Simulation and Optimization of JPEG and Motion JPEG

Christos Bohoris

MSc Telematics (Telecommunications and Computer Engineering)

1998

Table of Contents

Abstract

The aim of this project was to examine the optimization possibilities of the standard JPEG code and to extend the capabilities of JPEG by adding a Motion JPEG coder. In the beginning of the project the possibility of optimization of the JPEG code, as written in C, was examined along with a thorough search for algorithms that could perform specific, time consuming tasks faster. Along with the study of the JPEG code, detailed results were obtained on the performance of each function in the algorithm for both encoding and decoding parts. From the study of the currently used algorithms, the performance results and the search for newer, more efficient algorithms, JPEG was found to perform quite well. The focus of the project was then directed in the implementation of the Motion JPEG addition. In the first steps of dealing with this task, various video coding standards such as H.263 and MPEG-2 had to be carefully studied. A complete motion JPEG coder consisting of a video encoder and decoder was finally implemented. The results were tested using a sequence of frames taken as parameters of the program, in line with the standard JPEG input structure.

Chapter 1 - History of JPEG Compression

1.1 Introduction to JPEG

The main problem for many applications of digital images is the huge amount of data required to represent an image directly. A digitized version of a single color picture at normal TV resolution typically contains about 1 million bytes. Such an image needs to be compressed for storage or transmission. The actual compression ratio varies from 100:1 to 2:1 depending on the specific application and the encoder/decoder complexity. State of the art techniques can compress images by a factor of 10 to 50 without significantly affecting the image quality. For the applications of storage or transmission on the bandwidth- limited channels which are widespread in today’s market, a standard image compression method is required to get more efficient use of media or channels and to enable interoperability of equipment developed by different manufacturers.

JPEG has recently been recognized as the most popular and efficient coding technique for continuous-tone still images. It can usually yield a satisfactory solution to most of the practical coding problems. The technique provides a flexible and comprehensive encoding framework that could open a broad range of still image applications. To meet the requirements of widely different applications JPEG defines four encoding modes:

1. Sequential Encoding: The image is encoded in a raster scan fashion from left to right and top to bottom.

2. Progressive Encoding: The image is encoded in multiple scans at the same spatial resolution.
3. Lossless Encoding: The image is encoded to guarantee exact recovery of every source sample value.
4. Hierarchical Encoding: The image is encoded at multiple spatial resolutions.

The whole implementation is standardized and internationally accepted, being the product of collaboration between many countries and standardization bodies, after many years of research.

1.2 Formation of the JPEG group

In 1982 researchers initiated activity in ISO on a color image compression standard. This resulted in the formation of a photographic experts group under ISO/TC97/SC2 Working Group 8, responsible for the development of a progressive data compression scheme that would be able to operate at the ISDN rate of 64Kbits/sec. In 1986 they were joined by the CCITT Study Group VIII. At that time, the collaboration between ISO and CCITT was actively encouraged in order to avoid the creation of competing, independently developed standards. The Joint Photographic Experts Group (JPEG) was therefore established for the purpose of developing a still image compression standard that would meet the needs of many different applications.

1.3 Work of the JPEG committee

The JPEG image compression standard was intended to cover the widest range of applications consistent with a number of predefined requirements. It should have the capability for sequential, progressive, lossy and lossless coding. The approach that JPEG would follow was selected after a number of competitive contests. Twelve proposals following widely different approaches were registered in March 1987, as candidates for the compression method: “Generalized Block Truncation Coding”, “Progressive Coding Scheme”, “Adaptive DCT”, “Component VQ”, “Quadtree Extension of Block Truncation Coding”, “Adaptive Discrete Cosine Transform”, “Progressive Recursive Binary Nesting”, “Adapting Transform and Differential Entropy Coding”, DCT with Low Block-to-Block Distortion”, “Block List Transform Coding of Images”, “HPC” and “DPCM using Adaptive Binary Arithmetic Coding”.

By June 1987, 10 proposals were supported with complete documentation and executable code. After extensive testing, the field was narrowed to three techniques. The Adaptive Discrete Cosine Transform proposed by the ESPRIT PICA group had the best image quality at high compression rates of even 0.75 bit/pixel. The proposal based on DPCM (Differential Pulse Code Modulation) with adaptive arithmetic coding had exceptional images at 0.25 bit/pixel. Finally, the progressive block truncation technique, proposed by the Japanese, did well with most requirements while having competitive image quality. The three methods were further refined and enhanced for further testing. At the January 1998 meeting in Copenhagen, extensive evaluation was done to compare the three methods. From the three finalists, the Adaptive Discrete Cosine Transform was selected because of the superior image quality and its demonstrated feasibility in both hardware and software forms.

1.4 JPEG Requirements

The goal of the JPEG group has been to develop an image compression technique that meets a number of diverse requirements, the most significant of which are the following:

- Provide state of the art image compression.
- Allow users to easily adjust the desired compression and image quality.

- The method should not be restricted by the image type. It should work independently of the image characteristics such as content, color space, dimensions and pixel resolution.
- It should have a modest amount of computational complexity that will easily allow both software and hardware implementations.
- Allow hierarchical encoding so that a low-resolution version of the image can be accessed without the need to decompress the image at full resolution.
- Allow both sequential and progressive coding.

All these requirements are fairly strict and required many years of design work before JPEG could actually guarantee them.

1.5 The Finalized JPEG Standard

Later in 1988, a Transform Technique Enhancement Group was formed within JPEG for the purpose of considering refinements and enhancements. The work continued till November 1994 when the CD for ISO/IEC 10918-3 was finalized. The whole project was successful, all the initial requirements were met and JPEG was finally able to deliver impressive results. Possible bit rates and qualities can be estimated as seen in the following Table (Table 1.1):

Compression rate (bit/pixel)	Quality
0.25 – 0.5	Moderate to good quality.
0.50 – 0.75	Good to very good quality.
0.75 – 1.5	Excellent images.
1.50 – 2.0	Indistinguishable images.

Table 1.1 – Possible Compression Rates and Quality.

Today JPEG is widely used in many different applications. JPEG chips are available for a low price and many manufacturers also include with it video-compression standards, such as H.261, MPEG and Motion JPEG. Typically, a 10MHz chip can compress a full-page 24-bit color, 300-dpi image from 25MB to 1MB in about 1 second, and the processing time continues to decrease rapidly. The fast implementation makes JPEG easily applicable to applications like color facsimile, high-quality newspaper wire photos, desktop publishing, medical imaging, imaging scanners, electronic digital cameras, and so forth.

Equally important is the fact that JPEG gave researchers the experience needed to move on to implementing video compression standards such as MPEG and H.263. The low manufacturing costs and speed of JPEG chips has also led many manufacturers to the development of an additional mode of operation for video sequences, called Motion JPEG. In Motion JPEG, each frame of a video stream is compressed independently using the JPEG algorithm. Real-time compression and decompression is possible using low cost video boards along with a JPEG chip. Motion JPEG plays a significant role for tasks like video editing but lucks audio support as well as the high levels of compression given by other standards like MPEG. A large variety of Motion JPEG implementations can be found today but due to luck of standardization most of them are vendor dependent.

Chapter 2 - JPEG and Image Compression

2.1 Overview of JPEG

JPEG is one of the most popular image compression mechanisms. It stands for Joint Photographic Experts Group (JPEG), the original name of the committee that wrote the standard. JPEG was designed to compress true-color or gray-scale images of natural, real-world scenes. It works well on photographs but not so well on letters or line drawings. JPEG is 'lossy', meaning that the compressed image is not identical to the original. It achieves excellent compression ratios by exploiting weaknesses in human visual perception, which is not capable of assimilating all of the information contained in a pixel image. Although JPEG was developed for compressing still photographs, if these can be decompressed at a sufficient rate then it can be used for digital video. This so-called 'motion JPEG' (M-JPEG) is an integral part to several digital video systems such as Apple's QuickTime.

2.2 JPEG Still Image Compression

The algorithms used in a compression process are known as codecs. The JPEG codec compresses images by removing some of the information contained in the pixels of an image. To achieve this, JPEG exploits known limitations of the human eye, notably the fact that small color changes are perceived less accurately than small changes in brightness. Thus, JPEG is intended for compressing images that will be looked at by humans. For applications where an image must be scanned and analyzed by a machine, the losses introduced by JPEG compression may be unacceptable. One example is photographic Astronomy where photographs of distant objects are analyzed and the smallest information contained in the image pixels may be important. A useful property of JPEG is that the degree of loss can be varied by adjusting compression parameters. This means that the user can tradeoff file size against output image quality. High compression means low image quality while low compression means excellent image quality. One can make extremely small files if the poor quality is acceptable in applications such as indexing of image archives. In most cases what users do is try to find the highest compression that still gives enough quality to satisfy their needs. Another important aspect of JPEG is that decoders can trade off decoding speed against image quality, by using fast but inaccurate approximations to the required calculations. Some viewers obtain remarkable speedups in this way. Encoders can also trade accuracy for speed, but there's usually less reason to make such a sacrifice when writing a file.

While most image file formats use an RGB (red, green, blue) value to describe each pixel value, the JPEG format converts this data to luminance (brightness) and chrominance (hue). This allows for separate compression of these two factors. Since the luminance is more important to our senses than the chrominance, the algorithm retains more of the luminance in the compressed file. The JPEG compression algorithm works on individual blocks of 8 x 8 pixels. It calculates a Discrete Cosine Transform (DCT) for the entire block, quantizes the DCT coefficients, and then applies a Variable Length Code compression scheme to the coefficients. It's in the quantization step where the loss of color information occurs. The DCT is the reason why JPEG doesn't do so well on sharp edges. The DCT tries to represent the image as a sum of mathematical curves. That works great on relatively smooth images but not so well on sharp jumps.

2.3 JPEG and the other Standards

For any standard to become popular it must offer significant advantages over the other standards of its kind. Here are some of the most important image compression formats available today:

Graphics Interchange Format (GIF)

The Graphics Interchange format (GIF) is a trademark of CompuServe Corporation who developed it for the efficient storage and transfer of image data. It is a lossless format that can store 8 bit color images, supporting transparency and multiple images in the same file. GIF works best with limited color drawings and not with photographs, JPEG's main strength. However when it comes to drawings the JPEG algorithm can have problems mainly with pictures containing sharp corners and uniform colors.

Tagged Interchange File Format (TIFF)

One popular image compression format is the Tagged Interchange File Format (TIFF). It was developed by Adobe Systems and Microsoft Corporation for the interchange of graphics and imaging data between software programs. A compressed TIFF file contains data encoded using the Huffman algorithm.

Portable Network Graphics (PNG)

A new, lossless image compression format with growing popularity and support is the Portable Network Graphics (PNG) format. PNG uses a deflate compression method (an LZ77 derivative) also used in the zip data compression algorithm. It can compress true color images with up to 48 bits per pixel, compared to JPEG's 24 bits. It also supports transparency and progressive display mode. It is designed to be simple, portable and flexible so that it can support future extensions.

I tested the above formats to see how well they perform compared to JPEG using a sample 24-bit bitmap which I converted to JPG, GIF, PNG and TIFF images. My sample results are as seen in the following table (Table 2.1).

Format	**BMP**	**JPG**	**GIF**	**PNG**	**TIFF**
Size (bytes)	2,359,350	271,068	406,374	537,869	754,158
Compression	None.	88%	83%	77%	68%
Quality	Excellent	Excellent	Good	Excellent	Excellent

Table 2.1 – Image Comparison Test.

JPEG gave the best compression result with the quality scale set to 95%. The quality of the image was excellent, better than the quality produced by the GIF format. The TIFF image had excellent quality but it produced the biggest output file. The PNG format produced an excellent quality image while still achieving significant compression. The following table (Table 2.2) lists the major characteristics of each format.

	JPEG	**GIF**	**TIFF**	**PNG**
Color Depth	24-bit	8-bit	24-bit	48-bit
Progressive Mode	Yes	Yes	No	Yes
Strength	Photographs	Drawings	Everything	Everything
Transparency	No	Yes	No	Yes

Table 2.2 – Image Characteristics Comparison.

2.4 Advantages of JPEG

Let's see some of the benefits we can achieve by compressing an image to JPEG format.

Save hard disk space in modern computer systems.

In our days it is fairly easy for anyone to own a database of photographs, pictures or images. JPEG compression can significantly reduce the size of pictures so that the total size of image databases remains small.

Reduce image manipulation times.

Raw scanned image files are typically very large, making their processing or manipulation difficult through graphics applications. Obviously JPEG compression can efficiently solve this problem by keeping their size small.

Achieve quicker image transmission times.

A JPEG image file can be transmitted much quicker than its original. This means both time and cost savings, as transmissions can be expensive. If a communications device includes the JPEG code inside a DSP chip it can encode images to JPEG format before transmission. The images can be transmitted and then decoded back to the original image at the receiver. This of course means some loss of quality but again transmission costs are also very important.

JPEG compression is very popular.

Indeed today it is difficult to find a graphics application that does not support the JPEG format. The fast, graphics-oriented modern computers along with the continuously growing popularity of the Internet have helped the JPEG format to become very popular among both professionals and home users.

<u>2.5 JPEG Performance</u>

JPEG compression achieves excellent results with photographs or true color images. Figure 2.1 shows a bitmap (Left) of 59KB size and a JPEG image (Right) of 9.8KB size. As it can be seen the JPEG image retains the quality of the original image while in this case it is about 6 times smaller. The quality factor of the JPEG image was set to 75%, a number that is considered to give good performance for most images.

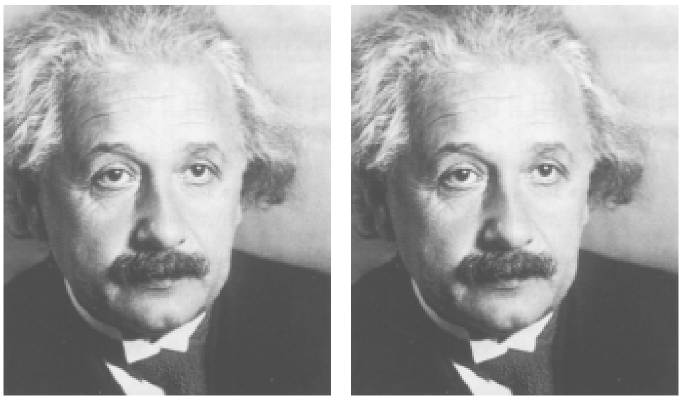

Figure 2.1 – A 56KB Bitmap (Left) compared to a 10KB JPEG Image (Right)

In reality the JPEG image is different from the original image. A magnified close-up of the previous picture as seen in Figure 2.2 shows differences between the original bitmap (Left) and the JPEG image (Right).

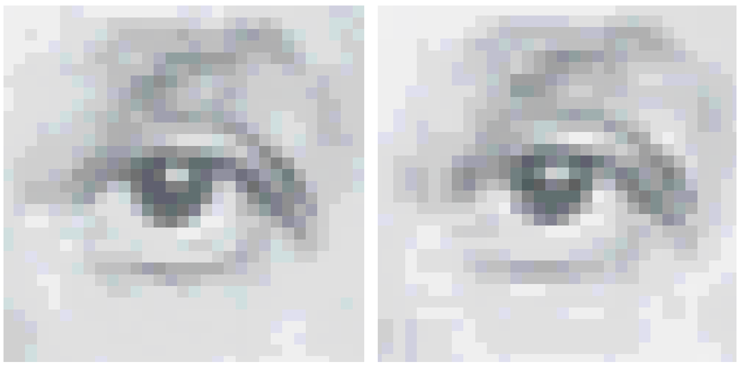

Figure 2.2 – A detail at 600% magnification of the photographs of figure 2.1. The differences between the original image (Left) and the JPEG image (Right) are now noticeable.

The more distinct differences between the two images appear in the parts that contain line edges and in the parts that contain a large area of uniform color. This reflects the weakness of JPEG in both these areas.

In another test performed, a 2.25MB bitmap was converted to various JPEG images of a variable quality factor. With quality set to 100% the size of the JPEG file was 763KB. For different values of the quality factor from 0% to 100% the following table (Table 2.3) was constructed, showing the corresponding file size (representing the output bit rate of the JPEG algorithm).

Quality (%)	**Size (KB)**
0	14.4
5	17.7
10	25.8
15	33.7
20	40.7
25	47.5
30	53.8
35	60
40	65
45	70.8
50	76.1
55	81.6
60	88.9
65	98.9
70	112
75	131
80	164
85	206
90	266
95	388
100	763

Table 2.3 – JPEG Quality and Image Size.

From table 2.3 the following graph (Figure 2.3) was created.

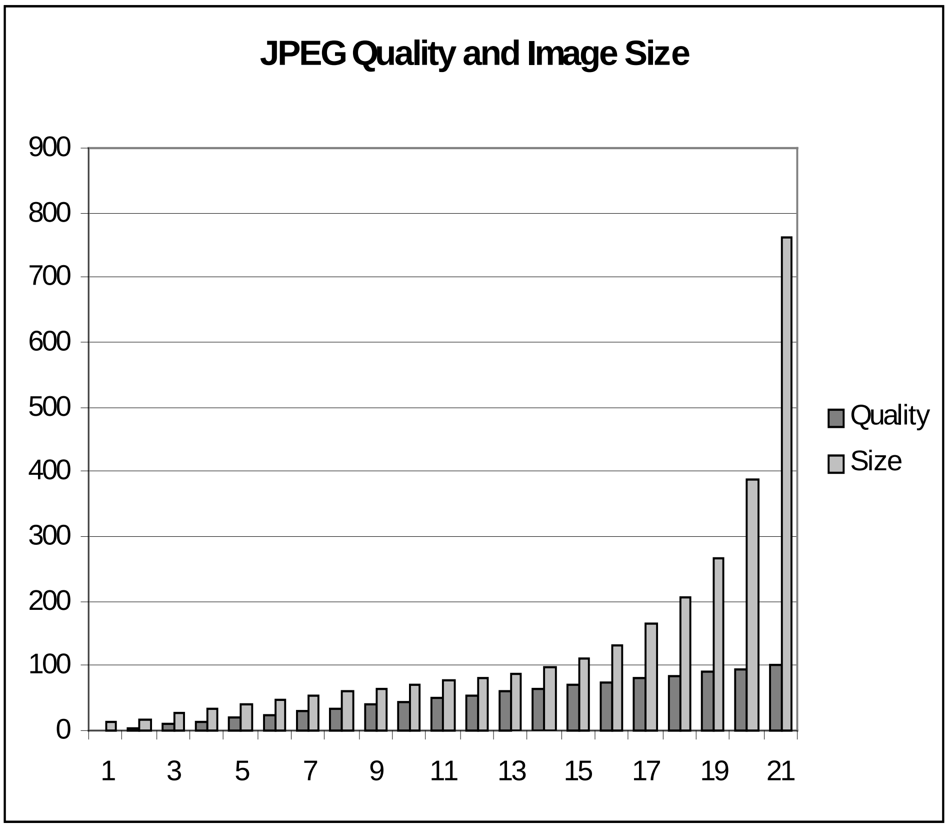

Figure 2.3 – Quality (%) and Image Size (KB).

From this graph it is seen that the size of the file dramatically reduces as the quality falls from 100% to around 85% while after that the decrease in size is smaller. The important conclusion obtained from this graph is that JPEG offers great file size reduction while the quality of the image is in high levels.

The performance of JPEG was also found to be very good compared to other popular image formats giving the best compression among all in my sample comparison test.

The JPEG format offers many advantages for users and most probably it will still be widely used in the future for both image and video compression purposes.

2.6 Motion JPEG and Video Compression

Uncompressed digital video can generate dozens of megabytes of data per second. That’s far more than any computer can cope with, so some method of compressing all that data is vital when working with digital video. There are several codecs in popular today, including Intel’s Indeo, MPEG and Motion JPEG (M-JPEG). Each compression format has its own advantages so that one’s choice depends on the type of application. Video editing requires the ability to work with individual frames something that can’t be done properly if much of the visual data for each frame has been lost during compression, as in the case of MPEG. Because of this, professional capture cards housed inside modern computer systems tend to use M-JPEG. M-JPEG is a multiple image version of JPEG. M-JPEG has no inter-frame compression so that a clip can be edited with single frame precision. Often, video files are captured and edited using M-JPEG, and then converted into a more compact format such as MPEG, when all the editing work has been done. Various vendors have applied M-JPEG compression to video sequences. Unfortunately, in the absence of any recognized M-JPEG standard, they've each done it differently. As a result these files are usually not compatible across different vendors.

Chapter 3 - The JPEG Algorithm

3.1 Color Space

Since Newton’s time it has been known that a wide spectrum of colors can be generated from a set of three primaries. These three primary colors are red, yellow, and blue and define the RGB (red, green, blue) color space. A 3-D representation of the RGB color space can be seen in the figure below (Figure 3.1).

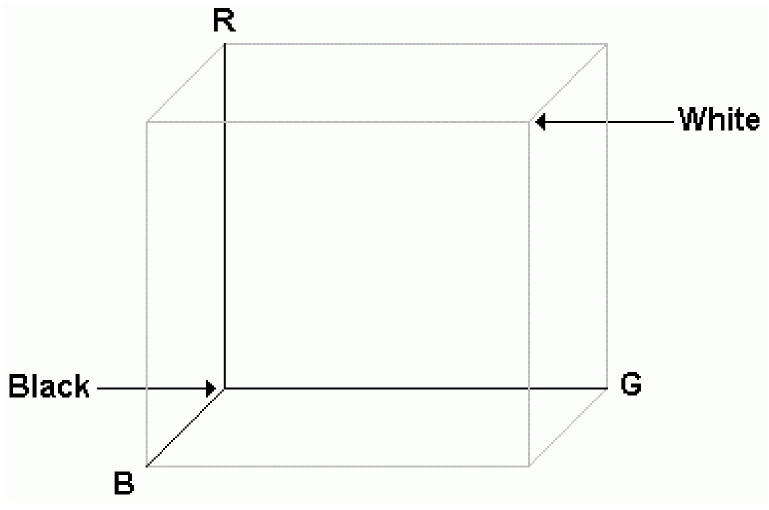

Figure 3.1 – The RGB Color Coordinate System.

In 1931 the Commission Internationale de L’ Eclairage (CIE) developed a television specification which introduced the concept of isolating the luminance (brightness) from the chrominance (hue). Based on the CIE specification the National Television System Committee (NTSC) in America, defined the transmission of signals in a luminance and chrominance format that formed a color space called YIQ. In Europe the PAL/SECAM television standards were defined that used a color space named YUV. The only difference of YUV and YIQ color spaces is a 33 degrees rotation in UV space. The JPEG format uses the YUV color space. YUV is related to RGB as seen in the following matrix,

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} \mathbf{0.299} & \mathbf{0.587} & \mathbf{0.114} \\ \mathbf{-0.148} & \mathbf{-0.289} & \mathbf{0.437} \\ \mathbf{0.615} & \mathbf{-0.515} & \mathbf{-0.100} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Alternatively,

$$Y = 0.299 (R - G) + G + 0.114 (B - G)$$
$$U = 0.493 (B - Y)$$
$$V = 0.877 (R - Y)$$

The following figure (Figure 3.2) illustrates the relationship between the YUV color space and the RGB color space.

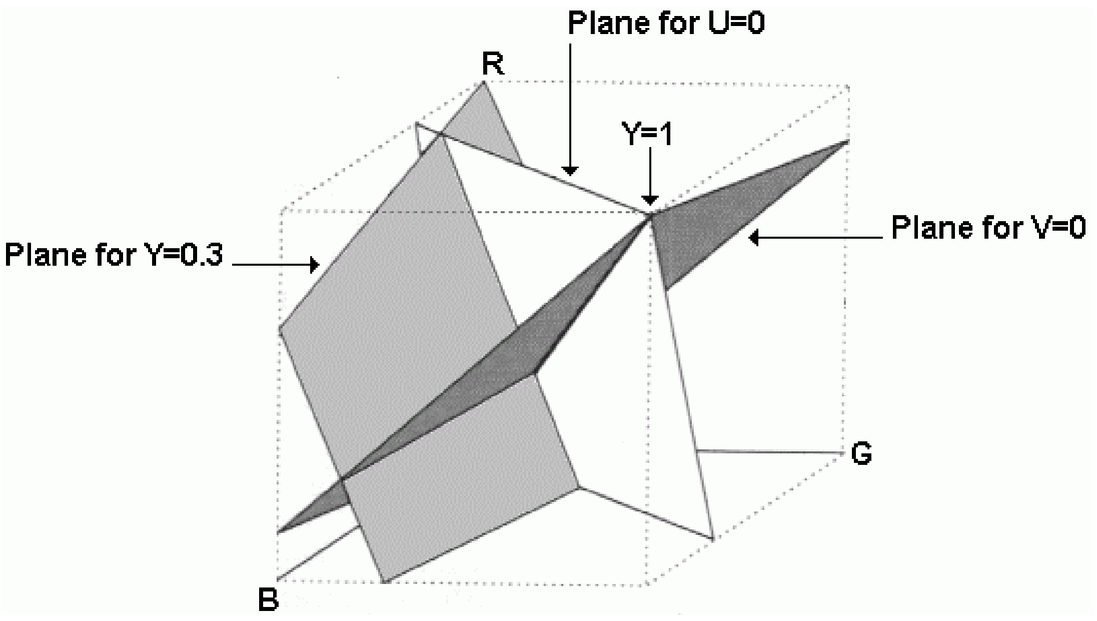

Figure 3.2 – Relationship between YUV and RGB.

For JPEG the YUV format has the advantage that concentrates more image information in the luminance and less in the chrominance. The result is that the YUV elements are less correlated and therefore, can be coded separately without much loss in efficiency. The conversion from RGB to YUV color space is usually the first step toward compressing an image.

3.2 The Baseline Model

In the following figure (Figure 3.3), we can easily see how the JPEG algorithm works both at the encoding (converting a raw image to JPEG format) and decoding phase (converting a JPEG image back to the original raw image). Note that for the decoding part the inverse steps are followed.

