 respectively where the quality of the 3D reconstruction can be assessed.

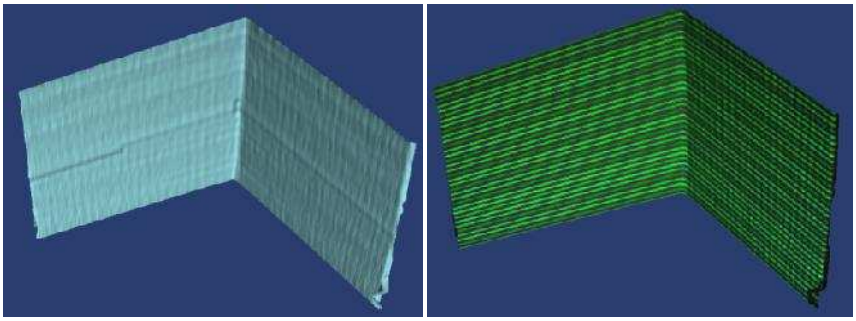


(a) Decompressed "Wall" 3D image Scale=[2,2,2], 3D shaded and texture, (b) Decompressed "Wall" 3D image Scale=[2,4,4]

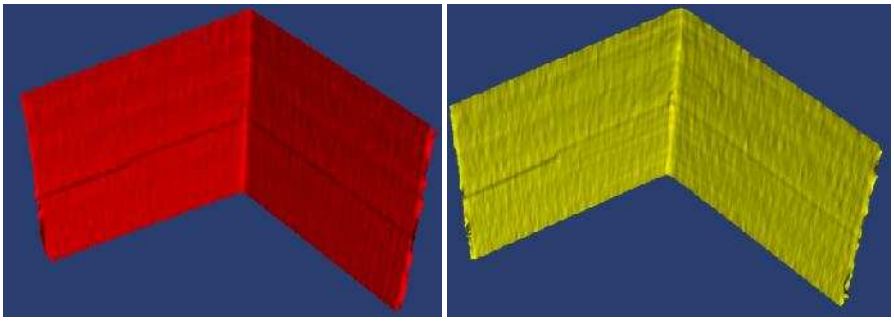


(c) Decompressed "Wall" 3D image Scale=[4,4,4], (d) Decompressed "Wall" 3D image Scale=[4,8,8], 3D shaded and texture

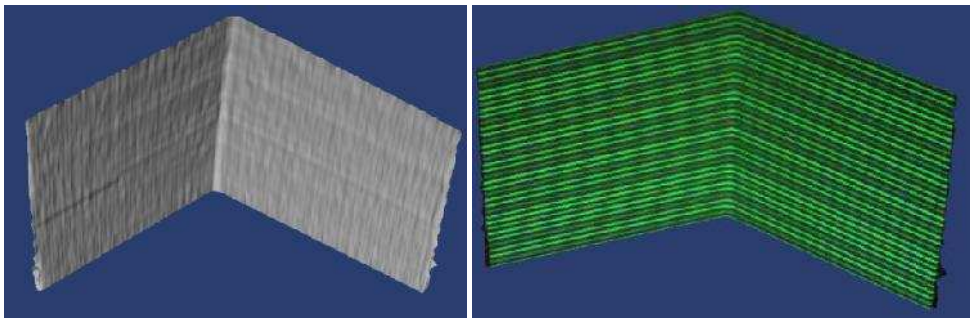
Figure 4.8: (a) Decompressed 3D wall image shaded by using high quality parameters applied on each layer in colour image "YCrCb". (b)Decompressed 3D wall image shaded by using normal quality parameters shows the details of 3D surface. (c) and (d) shows decompressed 3D surface at low quality parameters, the degradation starts appearing on the 3D wall image.



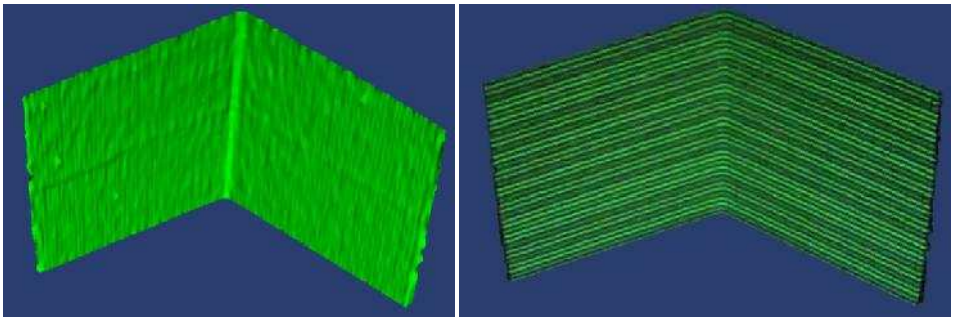
(a) Decompressed "Corner" 3D image Scale=[2,2,2], 3D shaded and texture



(b) Decompressed "Corner" 3D image Scale=[2,4,4] (c) Decompressed "Corner" 3D image Scale=[4,4,4]

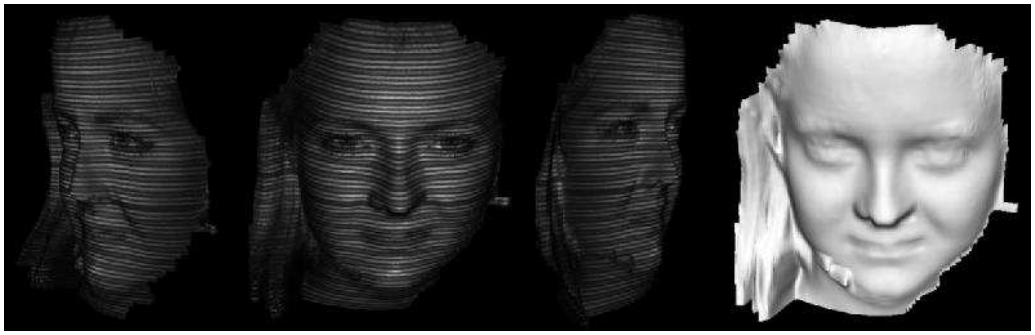


(d) Decompressed "Corner" 3D image Scale=[4,8,8], 3D shaded and texture

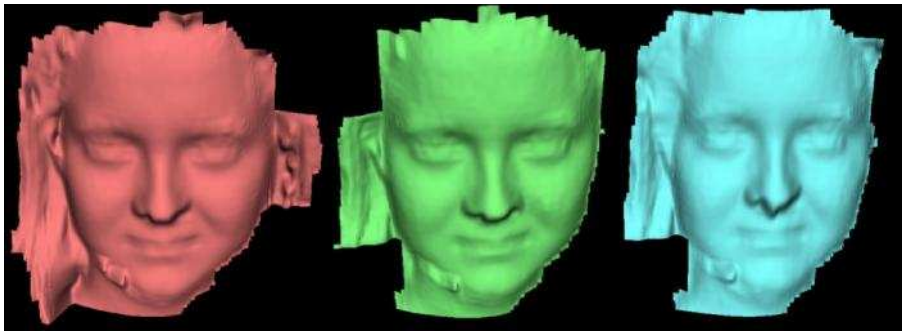


(e) Decompressed "Corner" 3D image Scale=[8,8,8], 3D shaded and texture

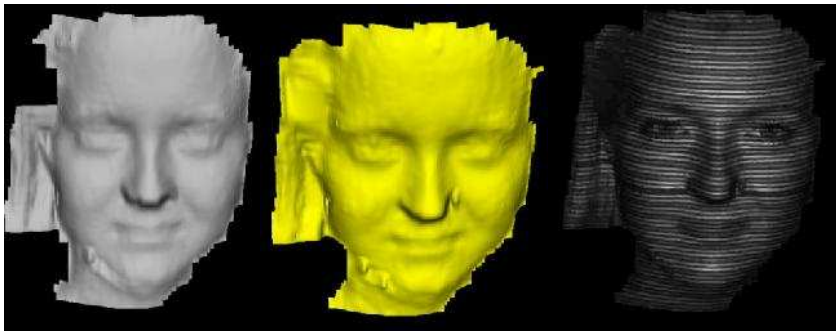
Figure 4.9: (a) Decompressed 3D corner image shaded by using high quality parameters applied on each layer in colour image "YCrCb". (b, c) Decompressed 3D Corner image shaded by using normal quality parameters shows the details of 3D surface. (d) Decompressed 3D Corner image shaded by using low quality parameters and the details of 3D surface still approximately same as the original. (e) Decompressed 3D Corner image shaded by using very low quality parameters and small amount of the degradation starts appearing on the 3D surface.



(a) Decompressed "Face1" 3D image Scale=2, block size =[4x4] 3D texture and shaded

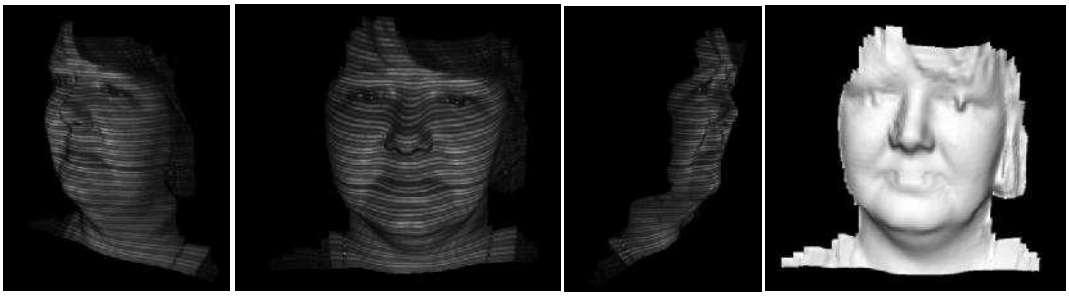


(b) Decompressed "Face1" 3D image Scale=2, block size=[4x4] 3D shaded (c) Decompressed "Face1" 3D image Scale=4, block size=[8x8] 3D shaded (d) Decompressed "Face1" 3D image Scale=5, block size=[8x8] 3D shaded

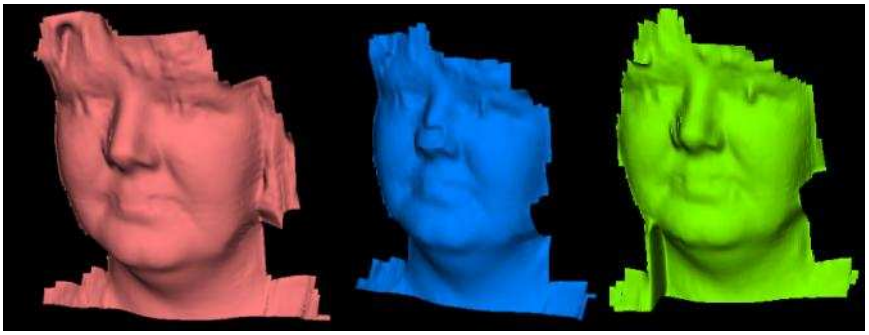


(e) Decompressed "Face1" 3D image Scale=6, block size=[8x8] 3D shaded (f) Decompressed "Face1" 3D image Scale=8 block size=[8x8] 3D shaded and texture

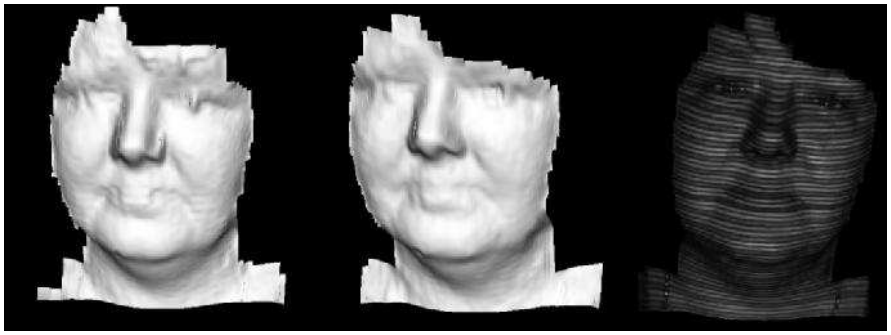
Figure 4.10: (a) Decompressed 3D Face1 image shaded by using high quality parameters applied on the grey-scale image. (b, c, d) Decompressed 3D Face1 image shaded by using normal quality parameters shows the details of 3D surface. (e) Decompressed 3D Face1 image shaded by using low quality parameters and the details of 3D surface still approximately the same as the original. (f) Decompressed 3D Face1 image shaded by using very low quality parameters, and small amount of the degradation starts appearing on the 3D surface.



(a) Decompressed "Face2" 3D image Scale=2, block size =[4x4] 3D texture and shaded

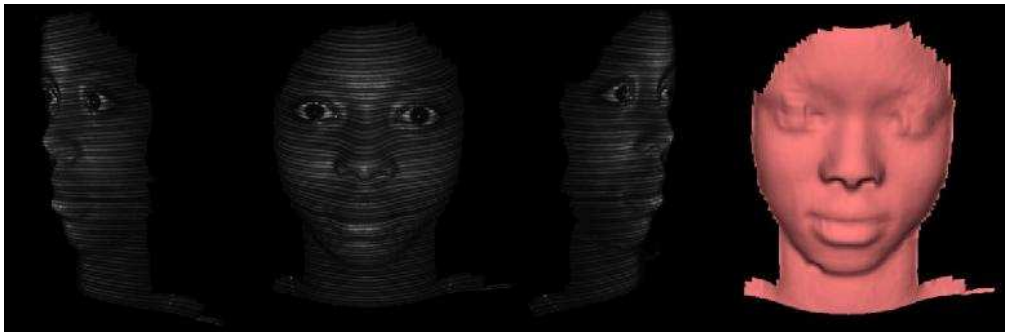


(b) Decompressed "Face2" 3D image Scale=2, block size=[8x8] 3D shaded (c) Decompressed "Face2" 3D image Scale=4, block size=[8x8] 3D shaded (d) Decompressed "Face2" 3D image Scale=5, block size=[8x8] 3D shaded

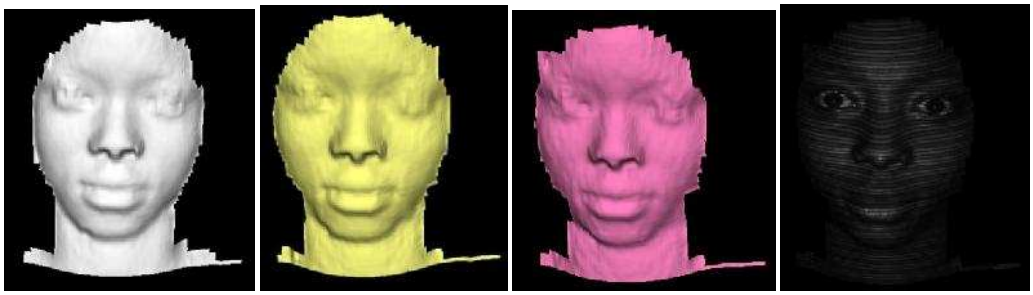


(e) Decompressed "Face2" 3D image Scale=6, block size=[8x8] 3D shaded (f) Decompressed "Face2" 3D image Scale=8 block size=[8x8] 3D shaded and texture

Figure 4.11: (a) and (b) Decompressed 3D Face2 image shaded by using high and normal quality parameters respectively, applied on the grey-scale image the details of 3D surfaces near to original 3D surface. (c) and(d) Decompressed 3D Face2 image shaded by using normal quality parameters and some details of 3D surface are changed. (e) and (f) Decompressed 3D Face2 image shaded by using low quality parameters, and small amount of the degradation appears on the 3D surface.



(a) Decompressed "Face3" 3D image Scale=2, block size =[4x4] 3D texture and shaded



(b) Decompressed "Face3" Scale=2, block size=[8x8] (c) Decompressed "Face3" Scale=4, block size=[8x8] (d) Decompressed "Face3" 3D image Scale=5, block size=[8x8] 3D shaded and texture

Figure 4.12: (a) Decompressed 3D Face3 image shaded by using high quality parameters applied on the grey-scale image. (b, c) Decompressed 3D Face3 image shaded by using normal quality parameters shows the details of the 3D surface is near to original 3D surface. (d) Decompressed 3D Face3 image shaded by using low quality parameters, and small amount of the degradation appeared on the 3D surface.

The following Tables: 4.6, 4.7, 4.8, 4.9 and 4.10 show time execution for the BSS-algorithm for each image using two types of pointers (***P***=5 and ***P***=10). The pointer refers to number of coefficients using in block for space search (i.e. searching in Limited-Data).

Table 4.6:BSS-Algorithm time execution for image: Wall.bmp

Parameters		**Time execution (sec.) for BSS-Algorithm, P=5**				**Time execution (sec.) for BSS-Algorithm, P=10**			
Luminance Y	**Chrominance [Cb , Cr]**	Y	Cb	Cr	Total time (s)	Y	Cr	Cb	Total time (s)
2	[2, 2]	10.95	6	11.87	28.82	7.8	4.36	7.45	19.61
2	[4, 4]	10	3.55	4.27	17.82	7.97	2.24	2.43	12.64
4	[4, 4]	3.66	2.9	4.29	10.85	3.19	2.6	3.0	8.79
4	[8, 8]	3.25	2.91	2.72	8.87	3	2.6	2.15	7.75

Table 4.7:BSS-Algorithm time execution for image:Corner.bmp

Parameters		**Time execution (sec.) for BSS-Algorithm, *P*=5**				**Time execution (sec.) for BSS-Algorithm, *P*=10**			
Luminance Y	**Chrominance [Cb , Cr]**	Y	Cb	Cr	Total time (s)	Y	Cr	Cb	Total time (s)
2	[2, 2]	36.61	9.28	21	66.89	27.51	9.42	14.25	51.18
2	[4, 4]	35.75	4.69	6.13	46.57	25.55	3.52	5.5	34.57
4	[4, 4]	5.44	4.58	6.48	16.50	6.22	3.57	5.99	15.78
4	[8, 8]	5.5	2.69	3	11.19	6.0	4.22	3.16	13.38
8	[8, 8]	2.94	2.93	3.32	9.19	2.93	4.46	3.41	10.8

Table 4.8:BSS-Algorithm time execution for image:FACE1.bmp

Parameters		BSS-Algorithm, ***P***=5	BSS-Algorithm, ***P***=10
Luminance Y	**Block size**	Total time(s)	Total time (s)
2	[4x4]	126.20	122.24
2	[8x8]	65.59	61.12
4	[8x8]	15.22	8.47
5	[8x8]	9.37	6.91
6	[8x8]	6.14	4.91
8	[8x8]	3.38	4.77

Table 4.9:BSS-Algorithm time execution for image:FACE2.bmp

Parameters		BSS-Algorithm, ***P***=5	BSS-Algorithm, ***P***=10
Luminance Y	**Block size**	Total time(s)	Total time (s)
2	[4x4]	81.83	76.58
2	[8x8]	44.69	43.24
4	[8x8]	7.53	8.42
5	[8x8]	7.76	8.59
6	[8x8]	4.74	5.24
8	[8x8]	4.39	4.49

Table 4.10:BSS-Algorithm time execution for image:FACE3.bmp

Parameters		BSS-Algorithm, ***P***=5	BSS-Algorithm, ***P***=10
Luminance Y	**Block size**	Total time(s)	Total time (s)
2	[4x4]	16.27	10.67
2	[8x8]	9.0	7.78
4	[8x8]	3.1	3.77
5	[8x8]	3.0	3.21

4.5. Comparison with JPEG and JPEG2000

Our approach is compared with JPEG and JPEG2000; these two techniques are used widely in digital image compression, especially for image transmission and video compression. The JPEG technique is based on the 2D DCT applied on the partitioned image into 8x8 blocks, and then each block encoded by RLE and Huffman coding [129, 133]. The JPEG2000 is

based on the multi-level DWT 9/7-daubechies filter, applied on the partitioned image and then each partition quantized and coded by Arithmetic coding. Most image compression applications allow the user to specify a quality parameter for the compression. If the image quality is increased the compression ratio is decreased and vice versa [131, 133]. The comparison is based on the 2D image and 3D image quality for testing the quality by Root-Mean-Square-Error (RMSE). Tables: 4.11, 4.12, 4.13, 4.14 and 4.15 show the comparison between the three methods for Wall, Corner, Face1, Face2 and Face3 respectively.

Table 4.11: Sequence of"Wall.bmp" 2D and 3D decompressed image by three methods, according to compressed size

Compression Ratio	Proposed Method		JPEG2000		JPEG	
	2D RMSE	3D RMSE	2D RMSE	3D RMSE	2D RMSE	3D RMSE
99.2%	2.8	6.44	2.6	0.26	9.4	0.33
99.4%	3.5	6.45	2.8	0.26	15.4	24.4
99.5%	3.7	0.49	3.1	0.55	**FAIL**	**FAIL**
99.6%	4.9	0.47	3.3	0.58	**FAIL**	**FAIL**

Table 4.12: Sequence of "Corner.bmp" 2D and 3D decompressed image by three methods, according to compressed size

Compression Ratio	Proposed Method		JPEG2000		JPEG	
	2D RMSE	3D RMSE	2D RMSE	3D RMSE	2D RMSE	3D RMSE
98.6%	5.8	0.07	4.7	0.05	11.2	1.15
98.9%	6.8	1.14	5.6	0.25	13.7	1.18
99.1%	7.1	1.14	6.1	1.83	16.0	1.81
99.3%	9.4	1.15	6.6	0.25	20	56.8
99.4%	9.9	0.34	7.2	1.17	**FAIL**	**FAIL**

Table 4.13: Sequence of "FACE1.bmp" 2D and 3D decompressed image by three methods, according to compressed size

Compression Ratio	Proposed Method		JPEG2000		JPEG	
	2D RMSE	3D RMSE	2D RMSE	3D RMSE	2D RMSE	3D RMSE
96.3%	4.3	0.85	4.0	0.66	4.0	1.8
97.9%	4.7	0.82	3.7	1.43	7.6	1.97
98.8%	5.4	1.48	4.7	1.8	12.2	116.5

99%	5.7	1.80	5.1	1.81	**FAIL**	**FAIL**
99.1%	6.1	1.98	5.4	1.93	**FAIL**	**FAIL**
99.3%	6.7	1.94	5.8	1.86	**FAIL**	**FAIL**

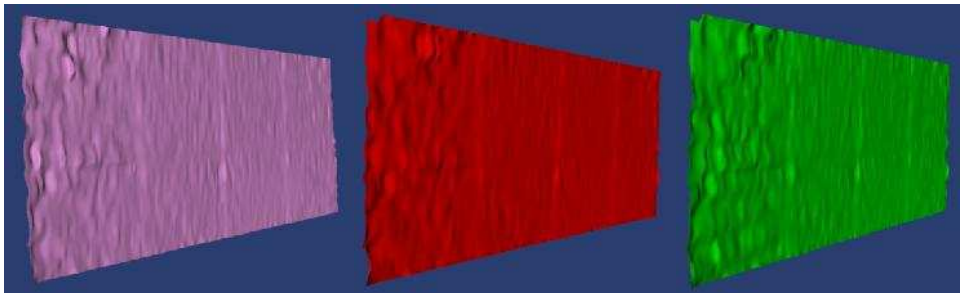
Table 4.14: Sequence of "FACE2.bmp" 2D and 3D decompressed image by three methods, according to compressed size

Compression Ratio	Proposed Method		JPEG2000		JPEG	
	2D RMSE	3D RMSE	2D RMSE	3D RMSE	2D RMSE	3D RMSE
97.2%	2.4	0.47	1.5	0.62	2.1	0.49
98.5%	2.8	0.66	1.8	0.5	2.6	15.0
99.1%	3.4	1.18	2.4	0.93	11.0	**FAIL**
99.4%	3.8	15.8	3.0	1.06	**FAIL**	**FAIL**
99.3%	4.1	14.9	3.1	15.2	**FAIL**	**FAIL**
99.5%	4.6	15.4	3.3	15.1	**FAIL**	**FAIL**

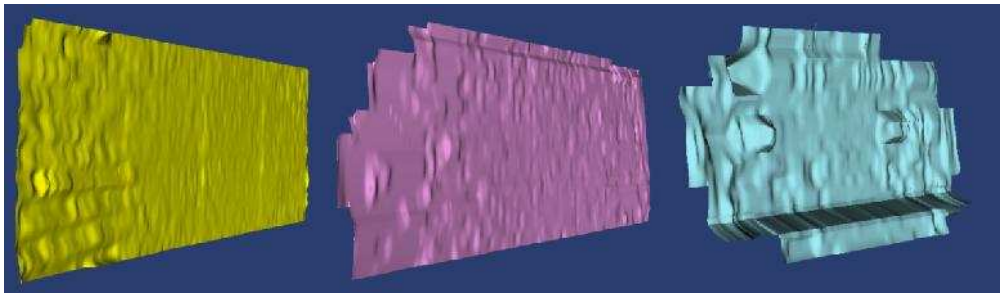
Table 4.15: Sequence of "FACE3.bmp" 2D and 3D decompressed image by three methods, according to compressed size

Compression Ratio	Proposed Method		JPEG2000		JPEG	
	2D RMSE	3D RMSE	2D RMSE	3D RMSE	2D RMSE	3D RMSE
97.6%	2.3	0.55	1.8	0.93	2.7	0.49
98.8%	2.6	0.55	2.4	0.97	9.7	**FAIL**
99.3%	3.3	0.59	2.9	0.67	**FAIL**	**FAIL**
99.4%	3.6	0.70	3.2	0.77	**FAIL**	**FAIL**

In the above tables: 4.11, 4.12, 4.13, 4.14 and 4.15 "**FAIL**" means that theJPEG algorithm was unable to compress/decompress an image at high compression ratio, while the other two methods (our proposed and JPEG2000) can compress/decompress successfully. In some cases, the 3D RMSE vary, if we compare it with 2D RMSE, this is because the dimensions of the original 3D image and 3D decompressed image are unmatched. In this case, the unmatched regions are discarded.On the other hand, RMSE is not enough to show the real comparison between these three methods.The following figures: 4.15, 4.16, 4.17, 4.18 and 4.19 shows the visual properties for the 3D decompressed images: Wall, Corner, FACE1, FACE2 and FACE3 respectively by using JPEG and JPEG2000 so a perceptual assessment of quality can be made.

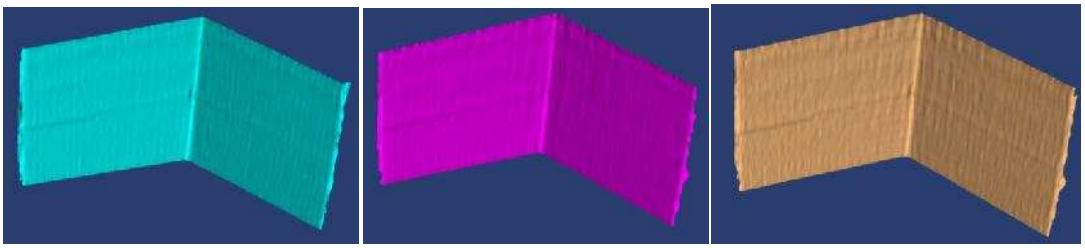


(a) JPEG2000, 3D RMSE= 0.26(b) JPEG2000, 3D RMSE=0.26 (c) JPEG2000, 3D RMSE=0.55
Compress size=27.2 Kbytes Compress size=21.7 KbytesCompress size=17.5 Kbytes

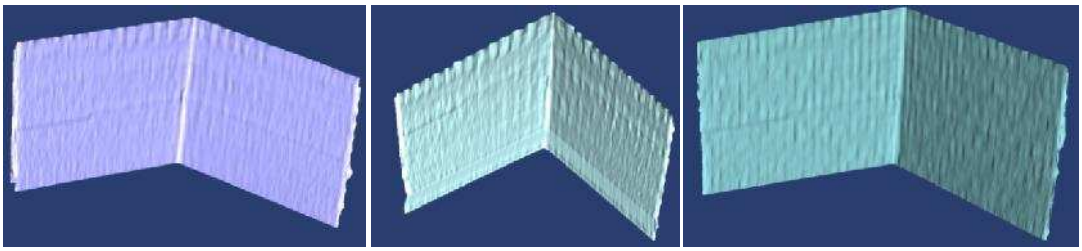


(d) JPEG2000, 3D RMSE=0.58 (e) JPEG, 3D RMSE=0.33 (f) JPEG, 3D RMSE=24.4
Compress size=13.5 Kbytes Compress size=27.2 Kbytes Compress size=21.7 Kbytes

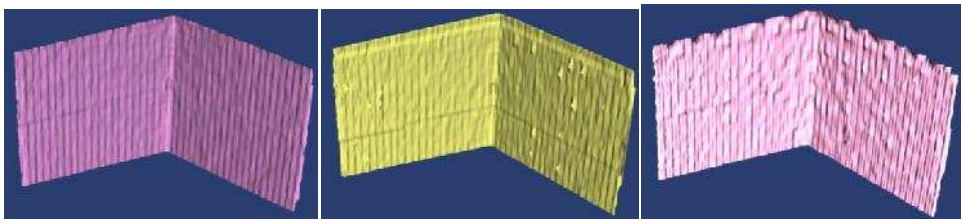
Figure 4.15:(a –d) Decompressed Wall image by using JPEG2000, (e,f) Decompressed Wall image by JPEG



(a) JPEG2000, 3D RMSE= 0.05 (b) JPEG2000, 3D RMSE=0.25 (c) JPEG2000, 3D RMSE=1.83
Compress size=52.2 Kbytes Compress size=39.9 Kbytes Compress size=33.4 Kbytes

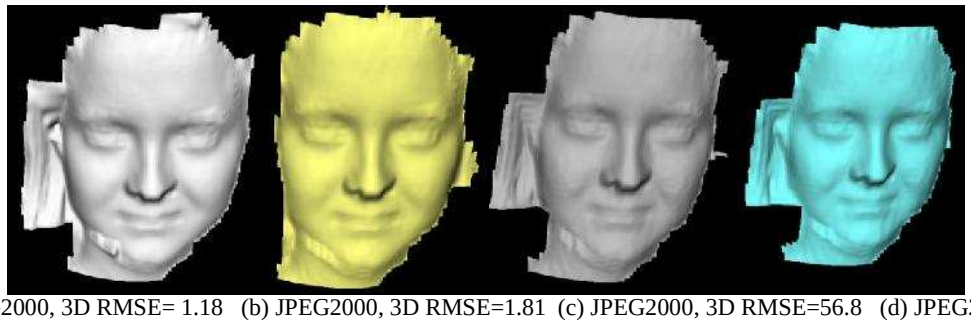


(d) JPEG2000, 3D RMSE= 0.25 (e) JPEG2000, 3D RMSE=1.17 (f) JPEG, 3D RMSE=1.15
Compress size=25.1 Kbytes Compress size=20.1 KbytesCompress size=52.2 Kbytes

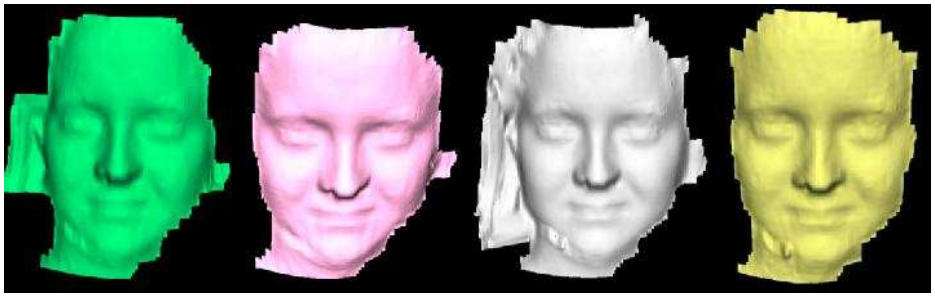


(g) JPEG, 3D RMSE= 1.18 Compress size=39.9 Kbytes (h) JPEG, 3D RMSE=1.81 Compress size=33.4 Kbytes (i) JPEG, 3D RMSE=56.8 Compress size=25.1 Kbytes

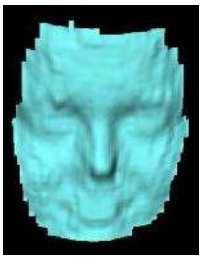
Figure 4.16: (a – e) Decompressed Corner image by JPEG2000, (f – i) Decompressed Corner image by JPEG.



(a) JPEG2000, 3D RMSE= 1.18 Compress size=51.6 Kbytes (b) JPEG2000, 3D RMSE=1.81 Compress size=28.4 Kbytes (c) JPEG2000, 3D RMSE=56.8 Compress size=16.3 Kbytes (d) JPEG2000, 3D RMSE=56.8 Compress size=13.5Kbytes

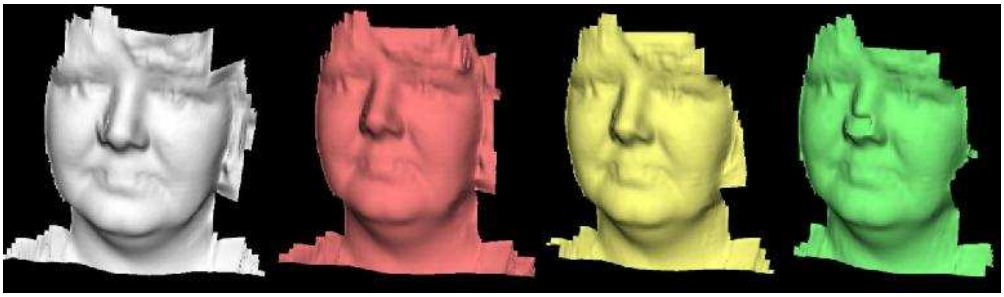


(e) JPEG2000, 3D RMSE=1.18 Compress size=11.6 Kbytes (f) JPEG2000, 3D RMSE=1.81 Compress size=9.0 Kbytes (g) JPEG, 3D RMSE=1.8 Compress size=51.6 Kbytes (h) JPEG, 3D RMSE=1.97 Compress size=28.4 Kbytes

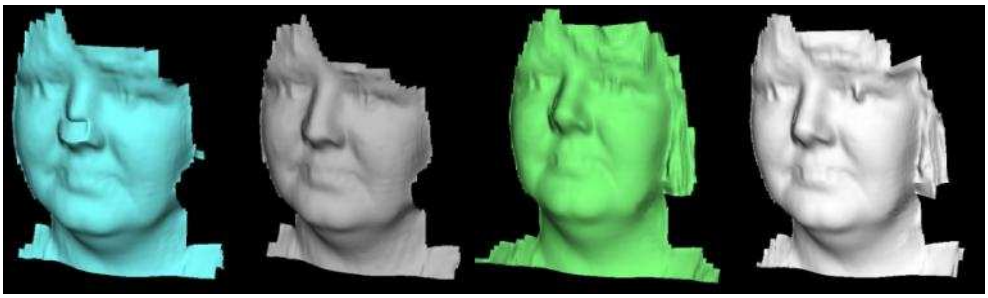


(i) JPEG, 3D RMSE=116.5
Compress size=16.3 Kbytes

Figure 4.17: (a – f) Decompressed FACE1 image by JPEG2000, (g – i) Decompressed FACE1 image by JPEG.

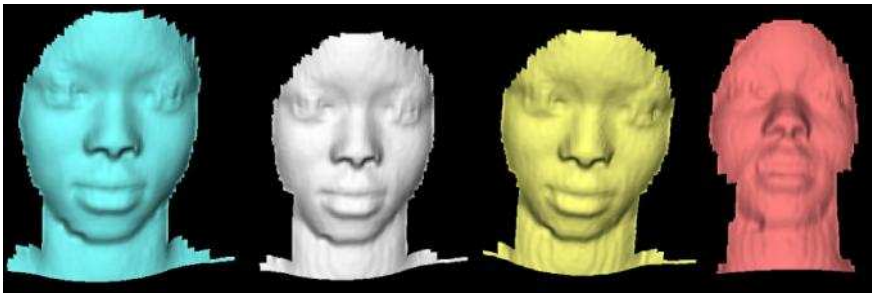


(a) JPEG2000, 3D RMSE=0.62 (b) JPEG2000, 3D RMSE=0.5(c) JPEG2000, 3D RMSE=0.93(d) JPEG2000, 3D RMSE=1.06
Compress size=39 Kbytes Compress size=20 Kbytes Compress size=11.6 KbytesCompress size=8.4 Kbytes

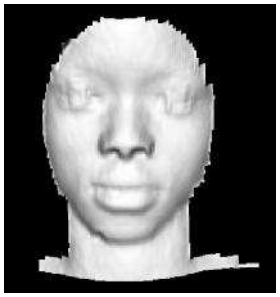


(e) JPEG2000, 3D RMSE=15.2 (f) JPEG2000, 3D RMSE=15.1(g) JPEG, 3D RMSE=0.49 (h) JPEG, 3D RMSE=15.0
Compress size=9.6 Kbytes Compress size=6.7 KbytesCompress size=39 Kbytes Compress size=20 Kbytes

Figure 4.18: (a – f) Decompressed FACE2 image by JPEG2000, (g, h) Decompressed FACE2 image by JPEG.



(a) JPEG2000, 3D RMSE=0.93 (b) JPEG2000, 3D RMSE=0.97 (c) JPEG2000, 3D RMSE=0.67(d) JPEG2000, 3D RMSE=0.77
Compress size=33 Kbytes Compress size=16.8 KbytesCompress size=9.4 Kbytes Compress size=7.7 Kbytes



(e) JPEG, 3D RMSE=0.49
Compress size=33 Kbytes

Figure 4.19: (a – d) Decompressed FACE3 image by JPEG2000, (e) Decompressed FACE3 image by JPEG.

4.6. Conclusion

In this Chapter we presented and demonstrated a new method of image compression used in 3D applications. The method is based on DWT and JPEG transformation with the proposed Matrix Minimization algorithm. The results showed that our approach introduced better image quality at higher compression ratios than JPEG and JPEG2000 being capable of accurate 3D reconstruction at higher compression ratios. On the other hand, it is more complex than JPEG2000 and JPEG. The most important aspects of the method and their role in providing high quality image with high compression ratios are discussed as follows [133]:

1- Using two transformations helped our compression algorithm to increase the number of high-frequency coefficients, and reduce the low-frequency domains leading to increased compression ratios.
2- The Matrix Minimization algorithm is used to collect each three coefficients from the AC-matrix, to a single floating-point value. This process converts a matrix into an array, leading to increased compression ratios and keeping the quality of the high-frequency coefficients.
3- The BSS-Algorithm represents the core of our decompression algorithm to reconstruct the exact original data (i.e. decompression algorithm), converting a one-dimensional array (i.e. the Minimized-Array) to the original matrix, and depends on the organized key-values and Limited-Data.
4- The key-values and Limited-Data are used in coding and decoding an image, without these information images cannot be reconstructed. This feature makes our approach useful in image encryption.
5- Our approach provides a better visual image quality compared to JPEG and JPEG2000. This is because our approach removes most of the block artefacts caused by the 8x8 two-dimensional DCT of the JPEG technique and this is due to the Matrix Minimization algorithm. Also, our approach uses a single level DWT rather than multi-level DWT of JPEG2000, for this reason blurring is removed by our approach.

However, there are more steps in the proposed compression and decompression algorithm than in the JPEG and JPEG2000 techniques.Also,the complexity of BSS-algorithm leads to increased execution time for decompression, because the iterative nature of the method is particularly complex.

Chapter 5

Enhanced DWT-DCT based Image Compression with Fast-Matching-Search Decompression

5.1. Introduction

We demonstrated in Chapters 3 and 4 that structured light images for 3D reconstruction can be compressed bythe proposed techniques with high accuracy and high compression ratios. The research addressed efficient compression of 2D images for 3D reconstruction and texture mapping. The proposed algorithms by Siddeq and Rodrigues [42,133,136] areuseful tomany 3D applications,scientific datasets, medical and engineering and so on.

In this Chapter,we describe a new method for image compression based on two separate transformations; a two-levelDWT and a two-level DCT, leading to an increased number of high-frequency matrices, which are then shrunk by the Enhanced Matrix Minimization algorithm. This Chapter demonstrates that our compression algorithm can achieve efficient image compression ratios up to 99.5% and superior accurate 3D reconstruction compared with standard JPEG and JPEG2000**[137]**.

5.2. The Proposed 2D Image Compression Algorithm

This section presents a novel lossy compression algorithm implemented via DWTand DCT. The algorithm starts with a two-level DWT. While all high frequencies (HL_1, LH_1, HH_1) of the first level are discarded, all sub-bands of the second level are further encoded. We then apply DCTto the low-frequency sub-band (LL_2) of the second level; the main reason for using DCTis to split into another low frequency and high-frequency matrices (DC and AC$Matrix_1$).The Enhanced Matrix Minimization algorithmis then applied to compress the AC-$Matrix_1$ and high frequency matrices (HL_2, LH_2, HH_2). The DC-$Matrix_1$ is subject to a second DCTwhose AC-$Matrix_2$ is quantized then subject to arithmetic coding together with DC-$Matrix_2$ and the output of EMMalgorithm as depicted inFigure5.1**[137]**.

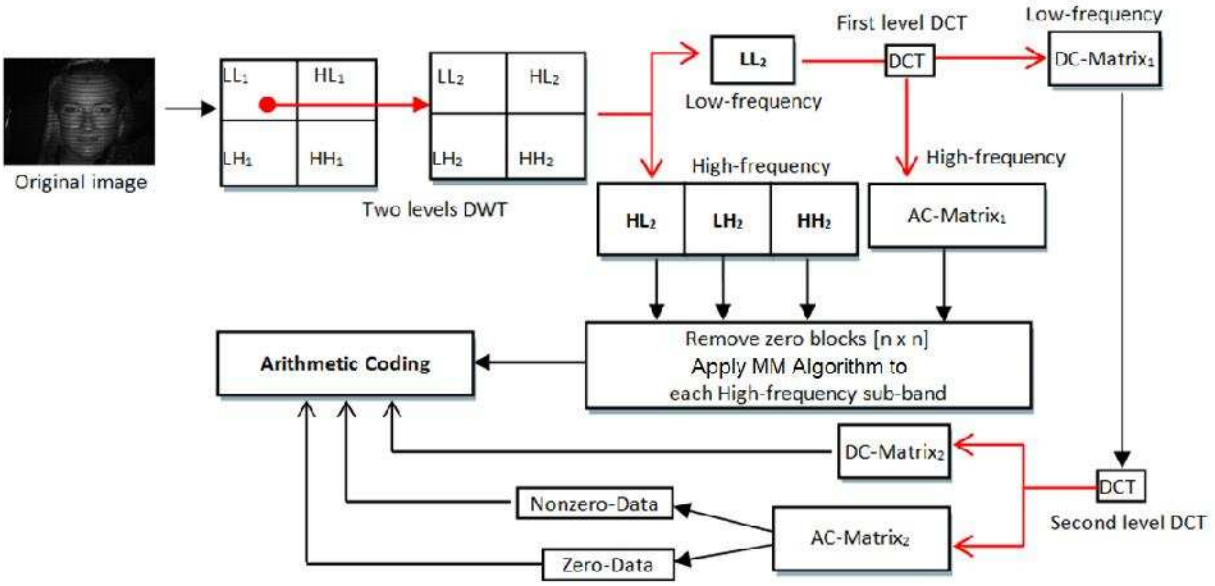


Figure5.1: Layoutproposed by the two-level DWT-DCT compression technique.

5.2.1. Two-Level Discrete Wavelet Transform (DWT)

The Wavelet Transform divides the image into four sub-bands: LL (i.e. which represents an approximation of the original image) and the other three represent details as described in Chapters 3 and 4. Practically, the details can be compressed into a few bytes, this is because these details do not affect significantly the image quality. Given this, we can set all detail coefficients to zero **[133,137]**. The DWT uses filters for decomposing an image; these filters helpto increase the number of zeros in the high frequency sub-bands. One common filter used in decomposition and composition is the *Daubechies Filter (db3)***[132,141]**.

In the proposed algorithm, the high frequency sub-bandof first level are set to zero (i.e. discard HL, LH and HH). These sub-bands do not affect image details. Additionally, only a small number of non-zero values are present in these sub-bands. In contrast, high-frequency sub-bands in the second level (HL2, LH2 and HH2) cannot be discarded, as this would significantly affect image quality. For this reason, high-frequency values in this region are quantized. The quantization process in this algorithm depends on the maximum value in each sub-band, as shown by the following equation **[137]**:

$$H_2 = \frac{H_2}{H_2 m} \tag{5.1}$$

$$H_2 m = \max(H_2)\, R \tag{5.2}$$

Where, "H_2" represents each high-frequency sub-band at second level in DWT (i.e. HL2, LH2 and HH2). While "H_2m" represents maximum value in a sub-band, and the "*Ratio*" value is used as a control for the maximum value, which is used to control image quality. For example, if the maximum value in a sub-band is 60 and *Ratio*=0.1, the quantization value is H_2m=6, this means all values in a sub-band are divided by 6.

Each sub-band has different priority for keeping image details. The higher priorities are: LH_2, HL_2 and then HH_2. Most of information about image details are in HL_2 and LH_2. If most of nonzero data in these sub-bands are retained, the image quality will be high even if some information is lost in HH_2.For this reason the *Ratio* value for HL_2, $LH_2$and $HH_2$is defined in the range ={0.1 … 0.5}.

5.2.2. Two Level Discrete Cosine Transform (DCT)

In this Section, we describe the two-levelDCT applied to low-frequency sub-band "LL_2" (see Figure5.1). A quantization is first applied to LL2 as follows. All values in LL_2 are subtracted by the minimum of $LL_2$and then divided by 2 (i.e. a constant even number). Thereafter, a two-dimensional DCTis applied to produce de-correlated coefficients. Each variable size block (e.g. 8x8)in the frequency-domain consists of: theDC-value at the first location, while all the other coefficients are called AC coefficients. The following steps illustrate thetwo-levelDCT implementation:

A- Organize LL_2 into 8x8 non-overlapping blocks (other sizes can also be used such as 16x16), then applyDCTto each block followed byquantization. The following equations represent theDCT and its inverse**[130,131,138]**

The quantization table is a matrix of the same block size that can be represented as follows:

$$Q(i,j) = B \qquad + (i+j) \tag{5.3}$$

$$Q(i,j) = Q(i,j)\, S \tag{5.4}$$

Where\i, j= 1,2,…Block , Scale= 1,2,3,… Block

After applying the two-dimensional DCT on each 8x8 or 16x16block, each block is quantized by the "*Q*" using dot-division-matrix, which truncates the results. This process removes insignificant coefficients and increases the number of zeros in each block. However, in the above Eq. (5.4), the factor "*Scale*" is used to increase/decrease the values of "*Q*". Thus, image details are reduced in case of *Scale*>1. There is no limiting range for this factor, because this depends on the DCT coefficients [137].

B- Convert each block to 1D array, and then transfer the first value to a new matrix called DC-Matrix. While the rest of data AC coefficients are saved into a new matrix called AC-Matrix. Similarity, the DC-Matrix is transformed again by DCT into another two different matrices: DCT-Matrix2 and AC-Matrix2. Figure5.2 illustrates the details of the two-level DCT applied to the sub-band LL_2.

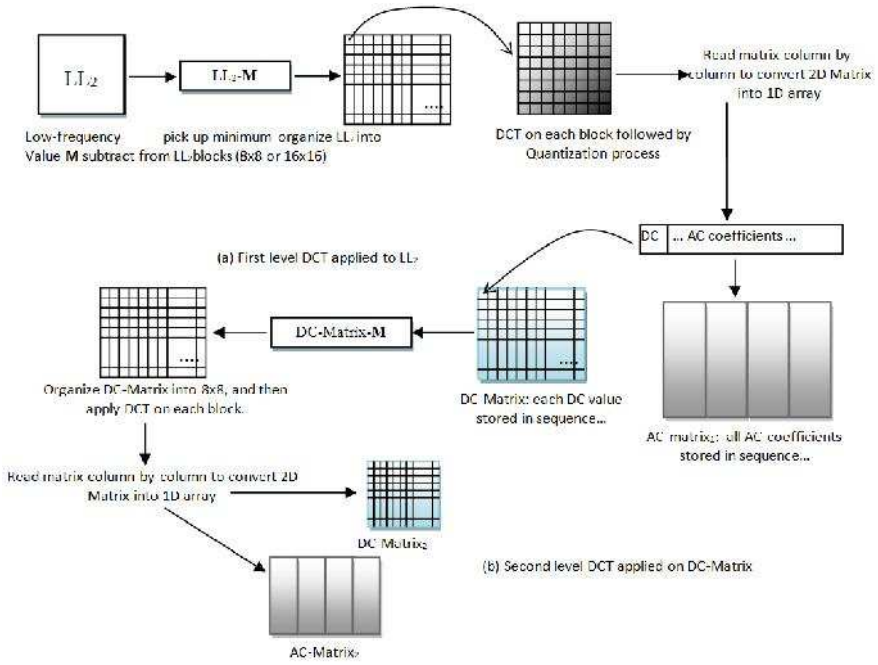


Figure5.2:(a) and (b) two levels DCT applied to LL_2

The result from DC-$Matrix_2$ size is very small and can be represented by a few bytes. On the other hand, the AC-$Matrix_2$ contains lots of zeros with only a few nonzero data. All zeros can be erased and just nonzero data are retained.

