

A Scalable, Secure, and Energy-Efficient Image Representation for
Wireless Systems

by

Tim Ho Tin Woo

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2004

© Tim Ho Tin Woo 2004

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The recent growth in wireless communications presents a new challenge to multimedia communications. Digital image transmission is a very common form of multimedia communication. Due to limited bandwidth and broadcast nature of the wireless medium, it is necessary to compress and encrypt images before they are sent. On the other hand, it is important to efficiently utilize the limited energy in wireless devices. In a wireless device, two major sources of energy consumption are energy used for computation and energy used for transmission. Computation energy can be reduced by minimizing the time spent on compression and encryption. Transmission energy can be reduced by sending a smaller image file that is obtained by compressing the original highest quality image. Image quality is often sacrificed in the compression process. Therefore, users should have the flexibility to control the image quality to determine whether such a tradeoff is acceptable. It is also desirable for users to have control over image quality in different areas of the image so that less important areas can be compressed more, while retaining the details in important areas. To reduce computations for encryption, a partial encryption scheme can be employed to encrypt only the critical parts of an image file, without sacrificing security. This thesis proposes a scalable and secure image representation scheme that allows users to select different image quality and security levels. The binary space partitioning (BSP) tree presentation is selected because this representation allows convenient compression and scalable encryption. The Advanced Encryption Standard (AES) is chosen as the encryption algorithm because it is fast and secure. Our experimental result shows that our new tree construction method and our pruning formula reduces execution time, hence computation energy, by about 90%. Our image quality prediction model accurately predicts image quality to within 2-3dB of the actual image PSNR.

Acknowledgements

I would like to thank my supervising professors, Professor C. Gebotys and Professor S. Naik for their advices given to me throughout the research and the writing of this thesis. Also, their financial support, provided to me in the form of a Research Assistantship, is greatly appreciated.

I would also like to thank my family and my friends for their support.

Finally, I am thankful to the Department of Electrical and Computer Engineering, University of Waterloo, for providing me with the necessary resources and a comfortable environment for conducting my research.

Table of Contents

CHAPTER 1 – INTRODUCTION.....	1
1.1 MOTIVATION	1
1.2 CONTRIBUTION.....	4
1.3 THESIS OUTLINE.....	5
CHAPTER 2 – BACKGROUND AND PREVIOUS RESEARCH.....	6
2.1 IMAGE COMPRESSION	6
2.2 BINARY SPACE PARTITIONING OF IMAGES.....	6
2.2.1 <i>Overview.....</i>	6
2.2.2 <i>Line Selection Criterion</i>	7
2.2.3 <i>Extension to Colour Images.....</i>	9
2.3 OTHER COMMON COMPRESSION TECHNIQUES.....	9
2.3.1 <i>TIFF.....</i>	10
2.3.2 <i>JPEG.....</i>	10
2.3.3 <i>Quadtree Compression</i>	12
2.3.4 <i>Wavelet Compression.....</i>	13
2.3.5 <i>Fractal Image Compression.....</i>	14
2.4 ADVANCED ENCRYPTION STANDARD.....	15
2.5 PARTIAL ENCRYPTION	16
2.5.1 <i>Overview.....</i>	16
2.5.2 <i>JPEG Partial Encryption.....</i>	17
2.5.3 <i>Quadtree Partial Encryption.....</i>	19
2.5.4 <i>Wavelet Partial Encryption.....</i>	19
2.6 SUMMARY	20
CHAPTER 3 – METHODOLOGY	23
3.1 SYSTEM OVERVIEW	23

3.2	GENERATION OF THE HIGHEST QUALITY IMAGE TREE.....	24
3.2.1	<i>Binary Split Tree Construction Method</i>	24
3.3	TESTBENCH MODULE	25
3.4	COMPRESSION	25
3.4.1	<i>Error Formula for Pruning</i>	26
3.4.2	<i>Error Representation at Nodes</i>	28
3.4.3	<i>Compression by Use of Colour Table</i>	29
3.4.4	<i>Using Custom Bit Length for Storing Integers</i>	29
3.5	SCALABLE ENCRYPTION MODULE.....	30
3.6	FINAL OUTPUT IMAGE.....	30
CHAPTER 4 – EXPERIMENTAL SETUP AND METRICS.....		33
4.1	PLATFORM.....	33
4.2	TEST IMAGE FILES	33
4.3	MEASURING PERFORMANCE	35
4.3.1	<i>Execution Time</i>	35
4.3.2	<i>Compression</i>	35
4.3.3	<i>Image Quality</i>	35
CHAPTER 5 – EXPERIMENTAL RESULTS		37
5.1	EFFECT OF TREE CONSTRUCTION MECHANISM.....	37
5.2	EFFECT OF ERROR CALCULATION FORMULA ON PRUNING	38
5.2.1	<i>Overview</i>	38
5.2.2	<i>Effect on Image Quality and File Size</i>	39
5.2.2.1	Total Square Error Formula	39
5.2.2.2	Fast Error Formula	42
5.2.3	<i>Effect on Execution Time</i>	46
5.2.4	<i>Average Error versus Total Error Representation</i>	46
5.3	DATA BIT COMPRESSION	48
5.3.1	<i>Line Coordinate File Compression</i>	48

5.3.2	<i>Colour Table Compression</i>	49
5.4	SUMMARY	50
CHAPTER 6 – ANALYSIS.....		51
6.1	FILE SIZES.....	51
6.1.1	<i>Tree File and Line Orientation File Size</i>	51
6.1.2	<i>Line Coordinate File Size</i>	52
6.1.3	<i>Colour File Size</i>	52
6.1.4	<i>Colour Table Size</i>	53
6.1.5	<i>Summary</i>	54
6.2	PERFORMANCE.....	54
6.2.1	<i>Operations required for the Tree Construction Mechanisms</i>	54
6.2.2	<i>Error Calculation Formula</i>	57
6.2.3	<i>Summary</i>	58
6.3	IMAGE QUALITY.....	58
6.3.1	<i>Execution Time versus Image Quality Threshold</i>	59
6.3.2	<i>Image Quality versus File Size</i>	60
6.3.3	<i>Comparison to other Image Compression Systems</i>	62
6.3.4	<i>Summary</i>	66
6.4	SCALABLE ENCRYPTION	66
6.4.1	<i>Encryption Time versus File Size</i>	66
6.4.2	<i>Encryption Time versus Partial Encryption Percentage</i>	68
6.4.3	<i>Encryption Time versus Encryption Key and Block Length</i>	69
6.5	SELECTIVE FOCUS REGIONS FEATURE.....	70
6.6	IMAGE QUALITY PREDICTION MODEL	71
6.7	FILE SIZE PREDICTION MODEL.....	73
CHAPTER 7 – DISCUSSION AND CONCLUSION		76
7.1	COMPARISON TO PREVIOUS RESEARCH	76

7.2	LIMITATIONS, ADVANTAGES AND DISADVANTAGES.....	77
7.3	CONCLUSION	78
7.4	FUTURE WORKS	79

List of Tables

Table 3.1– Security Level Parameters.....	25
Table 5.1 – Tree Construction Time for Various Images	38
Table 5.2 – Peppers Image Bit Rates Using Total Square Error Pruning.....	42
Table 5.3 – Frymire Image Bit Rates Using Total Square Error Pruning.....	42
Table 5.4 – Peppers Image Bit Rates Using Fast Error Pruning.....	45
Table 5.5 – Frymire Image Bit Rates Using Fast Error Pruning.....	45
Table 5.6 – Execution Time (ms) for Computing Node Error.....	46
Table 5.7 – Line Coordinate Compression File Sizes for Various Images	49
Table 5.8 – Colour Table Compression File Sizes for Various Images.....	49
Table 6.1 – Pruning Time at different thresholds	60
Table 6.2 – TIFF and BSP Tree File Sizes for Various Images	62
Table 6.3 – JPEG and BSP Tree File Sizes for Peppers Image.....	65
Table 6.4 – JPEG and BSP Tree File Sizes for Frymire Image.....	65
Table 6.5 – Encryption Time at different thresholds.....	67
Table 6.6 – Encryption Time (ms) for Different Key and Block Length.....	69
Table 6.7 – Parameters for Image Quality Prediction Model	73
Table 6.8 – Parameters for File Prediction Model.....	75
Table 7.1 – Compression performance compared to other systems	77

List of Figures

Figure 2.1 – Representation of Image in BSP tree form	7
Figure 2.2 – JPEG zigzag pattern for 8×8 block.....	11
Figure 2.3 – Representation of Image in quadtree form	13
Figure 2.4 – AES “State” block.....	16
Figure 2.5 – Resulting JPEG image after discarding DC and AC coefficients	18
Figure 3.1 – Proposed Image Compression and Encryption System.....	23
Figure 3.2 – Compression Using Colour Table	29
Figure 3.3 – BSP tree traversal order for serializing to file	32
Figure 4.1 – Test Images Used for this Thesis.....	34
Figure 5.1 – Peppers Image at different thresholds Using Total Square Error Pruning	40
Figure 5.2 – Frymire Image at different thresholds Using Total Square Error Pruning.....	41
Figure 5.3 – Peppers Image at different thresholds Using Fast Error Pruning	43
Figure 5.4 – Frymire Image at different thresholds Using Fast Error Pruning.....	44
Figure 5.5 – Peppers Image at different thresholds Using Average Fast Error Pruning.....	47
Figure 5.6– PSNR versus Bit Rate for Average and Total Error Representation	48
Figure 6.1 – Pruning Time versus Pruning Threshold.....	59
Figure 6.2 – PSNR versus Normalized File Size	60
Figure 6.3 – PSNR versus Bit Rate	61
Figure 6.4 – Peppers JPEG and BSP Tree Image at the best compression and quality settings.....	63
Figure 6.5 – Frymire JPEG and BSP Tree Image at the best compression and quality settings.....	64
Figure 6.6 – Encryption Time versus Encrypted File Size.....	67
Figure 6.7 – Encryption Time versus Percentage of Tree File Encrypted	68
Figure 6.8 – Encryption Time versus Key and Block Length	69

Figure 6.9 – Lena Image with Selective Threshold Regions	71
Figure 6.10 – Estimated PSNR versus threshold	72
Figure 6.11 – Estimated PSNR versus File Size	74

Chapter 1 – Introduction

1.1 Motivation

With the increasing demand and advances in digital image technology, many cellular phones and personal digital assistants (PDAs) are now equipped with colour displays and digital lens. Users can now take and send pictures to share with others very conveniently with these embedded devices. The increasing popularity of wireless digital image communication presents a new challenge to image compression and encryption. This is because the requirements for sending images in a wireless environment with handheld portable devices are in many ways different from a conventional environment.

Wireless communication differs from wired communication in several ways. First, wireless communication has a different security requirement. Wireless medium is physically insecure because anyone can receive the signals in open air. Therefore, to avoid unauthorized access to the data, it is necessary to encrypt the data before it is sent. Also, wireless medium has a much lower bandwidth than wired medium. In addition, wireless communication is often asymmetrical: the uplink is often much slower than the downlink. For example, the latest European Universal Mobile Telecommunications System (UMTS) standard for 3G cellular systems can be configured in asymmetrical mode to allocate more channels for downlink [1]. Therefore, it is necessary to compress the data before transmission to save transmission time, since the uplink may be slow even for such “high-speed” wireless systems. The inherent insecurity and lower bandwidth of the wireless medium and limited energy in wireless devices call for the use of a good compression and encryption algorithms before an image is transmitted.

Portable devices run on a limited energy source such as batteries. Therefore, to prolong the time before recharging or replacement of the batteries, energy utilization must be efficient. From a software perspective, there are several strategies for optimizing energy [2], [3], [4]. These strategies focus on minimizing the energy cost for computations and memory accesses. For a wireless device, in addition to energy costs for computations and memory accesses, energy used for RF transmission is also a significant source of energy consumption. Therefore, the *total* energy

consumption consists of energy used for computations, data accesses, and transmission energy. This thesis focuses on reducing the computation energy and transmission energy since they are the two most significant sources of energy consumption. It is found in [5] that a typical CPU for PDA consumes about 90-150mW in running mode, 36mW in idle mode, and 0.9mW in sleeping mode. The transceiver consumes about 210-540mW for transmitting and 180-240mW for receiving data. Therefore the power used for computation and transmission is quite significant. The energy consumed is equal to the power multiplied by the actual execution/transmission time it takes for such tasks; hence, to minimize computation energy, one can reduce the execution time of the compression and the encryption algorithm, so that less energy is consumed in the switching activities in the processor when computing. To reduce transmission energy, one can reduce the amount of data to be sent, which is achieved by reducing the image file size by compression.

The compression and encryption system must maintain a good quality for the image. There are often tradeoffs between these different goals. For example, to have a higher quality image, the image would often require more storage space, which increases the energy cost for encryption and transmission. As another example, to increase security, it would cost more energy to encrypt the image by selecting a more secure key and block length and/or increase the fraction of the file to be encrypted. Although the study of image quality and execution time tradeoff is not a novel idea, it deserves special attention for a wireless device because of its limited computation power, memory and energy source. At the time of writing, a typical PC consists of a 2.8GHz processor, with 512MB RAM. A typical PDA (e.g. PalmOne Tungsten T3) consists of a 400MHz processor, with 64MB RAM, which is only a fraction of the computation power and memory available to a PC. Therefore, it is especially crucial for the image compression and encryption system to be optimized for a wireless PDA.

Because of these energy/image quality/file size tradeoffs, it is necessary for the image representation system to be flexible and scalable. It is the end user who needs to ultimately decide how secure and how good that an image has to be. Therefore, the user should have the flexibility to be able to control the security level and quality level of the image. In summary, the main requirements for wireless devices for image communications are:

1. **Image Quality** – Images should be of acceptable quality to the user. The *perception* of image quality is important. The compression system should retain the details of the perceptually important areas of the image.
2. **Compression** – Images should be compressed to reduce bandwidth requirements and save transmission energy costs.
3. **Security** – Since anyone can intercept the signals in the wireless medium, images should be encrypted to provide privacy. At the same time, encryption energy costs should be minimized by minimizing the amount of data that need to be encrypted.
4. **Scalability** – The image representation system should allow flexible compression. Since image quality is often compromised in the compression process, the image representation system should provide a mechanism to facilitate convenient manipulation of the image according to user requirements. For example, it should allow the image to compress at different compression ratios. It should also allow the user to use different compression ratios for different regions of the image such that the details in the perceptually important regions of the image are retained.
5. **Energy Efficient** – The image representation system should be kept simple so that the wireless devices should be able to generate, compress, and encrypt an image with a few computations. This saves computation energy costs in an energy constrained wireless device. Transmission energy cost is reduced by using an efficient compression system to reduce the amount of data being sent by the transceiver.

This thesis proposes an image compression and encryption system that is energy efficient, provides a mechanism for tradeoffs in image quality, and provides the users the flexibility to adjust the image security and quality as desired. Ideally, the system should be able to adapt to changes in the environment. For example, when the bandwidth is low or when the battery energy is low, the application should be able to adjust the image quality and security settings. The work presented here provides the flexibility to accomplish that by allowing user to supply different input parameters. This work can be extended to fit as part of a Quality of Service framework such as the one proposed in [6].

To provide flexibility and high compression performance, while still being energy efficient, this thesis proposes to use the Binary Space Partitioning (BSP) tree structure to represent an image [14]. BSP tree representation allows convenient compression to different image quality in different regions of the same image. The tree structure also allows convenient scalable encryption. Because of the tree structure representation, only a fraction of the entire image file needs to be encrypted while still providing good security. We have selected the Advanced Encryption Standard (AES) as the cryptographic algorithm, although BSP tree representation will support any block cipher algorithms such as DES or RC4. To enhance compression ratio, advanced coding techniques can be incorporated into our proposed system.

1.2 Contribution

The thesis's contributions are:

1. We have presented a simple yet efficient BSP tree construction technique for quickly converting an image into the BSP form. This technique is presented in Section 3.2.1.
2. We have extended an existing BSP greyscale image compression algorithm proposed in [14] to work for colour images in Section 2.2.3. We have also proposed a new compression scheme to compress colour images by using a colour index table in Section 3.4.3.
3. We have proposed a new pruning formula for calculating node errors in BSP tree lossy compression in Section 3.4.1. This formula is especially effective with our new tree construction method.
4. We have applied the concept of *partial encryption* to BSP tree compression. See Section 3.5.
5. In Section 6.5, we have introduced the concept of *selective threshold* for images. Our system allows the user to select a different image quality level for different areas of the image. This allows the user to compress the image without losing the details in

important areas of the image; thus, the impact on the *perception* of image quality is minimal.

6. We have analyzed the relationships between image quality and different pruning parameters. Using these relationships, we have introduced an image quality and file size prediction model in Section 6.6 and Section 6.7 for predicting the resulting image quality *before* the image is generated.

1.3 Thesis Outline

This thesis is organized as follows. *Chapter 2 –Background and Previous Research* gives the background concepts required to understand this thesis. It also discusses some previous work on related areas. *Chapter 3 –Methodology* gives the system overview of the components of the proposed system. *Chapter 4 –Experimental Setup and Metrics* shows how the experiments are set up. *Chapter 5 –Experimental Results* shows the experimental results by varying different parameters. *Chapter 6 –Analysis* analyzes the new proposed scheme in detail. Finally, *Chapter 7 –Discussion and Conclusion* discusses how the results in this thesis compare to from previous work, followed by a summary of the advantages and limitations of the new scheme and the concluding remarks. It will also highlight some future directions for the thesis work.

Chapter 2 – Background and Previous Research

2.1 Image Compression

Image compression algorithms reduce image size by removing redundant information. Compression can be lossless or lossy. In lossless compression, no image data is lost. The compressed image looks the same as the original image. In lossy compression, the compression process removes data to compress. There will be a loss in image quality and the compression process is irreversible, meaning that one can never get the original image quality back after the compression.

2.2 Binary Space Partitioning of Images

2.2.1 Overview

Binary space partitioning (BSP) tree has been used in several application areas. First, it is a convenient representation for 3D-graphics for representation of spatial relationship between objects. It allows easy determination of which objects should lie in front of or hide from the other objects, thus allowing quick rendering of visibility of object surfaces [9],[10]. It also has important applications in image processing. For example, since information that is more important is organized closer to the root node, BSP tree structure allows quick recognition of objects [11] and quick information retrieval [12], [13].

In [14], Radha, Vetterli and Leonardi have proposed the use of a binary space partitioning (BSP) tree to represent an image. In this representation, the image is broken down into simple geometric regions using partitioning lines. The image is divided until the pixels in the region are homogenous (for lossless mode) or similar enough (for lossy mode). Figure 2.1 shows how an

image of a square is partitioned into rectangular regions. Note that BSP tree partitioning lines align with the actual object boundaries, resulting in a very efficient and compact storage.

Unlike our BSP representation, their version of BSP tree supports slanted lines. The advantage of supporting slanted lines is that the partitioning lines will align with actual object boundaries better. The tradeoff is that the number of line candidates considered at each partitioning iteration is significantly increased, which resulted in more computations in the line selection process. To reduce the energy cost for computations, our BSP representation only allows horizontal and vertical partition lines.

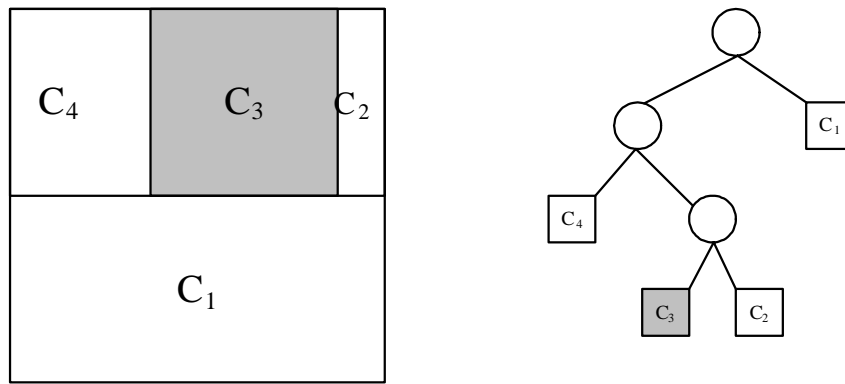


Figure 2.1 – Representation of Image in BSP tree form

BSP tree technique compresses an image by merging pixels that are homogeneous (lossless mode) or similar enough (lossy mode), thus reducing the memory used to store the colour of each pixel. In the final BSP tree, each non-terminal node is associated with a partition line and each leaf node is associated to a particular region of the image. The non-terminal nodes store the line parameters while the leaf nodes store the colour of the associated regions.