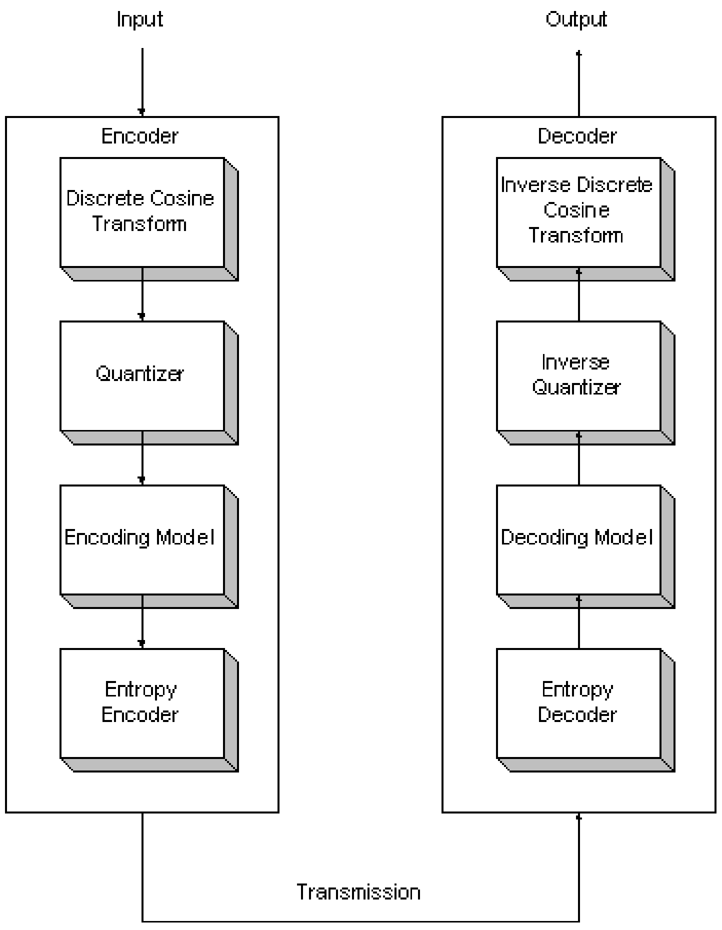

Figure 3.3 – The JPEG Block Diagram.

The JPEG baseline (ADCT) model consists of the following stages,

- Transformation Stage: Concentrates the information energy to the first few transform coefficients.
- Quantizer: Causes a controlled loss of information.
- Two coding Stages: Further compress the image data.

3.3 Transformation Stage

For each separate color component, the image is broken into 8 by 8 blocks of picture elements, which join end to end across the image. The transform method chosen by JPEG is the two-dimensional 8 by 8 DCT which can be obtained by performing a one dimensional DCT on the columns and a one dimensional DCT on the rows of the image. The choice of DCT in JPEG is motivated by the many benefits it offers:

- DCT is image independent. This is an important issue since an image dependent algorithm implies that additional computations need to be performed.
- DCT is an orthogonal transform. If in matrix form the DCT output is $Y = T X T^t$, then the inverse transform is $X = T^t Y T$ where the transformation from X to Y is simply the forward DCT.
- An important property of the 2-D DCT and IDCT transforms is separability. This means that the 2-D DCT can be obtained by first performing 1-D DCTs of the rows followed by 1-D DCTs of the columns. This property can be exploited to simplify the hardware requirements of a hardware design at the expense of slight increase in the overall operations count.
- DCT computations can be performed with a variety of algorithms. All these algorithms have different advantages and disadvantages mainly in what regards simplicity and speed of execution.

An explicit formula for the two dimensional 8 by 8 DCT can be written in terms of the pixel values f (i, j), and the frequency domain transform coefficients f (u, v) as,

$$F(u,v) = (1/4)C(u)C(v)\sum_{i=0}^{7}\sum_{j=0}^{7} f(i,j)\cos((2i+1)u\pi/16)\cos((2j+1)v\pi/16)$$

Where,

$$C(x) = \begin{Bmatrix} 1/\sqrt{2} & \text{x = 0} \\ 1 & \text{otherwise} \end{Bmatrix}$$

The inverse of the two-dimensional DCT is written as,

$$f(i,j) = (1/4)\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)F(u,v)\cos((2i+1)u\pi/16)\cos((2j+1)v\pi/16)$$

Let’s consider the example of an image consisting of two different colors. One of the colors is represented by an ‘O’ and the other by an ‘X’ as shown below

```
OOOXXOOO
OXXXXXXO
XXXXXXXX
XXXXXXXX
XXXXXXXX
XXXXXXXX
OXXXXXXO
OOOXXOOO
```

If we assign O = –10 and X= 10 (values –10 and 10 are selected to make the calculations easier) the image information looks like (Table 3.1),

–10	–10	–10	10	10	–10	–10	–10
–10	10	10	10	10	10	10	–10
10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10
–10	10	10	10	10	10	10	–10
–10	–10	–10	10	10	–10	–10	–10

Table 3.1 – DCT input values.

The frequency domain representation, with all values rounded to the nearest integer can be seen in the following table (Table 3.2),

40	0	–26	0	0	0	–11	0
0	0	0	0	0	0	0	0
–45	0	–24	0	8	0	–10	0
0	0	0	0	0	0	0	0
–20	0	0	0	20	0	0	0
0	0	0	0	0	0	0	0
–3	0	10	0	18	0	4	0
0	0	0	0	0	0	0	0

Table 3.2 – DCT output values.

Notice the zeroes produced in the above table. They actually represent the reduction of data caused by the transform. We now have many zero coefficients as well as a concentration of the energy around the upper left corner of the matrix. The energy distribution has now changed but the total energy still remains the same, because the DCT is a unitary transformation. Also since the DCT is unitary the maximum value of each 8 by 8 DCT coefficient is limited to a factor of eight times the original values. When the inverse DCT of the block is calculated and the values are rounded to the nearest integer we come back to the original image, although an exact reconstruction is not possible with an integer value DCT.

The straightforward calculation of DCT for a JPEG 8x8 block suggests that each coefficient needs 64 multiplies and 63 additions. However, this gives a misleading impression about complexity. Many cost-effective hardware and software implementations are already available for the simple DCT, implementing both the DCT alone and the DCT based JPEG compression. Through the years many fast DCT algorithms have been also implemented.

3.4 Quantization

The coefficients of the DCT are quantized so that their magnitude reduces and the number of zero value coefficients increases. The uniform midstep quantizer is used for the JPEG baseline method, where the stepsize is varied according to the coefficient location and the color component that is encoded. The quantizer transfer function can be seen in figure 3.4. Quantization is the lossy stage in the JPEG coding scheme. If we quantize too coarse, we may end up with images that look "blocky", but if we quantize too fine, we may spend useless bits coding noise. The quantization process is controlled by the Q-Factor, a number that is used to easily change the default quantization matrix. If the Q-Factor is 0, we bypass this step. A high value of the Q-Factor means high levels of compression, a low value of the Q-Factor means better image quality. For the quantization process each DCT value V is transformed as in the following equation,

$$V_Q = \text{round}(V / 2 \text{ x } Q)$$

As it can be seen, quantization reduces the accuracy of the DCT values with lower precision resulting to a lower bit rate in the compressed data stream. Due to this loss of precision any reconstructed DCT values will be approximations of the values taken before the quantization. It is very important though that the quantization factor Q is carefully chosen to avoid unwanted distortions.

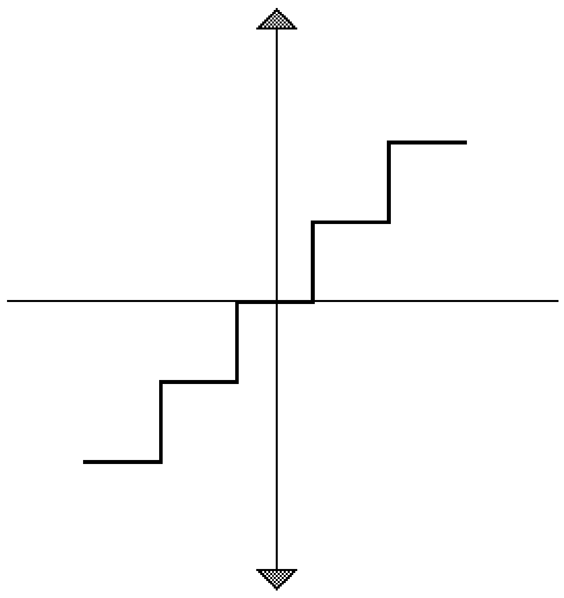

Figure 3.4 – The quantizer Transfer Function.

3.5 Coding Model

The purpose of the quantizer is to rearrange the quantized DCT coefficients into a zigzag pattern (Figure 3.5), with the lowest frequencies first and the highest frequencies last. The reason this is done is because the zigzag pattern can increase the run-length of zero coefficients found in the block. The assumption is that the lowest frequencies tend to have larger coefficients and the highest frequencies are, by the nature of most pictures, predominantly zero. The DC coefficients of the image often vary slightly between successive blocks. The coding of the DC coefficient exploits this property through Differential Pulse Code Modulation (DPCM). This technique codes the difference between the quantized DC coefficient of the block and the quantized DC coefficient of the previous block.

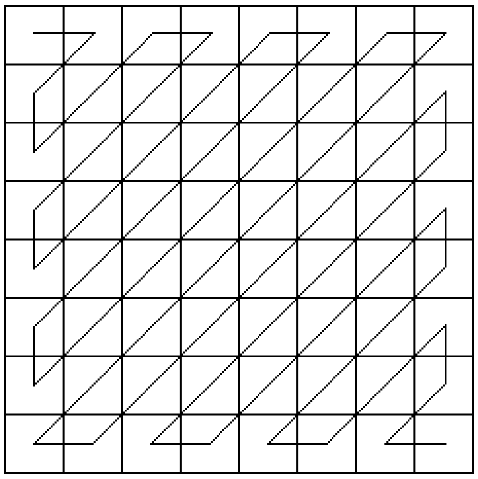

Figure 3.5 – The zigzag pattern followed for an 8x8 block.

After calculation of the DPCM code, the actual DC code is then given by the size of bits of the DPCM code followed by its significant bits. The quantized AC coefficients usually contain runs of consecutive zeroes. Therefore, a coding advantage can be obtained by using a run-length technique.

Run length coding accepts a series of input values and then checks the sequence for zeroes. Let’s consider an example sequence of number as given below,

0, 0, 0, 0, 5, 0, 0, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 2, 0, 0, 0, 6

This sequence contains 30 different values. Typically, because of the DCT and quantization process, many zeroes will appear just like in our example. Run length coding achieves further compression by transforming a sequence to a new smaller sequence which includes the number of zeroes appearing before a non-zero value along with the value itself. After run length our sequence becomes, 4-5, 2-3, 0-1, 12-4, 3-2, 3-6

There are 4 0s followed by a 5, 2 0s followed by a 3 and so on. This sequence contains only 12 values and thus compression was achieved.

3.6 Entropy Coding

The block codes from the DPCM and run-length models can be further compressed using entropy coding. For the baseline JPEG method, a Huffman coder is used to compress the data closer to symbol entropy. One reason for using a Huffman coder is that it is easily implemented in hardware. In order to compress data symbols, the Huffman coder creates shorter codes for frequently occurring symbols and longer codes for occasionally occurring symbols. Let’s consider the example of encoding an excerpt from Michael Jackson’s song ‘Bad’.

Because I’m bad, I’m bad come on
Bad, bad really, really bad
You know I’m bad, I’m bad
You know it
Bad, bad really, really bad
You know I’m bad, I’m bad
Bad, bad really, really bad

The first step in creating Huffman codes is to create a table assigning a frequency count to each phrase. In the above lyrics, ignoring capitalization, this is shown in the following table (Table 3.3).

Phrase	**Symbol**	**Frequency**
Because	B	1
I’m	I	6
Bad	B	15
Come on	C	2
It	I	1
Really	R	6
You know	Y	4

Table 3.3 – Huffman coding table.

Initially all symbols are designated as the leaf nodes of a tree. Starting from the two least weight nodes, the pair ‘Because’ and ‘It’ is aggregated into a new node. The node holds symbols ‘b’ and ‘i’, each occurring only once (1 + 1) so the name of the node is (bi 2). This process is repeated until the entire symbol set is represented by a single node as shown in the following figure (Figure 3.6),

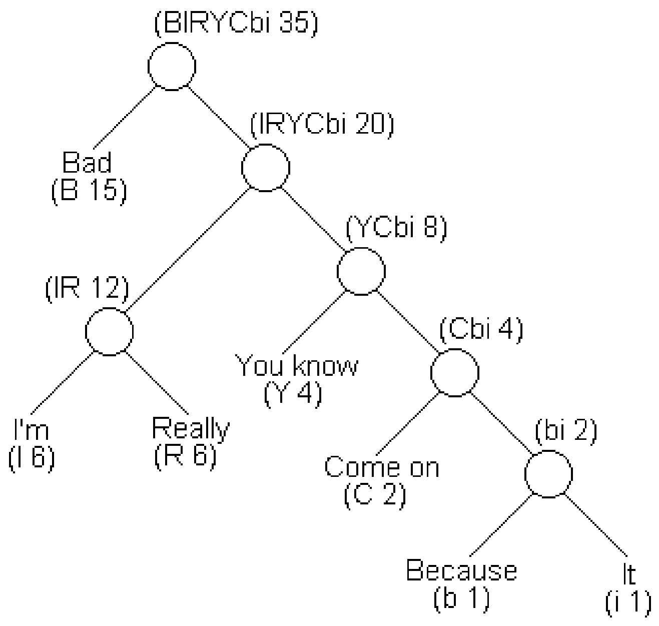

Figure 3.6 – Huffman Coding.

A Huffman code is generated for each symbol by attaching a binary digit to each branch. This digit is formed by assigning the binary digit 1 to the left branches and the binary digit 0 to the right branches. The code is generated by following the path of branches from the top node to the symbol leaf node. For example for the phrase ‘Come on’ we get the code 0001 as it is explained in the following table (Table 3.4).

Start Node	Direction	Stop Node	Digit
(BIRYCbi 35)	Left	(IRYCbi 20)	0
(IRYCbi 20)	Left	(YCbi 8)	0
(YCbi 8)	Left	(CBi 4)	0
(CBi 4)	Right	(C 2)	1

Table 3.4 – The Huffman technique.

A full table for each symbol is shown in the following table.

Phrase	Symbol	Frequency	Code Length	Code
Because	B	1	5	00001
I’m	I	6	3	011
Bad	B	15	1	1
Come on	C	2	4	0001
It	I	1	5	00000
Really	R	6	3	010
You know	Y	4	3	001

Table 3.5 – Table of Huffman symbols.

To encode each symbol we just have to output its code word to the bit stream. Our coding efficiency can be calculated by comparing the number of bits required to realize the lyrics. For the Huffman code of our example, the bit length is,

15(1) + (6+6+4) 3 + (1+1)5 = 81 bits.

In comparison, for a three-bit code, the bit length would be,

(1+6+15+2+1+6+4) 3 = 105 bits.

While for an ideal 7-symbol code, the bit length is,

35 $\log_2(7)$ = 98.3 bits.

As we can see the Huffman coder compresses the data by a factor of about 20 percent.

3.7 The Motion JPEG algorithm

The motion JPEG algorithm extends the standard JPEG idea to add the capability of coding sequences of video frames. Motion JPEG initially divides each frame into a number of macroblocks. Each macroblock represents a 16 by 16 array of luminance pixels and 2, 8 by 8 arrays of chrominance in the frame (Figure 3.7 – 3.8).

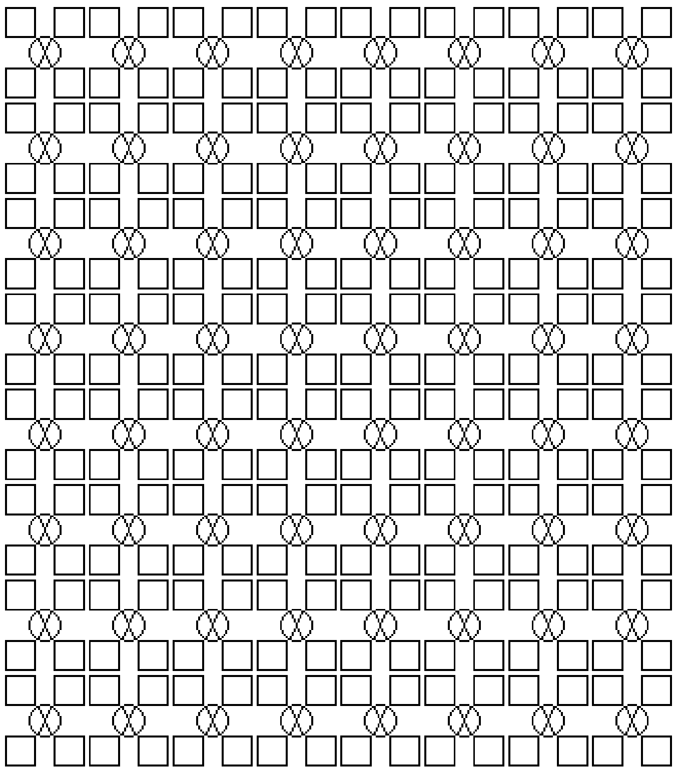

Figure 3.7 – Composition of a macroblock.

These macroblocks are the basic blocks used for detecting motion, encoding and producing output motion JPEG type frames. The most important parts of the algorithm are the motion estimation and the local, Motion JPEG encoder.

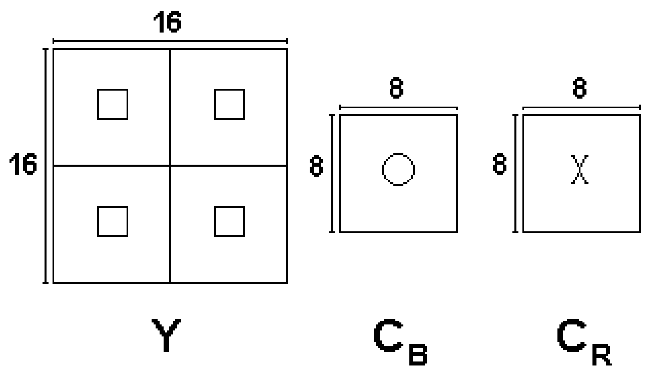

Figure 3.8 – A macroblock arrangement.

3.8 Motion Estimation

The purpose of motion estimation is to determine which of the macroblocks in a frame experienced motion. Motion estimation techniques fall into two categories:

1. Pixel by pixel motion estimation, called pixel-recursive algorithms (PRA).
2. Block by block motion estimation, called block-matching algorithms (BMA).

PRA techniques are giving the best results but they are rarely used because they are inherently complex, and the motion estimation algorithms sometimes run into convergence problems. As a compromise, BMA, even though not optimal, has been widely used.

In the case of a Motion JPEG coder, in order to discover motion, the current frame is compared with the previous frame. If there was some motion between the two then the frame is set to INTER mode. In the case that there was no motion detected the frame is set to INTRA. Obviously all the macroblocks of an INTRA frame are INTRA as well. An INTER frame can contain both INTER and INTRA macroblocks. Motion estimation decides in which macroblocks there was motion and sets them to INTER mode for further encoding and in which there was no motion detected that will be set to

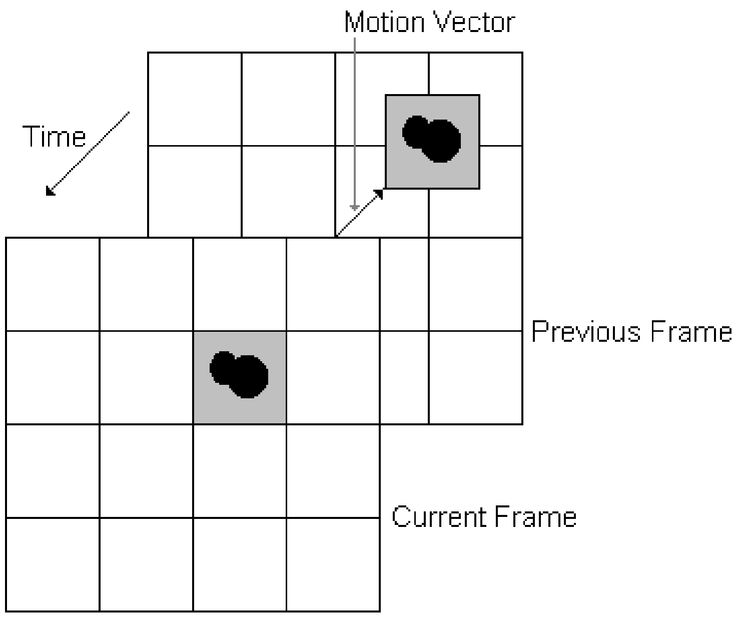

Figure 3.9 – Motion Estimation.

INTRA mode. For every INTER macroblock the value of the motion vector has to be determined. A motion vector is a way to measure the displacement of an object in a macroblock. The whole process can be seen in Figure 3.9.

Chapter 4 - Experimental Work

4.1 JPEG Optimization

4.1.1 JPEG Performance Analysis

The first aim of the project was to identify parts of the JPEG code that could be optimized. Code and algorithm optimization were both considered as ways of making code faster with a parallel concern on keeping code size small. The first thing that was examined was which parts of the JPEG algorithm are actually time-consuming as it is these functions that the optimization should focus on. To determine this, profiling of the code was performed using the standard "gprof" UNIX tool for both the encoding and the decoding parts of the algorithm. The first phase of profiling tested the encoding part of the algorithm, by encoding a 600x399 image using the command line JPEG interpreter. The 10 slowest functions encountered in the JPEG endoder are listed in table 4.1 while the complete profiling results and explanations can be seen in Appendix A. On the top of the list of the most time consuming functions is,

```
extern void ChenDct(int *, int *); /* Module: chendct.c */
```

This function is the Chen implementation of a forward discrete cosine transform (DCT).

% Time	**Cumulative Seconds**	**Self seconds**	**Calls**	**Self us/call**	**Total us/call**	**Function**
28.75	0.23	0.23	5700	40.35	40.35	ChenDct
16.25	0.51	0.13	5700	22.81	22.81	Quantize
10.00	0.59	0.08	45200	1.77	1.77	ReadXBuffer
6.25	0.64	0.05	5700	8.77	8.77	EncodeAC
5.00	0.68	0.04	58253	0.69	0.69	EncodeHuffman
3.75	0.71	0.03	45600	0.66	0.66	ReadXBound
3.75	0.74	0.03	5700	5.26	5.26	PreshiftDctMatrix
2.50	0.76	0.02	110801	0.18	0.18	meputv
1.25	0.77	0.01	5700	1.75	1.75	BoundDctMatrix
1.25	0.78	0.01	5700	1.75	21.05	ReadBlock

Table 4.1 – The 10 slowest functions of the JPEG Encoder.

This version of DCT was introduced in 1977 by Wen-Hsiung Chen, C. Harrison Smith and S. C. Fralick and it is known as Fast Discrete Cosine Transform (FDCT). The FDCT version of DCT uses matrices to obtain a 6 times better performance than the original DCT.

Another two ‘slow’ functions follow closely the one just described,

```
extern void Quantize(int *, int *); /* Module: transform.c
*/
static void ReadXBuffer(int, int *, BUFFER *); /* Module:
io.c */
```

Function ‘Quantize’ represents the quantization part of the algorithm as described in the previous chapter. It quantizes an input matrix and puts the results in an output matrix. ‘ReadXBuffer’ on the other hand fetches elements from the buffer structure into storage. This may actually amount to an arbitrary number of characters depending on the word size.

For the decoding phase, table 4.2 shows the profiling results for the 10 slowest functions encountered in the JPEG decoder, while a full listing can be seen in Appendix B.

% Time	**Cumulative Seconds**	**Self seconds**	**Calls**	**Self us/call**	**Total us/call**	**Function**
30.00	0.24	0.24	5700	42.11	42.11	ChenIDct
7.50	0.48	0.06	5700	10.53	10.53	IQuantize
6.25	0.53	0.05	45600	1.10	2.19	WriteXBound
6.25	0.58	0.05	45200	1.11	1.11	WriteXBuffer
5.00	0.62	0.04	58128	0.69	1.08	DecodeHuffman
5.00	0.66	0.04	5700	7.02	7.02	IZigzagMatrix
5.00	0.70	0.04	5700	7.02	24.56	WriteBlock
3.75	0.73	0.03	51609	0.58	0.72	megetv
2.50	0.75	0.02	46274	0.43	0.43	pgetc
2.50	0.77	0.02	5700	3.51	19.35	DecodeAC

Table 4.2 – The 10 slowest functions of the JPEG Decoder.

The most time consuming functions here are not surpassingly the inverse of the ones found in the encoding part.

```
extern void ChenIDct(int *, int *); /* Module: chendct.c */
extern void IQuantize(int *, int *); /* Module: transform.c
*/
static void WriteXBound(int, int *, BUFFER *); /* Module:
io.c */
```

Function ChenIDct implements the Chen (FDCT) inverse DCT. IQuantize takes an input matrix, performs inverse quantization to it and puts the results in an output matrix. WriteXBound writes an integer array input to the buffer. It also checks to see whether the bounds of the image width are exceeded, and if so, the excess information is ignored.

After the examination of both the JPEG code and search for available algorithms that could bring further performance gains the above functions were found to have reached very good levels of optimization, already. At this point the focus of the project was turned on the implementation of a Motion JPEG module able of compressing motion pictures, as an addition to the existing JPEG code.

4.2 The Motion JPEG Encoder

4.2.1 Addition of a Motion JPEG Encoder

In the beginning of the design, the standard JPEG code was modified to accept a new parameter 'm' that indicates Motion JPEG mode of operation. In the main function of the JPEG code a new parameter 'm' was added for this purpose. A parameter 'e' indicates encoding motion JPEG mode where a series of raw frames can be encoded into Motion JPEG type files. The command line format is as given below,

```
jpeg –me –iw Width –ih Height –in NumberOfFrames FrameList
```

After the ‘me’ parameter, indicating the Motion JPEG encoding mode the width and height of the frames in the sequence has to be given. This is followed by the number of frames in the sequence and the filenames of the frames used. An example of a typical command line used is given below,

```
jpeg –me –iw 48 –ih 64 –in 10 frame1.raw frame2.raw frame3.raw
frame4.raw frame5.raw frame6.raw frame7.raw frame8.raw frame9.raw
frame10.raw
```

This command line will encode the 10 raw images, 48 pixels wide and 64 pixels in height into Motion JPEG type frames. The standard JPEG command line follows a very strict approach on its command line input. At this stage many problems had to be overcomed in order to guarantee the correctness of the input. The next step is to process the frames in the command line one by one, starting by defining the macroblocks in the current frame. The 10 frames of the ‘falling ball’ sequence, used for the whole coder can be seen in the following figure (Figure 4.1).