5.2.3 Enhanced Matrix Minimization Algorithm

Each high-frequency sub-band contains lots of zeros with a few nonzero data. We propose a technique to eliminate block of zeros, and store blocks of nonzero data in an array. This technique is useful for squeezing all high-frequency sub-bands, this process is labelled *Eliminate Zeros and Store Nonzero data (EZSN)* in Figure 5.3, applied to each high frequency independently. The EZSN algorithm starts to partition the high-frequency sub-bands into non-overlapping blocks [K x K], and then search for nonzero blocks (i.e. search for at least one

nonzero inside a block). If a block contains any nonzero data, this block will be stored in the array called *Reduced-Array*, with the position of that block. Otherwise, the block will be ignored, and the algorithm continues to search for other nonzero blocks. The EZSN algorithm is illustrated in List 5.1 below**[137]**.

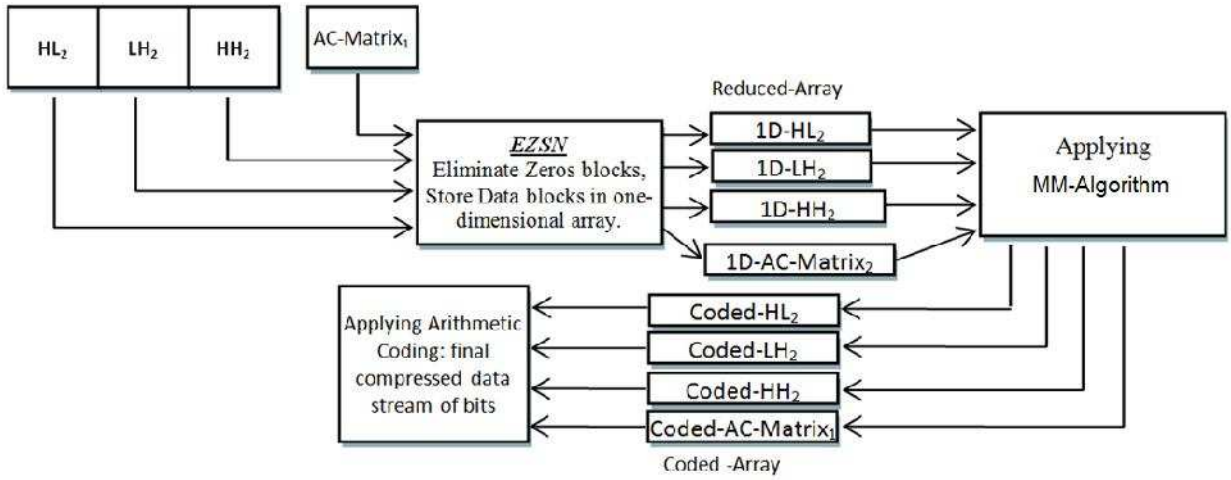


Figure5.3;Layoutof the Enhanced Matrix Minimization Algorithm

List 5.1:EZSN Algorithm

```
K=8;  %% block size = [KxK]
I=1; LOC=1
while (I< column size for high-frequency sub-band)
J=1;
while (J< row size for high-frequency sub-band)
Block[1..K*K]= Read_Block_from_Matrix(I,J) ;  %%read 8x8 block from high-frequency sub-band
if(Check_Block(Block) = = 'nonzero')  %%check the "Block" content,hasita nonzero?
POSITION [LOC] =I; POSITION [LOC+1] =J; %% Save original location for block contains
        %%nonzero data in high-frequency sub-band
LOC=LOC+2;
Forn=1: Block_Size* Block_Size
Reduced_Array[P]= Block[n]; %%save nonzero data in new array
++P;
Endfor
Endif
J=J+K;
Endwhile% inner loop
  I=I+K;
Endwhile% outer loop
```

After each sub-band is squeezed into an array, thereafter, the Matrix Minimizationalgorithmis applied to each reduced array independently. This method reduces the array size by 66%, the calculation depends on key values and coefficients of the reduced array, and the result is stored in a new array called Minimized-Array. The following equation represents the Matrix Minimizationalgorithm **[133,134,137]**:

$$Minimzed\text{-}Array(P)=Key(1)RA(L) + Key(2)RA(L+1) + Key(3)RA(L+2) \quad (5.5)$$

Note that "*RA*" represents Reduced-Array (HL_2, LH_2, HH_2 and AC-$Matrix_1$)
L= 1, 2, 3,… N-3, " N" is the size of Reduced-Array
P= 1,2,3…N/3

The Keyvaluesin the above Eq. (5.5) are generated by a key generator algorithm. Initially compute the maximum value in reduced sub-band, and three keys are generated according to the following equations **[137]**:

$$M = 1.5 \max(H) \quad (5.6)$$

$$K_1 \geq 1 \quad (5.7)$$

$$K_2 = K_1 + M + F \quad (5.8)$$

$$K_3 = F \qquad M\,(K_1 + K_2) \quad (5.9)$$

Key=[K_1, K_2, K_3];% Final Key values for compression and decompression

Where ***M*** represents the maximum value in the high-frequency matrix*H*, Factor ≥ 1 and $K_1 \geq$ 1are integer values. Each reduced sub-band (See Figure5.3) has its own key value. This depends on the maximum value in each sub-band.The idea for the key is similar to weights used in a Perceptron Neural Network:$P=AW_1+BW_2+CW_3$, where $W_i$are the weight values generated randomly and "A","B" and "C" are data. The output of this summation is "P" and there is only one possible combination for the data values given W_i(See Section 3.2.3)**[133,134,137].**

The Matrix Minimization algorithmproducesa minimized-array that contains lots of zeros with a few nonzero data. In this case,we separate zeros from nonzero data, as shown in Figure 5.4.The zero-array can be computed easily by calculating the number of zeros between two nonzero data. For example, assume the following Minimized-Array=[0.5, 0, 0, 0, 7.3, 0, 0, 0, 0, 0, -7],the zero-array will be [0,3,0,5,0]where the zeros in red refer to nonzero data existing at these positions in the original Minimized-Array and the numbers in black refer to the number of zeros between two consecutive non-zero data. **[137]**.

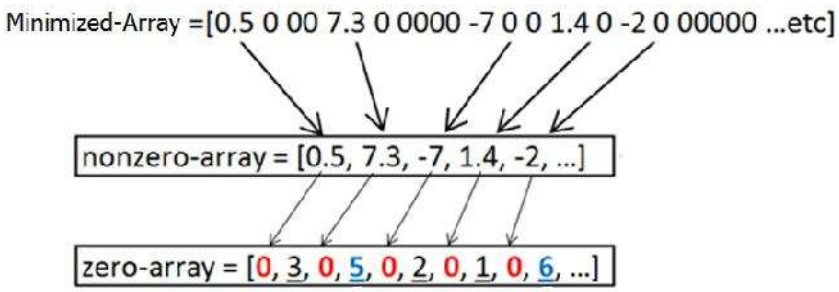


Figure 5.4: SeparateMinimized-Array into zero-array and nonzero-array

Before applying the Enhanced Matrix Minimization algorithm, our compression algorithm computes the probability of theReduced-Array (i.e. compute the probability for each HL_2, LH_2, HH_2 and AC-Martix$_1$). These probabilities are called *Limited-Data*, which is used later in the decompression stage (see Figure 3.5)

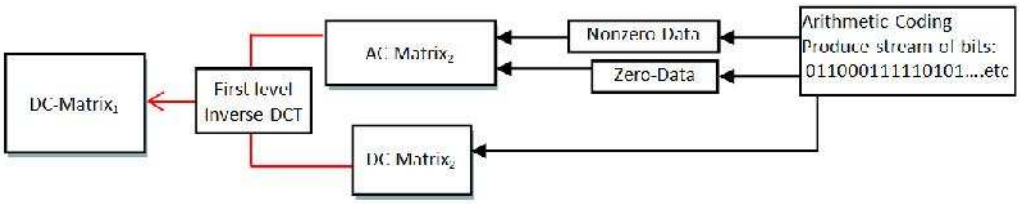
The final step of the compression algorithm is arithmetic coding, which is one of the important methods used in data compression; it takes a stream of data and convert it toa single floating point value. These values lie in the range less than one and greater than zero that, when decoded,return the exact stream of data. The arithmetic coding needs to compute the probability of all data and assign a range for each data (low and high) to generate streams of bits **[33,57,131]**.

5.3 The Fast-Matching-Search Decompression Algorithm (FMS-Algorithm)

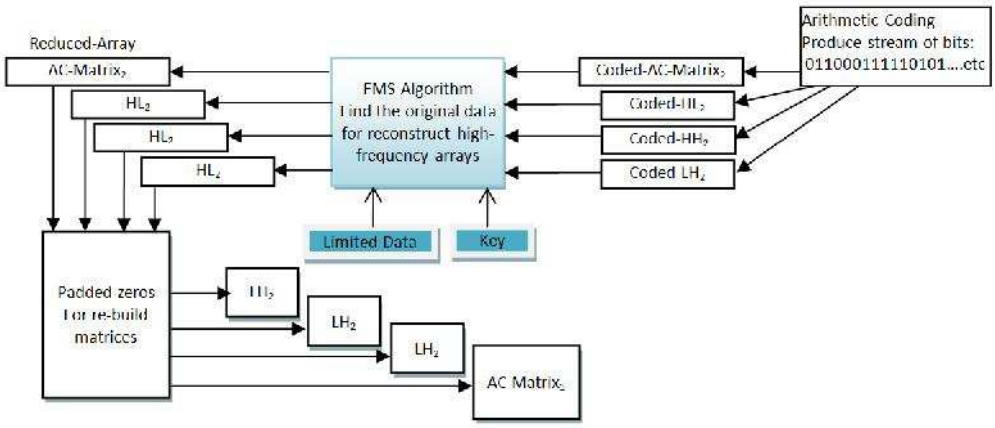
The proposed decompression algorithm is the inverse of compression and consists of three stages **[137]**:

1) First level Inverse DCTto reconstruct the DC-Matrix$_1$;
2) Apply the FMS-Algorithm to decode each sub-band independently (i.e. HL_2, LH_2, HH_2, AC-Matrix$_2$);
3) Apply the second level inverse DCT with two levels inverse DWTto reconstruct the 2D image.

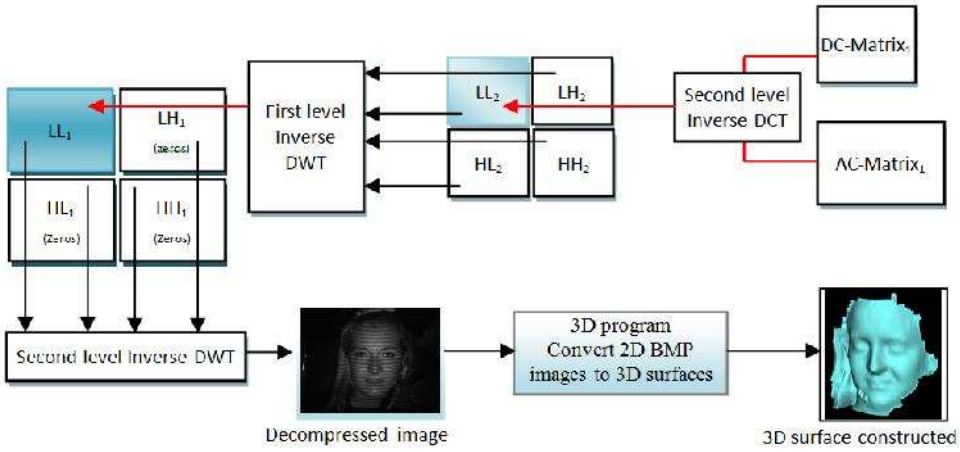
Once the 2D image is reconstructed, we apply structured light 3D reconstruction algorithms to obtain an approximation of the original 3D surface, from which errors can be computed for the entire surface. Figure5.5 shows the layout of the main steps in the proposed decompression algorithm.



(a) First level inverse DCT



(b) FMS-Algorithm applied to reconstruct high-frequency matrices



(c) Two levels inverse DWT with two levels inverse DCT applied to decompress 2D image

Figure5.5: (a),(b) and (c)represents layout of the proposed Decompression algorithm

The FMS-Algorithm has been designed to recover the original high frequency data. The compressed data contains information about the compression keys and probability data (Limited-data) followed by streams of compressed high frequency data. Therefore, the FMS algorithm picks up each compressed high frequency data and reads information (key values and Limited-Data) from which the original high frequency data are recovered. TheFMS-Algorithm is illustrated through the following steps A and B:

A) Initially, Limited-Data are copied (in memory) three times into separated arrays. This is because expanding the compressed data with the three keys resembles an interconnected data, similar to a network as shown in Figure5.6

B) Pick up a data item from"*D*", the compressed array (i.e. Coded-HL_2 or Coded-LH_2 or Coded-HH_2 or Coded-AC-$Matrix_1$) and search for the combination of (A,B,C) with respective keys that satisfy *D*. Return the triplet decompressed values (A,B,C).

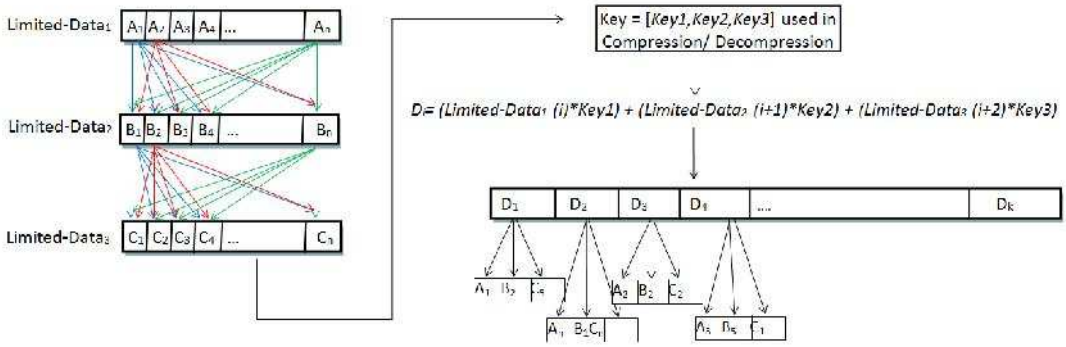


Figure5.6: First stage FMS-Algorithm for reconstructing high frequency data from Limited-Data. A, B and C are the uncompressed data which are determined by the unique combination of keys.

Since the three arrays of Limited-Data contain the same values, that is A1=B1=C1, A2=B2=C2, and so on the searching algorithm computes all possible combinations of A with Key1, B with Key2 and C with Key3 that yield a result D. As a means of an example consider that Limited-$Data_1$=[A1 A2 A3] , Limited-$Data_2$=[B1 B2 B3] and Limited-$Data_3$=[C1 C2 C3]. Then, according to Eq.(10) these represent RA(L), RA(L+1) and RA(L+2) respectively, the equation is executed 27 times (3^3=27) testing all indices and keys. One of these combinations will match the data in (D) (i.e. the original high frequency Coded-LH_2 or Coded-HL_2 or Coded-HH_2 or Coded-AC-$Matrix_1$) as described in Figure5.6. The match indicates that the unique combination of A,B,C are the orgininal data we are after[137].

The searching algorithm used in our decompression method is called *Binary Search Algorithm*, the algorithm finds the original data (A,B,C) for any inputfromarray "*D*". For the binary search, the array should be arranged in ascending order. In each step, the algorithm compares the input value with the middle of element of the array "*D*". If the value matches, then a matching element has been found and its position is returned**[139]**. Otherwise, if the search is less than the middle element of "*D*", then the algorithm repeats its action on the sub-array to the left of the middle element or, if the value is greater, on the sub-array to the right. There is no probability for "*Not Matched*", because the FMS-Algorithm computed all compression data possibilities previously.

After the Reduced-Arrays (LH_2, LH_2, HH_2 and AC-$Matrix_1$) are recovered, their full corresponding high frequency matrices are re-build by placing nonzero-data in the exact locations according to EZSN algorithm (see List 5.1). Then the sub-band $LL_2$is reconstructed by combining the DC-$Matrix_1$and AC-$Matrix_1$ followed bythe inverse DCT. Finally, a two-

levelinverseDWTis applied to recover the original 2D BMP image (see Figure5.5(c)). Once the 2D image is decompressed, its3D geometric surface is reconstructed such that error analysis can be performed in this dimension [137].

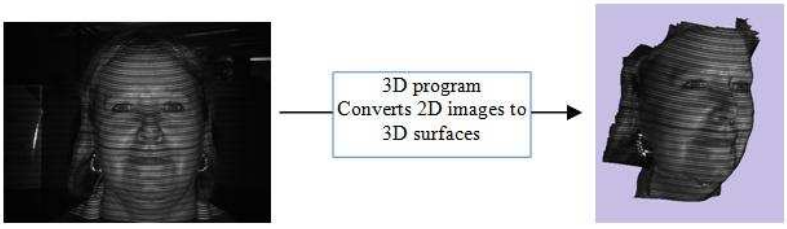
5.4 Experimental Results

The results described below use MATLAB-R2013a for performing2D image compression and decompression, and 3D surface reconstruction was performed with our own software developed within the GMPR group (See Section 3.4)running on an AMD quad-core microprocessor. The justification for introducing 3D reconstruction is that we can make use of a new set of metrics in terms of error measurements and perceived quality of the 3D visualization to assess the quality of the compression/decompression algorithms **[135, 136]**.

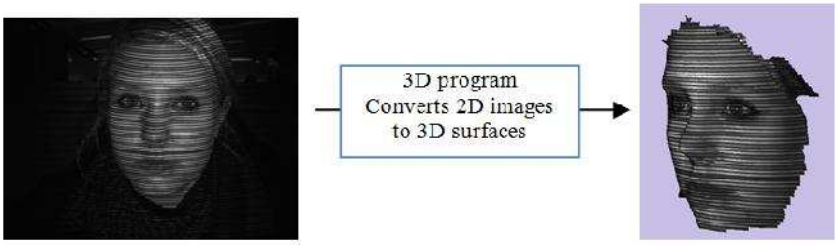
The results in this chapter are divided into two parts: first, we apply the proposed image compression and decompression methods to 2D greyscale images of human faces. The original and decompressed images are used to generate 3D surface models from which direct comparisons are made in terms of perceived quality of the mesh and objective error measures such as RMSE. Second, we repeat all procedures for 2Dcompression, 2D decompression, 3D reconstruction but this time with colour images of objects other than faces. Additionally, the computed 2D and 3D RMSEare used directly for comparison with JPEG and JPEG2000 techniques.

5.4.1 Compression, Decompression and 3D Reconstruction fromGreyscaleImages

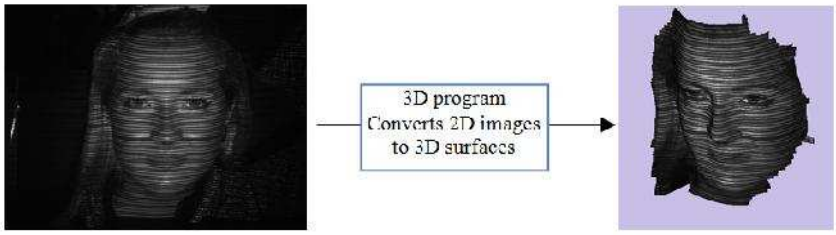
As described above the proposed image compression started with DWT. The level of DWT decomposition affects the image quality also the compression ratio, so we divided the results into two parts to show the effects of each independently: single level DWTand two-level DWT. Figure 5.7 shows the original 2Dhuman faces tested by the proposed algorithm. Table5.1, Table5.2 and Table5.3 show the compressed size by using our algorithm with a single level and two-levelDWTforFace1, Face2 and Face3 respectively.



(a) Original 2D Face1dimensions 1392 x 1040, Image size: 1.38 Mbytes, converted to 3D surface



(b) Original 2D Face2dimensions 1392 x 1040, Image size: 1.38 Mbytes, converted to 3D surface



(c) Original 2D Face3dimensions 1392 x 1040, Image size: 1.38 Mbytes, converted to 3D surface

Figure 5.7:(a), (b) and (c) original 2D images (left) with 3D surface reconstruction (right).

Table5.1: Compressed size for 2D image Face1

Applied single level DWT						
Single level DWT High-frequencies (HL,LH and HH)			DCT parameters		Compressed size (KB)	Compression Ratio
			Block size	Scale		
Discarded			8×8	0.5	51.4	96.3%
Discarded			8×8	1	29.33	97.9%
Discarded			8×8	2	15.6	98.8%
Discarded			8×8	3	10.58	99.2%
Discarded			8×8	4	8	99.4%
Discarded			8×8	5	6.37	99.5%
Discarded			16×16	0.5	28.52	97.9%
Discarded			16×16	1	14.74	98.9%
Discarded			16×16	2	7.38	99.4%
Applied twolevelsDWT						
Two levels DWTHigh-frequencies Ratio value (Eq.(5))			DCT parameters		Compressed size (KB)	Compression Ratio
			Block size	Scale		
LH_2	HL_2	HH_2				
0.3	0.3	0.3	29.93	0.5	29.93	97.8%
0.3	0.3	0.3	17.55	1	17.55	98.7%
0.3	0.3	0.3	9.7	2	9.7	99.3%
0.3	0.3	0.3	6.74	3	6.74	99.5%
0.3	0.3	0.3	21	0.5	21	98.5%
0.3	0.3	0.3	10.54	1	10.54	99.2%
0.3	0.3	0.3	5.19	2	5.19	99.6%

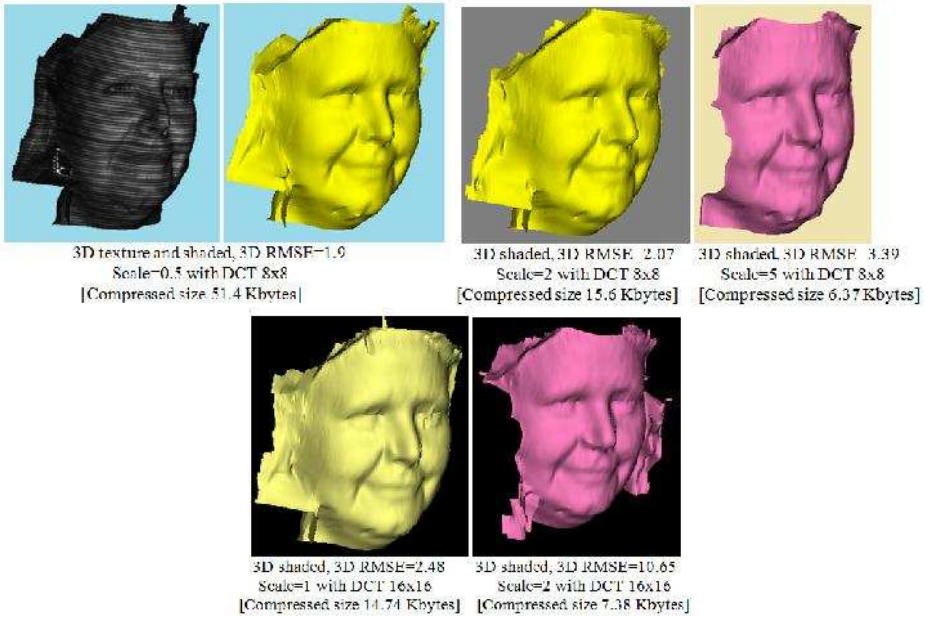
Table 5.2: Compressed size for 2D image Face2

Applied single level DWT					
Single level DWT High-frequencies (HL, LH and HH)			DCT parameters		Compressed size (KB)	Compression Ratio
			Block size	Scale		
Discarded			8×8	0.5	47	96.6%
Discarded			8×8	1	26.85	98%
Discarded			8×8	2	14.42	98.9%
Discarded			8×8	3	9.91	99.2%
Discarded			8×8	4	7.4	99.4%
Discarded			16×16	0.5	25.33	98.2%
Discarded			16×16	1	13.3	99%
Discarded			16×16	2	6.77	99.5%
Applied Two levels DWT						
TwolevelsDWT High-frequencies Ratio value (Eq.(5))			DCT parameters		Compressed size (KB)	Compression Ratio
			Block size	Scale		
LH_2	HL_2	HH_2				
0.3	0.3	0.3	8×8	0.5	35	97.5%
0.3	0.3	0.3	8×8	1	19.34	98.6%
0.3	0.3	0.3	8×8	2	9.9	99.2%
0.3	0.3	0.3	8×8	3	6.41	99.5%
0.3	0.3	0.3	8×8	4	4.66	99.6%
0.3	0.3	0.3	16×16	0.5	26.3	98.1%
0.3	0.3	0.3	16×16	1	12.96	99%
0.3	0.3	0.3	16×16	2	5.76	99.5%

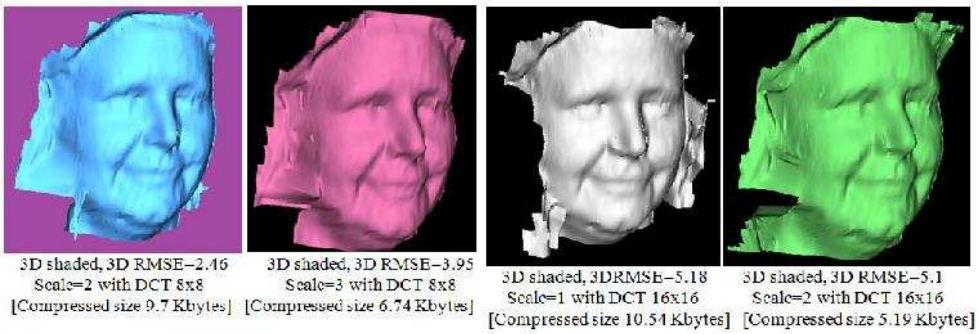
Table 5.3: Compressed size for 2D image Face3

Applied single level DWT						
Single level DWT High-frequencies (HL, LH and HH)			DCT parameters		Compressed size (KB)	Compression Ratio
			Block size	Scale		
Discarded			8×8	0.5	49.53	96.4%
Discarded			8×8	1	27.5	98%
Discarded			8×8	2	14.31	98.9%
Discarded			8×8	3	9.45	99.3%
Discarded			8×8	4	7.13	99.4%
Discarded			16×16	0.5	26.83	98.1%
Discarded			16×16	1	13.53	99%
Discarded			16×16	2	6.52	99.5%
Applied Two levels DWT						
Two levels DWT High-frequencies Ratio value Eq.(5)			DCT parameters		Compressed size (KB)	Compression Ratio
			Block size	Scale		
LH_2	HL_2	HH_2				
0.3	0.3	0.3	8×8	0.5	35.78	97.4%
0.3	0.3	0.3	8×8	1	19.6	98.6%
0.3	0.3	0.3	8×8	2	9.63	99.3%
0.3	0.3	0.3	8×8	3	6.2	99.5%
0.3	0.3	0.3	16×16	0.5	25.78	98.1%
0.3	0.3	0.3	16×16	1	12.4	99.1%
0.3	0.3	0.3	16×16	1.7	6.45	99.5%

The proposed decompression algorithm (see Section 5.3) applied to the compressed image data recovers the 2D images which are then used by the 3D reconstruction to generate the respective 3D surface. The following figures Figure5.9, Figure5.10 and Figure5.11 show high-quality, median-quality and low-quality compressed images for; Face1, Face2 and Face3. Also, Table5.4, Table5.5 and Table5.6 show the time execution for the FMS-Algorithm.



(a) Decompressed 2D BMP images at Single level DWT converted to 3D surface; 3D surface with scale=0.5 represents high quality image comparable to the original image, and 3D surface with scale=2 represents median quality image approximately high quality image. Also 3D surface with scale=5 is low quality image some parts of surface failing to reconstruct. Additionally, using a block size of16×16 DCTfurther degrades the 3D surface.



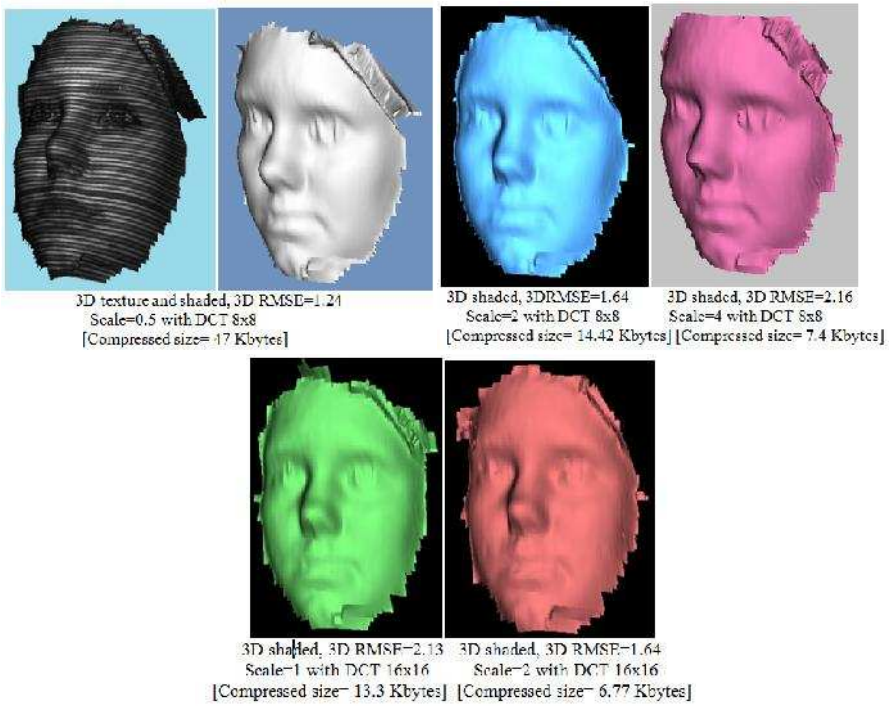
(b) Decompressed 2D BMP images at two levels DWT converted to 3D surface; 3D surface with scale=2, 3 and DCT 8x8 represent low quality image surface with degradation. Additionally, using a block size of 16x16 DCTfurther degrades the 3D surface.

Figure5.9: (a) and (b) decompressed 2D image Face1 by our proposed decompression method, and then converted to a 3D surface.

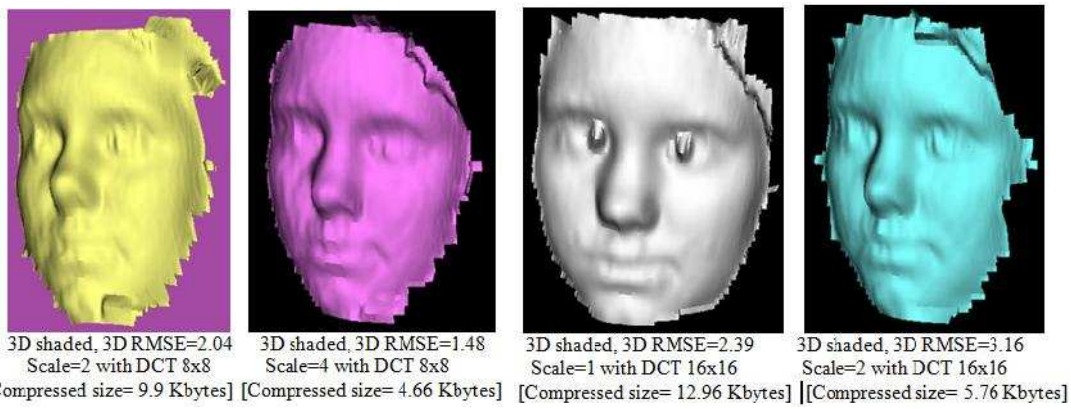
Table 5.4:FMS time execution for Decompressed image Face1

Applied single level DWT with two levels DCT							
DCT		Decompressed RMSE		FMS-Algorithm Time execution (s)			
Block	Scale	2D RMSE	3D RMSE	AC-Matrix$_1$	LH	HL	HH
8×8	0.5	4.42	1.9	4.25	Discarded	Discarded	Discarded
8×8	1	4.83	2.31	1.31	Discarded	Discarded	Discarded
8×8	2	5.58	2.07	0.23	Discarded	Discarded	Discarded
8×8	3	6.21	2.4	0.1	Discarded	Discarded	Discarded
8×8	4	6.87	3.34	0.046	Discarded	Discarded	Discarded
8×8	5	7.41	3.39	0.046	Discarded	Discarded	Discarded
16×16	0.5	6.0	3.96	2.24	Discarded	Discarded	Discarded
16×16	1	6.63	2.48	0.23	Discarded	Discarded	Discarded

Applied Two levels DWT with two levels DCT											
DCT		Scale Used in DCT and second level DWT			Decompressed RMSE		FMS-Algorithm Time execution (s)				
Block	Scale	LH_2	HL_2	HH_2	2D RMSE	3D RMSE	AC-Matrix$_1$	LH_2	HL_2	HH_2	
8×8	0.5	0.3	0.3	0.3	6.79	2.19	9.1	0.031	≈ 0		
8×8	1	0.3	0.3	0.3	7.5	4.77	1.75	≈ 0	0.031		
8×8	2	0.3	0.3	0.3	8.1	2.46	0.24	≈ 0	≈ 0		
8×8	3	0.3	0.3	0.3	8.46	3.95	0.1	≈ 0	≈ 0		
16×16	0.5	0.3	0.3	0.3	8.1	3.69	2.82	0.031	≈ 0		
16×16	1	0.3	0.3	0.3	8.89	5.18	0.65	≈ 0	≈ 0		
16×16	2	0.3	0.3	0.3	9.62	5.1	0.18	≈ 0	≈ 0		



(**a**) Decompressed 2D BMP images at Single level DWT converted to 3D surface; 3D surface with scale=0.5 represents high quality image like the original image, and 3D surface with scale=2 represents a median quality image. Also, 3D surface with scale=4 is low quality withsurface slightlydegraded. Additionally, a block size of 16x16 DCT used in our approach degrades some parts of 3D surface.



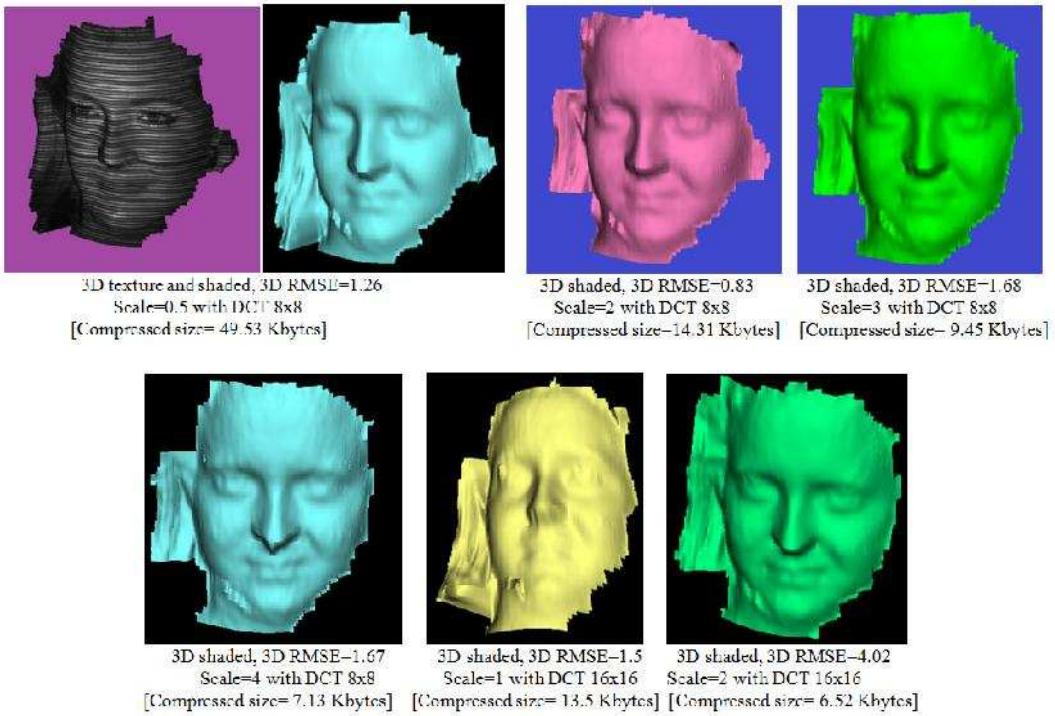
(**b**) Decompressed 2D BMP images at two levels DWT converted to 3D surface; 3D surface with scale=2, 4 withDCT 8x8 represent low quality image surface with degradation. Similarly, a block size of 16x16 DCT used in our approach degrades the 3D surface.

Figure 5.10: (a) and (b) Decompressed 2D image of Face2 image by our proposed decompression method, and then converted to 3D surface.

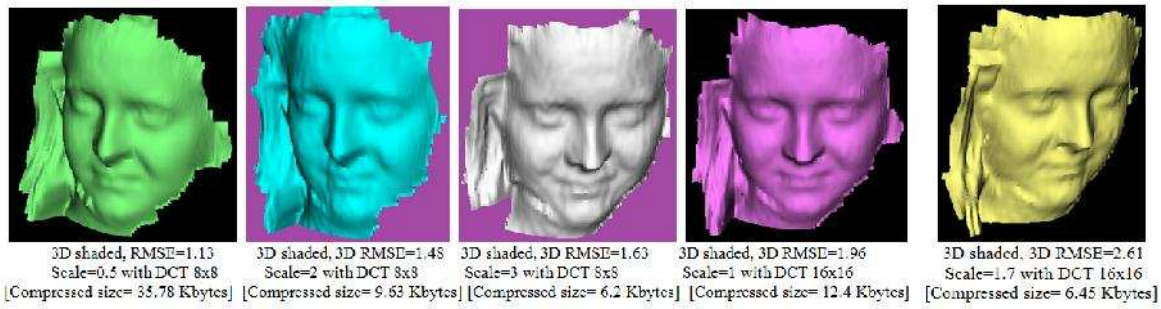
Table 5.5: FMS time execution for Decompressed image Face2

Applied single level DWT with two levels DCT							
DCT		Decompressed RMSE		FMS-Algorithm Time execution (s)			
Block	Scale	2D RMSE	3D RMSE	AC-Matrix$_1$	LH	HL	HH
8x8	0.5	4.88	1.24	7 .06	Discarded	Discarded	Discarded
8x8	1	5.22	2.19	1.09	Discarded	Discarded	Discarded
8x8	2	5.86	1.64	0.208	Discarded	Discarded	Discarded
8x8	3	6.46	2.55	0.109	Discarded	Discarded	Discarded
8x8	4	6.96	2.16	0.046	Discarded	Discarded	Discarded
16x16	0.5	5.57	1.7	3.4	Discarded	Discarded	Discarded
16x16	1	6.09	2.13	0.56	Discarded	Discarded	Discarded
16x16	2	6.94	1.63	0.093	Discarded	Discarded	Discarded

Applied Two levels DWT with two levels DCT										
DCT		Scale Used in DCTand second level DWT			Decompressed RMSE		FMS-Algorithm Time execution (s)			
Block	Scale	LH_2	HL_2	HH_2	2D RMSE	3D RMSE	AC-Matrix$_1$	LH_2	HL_2	HH_2
8x8	0.5	0.3	0.3	0.3	5.16	1.33	13.15	0.046	0.046	0.031
8x8	1	0.3	0.3	0.3	5.76	2.07	2.1	0.031	≈ 0	≈ 0
8x8	2	0.3	0.3	0.3	7.04	2.04	0.35	0.062	≈ 0	≈ 0
8x8	3	0.3	0.3	0.3	7.91	1.39	0.109	≈ 0	≈ 0	≈ 0
8x8	4	0.3	0.3	0.3	8.43	1.48	0.062	≈ 0	≈ 0	0.031
16x16	0.5	0.3	0.3	0.3	5.73	2.08	2.96	0.093	0.031	0.031
16x16	1	0.3	0.3	0.3	6.44	2.39	0.59	0.031	0.031	≈ 0
16x16	2	0.3	0.3	0.3	7.85	3.16	0.124	0.015	≈ 0	≈ 0



(a) Decompressed 2D BMP images at Single level DWT converted to 3D surface; 3D surface with scale=0.5 represent high quality image like the original image, and 3D surface with scale=2 represents median quality image, 3D surface with scale=3 and 4 are low quality image with slightly degraded surface, while using 16x16 block size does not seem to degrade the 3D surface.



(b) Decompressed 2D BMP images at two levels DWT converted to 3D surface; 3D surface with scale=3, 4 and DCT 8x8 represent low quality image degraded surface. However, the block size of 16x16 DCT used in our approach has a better quality 3D surface for higher compression ratios.

Figure5.11: (a) and (b): Decompressed 2D of Face3 image by our proposed decompression method, and then converted to a 3D surface.

Table 5.6:FMS time execution for Decompressed image Face3

Applied single level DWT with two levels DCT							
DCT		Decompressed RMSE		FMS-Algorithm Time execution (s)			
Block	Scale	2D RMSE	3D RMSE	AC-Matrix$_1$	LH	HL	HH
8x8	0.5	4,29	1.26	5.91	Discarded	Discarded	Discarded
8x8	1	4.7	1.32	1.1	Discarded	Discarded	Discarded
8x8	2	5.43	0.83	0.171	Discarded	Discarded	Discarded
8x8	3	6.09	1.68	0.093	Discarded	Discarded	Discarded

8x8	4	6.67	1.67	0.046	Discarded	Discarded	Discarded
16x16	0.5	1.94	1.29	3.12	Discarded	Discarded	Discarded
16x16	1	1.52	1.5	0.59	Discarded	Discarded	Discarded
16x16	2	6.65	4.02	0.1	Discarded	Discarded	Discarded

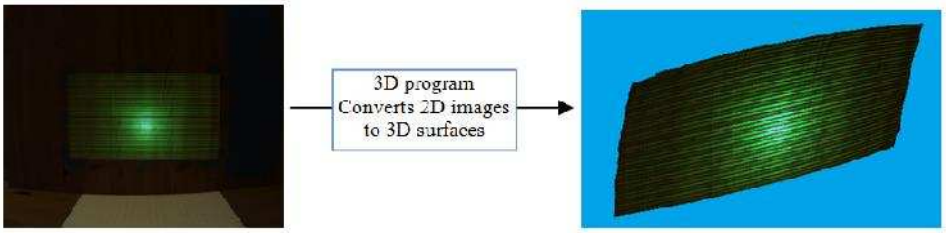
Applied Two levels DWT with two levels DCT

DCT		Scale Used in DCT and second level DWT			Decompressed RMSE		FMS-Algorithm Time execution (s)			
Block	Scale	LH_2	HL_2	HH_2	2D RMSE	3D RMSE	AC-Matrix$_1$	LH_2	HL_2	HH_2
8x8	0.5	0.3	0.3	0.3	4.81	1.13	12.00	0.078	0.046	≈ 0
8x8	1	0.3	0.3	0.3	5.53	1.83	1.96	0.015	≈ 0	≈ 0
8x8	2	0.3	0.3	0.3	6.7	1.48	0.28	0.031	≈ 0	≈ 0
8x8	3	0.3	0.3	0.3	7.47	1.63	0.078	≈ 0	0.031	≈ 0
16x16	0.5	0.3	0.3	0.3	5.58	2.08	4.69	0.062	0.015	0.031
16x16	1	0.3	0.3	0.3	6.42	1.96	0.98	0.046	≈ 0	≈ 0
16x16	1.7	0.3	0.3	0.3	7.38	2.61	0.171	≈ 0	≈ 0	≈ 0

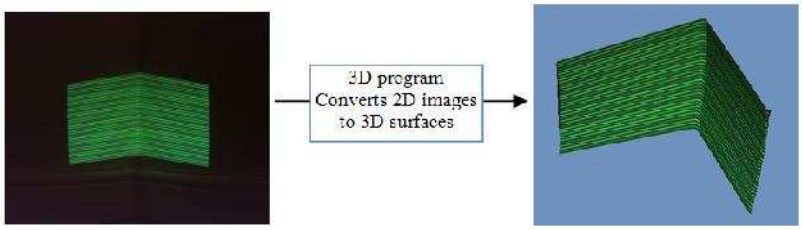
It is shown through the pictures and tables above that the proposed compression algorithm is successfully applied to greyscale images. Table 5.1, Table5.2 and Table 5.3 show a compression of more than 99% of the original image size compressed and the reconstructed 3D surfaces still preserve most of their quality. Some images are compressed by DCT with block size of 16x16 are also shown capable of generating high quality 3D surface. Also, there is not much difference between block sizes of 8x8 and 16x16 for high quality reconstruction images with “Scale=0.5”.

5.4.2 Compression, Decompression and 3D Reconstruction from Colour Images

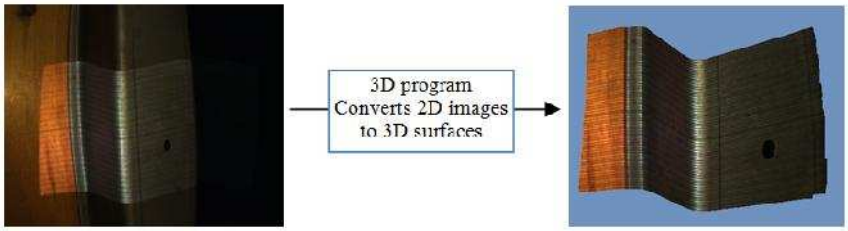
Colour images contain red, green and blue layers. In JPEG and JPEG2000 colour layers are transformed to “YCbCr” layers before compression. This is because most of information about images available in layer “Y” while other layers “CrCb” contain less information **[118, 140]**. The proposed image compression was tested with YCbCr layers, and then applied on true colour layers (Red, Green and Blue). Figure5.12 shows the original colour images tested by our approach.



(**a**) Original 2D “Wall "dimensions 1280 x 1024, Image size: 3.75 Mbytes, converted to 3D surface



(**b**) Original 2D "Room"dimensions 1280 x 1024, Image size: 3.75 Mbytes, converted to 3D surface



(**c**) Original 2D "Corner"dimensions 1280 x 1024, Image size: 3.75 Mbytes, converted to 3D surface

Figure5.12: (a), (b) and (c) Original 2D images with 3D surface conversion

First theWall,Roomand Corner images as depicted in above Figure5.12 were transformed toYcbCrbefore applying our proposed image compression– both using single level and two-level DWT decomposition. Second, our approach was applied on the same colour images but this time using true colour layers.Table5.7, 5.8 and 5.9 showsthecompressed size for the colour images by the proposed compression algorithm.Figure5.14, 5.15 and 5.16 show decompressed colour images (Wall, Room and Corner respectively) as 3D surface. Additionally, Table5.10, 5.11 and 5.12illustrate theexecution time for the FMS-Algorithmat single level DWT for the colour images.Similarly, Table5.13, 5.14 and 5.15 show theFMS-Algorithm execution time forthe same colour images by using two-level DWT.