2.2.2 Line Selection Criterion

To select the partitioning lines, Radha, Vetterli and Leonardi propose a Least-Square-Error (LSE), recursive method in [15] for *greyscale* images. Since this method yields the optimal partition line selection, it will be referred to as the “Optimal Line Selection Method” in this thesis. In this method, a *LSE Partitioning Line (LPL) transform* is calculated for all the potential candidate lines. The LPL transform $L(h)$ of a 2-D continuous function $I(x,y)$ over a region R is defined as:

$$L(h) = \frac{\int_{x_1}^{x_2} xI(x, h)dx}{\int_{x_1}^{x_2} I(x, h)dx} \quad (2.1)$$

where h is a straight line intersecting the boundary of region R at two points p_1 and p_2 , and x_1 and x_2 are the x -coordinates of p_1 and p_2 , respectively. They state that there is a necessary condition for a line to be optimal. The condition is:

$$|L(h) - x_c(h)| < T_c \quad (2.2)$$

where T_c is a small positive number and x_c is the midpoint between x_1 and x_2 and is calculated by:

$$x_c \equiv \frac{1}{2}(x_1 + x_2) \quad (2.3)$$

The LPL transform of Eq. (2.1) and the condition in Eq. (2.2) eliminate all the candidate lines that are certainly not optimal. For the remaining lines, the total square error is calculated. At any iteration i in the recursive partitioning process, the total square error E_i for a region of dimension $M \times N$ is defined as:

$$E_i = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [I(x, y) - \hat{I}(x, y)]^2 \quad (2.4)$$

where $I(x,y)$ is the actual pixel value at (x,y) and $\hat{I}(x,y)$ is the mean value used to approximate the pixels in the region. The line with the least total error is selected as the partitioning line.

To reduce the computations for partitioning an image, the partition lines in our work are simply either horizontal lines or vertical lines. Each non-terminal node stores the line coordinate and a bit flag indicating whether that line is horizontal or vertical.

2.2.3 Extension to Colour Images

Since Eq. (2.1), (2.2) and (2.4) are all defined for greyscale images only, we have modified them for colour images. Unlike a greyscale image, where each pixel has only one value, a colour image has three values for each pixel. There exist several colour coordinate systems, each with their own advantages and disadvantages. Since the RGB system is the one used for Windows Bitmap, it is the system used in the experimental work.

RGB stands for red, green and blue, and these are the respective colours in each coordinate. Before applying Eq. (2.1), (2.2) and (2.4) to colour images, the RGB colour pixels are first converted to the luminance value (Y) according to Eq. (2.5):

$$Y = 0.299R + 0.587G + 0.114B \tag{2.5}$$

The luminance value is a weighted average of the three colours according to the human eye's sensitivity to each colour [8]. Other than RGB, there are several colour systems, such as YCbCr and CMY. No matter which colour coordinate system is chosen, we can always convert them to the luminance value since all the colour coordinate systems are interchangeable [8]. After applying Eq. (2.5), we now have a single intensity value for each pixel. Next, we directly apply the line selection algorithm and error calculation formula on the image as if it was a greyscale image.

Our extension to colour image is very simple compared to other colour thresholding schemes such as the one proposed in [7]. Therefore, it is easy to implement and this saves energy in execution.

2.3 Other Common Compression Techniques

There exist other good compression algorithms, such as JPEG [8], quadtree [39], wavelet [37] and fractal compression [27]. Reference [41] gives a good reference on these algorithms.

2.3.1 TIFF

TIFF is an acronym for “Tagged Image File Format” [16]. It is developed by Aldus (now merged with Adobe Systems) and Microsoft Corporation in an effort to provide a platform independent image format for different image processing applications. It supports many different sets of colour coordinate systems such as RGB, YCbCr and CMYK. It also supports uncompressed or compressed mode. The most common compression algorithm used in TIFF file is the (Lempel-Ziv-Welch) LZW compression, which is the same algorithm used for ZIP files [17].

LZW is a lossless data compression algorithm developed for text data [18]. It compresses data by substituting frequently occurring data patterns with the set of symbols in the “dictionary” that is expanded in vocabulary as the compression algorithm progresses. However, since this algorithm is designed for text data, its compression performance is not as great as other specifically designed image compression algorithms. In our experimental results in Section 6.3.3, we found that our BSP tree compression scheme could outperform TIFF for most images.

2.3.2 JPEG

JPEG is an acronym for “Joint Photographic Experts Group.” This international group has been working with the International Organization for Standardization (ISO), the International Telegraph and Telephone Consultative Committee (CCITT), and the International Electrotechnical Commission (IEC) to develop a standard for colour image compression [8]. As this standard became popular, the name of the group became attached to the standard itself.

The assumption behind JPEG compression is that images have smooth transitions (i.e. low frequency) in pixel intensities. Therefore, high frequency details can be removed without much sacrifice in image quality. This is usually true for photographic images. However, for computer graphics or line art images, JPEG compression often results in blurred edges.

JPEG uses a discrete cosine transform (DCT) to convert the image data from the spatial to the frequency domain. The image is divided into smaller blocks of 8×8 pixels. Then the values of each of the 64 pixels in this 8×8 block are scanned from the top-left corner towards the bottom-right in a zigzag pattern, as illustrated in Figure 2.2.

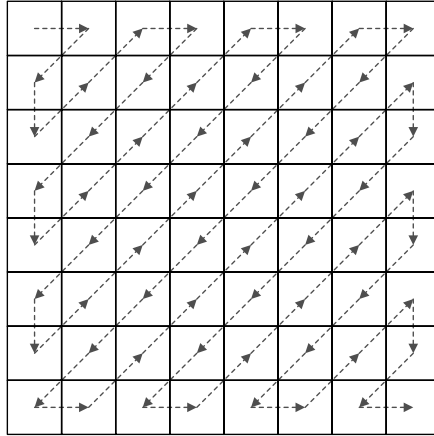


Figure 2.2 – JPEG zigzag pattern for 8×8 block

This process converts a two-dimensional 8×8 block into a one-dimensional vector with 64 values. Then, DCT is applied to the pixel values. After DCT, there will be 64 coefficients, with the first coefficient representing the DC component (i.e. average value of the block) and the rest of the AC coefficients representing the AC components of the block in increasing higher frequencies. The standard JPEG will output 64 coefficients, although one can specify JPEG to output fewer coefficients by choosing a smaller virtual block size (VBS). The DCT coefficients are then quantized (divided by a certain value) according to a quantization table. This quantization step has the most significant impact on the final output image quality. The compression ratio is controlled by the values in the quantization table. If the quantization level is increased, it will require fewer bits to represent the image but the image quality will be reduced, and vice versa. The results are then further compressed by Huffman coding. In Huffman coding, the more frequently occurring values are represented using fewer bits. In the case of JPEG, the AC coefficients are usually small; therefore, they are represented with fewer bits. The mapping between the actual value and its encoded value is done by looking up a Huffman encoding tree.

There have been several studies on the effect of varying JPEG parameters on the image quality, latency, and energy. In [19], the authors have studied the effect of varying quantization table on image quality and compression ratio. They have developed a metric for predicting the computation cost required to compress an image from estimating the number of DCT blocks and predicting the file size by analyzing the image's chrominance values. Their idea is to spend time in

compressing (i.e. transcoding) the image only if the benefits of compression outweigh the computation costs for compressing the image. [20], [21] have studied the effect of varying quantization levels and virtual block size on the impact of image quality. As smaller VBSs are chosen, the image quality and the energy used in DCT decreases. The communication energy is also decreased because of a smaller file size. However, at the same time, to keep the same image quality, one must decrease the quantization levels when the VBS is decreased, which will increase the file size and thus the communication energy. Therefore, the overall energy must be minimized by choosing the correct combination of VBS and quantization level. In their methodology, they proposed to use a table to store pre-computed values calculated for a number of images for each VBS and quantization level combinations. This table will be used at runtime to decide the best combination of VBS and quantization level that minimizes the overall energy and satisfy the image quality constraints. Although their approach does not require complete decompression to re-scale an image to different quality, it still requires re-quantization and re-encoding of entropy parameters, which consumes significant energy too. Our BSP tree compression scheme consumes only a small amount of energy when re-scaling because once the error associated with each node is computed, no re-computations are necessary when pruning the tree.

Another difference between JPEG and BSP tree scheme is that since JPEG works in the frequency domain, all changes in the compression parameters affect the entire image. Also, the entire image is stored as a single file, making JPEG a less ideal candidate for partial encryption.

2.3.3 Quadtree Compression

Quadtree compression [39], [40] is similar to BSP tree compression except in the way of how an image is partitioned. In quadtree compression, the image is split into four quadrants (instead of two regions) recursively, until the threshold criterion is reached. Similar to BSP trees, the quadtree image data consists of two parts: the tree structure and the colour leaf nodes. However, unlike BSP trees, the quadtree non-terminal nodes do not state partitioning lines explicitly. In a quadtree, the line coordinates are implicit because the region is always split in the middle to form new quadrants. Figure 2.3 shows how the same square image used in Figure 2.1 is split into quadrants to build a quadtree. Note that if the object boundaries do not align with the

quadtree partitioning lines, many more nodes are created (compared to the BSP tree in Figure 2.1), resulting in a less compact file size.

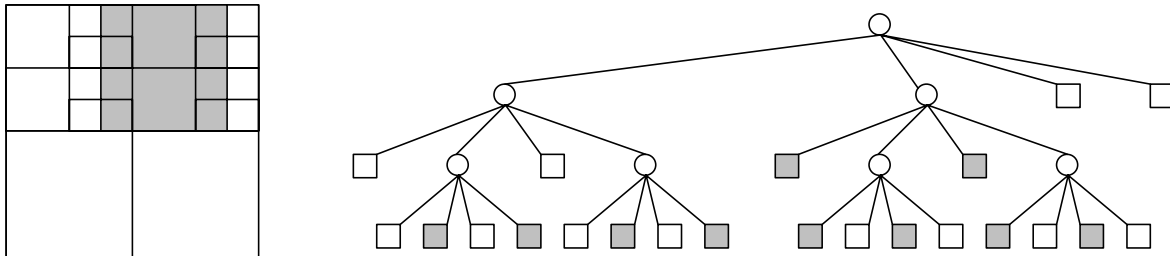


Figure 2.3 – Representation of Image in quadtree form

Similar to the BSP tree algorithm, the quadtree algorithm can operate in both lossless and lossy mode. In the lossless mode, the leaves contain the homogenous colour regions. In the lossy mode, the four quadrants are merged to form a new leaf if the error is smaller than an acceptable threshold.

2.3.4 Wavelet Compression

A recent standard, JPEG2000, uses wavelet transforms to compress an image. In wavelet compression [17], [37], the pixels of the image are first serialized into an image function. Then, this image function in spatial domain first goes through a discrete wavelet transform (DWT). A wavelet is a function that passes above and below the x-axis and the total area enclosed by this function is zero. The wavelet function is orthonormal to the image function. By passing the image function through a specially designed wavelet function, the image function will be decomposed into wavelet coefficients, similar to the case in DCT. The wavelet coefficients are divided into four subbands (i.e. LL, LH, HL, HH) in the 2D wavelet transform, with two subbands (lowpass L and highpass H subbands) in each of horizontal and vertical direction. The low-pass subbands represent the low-resolution version of the image, with the high-pass subbands adding the residual details of the original image. The wavelet transform can be performed again on the LL subband to refine the details in the image to form higher iterative levels (i.e. transform level TL). Similar to JPEG, the wavelet coefficients are then quantized and entropy encoded.

In a class of wavelet-based transforms called Set Partitioning in Hierarchical Tree (SPIHT) algorithms [23], the wavelet coefficients are organized in pyramid levels according to importance, with each level adding additional details to the image. Therefore, the wavelet coefficients form a tree. A zerotree is generated to indicate whether the coefficients in the tree are significant and needs to be decomposed further. The resulting image for the SPIHT algorithm consists of two parts: the zerotree structure and the wavelet coefficients component.

Several energy efficient wavelet image compression schemes have been proposed. In [24], [25], the authors have proposed varying parameters such as the transform level, elimination level (EL), and the quantization level (QL). The transform level controls the tradeoff in computation energy in DWT and image quality. The elimination level controls the computation costs and image quality by eliminating the computation of the highpass subbands during the DWT stage. QL affects the image quality in wavelet transform similarly as in the case for JPEG quantization levels. Again, the authors have used a lookup table for deciding the best combination of parameters that minimizes energy while satisfying user constraints. Similar to the JPEG schemes, such schemes requires re-computation of quantization and entropy coding steps, which consumes fair amount of energy too. In [26], a distributed approach to wavelet compression has been proposed to distribute energy use in a wireless adhoc network evenly at different nodes. In such a system, the task for the wavelet transform, which consumes the majority of computation energy in wavelet compression, is processed in a distributed fashion to the various nodes in the adhoc network.

2.3.5 Fractal Image Compression

Fractal compression techniques are based on the idea of *iterative contractive transformation* and *collage theorem* to exploit self-similarities in natural images [27], [29], [41]. Natural images often exhibit some self-similarities within the image itself. For example, the peak of a mountain may look similar to a zoom-in version of the entire mountain. In fractal image compression, the algorithm generates a set of building blocks of sub-images called *domain cells* by examining the entire image. Then, the entire image is partitioned into non-overlapping square cells called *range cells*. For each range cell, a domain cell is selected to approximate the range cell by applying appropriate transformations to the domain cell. Examples of such transformations includes, but

not limited to, scaling, rotating and shifting transformations. The resulting image is a tree structure similar to BSP trees. However, instead of storing the line parameters, each node is associated with the transformation parameters to generate the region the node represents. It is found that fractal image compression can perform well at high compression ratios compared to JPEG [28].

2.4 Advanced Encryption Standard

To send an image securely in wireless medium, it is necessary to encrypt the image. An encryption algorithm scrambles the data randomly according to a key so that no other parties can observe the data unless they know the key. At the time of writing, the most common symmetric key cryptographic algorithms are the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES). Since the National Institute of Standards and Technology (NIST) recommends AES to replace DES, we have chosen AES as the encryption algorithm for images.

The Advanced Encryption Standard algorithm was a project started to replace the DES algorithm [30]. The algorithm is an iterated block cipher which operates on 128, 192 or 256-bit data blocks and supports key lengths of 128, 192 and 256 bits. The key and block lengths can be specified independently. If the actual plaintext data length is not a multiple of the data block size, padding bytes must be added before encryption. The number of rounds of the algorithm depends on the key and the block length used. The AES algorithm begins by XOR-ing the input block with the 1st round key and the result of is called a “State” block. The “State” block is then separated into a two-dimensional array with each element being a byte. Figure 2.4 shows an example of an AES state block of 192 bits. Each element in the state block represents a byte.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

Figure 2.4 – AES “State” block

For each round, four operations are performed on “State”. The first operation is byte substitution where each byte of “State” is transformed using a S-box. The S-box is formed using finite field [30]. The second operation is row shifting where each row of State is cyclic shifted by a specific value. The third operation is **MixColumn()** where each column of state is operated on by a specific matrix operation. The last operation is **KeyAddition()** where the round key of the next round is XOR-ed with State. The process continues until the number of required rounds for the particular key and block length combination has elapsed, and the result is the cipher text.

The software implementation of this algorithm was obtained from the NIST AES website [30]. It is again separated into three layers where the outermost layer is the main program which calls the **encrypt** or **decrypt** function. In this case, it is **encryptblock()** and **decryptblock()**. These functions call the **rijndael_encrypt()** and **rijndael_decrypt()** function which performs the operation described above. Before the main function calls **encrypt** or **decrypt**, it calls the function **KeySchedule()** to form the keys for each round. Since AES does not change the data file size (other than the few padding bytes added to make the file size a multiple of the block size), encryption size is directly proportional to the input file size. This is consistent with our results in Section 6.4.1.

2.5 Partial Encryption

2.5.1 Overview

Cryptographic computations are often computationally intensive. Although AES is already designed with efficiency in mind, it can be seen that there are still quite a lot of byte shuffling and

XOR-ing repeated for multiple rounds. Therefore, to save energy, the encryption stage should minimize the number of cryptographic computations by encrypting only the absolutely necessary parts of the file. This is the concept behind *partial encryption*, in which only the important parts of an image are encrypted, without sacrificing security. In the case of BSP tree images, only the tree (and the line coordinates for additional security) needs to be encrypted. With only the unencrypted colour values, it is difficult to reconstruct the image. Different partial encryption schemes have been proposed for different image compression systems. These partial encryption schemes are examined here.

2.5.2 JPEG Partial Encryption

For JPEG, Tang has proposed the use of a random permutation of the coefficients [31]. His idea is to encrypt the DC coefficients and randomly scramble the AC coefficients according to a secret key. However, it has been shown in [32] that such a scheme is not secure. First, by statistic analysis, it is not difficult to obtain the AC coefficients' locations by trial and error. After the attacker has gained knowledge about the AC coefficients, it is possible for the attacker to reconstruct the image by the AC coefficients alone without the knowledge of the DC coefficients. This thesis demonstrates the insecurity of partially encrypting JPEG coefficients in Figure 2.5. In Figure 2.5, we have shown the resulting image after discarding the DC and the first few AC coefficients (by setting these coefficients to zero). An edge detection filter has been used to reconstruct the resulting image. It can be seen that even after encrypting half of the 64 coefficients, the edges of the original image is still clearly visible in Figure 2.5d.

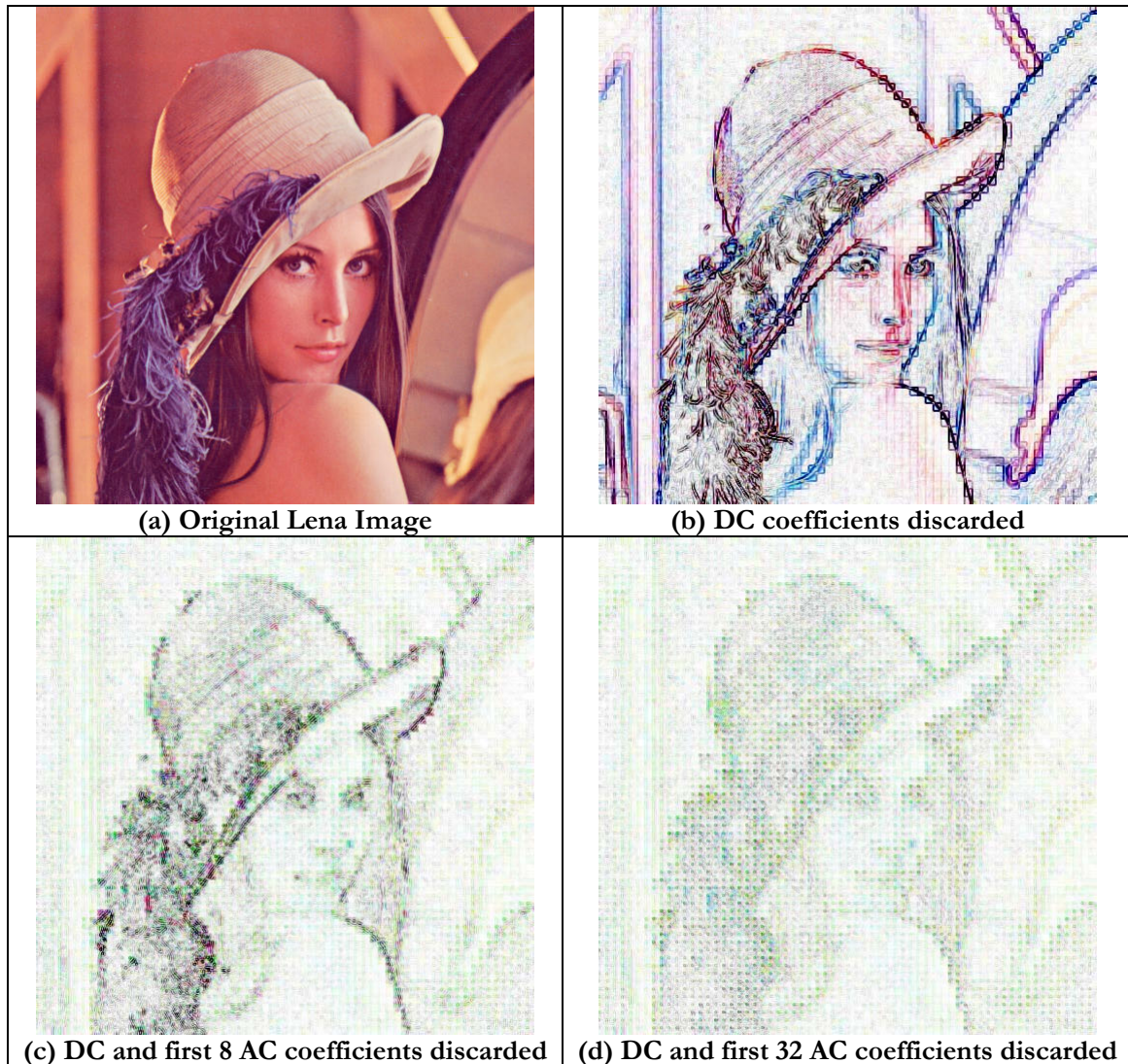


Figure 2.5 – Resulting JPEG image after discarding DC and AC coefficients

Since partially encrypting the coefficients does not work, a random permutation of the Huffman encoding tree has also been proposed in [34] and [35]. This scheme works by permuting the Huffman coding tree according to a secret key. Without knowing the secret key to synchronize the Huffman tree, the attacker would not be able to decode the Huffman-coded coefficients. However, randomly permuting the shape of the Huffman table makes compression less efficient because the Huffman tree would not be optimal for the data sequence [36]. A scalable partial encryption of both the inter-frame and intra-frame data for MPEG is proposed in [33]. Their system partially encrypts the first few coefficients of each DCT block too. Therefore, it suffers from the same problems as partially encrypting only the first few JPEG coefficients. In

fact, the authors only claim that this system is only intended to discourage video piracy. It is not a very strong secure system.