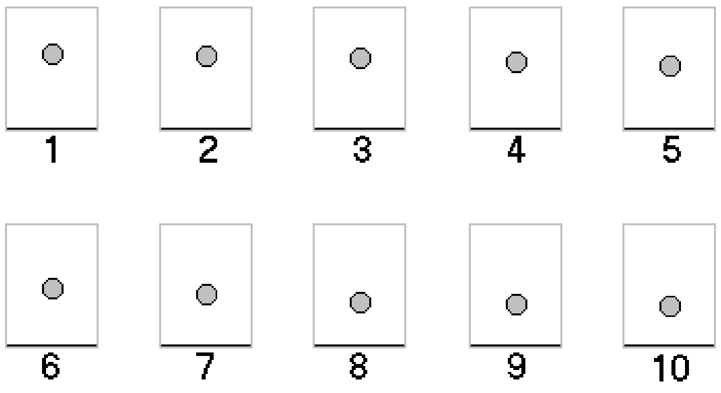

Figure 4.1 – The ‘falling ball’ sequence.

4.2.2 Setting the Macroblocks

Each macroblock is taken as an array of 16 by 16 pixels of luminance values. The number of macroblocks that can be fit in the full width and height of the frame is first calculated. After that the initial frame coordinates of each macroblock are calculated and the values of luminance in each of them are assigned. For the ‘falling ball’ sequence the macroblocks are assigned as seen in the following figure (Figure 4.2).

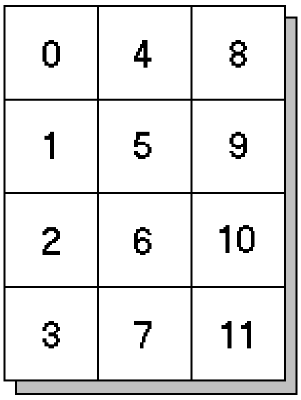

Figure 4.2 – The assigned Macroblocks.

In macroblock 5 of frame 1 that contains the luminance values representing the ball the following macroblock values are contained (Figure 4.3)

```
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255   0   0   0   0   0 255 255 255 255 255
255 255 255 255 255   0 192 192 192 192 192   0 255 255 255 255
255 255 255 255   0 192 192 192 192 192 192 192   0 255 255 255
255 255 255   0 192 192 192 192 192 192 192 192 192   0 255 255
255 255 255   0 192 192 192 192 192 192 192 192 192   0 255 255
255 255 255   0 192 192 192 192 192 192 192 192 192   0 255 255
255 255 255   0 192 192 192 192 192 192 192 192 192   0 255 255
255 255 255   0 192 192 192 192 192 192 192 192 192   0 255 255
255 255 255 255   0 192 192 192 192 192 192 192   0 255 255 255
255 255 255 255 255   0 192 192 192 192 192   0 255 255 255 255
255 255 255 255 255 255   0   0   0   0   0 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
```

Figure 4.3 – The Luminance values in macroblock 5 of frame 1.

The luminance values have a range from 0 to 255 with zero representing black and 255 representing white pixels. All other values represent the different colors in between them.

The top-left and bottom-right frame coordinates assigned for the macroblocks of each frame can be seen in Table 4.3.

Macroblock	**Top-Left**	**Bottom-Right**
0	(0, 0)	(15, 15)
1	(0, 16)	(15, 31)
2	(0, 32)	(15, 47)
3	(0, 48)	(15, 63)
4	(16, 0)	(31, 15)
5	(16, 16)	(31, 31)
6	(16, 32)	(31, 47)
7	(16, 48)	(31, 63)
8	(32, 0)	(47, 15)
9	(32, 16)	(47, 31)
10	(32, 32)	(47, 47)
11	(32, 48)	(47, 63)

Table 4.3 – Starting and Ending frame Coordinates of Macroblocks.

These macroblocks are the basic elements of the whole design and they are used extensively throughout the coder. The next step in this stage is setting the search windows.

4.2.3 Setting the Search Windows

A fundamental task of any video coder design is to detect motion between subsequent frames. In the Motion JPEG coder implemented this is done on the basis of macroblocks. For every macroblock in the current frame an area surrounding its occurrence in the previous frame has to be searched for motion. This area is called a search window. In the Motion JPEG coder designed the search windows cover an area of a maximum of 48 x 48 pixels (Figure 4.4).

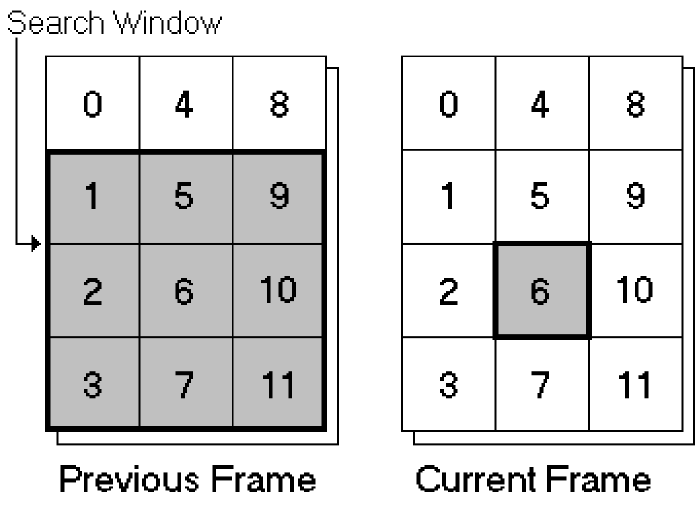

Figure 4.4 – The Search Window of Macroblock 6.

A search window actually covers the area including the occurrence of the macroblock in the previous frame and all its surrounding macroblocks. The frame coordinates of the search window set for each macroblock are shown in the following table (Table 4.4).

Search Window	**Top-Left**	**Bottom-Right**
0	(0, 0)	(31, 31)
1	(0, 0)	(31, 47)
2	(0, 16)	(31, 63)
3	(0, 32)	(31, 63)
4	(0, 0)	(47, 31)
5	(0, 0)	(47, 47)
6	(0, 16)	(47, 63)
7	(0, 32)	(47, 63)
8	(16, 0)	(47, 31)
9	(16, 0)	(47, 47)
10	(16, 16)	(47, 63)
11	(16, 32)	(47, 63)

Table 4.4 – The Search Windows Coordinates in a frame.

Now that the macroblocks are assigned and the search window for each macroblock is set the next step is performing motion estimation.

4.2.4 Motion Estimation

Motion estimation takes each macroblock of the current raw frame and searches through its search window in the previous raw frame in order to detect motion. If motion is detected in a macroblock then the macroblock is set to INTER mode, otherwise it is INTRA. If a frame contains at least one INTER macroblock then the frame is INTER, otherwise it is set to INTRA mode. In order to make an INTER or INTRA macroblock decision a number of parameters have to be calculated first.

The mean value of a macroblock is defined as,

$$MBmean = \frac{\left(\sum_{i=1,j=1}^{16,16} current\right)}{256}$$

From this equation we can see that the mean value of a macroblock is the sum of all its contained values divided by the total number of values in the macroblock. Having calculated the mean value, the parameter A can be calculated as,

$$A = \sum_{i=1,j=1}^{16,16} \left|current - MBmean\right|$$

The motion estimation algorithm uses a shifting macroblock, which is placed in all possible positions inside the search window. From the values of the current macroblock and the shifting macroblock the Sum of Absolute Distortions (SAD) is calculated. SAD is defined as,

$$SAD = \sum_{i=1,j=1}^{16,16} \left|current - shift\right|$$

The value obtained from this equation is further reduced by 100 and SAD becomes,

$$SAD = SAD - 100$$

The algorithm scans the search window through all possible shifting macroblocks and keeps the minimum value of SAD found. INTRA mode is finally choosen if,

$$A < (SAD_{\min imum} - 500)$$

If a macroblock is found to be INTER its motion vector is calculated as the difference of the current macroblock top-left frame coordinates and the top-left frame coordinates of the best matching macroblock. For the frame sequence used the algorithm produced the following results (Table 4.5),

Current Frame	**INTER Macroblock**	**Motion Vector**
Frame1.raw	NONE	NONE
Frame2.raw	5	(0, 1)
Frame3.raw	5	(0, 1)
Frame4.raw	5	(0, 2)
	6	(0, 2)
Frame5.raw	5	(0, 3)
	6	(0, 3)
Frame6.raw	5	(0, 3)
	6	(0,3)
Frame7.raw	6	(0, 3)
Frame8.raw	6	(0, 4)
Frame9.raw	6	(0, 1)
Frame10.raw	6	(0, 1)

Table 4.5 – Motion Vector Results.

The above results accurately describe the falling of the ball in each frame of the sequence.

The next stages of the encoding process involve the encoding of the INTER macroblocks found.

4.2.5 The Encoding Process

The first step of the encoding process is to construct the residual matrix. The residual matrix for an INTER macroblock is defined as,

$$\begin{bmatrix} c(0,0)-bm(0,0) & c(1,0)-bm(1,0) & \ldots & c(15,0)-bm(15,0) \\ c(0,1)-bm(0,1) & \ldots & \ldots & c(15,1)-bm(15,1) \\ \vdots & & & \vdots \\ c(0,15)-bm(0,15) & \ldots & \ldots & c(15,15)-bm(15,15) \end{bmatrix}$$

In this matrix c(x, y) represents a value in the current INTER macroblock and bm(x, y) represents a value in the best matching macroblock. Each element of the residual matrix is the difference between the current macroblock value and the best matching macroblock in the previous frame for the corresponding position. The values contained in the residual matrix are processed through the Discrete Cosine Transform (DCT). The DCT algorithm used in the Motion JPEG encoder uses the following equation,

$$DCTMatrix(u,v) = (1/4)C(u)C(v)\sum_{i=0}^{15}\sum_{j=0}^{15} RMatrix(i,j)\cos((2i+1)u\pi/16)\cos((2j+1)v\pi/16)$$

Where,

$$C(x) = \begin{Bmatrix} 1/\sqrt{2} & \text{x} = 0 \\ 1 & \text{otherwise} \end{Bmatrix}$$

The DCT Matrix obtained passes through the quantization process. Each value in the DCT matrix is divided by (2 * Q) where Q is the Quantization Factor. A small value of Q allows better output quality to be obtained while a large value of Q will finally give higher compression. Quantization is the irreversibly lossy part of the encoding process. For the Motion JPEG encoder designed a value of 5 was assigned to Q, favouring better quality. Figure 4.5 shows some typical results. In that figure we should notice the appearance of many zeroes along with the concentration of the information energy away from the center of the macroblock. The 2-dimensional quantization matrix is stored into a 1-dimensional array using the zig-zag technique. The zig-zag arranged values are now ready for the run-length algorithm.

3	4	4	3	3	2	1	0	0	0	−1	−2	−3	−3	−4	−4
4	6	5	5	4	3	2	1	0	−1	−2	−3	−4	−5	−5	−6
4	5	5	4	4	3	2	1	0	−1	−2	−3	−4	−4	−5	−5
3	5	4	4	3	2	2	1	0	−1	−2	−2	−3	−4	−4	−5
3	4	4	3	3	2	1	0	0	0	−1	−2	−3	−3	−4	−4
2	3	3	2	2	1	1	0	0	0	−1	−1	−2	−2	−3	−3
1	2	2	2	1	1	0	0	0	0	0	−1	−1	−2	−2	−2
0	1	1	1	0	0	0	0	0	0	0	0	0	−1	−1	−1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	−1	−1	−1	0	0	0	0	0	0	0	0	0	1	1	1
−1	−2	−2	−2	−1	−1	0	0	0	0	0	1	1	2	2	2
−2	−3	−3	−2	−2	−1	−1	0	0	0	1	1	2	2	3	3
−3	−4	−4	−3	−3	−2	−1	0	0	0	1	2	3	3	4	4
−3	−5	−4	−4	−3	−2	−2	−1	0	1	2	2	3	4	4	5
−4	−5	−5	−4	−4	−3	−2	−1	0	1	2	3	4	4	5	5
−4	−6	−5	−5	−4	−3	−2	−1	0	1	2	3	4	5	5	6

Figure 4.5 – Quantized values of Macroblock 5, Frame 2.

In the first stage of run length coding the values of the zig-zag array are scanned and for each non-zero value found, the number of subsequent zeroes before it is recorded. The number of zeroes counted represents the RUN parameter. The actual value of the non-zero number represents the LEVEL parameter. A third parameter LAST indicates whether the current number is the last value of the array. LAST is set to be 1 for the last value and 0 for all the other values in the array. These parameters are used along with the Huffman Table to assign a proper codeword for each set of parameters. In that way runlength coding achieves further compression by greatly reducing the number of values required for the Motion JPEG output.

4.2.6 The Motion JPEG type Frames

At this point the encoding part is finished and what remains is to output the information obtained to Motion JPEG type files. The encoder produces a sequence of .mjg motion JPEG files. The first frame in the sequence is INTRA and the information of all the INTRA macroblocks is stored in the output file. The INTER frames contain the information of the encoded INTER macroblocks whole all their INTRA macroblocks are taken by the decoder from their previous INTRA appearance in the sequence. The following figure (Figure 4.6) displays the structure of information in an INTER frame.

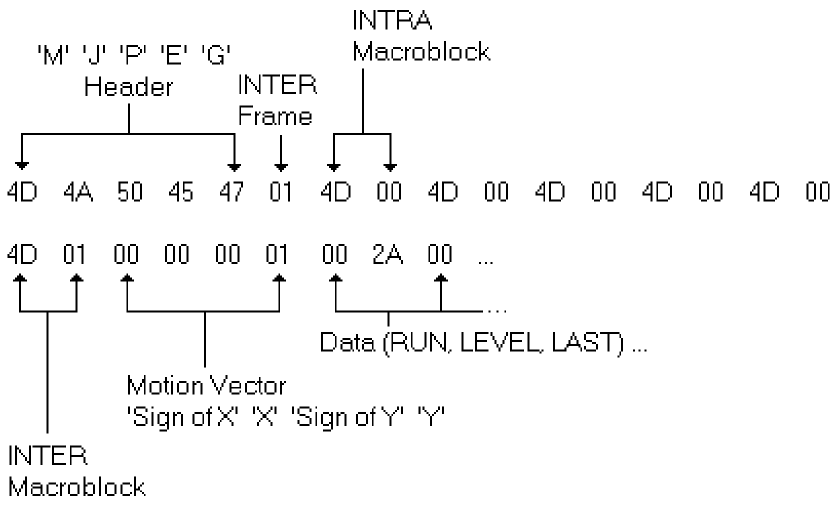
INTRA
Macroblock
'M' 'J' 'P' 'E' 'G'
Header
INTER
Frame
4D 4A 50 45 47 01 4D 00 4D 00 4D 00 4D 00 4D 00
4D 01 00 00 00 01 00 2A 00 ...
Data (RUN, LEVEL, LAST) ...
Motion Vector
'Sign of X' 'X' 'Sign of Y' 'Y'
INTER
Macroblock

Figure 4.6 – Initial segment of Information stored in Frame2.mjg.

The following table (Table 4.6) shows the concstructed motion JPEG type file and their size.

Frame Filename	**Size (Bytes)**
Frame1.mjg	3102
Frame2.mjg	601
Frame3.mjg	601
Frame4.mjg	1298
Frame5.mjg	1160
Frame6.mjg	1130
Frame7.mjg	673
Frame8.mjg	613
Frame9.mjg	601
Frame10.mjg	589

Table 4.6 – The constructed Motion JPEG frames.

Every frame of the raw sequence is 3072 bytes in size and the size of the whole 10 frame raw sequence is 30720 bytes. The size of the 10 frame motion JPEG sequence is 10368 bytes which means a significant amount compression was achieved and the resulting sequence is about a third of the size of the original.

4.3 The Motion JPEG Decoder

4.3.1 Addition of a Motion JPEG Decoder

In the beginning of the motion JPEG decode design a new parameter ‘d’ was added to the standard motion JPEG mode parameter ‘m’. In this mode a series of motion JPEG frames can be decoded to raw frames. The command line format is as given below,

```
jpeg –md –iw Width –ih Height –in NumberOfFrames FrameList
```

After the ‘md’ parameter, indicating the Motion JPEG decoding mode the width and height of the frames in the sequence has to be given. This is followed by the number of frames in the sequence and the filenames of the frames used. An example of a typical command line used is given below,

```
jpeg –md –iw 48 –ih 64 –in 10 frame1.mjg frame2.mjg frame3.mjg
frame4.mjg frame5.mjg frame6.mjg frame7.mjg frame8.mjg frame9.mjg
frame10.mjg
```

This command line will decode the 10 motion JPEG frames, 48 pixels wide and 64 pixels in height into raw frames. The next step is to process the frames given in the command line one by one, starting by setting the ‘Virtual’ frame.

4.3.2 The Idea of a Virtual Frame

A virtual frame contains all the raw information needed by the decoder to reconstruct the current frame. The first frame of the motion JPEG sequence is always INTRA. The decoder initially reads the information contained in the first frame and transforms the macroblock structured information into raw information. This information is stored in the current virtual frame. When the decoder needs to process an INTER frame it decodes the information contained in the INTER macroblocks and then updates the virtual frame with the current changes in raw information. The virtual frame solves the problem of having to look for changes in all the previous frames, for INTRA information not included in the current frame. It also simplifies the whole design as for every current frame decoded, the decoder only has to output the updated information of the virtual frame.

4.3.3 The Decoding Process

Through the decoding process all frames in the sequence are decoded one by one. When the decoder reaches an INTER frame it searches for all INTER macroblocks and gathers all the information left for them by the encoder. This information includes the motion vector of the macroblock and the encoded data.

The first step is to pass the encoded information of the macroblock throught the inverse run-length coding. The inverse run-length algorithm searches through the Huffman table (borrowed from H.263) to find a combination of RUN, LEVEL and LAST parameters that matches the current code word. In that way all the code words in the macroblock are assigned a set of RUN, LEVEL and LAST parameters. From the RUN parameter the number of zeroes proceeding the non-zero LEVEL value are set, and the whole process repeats until the LEVEL value having a LAST parameter equal to 1 is reached. In that way the quantized values in the macroblock are restored.

These values are stored in a 1-Dimensional array and they have to be rearranged in a 2-Dimensinal matrix using the inverse zig-zag technique. The inverse zig-zag algorithm uses an index to rearrange the information in a 16 by 16 matrix.

The matrix obtained is now processed through an inverse quantization algorithm where all its values are multiplied by (2*Q) where Q is the Quantization Factor. The output matrix is now ready for the Inverse Discrete Cosine Transform (IDCT). The equation used by the IDCT is,

$$iDCTMatrix(i, j) = (1/4)\sum_{u=0}^{15}\sum_{v=0}^{15} C(u)C(v)iRMatrix(u,v)\cos((2i+1)u\pi/16)\cos((2j+1)v\pi/16)$$

Where again,

$$C(x) = \left\{\begin{matrix} 1/\sqrt{2} & \text{x} = 0 \\ 1 & \text{otherwise} \end{matrix}\right\}$$

The values obtained are the values of the inverse residual matrix. Using the motion vector of the macroblock we can obtain the values of the best matching macroblock. These values along with the values in the inverse residual matrix can finally give raw data using the following transform,

$$\begin{bmatrix} ir(0,0) + bm(0,0) & ir(1,0) + bm(1,0) & \ldots & ir(15,0) + bm(15,0) \\ ir(0,1) + bm(0,1) & \ldots & \ldots & ir(15,1) + bm(15,1) \\ \vdots & & & \\ ir(0,15) + bm(0,15) & \ldots & \ldots & ir(15,15) + bm(15,15) \end{bmatrix}$$

In this matrix raw data are obtained by addition of each value ir(x, y) in the inverse residual matrix with each value bm(x, y) in the best matching macroblock. For each raw macroblock value, its correct position in the raw frame must now be found. Each value is set in the appropriate position in the virtual frame which is now updated to reflect the current changes.

4.3.4 The Decoded Raw Frames

For every current motion JPEG frame decoded the virtual frame is updated before the values are placed in a raw frame. All 10 frames in the sequence where decoded successfully giving raw (.ra0) files of 3072 bytes in size. The following figure (Figure 4.7) shows the differences in information from a portion of the original raw image and the decoded raw image as taken from macroblock 5 of frame 2.

```
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255   0   0   0   0   0 255 255 255 255 255
255 255 255 255 255   0 192 192 192 192 192   0 255 255 255 255
255 255 255 255   0 192 192 192 192 192 192 192   0 255 255 255
255 255 255   0 192 192 192 192 192 192 192 192 192   0 255 255
255 255 255   0 192 192 192 192 192 192 192 192 192   0 255 255
255 255 255   0 192 192 192 192 192 192 192 192 192   0 255 255
255 255 255   0 192 192 192 192 192 192 192 192 192   0 255 255
255 255 255   0 192 192 192 192 192 192 192 192 192   0 255 255
255 255 255 255   0 192 192 192 192 192 192 192   0 255 255 255
255 255 255 255 255   0 192 192 192 192 192   0 255 255 255 255
255 255 255 255 255 255   0   0   0   0   0 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
```

```
255 244 246 254 255 255 249 255 255 249 255 255 254 246 244 255
255 255 255 255 255 232 255 235 235 255 232 255 255 255 255 255
255 255 255 255 255 255 254 239 239 254 255 255 255 255 255 255
255 241 255 255 255 237 253 255 255 253 237 255 255 255 241 255
231 255 255 255 235 255   0   0   0   0   1 235 255 255 255 231
255 255 251 235 255  20 171 206 206 171 212  42 235 251 255 255
255 255 255 255   0 166 205 211 211 205 166 174  19 255 255 255
236 255 255   3 186 181 190 181 181 190 181 186 195  22 255 236
236 255 255   3 186 181 190 181 181 190 181 186 195  22 255 236
255 255 255  19 174 166 205 211 211 205 166 174 211   2 255 255
255 255 251   0 234 212 171 206 206 171 212 234 172   0 255 255
231 255 255  12 172 193 177 179 179 177 193 172 204   8 255 231
255 241 255 255   9 174 190 224 224 190 174 201  20 255 241 255
255 255 255 255 255  19 191 176 176 191 211   1 255 255 255 255
255 255 255 255 255 232  36   0   0  36   0 255 255 255 255 255
255 244 246 254 255 255 249 255 255 249 255 255 254 246 244 255
```

Figure 4.7 – Differences of Information in the macroblock 5 area between the original frame2.raw (Top) and the decoded frame2.ra0 (Bottom).

A comparison between an original raw and decoded raw frame in the sequence can be seen in the following figure (Figure 4.8),

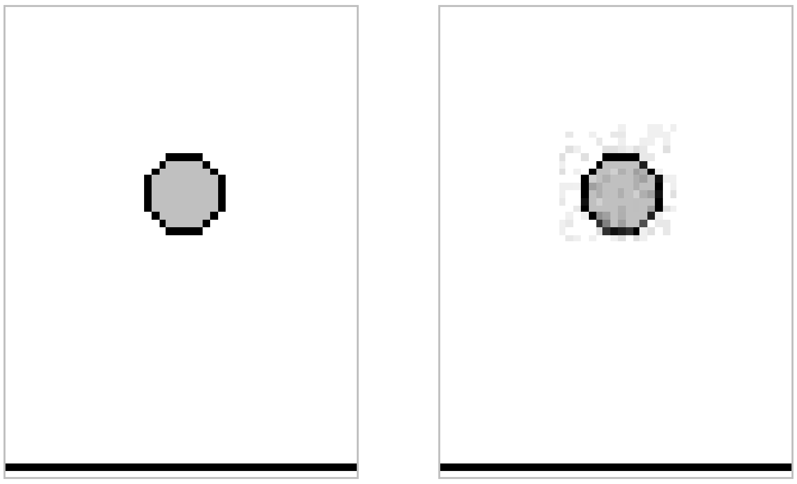

Figure 4.8 – The original raw Frame2.raw (Left) and the decoded raw Frame2.ra0 (Right).

The peak signal to noise ratio (PSNR) was also calculated for the decoded frames of the sequence and the results can be seen in the following table (Table 4.7).

Frame	**PSNR (dB)**
Frame1.ra0	-
Frame2.ra0	50.580
Frame3.ra0	48.179
Frame4.ra0	41.729
Frame5.ra0	40.331
Frame6.ra0	39.630
Frame7.ra0	38.376
Frame8.ra0	37.847
Frame9.ra0	37.860
Frame10.ra0	37.932

Table 4.7 – The PSNR of the frames in the sequence.

The total decrease in the value of PSNR reflects the increase of the noise in the signal, as the frames in the sequence are subsequently decoded.

Chapter 5 - Conclusions

5.1 Project Discussion and Conclusions

5.1.1 JPEG Performance

The JPEG image compression format provides the most efficient image compression standard available today. It is a well designed product of the collaboration between many researchers, for many years. The standard JPEG code written in C has been refined and optimized many times both before a finallized JPEG standard was defined as well as in later times. The technique achieves its results based on a number of fundamental and universal mathematical algorithms that are now considered as classics. Such an algorithm is the Discrete Cosine Transformation (DCT) of numbers, which can be found in the heart of the JPEG design. The DCT algorithm in its simplest form can be still found in many hardware JPEG implementations offering good performance and low manufacturing costs. As computers become more powerful and communications requirements more demanding, a need appears for the ultimate in terms of speed of execution of a software or hardware component. Since the DCT algorithm is the most time-consuming function in a typical JPEG design many ‘Fast’ DCT (FDCT) algorithms were invented throught the years in order to perform this task as fast as possible. In the standard JPEG C code used, the DCT process is implemented using the Chen, Smith and Frallick version of FDCT. Thorough study of this algorithm and its C code implementation revealed two of its most important qualities. At first the algorithm offered very fast execution times that outperform other FDCTs. In fact the algorithm is documented to be ‘6’ times faster than the simple DCT. The second characteristic is that the algorithm enables simplicity in software implementation. This property doesn't give much space for anyone to consider code optimization as a way of making the algorithm faster. In general, the whole JPEG design was found to be a stable and mature product with obvious signs of the tremendous amount of work being done in almost every part of its implementation.

5.1.2 Motion JPEG for Video Compression

The JPEG standard offers high levels of compression for still images while retaining good image quality. A Motion JPEG addition exploits the basic ideas of video coding to give JPEG the cababililty of handling video sequences. Today with the expansion of multimedia applications there is a lot of market space for Motion JPEG and in that sense it is a pity that it is not yet standardized. Still, a large number of both software and hardware Motion JPEG implementations can be found today, but due to lack of standardization these are not compatible across different vendors. Compared to other video compression techniques such as MPEG, motion JPEG doesn't support sound and achieves significally lower compression levels. Motion JPEG on the other hand has the advantage of allowing easy editing of the frames in a video sequence even with single frame precision. In most cases today, a video sequence recorded will be first converted into Motion JPEG format, then it will be edited and finally it will be again converted to a more efficient video compression format, such as MPEG-2, for storage or transmission.