Table5.7:Compressed sizes for 2D colour image Wall

Applied single level DWT with two levels DCT						
Single level DWT High-frequencies(HL, LH and HH) For all colour layer	DCT parameters				Compressed size (KB)	Compression Ratio
	Block size	Scale				
		Y	Cb	Cr		
Ignored	8x8	0.5	1	1	26.3	99.3%
Ignored	8x8	1	2	2	14.23	99.6%
Ignored	8x8	2	4	4	7,53	99.8%
Ignored	16x16	0.5	1	1	12.3	99.6%
Ignored	16x16	1	2	2	6.55	99.8%
Single level DWT High-Frequencies	Block size	Red	Green	Blue	Compressed Size (KB)	Compression Ratio
Ignored	8x8	1	1	1	24.42	99.3%
Ignored	8x8	3	3	3	8.47	99.7%
Ignored	8x8	5	5	5	5.46	99.8%
Ignored	16x16	1	1	1	11.47	99.7%
Ignored	16x16	2	2	2	5.9	99.8%
Ignored	16x16	2.5	2.5	2.5	4.85	99.8%

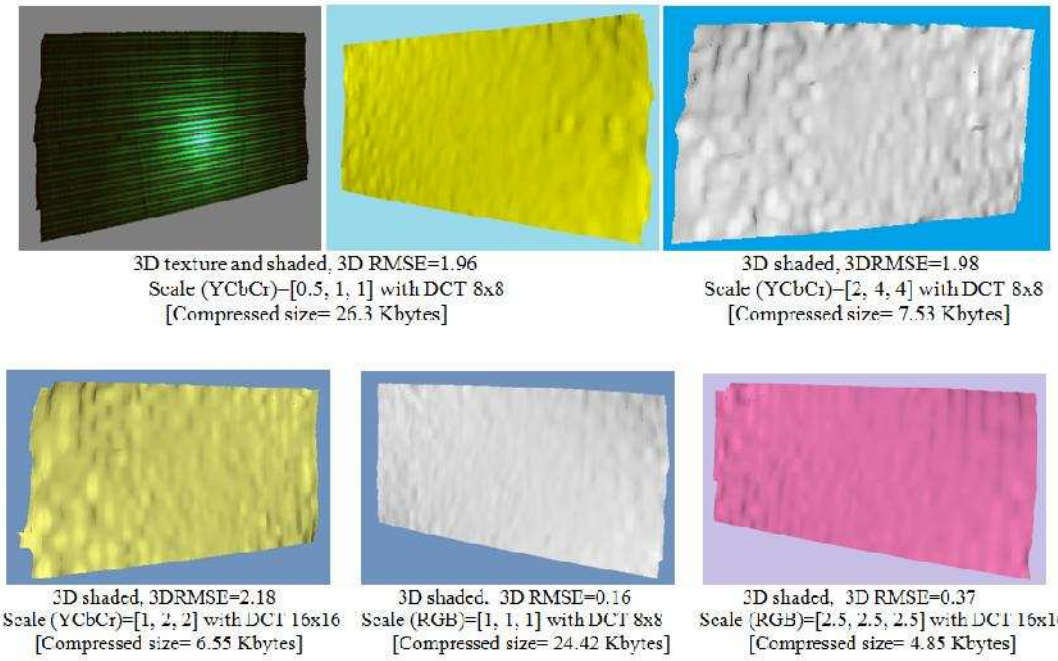
Applied Two levels DWT with two levels DCT								
Two levels DWT High-frequencies Ratio value (Eq.(5)) for all colour layers			DCT parameters				Compressed size(KB)	Compression Ratio
			Block size	Scale				
LH_2	HL_2	HH_2		Y	Cb	Cr		
0.3	0.3	0.3	8x8	0.5	1	1	28.1	99.2%
0.3	0.3	0.3	8x8	1	2	2	11.8	99.6%
0.3	0.3	0.3	16x16	0.5	1	1	21.8	99.4%
0.3	0.3	0.3	16x16	1	2	2	8.29	99.7%
Two levels DWT High-Frequencies			Block size	Red	Green	Blue	Compressed Size (KB)	Compression Ratio
0.3	0.3	0.3	8x8	1	1	1	19.32	99.4%
0.3	0.3	0.3	8x8	2	2	2	8.4	99.7%
0.3	0.3	0.3	16x16	1	1	1	12.89	99.6%
0.3	0.3	0.3	16x16	2	2	2	5	99.8%

Table5.8:Compressed size for 2D colour image Room

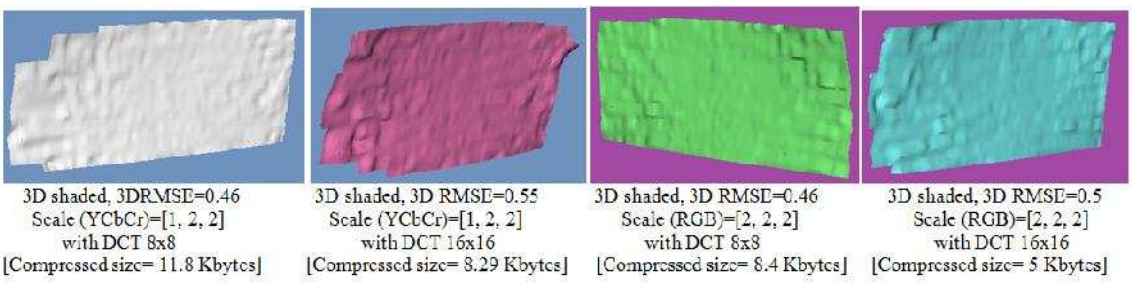
Applied single level DWT with two levels DCT						
Single level DWT High-frequencies(HL, LH and HH) For all colour layer	DCT parameters				Compressed size (KB)	Compression Ratio
	Block size	Scale				
		Y	Cb	Cr		
Ignored	8x8	0.5	1	1	58.21	98.4%
Ignored	8x8	1	2	2	34	99.1%
Ignored	8x8	2	4	4	18.92	99.5%
Ignored	16x16	0.5	1	1	29.86	99.2%
Ignored	16x16	1	2	2	16.19	99.5%
Ignored	16x16	2	4	4	8.07	99.7%
Single level DWT High-Frequencies	Block size	Red	Green	Blue	Compressed Size (KB)	Compression Ratio
Ignored	8x8	1	1	1	53.25	98.6%
Ignored	8x8	3	3	3	17.66	99.5%
Ignored	8x8	5	5	5	10.73	99.7%
Ignored	8x8	9	9	9	6.28	99.8%
Ignored	16x16	1	1	1	23.83	99.3%
Ignored	16x16	3	3	3	7.91	99.7%
Ignored	16x16	5	5	5	4.65	99.8%
Ignored	16x16	7	7	7	3.38	99.9%
Applied Two levels DWT with two levels DCT						
Two levels DWT High-frequencies Ratio value (Eq.(5)) for all colour layers	DCT parameters				Compressed size (KB)	Compression Ratio
	Block size	Scale				
LH_2 HL_2 HH_2		Y	Cb	Cr		
0.3 0.3 0.3	8x8	0.5	1	1	47.25	98.7%
0.3 0.3 0.3	8x8	1	2	2	23.96	99.3%
0.3 0.3 0.3	8x8	2	4	4	11.85	99.6%
0.3 0.3 0.3	16x16	0.5	1	1	35.94	99%
0.3 0.3 0.3	16x16	1	2	2	17.62	99.5%
0.3 0.3 0.3	16x16	2	4	4	7.94	99.7%
Two levels DWT High-Frequencies	Block size	Red	Green	Blue	Compressed Size (KB)	Compression Ratio
0.3 0.3 0.3	8x8	1	1	1	49	98.7%
0.3 0.3 0.3	8x8	3	3	3	10.42	99.7%
0.3 0.3 0.3	8x8	5	5	5	5.12	99.8%
0.3 0.3 0.3	16x16	1	1	1	36.34	99%
0.3 0.3 0.3	16x16	3	3	3	6.61	99.8%
0.3 0.3 0.3	16x16	5	5	5	2.93	99.9%

Table5.9: Compressed size for 2D colour image Corner

Applied single level DWT with two levels DCT							
Single level DWT High-frequencies(HL, LH and HH) For all colour layer			DCT parameters			Compressed size (KB)	Compression Ratio
			Block size	Scale			
				Y	Cb	Cr	
Ignored			8x8	0.5	1	1	31.59	99.1%
Ignored			8x8	1	2	2	15.85	99.5%
Ignored			16x16	0.5	1	1	15.39	99.5%
Ignored			16x16	0.8	6	6	7.8	99.7%
Single level DWT High-Frequencies			Block size	Red	Green	Blue	Compressed Size (KB)	Compression Ratio
Ignored			8x8	1	1	1	44.5	98.8%
Ignored			8x8	2	2	2	22	99.4%
Ignored			16x16	1	1	1	21.16	99.4%
Ignored			16x16	2	2	2	10	99.7%
Applied Two levels DWT with two levels DCT							
Two levels DWT High-frequencies Ratio value (Eq.(5)) for all colour layers			DCT parameters				Compressed size (KB)	Compression Ratio
			Block size	Scale				
LH_2	HL_2	HH_2		Y	Cb	Cr		
0.3	0.3	0.3	8x8	0.5	1	1	31.72	99.1%
0.3	0.3	0.3	8x8	1	2	2	13.11	99.6%
0.3	0.3	0.3	16x16	0.5	1	1	23.14	99.3%

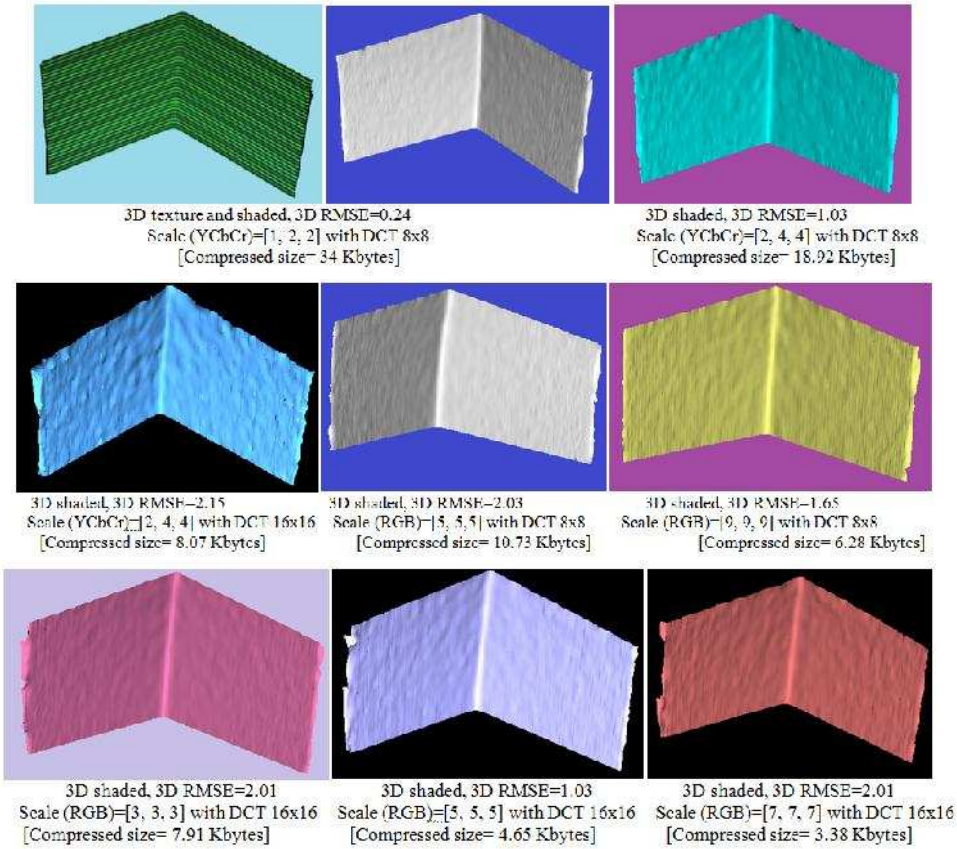


(**a**) Decompressed 2D BMP images at single level DWT converted to 3D surface; decompressed 3D surface by using RGB layer has better quality than YCbCr layer.

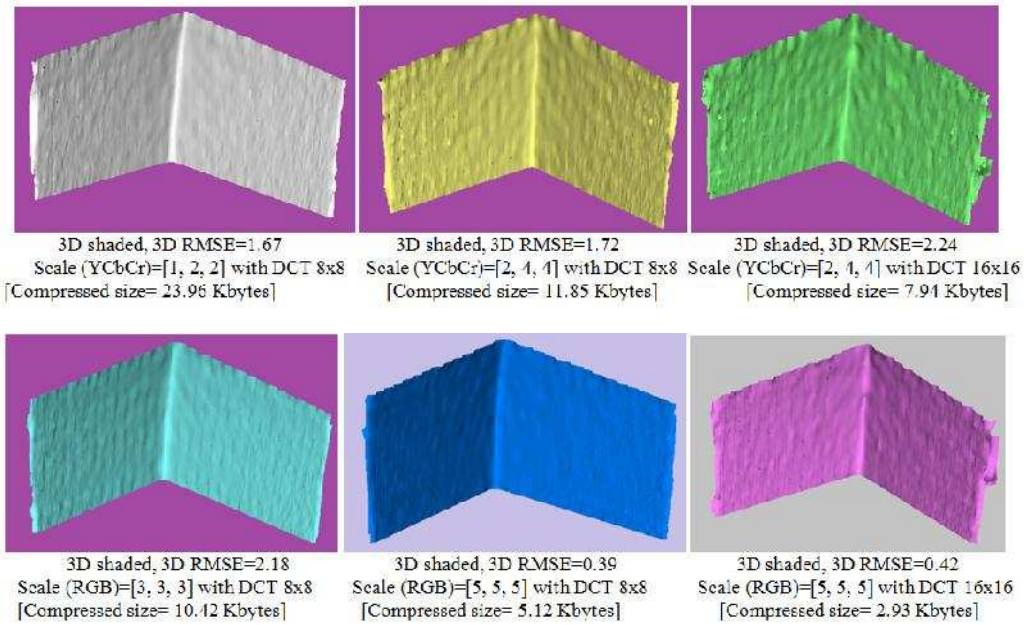


(b) Decompressed 2D BMP images at two levels DWT converted to 3D surface, also decompressed 3D surface using RGB layer has better quality than YCbCr layer at higher compression ratio.

Figure5.14: (a) and (b) Decompressed colour Wall image by our proposed decompression approach, and then converted to 3D surface



(a) Decompressed 2D BMP images at Single level DWT results converted to 3D surface; decompressed 3D surface by using RGB layer has better quality than YCbCr layer at higher compression ratio using both block sizes of 8x8 or 16x16 by DCT.



(b) Decompressed 2D BMP images at two levelsDWTconverted to 3D surface; decompressed 3D surface by using RGB layer has better quality than YCbCr layer at higher compression ratio using block sizes of 8x8, also by using block 16x16 the surface is still approximately non-degraded.

Figure5.15: (a) and (b) Decompressed colour Roomimageby our proposed decompression method, and then converted to 3D surface

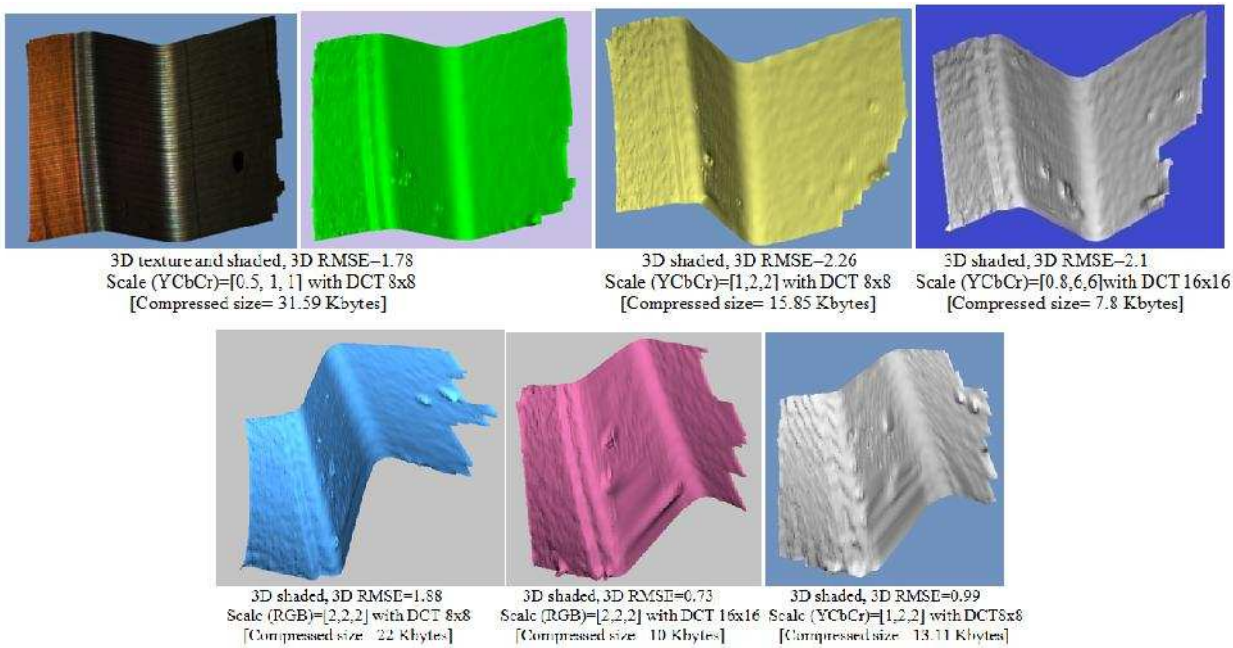


Figure5.16: Decompressed colour "Corner"image by our proposed decompression method, and then converted to 3D surface. The decompressed 3D surface at single level DWT using YCbCr has better quality at higher compression ratio than using RGBlayers, also at two levels DWT degradation appears and some parts from surface fail to reconstruct.

Table5.10:Estimatedexecution time for FMS-Algorithm at single level DWTforWall image

Compressed size (KB)	Block size	Time execution (s)		
		Y	Cb	Cr
		AC-Matrix$_1$	AC-Matrix$_1$	AC-Matrix$_1$
26.3	8x8	1.77	0.093	0.187
14.23	8x8	0.202	0.031	0.078
7.53	8x8	0.062	≈0	0.031
12.3	16x16	0.54	0.015	0.031
6.55	16x16	0.109	≈0	0.031
Compressed size (KB)	Block size	Red	Green	Blue
		AC-Matrix$_1$	AC-Matrix$_1$	AC-Matrix$_1$
24.42	8x8	0.12	1.27	0.12
8.47	8x8	≈0	0.078	0.031
5.46	8x8	≈0	0.046	≈0
11.47	16x16	0.078	0.46	0.093
5.9	16x16	0.015	0.078	0.031
4.85	16x16	0.031	0.031	≈0

Table 5.11:Estimatedexecution time for FMS-Algorithm at single level DWT for Roomimage

Compressed size (KB)	Block size	Time execution (s)		
		Y	Cb	Cr
		AC-Matrix	AC-Matrix	AC-Matrix
58.21	8x8	1.2	0.093	0.17
34	8x8	0.17	0.046	0.046
18.92	8x8	0.046	0.046	0.031
29.86	16x16	0.96	0.062	0.093
16.19	16x16	0.156	0.046	0.046
8.07	16x16	0.046	0.031	≈0
Compressed size (KB)	Block size	Red	Green	Blue
		AC-Matrix	AC-Matrix	AC-Matrix
53.25	8x8	0.062	1.4	0.21
17.66	8x8	≈0	0.124	0.015
10.73	8x8	≈0	0.062	≈0
6.28	8x8	≈0	0.031	≈0
23.83	16x16	0.046	0.68	0.078
7.91	16x16	≈0	0.062	0.031
4.65	16x16	≈0	0.046	≈0
3.38	16x16	≈0	0.031	≈0

Table 5.12:Estimatedexecution time for FMS-Algorithm at single level DWT for Corner image

Compressed size (KB)	Block size	Time execution (s)		
		Y	Cb	Cr
		AC-Matrix	AC-Matrix	AC-Matrix
31.59	8x8	1.2	0.031	0.015
15.85	8x8	0.3	0.031	0.031
15.39	16x16	1.07	0.031	0.046
7.8	16x16	0.5	≈0	≈0
Compressed size (KB)	Block size	Red	Green	Blue
		AC-Matrix	AC-Matrix	AC-Matrix
44.5	8x8	0.34	0.48	0.34
22	8x8	0.06	0.09	0.09
21.16	16x16	0.46	0.48	0.37
10	16x16	0.1	0.14	0.1

Table5.13: Estimated execution time for FMS-Algorithm at two levels DWT for Wall image

Compressed size (KB)	Block size	Y				Cb				Cr			
		AC-Matrix$_1$	HL_2	LH_2	HH_2	AC-Matrix$_1$	LH_2	HL_2	HH_2	AC-Matrix$_1$	HL_2	LH_2	HH_2
28.1	8x8	3.1	0.062	0.062	0.031	0.062	0.03	≈0	≈0	0.24	0.031	≈0	≈0
11.8	8x8	0.6	0.015	≈0	≈0	0.046	0.031	≈0	≈0	0.046	0.031	≈0	≈0
21.8	16x16	1.5	0.031	0.031	≈0	0.046	0.03	0.031	≈0	0.093	0.031	≈0	≈0
8.29	16x16	0.026	0.015	≈0	0.031	0.031	0.03	≈0	≈0	0.062	0.015	≈0	≈0
Compressed size (KB)	Block size	Red				Green				Blue			
		AC-Matrix$_1$	HL_2	LH_2	HH_2	AC-Matrix$_1$	HL_2	LH_2	HH_2	AC-Matrix$_1$	HL_2	LH_2	HH_2
19.32	8x8	0.23	0	≈0	≈0	1.5	0.031	0.031	≈0	0.26	≈0	≈0	≈0
8.4	8x8	0.062	0.031	≈0	≈0	0.24	≈0	≈0	≈0	0.078	≈0	0.031	≈0
12.89	16x16	0.21	0.078	≈0	0.031	0.081	≈0	≈0	≈0	0.17	≈0	0	≈0
5	16x16	0.062	≈0	≈0	≈0	0.093	0.031	0.031	≈0	0.062	≈0	≈0	≈0

Table5.14:Estimatedexecution time for FMS-Algorithm at two levels DWT for Room image

Compressed size (KB)	Block size	Y				Cb				Cr			
		AC-Matrix$_1$	HL_2	LH_2	HH_2	AC-Matrix$_1$	HL_2	LH_2	HH_2	AC-Matrix$_1$	HL_2	LH_2	LH_2
47.25	8x8	0.15	0.046	≈0	≈0	0.1	0.031	0.031	≈0	2.19	0.078	0.062	0.062
23.96	8x8	0.046	0.015	≈0	≈0	0.015	0.031	≈0	≈0	0.43	0.078	0.031	0.031
11.85	8x8	≈0	≈0	≈0	≈0	≈0	0.031	≈0	≈0	0.078	0.015	≈0	≈0
35.94	16x16	0.124	0.031	≈0	≈0	0.078	0.015	0.031	≈0	1.21	0.062	0.015	0.015
17.62	16x16	0.015	0.031	≈0	≈0	0.031	0.031	≈0	≈0	0.28	0.031	0.031	0.031
7.94	16x16	0.015	0.031	≈0	≈0	0.031	0.031	0.031	≈0	0.093	0.031	≈0	≈0
Compressed size (KB)	Block size	Red				Green				Blue			
		AC-Matrix$_1$	HL_2	LH_2	LH_2	AC-Matrix$_1$	HL_2	LH_2	HH_2	AC-Matrix$_1$	HL_2	LH_2	LH_2
49	8x8	0.35	0.046	0.031	0.031	1.76	0.015	≈0	≈0	0.062	≈0	≈0	≈0
10.42	8x8	0.046	≈0	≈0	≈0	0.124	≈0	≈0	≈0	≈0	≈0	≈0	≈0
5.12	8x8	0.031	≈0	≈0	≈0	0.062	≈0	≈0	≈0	≈0	≈0	≈0	≈0
36.34	16x16	0.17	0.046	0.062	0.062	1.27	0.031	≈0	≈0	0.015	0.031	≈0	≈0
6.61	16x16	≈0	≈0	≈0	≈0	0.1	≈0	≈0	≈0	≈0	≈0	≈0	≈0
2.93	16x16	0.031	≈0	≈0	≈0	0.062	≈0	≈0	≈0	≈0	0.031	≈0	≈0

Table5.15: Estimatedexecution time for FMS-Algorithm at two levels DWT for Corner image

Compressed size (KB)	Block size	Y				Cb				Cr			
		AC-Matrix$_1$	LH_2	HL_2	HH_2	AC-Matrix$_1$	LH_2	HL_2	HH_2	AC-Matrix$_1$	LH_2	HL_2	HH_2
31.72	8x8	4	0.016	0.031	≈0	0.093	≈0	0.015	≈0	0.1	0.031	0.031	≈0
13.11	8x8	0.85	≈0	0.031	≈0	0.031	≈0	≈0	≈0	0.046	≈0	0.031	≈0
23.14	16x16	0.2.49	0.015	0.016	≈0	0.062	0.031	≈0	≈0	0.1	0.031	0.031	≈0

It can be seen from the above figures and tables that a single level DWTis applied successfully to the colour images using both YCbCr and RGB layers. Also, the two-level DWT gives good performance. However, the two-level DWT did not perform well on YCbCr layer at higher compression ratios. Both colour images Wall and Room contain green stripe lines; this renders RGB layers more appropriate to be used with the proposed approach. On the other hand, for the image Corner, the YCbCr layer proved more appropriate.

Each layer from true colour RGB, compress independently without change in colcour format, and this new feature added to our proposed algorithm to compress true colour images RGB without needs to any kind of layer transformations. Additionally, same colour images are transformed to different colour format (YcbCr) to show our compression algorithm ability to compress the images.

JPEG and JPEG2000 are tested with the same 2D images for comparison with our proposed method based on similar compression ratios; we also tested imagequalitythroughRoot-Mean-Square-Error (RMSE).We also show the visualization of 3D surfaces for decompressed 2D images by JPEG and JPEG2000 as a means of comparison.

Table5.16: Comparison JPEG2000 and JPEG with our approach for Face1 image

Proposed algorithm			JPEG2000		JPEG	
Compression Ratio	2D RMSE	3D RMSE	2D RMSE	3D RMSE	2D RMSE	3D RMSE
97.9%	4.83	2.31	3.48	3.31	12	2.73
99.2%	6.21	2.4	5.33	3.49	Not Applicable	Not Applicable
99.5%	7.41	3.39	6.32	2.91	Not Applicable	Not Applicable

Table5.17: Comparison JPEG2000 and JPEG with our approach for Face2 image

Our Proposed algorithm			JPEG2000		JPEG	
Compression Raio	2D RMSE	3D RMSE	2D RMSE	3D RMSE	2D RMSE	3D RMSE
98%	5.22	2.19	4.1	1.51	8.57	1.67
98.9%	5.86	1.64	5.3	2.68	Not Applicable	Not Applicable
99.4%	6.96	2.16	6.61	2.57	Not Applicable	Not Applicable
99.6%	8.43	1.48	7.61	2.62	Not Applicable	Not Applicable

Table5.18: Comparison JPEG2000 and JPEG with our approach for Face3 image

Proposed algorithm			JPEG2000		JPEG	
Compression Raio	2D RMSE	3D RMSE	2D RMSE	3D RMSE	2D RMSE	3D RMSE
98%	4.7	1.32	3.83	1.25	7.64	1.78
98.9%	5.43	0.83	5.05	1.82	Not Applicable	Not Applicable
99.5%	6.65	4.02	6.54	1.85	Not Applicable	Not Applicable

Table5.19:ComparisonJPEG2000 and JPEG with our approach for Wall image

Proposed algorithm			JPEG2000		JPEG	
Compression Raio	2D RMSE	3D RMSE	2D RMSE	3D RMSE	2D RMSE	3D RMSE
99.3%	4.37	1.96	2.63	0.31	9.66	0.66
99.7%	4.36	0.2	3.47	0.49	Not Applicable	Not Applicable
99.8%	4.85	0.37	4.79	1.1	Not Applicable	Not Applicable

Table 5.20:comparison JPEG2000 and JPEG with our approach for Room image

Proposed algorithm			JPEG2000		JPEG	
Compression Ratio	2D RMSE	3D RMSE	2D RMSE	3D RMSE	2D RMSE	3D RMSE
99.5%	6.36	0.12	7.59	0.23	20	113.59 (**not matched**)
99.8%	9.0	1.65	11.33	1.35	Not Applicable	Not Applicable
99.9%	9.22	2.21	13.59	94.67 (**not matched**)	Not Applicable	Not Applicable
99.9%	11.26	0.42	15.06	Not Applicable	Not Applicable	Not Applicable

Table 5.21:comparison JPEG2000 and JPEG with our approach for Corner image

Proposed algorithm			JPEG2000		JPEG	
Compression Ratio	2D RMSE	3D RMSE	2D RMSE	3D RMSE	2DRMSE	3DRMSE
99.1%	3.58	1.78	3.12	1.92	5.86	16.46
99.5%	4.59	0.42	3.86	74.64 (**not matched**)	Not Applicable	Not Applicable
99.7%	6.5	2.1	4.64	69.55 (**not matched**)	Not Applicable	Not Applicable

In Tables5.16,5.17, 5.18, 5.19, 5.20 and 5.21"not matching" means the relevant algorithm cannot compress to the required size successfully.

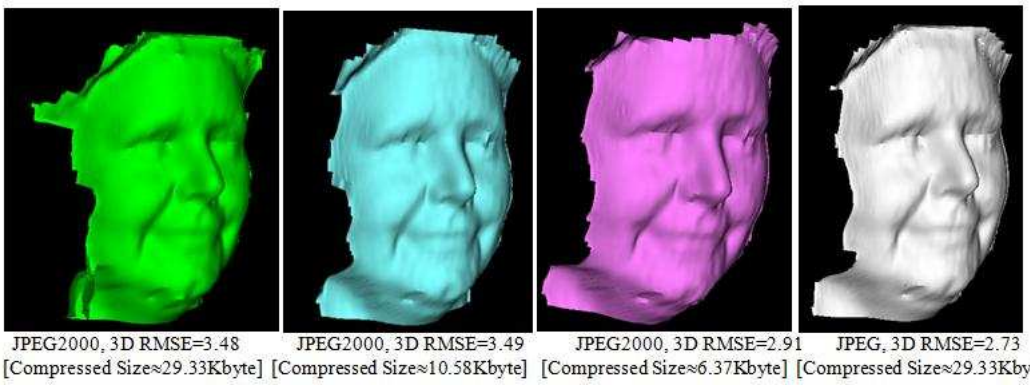


Figure 5.17: Decompressed 2D Face1 image by using JPEG2000 and JPEG algorithm, JPEG algorithm can' t compress the 2D Face1 image under 29 KB.

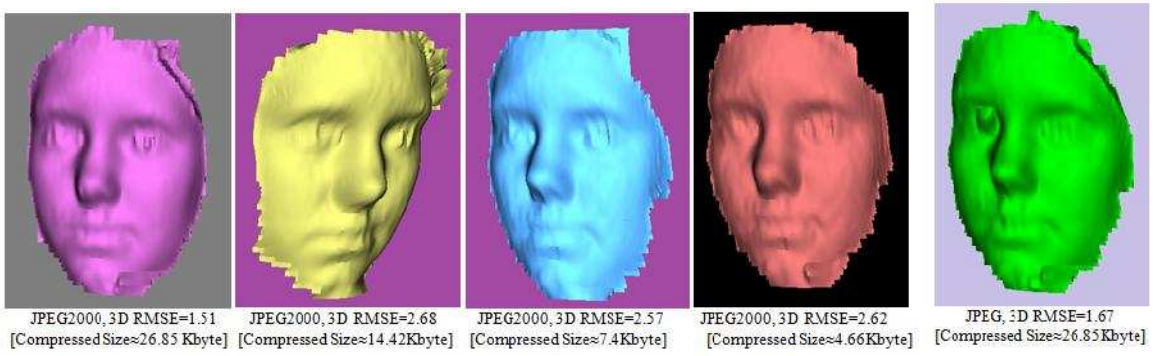


Figure 5.18: Decompressed 2D Face2 image by using JPEG2000 and JPEG algorithmdegradation appeared by JPEG on the surface at 26 KB.

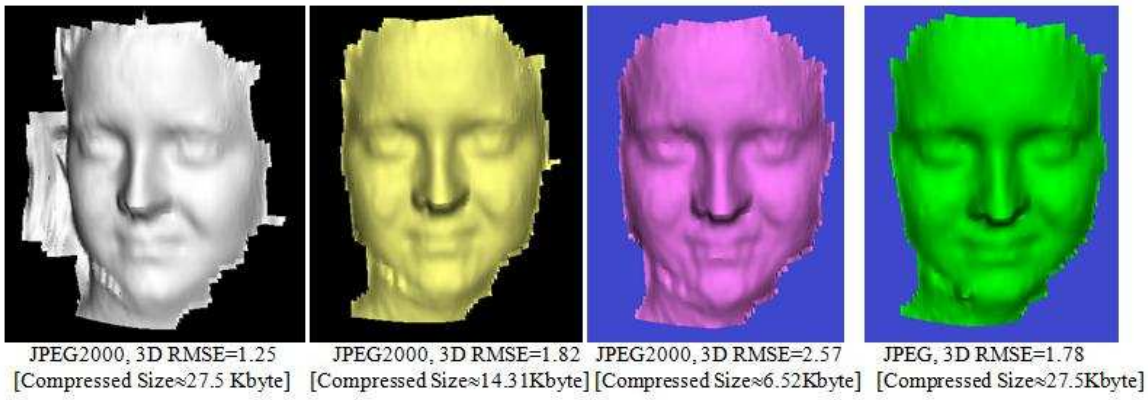


Figure 5.19: Decompressed 2D Face3 image by using JPEG2000 and JPEG algorithm, parts of images failed to reconstruct in 3D by JPEG2000 algorithm at 14 KB, degradation appears on the surface by JPEG2000 under 7 KB, also JPEG algorithm degrades the surface at compressed size 27 KB(JPEG fails to compress under 27 KB).

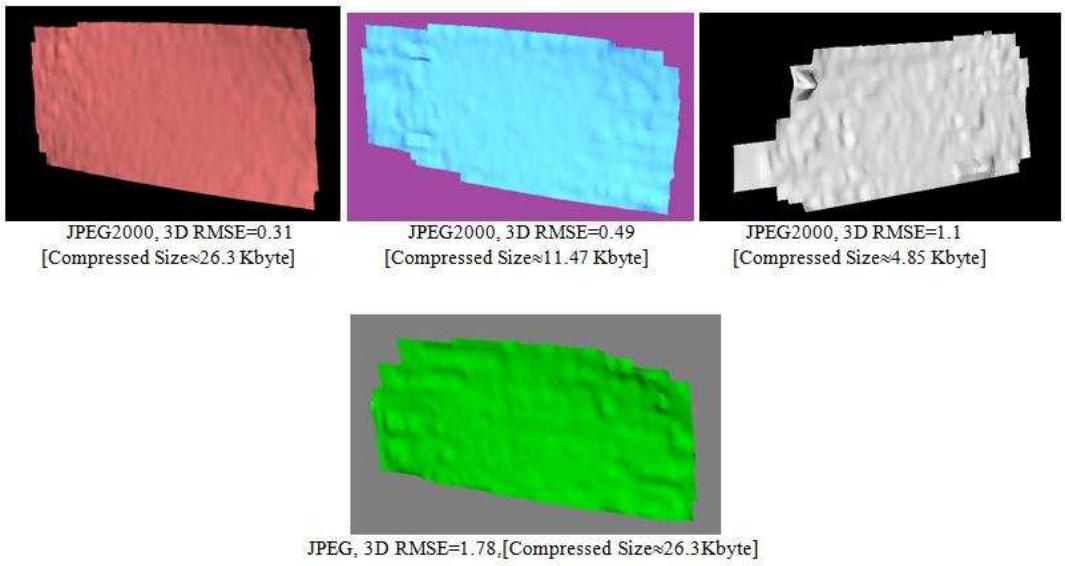


Figure 5.20: Decompressed Wall image by using JPEG2000 and JPEG algorithm, degradation appears on surface by JPEG2000 fewer than12 KB, also JPEG algorithm degrades the surface at compressed size 27 KB, JPEG fails to compress under 27 KB.

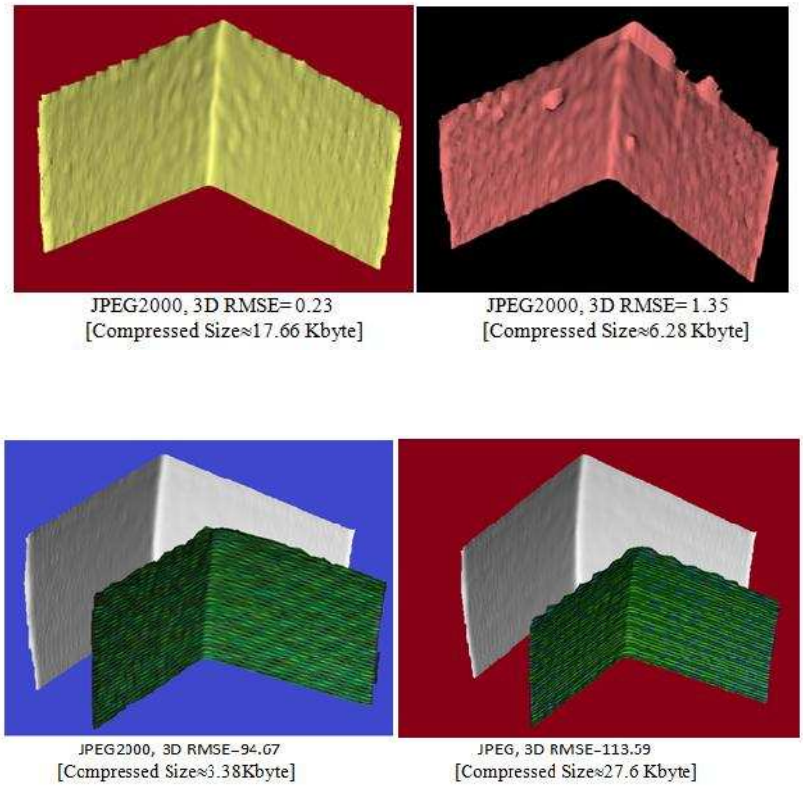
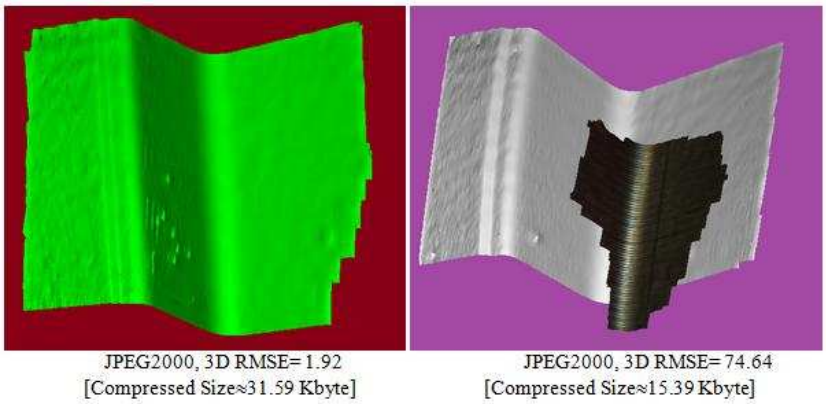


Figure 5.21: DecompressedRoom image by using JPEG2000 and JPEG algorithm, degradation appears on surface by JPEG2000 fewer than 6 KB, also JPEG2000 cannot reconstruct 3D surface matches with original surface (grey colour) at 4 KB, similarly, JPEG algorithm fails to reconstruct 3D surface matching with original surface at 27 KB



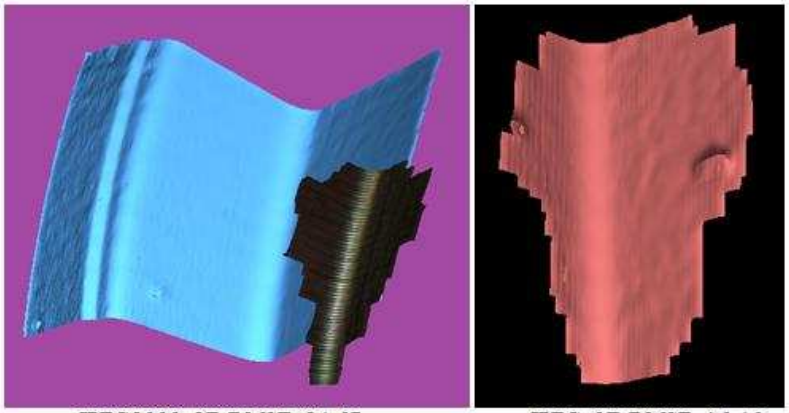


Figure5.22: Decompressed Corner image by using JPEG2000 and JPEG algorithm, top-left surface decompressed successfully by JPEG2000, in top-right decompressed surface by JPEG2000 not matched with original surface (grey surface). Under 15KB, similarly, JPEG algorithm fails to compress successfully fewer than 31 KB.

5.5. Conclusion

This Chapter has presented and demonstrated a novel method for image compression and comparedthequality of compression through 3D reconstruction, 3D RMSE and the perceived quality of the 3D visualisation. The method is based on a two-levelDWT transform and two-level DCTtransform in connection with the proposed Matrix Minimization algorithm. The results showed that our approach introduced better image quality at higher compression ratios than JPEG and JPEG2000 being capable of accurate 3D reconstructing at higher compression ratios. On the other hand, it is more complex than JPEG2000 and JPEG. The most important aspects of the method and their role in providing high quality image with high compression ratios are discussed as follows:

1- In a two-levelDCT, the first level separates the DC-values and AC-values into different matrices; the second level DCTis then applied to the DC-valuesand this generates two new matrices. The size of the two new matrices are only a few bytes long (because they contain many zeros), this process increases the compression ratio.

2- Since most of the high-frequency matrices contain lot of zeros as above, in this chapter we used the EZSN algorithm, to eliminate zeros and keep non-zero data. This process keeps significant information while reducing matrix sizesup to 50% or more.

3- The Matrix Minimization algorithm is used to replace each three coefficients from the high-frequencies matricesby a single floating-point value. This process converts each high-frequency matrix into a one-dimensional array, leading to increased compression ratios while keeping the quality of the high-frequency coefficients.

4- The FMS-Algorithm represents the core of our search algorithm for finding the exact original data fromaone-dimensional array (i.e. Reduced-Array) converting toa matrix, and depends on the organized key-values and Limited-Data.According to time execution tables, the FMS-

Algorithm finds values in a few microseconds, for some high-frequencies needs just few nanoseconds at higher compression ratios.

5- The key-values and Limited-Data are used in coding and decoding an image, without these information images cannot be reconstructed.

6- Our proposed image compression algorithm was tested on true colour images (i.e. Red, Green and Blue), obtained higher compression ratios and high image quality for images containing green striped lines. This makes our proposed compression algorithm featured more than JPEG and JPEG2000, because these methods can' t not compress 2D images without using YcbCr format. Additionally, our approach has been tested on YCbCr layers with good quality at higher compression ratios. This makes our algorithm run better no both colour formats (RGB and YcbCr).

7- Our approach gives better visual image quality compared to JPEG and JPEG2000. This is because our approach removes most of the block artefacts caused by the 8x8 two-dimensional DCT. Also,our approach uses asingle level DWTor two-levelDWT rather than multi-level DWTas in JPEG2000; for this reason blurring typical of JPEG2000 is removed inourapproach. JPEG and JPEG2000 failed to reconstruct a surface in 3D when compressed to higher ratioswhileit is demonstrated that our approach can successfully reconstruct the surface and thus, is superior to both on this aspect.

However, there isa larger number of steps in the proposed compression and decompression algorithm than in JPEG and JPEG2000. Also, the complexity of FMS-algorithm leads to increased execution time for decompression, because this algorithmis based on a binary search method.

Chapter 6

DCT and DST based Image Compression with Fast-Matching-Search Decompression

6.1. Introduction

The two most widely used image compression transforms are the discrete cosine transform (DCT) and the discrete wavelet transform (DWT) [119,151]. The DCT is usually applied to small, regular blocks of image samples (e.g. 8x8 squares) and the DWT is usually applied to larger image sections or to complete images. Many alternatives have been proposed, for example 3D transforms (dealing with spatial and temporal correlation), variable block size transforms, fractal transforms, and Gabor analysis. The DCT has proved particularly useful and it is at the core of most current generation of image and video coding standards, including JPEG, H.261, H.263, H.263+, MPEG-l, MPEG-2 and MPEG-4 [22,152].

In line with previous chapters, we focus on compressing 2D image data appropriate for 3D reconstruction. This includes 3D reconstruction from structured light images, and 3D reconstruction from multiple viewpoint images. In previous publications, we have argued that while geometry and connectivity of a 3D mesh can be tackled by several techniques such as high degree polynomial interpolation [114] or partial differential equations [125,135], the issue of efficient compression of 2D images both for 3D reconstruction and texture mapping has not yet been addressed in a satisfactory manner. Using structured light techniques for 3D reconstruction, surface patches can be compressed as a 2D image together with 3D calibration parameters, transmitted over a network and remotely reconstructed (geometry, connectivity and texture map) at the receiving end with the same resolution as the original data [133, 136].

In previous chapters we proposed a methodwhere a single level DWT is followed by a DCT on the LL sub-band yielding the DC component and the AC-matrix. A second DWT is applied to the DC components whose second level LL2 sub-band is transformed again by DCT. A matrix minimization algorithm was applied to the AC-matrix and other sub-bands. Compression ratios of up to 98% were achieved with a sequential search algorithm being used at decompression stage.In Chapter 4 we proposed atechnique[42]where a DWT was applied to variant arrangements of data blocks followed by arithmetic coding. The novel aspect of that paper is at decompression stage, where a Block Sequential Search Algorithm was proposed and demonstrated. Compression ratios of up to 98.8% were achieved. In Chapter 5[137]a two-level DWT was applied followed by a DCT to generate a DC-component array and an MA-Matrix (Multi-Array Matrix). The MA-Matrix was then partitioned into blocks and a minimization algorithm coded each block followed by the removal of zero valued coefficients and arithmetic coding. At decompression stage, a new algorithmcalled Fast-Match-Search decompression was used to reconstruct the high-frequency matrices by computing data probabilities through a binary search algorithm in association with a look up table. A comparative analysis of various

combinations of DWT and DCT block sizes was performed, with compression ratios up to 99.5%.

This Chapter introduces a new method for 2D image compression whose quality is demonstrated through accurate 3D reconstruction using structured light techniques and 3D reconstruction from multiple viewpoints. The method is based on two discrete transforms: 1) A one-dimensional Discrete Cosine Transform (DCT)is applied to each row of the image followed by quantization of the high frequencies. 2) The output from the previous step is transformed again by a one-dimensional Discrete Sine Transform (DST), which is applied to each column of data generating new sets of high-frequency components. The output is then divided into two parts where the low-frequency components are compressed by arithmetic coding and the high frequency ones by a high frequency minimization algorithm[158].

At decompression stage,a binary search algorithm is used to recover the original high frequency components. The technique is demonstrated by compressing 2D images up to 99% compression ratio. The decompressed images, which include images with structured light patterns for 3D reconstruction and from multiple viewpoints, are of high perceptual quality yielding accurate 3D reconstruction. Perceptual assessment and objective quality of compression are compared with JPEG and JPEG2000 through 2D and 3D RMSE. Results show that the proposed compression method is superior to both JPEG and JPEG2000 concerning 3D reconstruction, and with equivalent perceptual quality to JPEG2000. The main steps in the compression algorithm are depicted in Figure 6.1.

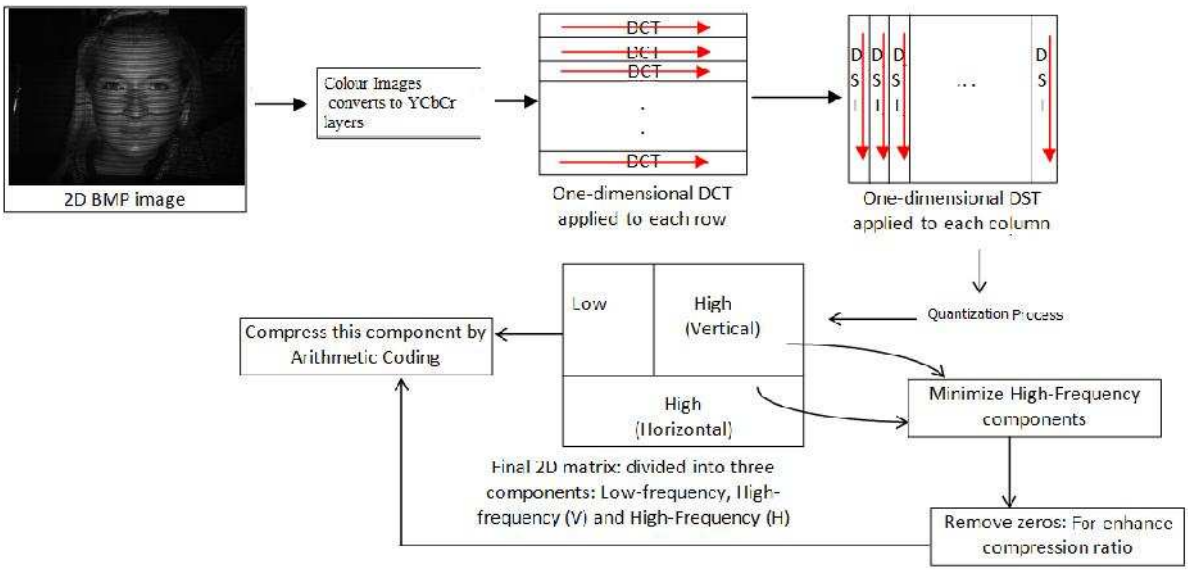


Figure 6.1:The main steps of the proposed compression algorithm.

6.2. Using the One-Dimensional Discrete Cosine Transform (DCT)

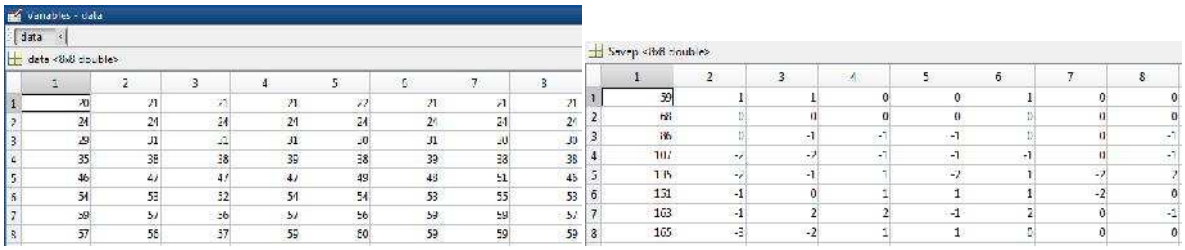
The one-dimensional DCT is used to transform each row from an image (spatial domain) to obtain the transformed data called "T_{dct}", as shown in the following [119,131,151, 158]:

$$T_{dct}(i) = \frac{\sqrt{2}}{n} C(i) \sum_{t=0}^{n-1} I(t) \cos\left[\frac{(2t+1)i\pi}{2n}\right] \tag{6.1}$$

$$I(t) = \frac{\sqrt{2}}{n} \sum_{j=0}^{n-1} C(j)\, T_{dct}(j) \cos\left[\frac{(2t+1)j\pi}{2n}\right] \tag{6.2}$$

$$\text{where } C(i) \begin{cases} = 2^{-1/2}, \text{ if } i = 0 \\ = 1 \ , \text{ if } i > 0 \end{cases}$$

Where i=0, 1, 2, 3, …,n-1 is the image row index, and the output is a set of DCT coefficients "$\boldsymbol{T_{dct}}$". The first coefficient is called the DC coefficient, and the rest are referred to as the AC coefficients. Notice that the coefficients are real numbers, and they are rounded off to integers. The important feature of the DCT is that it is useful in image compression [134]. It takes correlated input data and concentrates its energy in just the first few transform coefficients. If the input data consists of correlated quantities,then most of the "n" transform coefficients produced by the DCT are zeros or small numbers [153], and only a few are large (normally the first data). The early coefficients contain the most important (low-frequency) image information and the later coefficients contain the less-important (high-frequency) image information [154, 158]. This feature allows good compression performance as a proportion of the less important coefficients can be discarded without much degradation in image quality. Figure 6.2 shows the DCT applied to each row of an8x8block without using scalar quantization.