2.5.3 Quadtree Partial Encryption

In [39] and [40], the authors have proposed a partial encryption scheme for quadtrees. Similar to BSP tree partial encryption, their idea is to encrypt only the tree part of the quadtree. Their idea does not seem to suffer from major security problems. However, due to the nature of quadtree compression, their method may yield less compact image files. As we have discussed in Section 2.3.3, each split in quadtree generates four children nodes instead of two. In addition, the partition lines are inflexible because the splits must be in the middle. Therefore, many more nodes are generated compared to BSP tree representation of the same image. As an example, only seven nodes are generated for the image in Figure 2.1 using BSP tree representation; in contrast, twenty-nine nodes are generated in Figure 2.3 using quadtree representation for the same image. Because of these extra nodes, quadtree representation may not be as energy efficient as BSP trees with respect to transmission energy.

2.5.4 Wavelet Partial Encryption

In [39] and [40], the authors have also extended the partial encryption idea to SPIHT zerotrees. Their idea is to only encrypt the zerotree of the SPIHT zerotree compression. They claim that only the zerotree part of an image needs to be encrypted. Without the zerotree, it is difficult to reconstruct the image without knowing where the coefficients belong to.

In [37], the author proposed another partial encryption scheme for wavelets. Since the wavelet filter function is needed to decode the wavelet coefficients, they proposed that only the wavelet filter function needs to be encrypted for secure transmission. This scheme requires the generation of a unique mother wavelet each time secure image transmission is required, and the generation of new wavelets is a time consuming process. Therefore, this scheme is not energy efficient.

In [38], a different wavelet partial encryption scheme has been proposed. The encryption is performed between the quantization and the entropy encoder stages. The parts to be encrypted

can be selected using three schemes: subband selection, data bit selection, and random selection of pixel data. In their method, the subband to be encrypted is first randomly selected. Then, the first bit of every quantized wavelet coefficient (i.e. quantization indices) in the higher subband is encrypted. Finally, the pixel position to be encrypted is randomly selected using the serial output of a Linear Feedback Shift Register (LFSR). Since this scheme encrypts before the entropy encoder, similar to the JPEG Huffman tree permutation encryption scheme, the compression performance of the entropy encoder may be less efficient because it is a well-accepted fact in cryptography that compression after encryption is less efficient.

2.6 Summary

Many of the existing image compression algorithms are designed to aim for the maximum compression rate. However, these high compression algorithms may be too complex for energy constrained handheld devices. To be energy efficient, the compression algorithm should require as few computations as possible while still being flexible to user needs and the environment.

Secondly, although there are several partial encryption schemes for images, many of them are either insecure or inefficient. JPEG partial encryption schemes have major security problems. Quadtree partial encryption is similar to the BSP tree scheme. However, since the partition lines are fixed in a quadtree, it may be inefficient in partitioning the image. For example, the image region is always partitioned into four quadrants even if two of the quadrants have the same colour. By comparing Figure 2.1 and Figure 2.3, one can see that BSP tree representation generates less nodes than quadtrees for the same image. The wavelet compression scheme in [37] is also too computationally intensive because constructing a new mother wavelet is a time consuming process. The wavelet encryption scheme in [38] may result in less efficient compression because compression is done after encryption. None of the existing compression and partial encryption schemes yields satisfactory performance when efficient energy utilization and convenient re-scaling without original image is the main goal.

The binary tree partitioning image compression scheme is chosen as the candidate for this research because it has the following advantages that other image compression techniques do not have.

1. **Data organization allows partial encryption** – Compared to transform-based compression algorithms such as JPEG, BSP tree algorithm separates the image structure, i.e. the tree, from the other less important data such as pixel colours. This allows *partial encryption* because it is difficult for the attacker to reconstruct the image by the line and pixel colour values alone. This saves encryption computation energy.
2. **Partitioning is efficient** – Unlike the quadtree technique, where the image split is implicit in the tree structure, BSP tree partitioning lines can align with actual image boundaries. This can result in a more efficient splitting of the image and a more compact file size, hence saving transmission energy.
3. **Tree structure allows convenient pruning and scaling to different image quality without original image** – The hierarchical structure of the BSP tree allows convenient pruning of an image. If a user wants a smaller file size, he/she can set a larger pruning threshold. Larger threshold allows merging of the nodes at the bottom of the tree, which reduces the size of the image (with some tradeoffs in image quality). Since the error are stored at nodes once the BSP tree is constructed, the BSP tree can be easily pruned to any arbitrary image quality with the original best quality image. This offers scalability and flexibility by dynamically adjusting the image quality according to the user need and environment to save transmission energy.
4. **Allows variable thresholds in different focus areas** – An image often has areas that are more important than other parts. The pruning threshold can be set to be lower in these important areas to allow a sharper image in these areas. The user has the option of selectively focusing on important areas, while using a higher threshold in other less important areas. This is a capability that wavelet and JPEG algorithms do not have. These algorithms transform the entire image from the spatial domain to another domain. Setting a higher compression ratio for these algorithms reduces quality for the *entire* image. Since the BSP tree technique operates in the spatial domain, it can allow different compression thresholds in different areas. This gives flexibility to the user by allowing the user to retain high

image quality in only the important areas of the image while compressing the less important areas for efficient transmission.

Chapter 3 – Methodology

This chapter presents the proposed scalable energy efficient image compression and partial encryption scheme. We present the specifics of how the BSP tree representation of the highest quality image is constructed, and the specifics of how the security, energy, and image quality parameters are used in different modules of the system.

3.1 System Overview

Figure 3.1 shows the block diagram for the proposed partial image compression and encryption system. The end user specifies his desired image quality and security levels to the system. From these user parameters, the test bench module generates the threshold for the compression and the encryption key length and percentage of nodes encrypted. The compression module then compresses the highest quality image and the resulting image is immediately encrypted to generate an encrypted image.

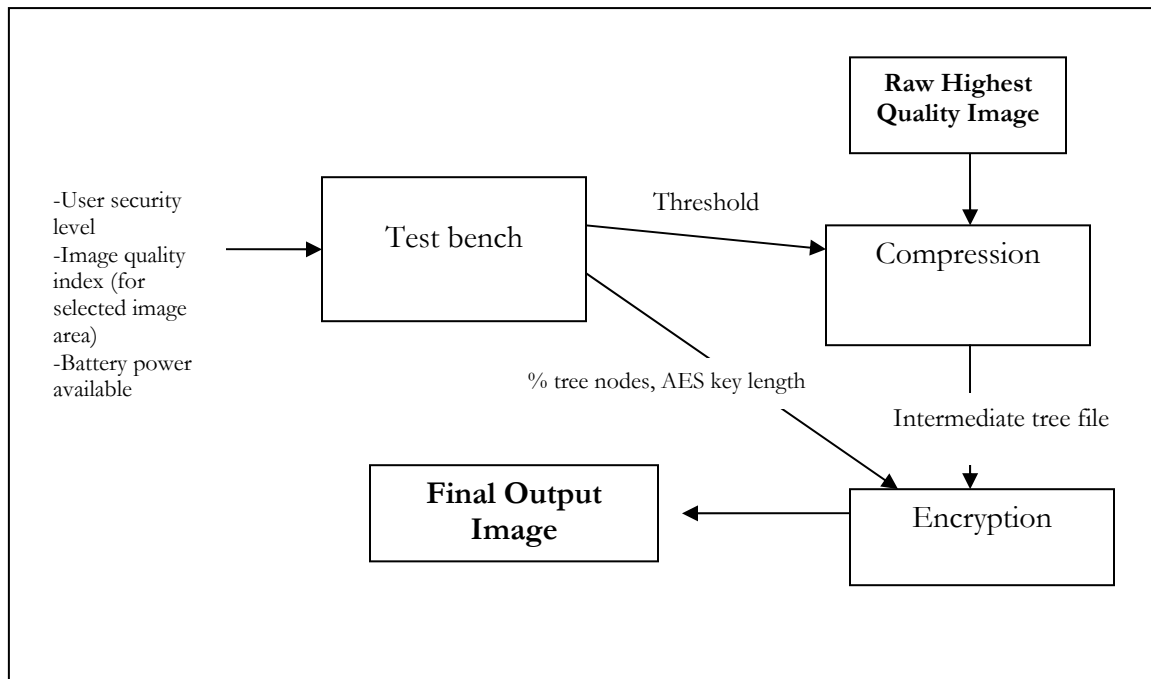


Figure 3.1 – Proposed Image Compression and Encryption System

3.2 Generation of the Highest Quality Image Tree

This step takes a Window Bitmap file as the input and generates the corresponding BSP tree representation of the image. To generate the highest quality image in tree form, the image needs to be partitioned into regions using partition lines. The bitmap image is read and each colour channel (red, green, blue) is stored as an array in the memory. The luminance (Y) for each pixel is calculated from the RGB colour and stored in another array. These luminance values are used for mean and error computation used for generating the partitioning lines (for the optimal line selection method) and/or pruning in the compression module later. The output tree image from this process is referred to as the “highest quality image” (i.e. no colour information is lost) in the rest of this thesis. Once the highest quality image is converted to the BSP tree format, the image can be compressed to different image quality without the need for the original bitmap image.

3.2.1 Binary Split Tree Construction Method

Our experimental results Section 5.1 shows that the original optimal line selection method in [15] is slow because it requires a lot of computations in the LSE transform and error calculation parts. Therefore, we have proposed a new tree construction method in this thesis. In this newly proposed binary split method, the algorithm simply bisects the longest dimension of the region of interest in two equal halves. This speeds up the line selection process significantly, since it eliminates the expensive LSE transform computation for determining the set of candidate lines. However, the trade-off is that the file size may not be optimal. In this method, for a region R with endpoints p_1 and p_2 and x_1 and x_2 as the x-coordinates (y-coordinates if it is a vertical line) of the two points, the midpoint x_c of the region is simply selected using Eq. (2.3). The image is partitioned into smaller rectangular regions until the colour of the region is homogenous or until the region size is equal to a pixel, in which case no further division is possible.

Although this splitting method violates the property of optimal alignment of BSP tree partitioning lines with the actual object edges, it is found in our experiments that, in practice, this new method still performs satisfactorily.

3.3 Testbench Module

The testbench module gives scalability and user flexibility to generate the compression and encryption parameters according to the user specified image quality, security requirements, and battery power. It receives the security level (0-5), image quality index (user-specified threshold values) from the user, and the battery status from the hardware. From these parameters, the test bench module generates the tree termination criteria threshold values for the compression module, the percentage of tree and partition lines encrypted, and the AES key length. Table 3.1 shows the parameter settings for each user security level.

Table 3.1– Security Level Parameters

Security Level	Tree Encrypted (%)	Line Encrypted (%)	Key Length
0	0	0	Not Encrypted
1	60	0	128
2	80	0	128
3	100	0	128
4	100	50	256
5	100	100	256

Image quality is controlled by the pruning threshold. This pruning threshold is specified as a divisor of the image tree’s root node’s total error. The image quality is predicted using the proposed quality metric in this thesis.

3.4 Compression

The compression module reduces the size of the image file. Some of the advantages of a smaller file size are:

1. It takes less energy to transmit a smaller size image.
2. It takes less time to encrypt the image, thus reducing the energy used for cryptographic computations.
3. It uses less storage space of memory constrained handheld devices.

The compression module accepts the pruning threshold from the testbench module. It also reads the highest quality image file, which can be downloaded from a server or generated locally. This best quality raw image is already in the tree form. The compression module compresses the raw image by several means:

1. It does a lossy compression by pruning the raw image tree according to the *pruning threshold*, which is expressed as *a fraction relative to the root node's error* in our experiments. This process reduces the file size by sacrificing the image quality. The process is lossy because it is non-reversible.
2. It can employ a colour table method to store the colours more efficiently if certain conditions are satisfied.
3. It can use a non-standard bit size to store the integers for the line coordinates and/or the colour indexes of a colour table.

There exist some advanced pruning techniques such as [44]. For simplicity, we focus our work using a simple error based pruning criterion.

3.4.1 Error Formula for Pruning

Before compression, the error associated with each node in the tree must be calculated. Informally, error is defined as *the amount of deviation from the actual pixel colour* if the pixel colour is approximated using the mean colour value for the region. Error is used for determining whether to *prune* the subtrees from each node.

We now introduce the concept of *pruning*. Pruning deletes and merges nodes in a BSP tree if the error is smaller than a user specified threshold. In the best quality image, the leaf nodes store the exact colours of the original image. When pruning this best quality tree, these leaf nodes are merged to form new leaf nodes if the merged node's error is below the *pruning threshold*, which is expressed as *a fraction of the root node's error* in this thesis. After these leaf nodes are merged, the mean colour value of the pixels in the region is used to approximate the new leaf node's colour. First, users will specify a relative threshold divisor value. Then, this value is multiplied by the root node's error value to generate the absolute threshold value that is used for pruning decision. Note

that we specify the threshold value relative to the root node's error instead of using an absolute threshold value. In our experiments, it is found that relative threshold is more convenient to handle for our image quality and file size prediction model described in Section 6.6 and 6.7.

Traditionally, error for the region is calculated using the total square error in Eq. (2.4). However, in our experiment, it is found that this formula is slow for two reasons. First, the square and sum operations in Eq. (2.4) are slow and tedious. Second, this total square error does not reuse previously calculated values. Error must be recalculated from scratch at each iteration. Therefore, we have proposed a new error calculation formula for image pruning. Our formula calculates error in a bottom-up manner and speeds up computation by reusing previously calculated error values. By reusing the results from the previous computations in the subtrees from the current node, this formula reduces the error calculation time significantly. First, at each iteration i , two parameters, σ_1 and σ_2 , are defined:

$$\begin{aligned} \sigma_1 &= \begin{cases} |m_L - m_i| & \text{if left child of the current node is a leaf} \\ \sigma_L & \text{if otherwise} \end{cases} \\ \sigma_2 &= \begin{cases} |m_R - m_i| & \text{if right child of the current node is a leaf} \\ \sigma_R & \text{if otherwise} \end{cases} \end{aligned} \tag{3.1}$$

where:

m_L and m_R are the mean values of the left and right child, respectively.

σ_L and σ_R are the previously calculated values of σ for the left and right child, respectively.

m_i is the mean value of the current node at iteration i .

The current node's standard deviation σ_i is calculated by:

$$\sigma_i = \begin{cases} 0 & \text{if the current node is a leaf} \\ \sqrt{\frac{A_L \sigma_1^2 + A_R \sigma_2^2}{A_L + A_R}} & \text{otherwise} \end{cases}$$

(3.2)

where A_L and A_R are the area of the left and right child, respectively.

Finally, the node's total error E_i is calculated by this formula:

$$E_i = \begin{cases} A_L |m_L + \sigma_1 - m_i| + A_R |m_i - (m_R - \sigma_2)| & \text{if } m_L \geq m_i \\ A_R |m_R + \sigma_2 - m_i| + A_L |m_i - (m_L - \sigma_1)| & \text{otherwise} \end{cases} \quad (3.3)$$

3.4.2 Error Representation at Nodes

The error can be represented at each node in two ways:

1. **Total Error** – Each node stores the total error directly. In other words, the computed value from Eq. (2.4) or (3.3) is stored directly at the node. The total error is affected by the amount of deviation of the mean value from the actual pixel colour as well as the area of the region.
2. **Average Error** – In this representation, the total error of the region is divided by the area of the region. Mathematically, the average error is:

$$\frac{1}{MN} E_i \quad (3.4)$$

where E_i is defined in Eq. (2.4) for the total square error formula or (3.3) for our proposed fast error formula. The average error is independent of the area of the region.

After calculating the error associated with each node, the image tree is pruned. Pruning the tree deletes nodes of the tree and merges the nodes by approximating the region colour with the mean value of the region's pixels.

3.4.3 Compression by Use of Colour Table

This is a new lossless compression technique proposed in this thesis. Often, the image can be compressed more with a colour table without any loss in quality if certain conditions are satisfied. Figure 3.2 illustrates how this method works. A colour table is first created to store all the colours used in the image. For our example in Figure 3.2, there are three colours in total: C_1 , C_2 and C_3 . Then, instead of storing the actual colour values, the index values (i.e. 01, 02 and 03 in our example) to the colour table are stored at the leaf nodes.

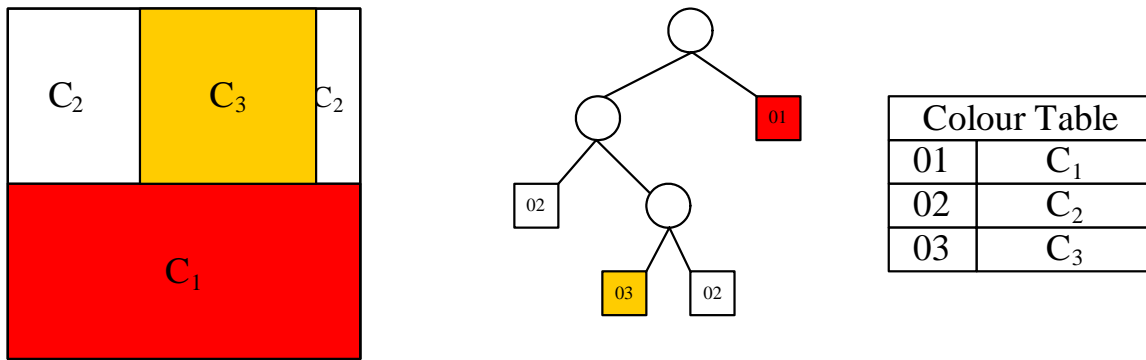


Figure 3.2 – Compression Using Colour Table

The assumption in colour table compression method is that the index values will require fewer bits to encode than the actual 24-bit colour values. However, the extra overhead of storing the colour table may offset the gain in memory savings from the leaf nodes. The conditions where the image will benefit from using a colour table are discussed in Section 6.1.4.

3.4.4 Using Custom Bit Length for Storing Integers

There are various integers used in the BSP tree. Examples of these integers include the line coordinates and the colour table indices (if the colour table is used). The programming language C uses standard bit sizes such as 8-bit, 16-bit and 32-bit for integers. However, the integers used in the BSP tree often do not make use of the full range of values available from these standard sizes. Therefore, one can possibly reduce the number of bits used for storing these integers by using only the minimum number of bits to represent the range of values that will ever be used. For example, for a 512×512 image, the maximum line coordinate value can only be 511.

Therefore, only 9 bits (instead of the standard 16-bit) is needed to represent all the possible values of the line coordinates. Eq. (3.5) shows the formula for finding the minimum number of bits required to represent a set of items $\{d_1, d_2, \dots, d_n\}$:

$$numBits = \lceil \log_2(\max\{d_1, d_2, \dots, d_n\}) \rceil \quad (3.5)$$

The trade-off for using a custom bit length is that some bit manipulation operations needs to be done because instructions in C can only access data in multiples of 8 bits.

3.5 Scalable Encryption Module

After compression, the resulting pruned image will be encrypted by the encryption module. The encryption module encrypts the intermediate image tree and/or partition line file from compression module according to the test bench generated parameters. It accepts parameters from the test bench such as the percentage of tree file to be encrypted, percentage of partition lines to be encrypted and the AES key length. It also accepts the intermediate image file generated by the compression module. Finally, it outputs the encrypted image file to be transmitted.

The fraction of file encrypted is first converted to the number of AES blocks encrypted. To calculate the total number AES blocks, divide the total number of bytes of the tree or partition line file by the AES block size (which can be 16, 24 or 32 bytes). Since AES requires the file size to be a multiple of the AES block size, padding bytes are added if that is not the case. The number of actual blocks that should be encrypted is computed by multiplying the total number of AES blocks by the percentages from the testbench. Then, AES is applied directly to each section with the chosen AES key length until the specified number of blocks has been encrypted. If there are remaining data bytes, they are appended after the encrypted section.