5.1.3 The Motion JPEG Encoder Design

The Motion JPEG encoder designed provides an extension to the standard JPEG for converting raw video frames to compressed Motion JPEG type frames. The command line input can accept an unlimited number of frames of a video sequence. Video coding techniques such as motion estimation were used to detect motion between subsequent frames. The information contained in the macroblocks that motion was detected is encoded and contained in the output motion JPEG frame. For the macroblocks where no motion was detected, the encoder sends an output that directs the decoder to obtain their information from a previous frame in the sequence. In that way high levels of compression are achieved. For the sequence used, the total size of the resulting motion JPEG sequence was 1/3 of the total size of the original raw sequence. The encoder implemented provides the decoder with information, sufficient for good quality reconstruction of the frame. This quality can be further improved if in the encoder the current raw frame is compared with the previous reconstructed frame.

5.1.4 The Motion JPEG Decoder Design

Through the Motion JPEG decoder, the motion JPEG files produced by the encoder can be successfully converter back to raw format. The decoder completes the whole design as a transmitter can convert a raw sequence to Motion JPEG sequence and transmit the sequence which will be decoded at the receiver to the original raw format. This of course implies some loss of quality in the frames but the reduction in size means faster transmission and thus reduced transmission costs. The decoder implementation uses the idea of a virtual frame, which constantly keeps updated raw frame information as the decoding progresses. The total decrease of the PSNR value obtained after decoding the whole sequence reflects the build-up of noise. The motion JPEG sequence used was successfully decoded into raw frames of acceptable quality.

References

- K. R. Rao, J. J. Hwang
  Techniques & Standards for Image - Video & Audio Coding
  Prentice Hall PTR, 1996
- W. B. Pennebaker, J. L. Mitchell
  JPEG Still Image Data Compression Standard
  International Thomson Publishing, 1993
- J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, D. J. LeGall
  MPEG Video Compression Standard
  Chapman & Hall, 1997
- A. C. Luther
  Principles of Digital Audio and Video
  Artech House Publishers, 1997
- W. Kou
  Digital Image Compression Algorithms and Standards
  Kluwer Academic Publishers, 1995
- V. Bhaskaran, K. Konstantinides
  Image and Video Compression Standards, Second Edition
  Kluwer Academic Publishers, 1997
- E. R. Dougherty, P. A. Laplante
  Introduction to Real-Time Imaging
  IEEE Press, 1995
- B. G. Haskell, A. Puri, Arun Netravali
  Digital Video: An Introduction to MPEG-2
  Chapman & Hall, 1997

Appendix A – Encoder Profiling Results

```
 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
seconds    for by this function and those listed above it.

self       the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

self       the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

Total      the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.
```

<u>Flat profile:</u>

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 28.75       0.23      0.23      5700     40.35     40.35  ChenDct
 18.75       0.38      0.15                                mcount
 16.25       0.51      0.13      5700     22.81     22.81  Quantize
 10.00       0.59      0.08     45200      1.77      1.77  ReadXBuffer
  6.25       0.64      0.05      5700      8.77     18.25  EncodeAC
  5.00       0.68      0.04     58253      0.69      0.87  EncodeHuffman
  3.75       0.71      0.03     45600      0.66      2.41  ReadXBound
  3.75       0.74      0.03      5700      5.26      5.26  PreshiftDctMatrix
  2.50       0.76      0.02    110801      0.18      0.18  meputv
  1.25       0.77      0.01      5700      1.75      1.75  BoundDctMatrix
  1.25       0.78      0.01      5700      1.75     21.05  ReadBlock
  1.25       0.79      0.01      5700      1.75      1.75  ZigzagMatrix
  1.25       0.80      0.01         1 10000.00 650000.00  JpegEncodeScan
  0.00       0.80      0.00     35174      0.00      0.00  bputc
  0.00       0.80      0.00      5701      0.00      0.00  UseACHuffman
  0.00       0.80      0.00      5701      0.00      0.00  UseDCHuffman
  0.00       0.80      0.00      5700      0.00      1.05  EncodeDC
  0.00       0.80      0.00      2857      0.00      0.00  InstallIob
  0.00       0.80      0.00      2850      0.00      0.00  InstallPrediction
  0.00       0.80      0.00      2000      0.00      0.00  ReadResizeBuffer
  0.00       0.80      0.00       334      0.00      0.00  yylook
  0.00       0.80      0.00        82      0.00      0.00  yylex
  0.00       0.80      0.00        75      0.00      0.00  BlockMoveTo
  0.00       0.80      0.00        70      0.00      0.00  enter
  0.00       0.80      0.00        70      0.00      0.00  hashpjw
  0.00       0.80      0.00        42      0.00      0.00  MakeLink
  0.00       0.80      0.00        32      0.00      0.00  MakeXBuffer
  0.00       0.80      0.00        17      0.00      0.00  getint
  0.00       0.80      0.00         8      0.00      0.00  mwseek
  0.00       0.80      0.00         8      0.00      0.00  mwtell
  0.00       0.80      0.00         8      0.00      0.00  swbytealign
  0.00       0.80      0.00         4      0.00      0.00  getstr
  0.00       0.80      0.00         3      0.00      0.00  CloseIob
  0.00       0.80      0.00         3      0.00      0.00  RewindIob
  0.00       0.80      0.00         2      0.00      0.00  CodeTable
  0.00       0.80      0.00         2      0.00      0.00  MakeEhuff
  0.00       0.80      0.00         2      0.00      0.00  MakeXhuff
  0.00       0.80      0.00         2      0.00      0.00  OrderCodes
  0.00      0.80     0.00        2     0.00     0.00  SizeTable
  0.00      0.80     0.00        2     0.00     0.00  SpecifiedHuffman
  0.00      0.80     0.00        2     0.00     0.00  WriteHuffman
  0.00      0.80     0.00        1     0.00     0.00  CheckBaseline
  0.00      0.80     0.00        1     0.00     0.00  CheckScan
  0.00      0.80     0.00        1     0.00     0.00  CheckValidity
  0.00      0.80     0.00        1     0.00     0.00
ClearFrameFrequency
  0.00      0.80     0.00        1     0.00     0.00  ConfirmFileSize
  0.00      0.80     0.00        1     0.00     0.00
  MakeConsistentFrameSize

  0.00      0.80     0.00        1     0.00     0.00  MakeFrame
  0.00      0.80     0.00        1     0.00     0.00  MakeImage
  0.00      0.80     0.00        1     0.00     0.00  MakeIob
  0.00      0.80     0.00        1     0.00     0.00  MakeScan
  0.00      0.80     0.00        1     0.00     0.00  MakeScanFrequency
  0.00      0.80     0.00        1     0.00     0.00  ResetCodec
  0.00      0.80     0.00        1     0.00     0.00  SetACHuffman
  0.00      0.80     0.00        1     0.00     0.00  SetDCHuffman
  0.00      0.80     0.00        1     0.00     0.00  WriteDht
  0.00      0.80     0.00        1     0.00     0.00  WriteDnl
  0.00      0.80     0.00        1     0.00     0.00  WriteDqt
  0.00      0.80     0.00        1     0.00     0.00  WriteEoi
  0.00      0.80     0.00        1     0.00     0.00  WriteSof
  0.00      0.80     0.00        1     0.00     0.00  WriteSoi
  0.00      0.80     0.00        1     0.00     0.00  WriteSos
  0.00      0.80     0.00        1     0.00     0.00  initparser
  0.00      0.80     0.00        1     0.00 650000.00  main
  0.00      0.80     0.00        1     0.00     0.00  mwclose
  0.00      0.80     0.00        1     0.00     0.00  mwopen
  0.00      0.80     0.00        1     0.00 650000.00  parser
  0.00      0.80     0.00        1     0.00     0.00  pushstream

                 Call graph (explanation follows)
```

```
 granularity: each sample hit covers 4 byte(s) for 1.54% of 0.65
seconds

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line with the
 index number at the left hand margin lists the current function.
 The lines above it list the functions that called this function,
 and the lines below it list the functions this one called.
 This line lists:
     index	A unique number given to each element of the table.
		Index numbers are sorted numerically.
		The index number is printed next to every function name so
		it is easier to look up where the function in the table.

     % time	This is the percentage of the ‘total’ time that was spent
		in this function and its children.  Note that due to
		different viewpoints, functions excluded by options, etc,
		these numbers will NOT add up to 100%.

     self	This is the total amount of time spent in this function.

     children	This is the total amount of time propagated into this
		function by its children.

     called	This is the number of times the function was called.
		If the function called itself recursively, the number
		only includes non-recursive calls, and is followed by
		a ‘+’ and the number of recursive calls.

     name	The name of the current function.  The index number is
		printed after it.  If the function is a member of a
		cycle, the cycle number is printed between the
		function’s name and the index number.


 For the function’s parents, the fields have the following meanings:
```

self This is the amount of time that was propagated directly from the function into this parent.

children This is the amount of time that was propagated from the function’s children into this parent.

called This is the number of times this parent called the function ‘/’ the total number of times the function was called. Recursive calls to the function are not included in the number after the ‘/’.

name This is the name of the parent. The parent’s index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined, the word ‘<spontaneous>’ is printed in the `name' field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

self This is the amount of time that was propagated directly from the child into the function.

children This is the amount of time that was propagated from the child's children to the function.

called This is the number of times the function called this child `/' the total number of times the child was called. Recursive calls by the child are not listed in the number after the `/'.

name This is the name of the child. The child's index number is printed after it. If the child is a member of a cycle, the cycle number is printed between the name and the index number.

```
If there are any cycles (circles) in the call graph, there is an
entry for the cycle-as-a-whole.  This entry shows who called the
cycle (as parents) and the members of the cycle (as children.)
The ‘+’ recursive calls entry shows the number of function calls that
were internal to the cycle, and the calls entry for each member shows,
for that member, how many times it was called from other members of
the cycle.
```

```
index % time    self  children    called     name
                0.01    0.64       1/1           parser [3]
[1]    100.0    0.01    0.64       1             JpegEncodeScan [1]
                0.23    0.00    5700/5700        ChenDct [5]
                0.13    0.00    5700/5700        Quantize [6]
                0.01    0.11    5700/5700        ReadBlock [7]
                0.05    0.05    5700/5700        EncodeAC [9]
                0.03    0.00    5700/5700        PreshiftDctMatrix [12]
                0.01    0.00    5700/5700        BoundDctMatrix [14]
                0.01    0.00    5700/5700        ZigzagMatrix [15]
                0.00    0.01    5700/5700        EncodeDC [16]
                0.00    0.00    5700/5701        UseDCHuffman [19]
                0.00    0.00    5700/5701        UseACHuffman [18]
                0.00    0.00    2854/2857        InstallIob [20]
                0.00    0.00    2850/2850        InstallPrediction [21]
                0.00    0.00       3/3           RewindIob [36]
                0.00    0.00       1/1           CheckScan [45]
                0.00    0.00       1/1           ClearFrameFrequency
[47]
                0.00    0.00       1/1           ResetCodec [55]
                0.00    0.00       1/1           WriteSos [64]
                0.00    0.00       1/1           WriteDnl [59]
-----------------------------------------------
                0.00    0.65       1/1           _start [4]
[2]    100.0    0.00    0.65       1             main [2]
```

```
                0.00    0.65       1/1           parser [3]
                0.00    0.00       1/1           MakeImage [51]
                0.00    0.00       1/1           MakeFrame [50]
                0.00    0.00       1/1           MakeScan [53]
                0.00    0.00       1/1           initparser [65]
-----------------------------------------------
                0.00    0.65       1/1           main [2]
[3]    100.0    0.00    0.65       1             parser [3]
                0.01    0.64       1/1           JpegEncodeScan [1]
                0.00    0.00      61/82          yylex [24]
                0.00    0.00      17/17          getint [30]
                0.00    0.00       4/4           getstr [34]
                0.00    0.00       3/2857        InstallIob [20]
                0.00    0.00       3/3           CloseIob [35]
                0.00    0.00       2/2           MakeXhuff [39]
                0.00    0.00       2/2           MakeEhuff [38]
                0.00    0.00       2/2           SpecifiedHuffman [42]
                0.00    0.00       1/1           CheckValidity [46]
                0.00    0.00       1/1           CheckBaseline [44]
                0.00    0.00       1/1           ConfirmFileSize [48]
                0.00    0.00       1/1           MakeIob [52]
                0.00    0.00       1/1           mwopen [67]
                0.00    0.00       1/1           mwclose [66]
                0.00    0.00       1/1
MakeConsistentFrameSize [49]
                0.00    0.00       1/1           WriteSof [62]
                0.00    0.00       1/1           WriteSoi [63]
                0.00    0.00       1/1           WriteEoi [61]
                0.00    0.00       1/1           WriteDqt [60]
                0.00    0.00       1/1           WriteDht [58]
                0.00    0.00       1/1           SetACHuffman [56]
                0.00    0.00       1/1           SetDCHuffman [57]
-----------------------------------------------
                                                 <spontaneous>
[4]    100.0    0.00    0.65                     _start [4]
                0.00    0.65       1/1           main [2]
-----------------------------------------------
                0.23    0.00    5700/5700        JpegEncodeScan [1]
```

```
[5]     35.4    0.23    0.00    5700             ChenDct [5]
-----------------------------------------------
                0.13    0.00    5700/5700        JpegEncodeScan [1]
[6]     20.0    0.13    0.00    5700             Quantize [6]
-----------------------------------------------
                0.01    0.11    5700/5700        JpegEncodeScan [1]
[7]     18.5    0.01    0.11    5700             ReadBlock [7]
                0.03    0.08   45600/45600       ReadXBound [8]
                0.00    0.00      75/75          BlockMoveTo [25]
-----------------------------------------------
                0.03    0.08   45600/45600       ReadBlock [7]
[8]     16.9    0.03    0.08   45600             ReadXBound [8]
                0.08    0.00   45200/45200       ReadXBuffer [10]
-----------------------------------------------
                0.05    0.05    5700/5700        JpegEncodeScan [1]
[9]     16.0    0.05    0.05    5700             EncodeAC [9]
                0.04    0.01   52553/58253       EncodeHuffman [11]
                0.01    0.00   46848/110801      meputv [13]
-----------------------------------------------
                0.08    0.00   45200/45200       ReadXBound [8]
[10]    12.3    0.08    0.00   45200             ReadXBuffer [10]
                0.00    0.00    2000/2000        ReadResizeBuffer [22]
-----------------------------------------------
                0.00    0.00    5700/58253       EncodeDC [16]
                0.04    0.01   52553/58253       EncodeAC [9]
[11]     7.8    0.04    0.01   58253             EncodeHuffman [11]
                0.01    0.00   58253/110801      meputv [13]
-----------------------------------------------
                0.03    0.00    5700/5700        JpegEncodeScan [1]
[12]     4.6    0.03    0.00    5700             PreshiftDctMatrix [12]
-----------------------------------------------
                0.00    0.00    5700/110801      EncodeDC [16]
                0.01    0.00   46848/110801      EncodeAC [9]
                0.01    0.00   58253/110801      EncodeHuffman [11]
[13]     3.1    0.02    0.00  110801             meputv [13]
                0.00    0.00   34841/35174       bputc [17]
-----------------------------------------------
                0.01    0.00    5700/5700        JpegEncodeScan [1]
```

```
[14]     1.5    0.01    0.00    5700             BoundDctMatrix [14]
-----------------------------------------------
                0.01    0.00    5700/5700        JpegEncodeScan [1]
[15]     1.5    0.01    0.00    5700             ZigzagMatrix [15]
-----------------------------------------------
                0.00    0.01    5700/5700        JpegEncodeScan [1]
[16]     0.9    0.00    0.01    5700             EncodeDC [16]
                0.00    0.00    5700/58253       EncodeHuffman [11]
                0.00    0.00    5700/110801      meputv [13]
-----------------------------------------------
                0.00    0.00       1/35174       swbytealign [33]
                0.00    0.00       2/35174       WriteSoi [63]
                0.00    0.00       2/35174       WriteEoi [61]
                0.00    0.00       6/35174       WriteDnl [59]
                0.00    0.00       8/35174       WriteDht [58]
                0.00    0.00      16/35174       WriteSos [64]
                0.00    0.00      21/35174       WriteSof [62]
                0.00    0.00      71/35174       WriteDqt [60]
                0.00    0.00     206/35174       WriteHuffman [43]
                0.00    0.00   34841/35174       meputv [13]
[17]     0.0    0.00    0.00   35174             bputc [17]
-----------------------------------------------
                0.00    0.00       1/5701        WriteDht [58]
                0.00    0.00    5700/5701        JpegEncodeScan [1]
[18]     0.0    0.00    0.00    5701             UseACHuffman [18]
-----------------------------------------------
                0.00    0.00       1/5701        WriteDht [58]
                0.00    0.00    5700/5701        JpegEncodeScan [1]
[19]     0.0    0.00    0.00    5701             UseDCHuffman [19]
-----------------------------------------------
                0.00    0.00       3/2857        parser [3]
                0.00    0.00    2854/2857        JpegEncodeScan [1]
[20]     0.0    0.00    0.00    2857             InstallIob [20]
-----------------------------------------------
                0.00    0.00    2850/2850        JpegEncodeScan [1]
[21]     0.0    0.00    0.00    2850             InstallPrediction [21]
-----------------------------------------------
                0.00    0.00    2000/2000        ReadXBuffer [10]
```

```
[22]     0.0    0.00    0.00    2000            ReadResizeBuffer [22]
-----------------------------------------------
                0.00    0.00     334/334        yylex [24]
[23]     0.0    0.00    0.00     334            yylook [23]
-----------------------------------------------
                0.00    0.00       4/82         getstr [34]
                0.00    0.00      17/82         getint [30]
                0.00    0.00      61/82         parser [3]
[24]     0.0    0.00    0.00      82            yylex [24]
                0.00    0.00     334/334        yylook [23]
                0.00    0.00      28/70         enter [26]
-----------------------------------------------
                0.00    0.00      75/75         ReadBlock [7]
[25]     0.0    0.00    0.00      75            BlockMoveTo [25]
-----------------------------------------------
                0.00    0.00      28/70         yylex [24]
                0.00    0.00      42/70         initparser [65]
[26]     0.0    0.00    0.00      70            enter [26]
                0.00    0.00      70/70         hashpjw [27]
                0.00    0.00      42/42         MakeLink [28]
-----------------------------------------------
                0.00    0.00      70/70         enter [26]
[27]     0.0    0.00    0.00      70            hashpjw [27]
-----------------------------------------------
                0.00    0.00      42/42         enter [26]
[28]     0.0    0.00    0.00      42            MakeLink [28]
-----------------------------------------------
                0.00    0.00      32/32         MakeIob [52]
[29]     0.0    0.00    0.00      32            MakeXBuffer [29]
-----------------------------------------------
                0.00    0.00      17/17         parser [3]
[30]     0.0    0.00    0.00      17            getint [30]
                0.00    0.00      17/82         yylex [24]
-----------------------------------------------
                0.00    0.00       2/8          WriteSof [62]
                0.00    0.00       2/8          WriteDqt [60]
                0.00    0.00       2/8          WriteSos [64]
                0.00    0.00       2/8          WriteDht [58]
```

```
[31]     0.0    0.00    0.00       8             mwseek [31]
-----------------------------------------------
                0.00    0.00       2/8           WriteSof [62]
                0.00    0.00       2/8           WriteDqt [60]
                0.00    0.00       2/8           WriteSos [64]
                0.00    0.00       2/8           WriteDht [58]
[32]     0.0    0.00    0.00       8             mwtell [32]
-----------------------------------------------
                0.00    0.00       1/8           WriteSoi [63]
                0.00    0.00       1/8           WriteEoi [61]
                0.00    0.00       1/8           WriteSof [62]
                0.00    0.00       1/8           WriteDnl [59]
                0.00    0.00       1/8           WriteDqt [60]
                0.00    0.00       1/8           WriteSos [64]
                0.00    0.00       1/8           WriteDht [58]
                0.00    0.00       1/8           mwclose [66]
[33]     0.0    0.00    0.00       8             swbytealign [33]
                0.00    0.00       1/35174       bputc [17]
-----------------------------------------------
                0.00    0.00       4/4           parser [3]
[34]     0.0    0.00    0.00       4             getstr [34]
                0.00    0.00       4/82          yylex [24]
-----------------------------------------------
                0.00    0.00       3/3           parser [3]
[35]     0.0    0.00    0.00       3             CloseIob [35]
-----------------------------------------------
                0.00    0.00       3/3           JpegEncodeScan [1]
[36]     0.0    0.00    0.00       3             RewindIob [36]
-----------------------------------------------
                0.00    0.00       2/2           SpecifiedHuffman [42]
[37]     0.0    0.00    0.00       2             CodeTable [37]
-----------------------------------------------
                0.00    0.00       2/2           parser [3]
[38]     0.0    0.00    0.00       2             MakeEhuff [38]
-----------------------------------------------
                0.00    0.00       2/2           parser [3]
[39]     0.0    0.00    0.00       2             MakeXhuff [39]
-----------------------------------------------
```

```
                0.00    0.00       2/2           SpecifiedHuffman [42]
[40]     0.0    0.00    0.00       2             OrderCodes [40]
-----------------------------------------------
                0.00    0.00       2/2           SpecifiedHuffman [42]
[41]     0.0    0.00    0.00       2             SizeTable [41]
-----------------------------------------------
                0.00    0.00       2/2           parser [3]
[42]     0.0    0.00    0.00       2             SpecifiedHuffman [42]
                0.00    0.00       2/2           SizeTable [41]
                0.00    0.00       2/2           CodeTable [37]
                0.00    0.00       2/2           OrderCodes [40]
-----------------------------------------------
                0.00    0.00       2/2           WriteDht [58]
[43]     0.0    0.00    0.00       2             WriteHuffman [43]
                0.00    0.00     206/35174       bputc [17]
-----------------------------------------------
                0.00    0.00       1/1           parser [3]
[44]     0.0    0.00    0.00       1             CheckBaseline [44]
-----------------------------------------------
                0.00    0.00       1/1           JpegEncodeScan [1]
[45]     0.0    0.00    0.00       1             CheckScan [45]
-----------------------------------------------
                0.00    0.00       1/1           parser [3]
[46]     0.0    0.00    0.00       1             CheckValidity [46]
-----------------------------------------------
                0.00    0.00       1/1           JpegEncodeScan [1]
[47]     0.0    0.00    0.00       1             ClearFrameFrequency
[47]
-----------------------------------------------
                0.00    0.00       1/1           parser [3]
[48]     0.0    0.00    0.00       1             ConfirmFileSize [48]
-----------------------------------------------
                0.00    0.00       1/1           parser [3]
[49]     0.0    0.00    0.00       1
MakeConsistentFrameSize [49]
-----------------------------------------------
                0.00    0.00       1/1           main [2]
[50]     0.0    0.00    0.00       1             MakeFrame [50]
```

```
-----------------------------------------------
                0.00    0.00      1/1         main [2]
[51]     0.0    0.00    0.00      1           MakeImage [51]
-----------------------------------------------
                0.00    0.00      1/1         parser [3]
[52]     0.0    0.00    0.00      1           MakeIob [52]
                0.00    0.00     32/32        MakeXBuffer [29]
-----------------------------------------------
                0.00    0.00      1/1         main [2]
[53]     0.0    0.00    0.00      1           MakeScan [53]
                0.00    0.00      1/1         MakeScanFrequency [54]
-----------------------------------------------
                0.00    0.00      1/1         MakeScan [53]
[54]     0.0    0.00    0.00      1           MakeScanFrequency [54]
-----------------------------------------------
                0.00    0.00      1/1         JpegEncodeScan [1]
[55]     0.0    0.00    0.00      1           ResetCodec [55]
-----------------------------------------------
                0.00    0.00      1/1         parser [3]
[56]     0.0    0.00    0.00      1           SetACHuffman [56]
-----------------------------------------------
                0.00    0.00      1/1         parser [3]
[57]     0.0    0.00    0.00      1           SetDCHuffman [57]
-----------------------------------------------
                0.00    0.00      1/1         parser [3]
[58]     0.0    0.00    0.00      1           WriteDht [58]
                0.00    0.00      8/35174     bputc [17]
                0.00    0.00      2/8         mwtell [32]
                0.00    0.00      2/2         WriteHuffman [43]
                0.00    0.00      2/8         mwseek [31]
                0.00    0.00      1/8         swbytealign [33]
                0.00    0.00      1/5701      UseDCHuffman [19]
                0.00    0.00      1/5701      UseACHuffman [18]
-----------------------------------------------
                0.00    0.00      1/1         JpegEncodeScan [1]
[59]     0.0    0.00    0.00      1           WriteDnl [59]
                0.00    0.00      6/35174     bputc [17]
                0.00    0.00      1/8         swbytealign [33]
```

```
-----------------------------------------------
                0.00    0.00       1/1          parser [3]
[60]     0.0    0.00    0.00       1            WriteDqt [60]
                0.00    0.00      71/35174      bputc [17]
                0.00    0.00       2/8          mwtell [32]
                0.00    0.00       2/8          mwseek [31]
                0.00    0.00       1/8          swbytealign [33]
-----------------------------------------------
                0.00    0.00       1/1          parser [3]
[61]     0.0    0.00    0.00       1            WriteEoi [61]
                0.00    0.00       2/35174      bputc [17]
                0.00    0.00       1/8          swbytealign [33]
-----------------------------------------------
                0.00    0.00       1/1          parser [3]
[62]     0.0    0.00    0.00       1            WriteSof [62]
                0.00    0.00      21/35174      bputc [17]
                0.00    0.00       2/8          mwtell [32]
                0.00    0.00       2/8          mwseek [31]
                0.00    0.00       1/8          swbytealign [33]
-----------------------------------------------
                0.00    0.00       1/1          parser [3]
[63]     0.0    0.00    0.00       1            WriteSoi [63]
                0.00    0.00       2/35174      bputc [17]
                0.00    0.00       1/8          swbytealign [33]
-----------------------------------------------
                0.00    0.00       1/1          JpegEncodeScan [1]
[64]     0.0    0.00    0.00       1            WriteSos [64]
                0.00    0.00      16/35174      bputc [17]
                0.00    0.00       2/8          mwtell [32]
                0.00    0.00       2/8          mwseek [31]
                0.00    0.00       1/8          swbytealign [33]
-----------------------------------------------
                0.00    0.00       1/1          main [2]
[65]     0.0    0.00    0.00       1            initparser [65]
                0.00    0.00      42/70         enter [26]
-----------------------------------------------
                0.00    0.00       1/1          parser [3]
[66]     0.0    0.00    0.00       1            mwclose [66]
```