Original data $\boldsymbol{T_{dct}}$: DCT applied to each row
(Coefficients are rounded off to integers)

Figure 6.2: (Left) Original block of data, (right) T_{dct} produced by applying one-dimensional DCT to each row independently.

6.3. One Dimensional Discrete Sine Transform (DST)

Our research has indicated thata one dimensional DCT works together with a one-dimensional DST yielding large amounts of high-frequency components. These high frequency components are useful to obtain high compression ratios comparable to the JPEG technique. In this research, we will apply one dimensional DST to each column of the transformed matrix "$\mathbf{T_{dct}}$" from previous section. The DST definition is represented as follows[153,155]:

$$T_{dst}(k) = \sum_{i=1}^{n} T_{dct}(i)\ Sin(\pi \frac{k*i}{n+1}) \tag{6.3}$$

k= 1,2,…*n, (where n: is column size)*

$$T_{dct}(i) = \frac{2}{n+1} \sum_{k=1}^{n} T_{dst}(k)\ Sin(\pi \frac{k*i}{n+1}) \tag{6.4}$$

Eq. (6.3) is used to transform "*n*" values of "T_{dct}" matrix into "*n*" coefficients. These are the low and high frequency coefficients containing important and less important image information. The one-dimensional DST is applied to each column of "T_{dct}" to produce a new transformed matrix "T_{dst}". The DST is equivalent to the imaginary part of the Discrete Fourier Transformation (DFT), while in this chapter the results of the DST are real numbers [156,157, 158]. The main advantage of using the DST for image compression in this context is that the DST preservers theimage quality encoded by the low frequency components of "T_{dct}" and increases the number of zeros, which can be discarded without loss of quality.

After the DST, we apply a quantization of the high frequency components of the transformed matrix "$\boldsymbol{T_{dst}}$". In this way, the quantization means losing only insignificant information from the matrix. Each coefficient in the matrix is divided by the corresponding number from a "*Quantization table*" and the result is rounded off to the nearest integer. The following equation is proposed as a quantization table.

$$Q(i,j)=(i+j)\ F \tag{6.5}$$

Where: F>0 and i,j=1,2,3,...,n x m (image dimensions)

In Eq. (6.5) "*F*" is a real number greater than zero. This value affects image quality asfor "*F*>1" image quality is decreased. There is no limit for *F*, however, from our experiments we suggest *F* from 0.1 to 10. Figure 6.3 shows the DST applied to each column and quantized by Eq.(6.5).

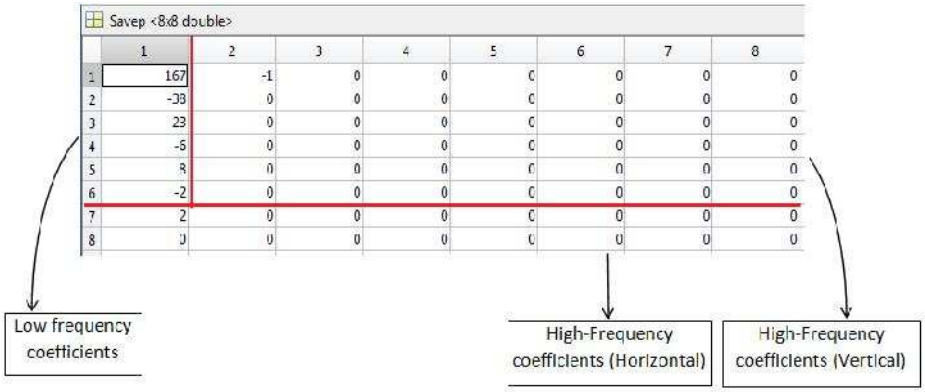


Figure 6.3: DST applied to each Column of T_{dct} followed by quantization with F=2 (cf. Eq.(6.5))

In the above example, low and high frequency components are controlled by the user. The low-frequency ones are not compressed any further, we just represent them in fewer bytes by arithmetic coding. Meanwhile, the high-frequency components either horizontal or vertical are compressed by the High-Frequency Minimization algorithm described in the next section.

6.4. High Frequency Minimization Algorithm

In this section, we describe an algorithm to convert the high-frequency coefficients (i.e. from previous section results passed to Minimization algorithm) into a compressed array called *Minimized-Array* through a matrix minimization method involving eliminating zeros and triplet encoding whose output is then subjected to arithmetic coding. Normally, the high frequency components contain large numbers of zeroes with a few nonzero data. The technique eliminates zeroes and enhancesthe compression ratio [134, 154, 158].

The high-frequency minimization algorithm is applied further reducing the size of high-frequency sub-matrix by 1/3.This process hinges on defining three key values and multiplying these by three adjacent entries in $\boldsymbol{H}$ which are then summed over producing single integer values(cf. Section 3.2.3) [42,137].Thus, each set of the three entries from $\boldsymbol{H}$ are converted into a single value which are then stored into a new coded array *Minimized_Array*. Assuming that n is the length of H , $i = 1,2, \dots, N-3$, and j is the index of new coded array, the following transformations define the high frequency encoding[133]:

$$M\_A_j = K_1 H_i + K_2 H_{(i+1)} + K_3 H_{(i+2)} \tag{6.6}$$

The key values$K_1, K_2, K_3$are generated by a key generator algorithm (cf. Section 5.2.3 - Key generator through Eq.(5.6)—(5.9))[137,154].

The keys are the weights and each triplet summationin theminimized-array that can later be recovered by estimating the $\boldsymbol{H}$ values(cf. Section 6.5) for the *Minimized_Array*. Following the models above, the *Minimized_Array* for the example in Figure 6.3 can be illustrated by the followingTable 6.1[158].

Table 6.1: From the example of Figure 6.3: each high-frequency sub-matrix is compressed independently

Assume that: **M = 2 (Maximum value in high-frequency sub-matrix: Horizontal)** **Keys K_1=1, K_2=5, K_3=18 (for both high-frequency components: Horizontal and Vertical)**		
High-Frequency Sub-matrix	**Compressed Size**	**Comments**
Minimized_Array$_{(Vertical)}$={-1,0,0, 0, ... 0}	compressed size 16	48(original size)/3 =16 data
Minimized_Array$_{(Horizontal)}$={2,0,0, 0,0,0}	compressed size 6	16(original size)/3 = 5.3 (last zero is alone)

Our compression method creates a new array of header data***H***, which is used later by the decompression algorithm to estimate the original data values. This information is kept in the header of the compressed file as a string (cf. Chapters3—5)).Per above example in Figure 6.3, the Limited-Data can be estimated from high-frequency sub-matrices (Horizontal and Vertical).Limited-Data$_{(Vertical)}$= {-1,0} and Limited-Data$_{(Horizontal)}$={2,0}.

The encoded triplets in the *Minimized_Array*may contain large number of zeros which can be further encoded through a process proposed algorithm in chapter 5 (cf. Figure 5.3) [137]. For example, assume the following encoded *Minimized_Array*={125, 0, 0,0,73, 0, 0,0,0,0, -17}.The zero array will be {0,3,0,5,0} where the zeros in red refer to nonzero data existing at these positions and the numbers in black refer to the number of zeros between two consecutive non-zero data. According to this method, the*Minimized_Array*both Horizontal and Vertical can be illustrated in Table 6.2.

Table 6.2: Each *Minimized-Array* is coded to zero-array and nonzero-array

High-Frequency Sub-matrix	**Zero-Array**	**Nonzero-Array**
Minimized_Array$_{(Vertical)}$={-1,0,0, 0 ... 0}	Zero$_{(Vertical)}$={**0**,5, 5, 5}	Nonzero_Array$_{(Vertical)}$={-1}
Minimized_Array$_{(Horizontal)}$={2,0,0, 0,0, 0}	Zero$_{(Horizontal)}$={**0**, 5}	Nonzero_Array$_{(Horizontal)}$={2}
Note: the "0" refers to the nonzero data in Nonzero-Arrays		

6.5. The Fast-Matching Search Decompression Algorithm

The decompression algorithm is the inverse of compression. First, decode the Minimized-Array for both horizontal and vertical components by combining thezero-array with the non-zero-array. Second, decode high-frequencies from the Minimized-Array using the fast matching search (FMS) algorithm[137].Third, inverse the DST and DCT to reconstruct the original 2D image.The images are then assessed on their perceptual quality and on their ability to reconstruct the 3D structures compared with the original images. Figure 6.4 illustrates the decompression method.

The *Fast Matching Search Algorithm (FMS)* has been designed to recover the original high frequency data. The compressed data contains information about the compression keys (K_1,K_2 and K_3) and Limited-Data followed by streams of compressed high frequency data. Therefore, the FMS algorithm picks up each compressed high frequency data and decodes it using the key values and compares whether the result is expressed in the Limited-Data. Given 3 possible values from Limited Data, there is only one possible correct result for each key combination, so the data is uniquely decoded. FMS-Algorithm is described in Chapter 5 (cf.Section 5.3)[137].

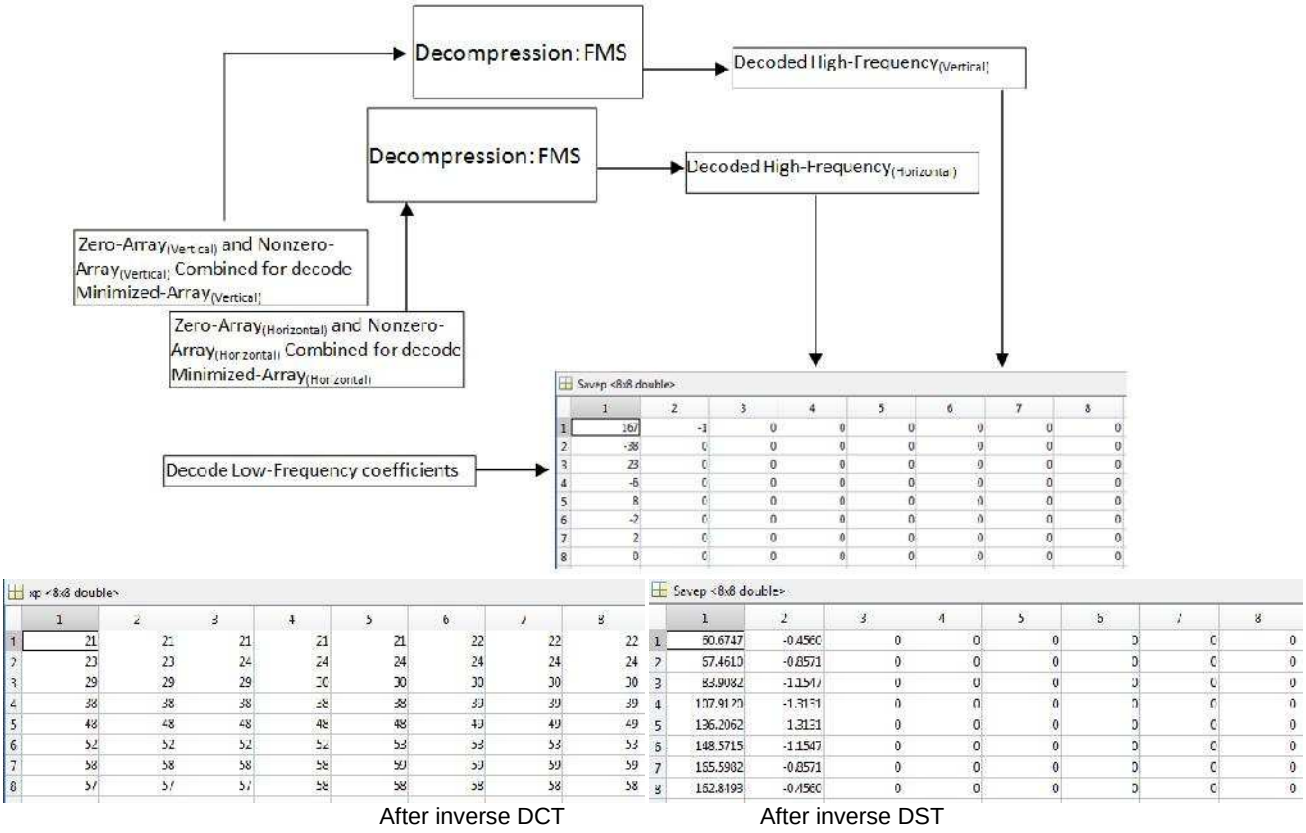


Figure 6.4: The steps in the decompression algorithm.

Once the horizontal and vertical high frequency components are recovered by the FMS-Algorithm, they are combined to regenerate the 2D matrix. Then each data from the matrix is multiplied by each data in ***Q*** (Eq. (6.5)) followed by the inverse DST (Eq. (6.4)) applied to each column. Finally, we multiply each data by ***F*** followed by the inverse DCT (Eq. (6.2)) applied to each row to recover the original 2D image as shown in Figure 6.4. If we compare the results in Figure 6.4 with the original 8x8 matrix of Figure 6.2, we findthat there is not much difference, and these differences do not affect image quality. This demonstrates that the proposedtechnique is very attractive for image compression.

6.6. Experimental Results

The experimental results described here were implemented in MATLAB R2013a and Visual C++ 2008 running on an AMD Quad-Core microprocessor. We describe the results in two parts: first, we apply the compression and decompression algorithms to 2D images that contain structured light patterns allowing 3D surface data to be generated from those patterns. The rationale is that a high-quality image compression is required otherwise the resulting 3D structure from the decompressed image will contain apparent dissimilarities when compared to the 3D structure obtained from the original (uncompressed) data. We report on these differences

in 3D through visualization and standard measures of RMSE-root mean square error. Second, we apply the method to general 2D images (with no structured light patterns) of different sizes and assess their perceived visual quality and RMSE. Additionally, we compare our compression method with JPEG and JPEG2000 through the visualization of 2D images, 3D surface reconstruction from multiple views and RMSE error measures.

6.6.1. Results for Structured Light Images and 3D Surfaces

3D surface reconstruction was performed with our own software developed within the GMPR group[125, 135, 136]. The justification for introducing 3D reconstruction is that we can make use of a new set of metrics in terms of error measures and perceived quality of the 3D visualization to assess the quality of the compression/decompression algorithms(cf. Section 3.4).

Figure 6.5 shows several test images used to generate 3D surfaces both in grayscale and colour. The top row shows two grayscale face images, FACE1 and FACE2 with size 1.37MB and dimensions 1392×1040 pixels. The bottom row shows colour images CORNER and METAL with size 3.75MB and dimension 1280×1024 pixels. We use the RMSE measure to compute the differences between decompressed images and original ones. The RMSE however, cannot give an absolute indication of which is the ‘ best’ reconstructed image or 3D surfaçeas errors may be concentrated in a region that may or may not be relevant to the perception of quality. To get a better assessment of quality, we analyse 3D surface images at various compression ratios.

Table 6.3: Structured light images compressed by our approach

Image Name	Original Image Size (MB)	Original Image size		Compressed Size (KB)	Compression Ratio	2D RMSE	3D RMSE
		DCT	DST				
FACE1	1.37	1	2	18.75	98.6%	4.82	1.51
		1	6	11.7	99.1%	6.22	1.54
FACE2	1.37	1	2	15.6	98.8%	1.89	2.25
		1	6	7.8	99.4%	2.56	2.67
CORNER	3.75	{1, 5, 5}	{2, 2, 2}	21.2	99.4%	5.56	1.36
		{1, 5, 5}	{2, 3, 3}	14.7	99.6%	7.0	0.5
METAL	3.75	{1, 5, 5}	{1, 5, 5}	27.5	99.2%	5.25	1.87
		{1, 5, 5}	{2, 5, 5}	12.1	99.6%	5.62	1.98

Table 6.3 shows the compressed size for our approach using two different values of quantization. First, the quantization scalar for FACE1 and FACE2 is 1. This means that after DCT each coefficient is divided by 1, this means rounding off each floating-point value to integer. Similarly, after DST the quantization equation is applied with F (cf. Eq. 6.5).

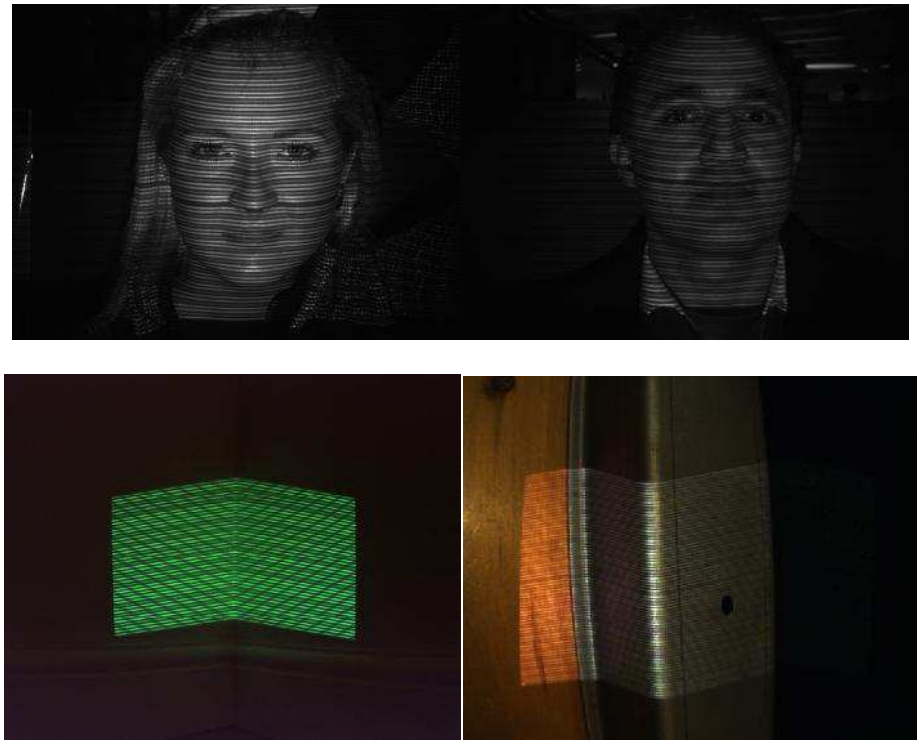
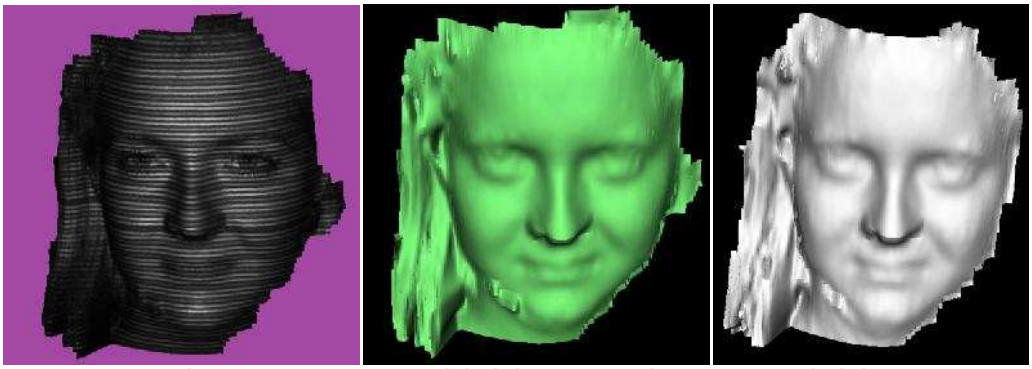
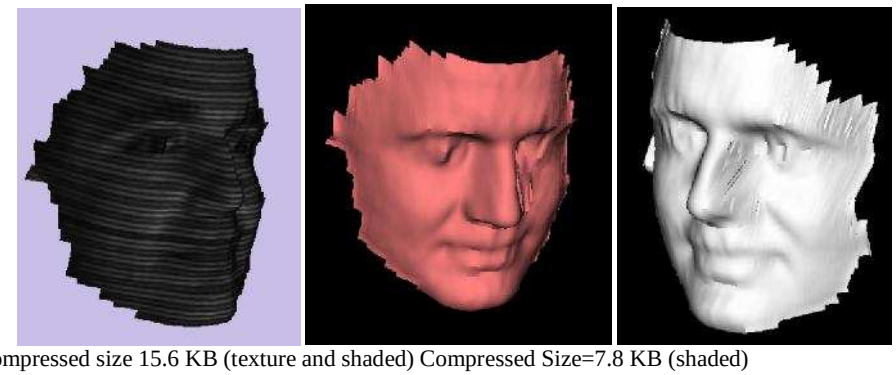


Figure 6.5: Structured light images used to generate 3D surfaces. Top row grayscale images FACE1 and FACE2, and colour images CORNER and METAL respectively.

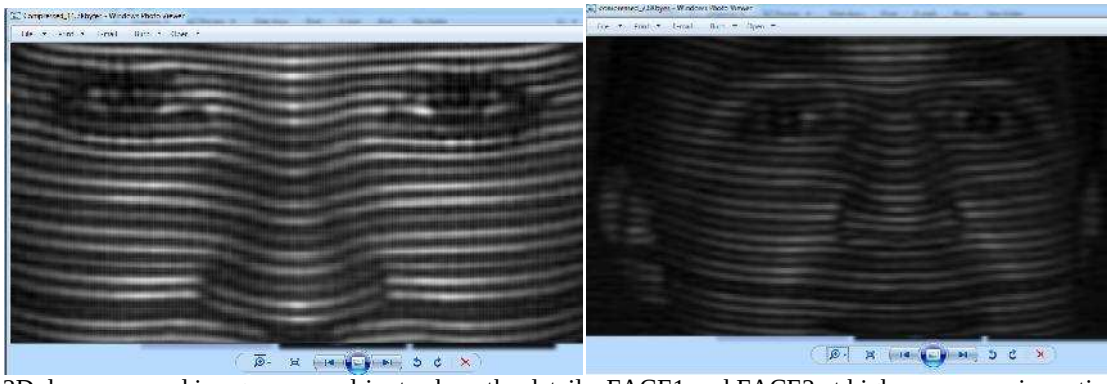
The colour images are defined by using colour transformation [22, 118] into YCbCr format. We then apply the proposed approach to each layer independently. For this reason, after DCT the quantization scalar for colour images is {1, 5, 5} for each layer of **Y, Cb** and **Cr** respectively.



FACE1: Compressed size 18.75 KB (texture and shaded). Compressed Size=11.7KB (shaded)
3D reconstructed FACE1 from decompressed image by our approach



FACE2: Compressed size 15.6 KB (texture and shaded) Compressed Size=7.8 KB (shaded)
3D reconstructed FACE2 from decompressed image by our approach



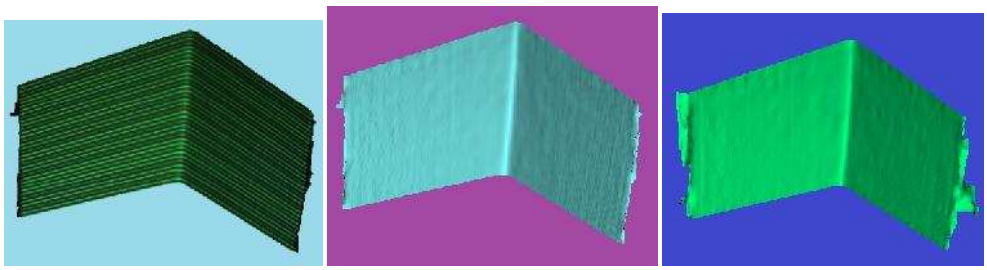
2D decompressed images zoomed-in, to show the details: FACE1 and FACE2 at higher compression ratio

Figure 6.6: Top: FACE1 shows decompressed 3D surface with texture and shaded at compressed size 18.7KB and 11.7KB. Middle: FACE2 shows decompressed 3D surface with texture and shaded at compressed size 15.6KB and 7.8KB. Bottom: details of 2D images FACE1 and FACE2 respectively at the higher (99%) compression ratio.

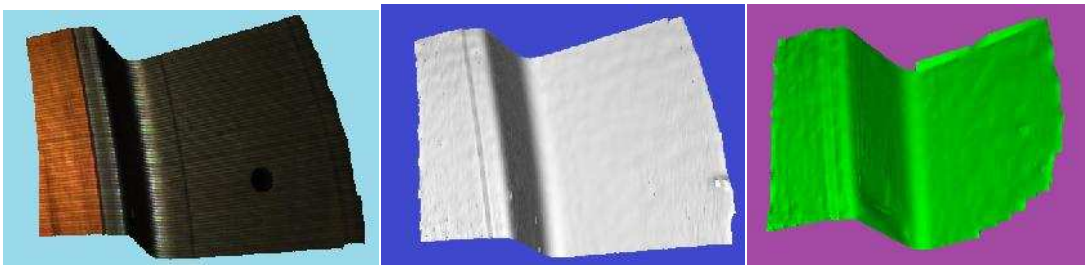
Figure 6.6 shows the visualization of the decompressed 2D images using different values for quantization. These decompressed images are converted to 3D surfaces. FACE1 on top of Figure 6.6 from left to right are higher quality surface per 3D RMSE. In fact, some parts of 3D surface have disappeared at higher compression ratio. But in FACE2 in the middle, the 3D reconstructed image at higher compression ratio is approximately the sameasfor low compression ratio. This means that 3D reconstruction depends on the structured light's quality in an image. Figure 6.6 (bottom) shows zoomed-in regions for the two images; the structure light patterns are clearly present even at 99% compression ratio.

Figure 6.7 shows 3D reconstructed surfaces for CORNER and METAL images respectively. On top, the quality of CORNER 3D surface at 99% compression ratio. But the 3D surface (top right) has some artefacts; this type of artefacts does not show in the original and decompressed 2D image at lower compression ratio. Artefacts appear when the structure light patterns are not clearly defined in the image, or are degraded after compression and decompression. In Figure 6.7 middle, the decompressed METAL image is converted to a 3D surface. The reconstructed 3D surface of middle right is degraded for all cases in which compression ratios exceed 99%. To

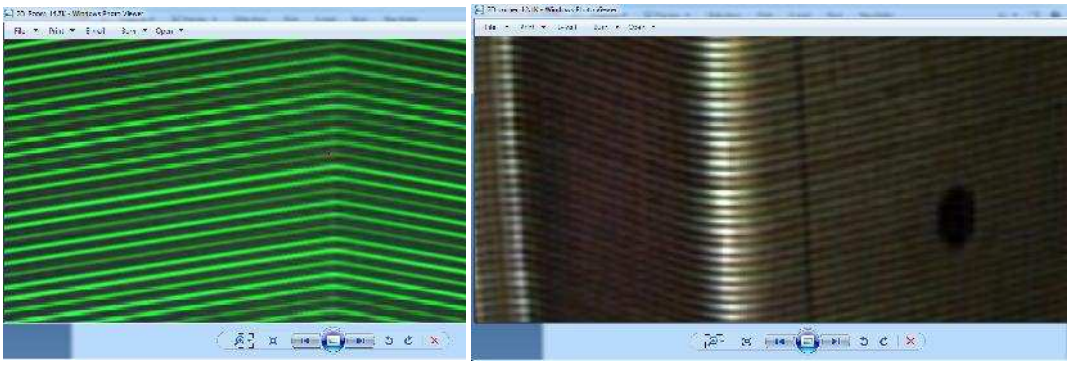
analyse 2D colour image compression, we zoomed-in the decompressed 2D images. It is shown that the structured light patterns are clearly visible at higher compression ratios of 99%.



CORNER: Compressed size 21.2 KB (texture and shaded) Compressed Size=14.7 KB (shaded)

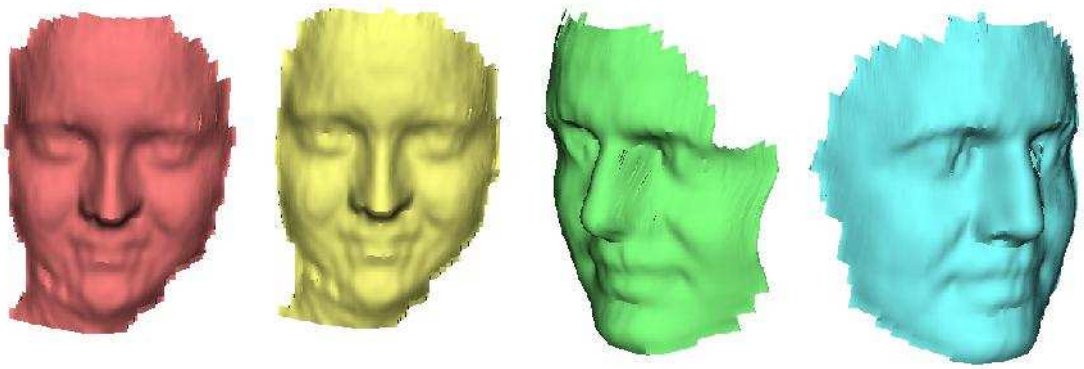


METAL: Left and middle: compressed size 27.5 KB (texture and shaded) Right: compressed Size=12.1 KB (shaded)

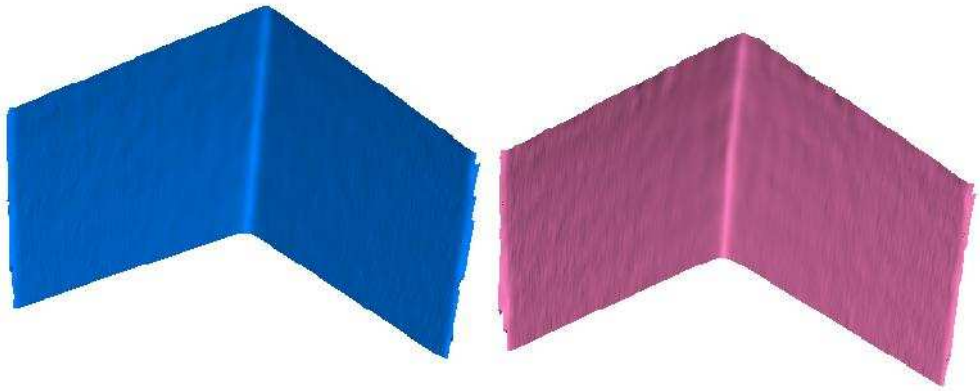


2D decompressed images zoomed-in, to show the details: CORNER and METAL at higher compression ratio

Figure 6.7: Top row:shows decompressed 3D surface of CORNER with texture and shaded at compressed sizes 21.2KBand 14.7KB. Middle row: shows decompressed 3D surface of METAL with texture and shaded at compressed sizes 27.5KB and 12.1KB. Bottom row: zoomed-in details for 2D images CORNER and METAL respectively at higher compression ratio.



JPEG2000 - 18.75 KB JPEG2000 - 11.7 KB JPEG2000 - 15.6 KB JPEG2000 - 7.8 KB



JPEG2000 - 21.2 KB JPEG2000 - 14.7 KB

Figure 6.8: Top: 3D reconstructed surface for FACE1 and FACE2 respectively using JPEG2000. Bottom: CORNER image successfully 3D reconstructed, while the METAL image failed 3D reconstruction.

Table 6.4: Compression and decompression of 3D images by JPEG2000 and JPEG at higher compression ratios

Image name	Compression Ratio	JPEG2000		JPEG	
		2D RMSE	3D RMSE	2D RMSE	3D RMSE
FACE1	99.1%	6.3	1.8	FAIL	FAIL
FACE2	99.4%	3.2	2.66	FAIL	FAIL
CORNER	99.6%	5.7	0.63	FAIL	FAIL
METAL	99.6%	4.17	FAIL	FAIL	FAIL

For a comparative analysis, we compressed and decompressed the 2D images by JPEG2000 and JPEG, then converted to a 3D surface.Figure 6.8 and Table 6.4describe the compressed and decompressed results for JPEG2000 only, as JPEG compression at equivalent ratios failed 3D reconstruction; that is, the images had so many artefacts that the 3D reconstruction algorithms were unable to successfully reconstruct a 3D surface. The comparison is based on applying the same compression ratios between JPEG2000 and our approach and show the visualization for the two methods. While the JPEG algorithm simply failed to compress the images at the required

ratio, it is important to stress that JPEG2000[37] cannot decompress some 2D images to equivalent quality 3D reconstruction as our method.Or,if it does, the 3D surface contains degradation. Figure 6.9 shows the compressed 2D images by JPEG2000 with zoomed in image details.

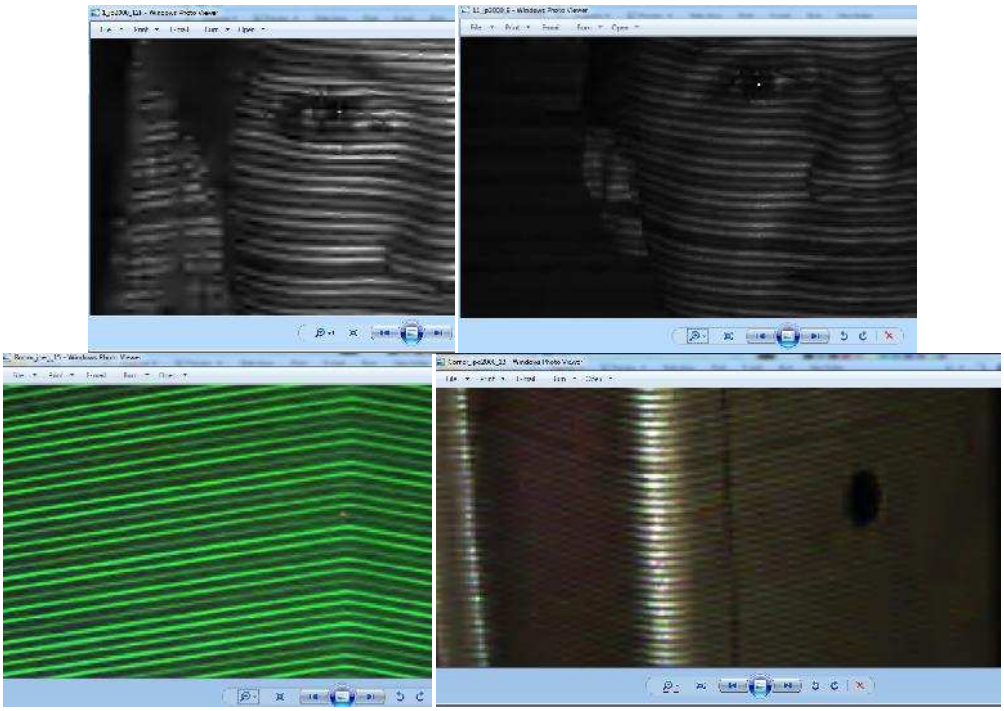


Figure 6.9. Details of 2D decompressed images by JPEG2000: Top: FACE1on the left is clearly blurred leading to degraded 3D reconstruction. Bottom: METAL image on the right is blurred rendering it unable to reconstruct a 3D surface.

6.6.2. Results for 2D Images

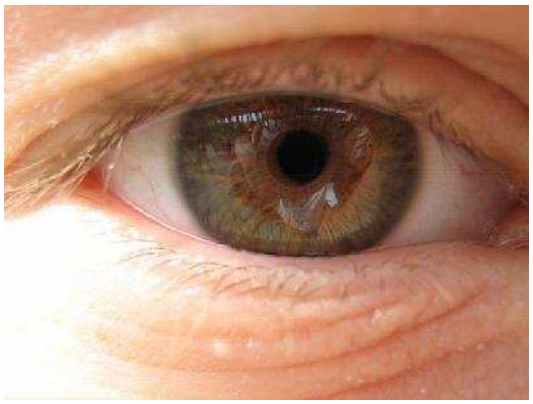
In this Section,we apply the algorithms to generic 2D images, that is, images that do not contain structured light patterns as described in the previous section. In this case, the quality of the compression is performed by perceptual assessment and by the RMSE measure. We use imageswith varying sizes from 2.25MB to 9MB. Also, we present a comparison with JPEG and JPEG2000 highlighting the differences in compressed image sizes and the perceived quality of the compression.

Figure 6.10(a) gives an indication of compression ratios achieved with our approach while in (b) is shown details with comparative analysis with JPEG2000 and JPEG. First, the decoded 'baby' image by JPEG2000 contains some blurring at places, while the same image decoded by our approach and JPEG are of higher quality. Second, the decoded 'eyes' image by JPEG algorithm had some block artefacts resulting in a lower quality compression. Also,the same image decoded by our approach and JPEG2000 at equivalent compression ratios, has excellent image quality.

Finally, the decoded 'girl' image by JPEG2000 is slightly degraded, while our approach and JPEG show good image quality.

Compressed size: 107.7 KB
Original size: 2.25 MB
Compression ratio: 95%

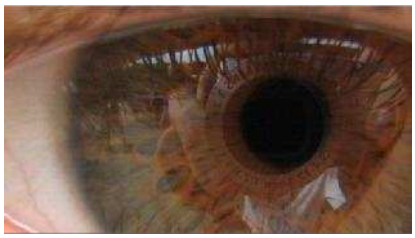
Compressed size: 59.4 KB
Original size: 3 MB
Compression ratio: 98%



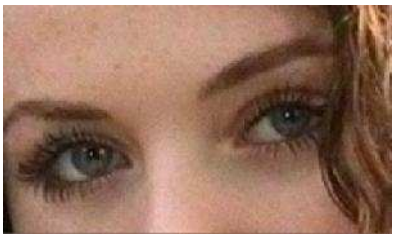
Compressed size: 59.9 KB
Original size: 9 MB
Compression ratio: 99%

(a) Compressed and decompressed 2D images by our approach

Our approach: RMSE=5.95



Our approach: RMSE=4.84



Our approach: RMSE=5.94

JPEG2000: RMSE=2.71

JPEG2000: RMSE=2.83

JPEG2000: RMSE=3.49

JPEG: RMSE=3.2

JPEG: RMSE=6.66

JPEG: RMSE=5.02

(b) Details of compression/decompression by our approach,JPEG2000 and JPEG respectively

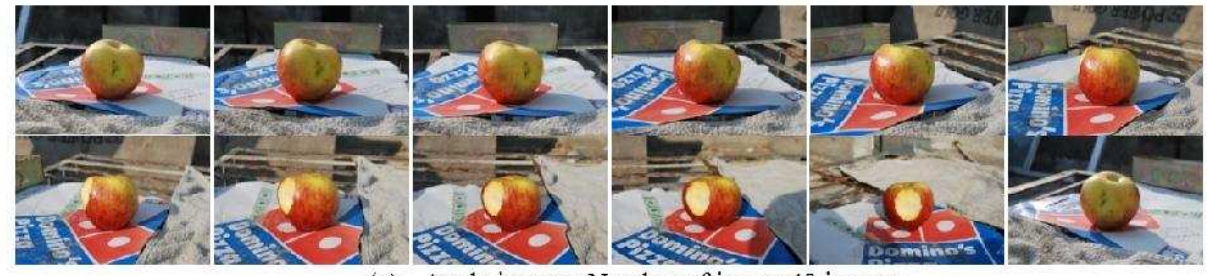
Figure 6.10: Compressed images by JPEG and JPEG2000 at equivalent compressed file sizesas with our approach.

Additionally, we applied our compression techniques to a series of 2D images and used**Autodesk 123DCatch** software to generate a 3D model from multiple images. The objective is to perform a direct comparison between our approach and both JPEG and JPEG2000 on the ability to perform 3D reconstruction from multiple views. Images are uploaded to the Autodesk server for processing which normally takes a few minutes. The 123D Catch software uses photogrammetric techniques to measure distances between objects producing a 3D model (i.e. image processing is performed by stitching a plain seam with correct sides together). The application may ask the user to select common points on the seam that could not be determined automatically [143, 144]. Compression sizes and RMSE for all images used are depicted in Table 6.5.

Table 6.5: Compressed sizes and 2D RMSE measures

Image Name	Number of images	Original image size (MB)	Quantization parameters used in DST			Compressed image size (MB)	Compression Ratio	Average compressed size of each image (MB)	Average 2D RMSE
			Y	Cb	Cr				
Baby	1	3	0.5	5	5	0.0594	98%	0.0594	5.95
Eyes	1	9	0.5	5	5	0.0599	99.3%	0.0599	4.84
Girl	1	2.25	0.5	5	5	0.1077	95.2%	0.1077	5.94
Apple	48	336	2	5	5	1.94	99.4%	0.0414	8.33
Face	28	200.7	1	5	5	1.72	99.1%	0.0629	5.68

Figure 6.11 shows two series of 2D images for objects “APPLE”, and “FACE” (all images areavailable from 123D Catch website). We start by compressing each series of images whose compressed sizes and 2D RMSE measures are shown in Table 6.5.A direct comparison of compression with JPEG and JPEG2000 is presented in Table 6.6. It is clearly shown that our approach and JPEG2000 can reach an equivalent compression ratio, while the JPEG technique cannot. It is important to stress that both our technique and JPEG depend on DCT. The main difference is that our approach is based on DCT with DST and the coefficients are compressed by the frequency minimization algorithm.This renders our technique far superior to JPEG as shown in the comparative analysis ofTable 6.6, where JPEG simply failed 3D reconstruction for images compressed to the same size as our technique.

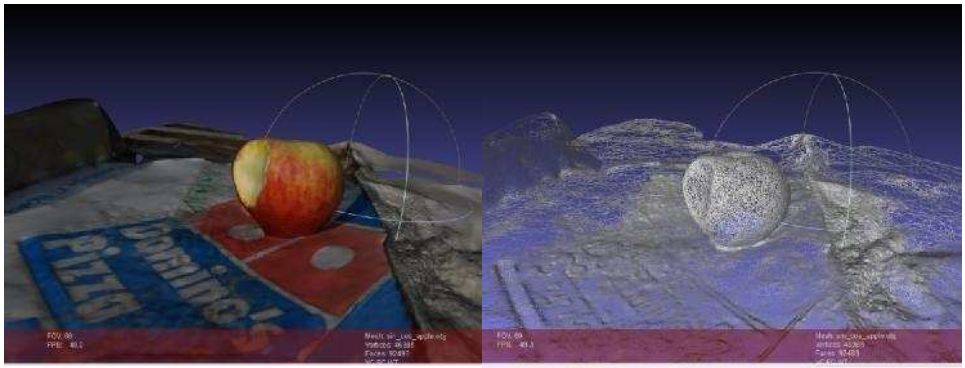


(a) Apple images: Number of image 18 images

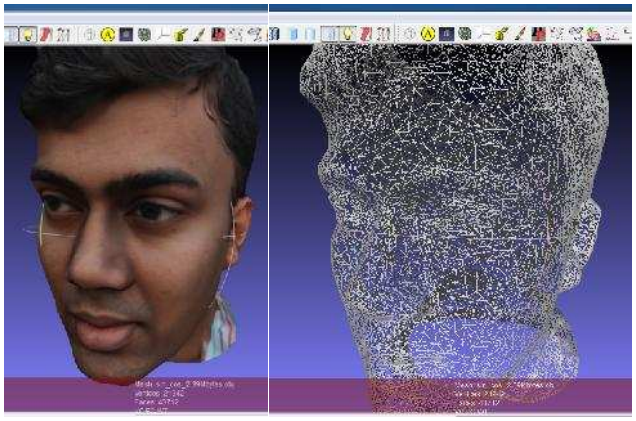
(b) Face images: Number of images 28 images

Figure 6.11: (a) and (b) show series of 2D images used to generate 3D models by 123D Catch.

In our method,DCT with DST are applied over the image as one block. The used low frequency block size for colour was 150x150, the scalar quantization for DCT was 1, 5 and 5 for each layer (Y,Cb and Cr) respectively. Furthermore, the quantization matrix used after DST performs an aggressive quantization, this means that approximately 50% of the coefficients are zero (i.e. the left bottom of the image matrix contains large number of zeros after the quantization process cf. Eq.(6.5)).



(a) 3D model for series of APPLEimages decompressed by our approach (48 images, average 2D RMSE=8.33, total compressed size=1.94 MB). The compression ratio for the 3D mesh is 99.4% for connectivity and vertices



(b) 3D model for series of FACE images decompressed by our approach (28 images, average 2D RMSE=5.68, total compressed size=1.72 MB). The compression ratio for the 3D mesh is 99.1% for connectivity and vertices

Figure 6.12: (a) and (b) Successful 3D reconstruction after compression by our approach.

Table 6.6: Comparison with JPEG and JPEG2000 techniques

Multiple 2D images	Original size (MB)	Compressed size (MB)	2D RMSE		
			Our approach	JPEG2000	JPEG
APPLE	336	1.94	9.5	6.58	FAIL
FACE	200.7	1.72	5.1	3.39	FAIL

6.7. Conclusions

This Chapter has presented and demonstrated a new method for image compression and illustrated the quality of compression through 2D and 3D reconstruction, 2D and 3D RMSE. Our compression algorithm is based on DCT applied to each row of an image, then followed by DST which is applied to each column of the matrix. After the transformation stage, the minimization of high frequency algorithm is used to reduce the number of high-frequency coefficients. The compression stage is then completed with arithmetic coding. In the decoding stage, the Fast-Matching-Search algorithm based on binary search is used to recover the original data. The results show that our approach introduces better image quality at higher compression ratios than JPEG and JPEG2000 as it can more accurately reconstruct 3D surfaces than both techniques. A slight disadvantage of the proposed method is that it is more complex than both JPEG2000 and JPEG. This is because our approach uses two types of transforms, and that neither JPEG nor JPEG2000 rely on a search method[158].

The most important aspects of the method and their role in providing high quality image with high compression ratios are identified as follows:

1. The one-dimensional DCT can be applied to an image row (i.e. largerarraysize≥ 8). Equally, the one-dimensional DST can be applied to each column of the output from DCT.
2. Theuser can ignore the scalar quantization to remove higher frequency coefficients (i.e. keeping more coefficients increases image quality).

3. The two-dimensional quantization (cf. Eq.(6.5)) provides a more aggressive quantization removing most of matrix contents as about 50% of the matrix entries are zero. Applying this over the DST can keep image quality at higher compression ratios.
4. The final transformed matrix is divided into: low-frequency sub-matrix, and horizontal and vertical high-frequency matrices.
5. The minimization of high frequency algorithm produces a Minimized-Array used to replace each three values from the high-frequency sub-bands by a single integer value. This process reduces the coefficients by 2/3 leading to increased compression ratios.
6. Since the Minimized-Array for both vertical and horizontal high-frequencies contains large number of zeros, we applied a new method to eliminate zeros and keep nonzero data. The process keeps significant information while reducing data up to 80%.
7. At decompression stage, the Fast-Matching-Search algorithm is the engine for estimating the original data from the minimized array and depends on the organized key values and the availability of a set of unique data. The efficient C++ implementation allows this algorithm to recover the high-frequency matrices very efficiently.

8. The key values and unique data are used for coding and decoding an image, without this information images cannot be recovered. This is an important point as a compressed image is equivalent to an encrypted image that can only be reconstructed if the keys are available. This has applications to secure transmission and storage of images and video data.
9. Our proposed image compression algorithm was tested on true colour and YCbCr layered images at high compression ratios. Additionally, the approach was tested on images resulting in better 3D reconstruction than JPEG2000 and JPEG.
10. The experiments indicate that the technique can be used for real-time applications such as 3D data files and video data streaming over the Internet.

Chapter 7

DCT and Matrix Minimization based Image Compression with Concurrent Fast-Matching-Search Decompression[1]

7.1. Introduction

As mentioned in previous Chapters, the Discrete Cosine Transform (DCT) is the basis of the popular JPEG file format, and most video compression methods and multi-media applications are generally based on it[117,119,127,138]. In other words, the image is divided into segments and the DCT is then applied to each segment creating a series of frequency components that correspond with detail levels of the image. Several forms of coding are applied to store only the most relevant coefficients.JPEG is more evident on large data repositories such as YouTube and cloud storage offered by several suppliers. With the increasing growth of network traffic and storage requirements, more efficient methods are needed for compressing image and video data with high quality reconstruction and potential significant reduction in storage size.

In this Chapter a new method for 2D image compression and reconstruction isproposed and demonstratedmaking used of the DCTandMatrix Minimization algorithm(described in previous chapters) at compression stage and a new concurrent binary search algorithm at decompression stage. The proposed image compression method in this chapter consists of five main steps:

(1) Divide the image into blocks and apply DCT to each block;
(2) Apply Matrix Minimization algorithm to AC-coefficients from each block to encode each block size 1:3 producing Minimized-Array;
(3) Build a look up table of compressed probability data to enable recover original high-frequencies data at decompression stage;
(4) Apply a delta or differential operator to the list of DC-components; and
(5) Apply arithmetic encoding to the outputs of steps (2) and (4).

Using a look up table at decompression stage, the concurrent binary search algorithm reconstructs all high-frequency AC-coefficients while the DC-components are decoded by reversing the arithmetic coding. Finally, the inverse DCT recovers the original image. We tested the technique by compressing and decompressing a range of 2D images, including images with structured light patterns for 3D reconstruction. The technique is compared with JPEG and JPEG2000 through 2D and 3D RMSE. Results demonstrate that the proposed compression method is perceptually superior to JPEG with equivalent quality to JPEG2000. Concerning 3D surface reconstruction from images, it is demonstrated that the proposed method is superiorto both JPEG and JPEG2000[159].

[1] This chapter was subject to a patent application, published as PCT on 1 Sep 2016. [160].