3.6 Final Output Image

The final output image that is ready for transmission is converted to a binary file, which consists of the following:

1. **Unencrypted header** – The header contains the necessary information for the receiver to the rest of the image file. It contains values such as the requested security level and the compression mode.
2. **Partially Encrypted Tree** – The encrypted tree contains the encrypted tree in the first part, and the unencrypted tree part if the entire tree is not encrypted.
3. **Partially Encrypted Partition Line** – This part contains the coordinates of the partitioning lines stored at the non-terminal nodes of the tree. This part can be unencrypted (for lower security levels), partially encrypted (for medium security levels), or fully encrypted (for higher security levels).
4. **Unencrypted Partition Line Orientations** – This part contains the orientations of the partitioning lines of the non-terminal nodes. The lines can be either horizontal or vertical and is represented using one bit per node.
5. **Unencrypted Colour** – This part contains the leaf node’s actual colour or the colour index if colour compression is used.
6. **Unencrypted Colour Table (optional)** – This part contains the colour table if colour table compression is used.

This output image is then serialized into bitstream before it can be encrypted. It is serialized in a pre-order depth first traversal fashion as illustrated in Figure 3.3. The traversal starts at the root node. The current node is visited first, then the right subtree is visited. After all the nodes in the right subtree are visited, then the left subtree is visited. The tree structure is stored in the tree file, with 1 representing a non-terminal node, and 0 representing a leaf. If the current node is a non-terminal node, the node’s line parameters L_1, L_2, \dots, L_N are stored in the line coordinate file and orientation file. The leaf colours C_1, C_2, \dots, C_N are stored in a separate colour file. The header file and the colour table file (if used) are generated to record the image parameters.

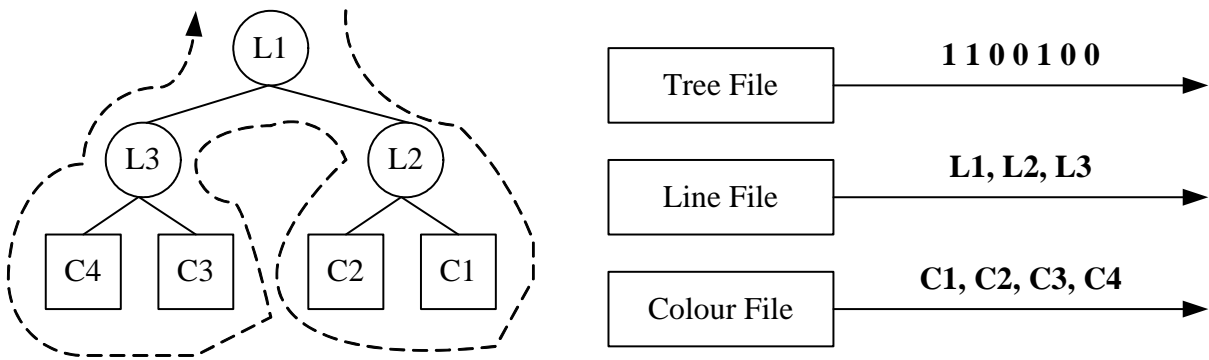


Figure 3.3 – BSP tree traversal order for serializing to file

Chapter 4 – Experimental Setup and Metrics

This chapter describes the platform and image files used for the experiments. It also describes the metrics used in the experiments to measure performance and image quality.

4.1 Platform

All source code used for this thesis is written in C and C++ in Microsoft Visual C++ 6.0. The program is compiled using the default settings for “Optimize for Speed” in Release mode. The program is then run on the target system. The target system is a Pentium III 700MHz system with 192MB RAM running on Microsoft Windows XP.

4.2 Test Image Files

The test images are obtained from [45] and shown in Figure 4.1. We have selected three photographic images: a 512×512 peppers image (the image used in [7], [29], [39]), a smaller 256×256 lena image (used in [14], [29], [40], [41], [13], [42], [43]), a larger 768×512 tulips image. In addition, we have selected two computer graphics images: a 512×512 frymire image and 512×512 serrano image. This set of test images allows us to observe how image size and types affect our system’s performance.

The image files are originally stored in TIFF format at the website and they are converted to Window 24-bit Bitmap format using the Windows Paint application before processing.

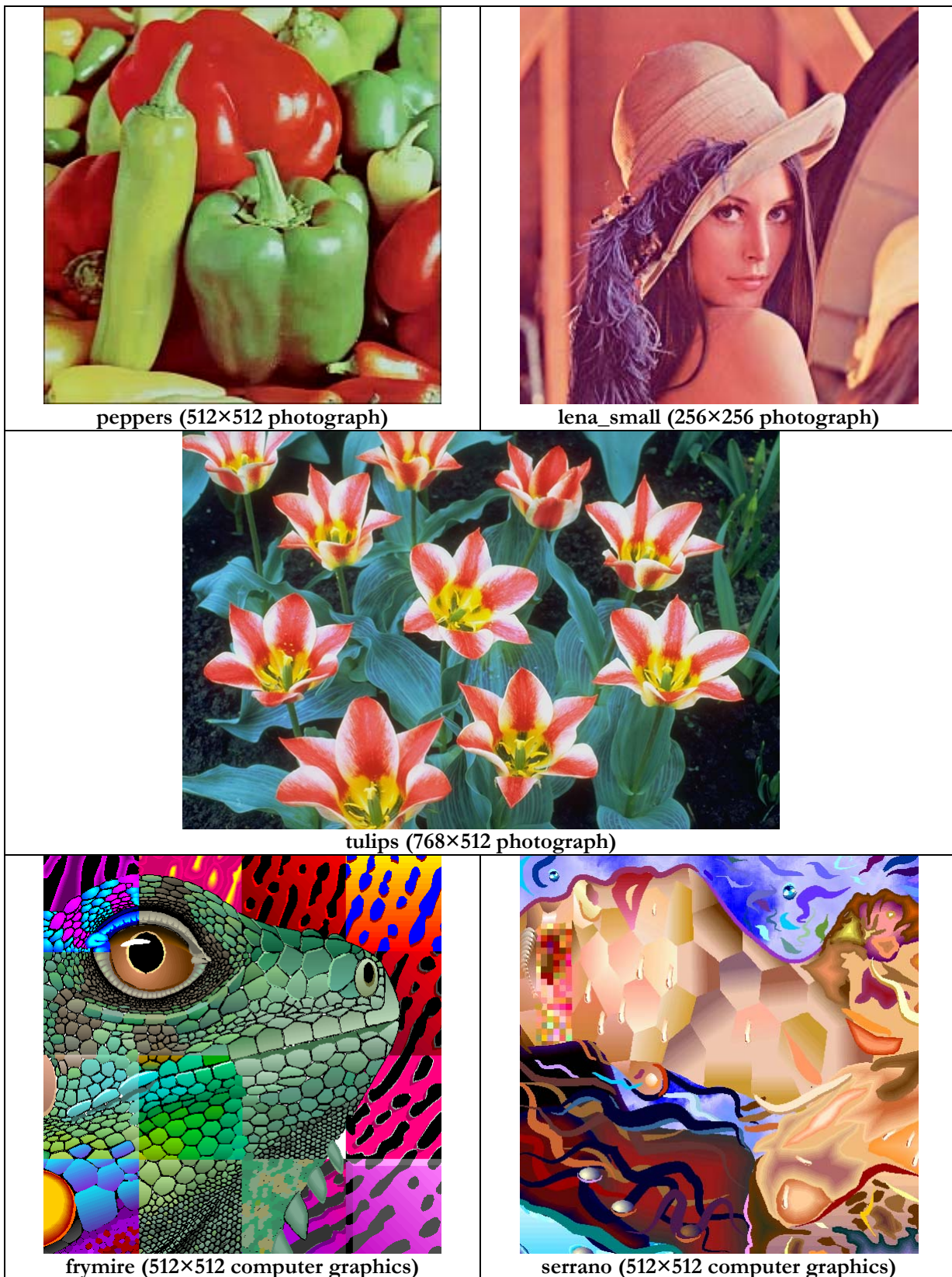


Figure 4.1 – Test Images Used for this Thesis

4.3 Measuring Performance

4.3.1 Execution Time

The execution time is measured by the C language `clock()` function. Just before the start of the function whose performance is of interest, the start time is recorded. And immediately at the termination of the function, the `clock()` function is called again to record the end time. The difference between the end time and the start time is the execution time of the function. Since the lowest resolution of the `clock()` function is one millisecond, if the function terminates too fast, the function is put inside a “for” loop and the average execution time per iteration is computed.

4.3.2 Compression

Compression bit rate is a common metric to measure and compare compression performance. It is defined as the total number of bits used to store the entire image divided by the total number of pixels in the image. For a $M \times N$ image, the bit rate r is defined as:

$$r \equiv 10 \log_{10} \frac{8 \cdot \text{FileSizeInBytes}}{MN} \quad (4.1)$$

4.3.3 Image Quality

A common way to quantitatively measure image quality is by measuring the Peak Signal to Noise Ratio (PSNR). The formula for calculating the PSNR for a greyscale image [14] is:

$$PSNR \equiv 10 \log_{10} \frac{V^2}{MSE} \quad (4.2)$$

where V is the peak-to-peak value of the pixel intensities within the test image and MSE is the mean square error for the entire region. Since in our work the pixel intensities is a weighted

average value of the three colour channels according to Eq. (2.5), $V=255$ in our experiment. MSE is defined as the total square error (Eq. (2.4)) divided by the region area [14]:

$$\begin{aligned}
 MSE &\equiv \frac{1}{MN} \cdot TotalError \\
 &\equiv \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [I(x, y) - \hat{I}(x, y)]^2
 \end{aligned}
 \tag{4.3}$$

For a colour image, since there are three channels RGB, we have extended Eq. (4.3) such that the total square error from each channel is added and the average square error is taken:

$$MSE_{colour} = \frac{1}{3MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left([I_R(x, y) - \hat{I}_R(x, y)]^2 + [I_G(x, y) - \hat{I}_G(x, y)]^2 + [I_B(x, y) - \hat{I}_B(x, y)]^2 \right)
 \tag{4.4}$$

where I_R, I_G, I_B and $\hat{I}_R, \hat{I}_G, \hat{I}_B$ and the actual and approximated value of the pixel colour for the red, green and blue channels respectively.

A small utility program was written to compute the PSNR ratio between two bitmap images. The resulting output images in tree format are converted to bitmap format. Then, these output images are compared with the original bitmap for computing the PSNR ratios.

Chapter 5 – Experimental Results

This chapter shows the experiment results of varying different parameters in generating an image. We will investigate the effect of the tree construction mechanism on the execution time and compression bit rate. We will also investigate how the error formula used for pruning affects the image quality and how using custom bit length for integers helps to improve the compression ratios.

5.1 Effect of Tree Construction Mechanism

The tree construction method affects how fast the best quality tree representation of image can be generated. In addition, the shape of the tree affects the pruning characteristics. This section presents the resulting images for the optimal line selection method and the binary split method.

In optimal line selection method, at every line split, the line with the minimum total error is selected. The file size is optimal for the best quality image. However, this method is computationally intensive.

Since the optimal line selection method is slow, we have proposed a new tree construction method, as discussed in Section 3.2.1. In this new method, the region is simply bisected into two halves without considering the error. Table 5.1 shows the tree construction time and the compression bit rates for both methods.

Table 5.1 – Tree Construction Time for Various Images

	Time (ms)			Bit Rate (bpp)		
	Optimal Line	Binary Split	Ratio of Binary Split to Optimal Split Time	Optimal Line	Binary Split	% File Size Increase
peppers	14120	1552	11%	31.94	34.06	6.6%
lena_small	3264	360	11%	32.40	33.39	3.1%
tulips	20800	2323	11%	32.37	34.41	6.3%
frymire	10535	951	9%	8.89	20.26	128%
serrano	11506	791	6.8%	6.86	14.24	107.5%

It can be seen that the binary split method is significantly faster than the optimal line selection method. It only uses a fraction of the optimal line execution time (about 7-11%) required for constructing the tree. The image quality for the best quality image is indistinguishable between the two. The tradeoff is a slight increase in file size. The exception is the frymire image and serrano image. For these computer graphics images, the file size doubles because binary splitting method splits regions that are otherwise merged using optimal line selection. This results in creating more nodes. However, the high speed in tree construction for the fast method may justify the increase in file size.

5.2 Effect of Error Calculation Formula on Pruning

5.2.1 Overview

Although the compression bit rates are similar in most cases for the two tree construction methods, they exhibit a larger difference when we try to prune the tree to compress the image. The tree generated by the optimal line selection method is unbalanced. There are some paths that go very deep. In contrast, the binary split method simply bisects the region at the dimension that is longest. Therefore, the tree generated by the binary split method is relatively balanced. In this section, we will explore the effects of the tree construction method on the results generated by different pruning formula.

As discussed in Section 3.4.1, the error needs to be evaluated at every pruning decision. Therefore, the error calculation formula has a major impact on the pruning execution time. This experiment explores how the error formula affects image quality, file size, and execution time.

5.2.2 Effect on Image Quality and File Size

5.2.2.1 Total Square Error Formula

In this experiment, the total square error formula in (2.4) is used to prune the peppers and frymire images. For both images, we apply the total error formula to prune the two versions of the “best quality” images, where one version is generated using the optimal line selection method and the other generated using the binary split method.

The resulting peppers and frymire images at different thresholds are shown in Figure 5.1 and Figure 5.2, respectively. The subtle differences between the images are highlighted with circles when necessary. The file sizes for the peppers and frymire image are shown in Table 5.2 and Table 5.3, respectively. The file settings for both experiments are compressed line coordinate file mode, uncompressed colour table mode, and total error threshold mode.

It can be seen that there are no major differences for the peppers image generated using either tree construction methods. However, for the frymire image, the optimal line tree version has a better image quality to compression ratio.







Threshold	Optimal Line Tree	Binary Split Tree
1.00×10 ⁻⁵		
8.00×10 ⁻⁵		
4.00×10 ⁻⁴		

Figure 5.1 – Peppers Image at different thresholds Using Total Square Error Pruning




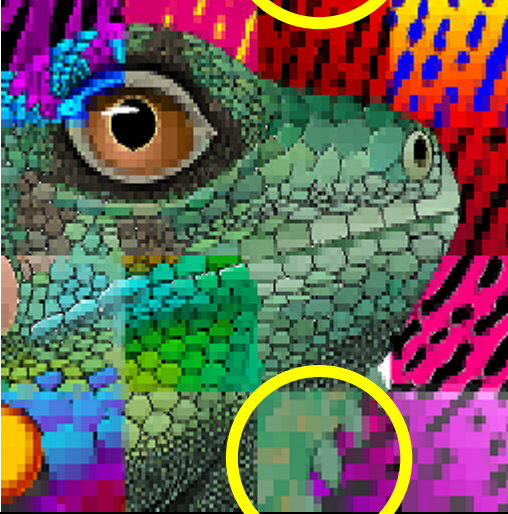


Threshold	Optimal Line Tree	Binary Split Tree
1.00×10 ⁻⁵		
8.00×10 ⁻⁵		
4.00×10 ⁻⁴		

Figure 5.2 – Frymire Image at different thresholds Using Total Square Error Pruning

Table 5.2 – Peppers Image Bit Rates Using Total Square Error Pruning

Threshold	Optimal Line		Binary Split	
	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)
0	31.94	54.46	34.06	56.53
5.00×10^{-6}	1.07	29.68	1.86	29.55
1.00×10^{-5}	0.78	28.36	1.40	27.85
2.00×10^{-5}	0.57	27.03	1.03	26.13
4.00×10^{-5}	0.42	25.69	0.72	24.33
8.00×10^{-5}	0.31	24.46	0.45	22.72
1.00×10^{-4}	0.28	24.06	0.39	22.39
2.00×10^{-4}	0.21	22.87	0.24	21.37
4.00×10^{-4}	0.16	21.80	0.14	20.41

Table 5.3 – Frymire Image Bit Rates Using Total Square Error Pruning

Threshold	Optimal Line		Binary Split	
	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)
0	8.89	49.66	20.26	52.34
5.00×10^{-6}	4.45	26.02	7.58	24.15
1.00×10^{-5}	3.71	23.08	5.96	20.88
2.00×10^{-5}	2.72	20.24	3.86	18.08
4.00×10^{-5}	1.87	18.15	2.37	16.42
8.00×10^{-5}	1.21	16.57	1.37	15.22
1.00×10^{-4}	1.04	16.16	1.13	14.91
2.00×10^{-4}	0.64	15.09	0.62	14.08
4.00×10^{-4}	0.37	14.08	0.34	13.35

5.2.2.2 Fast Error Formula

In this experiment, the proposed fast error formula in (3.3) is used to prune the images at different thresholds. Similar to the previous experiment, the fast error formula is used to prune a optimal line tree and a binary split tree.

The resulting peppers and frymire images at different thresholds are shown in Figure 5.3 and Figure 5.4, respectively. The file sizes for the peppers and frymire image are shown in Table 5.4 and Table 5.5, respectively.







Threshold	Optimal Line Tree	Binary Split Tree
1.00×10 ⁻⁵		
8.00×10 ⁻⁵		
4.00×10 ⁻⁴		

Figure 5.3 – Peppers Image at different thresholds Using Fast Error Pruning







Threshold	Optimal Line Tree	Binary Split Tree
1.00×10 ⁻⁵		
8.00×10 ⁻⁵		
4.00×10 ⁻⁴		

Figure 5.4 – Frymire Image at different thresholds Using Fast Error Pruning

Table 5.4 – Peppers Image Bit Rates Using Fast Error Pruning

Threshold	Optimal Line		Binary Split	
	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)
0	31.94	54.46	34.06	56.53
5.00×10⁻⁰⁶	3.49	34.54	8.82	37.73
1.00×10⁻⁰⁵	2.43	32.94	6.16	35.70
2.00×10⁻⁰⁵	1.67	31.36	4.38	33.76
4.00×10⁻⁰⁵	1.12	29.67	3.13	31.89
8.00×10⁻⁰⁵	0.76	27.91	2.23	29.93
1.00×10⁻⁰⁴	0.66	27.44	2.00	29.17
2.00×10⁻⁰⁴	0.42	24.91	1.45	27.25
4.00×10⁻⁰⁴	0.26	23.10	1.04	25.30

Table 5.5 – Frymire Image Bit Rates Using Fast Error Pruning

Threshold	Optimal Line		Binary Split	
	Bit Rate (bpp)	PSNR (dB)	Bit Rate (bpp)	PSNR (dB)
0	8.89	49.66	20.26	52.34
5.00×10⁻⁰⁶	5.32	31.51	9.42	30.94
1.00×10⁻⁰⁵	5.02	29.23	8.94	29.07
2.00×10⁻⁰⁵	4.69	27.38	8.36	26.95
4.00×10⁻⁰⁵	4.17	24.82	7.40	24.25
8.00×10⁻⁰⁵	3.17	21.59	5.44	20.56
1.00×10⁻⁰⁴	2.76	20.39	4.58	19.41
2.00×10⁻⁰⁴	1.59	17.55	2.44	16.77
4.00×10⁻⁰⁴	0.81	15.62	1.13	15.08

The file settings for both experiments are compressed line coordinate file mode, uncompressed colour table mode, and total error threshold mode.

With respect to image quality, it is found that the fast error formula works slightly better with binary split generated trees. The optimal line generated image’s quality degrades quickly as the threshold increases. For example, at a threshold of 5×10^{-6} of the root node’s error, the bit rate decreases quickly to 3.49bpp at 34.54dB, while the binary split tree has a slower drop in bit rate (8.82bpp) image quality (37.73dB). The sharp drop in bit rate and image quality for optimal line trees is due to the observation that these trees are usually unbalanced. Therefore, the effect of error propagation is less than an unbalanced tree where some paths to the root may be very long. As a result, the root’s total error deviates from the actual total square error a lot. The total square formula works better with optimal line generated trees because these trees are highly unbalanced.

5.2.3 Effect on Execution Time

The total execution time spent in computing the errors at each node is shown in Table 5.6.

Table 5.6 – Execution Time (ms) for Computing Node Error

	Optimal Line Tree		Binary Split Tree	
	Total Square Error	Our Fast Error Formula	Total Square Error	Our Fast Error Formula
peppers	5939	601	4487	631
frymire	1282	170	2604	391

It can be seen that the fast error formula reduces the error computation time significantly. For both the peppers and frymire images, the execution time is reduced by almost 90%. Also, the slow error method works slightly better on binary split tree version than the optimal line tree version of peppers. Again, this is due to the unbalanced tree shape of optimal line trees. Therefore, a lot of the calculations done at the deeper levels need to be performed again. For the frymire image, the behaviour is opposite. The binary split tree version is slower. This is because for the frymire image, the best quality image actually has a lot more nodes than the optimal line version.

5.2.4 Average Error versus Total Error Representation

How the error is represented also has an impact on the pruning results. In Section 3.4.2, we have discussed that the error at each node can be specified as either the average error per unit area or the total error for the region. This section demonstrates the resulting images using the two representations.

Figure 5.5 shows the pruning results using average error representation with the fast error formula for a binary split generated peppers image.