```
                0.00    0.00       1/8           swbytealign [33]
-----------------------------------------------
                0.00    0.00       1/1           parser [3]
[67]     0.0    0.00    0.00       1             mwopen [67]
                0.00    0.00       1/1           pushstream [68]
-----------------------------------------------
                0.00    0.00       1/1           mwopen [67]
[68]     0.0    0.00    0.00       1             pushstream [68]
-----------------------------------------------
```

```
Index by function name

  [25] BlockMoveTo            [54] MakeScanFrequency      [62] WriteSof
  [14] BoundDctMatrix         [29] MakeXBuffer            [63] WriteSoi
  [44] CheckBaseline          [39] MakeXhuff              [64] WriteSos
  [45] CheckScan              [40] OrderCodes             [15]
ZigzagMatrix
  [46] CheckValidity          [12] PreshiftDctMatrix      [17] bputc
   [5] ChenDct                 [6] Quantize               [26] enter
  [47] ClearFrameFrequency     [7] ReadBlock              [30] getint
  [35] CloseIob               [22] ReadResizeBuffer       [34] getstr
  [37] CodeTable               [8] ReadXBound             [27] hashpjw
  [48] ConfirmFileSize        [10] ReadXBuffer            [65]
initparser
   [9] EncodeAC               [55] ResetCodec              [2] main
  [16] EncodeDC               [36] RewindIob             (166) mcount
  [11] EncodeHuffman          [56] SetACHuffman           [13] meputv
  [20] InstallIob             [57] SetDCHuffman           [66] mwclose
  [21] InstallPrediction      [41] SizeTable              [67] mwopen
   [1] JpegEncodeScan         [42] SpecifiedHuffman       [31] mwseek
  [49] MakeConsistentFrameSize [18] UseACHuffman          [32] mwtell
  [38] MakeEhuff              [19] UseDCHuffman            [3] parser
  [50] MakeFrame              [58] WriteDht               [68]
pushstream
```

```
  [51] MakeImage             [59] WriteDnl              [33]
swbytealign
  [52] MakeIob               [60] WriteDqt              [24] yylex
  [28] MakeLink              [61] WriteEoi              [23] yylook
  [53] MakeScan              [43] WriteHuffman
```

Appendix B – Decoder Profiling Results

```
%          the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
seconds    for by this function and those listed above it.

self       the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

self       the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

Total      the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
         the gprof listing if it were to be printed.
```

Flat profile:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 30.00      0.24     0.24     5700    42.11    42.11  ChenIDct
 22.50      0.42     0.18                             mcount
  7.50      0.48     0.06     5700    10.53    10.53  IQuantize
  6.25      0.53     0.05    45600     1.10     2.19  WriteXBound
  6.25      0.58     0.05    45200     1.11     1.11  WriteXBuffer
  5.00      0.62     0.04    58128     0.69     1.08  DecodeHuffman
  5.00      0.66     0.04     5700     7.02     7.02  IZigzagMatrix
  5.00      0.70     0.04     5700     7.02    24.56  WriteBlock
  3.75      0.73     0.03    51609     0.58     0.72  megetv
  2.50      0.75     0.02    46274     0.43     0.43  pgetc
  2.50      0.77     0.02     5700     3.51    19.35  DecodeAC
  1.25      0.78     0.01   234304     0.04     0.10  megetb
  1.25      0.79     0.01     5700     1.75     1.75  BoundIDctMatrix
  1.25      0.80     0.01     5700     1.75     1.75
PostshiftIDctMatrix
  0.00      0.80     0.00    47426     0.00     0.00  bgetc
  0.00      0.80     0.00     5700     0.00     1.70  DecodeDC
  0.00      0.80     0.00     5700     0.00     0.00  UseACHuffman
  0.00      0.80     0.00     5700     0.00     0.00  UseDCHuffman
  0.00      0.80     0.00     2854     0.00     0.00  InstallIob
  0.00      0.80     0.00     2850     0.00     0.00  InstallPrediction
  0.00      0.80     0.00     2032     0.00     0.00  FlushBuffer
  0.00      0.80     0.00       78     0.00     0.00  FlushIob
  0.00      0.80     0.00       75     0.00     0.00  BlockMoveTo
  0.00      0.80     0.00       32     0.00     0.00  MakeXBuffer
  0.00      0.80     0.00       16     0.00     0.00  mrtell
  0.00      0.80     0.00       14     0.00     0.00  ScreenMarker
  0.00      0.80     0.00       12     0.00     0.00  DoMarker
  0.00      0.80     0.00       12     0.00     0.00  bgetw
  0.00      0.80     0.00        6     0.00     0.00  bpushc
  0.00      0.80     0.00        4     0.00     0.00  CodeTable
  0.00      0.80     0.00        4     0.00     0.00  DecoderTables
  0.00      0.80     0.00        4     0.00     0.00  MakeDhuff
  0.00      0.80     0.00        4     0.00     0.00  MakeXhuff
  0.00      0.80     0.00        4     0.00     0.00  ReadDht
  0.00      0.80     0.00        4     0.00     0.00  ReadHuffman
  0.00      0.80     0.00        4     0.00     0.00  SizeTable
  0.00      0.80     0.00        3     0.00     0.00  CloseIob
  0.00      0.80     0.00        3     0.00     0.00  SeekEndIob
  0.00      0.80     0.00        2     0.00     0.00  ReadDqt
  0.00      0.80     0.00        2     0.00     0.00  ScreenAllMarker
  0.00      0.80     0.00        2     0.00     0.00  SetACHuffman
  0.00      0.80     0.00        2     0.00     0.00  SetDCHuffman
  0.00      0.80     0.00        1     0.00     0.00  CheckBaseline
  0.00      0.80     0.00        1     0.00     0.00  CheckScan
  0.00      0.80     0.00        1     0.00     0.00  CheckValidity
  0.00      0.80     0.00        1     0.00 620000.00  JpegDecodeFrame
  0.00      0.80     0.00        1     0.00 620000.00  JpegDecodeScan
  0.00      0.80     0.00        1     0.00     0.00
MakeConsistentFileNames

  0.00      0.80     0.00        1     0.00     0.00
MakeConsistentFrameSize

  0.00      0.80     0.00        1     0.00     0.00  MakeFrame
  0.00      0.80     0.00        1     0.00     0.00  MakeImage
  0.00      0.80     0.00        1     0.00     0.00  MakeIob
  0.00      0.80     0.00        1     0.00     0.00  MakeScan
  0.00      0.80     0.00        1     0.00     0.00  MakeScanFrequency
  0.00      0.80     0.00        1     0.00     0.00  ReadSof
  0.00      0.80     0.00        1     0.00     0.00  ReadSos
  0.00      0.80     0.00        1     0.00     0.00  ResetCodec
  0.00      0.80     0.00        1     0.00 620000.00  main
  0.00      0.80     0.00        1     0.00     0.00  mrclose
  0.00      0.80     0.00        1     0.00     0.00  mropen
  0.00      0.80     0.00        1     0.00     0.00  pushstream

Call graph (explanation follows)
```

```
granularity: each sample hit covers 4 byte(s) for 1.61% of 0.62 seconds

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line with the
 index number at the left hand margin lists the current function.
 The lines above it list the functions that called this function,
 and the lines below it list the functions this one called.
 This line lists:
     index  A unique number given to each element of the table.
            Index numbers are sorted numerically.
            The index number is printed next to every function name so
            it is easier to look up where the function in the table.

     % time This is the percentage of the ‘total’ time that was spent
            in this function and its children.  Note that due to
            different viewpoints, functions excluded by options, etc,
            these numbers will NOT add up to 100%.

     self   This is the total amount of time spent in this function.

     children     This is the total amount of time propagated into this
                  function by its children.

     called This is the number of times the function was called.
            If the function called itself recursively, the number
            only includes non-recursive calls, and is followed by
            a ‘+’ and the number of recursive calls.

     name   The name of the current function.  The index number is
            printed after it.  If the function is a member of a
            cycle, the cycle number is printed between the
            function’s name and the index number.
```

For the function’s parents, the fields have the following meanings:

self This is the amount of time that was propagated directly from the function into this parent.

children This is the amount of time that was propagated from the function’s children into this parent.

called This is the number of times this parent called the function ‘/’ the total number of times the function was called. Recursive calls to the function are not included in the number after the ‘/’.

name This is the name of the parent. The parent’s index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined, the word ‘<spontaneous>’ is printed in the ‘name’ field, and all the other fields are blank.

For the function’s children, the fields have the following meanings:

self This is the amount of time that was propagated directly from the child into the function.

children This is the amount of time that was propagated from the child’s children to the function.

called This is the number of times the function called this child ‘/’ the total number of times the child was called. Recursive calls by the child are not listed in the number after the ‘/’.

name This is the name of the child. The child’s index number is printed after it. If the child is a member of a cycle, the cycle number is printed

```
           between the name and the index number.

 If there are any cycles (circles) in the call graph, there is an
 entry for the cycle-as-a-whole.  This entry shows who called the
 cycle (as parents) and the members of the cycle (as children.)
 The '+' recursive calls entry shows the number of function calls that
 were internal to the cycle, and the calls entry for each member shows,
 for that member, how many times it was called from other members of
 the cycle.
```

```
index % time    self  children    called     name
                0.00    0.62       1/1           main [3]
[1]    100.0    0.00    0.62       1             JpegDecodeFrame [1]
                0.00    0.62       1/1           JpegDecodeScan [2]
                0.00    0.00       3/2854        InstallIob [22]
                0.00    0.00       3/78          FlushIob [25]
                0.00    0.00       3/3           SeekEndIob [41]
                0.00    0.00       3/3           CloseIob [40]
                0.00    0.00       2/2           ScreenAllMarker [43]
                0.00    0.00       1/1           mropen [60]
                0.00    0.00       1/1           mrclose [59]
-----------------------------------------------
                0.00    0.62       1/1           JpegDecodeFrame [1]
[2]    100.0    0.00    0.62       1             JpegDecodeScan [2]
                0.24    0.00    5700/5700        ChenIDct [5]
                0.04    0.10    5700/5700        WriteBlock [6]
                0.02    0.09    5700/5700        DecodeAC [7]
                0.06    0.00    5700/5700        IQuantize [10]
                0.04    0.00    5700/5700        IZigzagMatrix [12]
                0.01    0.00    5700/5700        PostshiftIDctMatrix
[17]
                0.01    0.00    5700/5700        BoundIDctMatrix [16]
                0.00    0.01    5700/5700        DecodeDC [18]
                0.00    0.00    5700/5700        UseDCHuffman [21]
                0.00    0.00    5700/5700        UseACHuffman [20]
                0.00    0.00    2850/2850       InstallPrediction [23]
                0.00    0.00    2850/2854       InstallIob [22]
-----------------------------------------------
                0.00    0.62       1/1          _start [4]
[3]    100.0    0.00    0.62       1            main [3]
                0.00    0.62       1/1          JpegDecodeFrame [1]
                0.00    0.00       1/1          MakeImage [52]
                0.00    0.00       1/1          MakeFrame [51]
                0.00    0.00       1/1          MakeScan [54]
-----------------------------------------------
                                                <spontaneous>
[4]    100.0    0.00    0.62                    _start [4]
                0.00    0.62       1/1          main [3]
-----------------------------------------------
                0.24    0.00    5700/5700       JpegDecodeScan [2]
[5]     38.7    0.24    0.00    5700            ChenIDct [5]
-----------------------------------------------
                0.04    0.10    5700/5700       JpegDecodeScan [2]
[6]     22.6    0.04    0.10    5700            WriteBlock [6]
                0.05    0.05   45600/45600      WriteXBound [8]
                0.00    0.00      75/78         FlushIob [25]
                0.00    0.00      75/75         BlockMoveTo [26]
-----------------------------------------------
                0.02    0.09    5700/5700       JpegDecodeScan [2]
[7]     17.8    0.02    0.09    5700            DecodeAC [7]
                0.04    0.02   52428/58128      DecodeHuffman [9]
                0.03    0.01   46725/51609      megetv [13]
-----------------------------------------------
                0.05    0.05   45600/45600      WriteBlock [6]
[8]     16.1    0.05    0.05   45600            WriteXBound [8]
                0.05    0.00   45200/45200      WriteXBuffer [11]
-----------------------------------------------
                0.00    0.00    5700/58128      DecodeDC [18]
                0.04    0.02   52428/58128      DecodeAC [7]
[9]     10.1    0.04    0.02   58128            DecodeHuffman [9]
                0.01    0.01  234304/234304     megetb [14]
-----------------------------------------------
                0.06    0.00    5700/5700       JpegDecodeScan [2]
```

```
[10]     9.7    0.06    0.00    5700             IQuantize [10]
-----------------------------------------------
                                1200             WriteXBuffer [11]
                0.05    0.00   45200/45200       WriteXBound [8]
[11]     8.1    0.05    0.00   45200+1200        WriteXBuffer [11]
                0.00    0.00    1200/2032        FlushBuffer [24]
                                1200             WriteXBuffer [11]
-----------------------------------------------
                0.04    0.00    5700/5700        JpegDecodeScan [2]
[12]     6.5    0.04    0.00    5700             IZigzagMatrix [12]
-----------------------------------------------
                0.00    0.00    4884/51609       DecodeDC [18]
                0.03    0.01   46725/51609       DecodeAC [7]
[13]     6.0    0.03    0.01   51609             megetv [13]
                0.01    0.00   17015/46274       pgetc [15]
-----------------------------------------------
                0.01    0.01  234304/234304      DecodeHuffman [9]
[14]     3.7    0.01    0.01  234304             megetb [14]
                0.01    0.00   29259/46274       pgetc [15]
-----------------------------------------------
                0.01    0.00   17015/46274       megetv [13]
                0.01    0.00   29259/46274       megetb [14]
[15]     3.2    0.02    0.00   46274             pgetc [15]
                0.00    0.00   46743/47426       bgetc [19]
-----------------------------------------------
                0.01    0.00    5700/5700        JpegDecodeScan [2]
[16]     1.6    0.01    0.00    5700             BoundIDctMatrix [16]
-----------------------------------------------
                0.01    0.00    5700/5700        JpegDecodeScan [2]
[17]     1.6    0.01    0.00    5700             PostshiftIDctMatrix
[17]
-----------------------------------------------
                0.00    0.01    5700/5700        JpegDecodeScan [2]
[18]     1.6    0.00    0.01    5700             DecodeDC [18]
                0.00    0.00    5700/58128       DecodeHuffman [9]
                0.00    0.00    4884/51609       megetv [13]
-----------------------------------------------
                0.00    0.00       8/47426       ReadDht [37]
                0.00    0.00      10/47426      ReadSos [57]
                0.00    0.00      11/47426      ReadSof [56]
                0.00    0.00      26/47426      ScreenMarker [29]
                0.00    0.00      84/47426      DoMarker [30]
                0.00    0.00     132/47426      ReadDqt [42]
                0.00    0.00     412/47426      ReadHuffman [38]
                0.00    0.00   46743/47426      pgetc [15]
[19]     0.0    0.00    0.00   47426            bgetc [19]
-----------------------------------------------
                0.00    0.00    5700/5700       JpegDecodeScan [2]
[20]     0.0    0.00    0.00    5700            UseACHuffman [20]
-----------------------------------------------
                0.00    0.00    5700/5700       JpegDecodeScan [2]
[21]     0.0    0.00    0.00    5700            UseDCHuffman [21]
-----------------------------------------------
                0.00    0.00       1/2854       ReadSos [57]
                0.00    0.00       3/2854       JpegDecodeFrame [1]
                0.00    0.00    2850/2854       JpegDecodeScan [2]
[22]     0.0    0.00    0.00    2854            InstallIob [22]
-----------------------------------------------
                0.00    0.00    2850/2850       JpegDecodeScan [2]
[23]     0.0    0.00    0.00    2850            InstallPrediction [23]
-----------------------------------------------
                0.00    0.00     832/2032       FlushIob [25]
                0.00    0.00    1200/2032       WriteXBuffer [11]
[24]     0.0    0.00    0.00    2032            FlushBuffer [24]
-----------------------------------------------
                0.00    0.00       3/78         JpegDecodeFrame [1]
                0.00    0.00      75/78         WriteBlock [6]
[25]     0.0    0.00    0.00      78            FlushIob [25]
                0.00    0.00     832/2032       FlushBuffer [24]
-----------------------------------------------
                0.00    0.00      75/75         WriteBlock [6]
[26]     0.0    0.00    0.00      75            BlockMoveTo [26]
-----------------------------------------------
                0.00    0.00      32/32         MakeIob [53]
[27]     0.0    0.00    0.00      32            MakeXBuffer [27]
-----------------------------------------------
```

```
                0.00    0.00       2/16          ReadSof [56]
                0.00    0.00       2/16          ReadSos [57]
                0.00    0.00       4/16          ReadDqt [42]
                0.00    0.00       8/16          ReadDht [37]
[28]     0.0    0.00    0.00      16             mrtell [28]
-----------------------------------------------
                0.00    0.00      14/14          ScreenAllMarker [43]
[29]     0.0    0.00    0.00      14             ScreenMarker [29]
                0.00    0.00      26/47426       bgetc [19]
                0.00    0.00      12/12          DoMarker [30]
-----------------------------------------------
                0.00    0.00      12/12          ScreenMarker [29]
[30]     0.0    0.00    0.00      12             DoMarker [30]
                0.00    0.00      84/47426       bgetc [19]
                0.00    0.00       4/4           ReadDht [37]
                0.00    0.00       2/2           ReadDqt [42]
                0.00    0.00       2/12          bgetw [31]
                0.00    0.00       1/1           ReadSof [56]
                0.00    0.00       1/1           ReadSos [57]
-----------------------------------------------
                0.00    0.00       1/12          ReadSos [57]
                0.00    0.00       2/12          ReadDqt [42]
                0.00    0.00       2/12          DoMarker [30]
                0.00    0.00       3/12          ReadSof [56]
                0.00    0.00       4/12          ReadDht [37]
[31]     0.0    0.00    0.00      12             bgetw [31]
-----------------------------------------------
                0.00    0.00       2/6           ReadDqt [42]
                0.00    0.00       4/6           ReadDht [37]
[32]     0.0    0.00    0.00       6             bpushc [32]
-----------------------------------------------
                0.00    0.00       4/4           ReadHuffman [38]
[33]     0.0    0.00    0.00       4             CodeTable [33]
-----------------------------------------------
                0.00    0.00       4/4           ReadHuffman [38]
[34]     0.0    0.00    0.00       4             DecoderTables [34]
-----------------------------------------------
                0.00    0.00       4/4           ReadDht [37]
```

```
[35]     0.0    0.00    0.00       4           MakeDhuff [35]
-----------------------------------------------
                0.00    0.00       4/4         ReadDht [37]
[36]     0.0    0.00    0.00       4           MakeXhuff [36]
-----------------------------------------------
                0.00    0.00       4/4         DoMarker [30]
[37]     0.0    0.00    0.00       4           ReadDht [37]
                0.00    0.00       8/16        mrtell [28]
                0.00    0.00       8/47426     bgetc [19]
                0.00    0.00       4/12        bgetw [31]
                0.00    0.00       4/4         MakeXhuff [36]
                0.00    0.00       4/4         MakeDhuff [35]
                0.00    0.00       4/4         ReadHuffman [38]
                0.00    0.00       4/6         bpushc [32]
                0.00    0.00       2/2         SetACHuffman [44]
                0.00    0.00       2/2         SetDCHuffman [45]
-----------------------------------------------
                0.00    0.00       4/4         ReadDht [37]
[38]     0.0    0.00    0.00       4           ReadHuffman [38]
                0.00    0.00     412/47426     bgetc [19]
                0.00    0.00       4/4         SizeTable [39]
                0.00    0.00       4/4         CodeTable [33]
                0.00    0.00       4/4         DecoderTables [34]
-----------------------------------------------
                0.00    0.00       4/4         ReadHuffman [38]
[39]     0.0    0.00    0.00       4           SizeTable [39]
-----------------------------------------------
                0.00    0.00       3/3         JpegDecodeFrame [1]
[40]     0.0    0.00    0.00       3           CloseIob [40]
-----------------------------------------------
                0.00    0.00       3/3         JpegDecodeFrame [1]
[41]     0.0    0.00    0.00       3           SeekEndIob [41]
-----------------------------------------------
                0.00    0.00       2/2         DoMarker [30]
[42]     0.0    0.00    0.00       2           ReadDqt [42]
                0.00    0.00     132/47426     bgetc [19]
                0.00    0.00       4/16        mrtell [28]
                0.00    0.00       2/12        bgetw [31]
                0.00    0.00       2/6           bpushc [32]
-----------------------------------------------
                0.00    0.00       2/2           JpegDecodeFrame [1]
[43]     0.0    0.00    0.00       2             ScreenAllMarker [43]
                0.00    0.00      14/14          ScreenMarker [29]
-----------------------------------------------
                0.00    0.00       2/2           ReadDht [37]
[44]     0.0    0.00    0.00       2             SetACHuffman [44]
-----------------------------------------------
                0.00    0.00       2/2           ReadDht [37]
[45]     0.0    0.00    0.00       2             SetDCHuffman [45]
-----------------------------------------------
                0.00    0.00       1/1           ReadSos [57]
[46]     0.0    0.00    0.00       1             CheckBaseline [46]
-----------------------------------------------
                0.00    0.00       1/1           ReadSos [57]
[47]     0.0    0.00    0.00       1             CheckScan [47]
-----------------------------------------------
                0.00    0.00       1/1           ReadSos [57]
[48]     0.0    0.00    0.00       1             CheckValidity [48]
-----------------------------------------------
                0.00    0.00       1/1           ReadSos [57]
[49]     0.0    0.00    0.00       1
MakeConsistentFileNames [49]

-----------------------------------------------
                0.00    0.00       1/1           ReadSof [56]
[50]     0.0    0.00    0.00       1
MakeConsistentFrameSize [50]

-----------------------------------------------
                0.00    0.00       1/1           main [3]
[51]     0.0    0.00    0.00       1             MakeFrame [51]
-----------------------------------------------
                0.00    0.00       1/1           main [3]
[52]     0.0    0.00    0.00       1             MakeImage [52]
-----------------------------------------------
                0.00    0.00       1/1           ReadSos [57]
```

```
[53]      0.0    0.00    0.00       1             MakeIob [53]
                 0.00    0.00      32/32          MakeXBuffer [27]
-----------------------------------------------
                 0.00    0.00       1/1           main [3]
[54]      0.0    0.00    0.00       1             MakeScan [54]
                 0.00    0.00       1/1           MakeScanFrequency [55]
-----------------------------------------------
                 0.00    0.00       1/1           MakeScan [54]
[55]      0.0    0.00    0.00       1             MakeScanFrequency [55]
-----------------------------------------------
                 0.00    0.00       1/1           DoMarker [30]
[56]      0.0    0.00    0.00       1             ReadSof [56]
                 0.00    0.00      11/47426       bgetc [19]
                 0.00    0.00       3/12          bgetw [31]
                 0.00    0.00       2/16          mrtell [28]
                 0.00    0.00       1/1
MakeConsistentFrameSize [50]
-----------------------------------------------
                 0.00    0.00       1/1           DoMarker [30]
[57]      0.0    0.00    0.00       1             ReadSos [57]
                 0.00    0.00      10/47426       bgetc [19]
                 0.00    0.00       2/16          mrtell [28]
                 0.00    0.00       1/12          bgetw [31]
                 0.00    0.00       1/1
MakeConsistentFileNames [49]
                 0.00    0.00       1/1           CheckValidity [48]
                 0.00    0.00       1/1           CheckBaseline [46]
                 0.00    0.00       1/1           CheckScan [47]
                 0.00    0.00       1/1           MakeIob [53]
                 0.00    0.00       1/2854        InstallIob [22]
                 0.00    0.00       1/1           ResetCodec [58]
-----------------------------------------------
                 0.00    0.00       1/1           ReadSos [57]
[58]      0.0    0.00    0.00       1             ResetCodec [58]
-----------------------------------------------
                 0.00    0.00       1/1           JpegDecodeFrame [1]
[59]      0.0    0.00    0.00       1             mrclose [59]
-----------------------------------------------
```

```
                0.00    0.00       1/1           JpegDecodeFrame [1]
[60]     0.0    0.00    0.00       1             mropen [60]
                0.00    0.00       1/1           pushstream [61]
-----------------------------------------------
                0.00    0.00       1/1           mropen [60]
[61]     0.0    0.00    0.00       1             pushstream [61]
-----------------------------------------------
Index by function name

  [26] BlockMoveTo            [49] MakeConsistentFileNames [45]
  SetDCHuffman
  [16] BoundIDctMatrix        [50] MakeConsistentFrameSize [39]
SizeTable
  [46] CheckBaseline          [35] MakeDhuff              [20]
  UseACHuffman
  [47] CheckScan              [51] MakeFrame              [21]
  UseDCHuffman
  [48] CheckValidity          [52] MakeImage               [6]
WriteBlock
   [5] ChenIDct               [53] MakeIob                 [8]
  WriteXBound
  [40] CloseIob               [54] MakeScan               [11]
  WriteXBuffer
  [33] CodeTable              [55] MakeScanFrequency      [19] bgetc
   [7] DecodeAC               [27] MakeXBuffer            [31] bgetw
  [18] DecodeDC               [36] MakeXhuff              [32] bpushc
   [9] DecodeHuffman          [17] PostshiftIDctMatrix     [3] main
  [34] DecoderTables          [37] ReadDht               (163) mcount
  [30] DoMarker               [42] ReadDqt                [14] megetb
  [24] FlushBuffer            [38] ReadHuffman            [13] megetv
  [25] FlushIob               [56] ReadSof                [59] mrclose
  [10] IQuantize              [57] ReadSos                [60] mropen
  [12] IZigzagMatrix          [58] ResetCodec             [28] mrtell
  [22] InstallIob             [43] ScreenAllMarker        [15] pgetc
  [23] InstallPrediction      [29] ScreenMarker           [61]
pushstream
   [1] JpegDecodeFrame        [41] SeekEndIob
   [2] JpegDecodeScan         [44] SetACHuffman
```