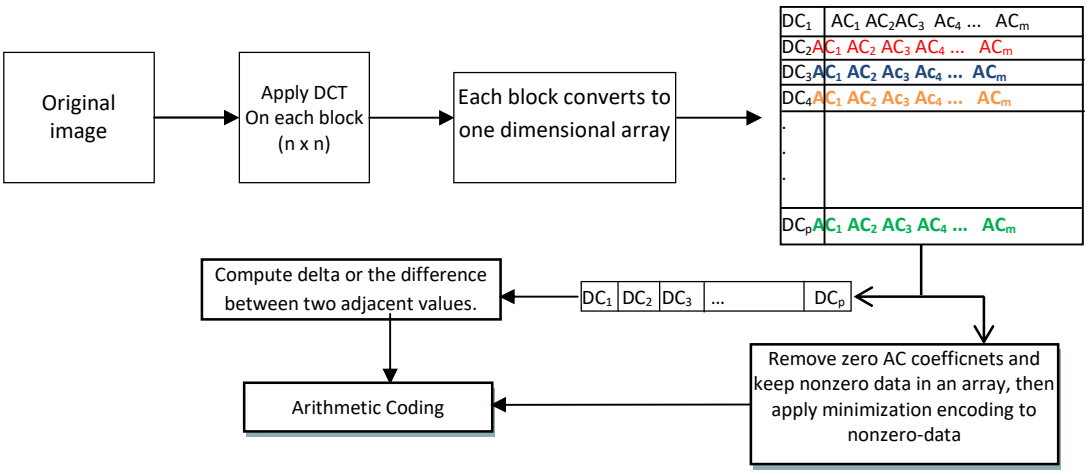


Figure 7.1:High level view of the proposed image compression algorithm

7.2. The Discrete Cosine Transform (DCT)

ThisSection describes how the DCT is applied tothe original image. First, the 2D image is divided into non-overlapping*nxn* blocks ($n \geq 8$) and then transformed by DCT to produce de-correlated coefficients. Each block in the frequency domain consist of: a DC-component at the first location of each block which is the average value of the samples in the block, and other coefficients called the AC coefficients as described in Eq.(3.1)[130,131,138].

The quantization of each block *n x n* can be represented as follows:

$$Q(i,j) = L(i+j) \tag{7.1}$$

Where $i,j = 1,2,\dots,n$ and the quantization factor is an integer $L > 1$.Each $n \times n$ block is quantized by Eq.(7.1) using dot-division-matrix whichtruncates the results. This process removes insignificant coefficients and increases the number of zeroes in each block.The parameter $L$is used to increase or decrease the values of Q. Thus, image details are reduced or lost as the value of Lincreases. The range of $L$is not limited a priori because it depends on the DCT coefficients and image resolution.The next step isto split the DC-components from eachquantizedblock $n \times n$ by savingthose intoa new array called *DC-Array*. Thenthe differences between two adjacent values in the DC-Array are computed (cf. Eq. (3.5)). This differential process generates coefficients that are correlated (generally the values are similar as the DC values of adjacent blocks tend to be similar) so their differences are small and more data are repeated. This process facilitates compression by arithmetic coding.

Meanwhile, the remaining AC coefficients (e.g. the 63 AC coefficients from an 8x8 block) are converted into a one dimensional array by scanning column-by-column and saved into a matrix called *AC-Matrix*. This matrix is subject to a process of eliminating all zeros followed by Matrix Minimization encodingalgorithm described next.

7.3. High FrequencyMinimization Encoding Algorithm

In this Section we introduce an algorithm to convert the AC-Matrix intoa compressed array called *Minimized-Array*through an EnhancedMatrix Minimization algorithm. The algorithm is enhanced byeliminating zeros and triplet encoding whose output is then subjected to arithmetic coding (cf. Section 5.2.3).Normally, the AC-Matrix contains a large number of zeroes with a few nonzero data. Herewe propose a technique to eliminate blocks of zeroes and store blocks of nonzero data into a one-dimensional array. The algorithm starts to partition the AC-Matrix into non-overlappingblocks $K \times K$where $K \geq 8$and then search for nonzero data inside the block. If the block contains nonzero data, such data will be stored into a reduced array $\boldsymbol{R}$.Otherwise, the block' s datawill be ignored, and the algorithm continues to search for nonzerodata in all blocks [137]. The algorithm is illustrated in List 5.1. (cf. Section 5.2.3)

Once only nonzero data are saved into the reduced$\boldsymbol{R}$array,the minimization of high-frequency encoding is applied furtherreducing its size by 1/3.This process hinges on defining three key values and multiplying these by three adjacent entries in R which are then summed over. Thus, each set of three entries from$\boldsymbol{R}$are converted into a single value which are then stored in a new coded minimizedarray[42,134,137]. Assuming that N is the length of R, $i = 1,2, \dots, N-3$ is the index of data in R **,** and p is the index of encoded*Minimized-Array*. The following transformations definethe minimizationof highfrequency encoding:

$$M\_A(p) = K_1 R(i) + K_2 R(i+1) + K_3 R(i+2) \tag{7.2}$$

Where the K_1, K_2 and K_3 are generated by a key generator (cf. Section 5.2.3)[137]. To map the *Limited-Data* values to each summation, extra information is needed to recover the original data as the problem is mathematically under-determined. This information is kept in the header of the compressed file as a string of unique data appearing in $\boldsymbol{R}$(cf. Figure 3.5).The minimized array may contain many zeros, and their removal is described in Chapter 5 (cf. Section 5.2.2) to increase the compression ratio[131].

7.4. Decompression Algorithm: Concurrent Binary Search Algorithm

While the DC-Array can be recovered by a simple addition process, the issue here is how to recover the reduced array $\boldsymbol{R}$that has been compressed into the minimized-array. For this purpose, we have devised anew *Concurrent Fast-Matching-Search Algorithm* (CFMS-Algorithm) derived from the single FMS-Algorithm described in Chapter 5 (cf. Section 5.3)

The reverse of the compression algorithm consists of three stages:

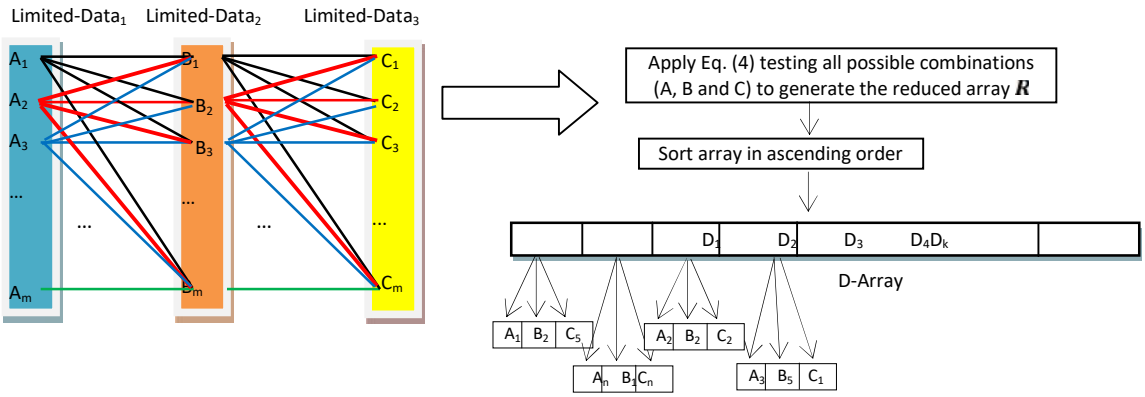
1) **Decode the DC-components**: the first step is to reverse the differential process of (cf. Eq. 3.7) by addition such that the encoded values in the DC-Array return to their original DC-components. This process takes the last value at position m, and adds it to the previous value, and then the total adds to the next previous value and so on.
2) **Decodethe minimized-array using the CFMS-Algorithm**: This novel algorithm has been designed to recover the reduced array $\boldsymbol{R}$from theminimized-array.

The compressed data contains information about the three compression keys defined in Eq.(7.3 – 7.6)and the probability data (limited data) followed by compressed streams of data. The CFMS-algorithm picks up in turn eachdata elementfrom the minimized-array and reconstructs the three keys recovering the triplet***R*** of data through a CFMS-Algorithm illustrated by steps A and B:

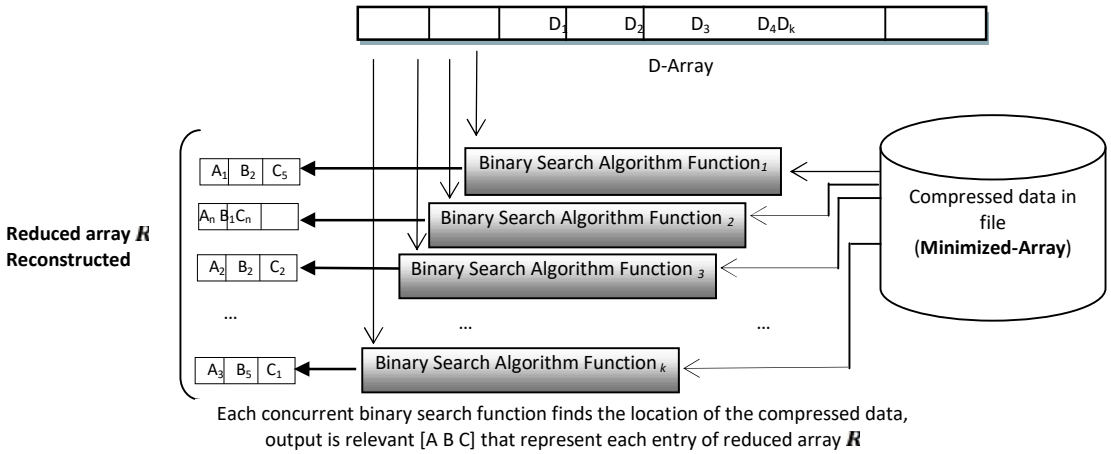
A) Initially, the estimated values defined in Limited-Data arrayare set to the same value, that is $A_1 = B_1 = C_1, A_2 = B_2 = C_2, A_3 = B_3 = C_3$. The searching algorithm computes all possible combinations of A withK_1, B with K_2 and C with K_3that yield a result keeping in D-array. As a means of an example consider that Limited-Data$_1$=[$A_1A_2A_3$] , Limited-Data$_2$=[$B_1B_2B_3$] and Limited-Data$_3$=[$C_1C_2C_3$]. Then, according to Eq.(7.2) these represent the coded summation respectively, and the equation is executed 27 times to build the *R* array, as described in Figure7.2(a). The match indicates that the unique combination of A,B and C are the original data (i.e. decompressed data)[137].

B) *A Binary Search algorithm*[139]is used to recoverthe data and their keys.Our design consists of k binary search algorithmsworking in concurrent toreconstruct the triplets of original data in the*R* array, as shown in Figure 7.2(b).At each step, each binary search algorithmtakes a single compressed data from minimized-arrayand compares with the middle element of the D-Array.If the values match, then a matching element has been found and its relevant (A,B and C) returned. Otherwise, if the search is less than the middle element the algorithm is repeated to the left of the middle element or, if the value is greater, to the right.Allbinarysearch algorithms aresynchronised[137].

3) **Combine the DC-components with AC-coefficients**: once the reduced array *R* is recovered in step 2, the corresponding high frequency AC-Matrix is re-built by placing the nonzero data in the exact locations defined by the algorithm inList-5.1. The DC-components and AC-coefficients are then followed by inverse quantization (dot-multiplication with Eq.(7.2) and the inverse DCT is applied to each block *n* x *n* Eq.(3.2), to recover approximately the original image.



(a) Compute all possibilities for keys with Limited-Data to reconstruct the reduced array $\boldsymbol{R}$



(b) Binary Search algorithms work in parallel to find group of decompressed data.

Figure 7.2: The CBS-Algorithm to reconstruct the reduced array R.

7.5. Experimental Results

The experimental results described here were implemented in MATLAB R2013a and Visual C++ 2008 running on an AMD Quad-Core microprocessor. We describe the results in two parts: first, we apply the compression and decompression algorithms to 2D images that contain structured light patterns allowing 3D surface data to be generated from those patterns. The rationale is that a high quality image compression is required otherwise the resulting 3D structure from the decompressed image will contain apparent dissimilarities when compared to the 3D structure obtained from the original (uncompressed) data. We report on these differences through visualization and standard measures of RMSE-root mean square error. Second, we apply the method to general 2D images (with no structured light patterns) of different sizes and assess their perceived visual quality and RMSE. Additionally, we compare our compression method with JPEG and JPEG2000 through visualization of 2D images and 3D surfaces and RMSE.

7.5.1. Results for Structured Light Images and 3D Surfaces

In Figure7.3 shows a number of test images used to generate 3D surfaces both in greyscale and colour. The top row shows two greyscale face images, FACE1 and FACE2 with size 1.37MB and dimensions 1392×1040 pixels. The bottom row shows colour images CORNER, WALL, METAL with size 3.75MB and dimension 1280×1024 pixels. As stated in previous Chapters, the RMSE although useful, is a single measure of error and may not give a clear indication to which reconstruction is `best'. This is so because errors could be concentrated in an area that we perceive as less important in the image, and this is more clearly seen by analysing the 3D surface images at various compression ratios.

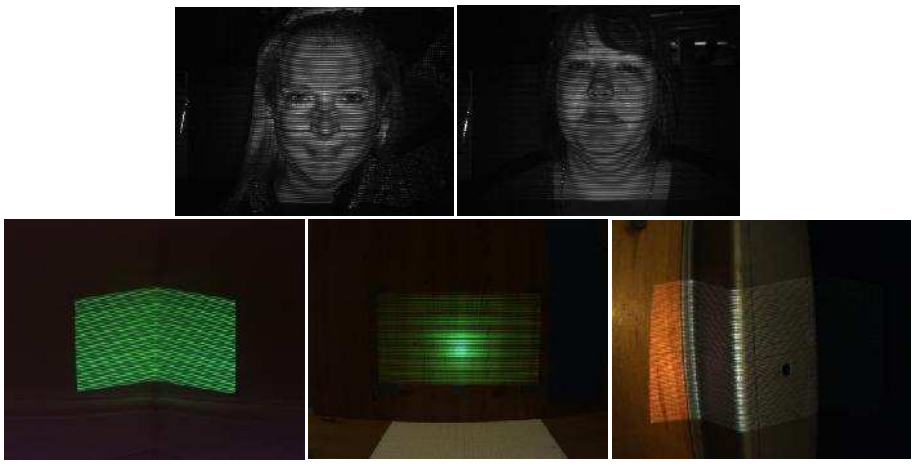


Figure 7.3:Structured light images used to generate 3D surfaces. Top row greyscale images FACE1 and FACE2, and colour images CORNER, WALL, METAL respectively.

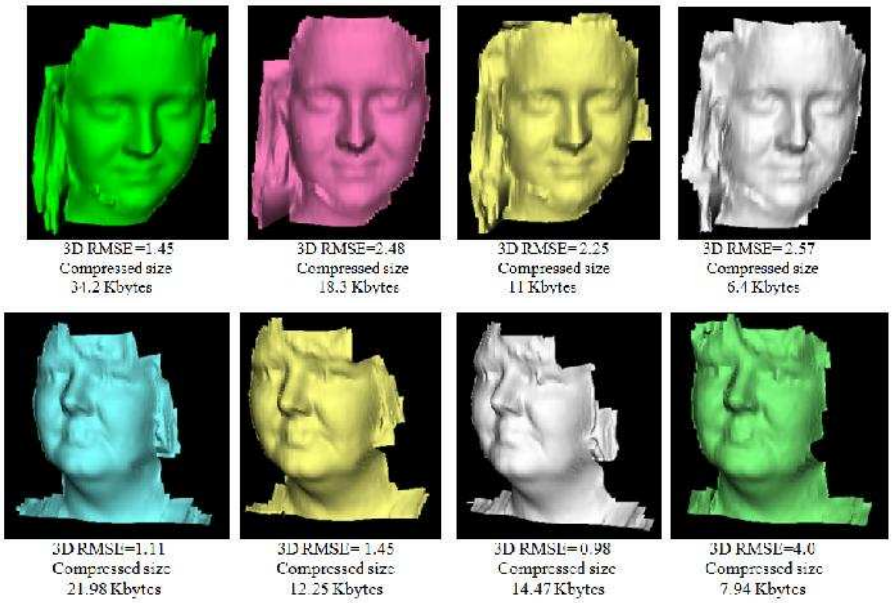


Figure 7.4:Reconstructed 3D surfaces from images FACE1 and FACE2 at various compression ratios.

Figure 7.4 shows a visualization of the decompressed images converted to 3D surfaces using different DCT block sizes (from 16×16 to 64×64). FACE1 on the top row from the left, the first and second 3D surfaces with RMSE of 1.45 and 2.48 are high quality surfaces comparable to the original one. The 3D surface with 3D RMSE of 2.25 represents median quality image while the 3D surface with 3D RMSE of 2.57 is low quality as some parts of surface are degraded. Note that the RMSE of 2.25 (third image from left) is lower than 2.48 (second image) but it's perceived quality is not higher, instead it is lower due to localised errors in less important areas of the face. Figure 7.4bottom row shows the decompressed FACE2 images. The 3D surfaces with 3D RMSE of 1.11 and 1.45 represent high quality surfaces comparable to the original surface, while the other two represent median to low quality with varying degrees of degradation. It is apparent here that because the RMSE algorithm only calculates the differences between valid surfaces points in two surfaces (original and reconstructed from compressed data) the dropping or disappearance of some areas on the surface will have a marked effect on the mean error.

Table 7.1: Proposed image compression and decompression applied to greyscale images (Original image size =1.37MB)

Image Name	Block size used by DCT	Factor F	Compressed image size	Compression Ratio	2D RMSE	3D RMSE
FACE1	16×16	5	34.2 KB	97.5%	4.0	1.45
	16×16	10	18.3 KB	98.6%	5.12	2.48
	32×32	5	20.7 KB	98.5%	4.79	2.25
	32×32	10	11 KB	99.2%	5.83	2.36
	(	10	6.4 KB	99.5%	6.65	2.57
FACE2	16×16	5	21.98 KB	98.4%	2.65	1.11
	16×16	10	12.25 KB	99.1%	3.32	1.45
	32×32	5	14.47 KB	98.9%	3.12	0.98
	32×32	10	7.94 KB	99.4%	3.8	4.0

Table 7.2: Proposed image compression and decompression applied to colour images (Original image size =3.75 MB)

Image Name	Block size used by DCT	Factor F for each layer [Y, Cb, Cr]	Compressed image size (KB)	Compression Ratio	2D RMSE	3D RMSE
WALL	64×64	[5,5,5]	14	99.6%	2.4	0.25
	64×64	[10, 10, 10]	7.62	99.8%	2.8	2.11
	64×64	[25, 25,25]	4.0	99.8%	3.5	0.59
CORNER	32×32	[10,10,10]	20	99.4%	5.34	0.14
	32×32	[20, 20, 20]	10	99.7%	6.7	0.65
	64×64	[30, 30,30]	5.1	99.8%	8.26	2.08
METAL	32×32	[2, 25, 25]	25.2	99.3%	4.19	1.89
	32×32	[5, 25, 25]	13.4	99.6%	4.48	2.04
	64×64	[5, 25, 25]	9.8	99.7%	4.73	2.00

Tables7.1 and 7.2 provide a quantitative view of compression concerning 2D structured light images and corresponding 3D surface reconstruction for a number of different DCT block sizes

and quantisation factors. The purpose is to analyse the sensitiveness of the algorithms to both parameters. In Table 7.1 there is only one value for quantization factor L as these are grey scale images and thus have only one colour channel to quantise. As expected, it is observed that by doubling the factor, the size of the compressed image is halved. On the other hand, by doubling the block size, the size of the compressed image is only reduced by about a third. It is also observed that no relationship exists concerning block size, factor, and RMSE (both in 2D and 3D). An image that is compressed to double the size of an earlier compression does not mean that its RMSE will be halved compared to the earlier RMSE. The reasons for this have been pointed out above as localised errors in the image will give rise to localised errors in the 3D structure and these do not necessarily correspond to our perception of better or worst.

Table 7.2depicts three parameters for quantization factor$F$, one for each channel as these are colour images. Here again by doubling the factor it is observed a halving of the compressed image size. Normally, it would not make sense to have different factor values for different colour channels, but this is a possibility that can be exploited especially in structured light applications where we know that patterns can be projected using a single colour channel (red, green or blue). The same comments above on RMSE also apply here.

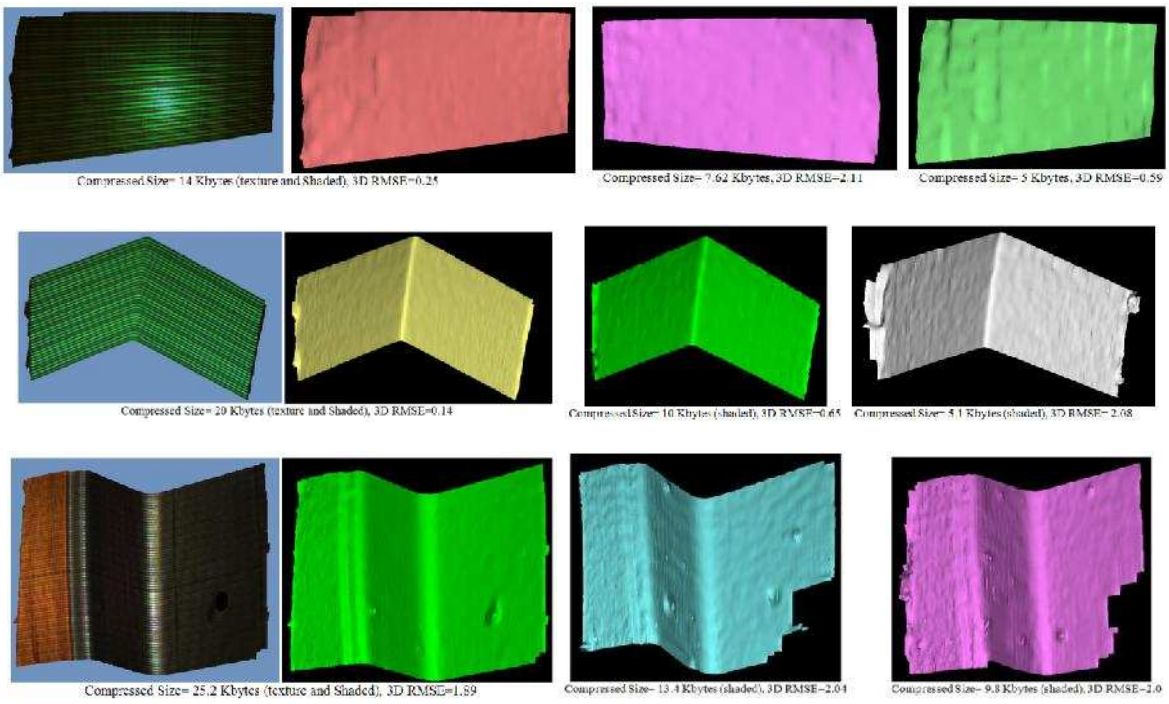


Figure.7.5:Reconstructed 3D surfaces for images WALL, CORNER and METAL after compression and decompression.

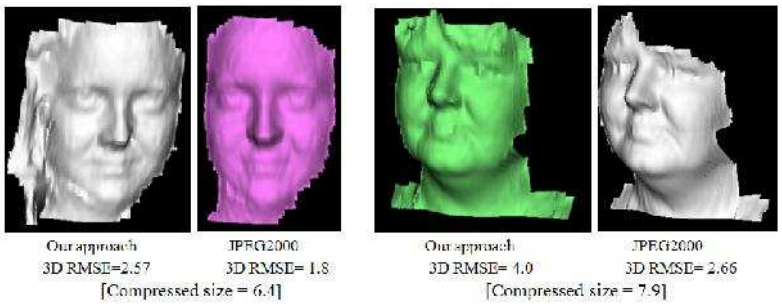
Figure 7.5 depicts the 3D surface images from the decompressed WALL, CORNER and METAL images. The first image on the left with texture mapping on is for information only. The remaining 3 shaded images were compressed by varying the DCT block size and the colour channels according to the data depicted in Table 7.2. Thus, the first rows of shaded images correspond to the first 3 entries in Table 7.2 and so on. The perceived quality of all reconstructed

3D surface images follows a similar pattern: as the quantisation factor F is increased, the size of the compressed file decreases with corresponding deterioration in quality and this is the expected behaviour.

Table 7.3: Compression and decompression of 3D images by JPEG2000 and JPEG at higher compression ratios

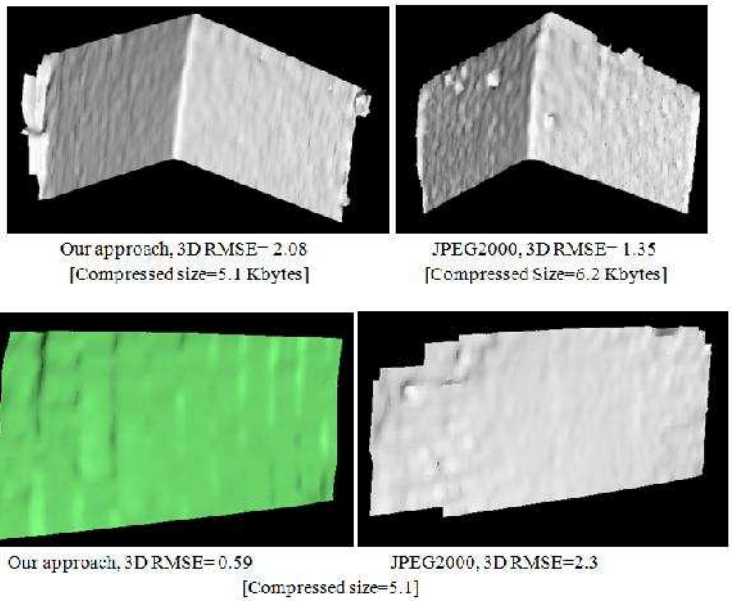
Image name	Compression Ratio	JPEG2000		JPEG	
		2D RMSE	3D RMSE	2D RMSE	3D RMSE
FACE1	99.5%	6.3	1.8	FAIL	FAIL
FACE2	99.4%	3.2	2.66	FAIL	FAIL
WALL	99.8%	3.8	2.3	FAIL	FAIL
METAL	99.8%	11.6	1.35	FAIL	FAIL
CORNER	99.6%	4.0	90	FAIL	FAIL

Table 7.3 and Figure 7.6 describe the compressed and decompressed results for JPEG and JPEG2000 with comparison with our approach. Here we compressed very aggressively and in Table 7.3 the JPEG algorithm simply failed to compress images at the required ratio with equivalent file sizes as our approach. This is indicated by "FAIL". An important point to note is that while JPEG2000 can compress to equivalent ratios or file sizes as our algorithm, the decompressed image is not of equivalent quality for the purposes of 3D reconstruction.

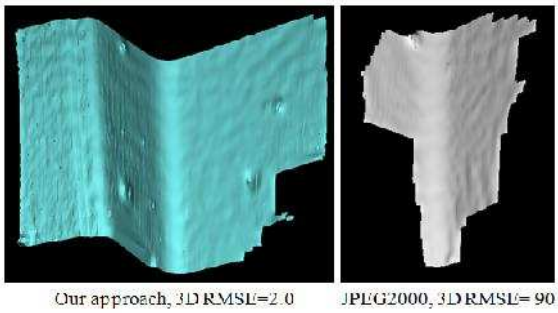


(**a**) The 3D reconstructed FACE1 (3D RMSE=1.8) by JPEG2000 degraded compared with our approach, also some parts are missing. FACE2 (3D RMSE=2.66) is compressed by JPEG2000 at higher compression ratio, but the top part of the surface is missing.

Figure 7.6 provides a direct comparison between our approach and JPEG2000 for quality assessment through visualisation of the reconstructed 3D surface. Each file containing structured light patterns was compressed to the same size using our method and JPEG2000. The visualisation clearly indicates that our method is superior to JPEG2000 concerning 3D reconstruction in all cases considered both in terms of perceived quality of the reconstruction and absolute RMSE.



(**b**) Top, the 3D reconstructed CORNER (3D RMSE=1.35) by JPEG2000 is more degraded than our approach. Bottom, the 3D reconstructed WALL (3D RMSE=2.3) by JPEG2000 has a higher compression ratio, but the top part of the surface is missing.



(**c**) The 3D reconstructed METAL (3D RMSE=90) by JPEG2000 is completely degraded compared with our approach.
Figure 7.6: (a), (b) and (c) Comparison of 3D reconstruction between our approach and JPEG2000

7.5.2. Results for 2D images

In this Section, we report on our approach applied to generic 2D images, that is, images that do not contain structured light patterns as described in the previous section. Table 7.4 tabulates compression results and comparison of our approach with the two compression algorithms JPEG2000 and JPEG respectively using 5 publicly available images with sizes varying from 0.5MB to 9MB. For each image, we used different block sizes from 8x8 to 64x64 as depicted in Table 7.4. Despite the RMSE limitations as an absolute measure of quality, the tabulated values indicate that JPEG has a much higher error than both our technique and JPEG2000. For this reason, it is the least desirable technique.

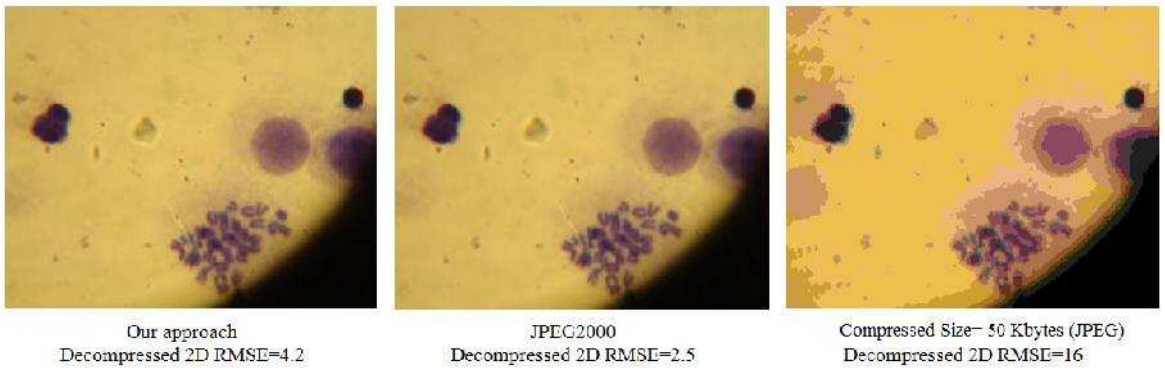
Table7.4: Proposed image compression and decompression applied to 2D images

Image Name	Original image size (MB)	Our approach			Our Approach 2DRMSE	JPEG2000 2D RMSE	JPEG 2D RMSE
		Block size used by DCT	Compressed image size (KB)	Compression Ratio			
X-ray	0.588	8 × 8	10	98.3%	5.0	3.2	11.88
Eye	9	64 × 64	14.2	99.8%	4.89	4.1	15.3
Girl	2.25	16 × 16	21.2	99%	10.48	6.4	21.1
Cell	8.5	64 × 64	9.8	99.8%	4.2	2.5	16
Baby	3	32 × 32	18.3	99.4%	5.3	3.5	15.5

(**a**) X-ray Compressed size to 10KB

(**b**) Eye image compressed to 14.2KB

(**c**) Girl image compressed to 21.2KB



(**d**) Cell image compressed to 9.8KB

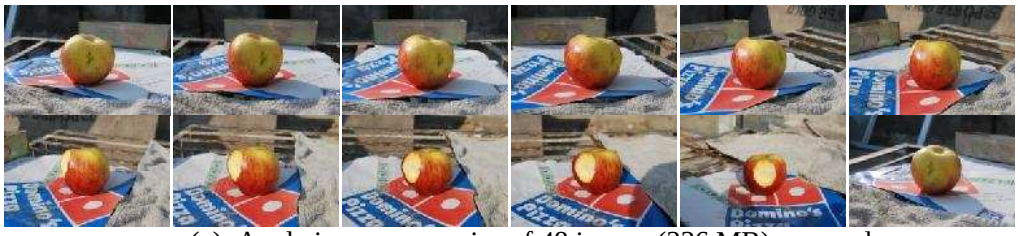
(e) Baby image compressed to 18.3KB

Figure7.7: (a-e) Comparative perceptual quality between our approach, JPEG2000 and JPEG

Figure 7.7depicts decompressed images by our approach with a comparison with JPEG2000 and JPEG. One can state that JPEG2000 seems to be the better technique for general 2D compression as it has a high perceived quality with low RMSE. Our technique is at comparable level to JPEG2000 concerning perceived quality, but with slightly higher RMSE. It is known that both JPEG and JPEG2000 are widely used in 2D image and video compression. This research has demonstrated that our proposed compression method can equally be used for the same purposes with the added advantage that it is superior to both JPEG and JPEG2000 concerning 3D surface reconstruction using structured light techniques.

7.5.3. 3D Modelling by using multiple images

Autodesk's 123D Catch software generates a 3D model from multiple pictures taken atdifferent angles (HD images recommended). These images are uploaded to the server for processing, which normally takes a few minutes. The program uses photogrammetric technology to measure distances between objects yielding a 3D model; in other words, image processing is performed by stitching a plain seam with correct sides together. However, the software may ask the user to select pointsfor connection that could not be automatically determined through online processing[143, 144].



(a) Apple images: a series of 48 images(336 MB) are used

(b) Face images: a series of 28 images(200 MB) are used

(c) Statute images: a series of 51 images (366 MB) are used

Figure 7.8(a, b and c): The selected series of images for 3D reconstruction from multiple points.

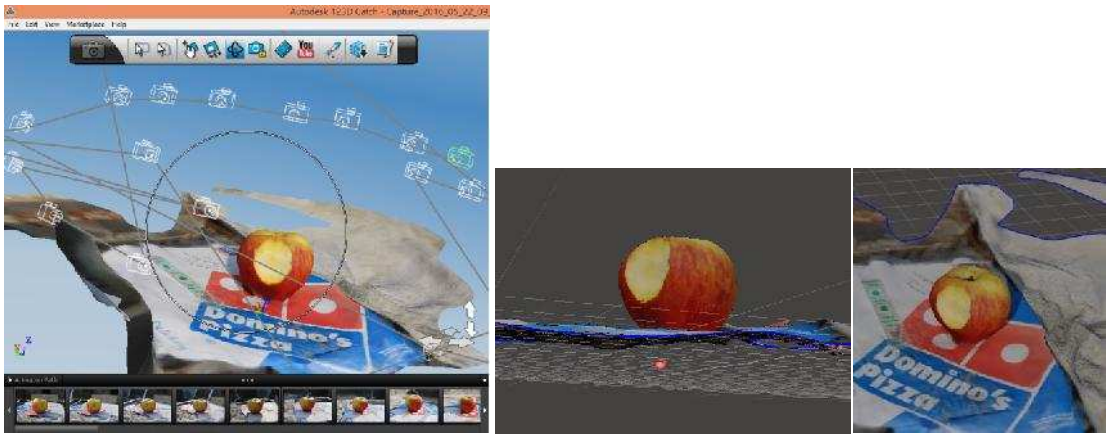
Table 7.5: shows our compression algorithm results

Multiple 2D images	Total Original BMP file size (MB)	Total original size as JPEG format at 100% High-Quality (MB)	Total Compressed size by our Approach (KB)	Compression Ratio	Quantization factor According to the layers: [R, G, B] Block size	2D RMSE
Apple	336	52.4	929	99.7%	[40,40,40] 16x16	9.5
Statue	366	58.5	916	99.7%	[20,30,30] 64x64	14.35
Face	200.7	45.5	784	99.6%	[11,30,30] 16 x 16	5.1

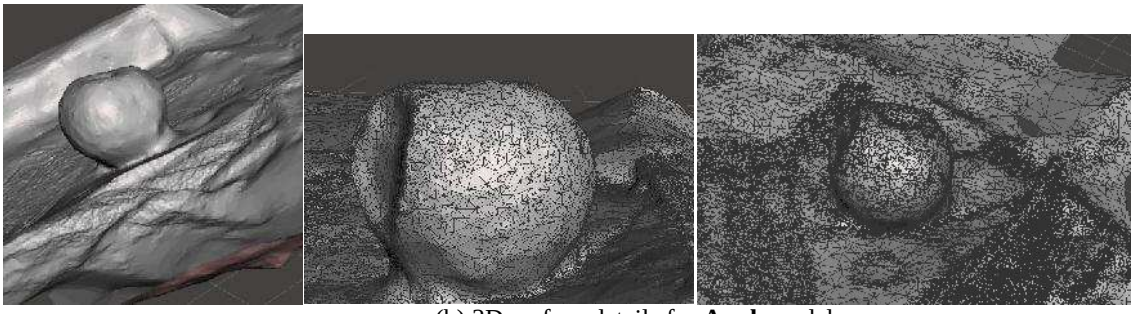
Table 7.6: Comparison with JPEG and JPEG2000 techniques

Multiple 2D images	Compressed size (KB)	2D RMSE			3D RMSE		
		Our Approach	JPEG2000	JPEG	Our Approach	JPEG2000	JPEG
Apple	929	9.5	6.58	FAIL	13.93	12.61	FAIL
Statue	916	14.35	13.81	FAIL	13.67	12.0	FAIL
Face	784	5.1	3.39	FAIL	14.73	12.35	FAIL

A comparative analysis is performed as follows. We compress a series of images using our method, JPEG and JPEG2000 and upload to the 123D Catch server for 3D reconstruction. We then analyse the quality of the returned 3D mesh obtained from the three compression methods. Figure 7.8 shows an original series of 2D images and Tables 7.5 and 7.6 depicts the compressed sizes by our approach compared with the other two techniques. Table 7.6 shows our approach and JPEG2000 to a maximum compression ratio, while the JPEG technique failed to reach that same compression ratio. Decompressed 2D images by our approach converted into a 3D surface by 123D Catch are depicted in Figures 7.9, 7.10 and 7.11. The JPEG2000 images converted into a 3D model are depicted in Figure 7.12.

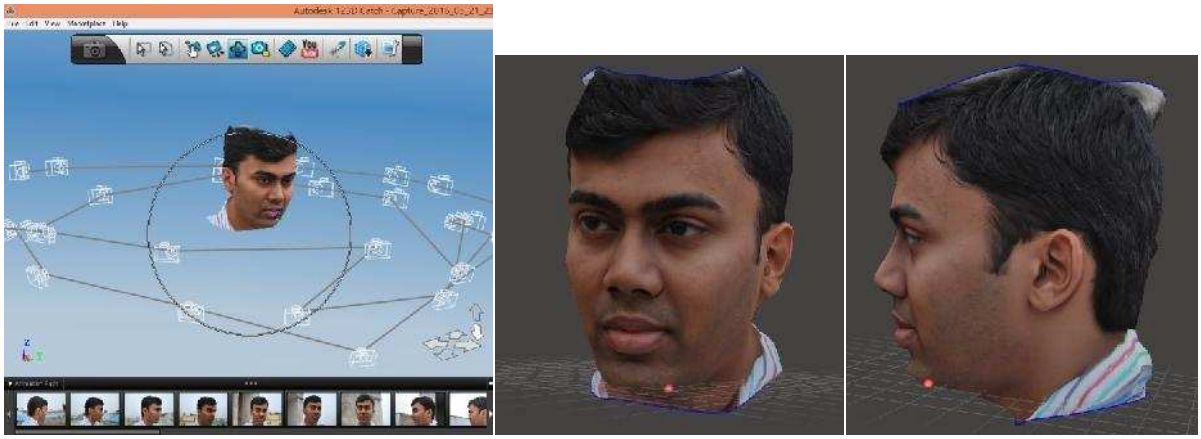


(a) Autodesk 123D Catch converts48 **Apple**imagesintoa 3D model

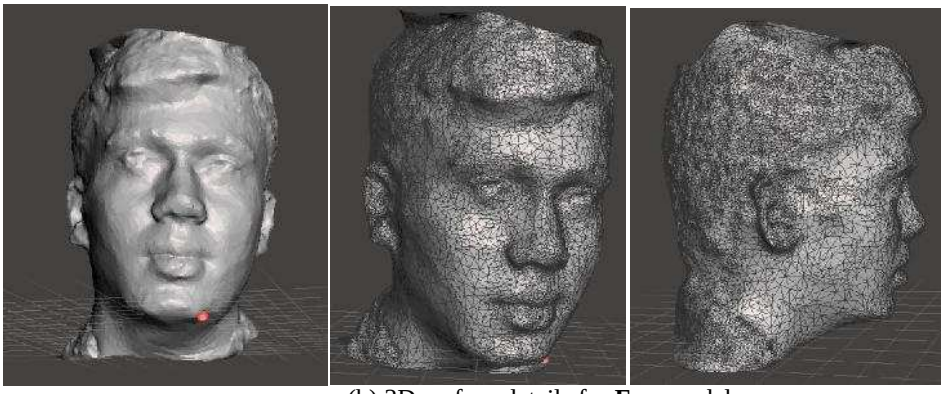


(b) 3D surface details for **Apple**model

Figure 7.9: (a, b) 3D model for series of 48 Apple images decompressed by our approach, average 2D RMSE for all decompressed images=9.5, total compressed size=929 KB. Theoretically, the achieved compression ratio for 3D mesh is 99.7% for connectivity and vertices.



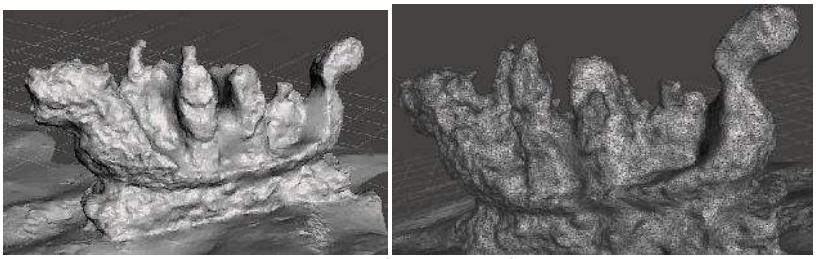
(a) Autodesk 123D Catch converts 28 **Face** images intoa 3D model



(b) 3D surface details for **Face**model

Figure 7.10: (a, b) 3D model for series of 28 FACE images decompressed by our approach, average 2D RMSE for all decompressed images=5.1, total compressed size=784 KB. Theoretically, the achieved compression ratio for 3D mesh is 99.6% for connectivity and vertices.

(a) Autodesk's 123D Catch converts 51 **Statue** images into a 3D model



(b) 3D surface details for **Statue**model

Figure 7.11: (a, b) 3D model for series of 51 Statue images decompressed by our approach, average 2D RMSE for all decompressed images=14.35, total compressed size=916 KB. Theoretically, the achieved compression ratio for 3D mesh is 99.7% for connectivity and vertices.

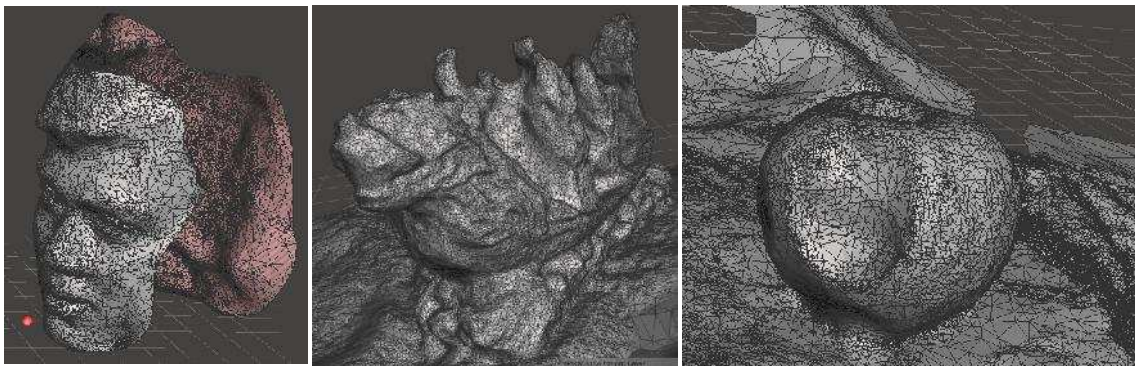


Figure 7.12: Results for JPEG2000: Left, the 3D Face model is degraded when coded by JPEG2000, (middle and right) the 3D models for Statue and Apple are successfully reconstructed from JPEG2000 images.

7.6. Conclusions

This Chapter has presented and demonstrated a new method for image compression and illustrated the quality of compression through 2D and 3D reconstruction, 2D and 3D RMSE and the perceived quality of the visualisation. The method is based on DCT together with the proposed differential process, minimization of high-frequency encoding and concurrent binary search algorithms. The results showed that our approach introduced better image quality at higher compression ratios than JPEG and equivalent perceived quality as JPEG2000. Furthermore, the proposed method can more accurately reconstructthe 3D surfaces at higher compression ratios than both techniques, i.e. in this respect it is also superior to JPEG2000. On the other hand, the methodis more complex than JPEG2000 and JPEG. The most important aspects of the method and their role in providing high quality image with high compression ratios are identified as follows[158]:

1- The DCT can be applied to large block sizes ≥ 8, and the DC-components and AC-coefficients are separated into different matrices by the proposed method and coded separately.

2- Since the AC-coefficients contain a large number of zeros, we applied a new methodto eliminate zeros and keep nonzero data. The process keeps significant information while reducing data up to 75%.

3- The Matrix Minimization algorithm, used to reduce high-frequencycomponents produced a Minimize-Arrayby replacing each three values from the AC-coefficients by a single floating-point value. This process reduces the coefficients leading to increased compression ratioswith faithful decoding.

4- The concurrent binary search algorithm represents the core of our search algorithm for finding the exact original data from the minimized-arrayand depends on the organised key values and the availability of the unique data. The efficient implementation of Visual C++ code allows the concurrent algorithms to recover the AC-coefficients in a few microseconds.

5- The key values and unique data are used for coding and decoding an image, without this information images cannot be recovered. This is an important point as a compressed image is equivalent to an encrypted image that can only be reconstructed if the keys are available. This has applications into secure transmission and storage of images and video data.

6- Our proposed image compression algorithm was tested on true colour and YCbCr layered images at high compression ratios. Additionally, the approach was tested on images resulting in better 3D reconstruction than JPEG2000 and JPEG.

7- The experiments indicate that the technique can be used for real-time applications such as 3D data objects and video data streaming over the Internet.

Chapter 8
3D Geometry and Connectivity Compression with Concurrent Fast Matching Search Decompression

8.1. Introduction

Polygonal meshes remain the primary representation for visualization of 3D data in a wide range of industries including architecture, geographic information systems, medical imaging, robotics, entertainment, and military applications among others. Because of its common use, it is desirable to compress polygonal meshes and exchanged over computer networks to reduce storage and reduce transmission time. 3D files encoded by Wavefront's OBJ file format are commonly used to sharing models due to its clear simple design. Normally, each OBJ file contains a large amount of data (e.g. vertices and triangulated faces, face normal directions) describing the mesh surface and other parameters such as illumination.

In Chapter 2 we briefly described previous work on mesh compression (geometry and connectivity). First we introduced Deering's work focused on geometry compression for data communication between the CPU and a graphic adapter [63]. Also, Rossignac and Taubin described the Topological Surgery (TS) method, which is a compression scheme used for maintaining manifold triangular mesh [67]. TS has been extendedfor use in compressed file format to encode VRML, and now it is also used in the MPEG-4 standard compression[145].

MPEG-4 coding is based on Topological Surgery and Progressive Forest Split Scheme (FPS)which is a new improved scheme for compression and transmission of 3D meshes in progressive form[146,147]. Many algorithms have beenproposed and incorporated into the standard which now supports the encoding of any polygonal mesh (also non-manifolds)[148,149].Additionally, there is no loss in the connectivity of triangle faces, and no duplication of geometry and property data related with vertices (x,y,z).The MPEG-4 standards are the state-of-the-art [150].

In this Chapter, we deal with direct compression of 3D data structures by Matrix Minimization algorithm, which is called Geometry Minimization algorithm used to compress mesh data (vertices and triangle faces). First, each vertex consisting of (x,y,z) coordinates are encoded into a single value by the GM algorithm. Second, triangle faces are encoded by computing the differences between two adjacent vertex locations, and then coded by the GM algorithm followed by arithmetic coding. We tested the method on large data sets achieving high compression ratios over 90% while keeping the same number of vertices and triangle faces as the original mesh. The decompression step is based on a Concurrent Fast Matching Search Algorithm (CFMS) to recover the structure of the 3D mesh. A comparative analysis of compression ratios is provided with several commonly used 3D file formats such as

MATLAB, VRML, OpenCTM and STL showing the advantages and effectiveness of our approach.

This Chapter is organized as follows: Section 8.2 introduces geometry coding and describes the proposedGeometry Minimization applied to vertex data. Section 8.3 describes lossless coding of mesh connectivity bythe GM-Algorithm, while section 8.4describes theConcurrentFast Matching Search algorithm (CFMS) used to reconstruct vertices and triangulated faces. Section 8.5 describes experimental results with a comparative analysis followed by conclusions in Section 8.6.

8.2. TheGeometry Minimization Algorithm (GM-Algorithm)

The algorithm starts with a lossy quantization process to convert vertices to integer values (because our approach works on integer numbers by truncating the vertices coordinates that can be represented by limited number of bits).