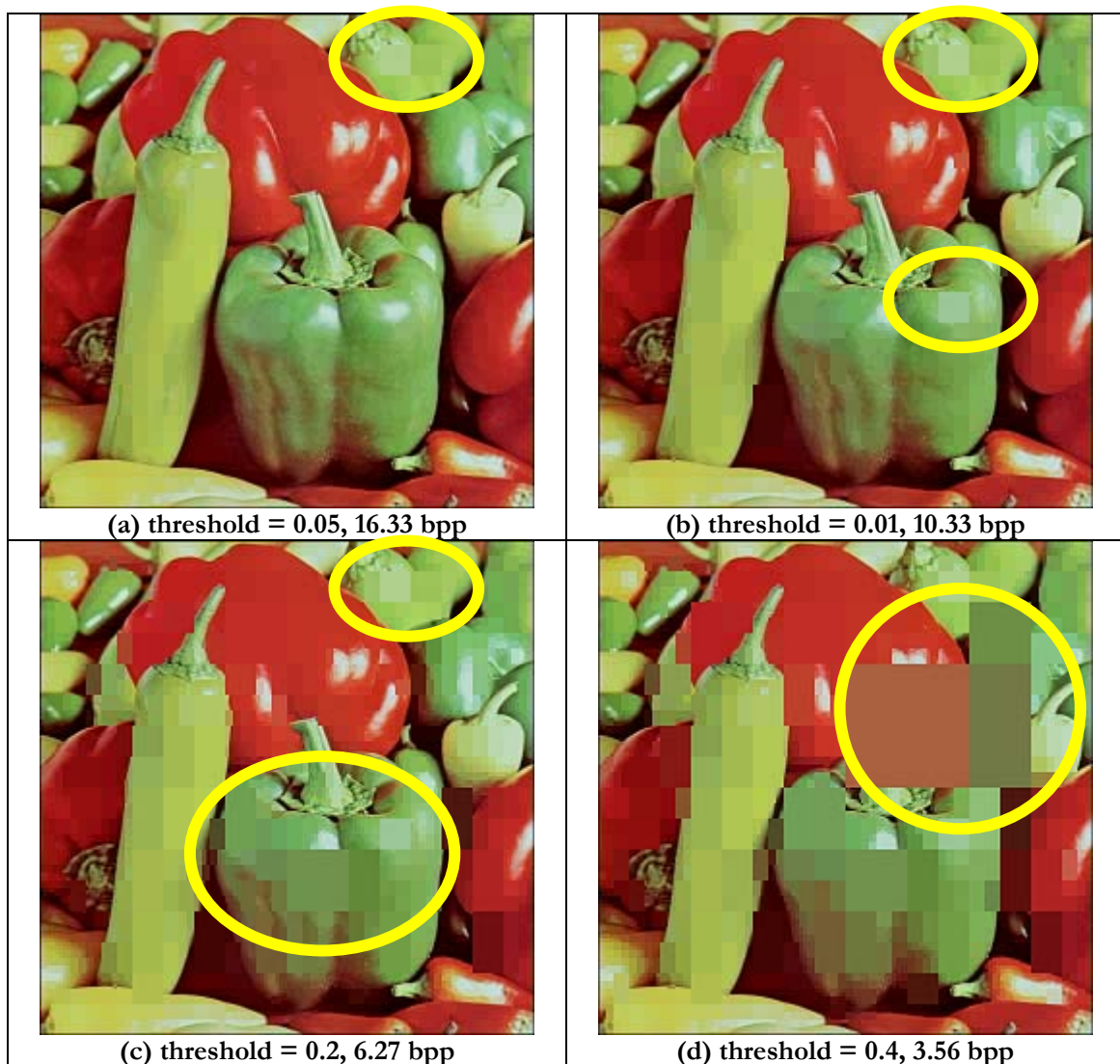


Figure 5.5 – Peppers Image at different thresholds Using Average Fast Error Pruning

It can be seen that the tree sometimes prunes at undesirable large regions. Therefore, we see large patches of “blocks” (highlighted with circles) for the images with a larger threshold. This is undesirable because large areas affect the overall image quality more than smaller regions. Even at a low threshold setting of 0.05 if root node’s error at low compression bit rate of 2.04bpp, an undesirable block is still visible at the upper right hand corner in Figure 5.5a. At higher threshold settings, a major part of the image is obscured by the “blocking” effect, as shown in Figure 5.5d.

In the total error case, the region area is implicitly considered. Larger regions will have a larger total error if the mean is different from the actual pixel value. Therefore, total error

representation results in an overall better image. Figure 5.6 shows the compression bit rates for total error versus average error representation.

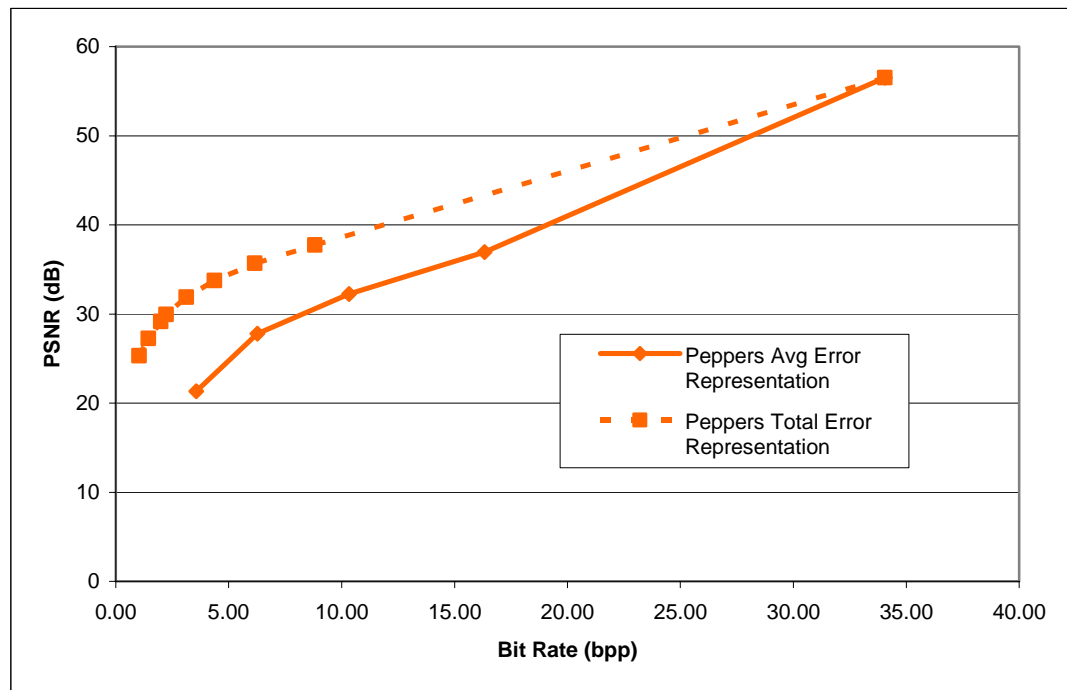


Figure 5.6– PSNR versus Bit Rate for Average and Total Error Representation

It can be seen that total error threshold representation always has a higher PSNR for the same bit rate. Therefore, total error threshold representation is recommended for BSP tree pruning.

5.3 Data bit Compression

5.3.1 Line Coordinate File Compression

In this experiment, the use of a non-standard bit size for representing the line coordinates is explored. In standard C/C++, an integer can be either 8-bit, 16-bit, or 32-bit. As discussed in Section 3.4.4, in practice, not the full range of the integer value is used. This experiment explores the use of a minimum length bit size for the numbers. Table 5.7 shows the file sizes before and after this compression.

Table 5.7 – Line Coordinate Compression File Sizes for Various Images

	Bits/Line Coordinate	Uncompressed Line File Size	Compressed Line File Size	% of Original File Size
peppers	9	496030	279017	56%
lena_small	8	125042	62521	50%
tulips	10	731476	457173	63%
frymire	9	295118	166004	56%
serrano	9	207426	116678	56%

It can be seen that file size decreases significantly if the minimum length bit size instead of the standard 16-bit length is used to represent the line coordinates. For a 512×512 image, only 9 bits are necessary to represent the lines. In comparison with storing the line coordinates using standard 16-bit number, this is a saving of 56%.

5.3.2 Colour Table Compression

In this experiment, the colour table compression technique discussed in Section 3.4.3 is used on the test images. Table 5.8 shows the resulting file size using this technique.

Table 5.8 – Colour Table Compression File Sizes for Various Images

	Uncompressed Colour File Size	Compressed Colour File Size	Colour Table Size	% of Original File Size
peppers	744048	527034	346683	117%
lena_small	187566	125044	151557	147%
tulips	1097217	777196	375828	105%
frymire	442680	221340	6387	51%
serrano	311142	142607	3762	47%

From Table 5.8, it can be seen that some images actually increase in size after applying this technique. This is because the number of colours has to be relatively few compared to number of nodes for this technique to be effective. For the peppers, lena and tulips image, the file size increases. However, for the serrano and frymire images, the file size decreases. In general, photographic images increase and computer graphics images decrease in size. The conditions for this technique to be useful are analyzed in more details in Section 6.1.3 and 6.1.4.

5.4 Summary

From the experimental results, we can draw several conclusions:

1. Although the optimal line selection algorithm yields a optimal size image tree, it is too computationally intensive; therefore, it is not optimal for energy. On the other hand, the binary split method can construct the BSP tree much faster, with a slight increase in file size.
2. Calculating the total square error in the traditional way is too slow. Hence, we have proposed a fast error formula and it yields satisfactory results when pruning images, especially for binary split method constructed images. Therefore, we recommend using the fast error pruning formula for binary split images.
3. We have explored the use of total and average square error representation at nodes. We have found that total error representation performs better because region area should be taken into consideration when pruning the image tree. In contrast, the region area is not accounted for in the average error representation.
4. The use of minimal bit length helps to reduce line coordinate file size significantly. We have discovered that the use of a colour table may or may not help compressing an image depending on the properties of the image. The condition for which the colour table method is useful is analyzed in Section 6.1.4.

Chapter 6 – Analysis

This chapter presents the analysis for the various characteristics of BSP tree representation. In particular, we will analyze the image file sizes, performance of the tree construction methods and pruning formula, image quality with respect to execution time and file size, and the encryption performance with respect to file size and block or key length. From the results of the analysis, we will derive a image quality and file size prediction model.

6.1 File Sizes

File size directly affects the encryption time and the storage space. This section analyzes the relationships between file sizes, image quality, and execution time. First, we will analyze the asymptotic bounds on file sizes. Second, we will derive the conditions for which the image will benefit from using the colour table method of compression.

6.1.1 Tree File and Line Orientation File Size

The worst-case tree file size can be analyzed by analyzing a binary tree [46]. In the worst-case scenario, each pixel is a distinct colour. Hence, no neighbouring pixels can be merged. For a $N \times N$ image, the number of pixels is N^2 . The maximum number of leaf nodes, denoted as L , is equal to N^2 . Let n denotes the number of non-terminal nodes. Since every node except the root has a parent, there must be $(L + n - 1)/2$ parents in the tree because every parent has two children. The number of non-terminal nodes is the same as the number of parents. Hence, $n = (L + n - 1)/2$. Rearranging gives:

$$\text{Number of non-terminal nodes: } n = L - 1 = N^2 - 1. \tag{6.1}$$

The total number of nodes is the sum of all non-terminal nodes and leaf nodes and is equal to $n+L$. Substitute L and rearrange, we get:

$$\text{Maximum total nodes} = 2N^2 - 1 \tag{6.2}$$

Since each node occupies one bit flag in the tree file, the maximum file size is $(2N^2 - 1)/8$ bytes. For a 512×512 image, the maximum tree file size is about 65536 bytes.

Similarly, each non-terminal node occupies one bit in the line direction file to indicate whether the line is horizontal or vertical, the maximum file size for the line direction file is $n/8 = (N^2 - 1)/8$ bytes. For a 512×512 image, the maximum line direction file size is 32768 bytes.

If neighbouring pixels have the same colour, these pixels can be merged according to the BSP compression algorithm. Our experiment results show that photographic images are close to the maximum file size. This is because in natural photographs, neighbouring pixels often differ slightly. However, for computer graphics and line art images, there are usually large homogenous regions. The tree and line direction file size is often a fraction of the maximum file size. Therefore, BSP lossless mode works better for computer graphics and line art images.

6.1.2 Line Coordinate File Size

The number of bits need to reference the lines in a $N \times N$ image is $\lceil \log_2 N \rceil$ bits. The number of non-terminal nodes is $(N^2 - 1)$. Therefore, the maximum line coordinate file size is $\lceil \log_2 N \rceil (N^2 - 1)/8$ bytes. For a 512×512 image, the maximum line file size is 294911 bytes.

6.1.3 Colour File Size

In the worst-case scenario, the maximum number of leaf nodes is equal to the number of pixels, which equals to N^2 for a $N \times N$ image. In the non-colour table mode, each leaf stores the 24-bit colour directly. Therefore, the maximum colour file size is $3N^2$ bytes. In the colour table mode, each node stores the index. If there are K colours in this image, then the number of bits

needed to index to this colour array is $\lceil \log_2 K \rceil$ bits. The maximum colour file size in colour table mode is therefore $(\lceil \log_2 K \rceil N^2) / 8$ bytes.

6.1.4 Colour Table Size

In colour table compression mode, each leaf node stores the index number to the colour palette table instead of the actual colour. Let K denote the number of distinct colours in the image. Assuming 24 bit (3 bytes) colour, the colour table size is $3 \cdot K$ bytes. The number of bits needed to index to this colour table is $\lceil \log_2 K \rceil$ bits. The amount of savings per node is:

$$\text{Savings/node} = 24 - \lceil \log_2 K \rceil \text{ bits/node} . \quad (6.3)$$

Let L denote the total number of leaf nodes in the tree. The total number of savings for the entire image file is:

$$\text{TotalSavings} = \frac{L(24 - \lceil \log_2 K \rceil)}{8} \text{ bytes} . \quad (6.4)$$

There is the extra overhead of storing the colour index table. Therefore, the final effective savings by using the colour table method compared to storing the colours directly is:

$$\text{Effective Savings} = \frac{L(24 - \lceil \log_2 K \rceil)}{8} - 3K \text{ bytes} . \quad (6.5)$$

It can be seen that we can have gain in savings only if:

$$\frac{L(24 - \lceil \log_2 K \rceil)}{8} > 3K . \quad (6.6)$$

By rearranging Eq. (6.6), we get

$$\lceil \log_2 K \rceil < 24(1 - \frac{K}{L}) .$$

(6.7)

From Eq. (6.7), it can be seen that images with relatively a few colours compared to number of nodes (i.e. small K/L) will benefit the most from colour table compression mode. This is typical of computer graphics and line art images. Therefore, colour table compression technique can compress these types of images better than natural photographic images.

6.1.5 Summary

From the analysis in this section, it can be seen that in lossless mode, BSP Tree compression algorithm performs better on computer graphics and line art images. These images have relatively few colours and larger homogenous regions. The larger homogenous regions make it possible to merge pixels. Hence, the image tree is smaller and results in reduction in tree, line direction, and line coordinate file size. The relatively few colours for these types of images allow the colour indexing method to store the colour file efficiently. In our experimental results in Table 5.8, it can be seen that computer graphics images such as the frymire and serrano images benefits from colour indexing technique by a reduction of colour file size of about 50%.

6.2 Performance

This section compares and discusses how the tree construction method and the error calculation formula affect the execution time, hence the energy consumed by the system. We will estimate the computation energy cost by comparing the number of operations required for the optimal line and the binary split tree construction techniques and discuss qualitatively how our proposed fast pruning formula help to reduce computation energy.

6.2.1 Operations required for the Tree Construction Mechanisms

In the original method, the LSE transform in Eq. (2.1) is calculated for every potential partitioning line. For the top integral in Eq. (2.1), a total of (x_2-x_1) multiplications and (x_2-x_1) additions is required. For the denominator, a total of (x_2-x_1) additions are required. Therefore, for

each LSE transform, if we add up the total number of operations required for the top and bottom integral, a total of (x_2-x_1) multiplications and $2(x_2-x_1)$ additions are required.

$$LSETransformOp = (x_2 - x_1)Multiplications + 2(x_2 - x_1)Additions . \quad (6.8)$$

For a horizontal line, x_1 is the x-coordinate of the beginning of the region and x_2 is the x-coordinate of the endpoint. Therefore, for any $p \times q$ sub-region of the image under consideration, (x_2-x_1) is the x-size (i.e. p) of the region. For a vertical line, (x_2-x_1) is the y-size (i.e. q) of the region. Substitute p and q into Eq. (6.8), we get:

$$LSETransformOp = \begin{cases} p \cdot Multiplications + 2p \cdot Additions, & \text{if candidate line is horizontal} \\ q \cdot Multiplications + 2q \cdot Additions, & \text{if candidate line is vertical} \end{cases} . \quad (6.9)$$

Every potential horizontal and vertical line is tested with LSE transform. For a $p \times q$ region, there are q horizontal lines and p vertical lines considered for the LSE transform. Let C_i denotes the total number of operations needed for sub-region i . The total number of operations is the sum of all the operations for the horizontal and vertical candidate lines. Multiply Eq. (6.9) by the total number of horizontal and vertical lines and add, the total number of operations, C_i , required for testing all the candidate lines for sub-region i in one iteration is:

$$\begin{aligned} C_i &= numHorizontalLines \cdot LSETransformOp + numVerticalLines \cdot LSETransformOp \\ &= q \cdot LSETransformOp + p \cdot LSETransformOp \\ &= q \cdot [p \cdot Multiplications + 2p \cdot Additions] + p \cdot [q \cdot Multiplications + 2q \cdot Additions] . \\ &= pq \cdot Multiplications + 2pq \cdot Additions + pq \cdot Multiplications + 2pq \cdot Additions \\ &= 2pq \cdot Multiplications + 4pq \cdot Additions \end{aligned} \quad (6.10)$$

Eq. (6.10) shows the total number of operations required for one iteration of the tree partitioning process. This calculation needs to be repeated when the region has been partitioned and new sub-regions are formed.

For the worst-case scenario of a $N \times N$ image, no pixels can be merged and the image forms $N \times N$ leaf nodes. Each iteration creates two sub-regions. Therefore, the number of lines

considered in each iteration increases at a rate of 2^i . Therefore the total number of LSE transform operations is:

$$\begin{aligned}
TotalLSETransformOperations &= \sum_{i=0}^{\log_2 N-1} numHorizontalLines \cdot C_i + \sum_{i=0}^{\log_2 N-1} numVerticalLines \cdot C_i \\
&= \sum_{i=0}^{\log_2 N-1} 2^i \cdot C_i + \sum_{i=0}^{\log_2 N-1} 2^i \cdot C_i \\
&= 2 \cdot \sum_{i=0}^{\log_2 N-1} 2^i \cdot C_i
\end{aligned} \tag{6.11}$$

Now we consider the number of pixels considered at each iteration. Although the number of lines considered at each iteration is doubled, the region size is halved at each split. For a horizontal line, the horizontal dimension (i.e. p) stays constant as N while the vertical dimension (i.e. q) is halved (divided by 2^i) at each iteration. Substituting value for p and q in C_i into Eq. (6.11), the total number of operations spent in LSE transform for a $N \times N$ image is:

$$\begin{aligned}
TotalLSETransformOperations &= 2 \cdot \sum_{i=0}^{\log_2 N-1} 2^i \cdot C_i \\
&= 2 \cdot \sum_{i=0}^{\log_2 N-1} 2^i \cdot \left(2 \frac{N^2}{2^i} \cdot Multiplications + 4 \frac{N^2}{2^i} \cdot Additions \right) \\
&= 2 \cdot \sum_{i=0}^{\log_2 N-1} (2N^2 \cdot Multiplications + 4N^2 \cdot Additions) \\
&= 2 \log_2 N \cdot (2N^2 \cdot Multiplications + 4N^2 \cdot Additions)
\end{aligned} \tag{6.12}$$

This does not include the time for calculating the total square error for the chosen candidate lines. Calculation of the total square error requires $p \times q$ additions and multiplications each time. There is at least one total square error calculation in each split. Using similar reasoning as in deriving Eq. (6.12), the minimum total operations spent in calculating total square error is:

$$\begin{aligned}
MinTotalSquareErrorOperations &= 2 \log_2 N \cdot TotalSquareErrorOp \\
&= 2 \log_2 N \cdot (N^2 \cdot Multiplications + N^2 \cdot Additions)
\end{aligned} \tag{6.13}$$

Adding Eq. (6.11) and (6.13) together, the minimum total number of operations required for constructing a optimal line selection BSP tree is:

$$\begin{aligned} \text{MinTotalOperations} &= 2\log_2 N \cdot \text{TotalSquareErrorOp} \\ &= 2\log_2 N \cdot (3N^2 \cdot \text{Multiplications} + 5N^2 \cdot \text{Additions}) \end{aligned} \quad (6.14)$$

This is only the lower bound of the total number of operations required for the optimal line selection process. Usually, more than one line passes the LSE transform condition. Therefore, the result in (6.13) is usually significantly greater.