Appendix C – Motion JPEG Coder Source Code

File: jpeg.c (modifications)

```
/************************************************************
Copyright (C) 1990, 1991, 1993 Andy C. Hung, all rights reserved.
PUBLIC DOMAIN LICENSE: Stanford University Portable Video Research
Group. If you use this software, you agree to the following: This
program package is purely experimental, and is licensed "as is".
Permission is granted to use, modify, and distribute this program
without charge for any purpose, provided this license/ disclaimer
```

```
notice appears in the copies.  No warranty or maintenance is given,
either expressed or implied.  In no event shall the author(s) be
liable to you or a third party for any special, incidental,
consequential, or other damages, arising out of the use or inability
to use the program for any purpose (or the loss of data), even if we
have been advised of such possibilities.  Any public reference or
advertisement of this source code should refer to it as the Portable
Video Research Group (PVRG) code, and not by any author(s) (or
Stanford University) name.
*************************************************************/
/*
************************************************************
jpeg.c

This file contains the main calling routines for the JPEG coder.

************************************************************
*/

/*LABEL jpeg.c */

/* Include files. */

#include "tables.h"
#include "globals.h"
#include "mjpeg.h"
#ifdef SYSV
#include <sys/fcntl.h>
#endif

/*
  Define the functions to be used with ANSI prototyping.
  */

/*PUBLIC*/

int main();
static void JpegEncodeFrame();
static void JpegDecodeFrame();
static void JpegDecodeScan();
static void JpegLosslessDecodeScan();
static void Help();

void MJPEGEncoderMotionJPEG(void);
void MJPEGDecoderMotionJPEG(void);

extern void PrintImage();
extern void PrintFrame();
extern void PrintScan();
extern void MakeImage();
extern void MakeFrame();
extern void MakeScanFrequency();
extern void MakeScan();
extern void MakeConsistentFileNames();
extern void CheckValidity();
extern int CheckBaseline();
extern void ConfirmFileSize();
extern void JpegQuantizationFrame();
extern void JpegDefaultHuffmanScan();
extern void JpegFrequencyScan();
extern void JpegCustomScan();
extern void JpegEncodeScan();

extern void JpegLosslessFrequencyScan();
extern void JpegLosslessEncodeScan();

/*PRIVATE*/

/* These variables occur in the stream definition. */

extern int CleartoResync;
extern int LastKnownResync;
extern int ResyncEnable;
extern int ResyncCount;
extern int EndofFile;
extern int EndofImage;

/* Define the parameter passing structures. */
IMAGE *CImage=NULL;           /* Current Image variables structure */
FRAME *CFrame=NULL;           /* Current Frame variables structure */
SCAN *CScan=NULL;             /* Current Scan variables structure */

/* Define the MDU counters. */
int CurrentMDU=0;             /* Holds the value of the current MDU */
int NumberMDU=0;              /* This number is the number of MDU’s */

int count = 1; /* MJPEG, used to assign a .1 to the output filename
                           for the first decoding and .2 for the subsequent
                                             and the last. */


/* Define Lossless info */

int LosslessPredictorType=0;  /* The lossless predictor used */
int PointTransform=0;         /* This parameter affects the shifting in io.c */
```

```
/* How we break things up */

int ScanComponentThreshold=SCAN_COMPONENT_THRESHOLD;

/* Define the support/utility variables.*/
int ErrorValue=0;             /* Holds error upon return */
int Loud=MUTE;                /* Loudness gives level of debug traces */
int HuffmanTrace=NULL;         /* When set, dumps Huffman statistics */
int Notify=1;                 /* When set, gives image size feedback */
int Robust=0;
static int LargeQ=0;          /* When set, large quantization is enabled */
int ComponentIndex;

/* We default to the Chen DCT algorithm. */
vFunc *UseDct = ChenDct;       /* This is the DCT algorithm to use */
vFunc *UseIDct = ChenIDct;     /* This is the inverse DCT algorithm to use */

/* Add some macros to ease readability. */
#define DefaultDct (*UseDct)
#define DefaultIDct (*UseIDct)
 int mjpegFlag = 0;
/*START*/

/*BFUNC

main() is first called by the shell routine upon execution of the
program.

EFUNC*/

int main(argc,argv)
     int argc;
     char **argv;
{
  BEGIN("main");
  int i;
  int Oracle=0;     /* Oracle means that we use the lexer interactively */
  int frameCount;

  MakeImage();      /* Construct the image structures */
  MakeFrame();
  MakeScan();

  if (argc == 1)    /* No arguments then print help info */
    {
      Help();
      exit(-1);
    }
```

```
ComponentIndex=1;  /* Start with index 1 (Could be zero, but JFIF compat) */
for(i=1;i<argc;i++)  /* Else loop through all arguments. */
  {
    if (!strcmp(argv[i],"-JFIF"))
      CImage->Jfif=1;
    else if (!strcmp(argv[i],"-ci"))
      ComponentIndex=atoi(argv[++i]);
    else if (*(argv[i]) == '-')       /* Strip off first "dash" */
      {
        switch(*(++argv[i]))
          {
          case 'a':                      /* -a Reference DCT */
            UseDct = ReferenceDct;
            UseIDct = ReferenceIDct;
            break;
          case 'b':                      /* -b Lee DCT */
            UseDct = LeeDct;
            UseIDct = LeeIDct;
            break;
          case 'd':                      /* -d Decode */
            CImage->JpegMode = J_DECODER;
            break;
          case 'k':                      /* -k Lossless mode */
            CImage->JpegMode = J_LOSSLESS;
            CFrame->Type=3;
            LosslessPredictorType = atoi(argv[++i]);
            break;
          case 'f':
            switch(*(++argv[i]))
              {
              case 'w':                 /* -fw Frame width */
                CFrame->Width[ComponentIndex] =
                  atoi(argv[++i]);
                break;
              case 'h':                 /* -fh Frame height */
                CFrame->Height[ComponentIndex] =
                  atoi(argv[++i]);
                break;
              default:
                WHEREAMI();
                printf("Illegal option: f%c.\n",
                       *argv[i]);
                exit(ERROR_BOUNDS);
                break;
              }
            break;
          case 'i':
```

```
  switch(*(++argv[i]))
   {
   case ’w’:                /* -iw Image width */
     CFrame->GlobalWidth = atoi(argv[++i]);
     if ((mjpegFlag == 1) || (mjpegFlag == 2))
          mSequence.width = CFrame->GlobalWidth;
     break;
   case ’h’:                /* -ih Image height */
     CFrame->GlobalHeight = atoi(argv[++i]);
     if ((mjpegFlag == 1) || (mjpegFlag == 2))
          mSequence.height = CFrame->GlobalHeight;
     break;
   /* Get the number of frames in the Motion JPEG sequence. */
   case ’n’:
          mSequence.length = atoi(argv[++i]);
          printf("HERE LENGTH: %d", mSequence.length);
          for (frameCount = 0; frameCount < mSequence.length;
          frameCount++)
          {
                 mSequence.filename[frameCount] = argv[frameCount + 8];
                 mSequence.numMbw = (int)(mSequence.width / MBSIZE);
                 mSequence.numMbh = (int)(mSequence.height / MBSIZE);
                 mSequence.numMBs = mSequence.numMbw *
                 mSequence.numMbh;
          }
          break;
   default:
     WHEREAMI();
     printf("Illegal option: i%c.\n",
           *argv[i]);
     exit(ERROR_BOUNDS);
     break;
   }
  break;
case ’h’:                    /* -h horizontal frequency */
  CFrame->hf[ComponentIndex] =
   atoi(argv[++i]);
  break;
   case ’m’:
          printf("Motion JPEG Mode.\n\n");
            switch(*(++argv[i]))
            {
                 case ’e’:
                   mjpegFlag = 1;
                   break;
                 case ’d’:
                        mjpegFlag = 2;
                        break;
                        default:
                               printf("Wrong Parameteres.\n");
                          exit();
                          break;
                 }
                 break;
#ifndef PRODUCTION_VERSION
            case ’l’:                   /* -l loudness for debugging */
              Loud = atoi(argv[++i]);
              break;
#endif
            case ’n’:                   /* Set non-interleaved mode */
              ScanComponentThreshold=1;
              break;
            case ’o’:                   /* -o Oracle mode (input parsing)*/
              Oracle=1;
              break;
            case ’p’:
              CFrame->DataPrecision = atoi(argv[++i]);
              if (!CFrame->Type) CFrame->Type = 1;
              break;
            case ’r’:                   /* -r resynchronization */
              CFrame->ResyncInterval = atoi(argv[++i]);
              break;
            case ’q’:                   /* -q Q factor */
              if (*(++argv[i])==’l’) LargeQ=1;
              CFrame->Q = atoi(argv[++i]);
              break;
            case ’v’:                   /* -v vertical frequency */
              CFrame->vf[ComponentIndex] = atoi(argv[++i]);
              break;
            case ’s’:                   /* -s stream file name */
              CImage->StreamFileName = argv[++i];
              break;
            case ’t’:
              PointTransform=atoi(argv[++i]);
              break;
#ifndef PRODUCTION_VERSION
            case ’x’:                   /* -x trace */
              HuffmanTrace = 1;
              break;
#endif
            case ’u’:                   /* -u disable width/size output */
              Notify=0;
              break;
            case ’y’:
              Robust=1;
              break;
```

```
          case ’z’:                    /* -z use default Huffman */
            CImage->JpegMode |= J_DEFAULTHUFFMAN;
            break;
          default:
            WHEREAMI();
            printf("Illegal option in command line: %c.\n",
                   *argv[i]);
            exit(ERROR_BOUNDS);
            break;
          }
      }
    else                               /* If not a "-" then a filename */
      {
        CFrame->cn[CFrame->GlobalNumberComponents++]= ComponentIndex;
        if (!CFrame->vf[ComponentIndex])
          CFrame->vf[ComponentIndex]=1;
        if (!CFrame->hf[ComponentIndex])
          CFrame->hf[ComponentIndex]=1;
        CFrame->ComponentFileName[ComponentIndex] = argv[i];
        ComponentIndex++;
      }
  }

if (Oracle)            /* If Oracle set */
  {
    initparser();      /* Initialize interactive parser */
    parser();          /* parse input from stdin */
    exit(ErrorValue);
  }

/* Otherwise act on information */

if (!(GetFlag(CImage->JpegMode,J_DECODER)) &&  /* Check for files */
    (CFrame->GlobalNumberComponents == 0))
  {
    WHEREAMI();
    printf("No component file specified.\n");
    exit(ERROR_BOUNDS);
  }

if (CImage->StreamFileName == NULL)            /* Check for stream name */
  {
    if (CFrame->ComponentFileName[CFrame->cn[0]])  /* If doesn’t exist */
      {                                          /* Create one. */
        CImage->StreamFileName =
          (char *) calloc(strlen(CFrame->ComponentFileName[CFrame->cn[0]])+6,
                          sizeof(char));
        sprintf(CImage->StreamFileName,"%s.jpg",
                 CFrame->ComponentFileName[CFrame->cn[0]]);
        }
      else
        {
          WHEREAMI();
          printf("No stream filename.\n");
          exit(ERROR_BOUNDS);
        }
    }
  if (GetFlag(CImage->JpegMode,J_DECODER))       /* If decoder flag set then */
    {                                            /* decode frame. */
      JpegDecodeFrame();
    }
  else
    {
      if (!(CFrame->GlobalWidth) || !(CFrame->GlobalHeight)) /* Dimensions ? */
          {
            WHEREAMI();
            printf("Unspecified frame size.\n");
            exit(ERROR_BOUNDS);
          }
          if (mjpegFlag == 0)
          {
                swopen(CImage->StreamFileName,0);          /* Open output file, index 0*/
        JpegEncodeFrame();                         /* Encode the frame */
        swclose();                                 /* Flush remaining bits */
          }
    }
        if (mjpegFlag == 1)
          MJPEGEncoderMotionJPEG();
        if (mjpegFlag == 2)
                MJPEGDecoderMotionJPEG();
  exit(ErrorValue);
}

void MJPEGEncoderMotionJPEG(void)
{
        int frameCount, mbCount;
        struct MJPEGFrame current, newPrevious, previous, temp;
        char* name;

printf("LENGTH: %d", mSequence.length);
        for (frameCount = 0; frameCount < mSequence.length; frameCount++)
        {
                CFrame->ComponentFileName[1] = mSequence.filename[frameCount];
                CImage->StreamFileName = mSequence.filename[frameCount];
                CImage->StreamFileName = (char *)
calloc(strlen(CFrame->ComponentFileName[1]) + 6, sizeof(char));
```

```
                frame[frameCount].frFilename = CFrame->ComponentFileName[1];
                current = frame[frameCount];

                frame[frameCount].frFilename = CFrame->ComponentFileName[1];


                newPrevious = frame[frameCount];

                /* Set frames. */
                MJPEGgetFrame(&current);
                MJPEGsetMacroblocks(&current);
                MJPEGgetFrame(&newPrevious);
                MJPEGsetMacroblocks(&newPrevious);

                if (frameCount == 0)
                {
                        current.frMode = 0;
                        previous = newPrevious;
                        name = MJPEGsetFilename(current.frFilename);
                        MJPEGwriteFirstFrame(&current, name);
                }

                /* Video Coder. */
                if (frameCount > 0)
                {
                        MJPEGmotionEstimation(&previous, &current);
                        for (mbCount = 0; mbCount < mSequence.numMBs; mbCount++)
                        {
                                if (current.mb[mbCount].mode == 1)
                                {
                                        MJPEGsetResidualMatrix(mbCount, &previous,
&current);
                                    MJPEGdct(mbCount, &current);
                                        MJPEGquantization(mbCount, &current);
                                        MJPEGzigzag(mbCount, &current);
                                        MJPEGrunLength(mbCount, &current);
                                }
                        }

                        previous = newPrevious;
                        name = MJPEGsetFilename(current.frFilename);
                        MJPEGwriteFrame(&current, current.frMode, name);
                }
        }

}
```

```
void MJPEGDecoderMotionJPEG(void)
{
        int count, filename, mbCount;
        char* current;
        double psnr;
        printf("LENGTH: %d", mSequence.length);
        for (count = 0; count < mSequence.length; count++)
        {
                current = mSequence.filename[count];

                MJPEGsetVirtualMBs(current);
                printf("%d PPPP\n", count);

                if (count == 0)
                {
                        MJPEGsetVirtualMBs(current);
                        MJPEGsetMBPositions();
                        MJPEGmakeRawFrame();
                        filename = MJPEGsetDecoderFilename(current);
                        MJPEGwriteVirtualFrame(filename);
                }
                if (count > 0)
                {
                        MJPEGgetMBModes(current);

                        for (mbCount = 0; mbCount < mSequence.numMBs; mbCount++)
                        {
                        printf("PASSED\n");
                                if (df.dMB[mbCount].mode == 1)
                                {
                                        MJPEGinvRunLength(mbCount);
                                        MJPEGinvZigzag(mbCount);

                                        MJPEGInvQuantization(mbCount);
                                        MJPEGinvDct(mbCount);
                                        MJPEGgetResidualMatrix(mbCount);
                                        MJPEGupdateVirtualFrame(mbCount);
                                        filename = MJPEGsetDecoderFilename(current);
                                        MJPEGwriteVirtualFrame(filename);
                                        filename = MJPEGgetPSNRFilename(current);
                                        psnr = MJPEGcalculatePSNR(filename);
                                }
                        }
                }
        }
}
```

```
/*END*/
```