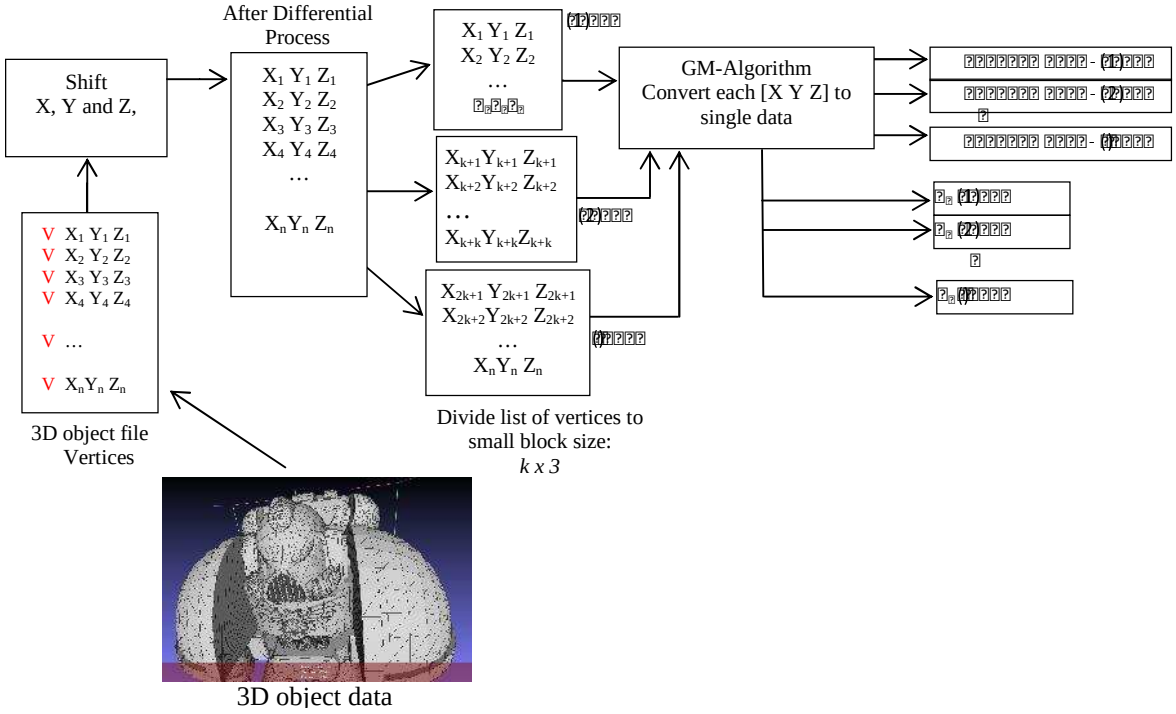


Figure 8.1, The GM-Algorithm applied to each block of vertices

A scale parameter α is used by the quantization process,it is used to move numbers from the exponent part to the mantissa part.In this way, the 3D structure can be reconstructed in the same units and scale as the original. The lossy quantization by α transforms each (*x*, y, z) coordinates into integers ranging from 0 to 2*B*–1, where *B* is the maximum number of bits needed to represent the quantized coordinates. Normally, 12bit – 16bit integers are sufficient to ensure geometric fidelity for most applications and most models. Thus, this lossy quantization step reduces the storage

cost of geometry from 96-bits to less than 36-bits. The quantization of vertices (*x*, *y*, *z*) is defined as:

$$V_{x,y,z} = f \qquad (V_{x,y,z}\alpha) \tag{8.1}$$

Where$\alpha \leq 10{,}000$.We reduced the number of bits for each vertex to less than 16-bit by calculating the differences between two adjacent coordinates for increased redundancy data and thus, making it more susceptible to compression. The differential process defined in Eq. (3.5) is applied toaxes X, Y and Z independently (cf. Figure 3.3)[42].

Once the differential process is applied to the vertices, the list of vertices is divided into blocks, and the GM-Algorithm is applied to each block of vertices (i.e. the vertex matrix from 3D object file is divided into *k* non-overlapping blocks) as illustrated in Figure8.1. The main reason for placing vertices into separate blocks is to speed up the compression and decompression steps. Each *k* block is reduced to an encoded data array. The GM-Algorithm is defined as taking three key values and multiplying these by three geometry coordinates (x, y, z) from a block of vertices which is then summed over to a single integer value (cf. Section 3.2.3). A compression key K_C is generated from vertex data (cf. Section 5.2.3 - key generator) as follows [137]. First, define *M* as a function of the maximum value in the data:

$$M = \max(V_X, V_Y, V_Z) + \frac{\mathrm{m}\quad (V_X, V_Y, V_Z)}{2} \tag{8.2}$$

Then define 3 compression keys as follows:

$$K_{C1} = r \qquad (0,1) \tag{8.3}$$

$$K_{C2} = (K_{C1} + M) + F \tag{8.4}$$

$$K_{C3} = (M * K_{C1} + M * K_{C2}) * F \tag{8.5}$$

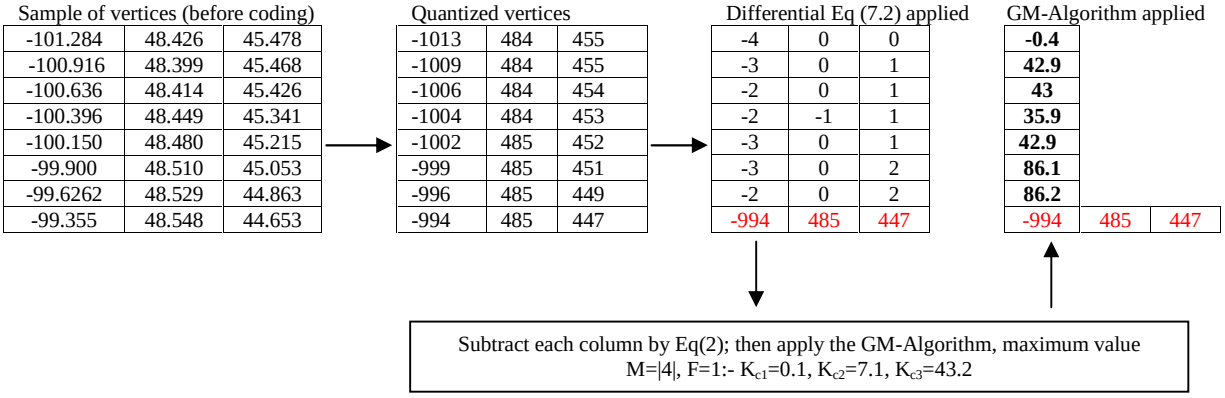
Where *F* is a positive factor multiplier, each vertex is then encoded as:

$$V(i) = V_x(i)K_{C1} + V_y(i)K_{C2} + V_z(i)K_{C3} \tag{8.6}$$

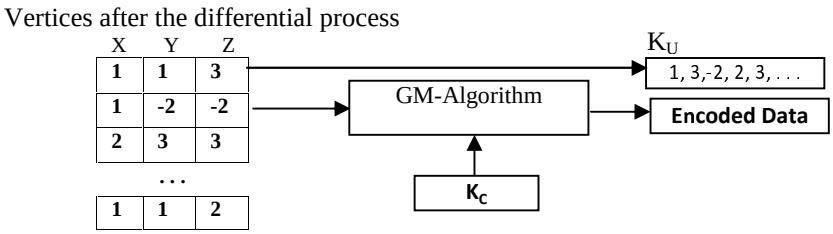
Figure 8.2(a) illustrates the GM-Algorithm applied asEq(8.6) to a sample of vertices. After applying the GM-Algorithm, the likelihood for each block of vertices is selected from which a *Ku* (i.e. Limited-Data –cf. Figure 3.5)is generated to be used in the decompression stage as illustrated in Figure 8.2(b) with a numerical example.

8.3. Connectivity Compression

Several algorithms have been developed to address the problem of compactly encoding the connectivity of polygonal meshes, both from theoretical and practical viewpoints. The short encoding of embedded graphs has been tackled as a theoretical problem, while compressing the incidence table of the triangle mesh in a 3D model as a practical problem.



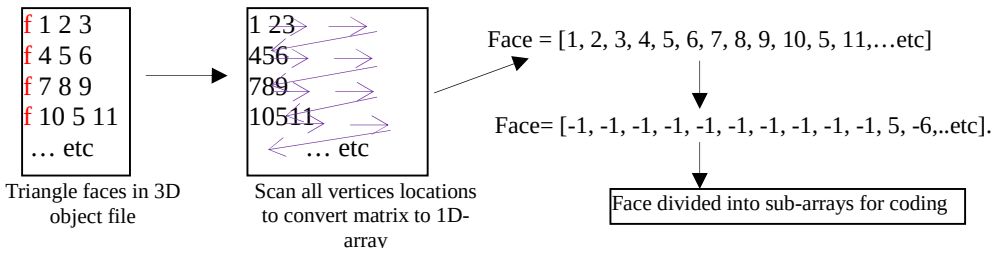
(a) Floating point vertices



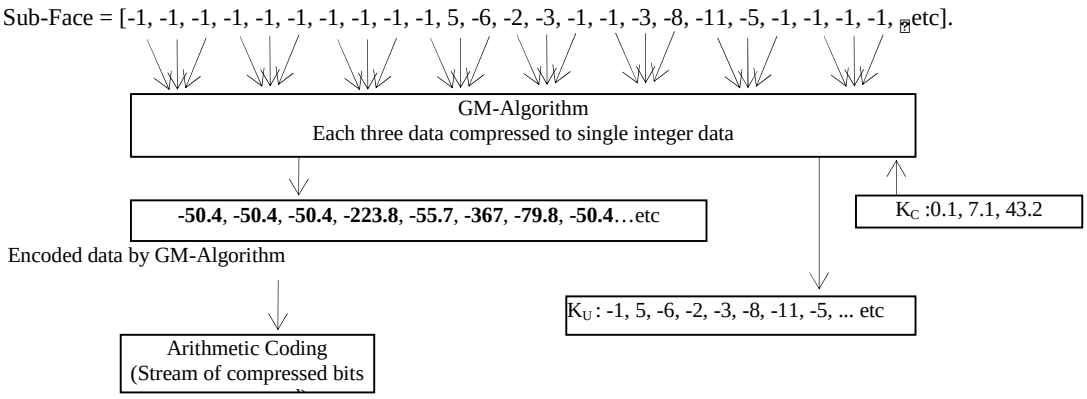
(b) Limited Data $K_u$computed from vertices X,Y and Z

Figure 8.2: (a): Sample of vertices compressed by GM-Algorithm, (b) The set of$K_U$valuesgenerated from a block of vertices

Triangulated meshes represent geometric connectivity. In a 3D OBJ file, each triangle is followed by reference numbers representing the location of the vertices in the 3D file. These reference numbers arearranged in ascending order in most 3DOBJ files. We refer to these as *regular triangles*. One of regular triangles' advantages is that they can be losslesscompressed ina few bits by applying a differential process (e.g. the differential processed definedby Eq. (3.5) applied to all reference numbers). The resulting 1D-array is divided into sub-arrays, and each sub-array encoded independently by the GM-Algorithm followed by arithmetic coding as illustrated in Figure 8.3.TheGM-Algorithm works in thesame way as applied to the vertices: three key values are generated and multiplied by three adjacent values which are then summed into singlevalue according to Eq. (8.6).



(a) Triangle Face are scannedrow-by-row



(b), 1D-array divided into sub-arrays, each sub-array encoded independently (See Figure 3.4)

Figure 8.3: (a) and (b): Lossless Triangle Mesh Compression by GM-Algorithm and Arithmetic Coding

8.4. Decompression Algorithm: Concurrent Fast Matching Search Algorithm (CFMS-Algorithm)

The decompression algorithm represents theinverseofcompression usingCFMS-Algorithm (cf. Section 7.4)to reconstruct vertices and meshconnectivity. First, theCFMS algorithm isapplied to encoded block of vertices to reconstruct the originalvertices as a point cloud. Second, theCFMS algorithm isapplied to each encoded sub-array resulting in the reconstructed triangle mesh sub-array.Thereafter, all the sub-arrays arecombinedto recover theincidence table of triangulated facesof the 3D model.Figure 8.4 shows the layout of the decompression algorithm.

The CFMS algorithm provides the means for fast recovery of both vertices and triangulated meshes, which has been compressed by three different keys (K_C) for each three entries. The header of the compressed file contains information about the compressed data namely K_C and K_U followed by streams of compressed encoded data. The CFMS algorithm uses a group of binary search algorithmworking concurrently (as mentioned in previous Chapter 7) and is illustrated through the following:

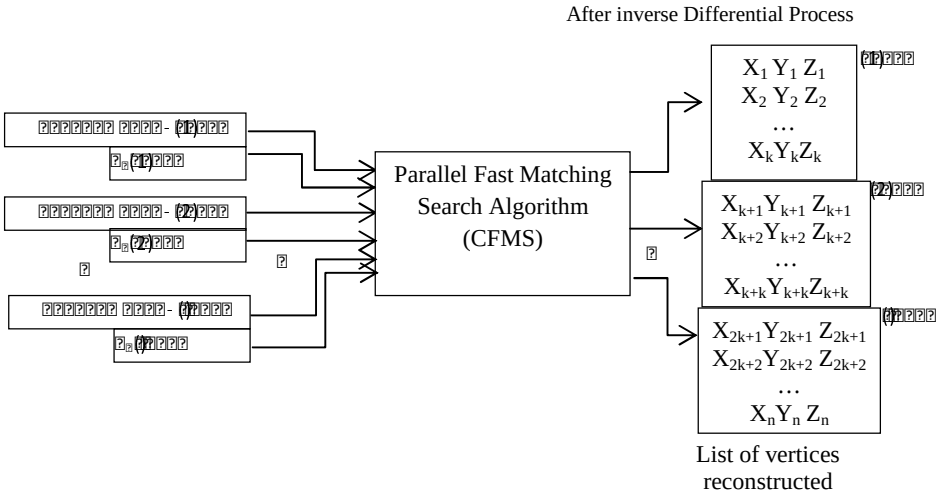
A) Initially, K_U is copied three times to separate arrays to estimates coordinates (x, x,z), that is $x_1=y_1=z_1$, $x_2=y_2=z_2$, $x_3=y_3=z_3$ the searching algorithm computes all possible combinations of x with $K_U(1)$, y with $K_U(2)$ and z with $K_U(3)$ that yield a result R-Array.As a means of an example consider that $K_U(1)=[x_1x_2x_3]$, $K_U(2)=[y_1y_2y_3]$ and $K_U(3)=[z_1z_2z_3]$. Then, Eq.(8.6) is executed 27 times to build R-Array, as described in Figure8.5(a). A match indicates that the unique combination of (x y z)are the coordinates of theoriginal vertex we are after.

B) A Binary Search algorithm is used to recover an original axes X, Y and Z[139]. The CFMS algorithm consists of k-Binray Search algorithms working concurrentlyto reconstruct k blocks of verticesin the list of vertices, as shown in Figure8.5 (in same way as triangle faces are decoded byk-Binary Search algorithms) (See Section 7.4).

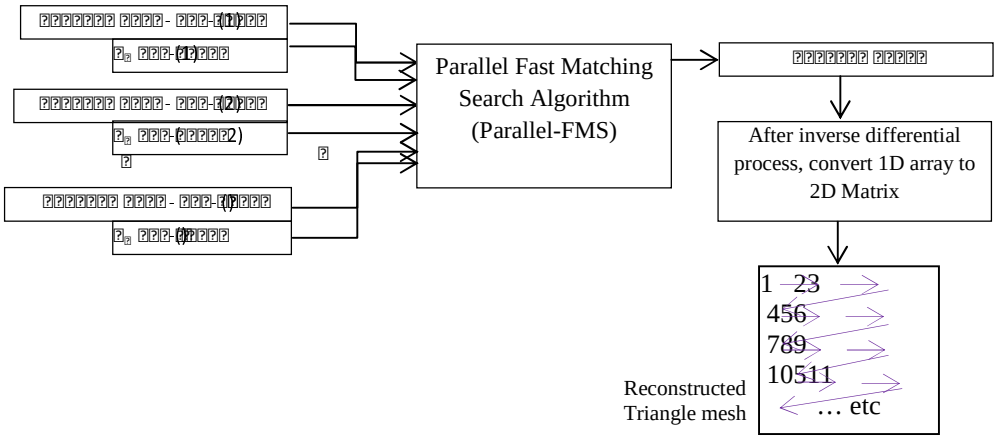
To decode triangle faces and vertices it is necessary to reverse the differential process of Eq. (8.2) by addition such that the original values are recovered. This process takes the last value at position *m*, and adds it to the previous value, and then the total adds to the next previous value and so on. The following equation defines the addition decoder [137].

$$A(i-1) = A(i-1) + A(i) \tag{8.7}$$

Where $i= m, (m-1), (m-2), (m-3),\ldots,2$



(a) Vertices (x,y,z) reconstructed



(b) Triangle mesh reconstructed

Figure 8.4: (a) and (b): Parallel-FMS Algorithm applied on encoded vertices and encoded triangle mesh

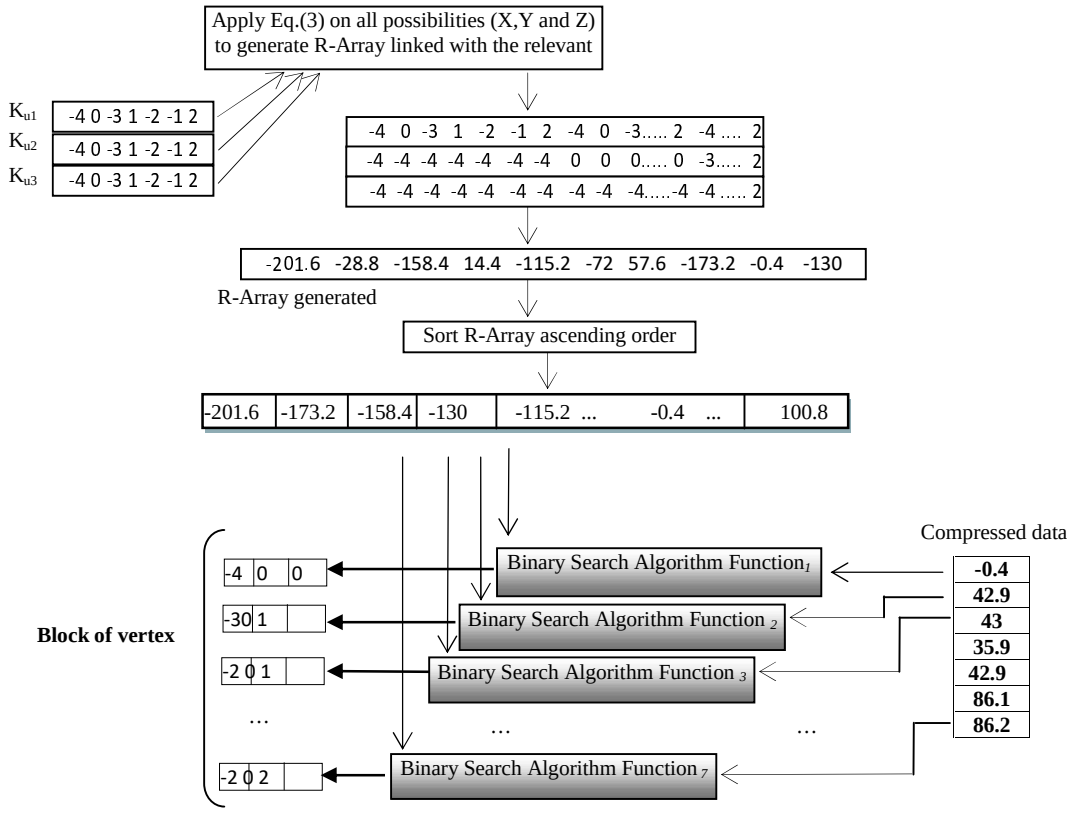


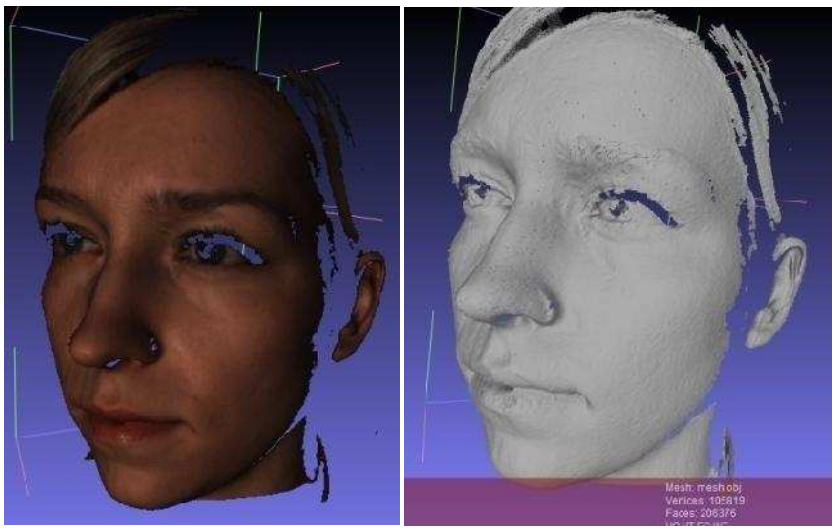
Figure 8.5: All Binary Search Algorithms run in Parallel to recover the sample of vertices, approximately at same time.

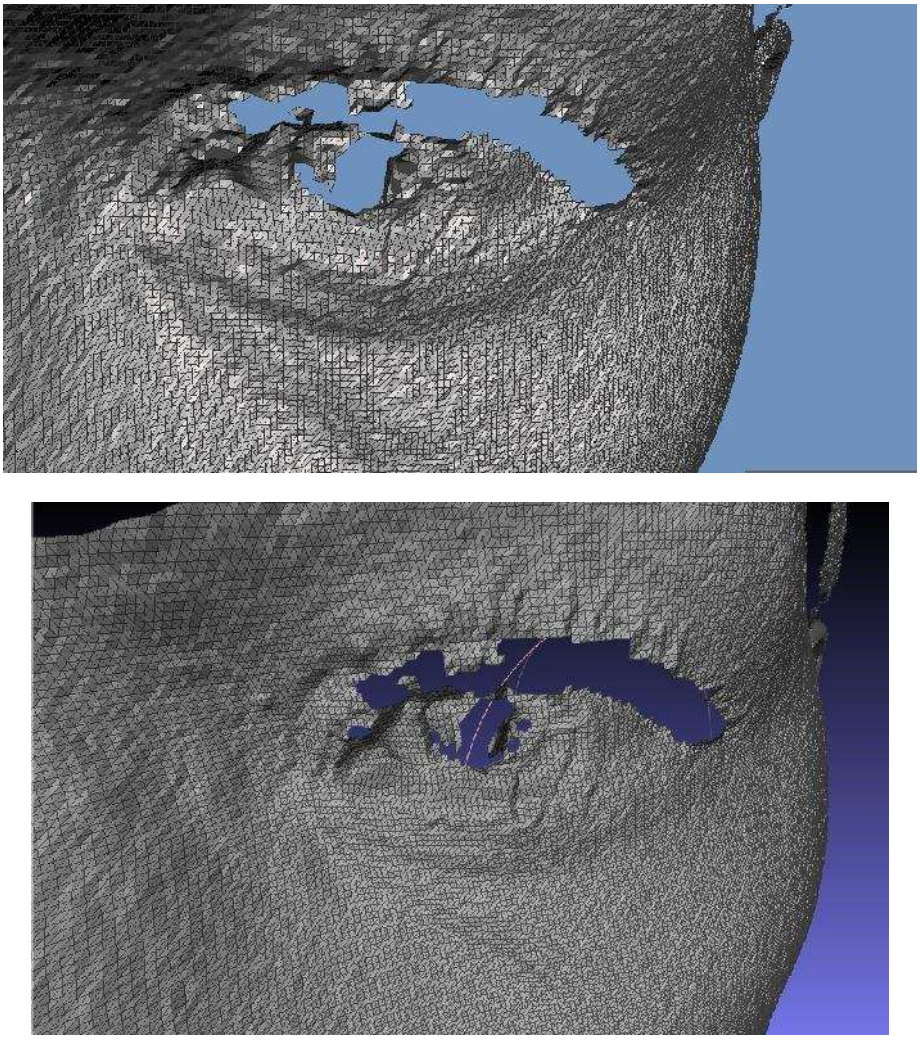
8.5. Experimental Results

Experimentsare implemented in MATLAB R2013a and Visual C++ 2008 running on an AMD Quad-Core microprocessor. We applied the compression and decompression algorithms to 3D data represented in OBJ file format. Data in OBJ file format can be generated by any software or device such as 3dsmax, CAD/CAM, 3D scanneror other. Table 8.1 shows our compression algorithm applied to each 3D OBJ file, and Figure 8.6 shows the visual properties of the decompressed 3D object data for 3D images respectively. Additionally, 3D RMSE are used to compare 3D original fileswith the recovered files.TheRoot Mean Square Error (RMSE) is used to refer to 3D mesh quality mathematically[119, 131]and can be calculated very easily by computing the differences between thegeometry of the decompressed and the original 3D OBJ files.

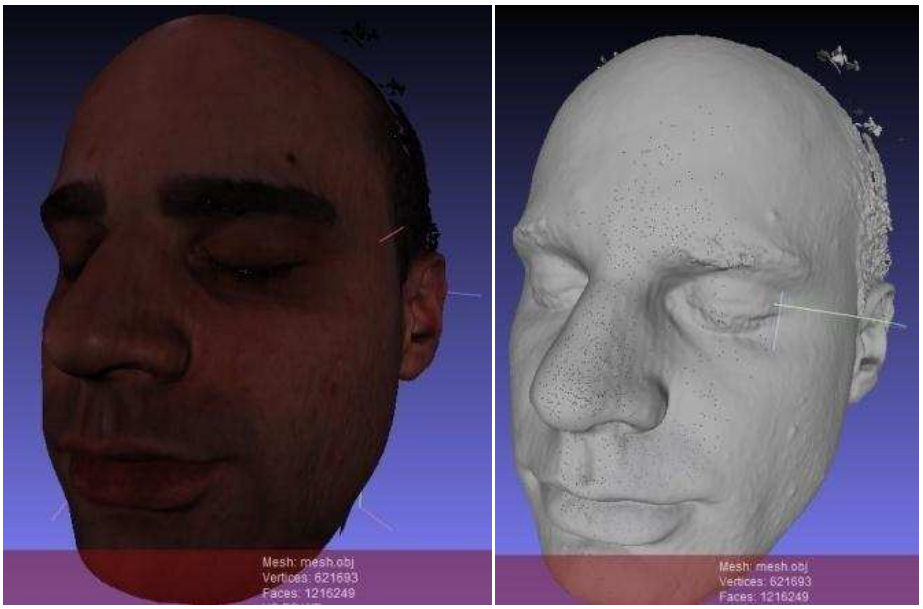
Table 8.1. 3D data compression results

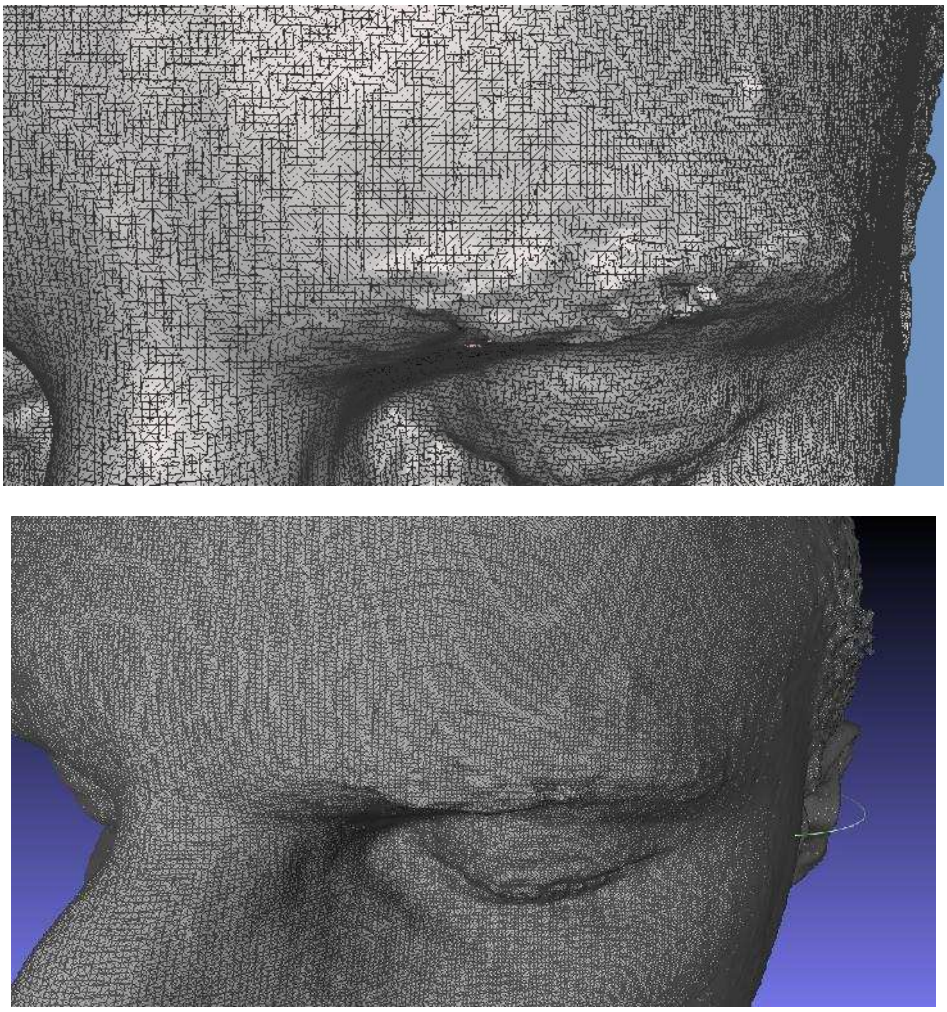
3D object Name	Original file size (MB)	Quantization value	Compressed file size (KB)	No. of Vertices (Compressed Size)	No. of Triangle faces (Compressed size)	3D RMSE (x y z)
Face1	13.3	10	213	105819 (187 KB)	206376 (26 KB)	0.288
Face2	96	10	3,700	621693 (1.8MB)	1216249 (1.9MB)	0.289
Angel	23.5	20	1,750	307144 (1.055MB)	614288 (715 KB)	0.288
Robot	1.5	400	88.9	23597 (56.3KB)	45814 (32.6KB)	0.289
Cup	0.057	2	3.5	594 (2.13 KB)	572 (1.36KB)	0.263
Knot	0.178	2	7.94	1440 (7.4 KB)	2880 (553 B)	0.027



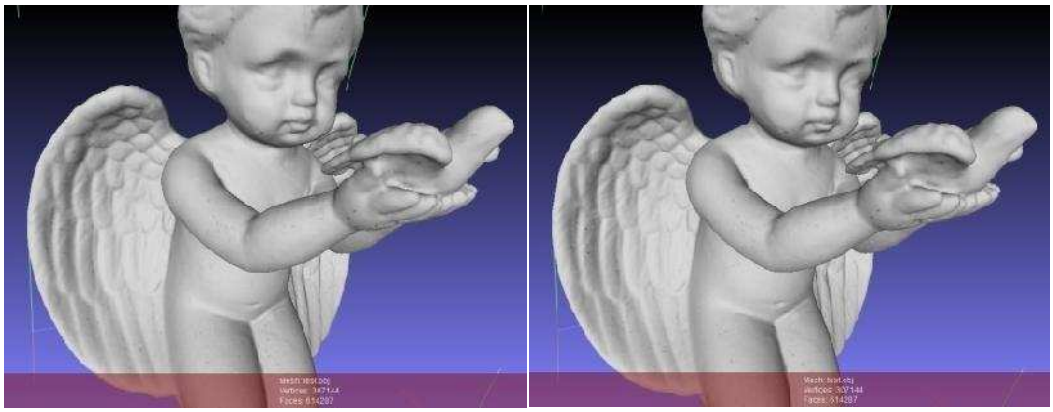


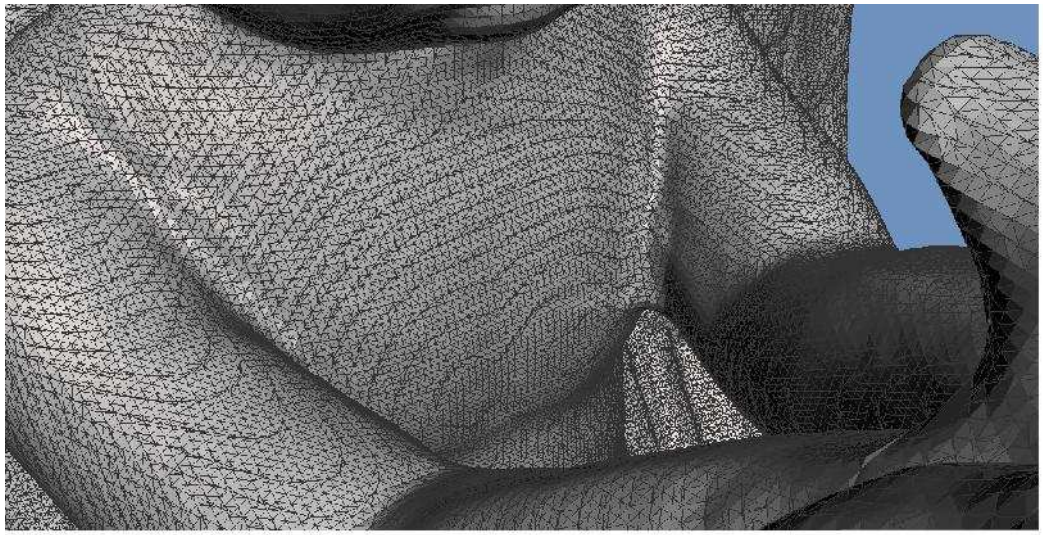
(a) (Top left) original 3D FACE1 object, (Top Right) reconstructed 3D mesh FACE1 without texture, compressed size: 213 KB, (middle) original 3D mesh zoomed by Autodesk application(bottom)reconstructed 3D mesh zoomed by Meshlab application.

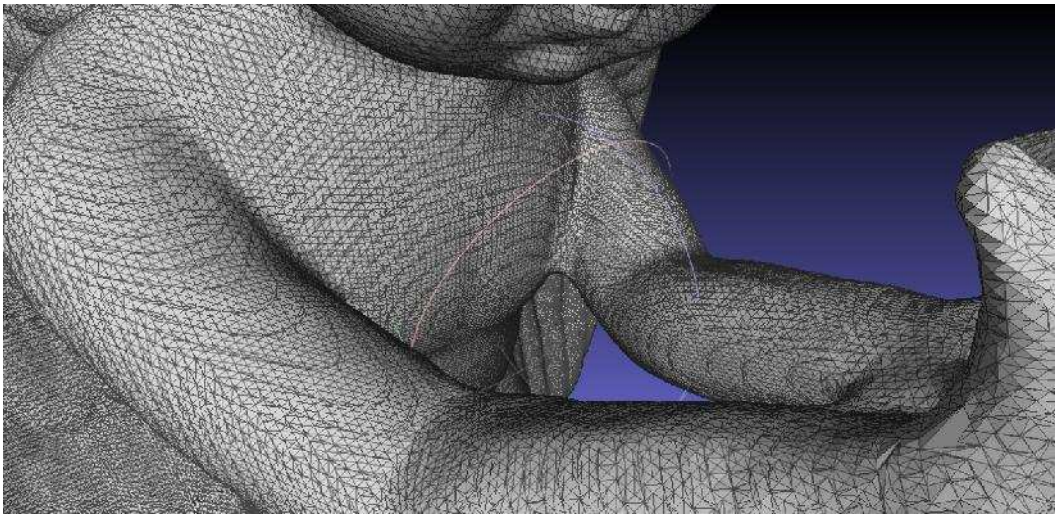




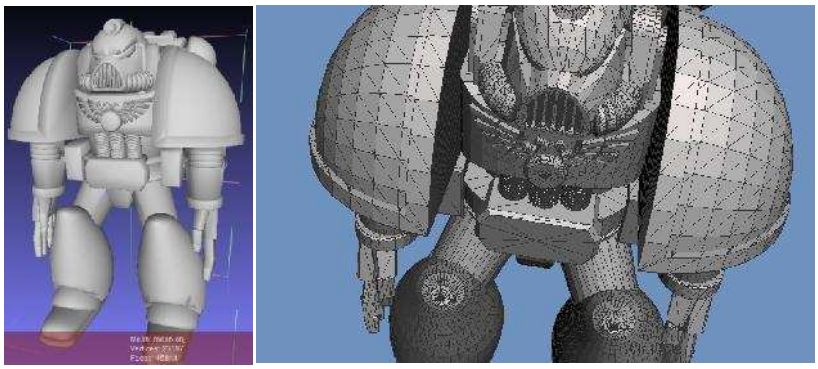
(b) (Top left) original 3D FACE2 object, (Top Right) reconstructed 3D mesh FACE2 without texture, compressed size: 3.7 MB, (middle), original 3D mesh zoomed by Autodesk application, (bottom) reconstructed 3D mesh zoomed by Meshlab application.

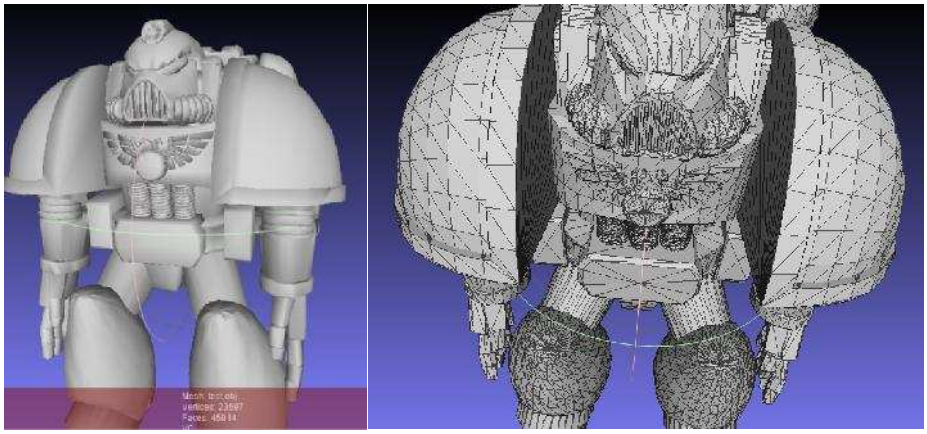




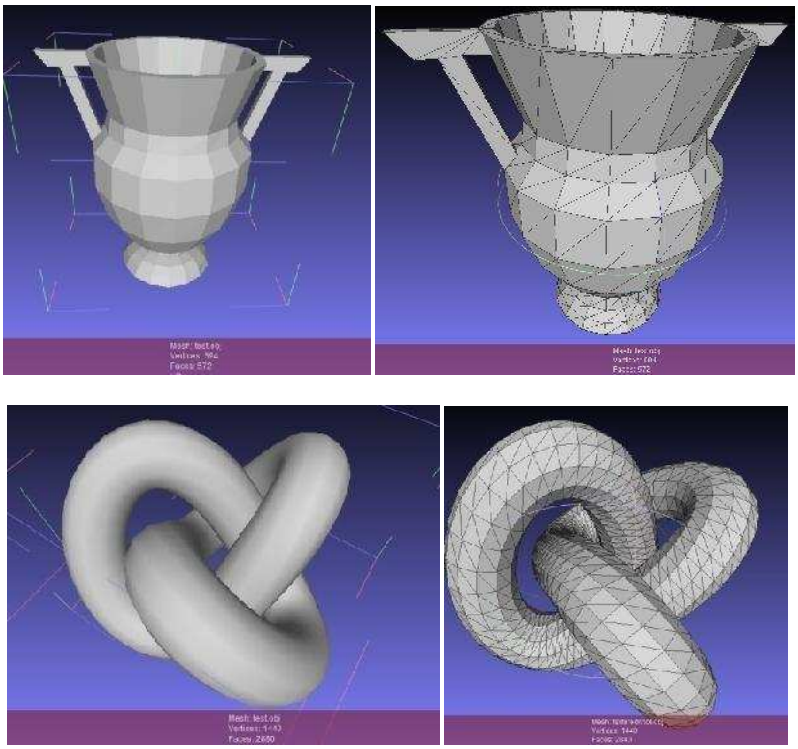


(c) (Top left) original 3D Angel object, (Right left) reconstructed 3D mesh Angel at compressed size: 1.75 MB, (middle) original 3D zoomed by Autodesk application, (bottom) reconstructed 3D mesh zoomed by meshlab application.





(d) (Top) original 3D Robot object, (bottom) reconstructed 3D mesh Robot,at compressed size: 88.9KB



(e) (Top) original and reconstructed 3D mesh cup, at compressed size: 3.5 KB, (bottom)original and reconstructed 3D mesh Knot, at compressed size: 7.94 KB

Figure 8.6: (a – e) shows decompressed 3D objects by the proposed algorithms

Tables8.2 and 8.3 show a comparisonofthe proposed method with the 3D file formats VRML, OpenCTM and STL. Wealso used a new simple fileformat referred here as MATLAB format. This format savesthegeometry, texture and triangle faces as lossless data, in separated matrices and all the matrices are collected into a single file. We investigate this format obtaining compression ratios over 50% for most of 3D OBJ files. In comparison, our approach uses aunique format to compress 3D files up to98%; this is mostly dependent on the triangle face details.

Table 8.2. Our approach compared with other encoding 3D data format according to compressed size

3D object Name	Original file size	**Proposed Algorithm**	MATLAB format	VRML format	OpenCTM	STL
Angel	23.5 MB	**1.75MB**	5.31 MB	23.2 MB	1.92 MB	29.2 MB
Face1	13.3 MB	**213 KB**	4.04 MB	9.19 MB	808 KB	9.84 MB
Face2	96MB	**3.7MB**	23.3 MB	47.7MB	3.7MB	57.9MB
Robot	1.5 MB	**88.9 KB**	449 KB	1.7 MB	151 KB	2.18 MB
Cup	57 KB	**3.5 KB**	12 KB	25.2 KB	3.24 KB	28 KB
Knot	178 KB	**7.94 KB**	23.6KB	95.4KB	14.2KB	140 KB
Total Compressed Size		**5.75 MB**	33.12 MB	81.9 MB	6.57 MB	99.28 MB
Mean Compression Ratio		**95.7 %**	**75.3 %**	**39.4%**	**95.1 %**	**26.2 %**

Table 8. 3. Our approach compared with other encoding 3D data format according to 3D RMSE

3D object	**Proposed Method**	MATLAB	VRML	OpenCTM	STL
Angel	**0.288**	0	0.0002	44.86	46.32
Face1	**0.289**	0	0.00021	64.79	42.05
Face2	**0.288**	0	0.000109	82.23	43.44
Robot	**0.289**	0	0	0.0587	0.137
Cup	**0.263**	0	0.00000075	37.7	39.2
Knot	**0.027**	0	0.000105	47.65	12.62

8.6. Conclusion

This research has presented and demonstrated a new method for 3D data compression and compared the quality of compression through 3D reconstruction, 3D RMSE and the perceived quality of the 3D visualisation. The method is based on minimization of geometric values to a stream of new integer data by theGM-Algorithm.Meshconnectivity is partitioned into groups of data,whereeach group iscompressed by theGM-Algorithm followed byarithmetic coding. We note that some of the existing 3D file formats do not efficiently encode geometry and connectivity,as a simple format developed in MATLAB showedhighercompression ratios than STL and VRML. The results show that our approach yields high quality encoding of 3D geometryand connectivity with high compression ratios compared toseveral standard 3D data formats. The slight disadvantage is a larger number of steps for decompression, leading to increased execution time at decoding stage, making our approach slower than the compared compression methods. Further research includes investigation of methods to speed up decoding, possibly by sorting the*R-Array*entries by frequency, and a comparative analysis with a larger number of 3D file formats and compression techniques.

Chapter 9

Conclusion and Future Work

9.1. Outcome of the Thesis

This thesis has investigated and demonstrated novel methods of data compression. We have focused on 2D images and 3D data structures, but the methods are generic to be applied to any kind of data. The techniques are based on applying a transformation (DCT/DWT/DST) and build matrices of low and high frequency components to which quantization can be applied. A main step of the method is the Matrix Minimization algorithm which converts groups of data to a single value through a set of data-dependent keys. A set of unique data is kept in the header of the compressed file to allow recovery of the original data. The unique data can be used as an encryption key as without this information, the file cannot be decompressed. This opens many opportunities for applications in the security domain.

A survey of previous work on 3D mesh compression is presented in Chapter 2. Chapter 3 introduces the Matrix Minimization algorithmwhich isapplied to 2D BMP images and results show that the algorithmis very efficient at compressing high-frequency data (from the transformed matrices). The thesis demonstrates through many experiments that theDCT and DWT can successfully work together to increase the number of high-frequency and decrease correlated low-frequency data. On these lines, we have demonstrated techniques to increase the number of high-frequency data which are more susceptible to compression without significant deterioration in quality. For instance, theDWT is used as a first transformation todivide an image into four sub-bands,followed byDCT applied to correlated low-frequency domain resulting in a split of this domain into new low-frequency and high-frequency domains. This double transformation increases high-frequency data (i.e. the matrices are ready for reduction by the Matrix Minimization algorithm).Finally, our work successfully compressed 2D images at high compression ratiosup to98% reducing the data to 2 orders of magnitude.Furthermore,we demonstratedalgorithms capable to reconstruct 2D structured light images used for 3D reconstruction with decompression based on a Sequential Search methodto reconstruct the original data.

In chapter 4, the JPEG algorithm is combined with a single level DWTtoreduce the number of transformation steps (i.e. the transformation steps used in Chapter 3 are more complex than the onesused in Chapter 4).Also,the searching methodat decompression stage was upgraded based on a multi sequential search algorithm to speed up the process. The drawback ofthis method is the complexity of the LSSalgorithm, leading to increased execution time due to the interactive nature of the method.

In chapter 5 we discussed anenhanced compression algorithm by replacing random key values with organized key values that would be more helpful incompressing high frequency data. Also, zero values were removed from the high-frequency domains leading to

increasedcompression ratios. Additionally, we designed a new searchalgorithm,Fast-Matching-Search (FMS-algorithm)to be used atdecompression stage. The algorithm is designed to reconstruct a large matrix in a few microseconds and depends on a binary search algorithm.It is demonstrated that the algorithm ismuch faster than the previous search algorithms presented in Chapters 3 and 4.

In Chapter 6 we investigated the use oftwo transformations: DST with DCT applied tothe same image. The method applied a DCT to each row of an image followed by DST applied to each column. The final transformed image is quantized to increase high-frequencies followed by dividing it into three regions. Two of theregions withhigher frequency domains are subject to the Matrix Minimization algorithm. The FMS-algorithm is used for decompression as described in Chapter 5. Results demonstrate the effectiveness of the approach both in terms of compression ratios and image quality. Additionally, there are fewer transformation steps than the approaches described in previous chapters.Furthermore, natural pictures are compressed up to 97% with high quality level of details. However, the complexity of thealgorithm means that it is slowerthan both JPEG and JPEG2000.

In chapter 7 we applied a single DCT over the entire image for compression with concurrent multi binary search algorithm for decompression. The advantage of this method at decompression stage is that it isfaster than a single binary search algorithm (FMS-algorithm).Another advantage is that it uses different sizes of blocks (8x8, 10x10 or 16x16 or any size up to 64x64) during DCT transformation.This algorithm was subject of a UK patent application which was accepted by the UK Patent Office and published as international PCT in September 2016[160]. Patent protection is now in the process to be extended to Europeand US.

3D object compression based on Geometry Minimization algorithm (i.e. Matrix Minimization algorithm applied to geometry data)is demonstratedin Chapter 8. The compression algorithm is applied tothe list of vertices (x, y and z) to convert them into a single value without any transformation (i.e. without use DCT, DST or DWT). Similarly, thealgorithm is applied to the list oftriangulated faces for compression. Consequently, the results showaccurate 3D reconstruction at higher compression ratios up to 97% within the set target of reduction of data by 2 orders of magnitude. On the other hand, the disadvantage is the longer compression and decompression times; execution times for 3D meshes are longer than for 2D imagesbecause a3D mesh containslargenumber of vertices and connectivity data. Furthermore, all data are compressed by the Geometry Minimization algorithm whose decompression step is computationally demanding.

Table 9.1 shows acomparative analysis for each method as described in separate Chapters of the thesis. The algorithms run on a computer withmicroprocessor: Quad-core AMD -2.4GHz, SDRAM: 6GBytes and Hard Disk: 640GBytes.

Table 9.1: Acomparative analysis between the proposed and demonstrated methods

Chapter	Algorithm C:Compression D:Decompression		Complexity (No. of steps)	Number of Discrete Transforms	Average Execution time (MM and Searching Method)	Average Compression Ratio
3	C: D:	DWT+DCT+MM SS-Algorithm	10	2	3—5 min	95% - 97%
4	C: D:	DWT+JPEG+MM BSS-algorithm	9	2	10—122 s	96% - 99%
5	C: D:	DWT+DCT+MM FMS-Algorithm	13	2	0.062—10 s	96% - 99.5%
6	C: D:	DCT+DST+MM FMS-Algorithm	7	2	0.1—10 s	95% - 99.4%
7	C: D:	DCT+MM CFMS-Algorithm	7	1	0.062— 10 s	95% -99.5%
8	C: D:	MM CFMS-Algorithm	5	0	3min – 2h	75% - 96%

9.2. Discussion of Results

It is demonstrated that the proposedcompression algorithms are very attractive for 2D image compression used in3D reconstruction from structured light.Theyexecute fastand are able to accurately compress 2D images at higher compression ratios than JPEG and JPEG2000up to 99%as shown inChapters4 and 5. Additionally, we used anenhanced Matrix Minimization compression algorithm (Chapter 6 and7) to compress a series of 2D images used for 3D reconstruction. The results show that thecompression and decompression algorithms worked successfully onthe series of 2D images with an average compression ratio of 99%. Also, we compared our approach with both JPEG and JPEG2000techniques. The comparative analysis presented in the various chapters of the thesis show thatour proposed methodsis superior to JPEG for all type of images,while the JPEG2000 method is approximately equivalent or slightly better than our approach for2D natural images. On the otherhand, our approach is shown to be superior to JPEG2000 concerning compression ratios and quality of reconstructed 3D mesh when applied to structured light images used in 3D reconstruction.