In the binary split case, for each split, only one addition and one multiplication is needed to find the midpoint using Eq. (2.3). The number of splits is equal to the number of non-terminal nodes thus there are N^2-1 splits. Therefore, only N^2-1 additions and N^2-1 multiplications are required. No calculation of the total square error is needed

It can be seen that the binary split method is much faster than the optimal line selection method. Although one can argue that neglecting the optimal line may make the tree suffer from the same problems of a quadtree, (where if the split lines do not align with the true object boundaries will result in a lot of nodes), the binary split method is still better than a quadtree because only two children nodes are created instead of four children nodes for each split. Therefore, if two neighbouring quadrants are homogenous, it will not split the homogenous nodes furthermore. On the other hand, in the quadtree case, it requires one more split because the partitioning will terminate. From the experimental results, it can be seen that the file size increase due to neglecting the optimal line is acceptable. Therefore, the binary split method is a simple yet efficient method to construct the BSP tree. It has the combined benefits of the efficiency of representation of a BSP tree and the efficiency in tree construction of a quadtree.

6.2.2 Error Calculation Formula

Error calculation is a bottom-up algorithm. One needs to traverse all the leaf nodes in order to get the exact pixel colour values to calculate the mean and error. Then, the mean at each node can be calculated in a bottom-up fashion. In the original Radha et al algorithm, the total square error needs to be calculated at every level of the tree. This may yield to better pruning

decisions, yet it is not efficient since this calculation needs to be repeated again at the higher levels. It is desirable to be able to reuse intermediate results from previous calculations.

Our proposed formula reuses the partial error previously calculated from the lower nodes. This speeds up the pruning exponentially. Calculation of the total square error requires N^2 . However, the proposed method in Eq. (3.3) reuses the children's total error and derives the new total error by adding a weighted average of the children node's error and standard deviation. No recalculation of error for the subtree regions is needed.

Total error is more useful than the average error per pixel because the error selection should be proportional to the region's area. A larger region cannot tolerate as much error as a smaller region because the error is more visible. The use of total error reflects the importance of the area should have on the pruning decision. A large area should not be pruned if the error is huge and a smaller area may be pruned for the same average error.

6.2.3 Summary

Execution time directly affects energy. The proposed algorithm aims to reduce energy consumption by cutting down the execution time for constructing the image tree and pruning the tree. In the new binary split tree construction method, the execution time is cut down significantly, at the expense of a slight increase in tree file size. In the fast error calculation method, the execution time is reduced by reusing results from previous calculations. The tradeoff is a slightly less accurate pruned image quality. However, this is justified by the significant decrease in execution time.

6.3 Image Quality

Image quality is another factor that affects the end user. This section analyzes the different relationships between the image quality and execution time and file size.

6.3.1 Execution Time versus Image Quality Threshold

Image quality threshold has a light effect on execution time. Figure 6.1 shows a plot of the execution time versus the pruning threshold (shown in Table 6.1) for the peppers image. It can be seen that the lower the image quality, the faster it takes to generate the image file. This is because if the pruning threshold is higher, more nodes are pruned at higher level of the tree. Thus, there will be less nodes to be considered for pruning. Pruning is implemented in a top to bottom fashion. It starts from reading the error values at the root node. If the threshold is higher, then the tree will be pruned at a higher level, hence eliminating the time needed to traversal the subtrees from the pruned node. However, the image quality threshold does not affect the execution time as significantly as the error computation formula.

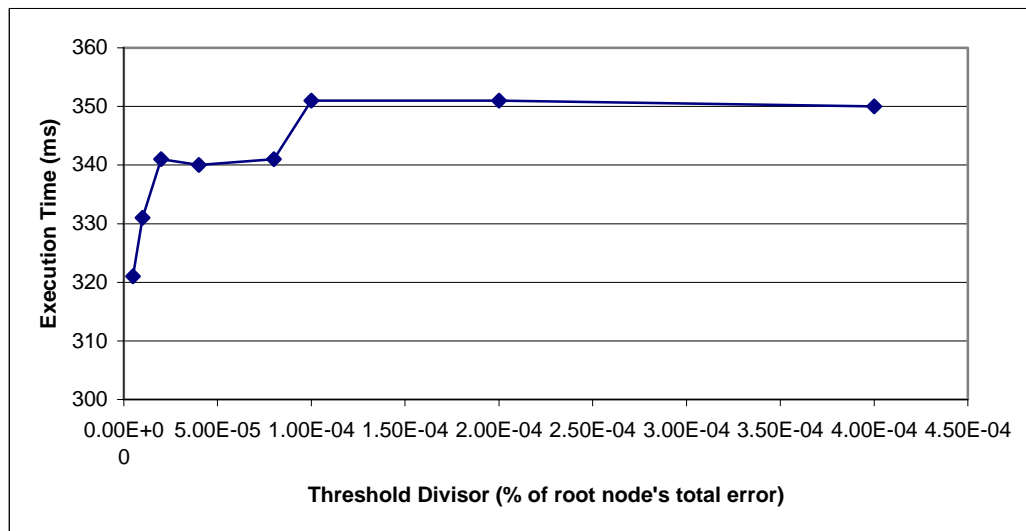


Figure 6.1 – Pruning Time versus Pruning Threshold

Table 6.1 – Pruning Time at different thresholds

Threshold	Total Pruning Time (ms)
0	0
5.00×10^{-6}	321
1.00×10^{-5}	331
2.00×10^{-5}	341
4.00×10^{-5}	340
8.00×10^{-5}	341
1.00×10^{-4}	351
2.00×10^{-4}	351
4.00×10^{-4}	350

6.3.2 Image Quality versus File Size

In general, the higher the image quality, the larger the file size is. Figure 6.2 shows the image quality and file size relationship for various test images. The file size is normalized as a fraction of the best quality file size, which is assigned a value of 1. Figure 6.3 views the same data from a slightly different perspective. The compression is expressed as the bit rate per pixels. The image data are generated without using the colour table compression method.

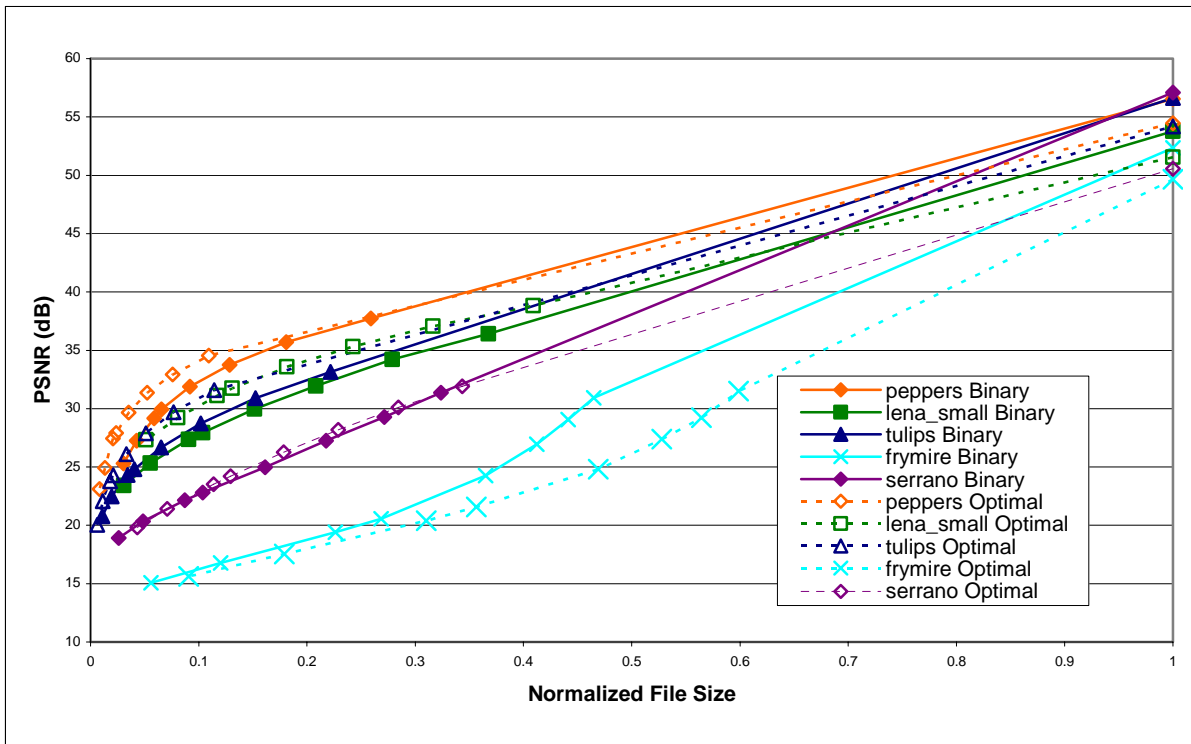


Figure 6.2 – PSNR versus Normalized File Size

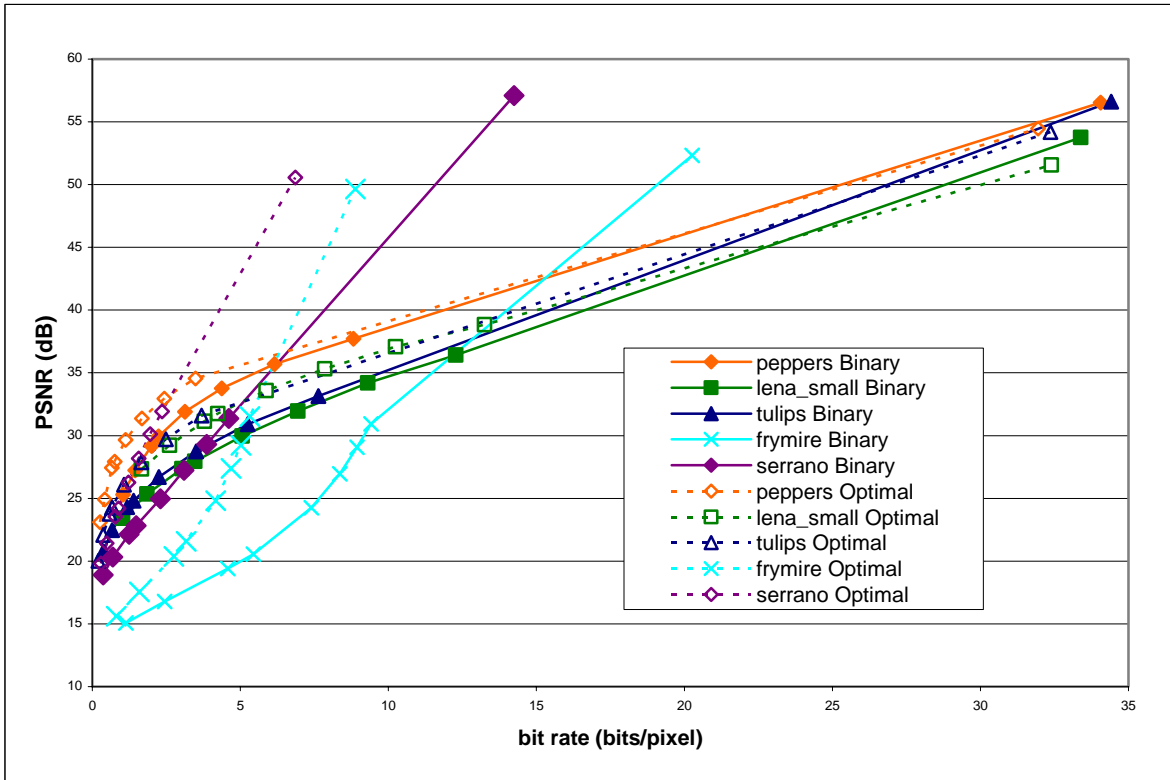


Figure 6.3 – PSNR versus Bit Rate

Figure 6.2 shows that at low to medium compression ratio (i.e. file sizes closer to 1), there is a linear relationship between the PSNR and file size. However, at higher compression ratios (high thresholds), the PSNR deviates from this linear relationship and behaves like an exponential relationship. This is evident by examining the trend at lower bit rate in Figure 6.3. This is because at higher compression ratios, many of the nodes that represent large region areas are pruned. Therefore, the image quality drops significantly as these large region nodes are deleted. Then, at about 20% of the best quality file size or 5bpp in Figure 6.3, the PSNR starts to increase linearly with the file size. This is because at low to medium compression ratios, the nodes that are pruned usually represent smaller regions, so the impact on overall quality is less when these nodes are merged. The exceptions to the general trend are the frymire image and the serrano image. Both images are computer graphic images. Both images do not exhibit the non-linear behaviour at lower bit rate.

In summary, the compression bit rate should operate in the linear region in order to have a relatively consistent image quality. Below that bit rate, image quality suffers significantly without much reduction in file size.

6.3.3 Comparison to other Image Compression Systems

Compared to TIFF, if a slight loss in image quality is allowed, then our compression method is superior to TIFF for most images. At a low threshold of 5.00×10^{-06} , the loss in image quality is almost not noticeable. At such threshold, our BSP compression scheme outperforms TIFF for most images, as shown in Table 6.2. Notes that PSNR is not included for TIFF files because TIFF LZW compression is lossless; therefore, PSNR for lossless images equal infinity. In Table 6.2, the images are generated in binary split tree construction, no colour table compression mode unless specified otherwise. The exception is the frymire image, which appears to be more suited for optimal line tree construction method with colour table compression.

Table 6.2 – TIFF and BSP Tree File Sizes for Various Images

	TIFF File Size (bytes)	Bit Rate (bpp)	BSP File Size (bytes)	Bit Rate (bpp)	PSNR (dB)
peppers	530764	16.20	289004	8.82	37.73
frymire (binary split, with colour table compression)	181672	5.54	232647	7.10	30.94
frymire (optimal line, with colour table compression)	181672	5.54	140615	4.29	31.51
serrano	166576	5.08	151071	4.61	31.36

JPEG images at the best compression and best quality settings in Adobe Photoshop are also generated for comparison. For the best compression JPEG image, a BSP tree image at a comparable bit rate is picked for comparison. The best quality BSP tree image is compared with the best quality JPEG image. Figure 6.4 shows the “best compression” and the “best quality” JPEG and BSP image for the peppers image and Figure 6.5 shows the results for the frymire image.





Threshold	JPEG	BSP Tree
Best Compression		
Best Quality		

Figure 6.4 – Peppers JPEG and BSP Tree Image at the best compression and quality settings

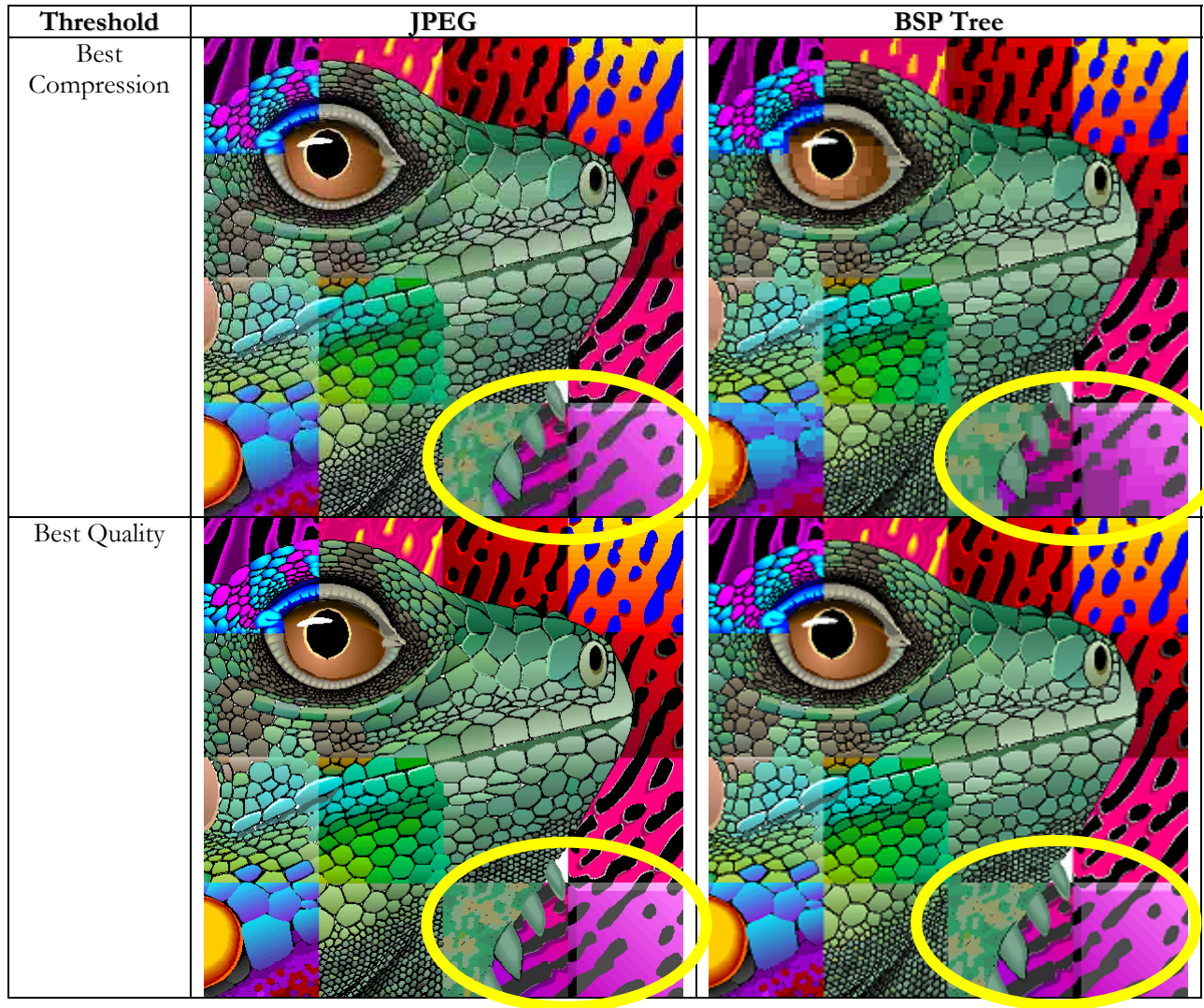


Figure 6.5 – Frymire JPEG and BSP Tree Image at the best compression and quality settings

Table 6.3 and Table 6.4 shows the file size, bit rates and the PSNR for the peppers and frymire images shown in Figure 6.4 and Figure 6.5, respectively.

Table 6.3 – JPEG and BSP Tree File Sizes for Peppers Image

	JPEG File Size (bytes)	Bit Rate (bpp)	PSNR (dB)	BSP File Size (bytes)	Bit Rate (bpp)	PSNR (dB)
Best Compression	23308	0.089	32.19	25660	0.098	29.53
Best Quality	249787	0.953	44.35	1046639	3.99	54.46

The BSP tree with the bit rate closest to the best compression JPEG is chosen for comparison. The optimal tree peppers image pruned at 10^{-5} threshold using total square error formula, with no colour table compression is chosen for comparison. The best quality image for comparison with the JPEG is the optimal tree image with the colour table compression disabled.

Table 6.4 – JPEG and BSP Tree File Sizes for Frymire Image

	JPEG File Size (bytes)	Bit Rate (bpp)	PSNR (dB)	BSP File Size (bytes)	Bit Rate (bpp)	PSNR (dB)
Best Compression	88012	0.335	21.13	86006	0.328	20.39
Best Quality	475277	1.813	47.49	200632	0.765	49.66

The BSP tree with the bit rate closest to the best compression JPEG is chosen for comparison. The optimal tree frymire image pruned at 10^{-4} threshold, with colour table compression enabled is chosen for comparison. Whether the error formula is slow or fast error does not seem to affect the results a lot. Therefore, it is not shown here. The best quality image for comparison with the JPEG is the optimal tree image with the colour table compression enabled.

It can be seen that BSP tree compression works better with the frymire image than the peppers image. For the frymire image, with colour compression enabled, the BSP tree compression can compress more than 2 times better at the best quality settings in Table 6.4. Even if we set JPEG at the best compression settings, BSP tree compression is still able to match the compression ratio of JPEG. However, for the peppers image, JPEG outperforms the BSP tree compression in all categories. Therefore, it can be concluded that JPEG is more suitable for natural photographic images, and BSP trees works better for computer graphics images.

6.3.4 Summary

In summary, the higher the image quality, the larger the file size and the pruning time will be. In turn, this translates to using more time for encryption and transmission. Hence, higher image quality consumes more energy. Our BSP tree scheme can outperform TIFF for most images. Also, BSP trees are more suitable for computer graphics images while JPEG is better for photographic images.