File: mjpegenc.c

```
/**
 ** Module: mjpegenc.c: Motion JPEG Encoder.
 **/

#include "mjpeg.h"
#include <stdio.h>
#include <math.h>

/* 3D VLC Table (H.263). */
int Table3DVLC[103][5] = {
/* Last, Run, Level, Bits, Code Word */
                                                 0,  0,  1,  3,  2,
                                                 0,  0,  2,  5,  15,
                                                 0,  0,  3,  7,  21,
                                                 0,  0,  4,  8,  23,
                                                 0,  0,  5,  9,  31,
                                                 0,  0,  6,  10, 37,
                                                 0,  0,  7,  10, 36,
                                               0,  0,  8,  11, 33,
                                          0,  0,  9,  11, 32,
                                                 0,  0,  10, 12, 7,
                                                 0,  0,  11, 12, 6,
                                                 0,  0,  12, 12, 32,
                                                 0,  1,  1,  4,  6,
                                                 0,  1,  2,  7,  20,
                                                 0,  1,  3,  9,  30,
                                                 0,  1,  4,  11, 15,
                                                 0,  1,  5,  12, 33,
                                                 0,  1,  6,  13, 80,
                                                 0,  2,  1,  5,  14,
                                                 0,  2,  2,  9,  29,
                                                 0,  2,  3,  11, 14,
                                                 0,  2,  4,  13, 81,
                                                 0,  3,  1,  6,  13,
                                                 0,  3,  2,  10, 35,
                                                 0,  3,  3,  11, 13,
                                                 0,  4,  1,  6,  12,
                                                 0,  4,  2,  10, 34,
                                                 0,  4,  3,  13, 82,
                                                 0,  5,  1,  6,  11,
                                                 0,  5,  2,  11, 12,
                                                 0,  5,  3,  13, 83,
                                                 0,  6,  1,  7,  19,
0,  6,  2,  11, 11,
0,  6,  3,  13, 84,
0,  7,  1,  7,  18,
0,  7,  2,  11, 10,
0,  8,  1,  7,  17,
0,  8,  2,  11, 9,
0,  9,  1,  7,  16,
0,  9,  2,  11, 8,
0,  10, 1,  8,  22,
0,  10, 2,  13, 85,
0,  11, 1,  8,  21,
0,  12, 1,  8,  20,
0,  13, 1,  9,  28,
0,  14, 1,  9,  27,
0,  15, 1,  10, 33,
0,  16, 1,  10, 32,
0,  17, 1,  10, 31,
0,  18, 1,  10, 30,
0,  19, 1,  10, 29,
0,  20, 1,  10, 28,
0,  21, 1,  10, 27,
0,  22, 1,  10, 26,
0,  23, 1,  12, 34,
0,  24, 1,  12, 35,
0,  25, 1,  13, 86,
0,  26, 1,  13, 87,
1,  0,  1,  5,  7,
1,  0,  2,  10, 25,
1,  0,  3,  12, 5,
1,  1,  1,  7,  15,
1,  1,  2,  12, 4,
1,  2,  1,  7,  14,
1,  3,  1,  7,  13,
1,  4,  1,  7,  12,
1,  5,  1,  8,  19,
1,  6,  1,  8,  18,
1,  7,  1,  8,  17,
1,  8,  1,  8,  16,
1,  9,  1,  9,  26,
1,  10, 1,  9,  25,
1,  11, 1,  9,  24,
1,  12, 1,  9,  23,
1,  13, 1,  9,  22,
1,  14, 1,  9,  21,
1,  15, 1,  9,  20,
1,  16, 1,  9,  19,
1,  17, 1,  10, 24,
1,  18, 1,  10, 23,
                                              1,  19, 1,  10, 22,
                                              1,  20, 1,  10, 21,
                                              1,  21, 1,  10, 20,
                                              1,  22, 1,  10, 19,
                                              1,  23, 1,  10, 18,
                                              1,  24, 1,  10, 17,
                                              1,  25, 1,  11, 7,
                                              1,  26, 1,  11, 6,
                                              1,  27, 1,  11, 5,
                                              1,  28, 1,  11, 4,
                                              1,  29, 1,  12, 36,
                                              1,  30, 1,  12, 37,
                                              1,  31, 1,  12, 38,
                                              1,  32, 1,  12, 39,
                                              1,  33, 1,  13, 88,
                                              1,  34, 1,  13, 89,
                                              1,  35, 1,  13, 90,
                                              1,  36, 1,  13, 91,
                                              1,  37, 1,  13, 92,
                                              1,  38, 1,  13, 93,
                                              1,  39, 1,  13, 94,
                                              1,  40, 1,  13, 95,
                                             -1,  -1, -1,  7,  3   };

#define LAST  0 /* Holds the index for LAST information. */
#define RUN   1 /* Holds the index for RUN information. */
#define LEVEL 2 /* Holds the index for LEVEL information. */
#define NBITS 3 /* Holds the index for BITS information. */
#define CODEW 4 /* Holds the index for CODE WORD information. */

/**
 ** Function:     void getFrame(struct MJPEGFrame* frame);
 ** Description:  This function opens a file containing a raw image and reads the
                      values
 **              of all the pixels inside it.
 ** Parameters:   A pointer to an MJPEGFrame structure.
 ** Return Value: NONE.
 **/
void MJPEGgetFrame(struct MJPEGFrame* frame)
{
    FILE* fp; /* The image file. */
    int iw,ih; /* Width iw and height ih in the raw image. */

       if ((mSequence.width % MBSIZE != 0) || (mSequence.height % MBSIZE != 0) ||
              (mSequence.width < MBSIZE) || (mSequence.height < MBSIZE))
       /* Detect an error if image dimensions cannot contain an exact number of
          macroblocks or image dimensions are smaller than a macroblock. */
       {
              printf("Incorrect image dimensions.\n");
              /* Display the error on screen. */
              exit(1); /* Exit with an unsuccessful parameter. */
       }

    if ((fp = fopen(frame->frFilename,"r")) == NULL)
       /* Open the image file. Detect error. */
       {
              printf("Cannot open file %s.\n", frame->frFilename); /* Display the error
on screen. */
              exit(1); /* Exit with an unsuccessful parameter. */
       }

       for (ih = 0; ih < mSequence.height; ih++)
       /* Take one row at a time until the full height is reached. */
    {
       for (iw = 0; iw < mSequence.width; iw++)
              /* Scan the row until the full width is reached. */
       {
           unsigned char val;

           fread(&val, 1, sizeof(unsigned char), fp); /* Read pixels. */
           frame->frValue[iw][ih] = (int)(val); /* Assign values. */

           if (frame->frValue[iw][ih] < 0.0 || frame->frValue[iw][ih] > 255.0)
                      /* Detect error if value is out of range. */
                      {
               printf("Grey level range error.\n");
                      /* Display the error on screen. */
                      exit(1); /* Exit with an unsuccessful parameter. */
                      }
       }
       }
       fclose(fp); /* Close the opened image file. */
}

/**
 ** Function:     void setMacroblocks(struct MJPEGFrame* frame);
 ** Description:  This function sets the macroblocks, 16x16 arrays of pixels,
 **               inside a frame. The function assigns macroblocks from left
 **               to right and top to bottom.
 ** Parameters:   A pointer to an MJPEGFrame structure.
 ** Return Value: NONE.
 **/
void MJPEGsetMacroblocks(struct MJPEGFrame* frame)
{
       int i, j, iw, ih, initw, tempA,
              startw = 0, starth = 0,
               numMbw, numMbh,
               mbCount; /* Counter of macroblocks */

    mbCount = 0; /* Initialize Macroblock counter. */

       /* Find the number of macroblocks that fit in width and height. */
       numMbw = (int)(mSequence.width / MBSIZE);
       numMbh = (int)(mSequence.height / MBSIZE);

       /* Scan image and set macroblocks from top to bottom and then
          left to right. */
       for (i = 0; i < numMbw; i++)
       {
               for (j = 0; j < numMbh; j++)
               {
                       /* Initial macroblock values. */
                       frame->mb[mbCount].sum = 0;
                       frame->mb[mbCount].mean = 0;

                       for (ih = 0; ih < MBSIZE; ih++)
/* Take one row at a time until the full height of macroblock is reached. */
                       {
                               initw = startw;

                               for (iw = 0; iw < MBSIZE; iw++)
       /* Scan the row until the full width of macroblock is reached. */
                               {
                                       if ((iw == 0) && (ih == 0))
                                       /* Set top left pixel coordinates of the
                                               macroblock. */
                                       {
                                               frame->mb[mbCount].xBegin = startw;
                                               frame->mb[mbCount].yBegin = starth;
                                       }

                                       if ((iw ==15) && (ih == 15))
                       /* Set bottom right pixel coordinates of the macroblock. */
                                       {
                                               frame->mb[mbCount].xEnd = startw;
                                               frame->mb[mbCount].yEnd = starth;
                                       }
                                       /* Assign values to macroblock pixels. */
                                       frame->mb[mbCount].mbValue[iw][ih] =
                                       frame->frValue[startw][starth];

                                       frame->mb[mbCount].sum +=
                                       frame->mb[mbCount].mbValue[iw][ih];
                                       startw++;
```

```
                              }
                              startw = initw;
                              starth++;
                      }
                                                    frame->mb[mbCount].mean =
                              (int)(frame->mb[mbCount].sum / 255.0);

                                                    tempA = frame->mb[mbCount].sum -
                              (frame->mb[mbCount].mean * 256);
                      frame->mb[mbCount].a = getAbs(tempA);

                      mbCount++; /* Go to the next macroblock. */
              }
              startw = startw + MBSIZE;
              starth = 0;
      }

      /* Set the total number of macroblocks assigned in the frame. */
      mSequence.numMBs = mbCount;
}


/**
 ** Function:     void getFrX(int x, int mbNum, struct MJPEGFrame* frame);
 ** Description:  This function takes a horizontal coordinate of a macroblock
 **               and returns a coordinate value relative to the frame.
 ** Parameters:   The horizontal coordinate x. The macroblock number mbNum.
 **               A pointer to an MJPEGFrame structure.
 ** Return Value: A horizontal coordinate.
 **/
int getFrX(int x, int mbNum, struct MJPEGFrame* frame)
{
       /* The horizontal coordinate at the begining of the macroblock + x. */
       return (frame->mb[mbNum].xBegin + x);
}

/**
 ** Function:     void getFrY(int y, int mbNum, struct MJPEGFrame* frame);
 ** Description:  This function takes a vertical coordinate of a macroblock
 **               and returns a coordinate value relative to the frame.
 ** Parameters:   The vertical coordinate x. The macroblock number mbNum.
 **               A pointer to an MJPEGFrame structure.
 ** Return Value: A horizontal coordinate.
 **/
int getFrY(int y, int mbNum, struct MJPEGFrame* frame)
{
       /* The vertical coordinate at the beginning of the macroblock + y. */
       return (frame->mb[mbNum].yBegin + y);
```

```
}

/**
 ** Function:     void setSearchWindow(struct MJPEGFrame* c, int mbNum,
 **                     int* ctlx, int* ctly, int* cbrx, int* cbry);
 ** Description:  Sets the search window of each macroblock.
 ** Parameters:   A pointer to an MJPEGFrame structure, the macroblock mbNum,
 **               the top-left coordinates ctlx-ctly and the bottom-right
 **               coordinates cbrx-cbry.
 ** Return Value: NONE.
 **/
void setSearchWindow(struct MJPEGFrame* c, int mbNum, int* ctlx, int* ctly, int* cbrx,
int* cbry)
{
       int temp;

       /* Set the top left coordinates of the search window. */
       temp = getFrX(0, mbNum, c); /* Get the top left X coordinate of the macroblock. */
       if (temp != 0) /* If there are other macroblocks on its left ... */
              *ctlx = temp - 16; /* ... Set the starting X coordinate of the search
window. */
       else
              *ctlx = 0; /* Otherwise the starting X coordinate of the search window is
0. */

       temp = getFrY(0, mbNum, c); /* Get the top left Y coordinate of the macroblock. */
       if (temp != 0) /* If there are other macroblocks above ... */
              *ctly = temp - 16; /* ... Set the starting Y coordinate of the search
window. */
       else
              *ctly = 0; /* Otherwise the starting Y coordinate of the search window is
0. */

       /* Set the bottom right coordinates of the search window. */
       temp = getFrX(15, mbNum, c); /* Get the bottom right X coordinate of the
macroblock. */
       if (temp != (mSequence.width - 1)) /* If there are other macroblocks on its right
... */
              *cbrx = temp + 16; /* ... Set the ending X coordinate of the search
window. */
       else
              *cbrx = (mSequence.width - 1); /* Otherwise the ending X coordinate of the
search window is width-1. */

       temp = getFrY(15, mbNum, c); /* Get the bottom right Y coordinate of the
macroblock. */
       if (temp != (mSequence.height - 1)) /* If there are other macroblocks below ... */
```

```
                *cbry = temp + 16; /* ... Set the ending Y coordinate of the search
window. */
        else
                *cbry = (mSequence.height - 1); /* Otherwise the ending Y coordinate of
the search window is height-1. */

        //printf("Frame:%s Macroblock:%d Search Window[(%d,%d), (%d,%d)]\n", c-
>frFilename, mbNum, *ctlx, *ctly, *cbrx, *cbry);
}



/**
 ** Function:     void MJPEGmotionEstimation(struct MJPEGFrame* p, struct MJPEGFrame* c);
 ** Description:  Performs motion estimation between the current and previous frame.
 ** Parameters:   Two MJPEGFrame pointers p and c to the previous and current frame.
 ** Return Value: NONE.
 **/
void MJPEGmotionEstimation(struct MJPEGFrame* p, struct MJPEGFrame* c)
{
        int count, l, t, r, b, ih, iw, jh, jw, sad = -100, mx, my, minSAD = 1e+5, motion =
0, prevSAD = 1e+5;
        struct MJPEGMacroblock shiftMb;
        char* sMode;

        c->frMode = INTRA; /* The frame is initialized to INTRA. */

        for (count = 0; count < mSequence.numMBs; count++)
        {
                        setSearchWindow(c, count, &l, &t, &r, &b);

                        for (ih = t; ih < (b - MBSIZE + 2); ih++)
                        {
                                for (iw = l; iw < (r - MBSIZE + 2); iw++)
                                {
                                        for (jh = 0; jh < 16; jh++)
                                        {
                                                for (jw = 0; jw < 16; jw++)
                                                {
                                                        sad += getAbs(c-
>mb[count].mbValue[jw][jh] - p->frValue[iw + jw][ih + jh]);

                                                }
                                        }

                                        if ((sad < prevSAD) && (sad < minSAD))
                                        {
                                                minSAD = sad; /* Keep the minimum SAD value
found. */
```

```
                                        mx = iw;
                                        my = ih;
                                }

                                prevSAD = sad;
                                sad = - 100;
                        }
                }
        /* Set motion vector. */
        if  (c->mb[count].a > minSAD - 500)
        /* Macroblock satisfies the condition, INTER. */
        {
                c->frMode = INTER; /* Motion found therefore frame is INTER. */
                c->mb[count].mode = INTER;
                c->mb[count].motionVector.xDiff = c->mb[count].xBegin - mx;
                c->mb[count].motionVector.yDiff = c->mb[count].yBegin - my;

                if (c->mb[count].motionVector.xDiff < 0)
                        c->mb[count].motionVector.sx = 1;
                else
                        c->mb[count].motionVector.sx = 0;

                if (c->mb[count].motionVector.yDiff < 0)
                        c->mb[count].motionVector.sy = 1;
                else
                        c->mb[count].motionVector.sy = 0;
        }
        else
        /* If not then the macroblock is INTRA. */
        {
                c->mb[count].mode = INTRA;
                c->mb[count].motionVector.xDiff = 0;
                c->mb[count].motionVector.yDiff = 0;
                c->mb[count].motionVector.sx = 0;
                c->mb[count].motionVector.sy = 0;
        }

        if (c->mb[count].mode == 0)
                sMode = "INTRA";
        else
                sMode = "INTER";
        /* Initilize to a large value. */
        prevSAD = 1e+5;
        minSAD = 1e+5;
        motion = 0;
    }
}
```

```
/**
 ** Function:     int getAbs(int num);
 ** Description:  Sets an absolute value of a number.
 ** Parameters:   An integer num.
 ** Return Value: The absolute value of num.
 **/
int getAbs(int num)
{
        /* Return an absolute value. */
    if (num < 0)
        return (-num);
    else
      return num;
}


/**
 ** Function:     void MJPEGsetResidualMatrix(int mbNum, struct MJPEGFrame* p,
 **               struct MJPEGFrame* c);
                       ** Description:  Sets the residual matrix of an INTER macroblock.
                    It uses the current
 **               macroblock and the best much found by motion estimation in the previous
 **               frame.
 ** Parameters:   A macroblock mbNum. Two MJPEGFrame pointers p and c to the previous
 **               and current frame.
 ** Return Value: NONE.
 **/
void MJPEGsetResidualMatrix(int mbNum, struct MJPEGFrame* p, struct MJPEGFrame* c)
{
        int ih, iw;

        for (ih = 0; ih < MBSIZE; ih++)
        {
                //printf("\n");
                for (iw = 0; iw < MBSIZE; iw++)
                {
                        c->mb[mbNum].rMatrix[iw][ih] =
                        c->mb[mbNum].mbValue[iw][ih] -
                        p->mb[mbNum].mbValue[iw - c->mb[mbNum].motionVector.xDiff][ih -
                                             c->mb[mbNum].motionVector.yDiff];
                        //printf("%3d ", c->mb[mbNum].rMatrix[iw][ih]);
                }
        }
}


/**
 ** Function:     double getC(int num);
 ** Description:  Calculates the value of the C coefficient needed by the DCT.
 ** Parameters:   An integer value num.
 ** Return Value: The DCT C coefficient.
 **/
double getC(int num)
{
        /* Standard DCT C coefficient rule. */
        if (num == 0)
                return (1.0 / sqrt(2.0));
        else
                return 1.0;
}

/**
 ** Function:     void MJPEGdct(int mbNum, struct MJPEGFrame* c);
 ** Description:  Calculates the DCT values of a 16 by 16 macroblock.
 ** Parameters:   A macroblock mbNum. An MJPEGFrame pointer c to the current frame.
 ** Return Value: NONE.
 **/
void MJPEGdct(int mbNum, struct MJPEGFrame* c)
{
        int u, v, i, j;
        double temp;

        /* Standard DCT transformation formula. */
        for (v = 0; v < MBSIZE; v++)
        {
                for (u = 0; u < MBSIZE; u++)
                {
                        for (i = 0; i < MBSIZE; i++)
                        {
                                for (j = 0; j < MBSIZE; j++)
                                {
                                        temp += (c->mb[mbNum].rMatrix[i][j] *
                                                 cos((2.0 * i + 1.0) * (u * PI) / 16.0) *
                                                        cos((2.0 * j + 1.0) * (v * PI) /
16.0));
                                }
                        }

                        /* Store each DCT value obtained. */
                        c->mb[mbNum].dctMatrix[u][v] = temp * (0.25 * getC(u) * getC(v));
                        temp = 0.0;

                        /* Output */
                        //printf("%.1f\t", c->mb[mbNum].dctMatrix[u][v]);
                }
                /* Output */
                //printf("\n");
        }
}

/**
 ** Function:     void MJPEGquantization(int mbNum, struct MJPEGFrame* c);
 ** Description:  Performs quantization to the DCT values of the macroblock.
 ** Parameters:   A macroblock mbNum. An MJPEGFrame pointer c to the current frame.
 ** Return Value: NONE.
 **/
void MJPEGquantization(int mbNum, struct MJPEGFrame* c)
{
        int ih, iw;
        double Q = 5.0; /* Q is set to 5. */

        for (ih = 0; ih < MBSIZE; ih++)
        {
                for (iw = 0; iw < MBSIZE; iw++)
                {
                        /* Divide all values by ( 2*Q ). */
                                                            c->mb[mbNum].qMatrix[iw][ih]
                                        = (int)(c->mb[mbNum].dctMatrix[iw][ih] / (2.0 * Q));
                }
        }
}

/**
 ** Function:     void MJPEGzigzag(int mbNum, struct MJPEGFrame* c);
 ** Description:  Transforms a 2-D matrix to a 1-D matrix using the zig-zag method.
 ** Parameters:   A macroblock mbNum. An MJPEGFrame pointer c to the current frame.
 ** Return Value: NONE.
 **/
void MJPEGzigzag(int mbNum, struct MJPEGFrame* c)
{
        /* Index of value coordinates inside the 2-D matrix to be followed. */
        int zzIndex[256][2] = {  0,0,     1,0,     0,1,     0,2,
                                  1,1,     2,0,     3,0,     2,1,
                                  1,2,     0,3,     0,4,     1,3,
                                  2,2,     3,1,     4,0,     5,0,
                                  4,1,     3,2,     2,3,     1,4,
                                  0,5,     0,6,     1,5,     2,4,
                                                      3,3,     4,2,     5,1,     6,0,
                                                      7,0,     6,1,     5,2,     4,3,
                                                      3,4,     2,5,     1,6,     0,7,
                                                      0,8,     1,7,     2,6,     3,5,
                                                      4,4,     5,3,     6,2,     7,1,
                                                       8,0,     9,0,     8,1,     7,2,
                                                      6,3,     5,4,     4,5,     3,6,
       2,7,     1,8,     0,9,    0,10,
       1,9,     2,8,     3,7,     4,6,
       5,5,     6,4,     7,3,     8,2,
       9,1,    10,0,    11,0,    10,1,
       9,2,     8,3,     7,4,     6,5,
       5,6,     4,7,     3,8,     2,9,
      1,10,    0,11,    0,12,    1,11,
      2,10,     3,9,     4,8,     5,7,
       6,6,     7,5,     8,4,     9,3,
      10,2,    11,1,    12,0,    13,0,
      12,1,    11,2,    10,3,     9,4,
       8,5,     7,6,     6,7,     5,8,
      4,9,    3,10,    2,11,    1,12,
      0,13,    0,14,    1,13,    2,12,
      3,11,    4,10,     5,9,     6,8,
      7,7,     8,6,     9,5,    10,4,
      11,3,    12,2,    13,1,    14,0,
      15,0,    14,1,    13,2,    12,3,
      11,4,    10,5,     9,6,     8,7,
       7,8,     6,9,    5,10,    4,11,
      3,12,    2,13,    1,14,    0,15,
      1,15,    2,14,    3,13,    4,12,
      5,11,    6,10,     7,9,     8,8,
       9,7,    10,6,    11,5,    12,4,
      13,3,    14,2,    15,1,    15,2,
      14,3,    13,4,    12,5,    11,6,
      10,7,     9,8,     8,9,    7,10,
      6,11,    5,12,    4,13,    3,14,
      2,15,    3,15,    4,14,    5,13,
      6,12,    7,11,    8,10,     9,9,
      10,8,    11,7,    12,6,    13,5,
      14,4,    15,3,    15,4,    14,5,
      13,6,    12,7,    11,8,    10,9,
      9,10,    8,11,    7,12,    6,13,
      5,14,    4,15,    5,15,    6,14,
      7,13,    8,12,    9,11,   10,10,
      11,9,    12,8,    13,7,    14,6,
      15,5,    15,6,    14,7,    13,8,
      12,9,   11,10,   10,11,    9,12,
      8,13,    7,14,    6,15,    7,15,
      8,14,    9,13,   10,12,   11,11,
      12,10,    13,9,    14,8,    15,7,
      15,8,    14,9,   13,10,   12,11,
      11,12,   10,13,    9,14,    8,15,
      9,15,   10,14,   11,13,   12,12,
13,11,   14,10,    15,9,   15,10,
14,11,   13,12,   12,13,   11,14,
10,15,   11,15,   12,14,   13,13,
                                                14,12,   15,11,   15,12,   14,13,
                                                13,14,   12,15,   13,15,   14,14,
                                                15,13,   15,14,   14,15,   15,15 };

        int iw, count = 0;

        for (iw = 0; iw < (MBSIZE * MBSIZE); iw++)
        {
                /* Set the zig-zag matrix by following the index coordinates. */
                c->mb[mbNum].zzMatrix[count] =
                c->mb[mbNum].qMatrix[zzIndex[iw][0]][zzIndex[iw][1]];

                count++;
        }
        c->mb[mbNum].zzMatrix[count] = -1;
}

/**
 ** Function:     void MJPEGrunLength(int mbNum, struct MJPEGFrame* c);
 ** Description:  Perform run length coding on the zig-zag matrix.
 ** Parameters:   A macroblock mbNum. An MJPEGFrame pointer c to the current frame.
 ** Return Value: NONE.
 **/
void MJPEGrunLength(int mbNum, struct MJPEGFrame* c)
{
        int i = 0, j, prev, curr, seq = 0, count = 0, index = 0;

        c->mb[mbNum].runLength[index][LEVEL] = prev = c->mb[mbNum].zzMatrix[0]; /* Set
first value ... */
        c->mb[mbNum].runLength[index][RUN] = 0;
        c->mb[mbNum].runLength[index][LAST] = 0;
                for (j = 0; j < 103; j++)
                {
                        if ((c->mb[mbNum].runLength[index][RUN] == Table3DVLC[j][RUN]) &&
        (getAbs(c->mb[mbNum].runLength[index][LEVEL]) == Table3DVLC[j][LEVEL]) &&
                (c->mb[mbNum].runLength[index][LAST] == Table3DVLC[j][LAST]))
                                        {
                                                c->mb[mbNum].runLength[index][CODEW] =
MJPEGsetVLC(c->mb[mbNum].runLength[index][LEVEL], Table3DVLC[j][CODEW]);
                                        }
                                }
        //printf("##index=%3d run = %d\t last = %d\t level = %d code =%d\n", index,
                //              c->mb[mbNum].runLength[index][RUN],
                        //      c->mb[mbNum].runLength[index][LAST],
        //                      c->mb[mbNum].runLength[index][LEVEL],
                //              c->mb[mbNum].runLength[index][CODEW]);
        index++;
        /* ... go on with the rest. */
        do
        {
                i++;
                curr = c->mb[mbNum].zzMatrix[i];
                if (prev == 0)
                {
                        seq = 1;
                        count++;
                        prev = curr;
                }
         if (curr != 0)
                {
                                if (i == 255)
                                /* Set the indication of last value. */
                                        c->mb[mbNum].runLength[index][LAST] = 1;
                                else
                                        c->mb[mbNum].runLength[index][LAST] = 0;

                                c->mb[mbNum].runLength[index][RUN] = count;
                                c->mb[mbNum].runLength[index][LEVEL] = curr;

                                for (j = 0; j < 103; j++)
                                {
                                        if ((c->mb[mbNum].runLength[index][RUN] ==
Table3DVLC[j][RUN]) &&
                                                (getAbs(c-
>mb[mbNum].runLength[index][LEVEL]) == Table3DVLC[j][LEVEL]) &&
                                                (c->mb[mbNum].runLength[index][LAST] ==
Table3DVLC[j][LAST]))
                                        {
                                                c->mb[mbNum].runLength[index][CODEW] =
MJPEGsetVLC(c->mb[mbNum].runLength[index][LEVEL], Table3DVLC[j][CODEW]);
                                        }
                                }
                                if (i == 256)
                                {
                                        c->mb[mbNum].runLength[index][CODEW] = 3;
                                        c->mb[mbNum].runLength[index][LAST] = 1;
                                }
                //      printf("##index=%3d run = %d\t last = %d\t level = %d code =%d\n",
index,
                //              c->mb[mbNum].runLength[index][RUN],
                //              c->mb[mbNum].runLength[index][LAST],
                //              c->mb[mbNum].runLength[index][LEVEL],
                //              c->mb[mbNum].runLength[index][CODEW]);
                                index++;
                            count = 0;
                                prev = curr;
                }
                else if (curr == 0)
                {
                        prev = curr;
                }

        }
        while (i != 256);
}

/**
 ** Function:     int MJPEGsetVLC(int rl, int vlc);
 ** Description:  Sets a correct code word. Function checks whether the level is
 **               positive or negative gives s the appropriate value and calculates
 **               the code word.
 ** Parameters:   The level rl and the vlc value of the 3D-VLC table.
 ** Return Value: The calculated code word.
 **/
int MJPEGsetVLC(int rl, int vlc)
{
        int s;

        if (rl >= 0)
        {
                s = 0;
                vlc = vlc << 1; /* Shift one position. */
        }
        else
        {
                vlc = abs(vlc);
                s = 1;
                /* Shift one position and add one. */
                vlc = vlc << 1;
                vlc++;
        }

        return vlc;
}

/**
 ** Function:     char* MJPEGsetFilename(char* c);
 ** Description:  Take the filename of the current frame and construct the
 **               appropriate motion JPEG filename.
 ** Parameters:   The filename of the current frame c.
 ** Return Value: The motion JPEG filename.
 **/
char* MJPEGsetFilename(char* c)
{
        int i, dotPosition;
        char curr[24];
        char* name;

        dotPosition = strcspn(c, "."); /* Find the first dot in the filename. */
        for (i = 0; i < dotPosition; i++)
                curr[i] = c[i]; /* Take the part before the dot. */
        curr[i] = '\0';
        strcat(curr, ".mjg"); /* Make motion JPEG filename. */
        strcpy(name, curr);

    return name;
}


/**
 ** Function:     void MJPEGwriteFrame(struct MJPEGFrame* c, char mode, char* name);
 ** Description:  Makes a motion JPEG file.
 ** Parameters:   An MJPEGFrame pointer c to the current frame. The mode of the frame
 **               and the output filename.
 ** Return Value: NONE.
 **/
void MJPEGwriteFrame(struct MJPEGFrame* c, char mode, char* name)
{
        FILE* fpMJPEG; /* The Output Motion JPEG file. */
        char identity[5] = "MJPEG";
        char mID = 'M';
        int count, i, codew, run, last, mbMode, xv, yv, xs, ys;

//      printf("MJPEGwriteFrame\n");
        fpMJPEG = fopen(name, "wb"); /* Create a binary file for writing. */
        /* Write MJPEG identity. */
        fwrite(&identity, sizeof(char[5]), 1, fpMJPEG);
        /* Write whether the frame is INTRA=0 or INTER=1. */
    fwrite(&mode, sizeof(char), 1, fpMJPEG);

        for (count = 0; count < mSequence.numMBs; count++)
        {
                if (c->mb[count].mode == INTER)
                {
                        /* Write Macroblock Identifier, M */
                fwrite(&mID, sizeof(char), 1, fpMJPEG);
                        /* Write that frame is INTER=1. */
                        mbMode = 1;
                fwrite(&mbMode, sizeof(char), 1, fpMJPEG);


                        xs = c->mb[count].motionVector.sx;
                        fwrite(&xs, sizeof(char), 1, fpMJPEG);

                        xv = getAbs(c->mb[count].motionVector.xDiff);
                        fwrite(&xv, sizeof(char), 1, fpMJPEG);

                        ys = c->mb[count].motionVector.sy;
                        fwrite(&ys, sizeof(char), 1, fpMJPEG);

                        yv = getAbs(c->mb[count].motionVector.yDiff);
                        fwrite(&yv, sizeof(char), 1, fpMJPEG);
                        i = -1;
                        do
                        {
                                i++;
                                /* Write RUN information. */
                                run = c->mb[count].runLength[i][RUN];
                                fwrite(&run, sizeof(char), 1, fpMJPEG);
                                /* Write LEVEL information. */
                                codew = c->mb[count].runLength[i][CODEW];
                                fwrite(&codew, sizeof(char), 1, fpMJPEG);
                                /* Write LAST information. */
                                last = c->mb[count].runLength[i][LAST];
                                fwrite(&last, sizeof(char), 1, fpMJPEG);
                //              printf("i = %d run = %xd codew = %xd last = %xd\n",
                        //      i, run, codew, last);
                        }
                        while (c->mb[count].runLength[i][LAST] == 0);
                }
                else
                {
                        /* Write Macroblock Identifier, M */
                fwrite(&mID, sizeof(char), 1, fpMJPEG);
                        /* Write that frame is INTRA=0. */
                        mbMode = 0;
                fwrite(&mbMode, sizeof(char), 1, fpMJPEG);
                }
        }

        fclose(fpMJPEG);
}


/**
 ** Function:     void MJPEGwriteFirstFrame(struct MJPEGFrame* c, char* name);
 ** Description:  Makes a motion JPEG file from the first frame in the sequence.
 ** Parameters:   An MJPEGFrame pointer c to the current frame and the output filename.
 ** Return Value: NONE.
 **/
```

```
void MJPEGwriteFirstFrame(struct MJPEGFrame* c, char* name)
{
       FILE* fpMJPEG; /* The Output Motion JPEG file. */
       char identity[5] = "MJPEG";
       char mID = ’M’;
       int mode = 0, count, i, j, mbMode = 0, value;

       //printf("MJPEGwriteFirstFrame\n");
       fpMJPEG = fopen(name, "wb"); /* Create a binary file for writing. */
       /* Write MJPEG identity. */
       fwrite(&identity, sizeof(char[5]), 1, fpMJPEG);
       /* Write INTRA information. */
    fwrite(&mode, sizeof(char), 1, fpMJPEG);

       for (count = 0; count < mSequence.numMBs; count++)
       {
              /* Write Macroblock Identifier, M */
       fwrite(&mID, sizeof(char), 1, fpMJPEG);
              /* Write that frame is INTRA=0. */
       fwrite(&mbMode, sizeof(char), 1, fpMJPEG);

              for (j = 0; j < MBSIZE; j++)
              {
                     for (i = 0; i < MBSIZE; i++)
                     {
                            /* Write RUN information. */
                            value = c->mb[count].mbValue[i][j];
                            if (value > 255)
                                    value = 255;
                            fwrite(&value, sizeof(char), 1, fpMJPEG);
                            //printf("###i = %d j = %d value = %d\n",
                            //i, j, value);
                     }
              }

       }

       fclose(fpMJPEG);
}
```