While in 3D mesh compression (Chapter 8) our approach is slow, the quality of the reconstructed mesh is high when applied in a lossy fashion and perfect, as expected, in a lossless way.The proposed 3D mesh compression compares favourably with other techniques related to 3D mesh compression (i.e. STL, VRML and OpenCTM) in terms of file sizes. However, the execution time for 3D meshes compression is slower than the comparedtechniques.

It is important to emphasize that theMatrix Minimization algorithmcanbe used inlossless and lossy compression. The algorithm itself is lossless, as it combines three data items into a single value. After decompression, the exact data is recovered which is RMSE=0 between the original and recovered data. In the experiments demonstrated in this thesis, the Matrix Minimization algorithm is used to compress discrete transformeddata normally in a lossy way. This means that the original image data aresubject toa lossy transformation followed by Matrix Minimization (i.e. lossless compression) of some data matrices.

9.3 Contributions to Knowledge

The main contributions to knowledge are discussed as follows.

- In this thesis, we proposeda new coding method for data compressionwhich is to convert three items of data into a single value. Thisnew algorithm called Matrix Minimization algorithm is experimentally demonstrated to compress 2D image data and 3D structures. Theoretically, after a discrete transformation step,a2Dimage contains large numbers of zeros spread over the image. The discerning feature of our proposed approach is to reduce the image size and to reduce the number of zeros. The experimental results in this thesis showed that our proposed method can increase compression ratios for an image up to 99.7% without much degrading image details (as described in Chapter 3,4,5,6 and 7).

- An importantfeature of the Matrix Minimization algorithm is that the algorithm itself is lossless and can be used tocompress 3D mesh data without the need for a prior discrete transformation (either lossy or lossless). A 3D mesh surface consists of vertices and triangle faces and our novel approach proved the ability to directly compress each of vertices and triangles independently according to the results showed in Chapter 8.

- Once the Matrix Minimization algorithm is applied, it is not possible to directly recover the original data items from mathematical expressions, as this is a mathematically under-determined problem. Various types of lossless decoding search methods are suggested and successfully worked torecover the compresseddata. In chapter 3, a Limited Sequential Search algorithm (LSS-Algorithm) was the first iterative method proposed in this thesis and it is shown that it can recover image data but it is not fast. Other novel methods were proposed and demonstrated such as the Fast-Matching Search algorithm (FMS) designed as a fastest decoding method (as described in Chapter 5). We verified that the FMS-algorithm can decode large amounts of matrix data in a few micro-seconds. This compares very favourably with the LSS-algorithm that requires approximately 5-10minutes todecode the sameamount of data.

- The thesis demonstrates methods to generate compression keys that depend on the data to be compressed; a different file would generate a different set of keys. These keys range from 24—192 bits. Furthermore, to be able to recover the compressed data by the Matrix Minimization algorithm, we proposed to keep in the file header the set of unique data (also referred to as probability data, or space domain data in the thesis). The set of keys together with unique data can be defined as an encryption key for the file, as without this set of data, the file cannot be decompressed.

- Finally, the methods presented in this thesis can be defined as a per-file compression-encryption. Each file will generate their own set of compression keys and their own set of unique data. This feature enables application in the security domain for safe transmission and storage of data. This is especially relevant in a cloud environment.

9.4. Future Work

The work presented in this thesis is by no means completed. There are further transformations and combinations thereof that have not been used in connection with the Matrix Minimization algorithm. Furthermore,the lossless Matrix Minimization algorithm can encode different types of data for example audio and general text files. The following points refer to the future research.

A) Partition the image into different non-overlapped blocks and each block is transformed by theDiscrete Fourier Transformation (DFT). This will generate two different matrices Real data and imaginary data, each of these matrices has their own quantization and each of these blocks are coded by the Matrix Minimization algorithm. The real differences will be apparentwhen we compare this new work with thepublished work in this thesis.

B) Exploit the Set Partitioning in Hierarchical Tree (SPIHT), as this algorithm embedshigh frequency and quantization depending ontheset number of bits and number of DWT levels. For this reason, the final transformed matrix is suitable for coding by theMatrix Minimization algorithm.

C)Future work on 3D mesh compression involves compressing vertices and using the Fast-Matching-Search algorithm toestimate triangle connectivity between three vertices, this approachdepends on minimum error between vertices.

D) Another approach to 3D mesh compression involvescompressing geometry alone(i.e. the set of vertices) and usethe Delaunay Triangulation method toestimate the connectivity between vertices.

E)The Matrix Minimization algorithm described in this thesis converts three data items to one. In future work, we planto convert six data items to a single value. This type of conversion needs two level keys, to convert three data items into one followed by converting each two-coded data to another new coded data set.

F) The most immediate priority is to develop novel methodsfor3D mesh compression. These compression algorithmswill consist of: 1) Compress each six vertices to one single number, keepingminimum information about originaldata.2) Compress each six of triangle faces using the same method.

REFERENCES

[1] T.C. Bell, J.G. Cleary, and I.H. Witten (1990). *Text Compression. Advanced Reference Series*. Prentice Hall, Englewood Cliffs, NJ.

[2] T.LiebchenandY.A.Reznik (2004). MPEG-4 ALS : An Emerging Standard for Lossless Audio Coding. InProceedingsoftheDataCompressionConference,DCC'04.IEEE,2004.

[3] D. Salomon (1998). *Data Compression: The Complete Reference*. Springer, 1998.

[4] Born, Gu¨nter (1995) The File Formats Handbook, London, New York, International Thomson Computer Press.

[5] R.M. Gray (1990). *Entropy and Information Theory*. Springer-Verlag, 1990.

[6] Parkinson's First Law (1957). *Work expands so as to fill the time available*, in Parkinson's Law and Other Studies in Administration, by Cyril Northcote Parkinson, Ballantine Books, New York.

[7] B.L. van der Waerden. (1985) *A History of Algebra*. Springer-Verlag.

[8] J.Rissanen (1978).ModelingbytheShortestDataDescription.*Automatica,14:465–471,1978.*

[9] Burrows, Michael, and D. J. Wheeler (1994) *A Block-Sorting Lossless Data Compression Algorithm*, Digital Systems Research Center Report 124, Palo Alto, CA.

[10] T. Berger (1971). *Rate Distortion Theory: A Mathematical Basis for Data Compression*. Prentice-Hall, Englewood Cliffs, NJ, 1971.

[11] B.P.Tunstall (1967).*SynthesisofNoiselessCompressionCodes.*Ph.D. thesis, GeorgiaInstitute of Technology, September 1967.

[12] J.R. Pierce (1961). *Symbols, Signals, and Noise—The Nature and Process of Communications*. Harper, 1961.

[13] T. Robinson. SHORTEN (1994): *Simple Lossless and Near-Lossless Waveform Compression*, 1994. Cambridge Univ. Eng. Dept., Cambridge, UK. Technical Report 156.

[14] M. Hans and R.W. Schafer (1998). AudioPak—An Integer Arithmetic Lossless Audio Code. In Proceedings of the Data Compression Conference, DCC '98. IEEE, 1998.

[15] J.A.Storer (1988). *Data Compression—Methods and Theory*. Computer Science Press, 1988.

[16] G. Held and T.R. Marshall (1991). *Data Compression*. 3rd edition, Wiley, 1991.

[17] G.K. Wallace (1991). The JPEG still picture compression standard. *Communications of the ACM, 34:31–44, April 1991.*

[18] W.B. Pennebaker and J.L (1993). *Mitchell. JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, 1993.

[19] M. Rabbani and P.W Jones (1991). *Digital Image Compression Techniques*, volume TT7 of Tutorial Texts Series. SPIE Optical Engineering Press, 1991.

[20] Carpentieri, B., M.J. Weinberger, and G. Seroussi (2000) .Lossless Compression of Continuous-Tone Images,*Proceedings of the IEEE, 88(11): pp 1797–1809.*

[21] K. Sayood (2003), editor, *Lossless Compression Handbook*. Academic Press, 2003.

[22] R.B. Arps and T.K. Truong (1994). Comparison of international standards for lossless still image compression. *Proceedings of the IEEE, 82:889–899, June 1994.*

[23] Burrows, Michael, et al. (1992) *On-line Data Compression in a Log-Structured File System*, Digital, Systems Research Center, Palo Alto, CA.

[24] M. Weinberger, G. Seroussi, and G. Sapiro (1998). *The LOCO-I Lossless Compression Algorithm: Principles and Standardization into JPEG-LS*. Technical Report HPL-98- 193, Hewlett-Packard Laboratory, November 1998.

[25] Y. Linde, A. Buzo, and R.M. Gray (1980). An algorithm for vector quantization design. *IEEE Transactions on Communications, COM-28:84–95, Jan. 1980.*

[26] M. Nelson and J.-L. Gailly (1996). *The Data Compression Book*. M&T Books, CA, 1996.

[27] X. Wu and N.D. Memon (1996). CALIC—A context based adaptive lossless image coding scheme. *IEEE Transactions on Communications, May 1996.*

[28] Mitchell, Joan L., W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, eds. (1997) *MPEG Video Compression Standard*, New York, Chapman and Hall and International Thomson Publishing.

[29] Wallace, Gregory K. (1991) The JPEG Still Image Compression Standard,*Communications of the ACM, 34(4):30–44.*

[30] Zhang, Manyun (1990) *The JPEG and Image Data Compression Algorithms.* (dissertation). University of California, Santa Cruz, 1990

[31] Bentley, J. L. et al. (1986) A Locally Adaptive Data Compression Algorithm, *Communications of the ACM, 29(4):320–330.*

[32] I.H. Witten, R. Neal, and J.G. Cleary (1987). *Arithmetic Coding for Data Compression*. Communications of the Association for Computing Machinery, 30:520–540, June 1987.
[33] Taubman, David S., and Michael W. Marcellin (2002) *JPEG 2000, Image Compression Fundamentals, Standards and Practice*, Norwell, MA, Kluwer Academic.
[34] ISO/IEC (2000), International Standard IS 15444-1 "*Information Technology—JPEG 2000 Image Coding System*." This is the FDC (final committee draft) version 1.0, 16 March 2000.
[35] JPEG 2000 Organization (2000) is http://www.jpeg.org/JPEG2000.htm.
[36] T. Acharya and P. S. Tsai. (2005) *JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures*. New York: John Wiley & Sons.
[37] Taubman, David (1999), High Performance Scalable Image Compression with EBCOT,*IEEE Transactions on Image Processing, 9(7):pp 1158–1170.*
[38] Strang, Gilbert, and Truong Nguyen (1996) *Wavelets and Filter Banks*, Wellesley, MA, Wellesley-Cambridge Press.
[39] Shapiro, J. (1993), Embedded Image Coding Using Zerotrees of Wavelet Coefficients," *IEEE Transactions on Signal Processing, 41(12):3445–3462, October*.
[40] ITU-T Recommendation (1998) *H.263. Video Coding for Low Bit Rate Communication*.
[41] M. M. Siddeq, M. A. Rodrigues(2014) A Novel Image Compression Algorithm for high resolution 3D Reconstruction, *3D Research. Springer Vol. 5 No.2.DOI 10.1007/s13319-014-0007-6*
[42] Mohammed Mustafa Siddeq, (2010), JPEG and Sequential Search Algorithm applied on Low-Frequency Sub-Band, *Journal of Information and Computing Science. 5(3). p:161-240*
[43] Adrien Maglo, Guillaume Lavou´e, FlorentDupont, C´elineHudelot, (2013). 3D mesh compression: survey, comparisons and emerging trends. *ACM Comput. Surv. 9, 4, Article 39, DOI:http://dx.doi.org/10.1145/0000000.0000000.*
[44] Pierre Alliez and Craig Gotsman. (2005). *Recent Advances in Compression of 3D Meshes*. Advances in Multiresolution for Geometric Modelling. pp. 3–26.
[45] C. Touma and C. Gotsman (1998), Triangle mesh compression,*Proceedings of Graphics Interface '98: Vancouver, British Columbia, Canada, 18 - 20 June 1998, 26-34*
[46] Pierre Alliezand Mathieu Desbrun. (2001). Progressive compression for lossless transmission of triangle meshes. *In Proceedings of SIGGRAPH. 195–202.*
[47] H. Hoppe (1996), Progressive meshes.*Computer Graphics, vol. 30, no. Annual Conference Series, pp. 99–108, 1996.*
[48] R. Pajarola and J. Rossignac, (2000) Squeeze: Fast and progressive decompression of triangle meshes,*in Proceedings of Computer Graphics International Conference, pp. 173–182.*
[49] M. Isenburg and J. Snoeylink, (2000) "Face fixer: Compressing polygon meshes with properties," *In Proceedings SIGGRAPH 2000, Computer Graphics Proceedings, pp. 263–270.*
[50] M. Garland and P. Heckbert (1998), Simplifying Surfaces with Color and Texture using Quadratic Error Metric. *Proceedings of IEEE Visualization, pp. 287-295, 1998.*
[51] Zhang Y, Hughes TJR, Bajaj C (2007). Automatic 3D Meshing for A Domain with Multiple Materials. *Proceedings of the 16th International Meshing Roundtable. pp. 367-386.*
[52] RomainArcila, C´edricCagniart, Franck H´etroy, Edmond Boyer, and FlorentDupont(2012). Segmentation of temporal mesh sequences into rigidly moving components. *Graphical Models 75(1), pp. 10–22.*
[53] Gabriel Taubin (1999), *3D Geometry Compression and Progressive Transmission*. The Euro graphics Association and Blackwell Publishers 1999.
[54] Wavefront and Java 3DOBJ Format (2015), www.eg-models.de/formats/Format Obj.html accessed Dec. 2015.
[55] Chandrajit L. Bajaj, Insung Ihm, andSanghun Park(2001). 3D RGB image compression for interactive applications. *ACM Trans. Graph. 20 (1), pp.10–38.*
[56] J. Rissanenand Glen G. Langdon (1979). Arithmetic Coding. *IBM Journal of Research and Development 23(2), pp. 149–162.*
[57] ZachiKarni and Craig Gotsman (2000). Spectral compression of mesh geometry. *In Proceedings of SIGGRAPH.279–286.*
[58] Lijun Chen and Nicolas D. Georganas. (2008). Region-based 3D Mesh Compression Using an Efficient Neighbourhood-based Segmentation. *Simulation 84 (5), 185–195.*
[59] Haeyoung Lee, Pierre Alliez, and Mathieu Desbrun(2002). Angle-Analyzer: A Triangle-Quad Mesh Codec.*Computer Graphics Forum 21(3), 383–392.*

[60] Chandrajit L. Bajaj, Valerio Pascucci, and Guozhong Zhuang(1999). Single-resolution compression of arbitrary triangular meshes with properties. *In Proceedings of the Data Compression Conference. 247–256.*

[61] Haeyoung Lee and Sujin Park(2005). Adaptive Vertex Chasing for the Lossless Geometry Coding of 3D Meshes. *In Advances in Multimedia Information Processing - PCM 2005. Lecture Notes in Computer Science, Vol. 3767. 108–119.*

[62] Chow (1997), Optimized geometry compression for real-time rendering. *In Proceedings ofVisualization.347–354.*

[63] Jeong-Hwan Ahn, Chang-Su Kim, and Yo-Sung Ho. (2006). Predictive compression of geometry, color andnormal data of 3-D mesh models. *IEEE Transactions on Circuits and Systems for Video Technology 16, 2 (2006), 291–299*

[64] UlugBayazit, UmutKonur, and Hasan F.Ates (2010). 3D Mesh Geometry Compression With Set Partitioning in the Spectral Domain. *IEEE Transactions on Circuits and Systems for Video Technology 20, 2 (2010), 179–188.*

[65] William Tutte(1963). A census of planar maps. *Canadian Journal of Mathematics 15 (1963), 249–271.*

[66] Gabriel Taubin and JarekRossignac(1998). Geometric compression through topological surgery. *ACM Transactionson Graphics 17 (1998), 84–115. Issue 2.*

[67] Pablo Diaz-Gutierrez, M. Gopi, and Renato Pajarola(2005). Hierarchyless Simplification, Stripification and Compression of Triangulated Two-Manifolds. *Computer Graphics Forum 24(3), pp. 457–467.*

[68] Jason Geng (2011), *Structured-light 3D surface imaging*: a tutorial copyright Optical Society of America, published March 31, 2011 (Doc. ID 134160)

[69] RODRIGUES, Marcos, ROBINSON, Alan, ALBOUL, Lyuba and BRINK, Willie (2006). 3D modelling and recognition. *In: ISCGAV'06 Proceedings of the 6th WSEAS International Conference on Signal Processing, Computational Geometry and Artificial Vision. World Scientific and Engineering Academy and Society (WSEAS), 72-76.*

[70] A. Robinson, M.A. Rodrigues, L. Alboul, Producing Animations from 3D Face Scans, *Game-On,6th Annual European GAME-ON Conference, De Montfort University, Leicester, UK, Nov 23–25, 2005.*

[71] M.A. Rodrigues, Alan Robinson, LyubaAlboul (2004). Apparatus and Methods for Three Dimensional Scanning, Multiple Stripe Scanning Method, *UK Patent Application No. 0402565.6, February 5th, 2004.*

[72] RODRIGUES, Marcos and ROBINSON, Alan (2011). *Fast 3D recognition for forensics and counter-terrorism applications.* In: AKHGAR, Babak and YATES, Simeon, (eds.) Intelligence management: knowledge driven frameworks for combating terrorism and organized crime. Advanced information and knowledge processing. London, Springer-Verlag, 95-109.

[73] Liu Ying, Dai Mingli, Han Zhongming, and DuanDagao (2010). An Edgebreaker& code-mode based connectivity compression for triangular meshes. *In Proceedings of the International Conference on Advanced Computer Control, Vol. 2. 96–101.*

[74] The Virtual Reality Modelling Language (VRML) (1997). ISO/IEC 14772-1, https://www.iso.org/obp/ui/#iso:std:iso-iec:14772:-1:ed-1:v1:en. Last access 2017

[75] G. Taubin, W. Horn, F. Lazarus, J. Rossignac (1998), Geometry coding and VRML, *Proc. IEEE 96 (6) (1998) 1228–1243.*

[76] J. Rossignac (1999), Edgebreaker: connectivity compression for triangle meshes, *IEEE Trans. Vis. Comput. Graph. 5 (1) (1999) 47–61.*

[77] D. Shikhare (2000), *State of the art in geometry compression*. Technical Report, National Centre forSoftware Technology, India, 2000.

[78] C. Gotsman, S. Gumhold, L. Kobbelt (2002), Simplification and compression of 3D meshes, *in: Tutorials on Multiresolution in Geometric Modelling, 2002.*

[79] M. Deering (1995), Geometry compression, *in: ACM SIGGRAPH, 1995, pp. 13–20.*

[80] E.M. Arkin, M. Held, J.S.B. Mitchell, S. Skiena (1996), Hamiltonian triangulations for fast rendering, *Vis. Computer. 12 (9) (1996) 429–444.*

[81] F. Evans, S. Skiena, A. Varshney (1996), *Completing sequential triangulations is hard*. Technical Report, Department of Computer Science, State University of New York at Stony Brook, 1996.

[82] F. Evans, S.S. Skiena, A. Varshney (1996), Optimizing triangle strips for fast rendering, *in: IEEE Visualization, 1996, pp. 319–326.*

[83] B. Speckmann, J. Snoeyink (1997), Easy triangle strips for tin terrain models. *In Proceedings of 9th CCCG, pages 239–244, 1997.*
[84] X. Xiang, M. Held, J. Mitchell (1999), Fast and efficient stripification of polygonal surface models, *in:ACM 1999 Symposium on Interactive 3D Graphics, 1999, pp. 71–78.*
[85] Maglo, A., Lavoue, G., Dupont, F. &Hudelot, C. (2015). 3D Mesh Compression: Survey, Comparisons, and Emerging Trends. *ACM Computing Surveys. 47(3) Article:44.*
[86] JiankunLi and C.-C.J. Kuo. (1998). A dual graph approach to 3D triangular mesh compression. *In Proceedingsof the International Conference on Image Processing, Vol. 2. pp. 891–894.*
[87] Peng, J., Kim, C. S. &Kuo, C. C. J. (2005). Technologies for 3D mesh compression: A survey. *Journal of Visual Communication and Image Representation. 16(6): 688-733.*
[88] Stefan Gumhold and Wolfgang StraBer (1998). Real time compression of triangle mesh connectivity. *In Proceedings of SIGGRAPH. 133–140.*
[89] Stefan Gumhold. (2000). *New Bounds on the Encoding of Planar Triangulations.* Technical Report WSI-2000-1. University of Tubingen.
[90] Davis King and JarekRossignac(1999). Guaranteed 3.67 V bit Encoding of Planar Triangle Graphs. *In Proceedings of the 11th Canadian Conference on Computational Geometry.*
[91] Davis King, Andrzej Szymczak, and Jaroslaw R. Rossignac(1999). *Connectivity Compression for Irregular Quadrilateral Meshes.* GVU Technical Report GIT-GVU-99-36.
[92] Andrzej Szymczak, Davis King, and JarekRossignac(2001). An Edgebreaker-based efficient compression scheme for regular meshes. *Computational Geometry 20, 12 (2001), 53–68.*
[93] Lawrence Ibarria, Peter Lindstrom, JarekRossignac, and Andrzej Szymczak(2003). Out-of-core Compression and Decompression of Large n-dimensional Scalar Fields. *Comput. Graph. Forum 22, 3 (2003), 343–348.*
[94] Volker Coors and JarekRossignac (2004). Delphi: geometry-based connectivity prediction in triangle mesh compression. *The Visual Computer 20 (2004), 507–520. Issue 8-9.*
[95] Haeyoung Lee, Pierre Alliez, and Mathieu Desbrun(2002). Angle-Analyzer: A Triangle-Quad Mesh Codec.*Computer Graphics Forum 21, 3 (2002), 383–392.*
[96] Boris Kronrod and Craig Gotsman(2000). Efficient Coding of Non-Triangular Mesh Connectivity. *In Proceedings of the 8th Pacific Conference on Computer Graphics and Applications. pp. 235.*
[97] Martin Isenburg(2002a). Compressing Polygon Mesh Connectivity with Degree Duality Prediction. *In Graphics Interface Conference Proceedings.*
[98] Andrei Khodakovsky, Pierre Alliez, Mathieu Desbrun, and Peter Schr¨oder(2002). Near-optimal connectivity encoding of 2-manifold polygon meshes. *Graphical Models 64 (2002), 147–168. Issue 3-4.*
[99] Eung-Seok Lee and Hyeong-SeokKo(2000). Vertex data compression for triangular meshes. *In Proceedings of Pacific Graphics. 225 –234.*
[100] Peter H. Chou and Teresa H. Meng(2002). Vertex data compression through vector quantization. *IEEE Transactions on Visualization and Computer Graphics 8, 4 (2002), 373–382.*
[101] Jae-Young Sim, Chang-Su Kim, and Sang-Uk Lee(2003). An efficient 3D mesh compression technique based on triangle fan structure. *Signal Processing: Image Communication 18, 1 (2003), 17–32.*
[102] Martin Isenburgand Pierre Alliez(2002b). Compressing polygon mesh geometry with parallelogram prediction. *In* Proceedings of the conference on Visualization ’02 (VIS ’02). *141–146.*
[103] B. Kronrod and C. Gotsman(2002). Optimized compression of triangle mesh geometry using prediction trees. *In Proceedings of the International Symposium on 3D Data Processing Visualization and Transmission. 602–608.*
[104] Dinesh Shikhare, SushilBhakar, and Sudhir P. Mudur(2001). Compression of Large 3D Engineering Models using Automatic Discovery of Repeating Geometric Features. *In Proceedings of the Vision Modeling and Visualization Conference. 233–240.*
[105] KangyingCai, Yu Jin, Wencheng Wang, QuQing Chen, Zhibo Chen, and Jun Teng(2009). Compression of massive models by efficiently exploiting repeated patterns. *In Proceedings of the ACM Symposium on Virtual Reality Software and Technology. 229–230.*
[106] Thomas Lewiner, Marcos Craizer, H´elio Lopes, Sin´esioPesco, Luiz Velho, and Esdras Medeiros(2006). Encode Geometry-driven compression for General Meshes. *Computer Graphics Forum 25, 4 (2006), 685–695.*

[107] Rapha¨elleChaine, Pierre-Marie Gandoin, and C´elineRoudet(2009). Reconstruction Algorithms as a Suitable Basis for Mesh Connectivity Compression. *IEEE Transactions on Automation Science and Engineering 6, 3 (2009), 443–453.*

[108] Martin Isenburg, Peter Lindstrom, and Jack Snoeyink(2005). Lossless compression of predicted floating-point geometry. *Computer-Aided Design 37(8) (2005), 869–877.*

[109] M. M. Siddeq, M. A. Rodrigues (2016) Novel 3D Compression Methods for Geometry, Connectivity and Texture, *3D Research. Springer*, Vol. 7 No.2. 10.1007/s13319-016-0091-x

[110] Siew, C.B. and A.A. Rahman (2012). Compression Techniques for 3D SDI. *FIG Working Week 2012 Knowing to manage the territory, protect the environment, evaluate the cultural heritage. Rome, Italy, 6-10 May 2012.*

[111] Brink, W., A. Robinson, and M. Rodrigues (2008). Indexing Uncoded Stripe Patterns in Structured Light Systems by Maximum Spanning Trees, *British Machine Vision Conference BMVC 2008, Leeds, UK, 1–4 Sep 2008*

[112] Robinson, A., L. Alboul and M.A. Rodrigues (2004). Methods for Indexing Stripes in Uncoded Structured Light Scanning Systems, *Journal of WSCG, 12(3), 2004, pp 371–378.*

[113] Rodrigues, M.A. and A. Robinson (2011). Real-time 3D Face Recognition using Line Projection and Mesh Sampling. In: EG 3DOR 2011 - *Eurographics 2011 Workshop on 3D Object Retrieval, Llandudno, UK, 10th April 2011. EurographicsAssociation.p9–16.*

[114] Rodrigues, M.A., A. Osman and A. Robinson (2010). Efficient 3D Data Compression Through Parameterization of Free-Form Surface Patches, *2010 International Conference on Signal Processing and Multimedia Applications (SIGMAP), 26-28 July 2010, p130–135.*

[115] COLLADA (2016). Digital Asset and FX Exchange Schema, https://collada.org, accessed May 2016.

[116] N. P. Weatherill, N.P. and O. Hassan (1994). Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints, International *Journal for Numerical Methods in Engineering, John Wiley & Sons, Ltd., 37(12)1981–2150.*

[117] Ali Al-Haj, (2007) "Combined DWT-DCT Digital Image Watermarking", *Science Publications, Journal of Computer Science 3 (9): 740-746.*

[118] Rafael C.Gonzalez, RichardE.Woods(2001).*Digital ImageProcessing,*Addison Wesley Publishing company.

[119] I.E. G.Richardson(2002)*VideoCodecDesign*, JohnWiley&Sons.

[120] G.SadashivappaandK.V.S.AnandaBabu, (2002) PERFORMANCE ANALYSIS OF IMAGE CODING USING WAVELETS, *IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.10.*

[121] K.R.Rao,P.Yip(1990), *Discretecosinetransform:Algorithms,advantages,applications,*AcademicPress,SanDiego, CA,1990.

[122] Tsai, M. and H. Hung,(2005). DCT and DWT based Image Watermarking Using Sub sampling, *in Proc. Of the 2005 IEEE Fourth Int. Conf. on Machine Learning and Cybernetics, pp: 5308-5313,China*.

[123] Grigorios D. N. D. Zervas, N. Sklavos and Costas E. Goutis(2008) Design Techniques and Implementation of Low Power High-Throughput Discrete Wavelet Transform Tilters for JPEG 2000 Standard,*WASETInternational Journal of Signal Processing Vo. 4, No.1.*

[124] S. Esakkirajan, T. Veerakumar, V. SenthilMurugan, and P. Navaneethan,(2008) Image Compression Using Multi-wavelet and Multi-stage Vector Quantization, *International Journal of Signal Processing Vol. 4, No.4, WASET.*

[125] M. Rodrigues, A. Osman and A. Robinson, (2013) Partial differential equations for 3D data compression and reconstruction, *Journal Advances in Dynamical Systems and Applications, Vol. 8 No. 2, 303-315.*

[126] M.Antonini,M.Barlaud, P.MathieuandI. Daubechies, (1992) Imagecodingusing wavelet transform,*IEEE Trans. on ImageProcessing, Vol. 1, No.2, pp:205–220.*

[127] C.Christopoulos, J. Askelof, and M.Larsson (2000) Efficient methods for encoding regions of interest intheupcoming JPEG 2000 still image coding standard,*IEEE Signal Processing Letters, Vol.7, No.9.*

[128] Sana Ktata, KaïsOuni, and NoureddineEllouze, (2009) A Novel Compression Algorithm for Electrocardiogram Signals based on Wavelet Transform and SPIHT,*WASET International Journal of Signal Processing Vol. 5, No.4.*

[129] I.E. G.Richardson(2002)*Video Codec Design: Developing Image and Video Compression Systems*, John Wiley & Sons, 17 Jun 2002
[130] N. Ahmed, T. Natarajan and K. R. Rao,(1974) Discrete cosine transforms, *IEEE* Transactions Computer.," vol. C-*23, pp. 90-93.*
[131] K. Sayood, (2000) *Introduction to Data Compression, 2nd edition*, Academic Press, Morgan Kaufman Publishers.
[132] P. Chen, Jia-Y. Chang (2013) An Adaptive Quantization Scheme for 2-D DWT Coefficients, *International Journal of Applied Science and Engineering Vol.11, No. 1.*
[133] SIDDEQ, M.M. and RODRIGUES, Marcos (2014). A new 2D image compression technique for 3D surface reconstruction. *Proceedings of 18th International Conference on Computers (part of CSCC'14). Recent advances in computer engineering series, 1(22). World Scientific and Engineering Academy and Society (WSEAS), 379-386.*
[134] M. M. Siddeq, G. Al-Khafaji, (2013) Applied Minimize-Matrix-Size Algorithm on the Transformed images by DCT and DWT used for image Compression, *International Journal of Computer Applications, Vol.70, No. 15.*
[135] M. Rodrigues, M. Kormann, C.Schuhler and P. Tomek(2013) Robot trajectory planning using OLP and structured light 3D machine vision. *Lecture notes in Computer Science Part II. LCNS, 8034 (8034). Springer,Heidelberg, 244-253.*
[136] M. Rodrigues, M. Kormann, C.Schuhler and P. Tomek (2013). Structured light techniques for 3D surface reconstruction in robotic tasks. In: KACPRZYK, J, (ed.) *Advances in Intelligent Systems and Computing.Heidelberg, Springer, 805-814.*
[137] SIDDEQ, M and RODRIGUES, Marcos (2015). A novel 2D image compression algorithm based on two levels DWT and DCT transforms with enhanced minimize-matrix-size algorithm for high resolution structured light 3D surface reconstruction. *3D Research, 6 (3), p. 26.*
[138] Taizo Suzuki and Masaaki Ikehara (2013), Integer fast lapped transforms based on direct-lifting of DCTs for lossy-to-lossless image coding, *EURASIP Journal on Image and Video Processing, doi:10.1186/1687-5281-2013-65*
[139] Knuth, Donald (1997). *Sorting and Searching: Section 6.2.1*: Searching an Ordered Table, The Art of Computer Programming 3 (3rd Ed.), Addison-Wesley. pp. 409–426. ISBN 0-201-89685-0
[140] Takahiko Horiuchi and Shoji Tominaga, (2008) Color Image Coding by Colorization Approach, *EURASIP Journal on Image and Video Processing Volume 2008,doi:10.1155/2008/158273*
[141] C. Matthew, Stamm and K. J. Ray Liu, (2010), WAVELET-BASED IMAGE COMPRESSION ANTI-FORENSICS, *Proceedings of 2010 IEEE 17th International Conference on Image Processing September 26-29, 2010, Hong Kong: 1737-1740*
[142] Gerald Schaefer (2014), Soft computing-based colour quantization, *EURASIP Journal on Image and Video Processing, doi:10.1186/1687-5281-2014-8.*
[143] Autodesk 123D (2016),https://en.wikipedia.org/wiki/Autodesk_123D, last accessed May-2016.
[144] 123D Catch (2016),http://www.123dapp.com/howto/catch, accessed May 2016
[145] G. Taubin, W.P. Horn, and F. Lazarus (1997). The VRML Compressed Binary Format, June 1997 http://www.research.ibm.com/vrml/binary.
[146] G. Taubin,W. Horn, and P. Borrel (1999). *Compression and transmission of multi resolution clustered meshes*. Technical Report RC-21398, IBM Research, February 1999.
[147] F. Bossen (1999) *On the Art of Compressing Three-Dimensional Polygonal Meshes and Their Associated Properties* .PhD thesis, École Poly technique Fédérale de Lausanne (EPFL), June 1999.
[148] A. Guéziec, G. Taubin, F. Lazarus, and W.P. Horn (1998). Converting sets of polygons to manifold surfaces by cutting and stitching. In IEEE Visualization'98 Conference *Proceedings, pages 383–390.*
[149] E.S. Jang, S.J. Kim, M. Song, M. Han, S.Y. Jung, and Y.S. Seo (1998). *Results of error resilient 3D mesh coding*. ISO/IEC JTC 1/SC 29/WG 11 Input Document No. M4251.
[150] J. Li and C.C. Kuo (1998). Progressive Coding of 3D Graphics Models - *Proceedings of the IEEE, 86(6):1052–1063.*
[151] S. A. Martucci (1994), Symmetric convolution and the discrete sine and cosine transforms,*IEEE Trans. Sig. Processing* SP-42, 1038-1051 (1994).
[152] H. B. Kekre, TanujaSarodeandPrachiNatu (2013), EFFICIENT IMAGE COMPRESSION TECHNIQUE USING FULL, COLUMN AND ROW TRANSFORMS ON COLOUR

IMAGE, *International Journal of Advances in Engineering & Technology, Vol. 6, Issue 1, pp. 88-100.*

[153] EvaldoGonqalvesPelaes and Yuzolano (1998), IMAGE CODING USING DISCRETE SINE TRANSFORM WITH AXIS ROTATION, *IEEE Transactions on Consumer Electronics, Vol. 44, No. 4, pp 1284-1290*

[154] M.M. Siddeq and RODRIGUES, Marcos (2015). Applied sequential-search algorithm for compression-encryption of high-resolution structured light 3D data. *In: BLASHKI, Katherine and XIAO, Yingcai, (eds.)MCCSIS : Multi-conference on Computer Science and Information Systems 2015. IADIS Press, 195-202.*

[155] Discrete Sine Transform (2016) https://en.wikipedia.org/wiki/Discrete_sine_transform, last accessed Nov. 2016.

[156] Swati Dhamija and Dr. Priyanka Jain (2011), Comparative Analysis for Discrete Sine Transform as a suitable method for noise estimation, *IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 5, No 3, September 2011*.

[157] Malini. S and Moni. R. S (2014), Use of Discrete Sine Transform for A Novel Image Denoising Technique, *International Journal of Image Processing (IJIP), Volume (8) Issue (4) 2014.*

[158] SIDDEQ, Mohammed and RODRIGUES, Marcos (2017). DCT and DST based Image Compression for 3D Reconstruction. *3D Research, 8 (5), 1-19.*

[159] SIDDEQ, Mohammed and RODRIGUES, Marcos (2017). A Novel High Frequency Encoding Algorithm for Image Compression. *EURASIP Journal on Advances in Signal Processing, 26. DOI: 10.1186/s13634-017-0461-4*

[160] Sheffield Hallam University, Mohammed M Siddeq, and Marcos A Rodrigues (2016). Image Data Compression and Decompression Using Minimize Size Matrix Algorithm. WO 2016/135510 A1.

Appendix A: Chapter 3

```
%Apply two stage Minimize_Matrix_Size Algorithm with DCT
% on the DWT for Image Compression

%%%% This type of search depends on the Limited Sequential Search Algorithm
%%% By Mohammed Mustafa Siddeq.....'
%%% Finished Work 25/7/2011

%%% Input :- "X" input image
%  "Fact" :- image quality = 0.02, 0.04, 0.06, 0.08, 0.1


% from MATLAB tool box....  "rice.png", "board.tif", "moon.tif", "coins.png"
%X = imread('coins.png');
%%%% from file...

clc;
X=imread('D:\Lectures\Images\2.bmp'); 'Image name..';
X=double(X);

%%%% Save decoded image in a file....%%%
file_wr='E:\Temp\t.bmp'; %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


Needs='No'; % this option make the algorithm work with or without high-
frequency
%Example Needs='Ye' or Needs='No'
%'No' :- without High-frequencies
%'Ye' :- with High-frequencies
C_Size=3; % (Row) -> Number of bytes to be compress by DCT  %%%%%%%%%%%%

OrSize=size(X); % original image size;

W_L=3; % weights length =2,3,4.

Fact=0.1; % this variable responsable for the image quality

%Wavelet filter name used for transformation..CoifN, dbN, symN,
%BiorN.N.....
Wavelete_Name='db3';

Color_=3;
%% Set color to [1,2,3] for RGB / set color to [0] for gray level...
if (Color_~=0)
[Y,U,V]=YUV_RGB(X); % YUV layers  for color images..
end;
if (Color_==0) IT=round(X(:,:,1)); end;% Gray level
if (Color_==1) IT=round(Y(:,:));   end;% Color layer 1
if (Color_==2) IT=round(U(:,:));   end;% Color layer 2
if (Color_==3) IT=round(V(:,:));   end;% Color layer 3
```

```
[LL,HL,LH,HH2]=dwt2(IT,Wavelete_Name);

LL=round(LL);



%------------- Quantization....
Time_exe=cputime;

QL=max(IT(:));
Q=QL.*Fact;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



for i=1:C_Size  for j=1:C_Size QL1(i,j)=Q+j+i; end; end;
QL1(1,1)=10;


if (Needs=='Ye')
% if (Fact<0.1)
%  QL2=(QL*0.1)*2;
%  QL3=(QL*0.1)*2;
%  QL4=(QL*0.1)*2;
% else

  QL2=(QL*Fact)*2;
  QL3=(QL*Fact)*2;
  QL4=(QL*Fact)*2;

 HL=round(HL./(QL2)); LH=round(LH./(QL3)); HH2=round(HH2./(QL4));

end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


if (Needs=='Ye')
%[THL,ReducedLH,LH]=SS_Algorithm(LH,4);
%[TLH,ReducedHL,HL]=SS_Algorithm(HL,4);
%[THH2,ReducedHH2,HH2]=SS_Algorithm(HH2,4);

%%% call by Visual C++ . NET
[ReducedLH,LH]=LSSalgorithm_VCCNET(LH);
[ReducedHL,HL]=LSSalgorithm_VCCNET(HL);
[ReducedHH2,HH2]=LSSalgorithm_VCCNET(HH2);



%[nonzeroLH]=Store_non_Zero_data(LH,128);
%[nonzeroHL]=Store_non_Zero_data(HL,128);
%[nonzeroHH]=Store_non_Zero_data(HH2,128);
end;
%%%%% Check if the Image matrix is color or grayscale %%%%%%%%%%%%%%%%%%%%
IT=0; IT=LL;
```

```
Size_=size(IT);
HH=Size_(1);
WW=Size_(2);
%%---------------------------- Pad zeros on each row and column
%Save(1:HH,WW:WW+Block2)=0;     %% Pad zeros on the Row...
%Save(HH:HH+Block1,1:WW)=0;     %% Pad zeros on the Column...
%-----------------------------------

IT(HH+1:HH+C_Size,WW+1:WW+C_Size)=0;

%%%%%%%%%%%%%% Transformation by one-Dimensinal DCT %%%%%%%%%%%%%%%%%%%%%
Save(1:HH,1:WW)=0; % Store AC values in new matrix called AC-Matrix;

PL=1; % pointer to store DC values
T(1:C_Size,1:C_Size)=0;
iH=1; PL1=1;
while(iH<=HH)
    jW=1; PL2=1;
while (jW<=WW)

for k1=0:C_Size-1 for k2=0:C_Size-1  T(k1+1,k2+1)=IT(k1+iH,k2+jW); end; end;

T=round(dct2(T)./QL1);

for k1=0:C_Size-1 for k2=0:C_Size-1   Save(k1+iH,k2+jW)=T(k1+1,k2+1);  end;
end;

     DC_M(PL1,PL2)=Save(iH,jW); % store DC values in new matrix "DC-Matrix"
     PL2=PL2+1;
     Save(iH,jW)=0;
     jW=jW+C_Size;
end; %While.....
 iH=iH+C_Size; PL1=PL1+1;
end;% While...
%%%%%%%%%%

%%%----------- Take differences between values at each column for AC-
Matrix......
S_=size(Save);
% -- Call function --- %% Second stage SS-A and DCT %%%%%%%%%%%%%%%%%%%%%%
[Su2,Su3,Su4,Su5,DC_M]=SS_A_with_One_DCT(DC_M,3,Wavelete_Name);



% -------- Apply our Estimater Sequential Search Algorithm
%[Table,Su,Reconst]=SS_Algorithm(Save,W_L);
[Su,Reconst]=LSSalgorithm_VCCNET(Save);
Save=0;
Save=Reconst; % Return original data to matrix "Save"

Time_exe=cputime-Time_exe;

% -------------------------Data Compression -- Part
```

```
 [Code2,Header3,Header4]=Arith_Code(Su); % Compress reduced data
%[Code3,Header5,Header6]=Arith_Code(Header3); % Compress reduced data
 S_=size(Su2); CodeSu2(1:S_(1)*S_(2))=Su2(1:S_(1),1:S_(2));
 [Code4,Header7,Header8]=Arith_Code(CodeSu2); % Compress reduced data
%[Code5,Header9,Header10]=Arith_Code(Header7); % Compress reduced data

 [A1,H1,H2]=Arith_Code(Su3); % Compress reduced data
 [A2,H3,H4]=Arith_Code(Su4); % Compress reduced data
 [A3,H5,H6]=Arith_Code(Su5); % Compress reduced data

if (Needs=='Ye')
if (max(ReducedLH(:))==0 && min(ReducedLH(:))==0)
        Code_LH=0;
else
        [Code_LH,T1,T2]=Arith_Code(ReducedLH);
end;

if (max(ReducedHL(:))==0 && min(ReducedHL(:))==0)
Code_HL=0;
else
       [Code_HL,T1,T2]=Arith_Code(ReducedHL);
end;
if (max(ReducedHH2(:))==0 && min(ReducedHH2(:))==0)
Code_HH2=0;
else
       [Code_HH2,T1,T2]=Arith_Code(ReducedHH2);
end;


 CompSize=size(Code2)+size(Code4)+size(Code_LH)+size(Code_HL)+size(Code_HH2);
else
  CompSize=size(Code2)+size(Code4);
end;

'Final Compressed Size ='
  CompSize=CompSize+size(A1)+size(A2)+size(A3)

%'Header Size '
  HSize=(size(Header4)+size(Header8))*16/8/1024

'Time execution :-'
  Time_exe


'----------------------- Apply Inverse DCT on each array form "Save"'
iH=1; L1=1;
while (iH<=HH)
    jW=1; L2=1;
while (jW<=WW)


for k1=0:C_Size-1
for k2=0:C_Size-1
```

```
T(k1+1,k2+1)=Save(k1+iH,k2+jW);
end;
end;
       T(1,1)=DC_M(L1,L2);
       L2=L2+1;

        T=round(idct2(T.*QL1));

for k1=0:C_Size-1
for k2=0:C_Size-1
             IT(k1+iH,k2+jW)=T(k1+1,k2+1);
end;
end;

        jW=jW+C_Size;
end; %While.....
    iH=iH+C_Size; L1=L1+1;
end;% For...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%----------------------------------------------------------------



 S_=size(HL);
 A(1:S_(1),1:S_(2))=IT(1:S_(1),1:S_(2));
 HL2(1:S_(1),1:S_(2))=0;
if (Needs=='Ye')

 HL=round(HL.*(QL2)); LH=round(LH.*(QL3)); HH2=round(HH2.*(QL4));

 IT2=idwt2(A,HL,LH,HH2,Wavelete_Name);
else
 IT2=idwt2(A,HL2,HL2,HL2,Wavelete_Name);
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DeI(1:OrSize(1),1:OrSize(2))=IT2(1:OrSize(1),1:OrSize(2));
imshow(uint8(DeI)); % show images
imwrite(uint8(DeI),file_wr); % image write on the disk
```