6.4 Scalable Encryption

The purpose of encryption is to scramble the tree data so that no one can read the data without knowledge of the secret key. Since the tree bits denote the tree structure of the image tree, it is not trivial to reconstruct the image just by examining the unencrypted colours. In lower security levels, the line coordinate file is unencrypted. Although it is theoretically possible to reconstruct the first few sections of the regions close to the root node by looking at the line coordinate values at the front of the file, the attacker will not be able to know whether the line coordinate values are associated with the left sub-tree or as a child at the deeper level. Therefore, the image is still quite reasonably secure without encrypting the lines.

6.4.1 Encryption Time versus File Size

The time it takes to encrypt the image file versus the tree file size using partial encryption is investigated. Table 6.5 shows the total encryption time for encrypting the peppers image pruned at different thresholds.

Table 6.5 – Encryption Time at different thresholds

Threshold	Tree File Size (bytes)	Line File Size (bytes)	Total Encrypted Size (bytes)	Percentage of Total File Size	Encryption Time (ms)
0	62005	279017	341022	30.56%	250.40
5.00×10^{-6}	16057	72250	88307	30.56%	63.10
1.00×10^{-5}	11217	50468	61685	30.56%	47.10
2.00×10^{-5}	7966	35842	43808	30.56%	31.00
4.00×10^{-5}	5701	25648	31349	30.56%	24.10
8.00×10^{-5}	4064	18279	22343	30.56%	18.00
1.00×10^{-4}	3644	16389	20033	30.56%	15.00
2.00×10^{-4}	2642	11882	14524	30.56%	12.00
4.00×10^{-4}	1891	8504	10395	30.56%	8.00

The results in Table 6.5 are generated using security level 5 parameters for a binary split peppers image using the fast error pruning formula. Figure 6.6 shows the plot of the data in Table 6.5.

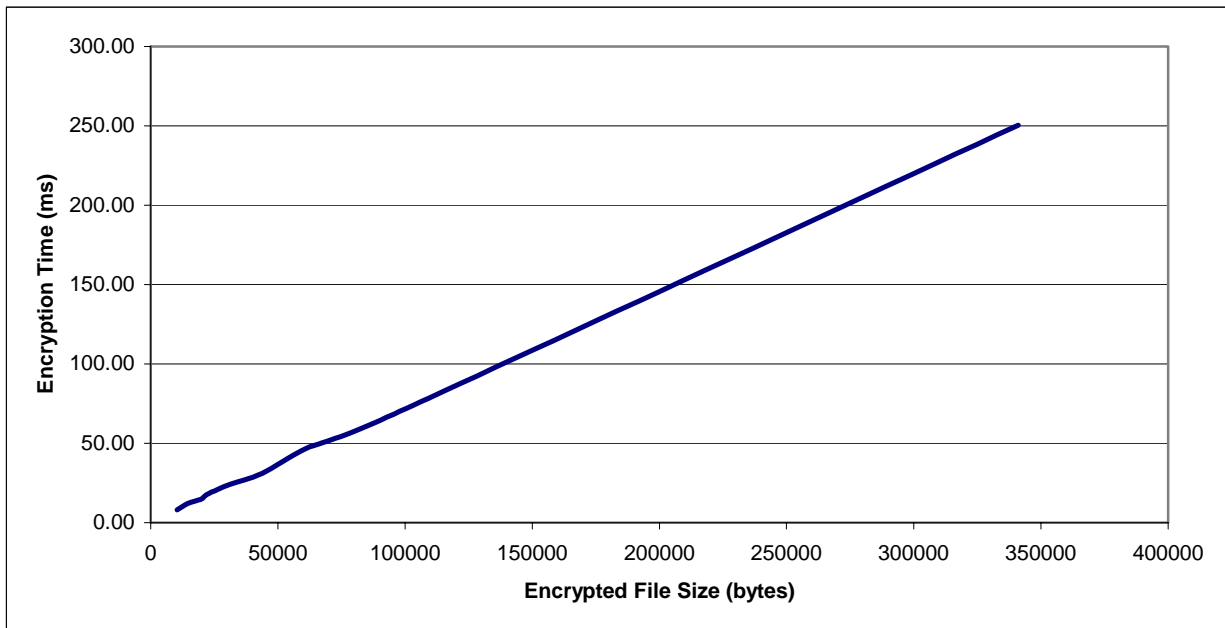


Figure 6.6 – Encryption Time versus Encrypted File Size

It can be seen that encryption time increases linearly as the file size increases. Since file size is related to image quality, it can be inferred that the encryption time increases as image quality increases. Therefore, there is a tradeoff between the encryption energy and the image quality.

6.4.2 Encryption Time versus Partial Encryption Percentage

We have investigated the time it takes to encrypt the tree and line coordinate files versus the fraction of file encrypted. Figure 6.7 shows the results for encrypting different percentages of tree file sizes for the different block sizes. The test image used is the best quality peppers image, constructed with binary split method.

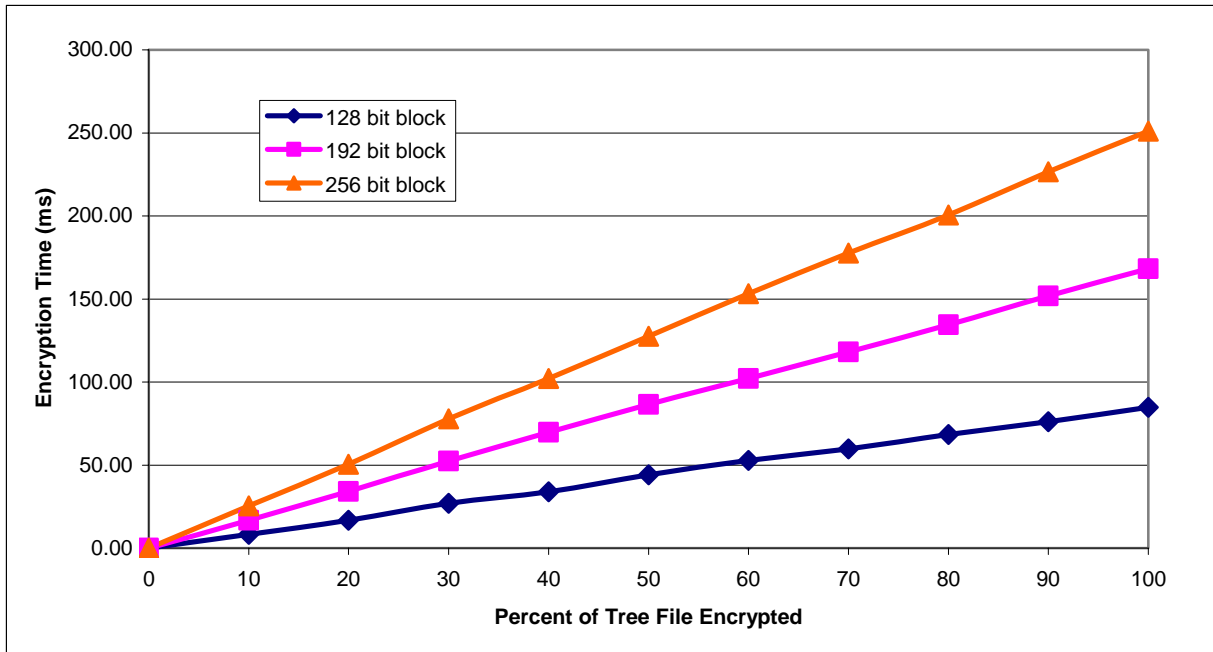


Figure 6.7 – Encryption Time versus Percentage of Tree File Encrypted

It can be seen clearly that the encryption time increases linearly as the percentage of files encrypted increases. This is because AES encryption is directly proportional to the number of input data blocks to be encrypted. To conserve energy, one should decrease the fraction of the tree file encrypted by selecting a lower security level.

6.4.3 Encryption Time versus Encryption Key and Block Length

We have also investigated how the encryption key and block size affects the encryption time. The best quality binary split frymire image is used as the input image for the encryption engine for encryption the tree file (36891 bytes).

Table 6.6 – Encryption Time (ms) for Different Key and Block Length

Key Length	128 bit block	192 bit block	256 bit block
128	38.07	82.47	131.20
192	44.07	86.13	134.90
256	50.07	100.77	151.53

Figure 6.8 shows the results in graphical form.

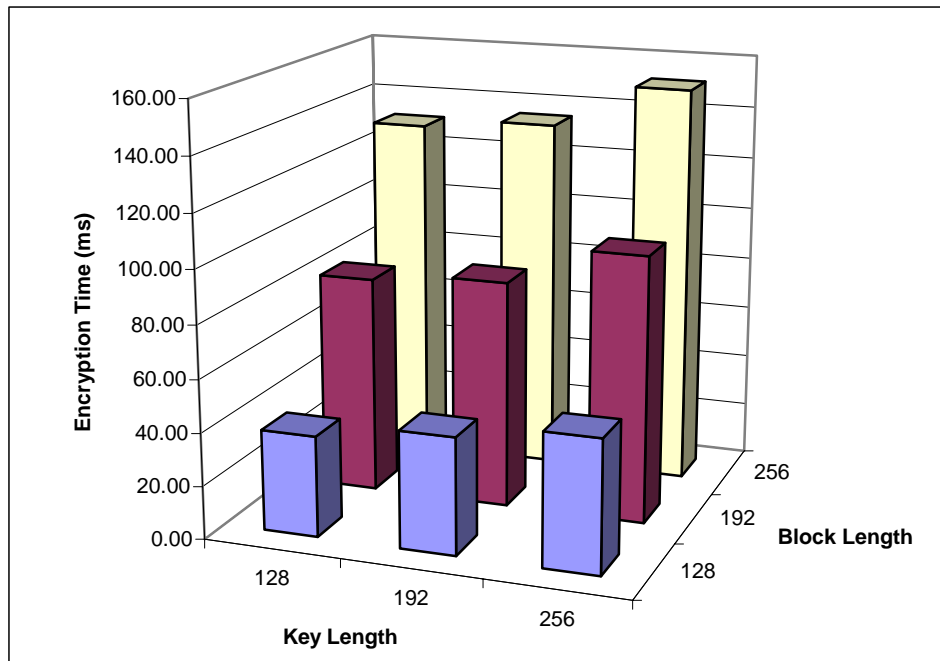


Figure 6.8 – Encryption Time versus Key and Block Length

From the results, it seems that key length does not affect the encryption time as strongly as the block length. As the key length increases, the encryption time only increases slightly. On the other hand, as the block length increases, the encryption time increases significantly.

In Table 6.2 and Table 6.4, the TIFF frymire image and the best quality JPEG frymire image file size are 181672 and 475277 bytes, respectively. For TIFF and JPEG, the entire file must be encrypted for it to be secure. On the other hand, our partial encryption system only needs to encrypt the BSP tree part of the frymire image, which is 36891 bytes for the best quality image. This is only 20.3% and 7.7% of the amount of data that needs to be encrypted for TIFF and JPEG, respectively. Therefore, we have saved encryption energy significantly.

6.5 Selective Focus Regions Feature

One of the major advantages of a BSP tree is the feature of selectively choosing important areas of the image to have a higher quality than the less important parts. For the less important areas, one can choose a higher threshold to compress the image more, while in the more important areas one needs to keep a higher quality image. For example, in Figure 6.9a, the lena image is pruned with the same threshold for the entire picture. The facial features and the background are pruned with the same quality. As a result, the facial features are blurred (highlighted with circles). On the other hand, in Figure 6.9b, the facial region is pruned with a lower threshold, while the less important background is pruned with a higher threshold. One can see that Figure 6.9b appears to have a higher quality than Figure 6.9a because the important facial details are conserved. Selective focus region feature should theoretically work well for quadtrees too, although it has not been suggested in previous research in [39] and [40].

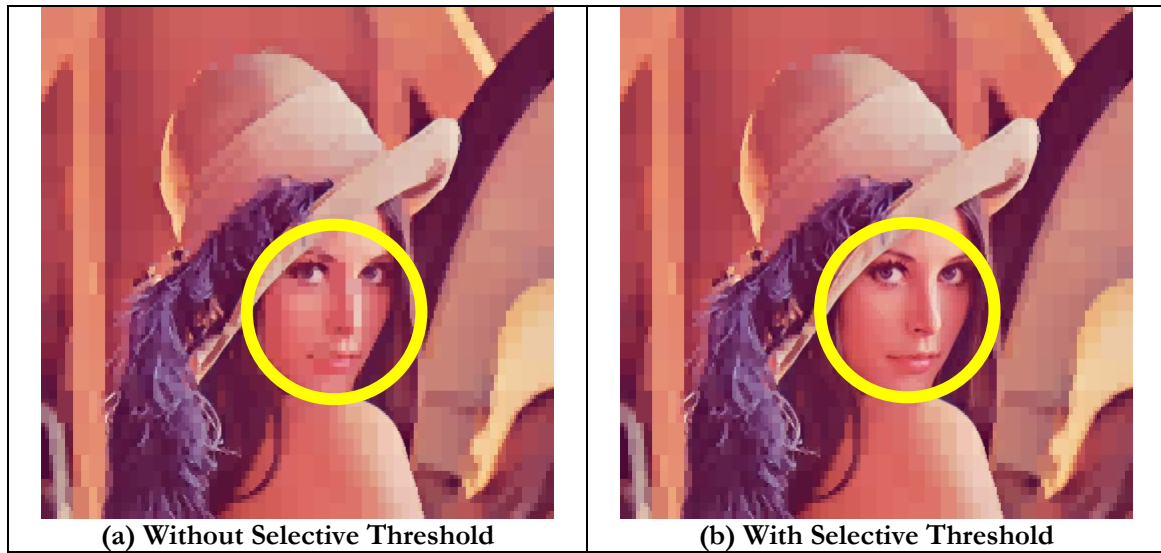


Figure 6.9 – Lena Image with Selective Threshold Regions

6.6 Image Quality Prediction Model

It is desirable to be able to predict image quality at different thresholds. In this section, we derive a simple image quality model to estimate the image quality given the best quality image's PSNR and the minimum acceptable PSNR.

From our experiments, we have observed that the image quality decreases exponentially as the threshold increases. Therefore, an exponential model has been used to estimate the image quality at different thresholds. Figure 6.10 shows the plot of the estimated curve and the actual curve of the PSNR versus threshold.

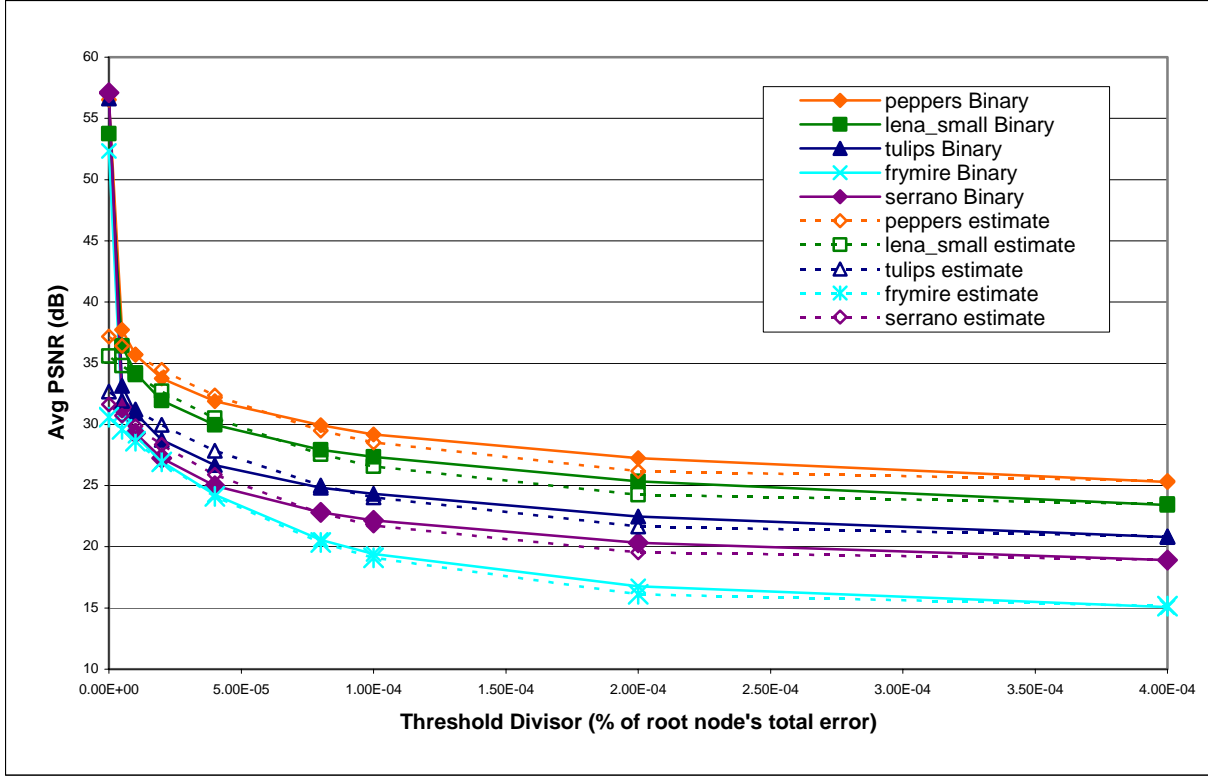


Figure 6.10 – Estimated PSNR versus threshold

In Figure 6.10, the solid lines are the average PSNR measured at different pruning thresholds, the dotted lines show the estimates according to the empirical formula:

$$\lambda_0 e^{-\theta(t+t_0)} + \phi \quad (6.15)$$

where λ_0 is the best quality image's PSNR, ϕ is the minimum acceptable quality's PSNR, θ is the decaying factor, t is the threshold divisor (expressed as the percentage of the root node's total error) and t_0 is the projected y-intercept for the estimated curves. The formula is similar to failure intensity formula found in software engineering [47].

λ_0 scales the estimated curve so that it decays at the same scale as the original curve. ϕ shifts the curves up so that the asymptotic value of the estimated curves will match the image's asymptotic value as the threshold moves to infinity. Note that PSNR usually does not approach

zero as the threshold moves to infinity because it will approach zero only if the MSE is equal to the peak magnitude squared (255^2 for 8-bit values). In other words, only when the approximated pixel value is at the extreme of the actual pixel value (i.e. estimated pixel value is black while the actual pixel is white). In practice, this does not happen. Therefore, ϕ is added to compensate for this fact. t_0 is added to the threshold to shift the curves to the left so that the estimate curves match the actual measurement. θ determines how quickly the image degrades as the threshold value increases. Although the values for ϕ , θ and t_0 vary for different images, they fall in surprisingly close ranges. It is found that θ is typically between $1.3\text{-}1.5 \times 10^4$ and t_0 is typically between $0.9\text{-}1.2 \times 10^4$. Table 6.7 shows the values for θ and t_0 and other parameters determined empirically in the experiments.

Table 6.7 – Parameters for Image Quality Prediction Model

	λ_0	Θ	t_0	Φ
peppers	56.53	1.3×10^4	1.2×10^4	25.30
lena_small	53.76	1.35×10^4	1.1×10^4	23.42
tulips	56.62	1.3×10^4	1.2×10^4	20.79
frymire	52.34	1.35×10^4	0.9×10^4	15.08
serrano	57.09	1.5×10^4	1.0×10^4	18.90

Note that the point at zero threshold is an exception. The y-intercept of the estimated curves equals to $\lambda_0 e^{-\theta t_0} + \phi$. This value lies between 30-40dB for all these images. However, the actual value at the y-intercept should be λ_0 . This should not present major problems since there is not much need to estimate the image quality at the zero threshold (it is already at the best quality). In Figure 6.10, it can be seen that the estimated curves quickly converge to the actual curves as the threshold moves away from zero. This simple model predicts the image quality quite accurately, usually within 2-3dB of the actual PSNR.

6.7 File Size Prediction Model

Similarly, we have also investigated the relationship between the PSNR and file size. Figure 6.11 shows the curves for the actual file size and the estimated file size for different PSNR for various images. The file sizes are normalized to the best quality image's file size to facilitate comparison for different images.