File: mjpegdec.c

```
/**
 ** Module: mjpegdec.c: Motion JPEG Decoder.
 **/
```

```
#include "mjpeg.h"
#include <stdio.h>
#include <math.h>


/* 3D VLC Table (H.263). */
int TableInv3DVLC[103][5] = {
/* Last, Run, Level, Bits, Code Word */
                                              0,  0,  1,  3,  2,
                                              0,  0,  2,  5,  15,
                                              0,  0,  3,  7,  21,
                                              0,  0,  4,  8,  23,
                                              0,  0,  5,  9,  31,
                                              0,  0,  6,  10, 37,
                                              0,  0,  7,  10, 36,
                         0,  0,  8,  11, 33,
                         0,  0,  9,  11, 32,
                                              0,  0,  10, 12, 7,
                                              0,  0,  11, 12, 6,
                                              0,  0,  12, 12, 32,
                                              0,  1,  1,  4,  6,
                                              0,  1,  2,  7,  20,
                                              0,  1,  3,  9,  30,
                                              0,  1,  4,  11, 15,
                                              0,  1,  5,  12, 33,
                                              0,  1,  6,  13, 80,
                                              0,  2,  1,  5,  14,
                                              0,  2,  2,  9,  29,
                                              0,  2,  3,  11, 14,
                                              0,  2,  4,  13, 81,
                                              0,  3,  1,  6,  13,
                                              0,  3,  2,  10, 35,
                                              0,  3,  3,  11, 13,
                                              0,  4,  1,  6,  12,
                                              0,  4,  2,  10, 34,
                                              0,  4,  3,  13, 82,
                                              0,  5,  1,  6,  11,
                                              0,  5,  2,  11, 12,
                                              0,  5,  3,  13, 83,
                                              0,  6,  1,  7,  19,
                                              0,  6,  2,  11, 11,
                                              0,  6,  3,  13, 84,
                                              0,  7,  1,  7,  18,
                                              0,  7,  2,  11, 10,
                                              0,  8,  1,  7,  17,
                                              0,  8,  2,  11, 9,
                                              0,  9,  1,  7,  16,
                                              0,  9,  2,  11, 8,
    0,  10, 1,  8,  22,
    0,  10, 2,  13, 85,
    0,  11, 1,  8,  21,
    0,  12, 1,  8,  20,
0,  13, 1,  9,  28,
    0,  14, 1,  9,  27,
    0,  15, 1,  10, 33,
    0,  16, 1,  10, 32,
    0,  17, 1,  10, 31,
    0,  18, 1,  10, 30,
0,  19, 1,  10, 29,
    0,  20, 1,  10, 28,
    0,  21, 1,  10, 27,
    0,  22, 1,  10, 26,
    0,  23, 1,  12, 34,
    0,  24, 1,  12, 35,
0,  25, 1,  13, 86,
    0,  26, 1,  13, 87,
    1,  0,  1,  5,  7,
    1,  0,  2,  10, 25,
    1,  0,  3,  12, 5,
    1,  1,  1,  7,  15,
    1,  1,  2,  12, 4,
    1,  2,  1,  7,  14,
    1,  3,  1,  7,  13,
    1,  4,  1,  7,  12,
    1,  5,  1,  8,  19,
    1,  6,  1,  8,  18,
    1,  7,  1,  8,  17,
    1,  8,  1,  8,  16,
    1,  9,  1,  9,  26,
    1,  10, 1,  9,  25,
    1,  11, 1,  9,  24,
    1,  12, 1,  9,  23,
    1,  13, 1,  9,  22,
    1,  14, 1,  9,  21,
    1,  15, 1,  9,  20,
    1,  16, 1,  9,  19,
    1,  17, 1,  10, 24,
    1,  18, 1,  10, 23,
    1,  19, 1,  10, 22,
    1,  20, 1,  10, 21,
    1,  21, 1,  10, 20,
    1,  22, 1,  10, 19,
    1,  23, 1,  10, 18,
    1,  24, 1,  10, 17,
    1,  25, 1,  11, 7,
    1,  26, 1,  11, 6,
                                                 1,  27, 1,  11, 5,
                                                 1,  28, 1,  11, 4,
                                                 1,  29, 1,  12, 36,
                                                 1,  30, 1,  12, 37,
                                                 1,  31, 1,  12, 38,
                                                 1,  32, 1,  12, 39,
                                                 1,  33, 1,  13, 88,
                                                 1,  34, 1,  13, 89,
                                                 1,  35, 1,  13, 90,
                                                 1,  36, 1,  13, 91,
                                                 1,  37, 1,  13, 92,
                                                 1,  38, 1,  13, 93,
                                                 1,  39, 1,  13, 94,
                                                 1,  40, 1,  13, 95,
                                                -1,  -1, -1,  7,  3   };

/**
 ** Function:     void MJPEGsetVirtualMBs(char* filename);
 ** Description:  Take the first Motion JPEG frame given read it and set
 **               the values of the virtual frame as macroblock values.
 ** Parameters:   The filename of the current frame.
 ** Return Value: NONE.
 **/
void MJPEGsetVirtualMBs(char* filename)
{
        int iw, ih, index = 0, i;
        unsigned char* data;

        for (i = 0; i < mSequence.numMBs; i++)
        {
                index = 0;
                data = MJPEGgetVirtualValues(i, filename);

                for (ih = 0; ih < MBSIZE; ih++)
                {
                        for (iw = 0; iw < MBSIZE; iw++)
                        {
                                /* Set virtual macroblock. */
                                vf.vMB[i].value[iw][ih] = data[index];
                                index++;
                        }
                }
        }
}

/**
 ** Function:     unsigned char* MJPEGgetVirtualValues(int mbNum, char* filename);
 ** Description:  Read frame and get macroblock values.
 ** Parameters:   The macroblock needed and the filename of the frame.
 ** Return Value: The macroblock values.
 **/
unsigned char* MJPEGgetVirtualValues(int mbNum, char* filename)
{
       FILE* fp;
       unsigned char tempData;
       int i = 0, count = -2, line = 0;
       long index = -1;
       unsigned char d[512];

       if ((fp = fopen(filename, "rb")) == NULL)
       {
              printf("Cannot open file\n");
              exit(1);
       }
       do
       {
              index++;
              fread(&tempData, sizeof(char), 1, fp);

              if (tempData == 77) /* if ’M’ macroblock reached. */
                     count++;
       }
       while (count != mbNum);

       index += 2;
       fseek(fp, index, SEEK_SET);
       fread(&d, sizeof(char), 256, fp);

       fclose(fp);

       return d;
}

/**
 ** Function:     void MJPEGsetMBPositions(void);
 ** Description:  Set the virtual macroblocks starting coordinates.
 ** Parameters:   NONE.
 ** Return Value: NONE.
 **/
void MJPEGsetMBPositions(void)
{

       int i, indexX = 0, indexY = -16, countx = -1, county = -1;

       for (i = 0; i < mSequence.numMBs; i++)
       {
```

```
		if (countx < (mSequence.numMbh - 1))
		{
			countx++;
			indexY += 16;
		}
		else
		if (countx == (mSequence.numMbh - 1))
		{
			countx = 0;
			indexX += 16;
			indexY = 0;
		}
			/* Set starting x, y coordinates. */
			vf.vMB[i].xInit = indexX;
			vf.vMB[i].yInit = indexY;
			//printf("MB:%d (%d,%d)\n", i, vf.vMB[i].yInit, vf.vMB[i].yInit);
	}
}

/**
 ** Function:     void MJPEGMakeRawFrame(void);
 ** Description:  Convert macroblock coordinate values to frame coordinate values.
 ** Parameters:   NONE.
 ** Return Value: NONE.
 **/
void MJPEGmakeRawFrame(void)
{
	int i, w, h, mbCount = 0;

	for (i = 0; i < mSequence.numMBs; i++)
	{

		for (h = 0; h < 16; h++)
		{
			for (w = 0; w < 16; w++)
			{
				vf.raw[w + vf.vMB[i].xInit][h + vf.vMB[i].yInit] =
vf.vMB[i].value[w][h];
			}
		}
	}
}

/**
 ** Function:     char* MJPEGsetDecoderFilename(char* c);
 ** Description:  Set the proper filename of the decoded frame.
 ** Parameters:   The motion JPEG frame filename.
 ** Return Value: The decoded frame filename.
 **/
char* MJPEGsetDecoderFilename(char* c)
{
        int i, dotPosition;
        char curr[24];
        char* name;
        name = malloc(24 * sizeof(char));
        dotPosition = strcspn(c, "."); /* Find the first dot in the filename. */
        for (i = 0; i < dotPosition; i++)
                curr[i] = c[i]; /* Take the part before the dot. */
        curr[i] = '\0';
        strcat(curr, ".ra0"); /* Make raw filename. */
        strcpy(name, curr);

    return name;
}

/**
 ** Function:     void MJPEGwriteVirtualFrame(char* filename);
 ** Description:  Create decoded frame and write data.
 ** Parameters:   The decoded frame filename.
 ** Return Value: NONE.
 **/
void MJPEGwriteVirtualFrame(char* filename)
{
        FILE* fp;
        int ih, iw;
        unsigned char temp;

        if ((fp = fopen(filename, "wb")) == NULL)
        {
                printf("Cannot open file\n");
                exit(1);
        }

        for (ih = 0; ih < mSequence.height; ih++)
        {
                for (iw = 0; iw < mSequence.width; iw++)
                {
                        temp = vf.raw[iw][ih];
                        fwrite(&temp, sizeof(unsigned char), 1, fp);
                }
        }
        fclose(fp);
}

/**
 ** Function:     void MJPEGgetMBModes(char* filename);
 ** Description:  Read a Motion JPEG frame and take the data
 **               of any INTER macroblock.
 ** Parameters:   The Motion JPEG frame filename.
 ** Return Value: NONE.
 **/
void MJPEGgetMBModes(char* filename)
{
       int i, size, count = -1, j;
       unsigned char* data;

       data = MJPEGgetFrameValues(filename, &size);

       for (i = 5; i < (size - 1); i++)
       {
              if (data[i] == 77)
              {
                     count++;
                     df.dMB[count].mode = data[i + 1];
                     if (df.dMB[count].mode == 1)
                     {
                            df.dMB[count].xs = data[i + 2];
                            df.dMB[count].xv = data[i + 3];
                            df.dMB[count].ys = data[i + 4];
                            df.dMB[count].yv = data[i + 5];
                            j = i + 6;
                            do
                            {
                                   df.dMB[count].data[j - (i + 6)] = data[j];
                                   j++;
                            }
                            while (data[j] != 77);
                            df.dMB[count].size = j - (i + 6);
                     }
              }
       }
}

/**
 ** Function:     unsigned char* MJPEGgetFrameValues(char* filename,
 **                                                  int* size);
 ** Description:  Read a Motion JPEG frame and take the contained data.
 ** Parameters:   The Motion JPEG frame filename and the number
 **               of values read.
 ** Return Value: The frame data read.
 **/
unsigned char* MJPEGgetFrameValues(char* filename,  int* size)
{
       FILE* fp;
```

```
        unsigned char mode = 0, data, ch[4048];
    int i = 0;

        if ((fp = fopen(filename, "rb")) == NULL)
        {
                printf("Cannot open file\n");
                exit(1);
        }

        do
        {
                data = 0;
                fread(&data, sizeof(char), 1, fp);
                ch[i] = data;
                i++;
        }
        while (!feof(fp));

        fclose(fp);
        *size = i;

        return ch;
}


/**
 ** Function:     void MJPEGgetVLC(int vlc, int* pos, int* neg);
 ** Description:  Calculates the possible values of a code word.
 ** Parameters:   The code word and its positive and negative values obtained.
 ** Return Value: NONE.
 **/
void MJPEGgetVLC(int vlc, int* pos, int* neg)
{
                *pos = vlc << 1; /* Shift one position. */


                vlc = abs(vlc);
                /* Shift one position and add one. */
                vlc = vlc << 1;
                vlc++;

                *neg = vlc;

}

/**
 ** Function:     void MJPEGinvRunLength(int mbNum);
 ** Description:  Perform inverse run length coding on the Motion JPEG data.
 ** Parameters:   A macroblock mbNum.
 ** Return Value: NONE.
 **/
void MJPEGinvRunLength(int mbNum)
{
       int i, j, k, index = -1, zzIndex = 0, negCode, posCode, mode, match, mp, mn;

       for (i = 0; i < df.dMB[mbNum].size; i=i+3)
       {
              index++;
              df.dMB[mbNum].runLength[index][0] = df.dMB[mbNum].data[i]; /* RUN */
              df.dMB[mbNum].runLength[index][1] = df.dMB[mbNum].data[i + 1]; /* LEVEL */
              df.dMB[mbNum].runLength[index][2] = df.dMB[mbNum].data[i + 2]; /* LAST */

              for (j = 0; j < 103; j++)
              {
                     MJPEGgetVLC(TableInv3DVLC[j][4], &posCode, &negCode);

                     if (
                            (df.dMB[mbNum].runLength[index][0] == TableInv3DVLC[j][1])
&& /* RUN */
                            (df.dMB[mbNum].runLength[index][2] == TableInv3DVLC[j][0])
&& /* LAST */
                            ((df.dMB[mbNum].runLength[index][1] == posCode) ||
                             (df.dMB[mbNum].runLength[index][1] == negCode))
                        )
                     {
                            match = j;
                            mp = posCode;
                            mn = negCode;
                     }
              }

              if (df.dMB[mbNum].runLength[index][0] != 0)
              {
                     for (k = 0; k < df.dMB[mbNum].runLength[index][0]; k++)
                     {
                            df.dMB[mbNum].zigzag[zzIndex] = 0;
                            zzIndex++;
                     }
              }

              if (df.dMB[mbNum].runLength[index][1] == mp)
              {
                     df.dMB[mbNum].zigzag[zzIndex] = TableInv3DVLC[match][2];
                     zzIndex++;
              }
              if (df.dMB[mbNum].runLength[index][1] == mn)
```

```
                {
                        df.dMB[mbNum].zigzag[zzIndex++] = -TableInv3DVLC[match][2];
                        zzIndex++;
                }
        }
}

/**
 ** Function:     void MJPEGinvZigzag(int mbNum);
 ** Description:  Perform inverse zig-zag on the Motion JPEG data.
 ** Parameters:   A macroblock mbNum.
 ** Return Value: NONE.
 **/
void MJPEGinvZigzag(int mbNum)
{
        /* Index of value coordinates inside the 2-D matrix to be followed. */
        int zzIndex[256][2] = {  0,0,     1,0,     0,1,     0,2,
                                           1,1,     2,0,     3,0,     2,1,
                                           1,2,     0,3,     0,4,     1,3,
                                               2,2,     3,1,     4,0,     5,0,
                                                      4,1,     3,2,     2,3,     1,4,
                                                      0,5,     0,6,     1,5,     2,4,
                                                      3,3,     4,2,     5,1,     6,0,
                                                      7,0,     6,1,     5,2,     4,3,
                                                      3,4,     2,5,     1,6,     0,7,
                                                      0,8,     1,7,     2,6,     3,5,
                                                      4,4,     5,3,     6,2,     7,1,
                                                     8,0,     9,0,     8,1,     7,2,
                                                      6,3,     5,4,     4,5,     3,6,
                                                      2,7,     1,8,     0,9,    0,10,
                                                      1,9,     2,8,     3,7,     4,6,
                                                      5,5,     6,4,     7,3,     8,2,
                                                      9,1,    10,0,    11,0,    10,1,
                                                      9,2,     8,3,     7,4,     6,5,
                                                      5,6,     4,7,     3,8,     2,9,
                                                     1,10,    0,11,    0,12,    1,11,
                                                     2,10,     3,9,     4,8,     5,7,
                                                      6,6,     7,5,     8,4,     9,3,
                                                     10,2,    11,1,    12,0,    13,0,
                                                     12,1,    11,2,    10,3,     9,4,
                                                      8,5,     7,6,     6,7,     5,8,
                                                     4,9,    3,10,    2,11,    1,12,
                                                     0,13,    0,14,    1,13,    2,12,
                                                     3,11,    4,10,     5,9,     6,8,
                                                     7,7,     8,6,     9,5,    10,4,
                                                     11,3,    12,2,    13,1,    14,0,
                                                     15,0,    14,1,    13,2,    12,3,
                                                     11,4,    10,5,     9,6,     8,7,
                                                      7,8,      6,9,     5,10,     4,11,
                                                     3,12,     2,13,     1,14,     0,15,
                                                     1,15,     2,14,     3,13,     4,12,
                                                     5,11,     6,10,      7,9,      8,8,
                                                      9,7,     10,6,     11,5,     12,4,
                                                     13,3,     14,2,     15,1,     15,2,
                                                     14,3,     13,4,     12,5,     11,6,
                                                     10,7,      9,8,      8,9,     7,10,
                                                     6,11,     5,12,     4,13,     3,14,
                                                     2,15,     3,15,     4,14,     5,13,
                                                     6,12,     7,11,     8,10,      9,9,
                                                     10,8,     11,7,     12,6,     13,5,
                                                     14,4,     15,3,     15,4,     14,5,
                                                     13,6,     12,7,     11,8,     10,9,
                                                     9,10,     8,11,     7,12,     6,13,
                                                     5,14,     4,15,     5,15,     6,14,
                                                     7,13,     8,12,     9,11,    10,10,
                                                     11,9,     12,8,     13,7,     14,6,
                                                     15,5,     15,6,     14,7,     13,8,
                                                     12,9,    11,10,    10,11,     9,12,
                                                     8,13,     7,14,     6,15,     7,15,
                                                     8,14,     9,13,    10,12,    11,11,
                                                    12,10,     13,9,     14,8,     15,7,
                                                     15,8,     14,9,    13,10,    12,11,
                                                     11,12,   10,13,     9,14,     8,15,
                                                     9,15,    10,14,    11,13,    12,12,
                                                 13,11,   14,10,    15,9,   15,10,
                                                 14,11,   13,12,   12,13,   11,14,
                                                 10,15,   11,15,   12,14,   13,13,
                                                 14,12,   15,11,   15,12,   14,13,
                                                 13,14,   12,15,   13,15,   14,14,
                                                 15,13,   15,14,   14,15,   15,15 };

        int iw, count = 0;

        for (iw = 0; iw < 256; iw++)
        {
               /* Set the quantization matrix by following the index coordinates. */
               df.dMB[mbNum].qMatrix[zzIndex[iw][0]][zzIndex[iw][1]] =
            df.dMB[mbNum].zigzag[count];
               count++;
        }
}

/**
 ** Function:     void MJPEGInvQuantization(int mbNum);
 ** Description:  Perform inverse quantization on the Motion JPEG data.
 ** Parameters:   A macroblock mbNum.
 ** Return Value: NONE.
 **/
void MJPEGInvQuantization(int mbNum)
{
        int ih, iw;
        double Q = 5.0; /* Q is set to 5. */

        for (ih = 0; ih < 16; ih++)
        {
                for (iw = 0; iw < 16; iw++)
                {
                        /* Multiply all values by ( 2*Q ). */
                        df.dMB[mbNum].dct[iw][ih] = (double) (df.dMB[mbNum].qMatrix[iw][ih]
* (2.0 * Q));
                }
        }
}

/**
 ** Function:     void MJPEGinvDct(int mbNum);
 ** Description:  Perform inverse DCT on the Motion JPEG data.
 ** Parameters:   A macroblock mbNum.
 ** Return Value: NONE.
 **/
void MJPEGinvDct(int mbNum)
{
        int u, v, i, j;
        double temp;


        for (j = 0; j < 16; j++)
        {
                for (i = 0; i < 16; i++)
                {


                        for (u = 0; u < 16; u++)
                        {
                                for (v = 0; v < 16; v++)
                                {

                                        temp += (df.dMB[mbNum].dct[u][v] * getC(u) * getC(v)
*
                                                                             cos((2.0 * i + 1.0) *
(u * PI) / 16.0) *

cos((2.0 * j + 1.0) * (v * PI) / 16.0));
                                }
                        }

                        df.dMB[mbNum].iMatrix[i][j] = temp * 0.25;
                        temp = 0.0;
                }
        }
}

/**
 ** Function:     void MJPEGgetResidualMatrix(int mbNum);
 ** Description:  Get the residual matrix of the macroblock.
 ** Parameters:   A macroblock mbNum.
 ** Return Value: NONE.
 **/
void MJPEGgetResidualMatrix(int mbNum)
{
        int ih, iw, xVector, yVector;

        if (df.dMB[mbNum].xs > 0)
                xVector = df.dMB[mbNum].xv;
        else
                xVector = - df.dMB[mbNum].xv;
                if (df.dMB[mbNum].ys > 0)
                yVector = df.dMB[mbNum].yv;
        else
                yVector = - df.dMB[mbNum].yv;

        for (ih = 0; ih < MBSIZE; ih++)
        {
                for (iw = 0; iw < MBSIZE; iw++)
                {
                        df.dMB[mbNum].rawMatrix[iw][ih] =
                        df.dMB[mbNum].iMatrix[iw][ih] +
                        vf.raw[iw + vf.vMB[mbNum].xInit + xVector][ih + vf.vMB[mbNum].yInit
+ yVector];
                        if (df.dMB[mbNum].rawMatrix[iw][ih] > 255)
                                df.dMB[mbNum].rawMatrix[iw][ih] = 255;
                        if (df.dMB[mbNum].rawMatrix[iw][ih] < 0)
                                df.dMB[mbNum].rawMatrix[iw][ih] = 0;
                }
        }
}

/**
 ** Function:     void MJPEGupdateVirtualFrame(int mbNum);
 ** Description:  Update virtual matrix to current data.
 ** Parameters:   A macroblock mbNum.
 ** Return Value: NONE.
 **/
void MJPEGupdateVirtualFrame(int mbNum)
{
        int i, j;
        //printf("\n\n\n");
        for (j = 0; j < 16; j++)
        {
                printf("\n");
                for (i = 0; i < 16; i++)
                {
                        vf.vMB[mbNum].value[i][j] = df.dMB[mbNum].rawMatrix[i][j];
                        //printf("%3d ", vf.vMB[mbNum].value[i][j]);
                        vf.raw[i + vf.vMB[mbNum].xInit][j + vf.vMB[mbNum].yInit] =
vf.vMB[mbNum].value[i][j];
                }
        }
}

/**
 ** Function:     char* MJPEGgetPSNRFilename(char* c);
 ** Description:  Get the proper filename of the raw frame.
 ** Parameters:   The motion JPEG frame filename.
 ** Return Value: The raw frame filename.
 **/
char* MJPEGgetPSNRFilename(char* c)
{
        int i, dotPosition;
        char curr[24];
        char* name;
        name = malloc(24 * sizeof(char));
        dotPosition = strcspn(c, "."); /* Find the first dot in the filename. */
        for (i = 0; i < dotPosition; i++)
                curr[i] = c[i]; /* Take the part before the dot. */
        curr[i] = '\0';
        strcat(curr, ".raw"); /* Make raw filename. */
        strcpy(name, curr);

    return name;
}
double getFAbs(double num)
{
        if (num >= 0.0)
                return num;
        else
                return -num;
}

double MJPEGcalculatePSNR(char* f)
```

```
{
	FILE* fp;
	unsigned char data;
	double psnrc = 0.0, psnr = 0, snr;
	int i = 0, j = 0, c = -1, num;


		printf("file=%s", f);
	if ((fp = fopen(f, "rb")) == NULL)
	{
		printf("Cannot open file\n");
		exit(1);
	}

	do
	{
		c++;
		fread(&data, sizeof(unsigned char), 1, fp);
		psnrc = (double)(data - vf.raw[i][j]);
		psnr += psnrc;
		/* printf("%d: %d - %d = %5.1f\n", c, vf.raw[i][j], data, psnrc); */
		i++;
		if (i == mSequence.width)
		{
			i = 0;
			j++;
		}
	}
	while (!feof(fp));
	fclose(fp);

	num = mSequence.width * mSequence.height;
	snr = (num * 255.0 * 255.0) / psnr;
	psnr = 10.0 * log10(snr);
	printf("\nPSNR =%8.3f\n", psnr);
	return psnr;
}
```

File: mjpeg.h

```
/**
 ** Module:      mjpeg.h
 ** Description: This is the header file of the Motion-JPEG library
 **              added to the standard JPEG code.
 ** Version:     1.0
 ** Created:     June - August 1998
 ** Author:      Christos Bohoris
 **
 **              Copyright (C) 1998 Christos Bohoris
 **
 **/

#ifndef MJPEG_DONE
#define MJPEG_DONE
#endif

#define MAXSIZE      128 /* The maximum width or height of a frame */
#define MBSIZE       16 /* The standard size of a macroblock (16x16). */
#define MAXNUMMB     64 /* The maximum number of macroblocks that can be contained in a
frame. */
#define MAXNUMFRAMES 16 /* The maximum number of frames in the video sequence. */
#define SWSIZE       48 /* The search window size. */
#define INTRA        0 /* INTRA mode, no motion observed. */
#define INTER        1 /* INTER mode, motion observed. */


/* ENCODER */

/**
 ** Structure:   MJPEGMotionVector
 ** Description: This structure represents a motion vector.
 ** Members:     xDiff, yDiff, the horizontal and vertical magnitudes of the motion
 **              vector. These values are positive for directions towards top or
 **              left and negative for directions towards bottom and right.
 **/
struct MJPEGMotionVector
{
        int xDiff, yDiff, sx, sy;
};

/**
 ** Structure:   MJPEGSequence
 ** Description: This structure contains a series of frames of the video sequence.
 ** Members:     The width and height of the frames in the sequence. The number of
 **              macroblocks numMBs in each frame. The number of frames, length,
 **              in the sequence. The filename of the each sequnce frame.
 **/
struct MJPEGSequence
{
        int width, /* The width ... */
                height, /* ... and height of the frames. */
                numMBs, /* Total number of macroblocks in the frame. */
                length, /* The number of frames in the sequence. */
                numMbw,
                numMbh;
        char* filename[40]; /* The filenames of the frames in the sequence. */
};

struct MJPEGFrameTable
{
        int mb[16];
        int value[256] ,mvx, mvy, sx, sy;
};

/**
 ** Structure:   MJPEGMacroblock
 ** Description: This structure contains the basic characteristics of a macroblock.
 ** Members:     The values of luminance of each pixel are held in
mbValue[XCoord][YCoord].
 **              The values of XCoord, YCoord are coordinates in the macroblock starting
from
 **              the top left macroblock pixel which has the coordinates (0,0). xBegin,
yBegin
 **              are the top left pixel coordinates of the macroblock relative to its
position
 **              in the frame. xEnd, yEnd are the bottom right pixel coordinates of the
 **              macroblock relative to its position in the frame.
 **/
struct MJPEGMacroblock
{
        int mbValue[MBSIZE][MBSIZE], /* Value of each pixel in the macroblock. */
            xBegin, /* Coordinates of pixel at the left top corner of the macroblock. */
                yBegin,
                xEnd, /* Coordinates of pixel at the right bottom corner of the
macroblock. */
```

```
               yEnd,
               sum, a, mean,
               mode; /* INTER or INTRA mode. */
       int rMatrix[MBSIZE][MBSIZE];
       double dctMatrix[MBSIZE][MBSIZE];
       int qMatrix[MBSIZE][MBSIZE];
       int zzMatrix[(MBSIZE * MBSIZE) + 1];
       struct MJPEGMotionVector motionVector;
       int runLength[256][5];
};


/**
 ** Structure:   MJPEGFrame
 ** Description: This structure contains the basic characteristics of a frame.
 ** Members:     frFilename is the filename of the frame. The structure contains
 **              an array of macroblocks mb that are contained in the frame.
 **              frValue[XCoord][YCoord] contains the values of luminance of each
 **              pixel in the frame.
 **/
struct MJPEGFrame
{
       char* frFilename; /* The name of the file containing the frame. */
       struct MJPEGMacroblock mb[MAXNUMMB]; /* The macroblocks in the frame */
       int frValue[MAXSIZE][MAXSIZE]; /* Value of each pixel in the frame. */
       char frMode;
};

/* Global Variables. */
struct MJPEGSequence mSequence; /* The video sequence. */
struct MJPEGFrame frame[MAXNUMFRAMES];

/* Function Definitions. */
void MJPEGgetFrame(struct MJPEGFrame* frame);
void MJPEGsetMacroblocks(struct MJPEGFrame* frame);
int getFrX(int x, int mbNum, struct MJPEGFrame* frame);
int getFrY(int y, int mbNum, struct MJPEGFrame* frame);
void MJPEGmotionEstimation(struct MJPEGFrame* p, struct MJPEGFrame* c);
int getAbs(int num);
double getC(int num);
void MJPEGsetResidualMatrix(int mbNum, struct MJPEGFrame* p, struct MJPEGFrame* c);
void MJPEGdct(int mbNum, struct MJPEGFrame* c);
void MJPEGquantization(int mbNum, struct MJPEGFrame* c);
void MJPEGzigzag(int mbNum, struct MJPEGFrame* c);
void MJPEGrunLength(int mbNum, struct MJPEGFrame* c);
int MJPEGsetVLC(int rl, int vlc);
char* MJPEGsetFilename(char* c);
void MJPEGwriteFrame(struct MJPEGFrame* c, char mode, char* name);
```

```
/* DECODER */
struct VirtualMB
{
	unsigned char value[16][16];
	int xInit, yInit;
};

struct VirtualFrame
{
	struct VirtualMB vMB[16];
	unsigned char raw[48][64];
};

struct DecMB
{
	char mode;
	int xv, yv, xs, ys, size;
	unsigned char data[512];
	int rawMatrix[16][16];
	double iMatrix[16][16];
	double dct[16][16];
	int qMatrix[16][16];
	int zigzag[256];
	int runLength[256][6];
};

struct DecFrame
{
	struct DecMB dMB[16];
};

struct VirtualFrame vf;
struct DecFrame df;

unsigned char* MJPEGgetVirtualValues(int mbNum, char* filename);
void MJPEGsetVirtualMBs(char* filename);
void MJPEGmakeRawFrame(void);
void MJPEGsetMBPositions(void);
void MJPEGwriteVirtualFrame(char* filename);
char* MJPEGsetDecoderFilename(char* c);
void MJPEGwriteVirtualFrame(char* filename);
unsigned char* MJPEGgetFrameValues(char* filename, int* size);
void MJPEGgetMBModes(char* filename);
void MJPEGinvRunLength(int mbNum);
void MJPEGgetVLC(int vlc, int* pos, int* neg);
void MJPEGinvZigzag(int mbNum);
void MJPEGInvQuantization(int mbNum);
void MJPEGinvDct(int mbNum);
void MJPEGgetResidualMatrix(int mbNum);
void MJPEGupdateVirtualFrame(int mbNum);
double MJPEGcalculatePSNR(char* f);
char* MJPEGgetPSNRFilename(char* c);
```