Appendix B: Chapter 4

```
% DWT and JPEG Block Seqeuntial Search Algorithm
% Enhanced Minimize-Matrix-Size Algorithm for High Resolution 3D Surface
Reconstruction for Gery/Colour

clear; clc;
O =imread('D:\Lectures\Images\imageg3.bmp');
Str_save='D:\Lectures\Images\t.bmp';

blocksize=8; %Block size...2,4,8,16. This means block size 2x2 or 4x4 or
8x8...
Quantization_L_F=2; %quantization luminance


% Time execution....
Total_time=cputime;

[IT,C1,H1,H2]=DWT_JPEG_MMS_2(O,Quantization_L_F,blocksize);
imshow(uint8(IT));
imwrite(uint8(IT),Str_save);

'Compress size in Kbytes = '
Compress=(C1)/1024

'Image Quality'
% compute the quality
[RMSE, SNR,P_SNR]=Peak_SNR(round(IT),O);

RMSE

'Total Time execution is :'
Total_time=cputime-Total_time

----------------------------------------------------------------------

function [IT,Compress_Data,H1,H2]=DWT_JPEG_MMS(x,quant_multiple,blocksize)
% combined JPEG with DWT for image compression (DWT-JPEG)
% by Mohammed Mustafa Siddeq
% 1/7/2012

%-----------------------------------------------------------------------
% INPUTS :\  Variables
% x: input image matrix....
% quant_multiple  : used for quantization...
% High_freq_value : used for high-freqeuncy quantization....

%---------quality for increase quantization value.....
%quant_multiple =4; % set the multiplier to change size of quant. levels

%clc; % clear all variables from previous sessions
x=double(x);
```

```
wavele_Name='db3';
%----------------------

% Apply Single stage DWT ....
 [A,V,H,D]=dwt2(x,wavele_Name);
 A=round(A);


V=0; D=0; H=0; % in case of High-freqeuncies are ignored ......


 %DCT_quantizer = [ 16 11 10 16 24 40 51 61;
%                  12 12 14 19 26 58 60 55;
%                  14 13 16 24 40 57 69 56;
%                  14 17 22 29 51 87 80 62;
%                  18 22 37 56 68 109 103 77;
%                  24 35 55 64 81 104 113 92;
%                  49 64 78 87 103 121 120 101;
%                  72 92 95 98 112 100 103 99 ];

for i=1:blocksize
for j=1:blocksize
      DCT_quantizer(i,j)=blocksize+(i+j);
if (DCT_quantizer(i,j)./3 ~= floor(DCT_quantizer(i,j)./3))
DCT_quantizer(i,j)=DCT_quantizer(i,j)+1;
end;
end;
end;
%%%%%%%%-----------------------------------------


sz = size(A);
rows = sz(1,1); % finds image's rows and columns
cols = sz(1,2);
colors = max(max(x)); % guess at the number of colors in the image
%x(1:sz(1),sz(2)+1:sz(2)+blocksize)=0;
A(sz(1)+1:sz(2)+blocksize,sz(2)+1:sz(2)+blocksize)=0;

% Prepare image for transform
% Level-shift the image (center intensity values around 0)
A =A - ceil(colors/2);
% Replicate edges of image to make its dimensions a multiple of blocksize
L=1;

i = 0;
for j = 0: blocksize - 1
DCT_trans(i+1, j + 1) = sqrt(1 / blocksize) * cos ((2 * j + 1) * i * pi / (2
* blocksize));
end
% Create DCT function
for i =1: blocksize - 1
for j = 0: blocksize - 1
   DCT_trans(i + 1, j + 1) = sqrt(2 / blocksize) * cos ((2 * j + 1) * i * pi
/ (2 * blocksize));
end
```

```
end

% Apply DCT on each block from image.....
jpeg_img = A - A;
for row = 1: blocksize: rows
for col = 1: blocksize: cols
% take a block of the image:
DCT_matrix = A(row: row + blocksize-1, col: col + blocksize-1);
DCT_matrix = DCT_trans * DCT_matrix * DCT_trans';
DCT_matrix = floor (DCT_matrix./ (DCT_quantizer(1:blocksize, 1:blocksize) *
quant_multiple)+0.5);


jpegimg(row: row + blocksize-1, col: col + blocksize-1) = DCT_matrix;

%-----------------------------------------------------------------------

end
end

% Zigzag for each sub-image 8x8
XZzag=blkproc(jpegimg,[blocksize, blocksize],'zigzag(jpegimg)');




[DC_Values,H_F]=Separate_DC_High(XZzag,blocksize*blocksize); % Seplit matrix
into DC and high-frequency matrix



% Applied Minimize-Matrix-Size Algorithm and Block LSS-Algorithm
[Arr,H_F_Est,Table]=Parallel_SSA_and_MMS(H_F);


% Combine DC values with reconstructed High-frequency matrix
[XZzag]=Combine_DC_High(DC_Values,H_F_Est,blocksize*blocksize);


[nonzero,count_zeros,matrix_size]=Count_Zeros_bewteen_Values(Arr);

%%%%%%%%%%%%%%%%%%%% for Test %%%%%%%%%%% Apply Minimize-Matrix-size
Algorithm on  "count_zeros" array
 Key1=[0.27849,0.91337,0.12698]; %Key1=Key_generater(1,20,1);
 [EData,Table]=Encryption_Coding_Minimize_Array_Algorithm(Key1,count_zeros);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% this function is new idea for lossless data compression
H2=0; H1=0; [C1]=lossless_data_compression(nonzero);
[C2,H3,H4]=Arith_Code(EData);
SDC_=size(DC_Values);
DD(1:SDC_(1)*SDC_(2))=DC_Values(1:SDC_(1),1:SDC_(2));
```

```
[C3,H5,H6]=Arith_Code(DD);

Compress_Data=C1+size(C2)+size(C3);


%--------- Begin of the Decompression --------------------
% Apply invese DCT on each block
% apply inverse Zigzag on each block
XdctQRR=blkproc(XZzag,[1 blocksize*blocksize],'zigzaginv(x)');



reconimg = A - A;
for row = 1: blocksize: rows
for col = 1: blocksize: cols

IDCTmatrix =XdctQRR(row: row + blocksize-1, col: col + blocksize-1);

IDCTmatrix = IDCTmatrix ...
.* (DCT_quantizer(1:blocksize, 1:blocksize) * quant_multiple);

IDCTmatrix = DCT_trans' * IDCTmatrix * DCT_trans;

reconimg(row: row + blocksize-1, col: col + blocksize-1) = IDCTmatrix;
end
end


A_A = reconimg(1:rows, 1:cols);
A_A =A_A + ceil(colors/2);

'Reconstruct 2D Image------------------';
         IT=idwt2(A_A,[],[],[],wavele_Name);

End % end of function
-----------------------------------------------------------------------

function [Arr,SAVE_Est,Table]=Parallel_SSA_and_MMS(SAVE)
% This function is used to Estimate SAVE matrix
%Key=[0.1 ,2.6, 17.6]; % Key used for Decoding....

%m=max(abs(SAVE(:)));
%[Key]=Key_generater(0.1, m, 2);

Key=[0.27849,0.91337,0.12698];


% compression part.....................................................
'Apply Minimize-Matrix-Size Algorithm'
S_=size(SAVE);
SAVE(1:S_(1),S_(2)+1:S_(2)+3)=0;
j=1;L=1; ArrS=round((S_(2)/3))*S_(1);
Arr(1:ArrS)=0;
```

```
while (j<S_(2))
for i=1:S_(1)
    Arr(L)=SAVE(i,j).*Key(1) + SAVE(i,j+1).*Key(2) + SAVE(i,j+2).*Key(3);
    L=L+1;
end;
 j=j+3;
end;
'compute Limited Data for SS-Algorithm'
%------------------Compute Table of Symbols
Data(1:S_(1)*S_(2))=SAVE(1:S_(1),1:S_(2));

Table(1)=Data(1);
S_AC=size(Data);
for jAC=1:S_AC(2)
        S_2AC=size(Table);Flag=0;
for kAC=1:S_2AC(2)
if (Table(kAC)==Data(jAC))
                Flag=1;
end;
end;
if (Flag==0) Table(S_2AC(2)+1)=Data(jAC); end;
end;

Table_Size=size(Table);
C=0;
while (C==0)
if (Table_Size(2)/10 == int32(Table_Size(2)/10))
        C=1;
else
        Table=[Table,0];
        Table_Size=size(Table);
end;
end;

 timeexec=cputime; % initlize the cpu time...
% Decompression part.....................................................
'Apply Parallel LSS-Algorithm.....'
SAVE_Est(1:S_(1),1:S_(2))=0;
j=1; L=1;
while (j<S_(2))
for i=1:S_(1)
   [Flag]=Block_SS_Algorithm_Test(Arr(L),Table,Key);
   SAVE_Est(i,j)=Flag(1); SAVE_Est(i,j+1)=Flag(2); SAVE_Est(i,j+2)=Flag(3);
   L=L+1;
end;
  j=j+3;
end;
'Total time execution : Parallel Search Algorithm'
cputime-timeexec   % show cpu time execution.....
end


-----------------------------------------------------------------------

function [Flag]=Block_SS_Algorithm_Test(Value,Limited_Data,Key)
```

```
%Block Sequential Search algorithm
% this algorithm is applied on each 3 data from matrix.


%Value=2.4; % the compressed value....
%Key=[423 ,7, 0.1]; % Key used for Decoding....
% probability data of the compressed matrix.
Num_Pointers=10;
Len=Num_Pointers; % pointers are worked together to find the result
Limited_Size=size(Limited_Data);
%-----------------------------------------------------------------
% Assign 10 pointers working together to find the solution .......
Seg_1=Limited_Data; P_S1=1:Num_Pointers; %P_S1=[1 2 3 4 5 6 7 8 9 10]; %first
segment with 10 pointers
Seg_2=Limited_Data; P_S2=1:Num_Pointers; %P_S2=[1 2 3 4 5 6 7 8 9 10]; %
second segment with 10 pointers
Seg_3=Limited_Data; P_S3=1:Num_Pointers; %P_S3=[1 2 3 4 5 6 7 8 9 10]; %
third segment with 10 pointers
Flag=[0 0 0]; % in case if no matchs

while (Flag(1)==0 && Flag(2)==0 && Flag(3)==0)

     k3=1;
while (k3<=Len && (Flag(1)==0 && Flag(2)==0 && Flag(3)==0))
      k2=1;
while (k2<=Len && (Flag(1)==0 && Flag(2)==0 && Flag(3)==0))
          k1=1;
while (k1<=Len && (Flag(1)==0 && Flag(2)==0 && Flag(3)==0))
Str=Seg_1(P_S1(k1)).*Key(1) + Seg_2(P_S2(k2)).*Key(2) +
Seg_3(P_S3(k3)).*Key(3);
if (Str==Value) Flag=[P_S1(k1) P_S2(k2) P_S3(k3)]; end;
             k1=k1+1;
end;
           k2=k2+1;
end;
        k3=k3+1;
end;
     P_S1=P_S1+Len; % incremet all pointers.... at same time
if (P_S1(Len)>Limited_Size(2)) P_S1=1:Num_Pointers; P_S2=P_S2+Len;  end;
if (P_S2(Len)>Limited_Size(2)) P_S2=1:Num_Pointers; P_S3=P_S3+Len;  end;
if (P_S3(Len)>Limited_Size(2)) P_S3=1:Num_Pointers;   end;


end; % End loop
% put the final result on the flage.......
 Flag(1)=Limited_Data(Flag(1));
 Flag(2)=Limited_Data(Flag(2));
 Flag(3)=Limited_Data(Flag(3));

end
```

Appendix C: Chapter 5

```
% Novel 2D image Compression Algorithm by two-levels DWT-DCT with
% Enhanced Minimize-Matrix-Size Algorithm for High Resolution 3D Surface
Reconstruction
% for 2D Images

% By
% Mohammed M. Siddeq
% Sheffield Hallam University - C3RI
% Art Computing Engineering Science
% Research_NO_3

%---------------------------------- the main program ------------
clear; clc;
O =imread('D:\Lectures\Images\image2.bmp');
Str_save='D:\Lectures\Images\t.bmp';

% using Color Transformation ....
Color_Transform=1; % if "Color_Transform=1" this parameter allow to user
converts original image [RGB to YCbCr]
% else "Color_Transform=0" this parameter keep original data
blocksize=8;              %Block size... 2,4,8..etc This means block size 2x2
or 4x4 or 8x8,..etc
Point_Theshold=64;        % point threshold.....
Quantization_L_F=[2,5,5]; % quantization Low frequency


% Time execution....
Total_time=cputime;


% Processing Begin ....
if (Color_Transform==1)
 Ycc_data = rgb2ycbcr(O); % Convert original image [RGB to YCbCr]
else
 Ycc_data=O;
end;


%Time_exe=cputime;% Time execution started .....


O=double(O); S_=size(O);
'----- Pre-Processor ------------'
[Y,U,V]=YUV_RGB(O);

 one_O=O(:,:,1);
 two_O=O(:,:,2);
three_O=O(:,:,3);

 one_O=reshape(one_O,1,S_(1)*S_(2));
 two_O=reshape(two_O,1,S_(1)*S_(2));
 three_O=reshape(three_O,1,S_(1)*S_(2));
```

```
 Pixel_ValueY=0; Pixel_ValueU=0; Pixel_ValueV=0;
 LOC=find(U>=Point_Theshold);

if (size(LOC,1)<=128) % this condtion stop saving RED postions... if number
of pixels > 1000
for i=1:size(LOC,1)
     Pixel_ValueY=[Pixel_ValueY,one_O(LOC(i))];
Pixel_ValueU=[Pixel_ValueU,two_O(LOC(i))];
Pixel_ValueV=[Pixel_ValueV,three_O(LOC(i))];
end;
end;



%---------------------- Compression and De-Compression Process
[IT,C1,H1,H2]=DWT_MMS_2(Ycc_data(:,:,1),Quantization_L_F(1),blocksize);
Decoded_Ycc_data(:,:,1)=IT(:,:);

[IT,C2,H3,H4]=DWT_MMS_2(Ycc_data(:,:,2),Quantization_L_F(2),blocksize);
Decoded_Ycc_data(:,:,2)=IT(:,:);

[IT,C3,H5,H6]=DWT_MMS_2(Ycc_data(:,:,3),Quantization_L_F(3),blocksize);
Decoded_Ycc_data(:,:,3)=IT(:,:);

%'Time execution....'
%Time_exe=cputime-Time_exe

Decoded_Ycc_data=uint8(Decoded_Ycc_data);
if (Color_Transform==1)
 re_O= ycbcr2rgb(Decoded_Ycc_data); % Convert Decompressed image [YCbCr to
RGB]
else
 re_O=Decoded_Ycc_data;
end;
%--------------Put the Cross-Pointes in exact location ------------------
 S_=size(re_O);
 one_O=re_O(:,:,1);
 two_O=re_O(:,:,2);
 three_O=re_O(:,:,3);

 one_O=reshape(one_O,1,S_(1)*S_(2));
 two_O=reshape(two_O,1,S_(1)*S_(2));
 three_O=reshape(three_O,1,S_(1)*S_(2));

if (size(LOC,1)<=128) % this condtion stop putting RED poits in exact
locations... if number of pixels > 1000
for i=1:size(LOC,1)
     one_O(LOC(i))=Pixel_ValueY(i);
     two_O(LOC(i))=Pixel_ValueU(i);
     three_O(LOC(i))=Pixel_ValueV(i);
end;
end;
```

```
 one_O=reshape(one_O,S_(1),S_(2));
 two_O=reshape(two_O,S_(1),S_(2));
 three_O=reshape(three_O,S_(1),S_(2));

 re_O(:,:,1)=one_O;
 re_O(:,:,2)=two_O;
 re_O(:,:,3)=three_O;

%%------------------------------------------------------------------------
 imshow(re_O);
 imwrite(re_O,Str_save);

'Compress size in Kbytes = '
Compress=(C1+C2+C3)/1024

'Image Quality'
% compute the quality
[RMSE, SNR,P_SNR]=Peak_SNR(re_O,O);
RMSE
P_SNR

'Total Time execution is :'
 Total_time=cputime-Total_time

 ------------------------------------------------------------------------

function [IT,Compress_Data,H1,H2]=DWT_MMS_2(x,quant_multiple,blocksize)
% combined DCT with DWT for image compression (DWT-JPEG)
% by Mohammed Mustafa Siddeq
% 13/5/2014

%------------------------------------------------------------------------
% INPUTS :\  Variables
% x: input image matrix....
% quant_multiple  : used for quantization...
% High_freq_value : used for high-freqeuncy quantization....

%---------quality for increase quantization value.....
%quant_multiple =4; % set the multiplier to change size of quant. levels

%clc; % clear all variables from previous sessions
x=double(x);
S_=size(x);
%x(S_(1)+1:S_(1)+blocksize,S_(2)+1:S_(2)+blocksize)=0;

wavele_Name='db3';

LEVEL_DWT=2; % choose "1" or "2" represents number of levels..

Ratio_VV2_choose_by_user=0.3; % this ratio is used to remove some data from
second level in hight freqeuncies8
Ratio_DD2_choose_by_user=0.3;
Ratio_HH2_choose_by_user=0.3;
%----------------------
```

```
nonzero_VV2=0; ZData_VV2=0; % initialize arrays before use it...
nonzero_HH2=0; ZData_HH2=0;
nonzero_DD2=0; ZData_DD2=0;


% Apply  DWT ....
 [A,V,H,D]=dwt2(x,wavele_Name);

if (LEVEL_DWT==2)
  [A2,VV2,HH2,DD2]=dwt2(A,wavele_Name);

 M_A=min(A2(:));
 A2=A2-M_A;
 A2=round(A2/2);

%%%%%%
 A=A2;%
%%%%%%

% Secnd level High-Freqeuncies.... each sub-band qunatized by using
ratio.....
 Q1=max(abs(VV2(:))); Q1=(Q1*Ratio_VV2_choose_by_user)*(quant_multiple);
 Q2=max(abs(HH2(:))); Q2=(Q2*Ratio_HH2_choose_by_user)*(quant_multiple);
 Q3=max(abs(DD2(:))); Q3=(Q3*Ratio_DD2_choose_by_user)*(quant_multiple);

 VV2=round(VV2/Q1); HH2=round(HH2/Q2); DD2=round(DD2/Q3);

%%%%% Each Qunatized sub-band and then minimized by Minimzation process
 [VV2_Est,nonzero_VV2,ZData_VV2]=Reduce_Matrix_Size(VV2);

 [DD2_Est,nonzero_DD2,ZData_DD2]=Reduce_Matrix_Size(DD2);

 [HH2_Est,nonzero_HH2,ZData_HH2]=Reduce_Matrix_Size(HH2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
else
    M_A=min(A(:));
    A=A-M_A;
   A=round(A/2);
end; %% end if statement....


 V=0; D=0; H=0; % in case of High-freqeuncies are ignored in first stage
......


for i=1:blocksize
for j=1:blocksize
      DCT_quantizer(i,j)=blocksize+(i+j);
end;
end;
%%%%%%%%%-------------------------------------------
```

```
sz = size(A);
rows = sz(1,1); % finds image's rows and columns
cols = sz(1,2);
colors = max(max(x)); % guess at the number of colors in the image
%x(1:sz(1),sz(2)+1:sz(2)+blocksize)=0;
A(sz(1)+1:sz(1)+blocksize,sz(2)+1:sz(2)+blocksize)=0;

% Prepare image for transform
% Level-shift the image (center intensity values around 0)
A =A - ceil(colors/2);
% Replicate edges of image to make its dimensions a multiple of blocksize
L=1;

i = 0;
for j = 0: blocksize - 1
DCT_trans(i+1, j + 1) = sqrt(1 / blocksize) * cos ((2 * j + 1) * i * pi / (2
* blocksize));
end
% Create DCT function
for i =1: blocksize - 1
for j = 0: blocksize - 1
   DCT_trans(i + 1, j + 1) = sqrt(2 / blocksize) * cos ((2 * j + 1) * i * pi
/ (2 * blocksize));
end
end

% Apply DCT on each block from image.....
jpeg_img = A - A;
for row = 1: blocksize: rows
for col = 1: blocksize: cols
% take a block of the image:
DCT_matrix = A(row: row + blocksize-1, col: col + blocksize-1);
DCT_matrix = DCT_trans * DCT_matrix * DCT_trans';
DCT_matrix = floor (DCT_matrix./ (DCT_quantizer(1:blocksize, 1:blocksize) .*
quant_multiple)+0.5);


jpegimg(row: row + blocksize-1, col: col + blocksize-1) = DCT_matrix;

%-----------------------------------------------------------------------

end
end

% Zigzag for each sub-image 8x8
XZzag=blkproc(jpegimg,[blocksize, blocksize],'one_dim(jpegimg)');



[DC_Values,H_F]=Separate_DC_High(XZzag,blocksize*blocksize); % Seplit matrix
into DC and high-frequency matrix
```

```
%%%%%%%%%%%  Transform DC-Values by 2D-DCT %%%%%%%%%%%%%%%%%%%%%%%%%
minV=min(DC_Values(:));
DC_Values=DC_Values+abs(minV);

DC2=blkproc(DC_Values,[blocksize blocksize],'dct2(DC_Values)');
DC2=round(DC2);

DC2_ZigZag=blkproc(DC2,[blocksize blocksize],'one_dim(DC2)');

% This function is used to separate DC values from AC values
[DC_Values2,H_F2]=Separate_DC_High(DC2_ZigZag,blocksize*blocksize);

%this function is used to separate zeros from non-zero data

[nonzero_Arry_H,ZeroArr_H,matrix_size]=Count_Zeros_bewteen_Values(H_F2);

%%%%%%%%%  Inverse 2D-DCT for reconstruct DC-Values %%%%%%%%%%%%%%%%
DC2=blkproc(DC2_ZigZag,[1 blocksize*blocksize],'inv_one_dim(DC2_ZigZag)');

DC_Values=blkproc(DC2,[blocksize blocksize],'idct2(DC_Values)');
DC_Values=round(DC_Values)-abs(minV);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Applied Minimize-Matrix-Size Algorithm and Fast Matching Search Algorithm
Key1=[0,0.91337,0.12698];
[H_F_Est, ZData, nonzero1]=FMS_Algorithm(H_F,Key1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%

% Combine DC values with reconstructed High-frequency matrix
[XZzag]=Combine_DC_High(DC_Values,H_F_Est,blocksize*blocksize);



%%%%%%%%%%%%%%%%%%%%%%-------- Data Compression Part %%%%%%%%%%%%%%%%%%
% this function is new idea for lossless data compression
H2=0; H1=0; C0=0;
[C0]=lossless_data_compression(nonzero1);

if (min(ZData(:))~=max(ZData(:)))
 [C1,H_11,H_22]=Arith_Code(ZData);
else
  C1=0; H_11=0; H_22=0;
end;
%[C0,H_11,H_22]=Arith_Code(nonzero1);

  SDC_=size(DC_Values2);
   DD(1:SDC_(1)*SDC_(2))=DC_Values2(1:SDC_(1),1:SDC_(2));

if (min(DD(:))~=max(DD(:)))
   [C2,H5,H6]=Arith_Code(DD);
else
    C2=0;
```

```
end;

if (min(ZeroArr_H(:))~=max(ZeroArr_H(:)))
  [C3,H7,H8]=Arith_Code(ZeroArr_H);
else
   C3=0;
end;


if (min(nonzero_Arry_H(:))~=max(nonzero_Arry_H(:)))
 [C4,H9,H10]=Arith_Code(nonzero_Arry_H);
else
 C4=0;
end;

if (LEVEL_DWT==2)
%%%%% Compress the second level sub-bands...
    [C_non_VV2]=lossless_data_compression(nonzero_VV2);

if (min(ZData_VV2(:))~=max(ZData_VV2(:)))
        [C_Z_VV2,H_11,H_22]=Arith_Code(ZData_VV2);
else
       C_Z_VV2=0;
end;


     [C_non_HH2]=lossless_data_compression(nonzero_HH2);

if (min(ZData_HH2(:))~=max(ZData_HH2(:)))
      [C_Z_HH2,H_11,H_22]=Arith_Code(ZData_HH2);
else
       C_Z_HH2=0;
end;

      [C_non_DD2]=lossless_data_compression(nonzero_DD2);

if (min(ZData_DD2(:))~=max(ZData_DD2(:)))
       [C_Z_DD2,H_11,H_22]=Arith_Code(ZData_DD2);
else
       C_Z_DD2=0;
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end;
Compress_Data=(C0)+size(C2)+size(C1)+size(C3)+size(C4);

if (LEVEL_DWT==2)
Compress_Data=Compress_Data+(C_non_VV2+C_non_HH2+C_non_DD2);
Compress_Data=Compress_Data+( size(C_Z_VV2)+size(C_Z_HH2)+size(C_Z_DD2));
end;
Compress_Data=Compress_Data(2);
```

```
%--------- Begin of the Decompression --------------------
% Apply invese DCT on each block


% apply inverse Zigzag on each block
XdctQRR=blkproc(XZzag,[1 blocksize*blocksize],'inv_one_dim(x)');


% aply inverse DCT ....
reconimg = A - A;
for row = 1: blocksize: rows
for col = 1: blocksize: cols

IDCTmatrix =XdctQRR(row: row + blocksize-1, col: col + blocksize-1);

IDCTmatrix = IDCTmatrix ...
.* (DCT_quantizer(1:blocksize, 1:blocksize) * quant_multiple);

IDCTmatrix = DCT_trans' * IDCTmatrix * DCT_trans;

reconimg(row: row + blocksize-1, col: col + blocksize-1) = IDCTmatrix;
end
end


A_A = reconimg(1:rows, 1:cols);
A_A =A_A + ceil(colors/2);


% Apply Inverse DWT for decode 2D image ...
'Reconstruct 2D Image------------------';
if (LEVEL_DWT==2)
             %% Normalize sub-bands size before apply inverse DWT...
             s_VHD=size(A_A); VV2(:,:)=0; HH2(:,:)=0; DD2(:,:)=0;
             VV2(1:s_VHD(1),1:s_VHD(2))=VV2_Est(1:s_VHD(1),1:s_VHD(2));
             HH2(1:s_VHD(1),1:s_VHD(2))=HH2_Est(1:s_VHD(1),1:s_VHD(2));
             DD2(1:s_VHD(1),1:s_VHD(2))=DD2_Est(1:s_VHD(1),1:s_VHD(2));

             A1=idwt2((A_A*2)+M_A,VV2*Q1,HH2*Q2,DD2*Q3,wavele_Name);
             IT=idwt2(A1,[],[],[],wavele_Name);
else
            IT=idwt2((A_A*2)+M_A,[],[],[],wavele_Name);
end;
         IT=round(IT);

End% End of Function
-----------------------------------------------------------------------


function [Decode_H_F, ZeroArr, nonzero_Arry]=FMS_Algorithm(H_F,Key1)
% Designed by
% Mohammed M. Siddeq
% date :- 24 - MAY.- 2014
% e-mail :- mamadmmx76@yahoo.com
%----------------------------------
```

```
%load 'SAVE_Data';
% start to cmpute time execution .....
Execution_Time=cputime;
%---------------------------------- Coding / Decoding working........

% initilize variables beofre use it
Keep_data_one_picea=1; % this option allow for the use to keep data (not need
to split ): all data in "nonzero_Arry"
%%%%%%%%%%%%%%%%%%%%%%%%% in case of "Keep_data_one_picea=0" this means
%%%%%%%%%%%%%%%%%%%%%%%%% split data to : Zero and Nonzero
nonzero_Arry=0;
ZeroArr=0;
Data_Length=3;
S_=size(H_F);
H_F(1:S_(1),S_(2)+1:S_(2)+Data_Length)=0;
%S_=size(H_F);
%Key1=[0.278498218867048, 0.913375856139019,0.126986816293506]; % random
number generated....
%MAX_V=max(abs(H_F(:)));
%Key1=Key_generater(0.1,MAX_V,2);
%Key1=[0.1, 0.3,0.6];
% Key1=[0.278498,0.91337585,0.1269868];

' Apply Minimize-Matrix-Size Algorithm.....(Coding / Compression)'

% compute the probability of the matrix for P1, P2 and P3
 Table1=[]; %Table2=[]; Table3=[];
for i=1:S_(1)
    L=1; j=1;
while (j<=S_(2))
        S=(Key1(1).*H_F(i,j))+(Key1(2).*H_F(i,j+1))+(Key1(3).*H_F(i,j+2));
        [Table1] = Generate_Limted_Data(Table1, H_F(i,j));
         [Table1] = Generate_Limted_Data(Table1, H_F(i,j+1));
          [Table1] = Generate_Limted_Data(Table1, H_F(i,j+2));
           j=j+Data_Length;
           MMS_H_F(i,L)=S;
           L=L+1;
end;

end;

% this function separate zeros and nonzeros data in separated arrays, to be
% easy for coding by arithmetic coding.......

if (Keep_data_one_picea==1)
   S_=size(MMS_H_F); nonzero_Arry(1:S_(1)*S_(2))=MMS_H_F(1:S_(1),1:S_(2));
else
  [nonzero_Arry,ZeroArr,matrix_size]=Count_Zeros_bewteen_Values(MMS_H_F);
end;

%%%%%%%  End of Compression  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% After result of coding is : "MMS_H_F", "Table1"
%%%%%% Final result of Compression is: "ZeroArr" and "nonzero_Arry"
%%%%%%%%%%%%%%%%%%%%%%%% ------------------------------------------------
```

```
%.
%.
%.
%.
%.
%.
%%%%%%%%%%%%%% Decompression Algorithm by Fast Matching Search algorithm
(FMS-Algorithm)

Execution_Time=cputime;

' FMS-Algorithm started ....'
%'1- compute all posibilites of P1, P2 and P3'
%1) Built Array contains compressed data for the all probaility (i.e.
Tabl1,Table2 and Table3)
 L=1; %test_find_postion(1:100)=0;
 Data(1:size(Table1,2)*size(Table1,2)*size(Table1,2),1:4)=0;

for i=1:size(Table1,2)
for j=1:size(Table1,2)
for k=1:size(Table1,2)

s=(Table1(k).*Key1(1))+(Table1(j).*Key1(2))+(Table1(i).*Key1(3));
             Data(L,1)=s; Data(L,2)=Table1(k); Data(L,3)=Table1(j);
Data(L,4)=Table1(i);
             L=L+1;
end;
end;
end;

%'2- starting for matching and decoding at same time...'
% 2) Start sorting according to summation .....
Data=sortrows(Data);
S_=size(Data); test_find_postion(1:S_(1))=Data(1:S_(1));
%3) using binary search for looking for original data
S_=size(MMS_H_F); Decode_H_F(1:S_(1),1:S_(2)*3)=0;
for i=1:S_(1)
    L=1;
for j=1:S_(2)
      LOC=binary_search(test_find_postion,MMS_H_F(i,j)); % matching result ,
and find the postion
% put the original data in exact postions
        Decode_H_F(i,L)=Data(LOC,2);
        Decode_H_F(i,L+1)=Data(LOC,3);
        Decode_H_F(i,L+2)=Data(LOC,4);
      L=L+3;
end;
end;
' Time execution:(Sec.)'
Execution_Time=cputime-Execution_Time

end% End of function .....
```

Appendix D: Chapter 6

```
%New Compression image method used by Cosine and Sine Transformation
% for high resolution 3D surface reconstraction
% Designed By : Mohammed M. Siddeq,
% C3RI - Sheffeid Hallam University
%        Sheffield - United Kingdom

% This porogram for Colour images.....
%---------------------------------- the main program ------------

clear; clc;
O=imread('D:\Lectures\Images\ImageG\11.bmp'); % read original image...
Str_save1='D:\Lectures\Images\ImageG\t.bmp';% decompressed image saved in
different name...


% Time execution....
Total_time=cputime;
Quantization_DCT=2;
Quantization_DST=1;



% Choose one of these algorithms
% function is compression and decompression at same time to show
%1) This algorithm compress each 3 data to 1 value.

[Compressed_Size,xp]=Cosine_Sin(O,Quantization_DST,Quantization_DCT);

'Compress size in Kbytes = '
Compressed_Size=Compressed_Size/1024

imshow(uint8(xp));
imwrite(uint8(xp),Str_save1);
'Total Time execution is :'
Total_time=cputime-Total_time

'Image Quality'
% compute the quality
[RMSE, SNR,P_SNR]=Peak_SNR(xp,O);
RMSE

-----------------------------------------------------------------------

function [Compressed_Size,xp]=Cosine_Sin(Im,L_DST,Q)
'Algorithm for DST with DCT working...'
%'1D-Cosine Transform with Sin Transform for Image compression... ';
%                  'By Mohammed Mustafa Siddeq';
% Input:\ image data = "Im"
%          scale quantization used in DST = "L_DST", limit- {1-10}
```

```
%          uniform quntization used in DCT = "Q"  ,  limit- {1-10}
% Output:\ Decompressed image = "xp"
%          Compressed data ="C_"
%          Header compressed data = "Tab"
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 Low_Freq_Data=150; % this vasriable is used to keeps low frequency data

 p(:,:)=double(Im(:,:,1));
%'----------------apply DCT ----------------------------------'
 S_=size(p);
for i=1:S_(1)
      j=1;Three(1:S_(2))=0;

while(j<=S_(2)-1)
for k=0:S_(2)-1 Three(k+1)=p(i,j+k);end;
         Three=round(dct(Three)./Q);
for k=0:S_(2)-1 Savep(i,j+k)=Three(k+1);end;
        j=j+S_(2);
end; %End while...
end;   %End for.......


% ----------------------Quantization matix
 SizeX=size(Savep);
Div_(1:S_(1),1:S_(2))=1;
for i=1:S_(1)
for j=1:S_(2)
         Div_(i,j)=(i+j)*L_DST;
end;
end;


%-------------Apply DST ----------------------------------------------------
Savep=round(dst(Savep)./Div_);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


DC_Matrix(1:Low_Freq_Data,1:Low_Freq_Data)=Savep(1:Low_Freq_Data,1:Low_Freq_D
ata);
 Savep(1:Low_Freq_Data,1:Low_Freq_Data)=0;


 [nonzero1,ZeroArr1,array_size1,datasize1,LOCi1,LOCj1,Limited_Data1] =
Coding_Half_Matrix(Savep);
if (max(nonzero1(:))~=min(nonzero1(:)))
  [Comp2,H2,H_2]=Arith_Code(nonzero1);
  [Comp3,H3,H_3]=Arith_Code(ZeroArr1);
else
  Comp2=0; Comp3=0;
end;


Data(1:Low_Freq_Data*Low_Freq_Data)=DC_Matrix(1:Low_Freq_Data,1:Low_Freq_Data
);
```

```
 [Comp1,H1,H_1]=Arith_Code(Data);

 Compressed_Size=size(Comp1,2)+size(Comp2,2)+size(Comp3,2);
%----------------End of the Compression -------------------------------------


%------------------------ Beginning of the Decompression
[Savep]=Decoding_Half_Matrix(nonzero1,ZeroArr1,array_size1,datasize1,LOCi1,LO
Cj1,Limited_Data1);
Savep(1:Low_Freq_Data,1:Low_Freq_Data)=DC_Matrix(1:Low_Freq_Data,1:Low_Freq_D
ata);
%'---------Inverse DST ------------------------------'
Temp(1:SizeX(1),1:SizeX(2))=Savep(1:SizeX(1),1:SizeX(2));
Savep=(idst(Temp.*Div_));
clear Temp;

%'----Inverse DCT-------------------------------------'
for i=1:SizeX(1)
      j=1;Three(1:S_(2))=0;
while(j<=SizeX(2)-1)
for k=0:S_(2)-1 Three(k+1)=Savep(i,j+k);end;
        Three=round(idct(Three.*Q));
for k=0:S_(2)-1 xp(i,j+k)=Three(k+1);end;
        j=j+S_(2);
end; %End while...
end; %End While.......

% normalization ....
% for make size for the decompressed image same orignal image size...
  xp=round(xp); S_=size(Im);
  Im=0; Im=xp(1:S_(1),1:S_(2));
 clear xp; xp=Im; clear Im;

%%%%%%%% save decompressed image and show it %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%imwrite(uint8(xp),MSAVE); %% used for grayscale images, or color
%show images.....
%imshow(uint8(xp));

End % End of Function
```

Appendix E: Chapter 7

```
%Novel Compression image method used by DCT with Enhanced Minimized-Matrix-
Size Algorithmfor high resolution Colour Images
% C3RI - Sheffeid Hallam University
%        Sheffield - United Kingdom

%---------------------------------- the main program ------------

Program_working=1; % this oprion makes the program working as
% [0]- means woking as function
%      under "main_series_2D_images_Compression_1.m"
% [1]- means the program work alone as manin program

USING_Real_Program=0; % this option make the user to use just simulation for
% compression and decompression. if "0" this means
% use simulation, "1" use real compression and decompression



if (Program_working==1)
   O =imread('D:\Lectures\Images\2D_image_to_3D_obj\Statue\1.jpg');
   Str_save1='D:\Lectures\Images\ct.MMS';
   Str_save2='D:\Lectures\Images\cd.bmp';
end;




% using Color Transformation ....
Color_Transform=1; % if "Color_Transform=1" this parameter allow to user
converts original image [RGB to YCbCr]
% else "Color_Transform=0" this parameter keep original data
blocksize=64;%Block size... 2,4,8. This means block size 2x2 or 4x4 or 8x8
Quantization_L_F=[20,30,30]; % quantization Low frequency


% Time execution....
 Total_time=cputime;


% Processing Begin ....

if (Color_Transform==1)
 Ycc_data = rgb2ycbcr(O); % Convert original image [RGB to YCbCr]
else
 Ycc_data=O;
end;



O=double(O); S_=size(O);
```

```
%---------------------- Compression and De-Compression Process
if (USING_Real_Program==0)
 [C1,IT]=Enhanced_DCT_MMS(Ycc_data(:,:,1),blocksize,Quantization_L_F(1));
  Decoded_Ycc_data(:,:,1)=IT(:,:);
   [C2,IT]=Enhanced_DCT_MMS(Ycc_data(:,:,2),blocksize,Quantization_L_F(2));
     Decoded_Ycc_data(:,:,2)=IT(:,:);

[C3,IT]=Enhanced_DCT_MMS(Ycc_data(:,:,3),blocksize,Quantization_L_F(3));
        Decoded_Ycc_data(:,:,3)=IT(:,:);

end;
%---------------------- Complete compression program saved data on a file....
if (USING_Real_Program==1)

[Compressed_Package_R]=Compress_Enhanced_DCT_MMS(Ycc_data(:,:,1),blocksize,Qu
antization_L_F(1));

[Compressed_Package_G]=Compress_Enhanced_DCT_MMS(Ycc_data(:,:,2),blocksize,Qu
antization_L_F(2));

[Compressed_Package_B]=Compress_Enhanced_DCT_MMS(Ycc_data(:,:,3),blocksize,Qu
antization_L_F(3));

  Compressed_Package.Compressed_Package_B=Compressed_Package_B;
  Compressed_Package.Compressed_Package_G=Compressed_Package_G;
  Compressed_Package.Compressed_Package_R=Compressed_Package_R;
  Compressed_Package.Color_Transform=uint8(Color_Transform);
%
  save(Str_save1,'Compressed_Package');
end;
%---------------------------------------- End of Compression--------------

%----------------Complete Decompression program read data from compressed
file---
clear Compressed_Package;
clear Compressed_Package_R;
clear Compressed_Package_G;
clear Compressed_Package_B;
clear Ycc_data; % Clear memory before start...

if (USING_Real_Program==1)
 Header=load(Str_save1, '-mat');
 Compressed_Package=Header.Compressed_Package;
 clear Header;  % delete header from memory

  Compressed_Package_B=Compressed_Package.Compressed_Package_B;
   Compressed_Package_G=Compressed_Package.Compressed_Package_G;
    Compressed_Package_R=Compressed_Package.Compressed_Package_R;

  Color_Transform=Compressed_Package.Color_Transform;
  [IT]=Decompress_Enhanced_DCT_MMS(Compressed_Package_R);
   Decoded_Ycc_data(:,:,1)=IT(:,:);
   clear IT; % delete "IT" from memory
  [IT]=Decompress_Enhanced_DCT_MMS(Compressed_Package_G);
   Decoded_Ycc_data(:,:,2)=IT(:,:);
  clear IT; % delete "IT" from memory
```

```
  [IT]=Decompress_Enhanced_DCT_MMS(Compressed_Package_B);
   Decoded_Ycc_data(:,:,3)=IT(:,:);
  clear IT; % delete "IT" from memory
end;
%---------------------------------------Reconstruct image ---------------
Decoded_Ycc_data=uint8(Decoded_Ycc_data);
if (Color_Transform==1)
  re_O= ycbcr2rgb(Decoded_Ycc_data); % Convert Decompressed image [YCbCr to
RGB]
else
  re_O=Decoded_Ycc_data;
end;
%--------------Put the Cross-Pointes in exact location -----------------
clear Decoded_Ycc_data;
clear Compressed_Package;


%--------------------------------------

if (USING_Real_Program==0)
'Compress size in Kbytes = '
        Compressed_Size=(C1+C2+C3)/1024
end;

 imshow(re_O);
if (Program_working==1)
   imwrite(re_O,Str_save2);
end;
'Image Quality'
% compute the quality
        [RMSE, SNR,P_SNR]=Peak_SNR(re_O,O);
         RMSE

'Total Time execution is :'
 Total_time=cputime-Total_time


----------------------------------------------------------------------

function [Compressed_Package]=Compress_Enhanced_DCT_MMS(I,Block_Size,R)

Compress_Time=cputime;
Diff=I;
%---- Apply DCT on the low freqeuncies....
S_=size(Diff);

 Q(1:Block_Size,1:Block_Size)=0;
for i=1:Block_Size
for j=1:Block_Size
         Q(i,j)=(i+j)*R;
end;
end;
% Create new matrix for transformation...
SAVE_Data(1:floor((S_(1)*S_(2))/(Block_Size*Block_Size)),1:Block_Size*Block_S
ize)=0;
```

```
Diff(S_(1):S_(1),S_(2):S_(2))=0;
L=1; i=1; %Data(1:Block_Size*Block_Size)=0;
while(i<S_(1)-Block_Size)
    j=1;
while(j<S_(2)-Block_Size)
     Data(1:Block_Size,1:Block_Size)=Diff(i:i+Block_Size-1,j:j+Block_Size-1);
     Data=round(dct2(Data)./Q);
     D=reshape(Data,1,Block_Size*Block_Size);
     SAVE_Data(L,1:Block_Size*Block_Size)=D(1:Block_Size*Block_Size);
      L=L+1;
      j=j+Block_Size;
end;
    i=i+Block_Size;
end;

% Eliminate half of matrix
SAVE_Data(1:size(SAVE_Data,1),floor(size(SAVE_Data,2)/2):size(SAVE_Data,2))=0
;

D1(1:size(SAVE_Data,1))=SAVE_Data(1:size(SAVE_Data,1),1);
SAVE_Data(1:size(SAVE_Data,1),1)=0;


% reduce matrix size to half.... by elimiate most insiginficat information..(
less information high freqeuncies)
SAVE_Half=SAVE_Data(1:size(SAVE_Data,1),1:floor(size(SAVE_Data,2)/2)-1); %
save half of transformed matrix before coding...

[nonzero1,ZeroArr,array_size,datasize,LOCi,LOCj,Limited_Data] =
Coding_Half_Matrix(SAVE_Half);



% Compress the DC-Values....
S_2=size(D1);
D1(1:S_2(2)-1)=D1(1:S_2(2)-1)-D1(2:S_2(2));
[Compressed_Data1,H1,H12]=Arith_Code(D1);
[Compressed_Data2,H2,H22]=Arith_Code(nonzero1);
[Compressed_Data3,H3,H32]=Arith_Code(ZeroArr);
[Compressed_Data4,H4,H42]=Arith_Code(LOCi);
[Compressed_Data5,H5,H52]=Arith_Code(LOCj);

 Compressed_Package.Compressed_Data1=uint8(Compressed_Data1);
 Compressed_Package.H12=int32(H12);
 Compressed_Package.H1=uint16(H1);

 Compressed_Package.Compressed_Data2=uint8(Compressed_Data2);
 Compressed_Package.H22=int32(H22);
 Compressed_Package.H2=uint16(H2);

 Compressed_Package.Compressed_Data3=uint8(Compressed_Data3);
 Compressed_Package.H3=uint32(H3); Compressed_Package.H32=int32(H32);
```

```
 Compressed_Package.Compressed_Data4=uint8(Compressed_Data4);
 Compressed_Package.H4=uint16(H4); Compressed_Package.H42=int32(H42);

 Compressed_Package.Compressed_Data5=uint8(Compressed_Data5);
 Compressed_Package.H5=uint16(H5); Compressed_Package.H52=int32(H52);


%%% Split the Limited-Data to KEY and Limited-Data

Compressed_Package.KEY=int64([Limited_Data(1),Limited_Data(2),Limited_Data(3)
,Limited_Data(4),Limited_Data(5)]);


 Compressed_Package.Limited_Data=[];
for L=6:size(Limited_Data,2)

Compressed_Package.Limited_Data=int16([Compressed_Package.Limited_Data,Limite
d_Data(L)]);
end;

 Compressed_Package.array_size=uint32(array_size);
 Compressed_Package.datasize=uint32(datasize);
 Compressed_Package.SAVE_Size=uint32(size(SAVE_Data));
 Compressed_Package.Diff_Size=uint32(size(Diff));
 Compressed_Package.Block_Size=uint8(Block_Size);
 Compressed_Package.R=uint8(R);


'show Compressed time (sec.) by Arithmetic Coding and Block Transformation'
 Compress_Time=cputime-Compress_Time
%%%% ---------------- END of Compression --------------------
end
```

Appendix F: Chapter 8

```
% 3DPointCloud Data and Mesh Connectivity Compression by a Novel Geometry
Minimization Algorithm
% This program to Compress 3D Triangle faces (mesh)
% by using Minimize-Matrix-Size Algorithm
% under test.......
% 04 - Aug - 2015
% by Mohammed M. Siddeq
%-------------------------------------------------------
clear;
   Path_='D:\Lectures\Images\Image3D\shape_3D\';
   write_='test.obj';
   read_='face_1.obj';
   matlab_='face_1.mat';
% read/write 3D object file .....
   sp=size(Path_);
   write_to_path=Path_; write_to_path(sp(2)+1:sp(2)+size(write_,2))=write_;
   Read_from_path=Path_; Read_from_path(sp(2)+1:sp(2)+size(read_,2))=read_;
   mat_Path=Path_; mat_Path(sp(2)+1:sp(2)+size(matlab_,2))=matlab_;

% use this function if you like to read 3D OBJ file data
%[H,v,vt,f1,f2,f3]=read_3D_obj_file(Read_from_path);

% use this function when you have all variables in MATLAB...

   load(mat_Path);

   Diff=1; % This variable used to compute the differences
%[1]- between two adjacent data compute differences,
%[2]- compute differenes for each column,

   Shift=10; % Quanize the vertices by using shift data left
% this option is allow to you change to Automatic shift
% or manual chnage. [0] - Auto Change, [1,2,3,..]- Manual Change
% [1,5,10,20,30,..10000] is number of postions to shift.
% work as factor
        v=v*Shift;  v1=round(v);
        xyz(1:size(v,1),1)=v1(1:size(v,1),1);
        xyz(1:size(v,1),2)=v1(1:size(v,1),2);
        xyz(1:size(v,1),3)=v1(1:size(v,1),3);
        clear v1;

xyz(1:size(v,1)-1,1)=xyz(1:size(v,1)-1,1)-xyz(2:size(v,1),1);
xyz(1:size(v,1)-1,2)=xyz(1:size(v,1)-1,2)-xyz(2:size(v,1),2);
xyz(1:size(v,1)-1,3)=xyz(1:size(v,1)-1,3)-xyz(2:size(v,1),3);

% Compress Vertcies... my using MMS-Algorithm
[Decoded_v,Compressed_v,v_Table]=MMS_3D_SS_Algorithm(xyz);

i=size(Decoded_v,1);
while (i>1)
    Decoded_v(i-1,1)=Decoded_v(i-1,1)+Decoded_v(i,1);
    Decoded_v(i-1,2)=Decoded_v(i-1,2)+Decoded_v(i,2);
    Decoded_v(i-1,3)=Decoded_v(i-1,3)+Decoded_v(i,3);
    i=i-1;
end;
```

```
% saved compressed vertices data in different file...
save(strcat(Path_,'Compressed_v.mat'),'Compressed_v','-mat');
%------------------End part one-----------------------------------------


%-------------------- Compress 3D Trinagles by using MMS-Algorithm
[Decoded_f1,CF,count,CF2,CF4,count2,CF5]=MMS_3D_Triangle(f1,Diff);

save(strcat(Path_,'Compressed_3D_Data.mat'),'CF','CF2','CF4','CF5','count','c
ount2','-mat');


% saved compressed Triangle faces data in different file...

% Last part ---------------Save vertices and triangle faces (just 3D mesh)
 write_3D_obj_to_file(H,Decoded_v,[],Decoded_f1,[],[],write_to_path);
%-------------------------------------------------------------------------
% 3D RMSE computed between original cloude data and estmated data.
'3D RMSE for x,y,z vertices'
[RMSE, SNR,P_SNR]=Peak_SNR(v,Decoded_v);
RMSE

-----------------------------------------------------------------------

function [Decoded_f1,CF,count,CF2,CF4,count2,CF5]=MMS_3D_Triangle(f1,Diff)
%-------------------- Compute the Difference between Trinagles


if (Diff==1)
% first option for compute differences...
f1=f1';
Data_f1(1:size(f1,1)*size(f1,2))=f1(1:size(f1,1),1:size(f1,2));
Data_f1(1:size(Data_f1,2)-1)=Data_f1(1:size(Data_f1,2)-1)-
Data_f1(2:size(Data_f1,2));
end;


if (Diff==2)
% second option for compute differences...
f1(1:size(f1,1)-1,1)=f1(1:size(f1,1)-1,1)-f1(2:size(f1,1),1);
f1(1:size(f1,1)-1,2)=f1(1:size(f1,1)-1,2)-f1(2:size(f1,1),2);
f1(1:size(f1,1)-1,3)=f1(1:size(f1,1)-1,3)-f1(2:size(f1,1),3);
Data_f1(1:size(f1,1)*size(f1,2))=f1(1:size(f1,1),1:size(f1,2));
end;


%-------------------------------------------------------------------------
L=1; Threshold_value=30000; % this veriable used to remove big values
% save small values...........

                            Big_Data=0;
for i=1:size(Data_f1,2)
```

```
if (abs(Data_f1(i))>=Threshold_value) Big_Data(L)=Data_f1(i); L=L+1;
Data_f1(i)=Threshold_value; end;
end;

Data_f1=reshape(Data_f1,size(f1,1),size(f1,2));

% in case if option worked.... else nothing happen
if (Diff==1)  Data_f1=Data_f1'; end;


[Decoded_f1,Compressed_f1,f1_Table]=MMS_3D_SS_Algorithm(Data_f1);

if (Diff==1) Decoded_f1=Decoded_f1'; end;
f1_(1:size(Decoded_f1,1)*size(Decoded_f1,2))=Decoded_f1(1:size(Decoded_f1,1),
1:size(Decoded_f1,2));

L=1;
for i=1:size(f1_,2)
if (f1_(i)==Threshold_value)
        f1_(i)=Big_Data(L); L=L+1;
end;
end;


if (Diff==1)
    i=size(f1_,2);
while (i>1)
        f1_(i-1)=f1_(i-1)+f1_(i);
        i=i-1;
end;

 Decoded_f1=reshape(f1_,size(f1,1),size(f1,2));
 Decoded_f1=Decoded_f1';
end;


if (Diff==2)

 Decoded_f1=reshape(f1_,size(Data_f1,1),size(Data_f1,2));

 i=size(Decoded_f1,1);
while (i>1)
    Decoded_f1(i-1,1)=Decoded_f1(i-1,1)+Decoded_f1(i,1);
    Decoded_f1(i-1,2)=Decoded_f1(i-1,2)+Decoded_f1(i,2);
    Decoded_f1(i-1,3)=Decoded_f1(i-1,3)+Decoded_f1(i,3);
    i=i-1;
end;
end;




%-------------------------------------------------------------------------
```

```
% Apply Arithmetic Coding ....
S_=size(Compressed_f1); CF=[]; CF2=[]; CF3=[]; count=[];

% if array size bigger than 1000 choose block size 100, else block size 10
if (S_(2)>=1000)
    Block=100; pad=100;
else
    Block=S_(2); pad=0;
end;

i=1;
while (i<(S_(1)*S_(2))-pad)
     Block_f1(1:Block)=Compressed_f1(i:i+Block-1);
     i=i+Block;
if (max(Block_f1(:))==min(Block_f1(:)))
         A1=max(Block_f1(:)); A2=0; A3=0;
else
          [A1,A2,A3]=Arith_Code(Block_f1);
end;
      CF=[CF, uint8(A1)];
      count=[count,uint8(size(A1,2)),uint8(size(A3,2))];
      CF2=[CF2,int32(A3)];
      CF3=[CF3,uint32(A2)];
end;
%-----------------------------
%count2=uint8(0); CF5=uint8(0);
 S_=size(Big_Data); CF4=[]; CF5=[]; CF6=[]; count2=[];
Block=100; i=1;
while (i<(S_(1)*S_(2))-Block)
     Block_f1(1:Block)=Big_Data(i:i+Block-1);
     i=i+Block;
if (max(Block_f1(:))==min(Block_f1(:)))
         A1=max(Block_f1(:)); A2=0; A3=0;
else
          [A1,A2,A3]=Arith_Code(Block_f1);
end;
      CF4=[CF4, uint8(A1)];
      count2=[count2,uint8(size(A1,2)),uint8(size(A3,2))];
      CF5=[CF5,int32(A3)];
      CF6=[CF6,uint32(A2)];
end;
% just check if Big_Data didn't compressed.....
if (S_(2)==1) CF4=0; CF5=0; CF6=0; count2=0; end;




end % End of function....
```