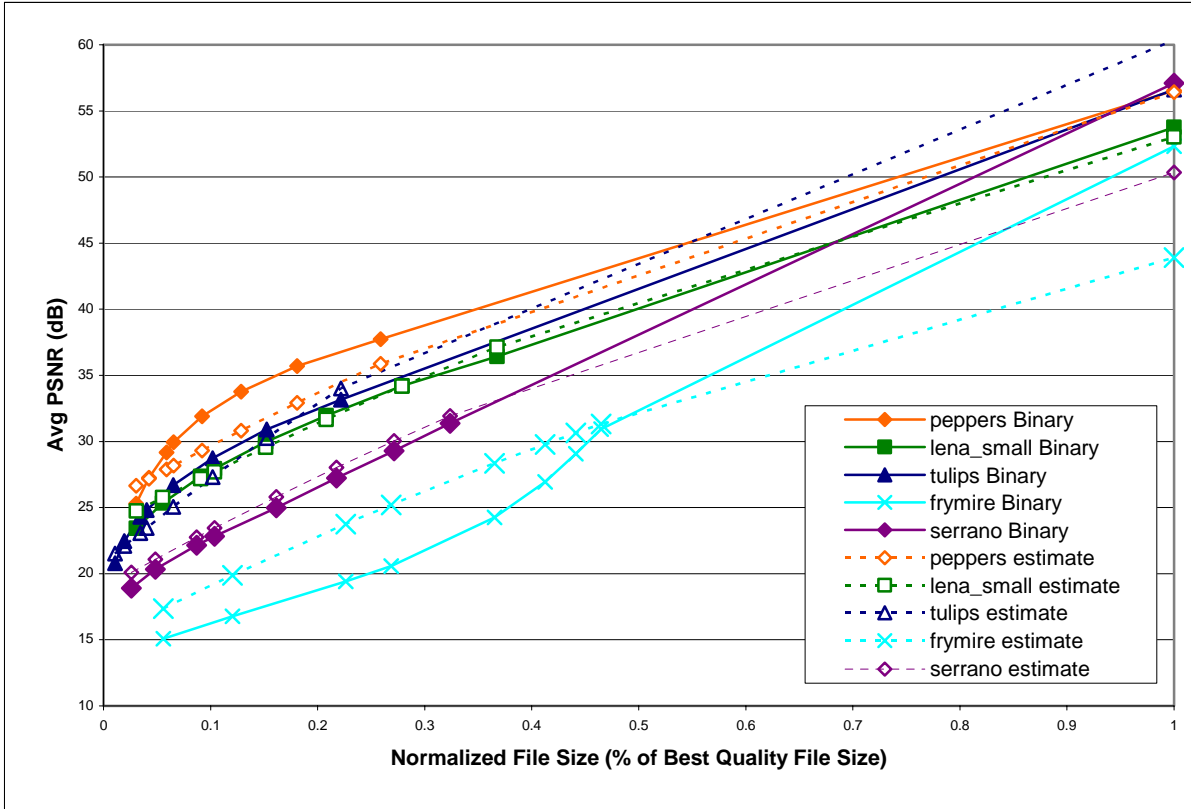


Figure 6.11 – Estimated PSNR versus File Size

In Figure 6.11, the solid lines are the average PSNR measured for different file sizes, the dotted lines show the estimates according to the empirical formula:

$$\lambda_0(1 - e^{-\theta s}) + \phi \tag{6.16}$$

where λ_0 and ϕ are the same parameters from Eq. (6.15), θ is a constant that is determined empirically for different image and s is the normalized file size (expressed as the percentage of the best quality file size). Table 6.8 shows the values for the parameters determined empirically in the experiments.

Table 6.8 – Parameters for File Prediction Model

	λ_0	Θ	Φ
peppers	56.53	0.8	25.30
lena_small	53.76	0.8	23.42
tulips	56.62	1.2	20.79
frymire	52.34	0.8	15.08
serrano	57.09	0.8	18.90

Note that θ is the same for most images except the tulips image. Therefore, this model can predict file size for the test images accurately without major adjustments. This prediction model allows users to predict the image file size for a given image quality. Therefore, users will be able to generate an appropriate image file size based on the required image quality and energy available.

Chapter 7 – Discussion and Conclusion

7.1 Comparison to Previous Research

Previous research on BSP compression focuses primarily on the compression aspect rather than the energy aspect. Sometimes even though these algorithms can compress an image at higher ratios, the computational cost may be too high for handheld energy-constrained devices to justify their use. Also, many of these researches are done on greyscale images. This thesis extends the idea to colour images in a simple yet effective way.

Compared to the previous proposed energy efficient schemes for image compression, our approach requires only a few overhead for re-scaling. Both the JPEG [20],[21] and wavelet [24],[25] energy efficient schemes require re-computations for the quantization and entropy coding stages. For our approach, once the best-quality BSP tree image representation is built, scaling it to any image quality is trivial since no re-computations of the node errors are required.

Although there are previous researches on partial encryption [31], [37], [38], [39], [40] and BSP compression [14], [15], they have not been considered together. To the best of our knowledge, this thesis is the first to use the benefits of a BSP tree to optimize for energy, while maintaining scalability and user flexibility.

This thesis has proposed the idea of selecting a different pruning threshold for different areas using a BSP tree. This allows users to conserve the details in the important parts of the image. After all, it is the end users' perception of the image that is ultimately important. Compression algorithms that affect the image globally do not allow such flexibility. For example, JPEG based systems uses cosine transform based compression techniques, which affects all the coefficients in each block. Thus, they do not allow selectively having different quality levels at different regions in the image. Quadtree system allows partial encryption and selective focus area; however, storage wise it is not as efficient as binary trees. We have proposed a binary split scheme in building the BSP tree, which combines the benefits of both worlds: it has the execution efficiency of quadtree in splitting the image, but also has the compact storage property of an

optimal line BSP tree. Our experiment results show that the file size only increases slightly when using a binary split scheme.

It is desirable to predict the image quality and file size before the image is generated. Previous research does not address this issue. We have presented a simple model for predicting image quality and file size. With such model, users will be able to predict the image quality and file size *before* the image is compressed.

Regarding the compression performance, we have compared the bit rate with other compression systems. For the data based on 8-bit greyscale images, we have multiplied the bit rate by three for a fair comparison to the 24-bit colour used in this system. Table 7.1 shows the comparison to other image compression systems. The values in parenthesis are the greyscale value multiplied by three.

Table 7.1 – Compression performance compared to other systems

	Quadtree		JPEG		Fractal Binary Tree		BSP Tree	
	Bit rate	PSNR	Bit rate	PSNR	Bit Rate	PSNR	Bit Rate	PSNR
peppers	0.516	N/A	N/A	N/A	0.60 (1.80)	32.5	1.67	31.37
lena_std	0.67 (2.01)	30.36	0.125 (0.375)	26.2	0.24 (0.73)	31.0	0.73	26.57
frymire	N/A	N/A	1.813	47.49	N/A	N/A	0.765	49.66

Our BSP tree compression system is comparable to other systems performance for the peppers image. However, for the lena image, other systems work better. Unfortunately, we are unable to find the data for other images from the literature. It would be helpful if we can compare the performance for computer graphics type of images.

7.2 Limitations, Advantages and Disadvantages

Our proposed BSP tree compression and encryption scheme is simple and energy efficient. It is easy to adjust image quality without many re-computations, thus saving computational costs. There is no need for the original image once the BSP tree has been created. It also gives users great flexibility by introducing the idea of variable image quality in different regions. This allows the system to enhance compression without sacrificing the quality for the important areas of the image. Our image quality prediction metric gives users the flexibility to choose a suitable image quality or file size that satisfies their needs for the image.

With respect to the compression performance, BSP trees are better suited for computer graphics images than photographic images when compared to JPEG. This is because BSP tree compression compresses better when there are larger homogenous regions, which is the case for computer graphics. In contrast, photographic images have smooth transitions between pixels. Therefore, neighbouring pixels are usually slightly different from each other. JPEG is more specialized in handling such cases. Our BSP tree scheme is also better than quadtrees in most cases because partitioning lines are more flexible and less nodes are created to represent the same image.

With respect to partial encryption performance, BSP is superior to JPEG. Many of the previous JPEG partial encryption schemes are either inefficient or insecure. BSP tree images only need to encrypt a small fraction of the total image size. Since only part of the data is encrypted, it is theoretically less secure than a complete encryption. However, we do not see our partial encryption scheme to suffer from any major weaknesses.

7.3 Conclusion

In this thesis, we have proposed a energy efficient system for compressing and encrypting images. Compression is performed to save the energy cost used for transmission. Encryption is performed for ensuring the confidentiality of the image. To conserve energy, partial encryption is performed by only encrypting the tree and line part of the image. Users are also given the choice of different security levels to decide the optimal tradeoffs in security and energy consumption.

With respect to compression performance, we have found that our proposed scheme compresses computer graphics better than photographic images. Since computer graphics have large homogenous regions, our BSP compression algorithm is effective in merging those neighbouring pixels together. Also, the number of colours relative to number of nodes is relatively few in computer graphics. Therefore, our proposed colour table compression scheme is very effective.

With respect to security, our partial encryption system saves energy by only encrypting the important parts of an image. Without the knowledge of the BSP tree structure, it is difficult for

the attacker to construct the image using the line coordinates and colours alone. Therefore, security is not compromised while energy is saved.

Image quality is affected by the compression ratio in a lossy BSP compression. The higher the threshold, the more compressed the image will be, while the image quality is sacrificed at the same time. It is up to the end user to decide whether the sacrifice in image quality is acceptable for different images. Therefore, users should have the flexibility to choose appropriate image quality that is acceptable for the particular image. Our proposed image quality and file size prediction model allows users to decide the optimal compression ratio that can generate an acceptable image quality. By letting users to choose the image quality before the image is generated, our system have saved energy otherwise spent on encrypting and transmitting the extra payload associated with a higher quality image.

Moreover, the perception of image quality is important too. In an image, there are usually areas that are more important than others. When compressing the image, it is necessary to keep the details in these important areas in order to maintain the overall perception of image quality. Our proposed BSP tree compression and encryption scheme offers such option by letting users to choose different pruning thresholds for different areas of the image. By choosing a higher threshold for the less important areas, while maintaining a lower threshold for the important areas, the impact on the perception of image quality is small. At the same time, energy is saved on encryption and transmission because the image file size is smaller.

7.4 Future Works

It would be worthwhile to investigate the following areas for future research:

1. **Improved Coding Techniques** – Due to time constraints, advanced coding techniques were not implemented in this thesis. Better coding techniques can improve the compression ratios without sacrificing image quality. In our experiments, the compression results for photographic images are worse than computer graphic images. The use of more advanced coding technique may help to improve that. From previous BSP tree related researches that make use of a more advanced coding technique, compression performance is very good even for

photographic images. We may be able to apply entropy coding techniques to enhance the compression of the colours and line coordinates. It is worthwhile to investigate the use of advanced coding techniques and their impact on energy and image quality.

2. **Generalized Model for Prediction of Image Quality and File size** – In this thesis, we have determined the parameters for the image quality and size prediction model empirically. These parameters are slightly different for different images. It would be desirable if these parameters can be determined automatically so that the model can be applied to all images.
3. **Deriving an Energy Prediction Metric** – Due to limited resources and time constraints, we have not ported our compression and encryption scheme to an actual embedded wireless hardware. It would be useful to obtain real energy measurements on such portable devices to analyze the energy required for each clock cycle in computation and the energy required to transmit each byte of data. From that, we can derive an energy model which helps users to predict the energy needed to compress and transmit an image at a given security level. This allows users to select an appropriate image quality and security level that is necessary for the image.
4. **Integration into an energy-aware pervasive environment** – With an energy prediction metric, our image compression and encryption scheme can be integrated into an energy-aware framework with monitors on battery status and location. At a low-risk environment (e.g. at home), a lower security level can be chosen to save encryption energy. At a high-risk environment, maximum security level needs to be chosen to protect the image. It would be desirable if the security level is a function of the location and the battery status. Therefore, it is worthwhile to investigate how our compression and encryption scheme can fit into such framework.

References

- [1] Thomas Neubauer, "UMTS - Universal Mobile Telecommunications System," <http://www.nt.tuwien.ac.at/mobile/projects/UMTS/en/>.
- [2] S. Naik, D. Wei, "Software Implementation Strategies for Power-Conscious Systems," ACM Mobile Networks and Applications (2001), vol. 6, pp. 291-305.
- [3] Eui-Young Chung, Luca Benini, Giovanni De Micheli, "Automatic Source Code Specialization for Energy Reduction," IEEE International Symposium on Low Power Electronics and Design (2001), 6-7 August 2001, pp. 80-83.
- [4] Sven Wuytack, Francky Catthoor, Hugo De Man, "Transforming Set Data Types to Power Optimal Data Structures," IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 15., No. 6, June 1996, pp. 619-629.
- [5] A. Sung, C. Gebotys, "Using LP Modeling for PDA Design," Technical Report, Department of Electrical and Computer Engineering, University of Waterloo, August 2004.
- [6] C.S. Ong, K. Nahrstedt and W. Yuan, "Quality of Protection for Mobile Multimedia Applications," Proceedings of IEEE International Conference on Multimedia and Expo (ICME 2003), Baltimore, MD, July 2003.
- [7] S. Sudirman and G. Qiu, "Colour Image Representation Using BSP Tree," International Conference on Colour in Graphics and Image Processing (CGIP 2000), 1-4 October 2000, Saint-Etienne, France.
- [8] W. B. Pennebaker and J. L. Mitchell, "JPEG Still Image Data Compression Standard," Van Nostrand Reinhold, New York, 1993.
- [9] C.W. Fu, T.T. Wong, W.S. Tong, C.K. Tang, and A.J. Hanson, "Binary-Space-Partitioned Images for Resolving Image-Based Visibility," IEEE Transactions on Visualization and Computer Graphics, vol. 10, No. 1, January-February 2004, pp. 58-70.
- [10] W. Li, K. Mueller, and A. Kaufman, "Empty Space Skipping and Occlusion Clipping for Texture-based Volume Rendering," IEEE Visualization 2003, 19-24 October 2003, pp. 317-324.
- [11] S.Y. Cho, Z. Chi and W.C. Siu, "Image Classification with Adaptive Processing of BSP Image Representation," Proceedings of IEEE 6th International Conference on Signal Processing, vol. 1, August 2002, pp. 925-928.
- [12] P. Salembier, "Binary Partition Tree as an Efficient Representation for Image Processing, Segmentation, and Information Retrieval," IEEE Transactions on Image Processing, vol. 9, No.4, April 2000, pp. 561-576.

- [13] D. Ivanov, E.Kuzmin and S. Burtsev, "Progressive Image Compression Using Binary Trees," Computer Graphics & Geometry (Internet journal), vol. 2, No. 2, 1999, <http://www.cgg.ru/>.
- [14] H. Radha, M. Vetterli, and R. Leoonardi, "Image Compression Using Binary Space Partitioning Trees," IEEE Transactions on Image Processing, vol. 5, No.12, December 1996, pp. 1610-1624.
- [15] H. Radha, M. Vetterli, and R. Leoonardi, "Fast Piecewise Constant Approximation of Images," SPIE Visual Communications and Image Processing 1991, vol. 1605, pp. 475-486.
- [16] Adobe Systems Incorporated, "TIFF (Tagged Image File Format Specification) Revision 6.0," June 3rd, 2002, <http://partners.adobe.com/asn/developer/pdfs/tn/TIFF6.pdf>.
- [17] PKWARE Incorporated, ".ZIP File Format Specification, version 6.2.0", April 26th, 2004, http://www.pkware.com/products/enterprise/white_papers/appnote.txt.
- [18] M. Nelson, "LZW Data Compression," Dr. Dobbs Journal, vol. 14, No. 10, October 1989, pp. 29-37, <http://www.dogma.net/markn/articles/lzw/lzw.htm>.
- [19] S. Chandra and C.S. Ellis, "JPEG Compression Metric as a Quality Aware Image Transcoding," Proceedings of the 2nd USENIX Symposium on Internet Technologies & Systems, Boulder, CO, October 1999.
- [20] C.N. Taylor, S. Dey, and D. Panigrahi, "Energy/Latency/Image Quality Tradeoffs in Enabling Mobile Multimedia Communication," in Software Radio - Technologies and Services (E. D Re, ed.), pp. 55-66, Springer Verlag, January 2001.
- [21] C.N. Taylor and S. Dey, "Adaptive Image Compression for Enabling Mobile Multimedia Communication", Proceedings of IEEE International Conference on Communication, Helsinki, June 2001.
- [22] J. M. Shapiro, "Embedded Image Coding using Zerotrees of Wavelet Coefficients." IEEE Transactions on Image Processing, vol. 41, No. 12, December 1993, pp. 3445-3462.
- [23] A. Said and W. A. Pearlman, "A New, Fast, and Efficient Image Codec based on Set Partitioning in Hierarchical Trees," IEEE Transactions on Circuits and Systems for Video Technology, vol. 6, No. 3, June 1996, pp. 243-250.
- [24] D.G. Lee and S. Dey, "Adaptive and Energy Efficient Wavelet Image Compression for Mobile Multimedia Data Services," Proceedings of International Conference on Communications, vol.4, pp. 2484-2490, New York, April 2002
- [25] D.G.Lee, D.Panigrahi, and S.Dey, "Network-Aware Image Data Shaping for Low-Latency and Energy-Efficient Data Services over the Palm Wireless Network", World Wireless Congress (3G Wireless), San Francisco, May 2003.

- [26] H. Wu and A. Abouzeid, "Energy Efficient Distributed JPEG2000 Image Compression in Multihop Wireless Networks", in Proceedings of the 4th Workshop on Applications and Services in Wireless Networks, Boston, MA, USA, 2004.
- [27] Y. Fisher, "Fractal Image Compression: Theory and Application," Springer-Verlag, 1995.
- [28] K. Barthel, J. Schuttemeyer, T. Voye, and P. Noll, "A new Image Coding Technique Unifying Fractal and Transform Coding," Proceedings of 1st IEEE International Conference on Image Processing, vol. 3, November 1994, pp. 112-116.
- [29] S. Lee and H. Aso, "An Alternating Binary-Tree Partitioning For Fractal Image Coding," International Workshop on Very Low Bitrate Video Coding, Kyoto, Japan, Oct 1999.
- [30] Federal Information Processing Standards Publication (FIPS), "Advanced Encryption Standard," Processing Standard Publication 197, November 26, 2001, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.
- [31] L. Tang, "Methods for Encrypting and Decrypting MPEG Video Data Efficiently," Proceedings of the 4th ACM International Conference on Multimedia, Boston, MA, 1996, pp. 219-229.
- [32] L. Qiao, K. Nahrstedt and M. Tam, "Is MPEG Encryption by Using Random List Instead of Zigzag Order Secure?" IEEE International Symposium on Consumer Electronics, Singapore, December 1997.
- [33] T. Kunkelmann and R. Reinema, "A Scalable Security Architecture for Multimedia Communication Standards," Proceedings of 4th IEEE International Conference on Multimedia Computing and Systems, Ottawa, Canada, 1997.
- [34] C.P. Wu and C.C.J. Kuo, "Fast Encryption Methods for Audiovisual Data Confidentiality," Proceedings of SPIE International Symposium on Information Technologies 2000, Boston, MA, vol. 4209, November 2000, pp. 284-295.
- [35] C.P. Wu and C.C.J. Kuo, "Efficient Multimedia Encryption via Entropy Codec Design," Proceedings of SPIE International Symposium on Electronic Imaging 2001, San Jose, CA, vol. 4314, January 2001.
- [36] C. Kailasanathan and R. Safavi Naini, "Compression Performance of JPEG Encryption Scheme," IEEE International Conference on Digital Signal Processing, 2002.
- [37] L. Vorwerk, T. Engel and C. Meinel, "A Proposal for a Combination of Compression and Encryption," Proceedings of SPIE Visual Communications and Image Processing 2000, Perth, Australia, pp. 694-702.
- [38] Y.H. Seo, D.W. Kim, J.S. Yoo, S. Dey, and A. Agrawal, "Wavelet Domain Image Encryption by Subband Selection and Data Bit Selection", World Wide Congress (3G Wireless), San Francisco, May 2003.

- [39] H. Cheng, "Partial Encryption for Image and Video Communication," Master's thesis, University of Alberta, 1998.
- [40] H. Cheng and X. Li, "Partial Encryption of Compressed Images and Videos," IEEE Transactions on Signal Processing, vol. 48, No. 8, August 2000, pp. 2439-2451.
- [41] R.J. Clarke, "Digital Compression of Still Images and Video," Academic Press Inc., London, Great Britain, 1995.
- [42] H.R. Rabiee, R.L. Kashyap, and S.R. Safavian, "Multiresolution Image Compression with BSP Trees and Multilevel BTC," IEEE 2nd International Conference on Image Processing (ICIP), Washington D.C., October 1995.
- [43] H.R. Rabiee, R.L. Kashyap, and S.R. Safavian, "Multiresolution Segmentation Based Image Coding With Hierarchical Data Structures," IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 1996, Atlanta, GA, May 1996.
- [44] P. Chou, T. Lookabaugh and R. Gray, "Optimal Pruning with Applications to Tree-Structured Source Coding and Modeling," IEEE Transactions on Information Theory, vol. 35, No.2, March 1989, pp. 299-315.
- [45] University of Waterloo, Waterloo BragZone and Fractals Repository,
<http://links.uwaterloo.ca/bragzone.base.html>
- [46] B.R. Preiss, "Data Structures and Algorithms: with Object-Oriented Design Patterns in C++," 1st Edition, John Wiley & Sons Ltd, 1999.
- [47] Hans van Vliet, "Software Engineering: Principles and Practice," 2nd Edition, John Wiley & Sons Ltd, Chichester, Great Britain, 2000.