DESIGN OF JPEG COMPRESSOR

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology

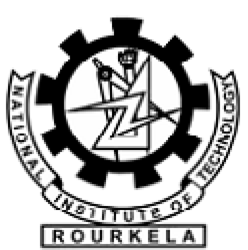
in

Electronics and Instrumentation Engineering

By

HARSH VARDHAN DWIVEDI

10507032



Department of Electronics and Communication Engineering

National Institute of Technology

Rourkela

2009

National Institute of Technology

Rourkela

CERTIFICATE

This is to certify that the thesis entitled, "DESIGN OF JPEG COMPRESSOR" submitted by Harsh Vardhan Dwivedi in partial fulfilments for the requirements for the award of Bachelor of Technology Degree in Electronics and Instrumentation Engineering at National Institute of Technology Rourkela is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University / Institute for the award of any Degree or Diploma.

Date: Prof. K.K Mahapatra

Dept. of Electronics and Communication Engineering

National Institute of Technology

Rourkela – 769008

ACKNOWLEDGEMENT

I would like to articulate my profound gratitude and indebtedness to my project guide **Prof. Dr. K.K Mahapatra** who has always been a constant motivation and guiding factor throughout the project time in and out as well. It has been a great pleasure for me to get an opportunity to work under him and complete the project successfully.
An undertaking of this nature could never have been attempted without aid and inspiration from the works of others whose details are mentioned in references section. I acknowledge my indebtedness to all of them. Last but not the least, my sincere thanks to all my friends who have patiently extended all kinds of help for accomplishing this undertaking.

Harsh Vardhan Dwivedi

CONTENTS

ABSTRACT……………………………………………………………….....05

CHAPTER 1: Theory behind JPEG..……………………………………...06

CHAPTER 2: Transient JPEG 2000.....................................……………...24

CHAPTER 3: The Program......................…………………………..………31

CHAPTER 4: MATLAB Results....................……………………………......59

REFERENCES……………………………………………………..………..64

ABSTRACT:

Images are generated, edited and transmitted on a very regular basis in a vast number of systems today. The raw image data generated by the sensors on a camera is very voluminous to store and hence not very efficient. It becomes especially cumbersome to move it around in bandwidth constrained systems or where bandwidth is to be conserved for cost purposes such as the World Wide Web. Such scenarios demand use of efficient image compressing techniques such as the JPEG algorithm technique which compresses the image to a high degree with little loss in perceived quality of the image. Today JPEG algorithm has become the de facto standard in image compression. MATLAB was used to write code for a program which could output a quantized DCT version of the input image and techniques for hardware implementation of JPEG algorithm in a speedy way were investigated.

Chapter 1
JPEG Theory

JPEG Theory

JPEG is an image compression standard used for storing images in a compressed format. It stands for Joint Photographic Experts Group. The remarkable quality of JPEG is that it achieves high compression ratios with little loss in quality.

JPEG format is quite popular and is used in a number of devices such as digital cameras and is also the format of choice when exchanging large sized images in a bandwidth constrained environment such as the Internet.

The JPEG algorithm is best suited for photographs and paintings of realistic scenes with smooth variations of tone and color. JPEG is not suited for images with many edges and sharp variations as this can lead to many artifacts in the resultant image. In these situations it is best to use lossless formats such as PNG, TIFF or GIF.

It is for this reason that JPEG is not used in medical and scientific applications where the image needs to reproduce the exact data as captured and the slightest of errors may snowball into bigger ones.

A JPEG image may undergo further losses if it is frequently edited and then saved. The operation of decompression and recompression may further degrade the quality of the image. To remedy this, the image should be edited and saved in a lossless format and only converted to JPEG format just before final transmittal to the desired medium. This ensures minimum losses due to frequent saving.

Image files saved in the JPEG format commonly have the extensions such as .jpg, .jpeg or .jpe.

A JPEG image consists of a sequence of *segments,* each beginning with a *marker*, each of which begins with a 0xFF byte followed by a byte indicating what kind of marker it is. Some markers consist of just those two bytes; others are followed by two bytes indicating the length of marker-specific payload data that follows. (The length includes the two bytes for the length, but not the two bytes for the marker.) Some markers are followed by entropy-coded data; the length of such a marker does not include the entropy-coded data. Note that consecutive 0xFF bytes are used as fill bytes for padding purposes.

Within the entropy-coded data, after any 0xFF byte, a 0x00 byte is inserted by the encoder before the next byte, so that there does not appear to be a marker where none is intended, preventing framing errors. Decoders must skip this 0x00 byte. This technique, called *byte stuffing*, is only applied to the entropy-coded data, not to marker payload data.

Common JPEG Markers

Short Name	Bytes	Payload	Name	Comments
SOI	0xFFD8	None	Start of Image	
SOF0	0xFFC0	Variable Size	Start of Frame (Baseline DCT)	Indicates that this is a baseline DCT-based JPEG, and specifies the width, height, number of components, and component subsampling (e.g., 4:2:0).
SOF2	0xFFC2	Variable Size	Start Of Frame (Progressive DCT)	Indicates that this is a progressive DCT-based JPEG, and specifies the width, height, number of components, and component subsampling (e.g., 4:2:0).
DHT	0xFFC4	Variable Size	Define Huffman Table	Specifies one or more Huffman tables
DQT	0xFFDB	Variable Size	Define Quantization Table	Specifies one or more quantization tables
DRI	0xFFDD	2 Bytes	Define Restart Interval	Specifies the interval between RST*n* markers, in macroblocks.
SOS	0xFFDA	Variable Size	Start of Scan	Begins a top-to-bottom scan of the image. In baseline DCT JPEG images, there is generally a single scan. Progressive DCT JPEG images usually contain multiple scans. This marker specifies which slice of data it will contain, and is immediately followed by entropy-coded data.
RSTn	0xFFD0 … 0xFFD7	None	Restart	Inserted every *r* macroblocks, where *r* is the restart interval set by a DRI marker. Not used if there was no DRI marker. The low 3 bits of the marker code, cycles from 0 to 7.
APPn	0xFFEn	Variable Size	Application Specific	For example, an Exif JPEG file uses an APP1 marker to store metadata, laid out in a structure based closely on TIFF
COM	0xFFFE	Variable Size	Comment	Contains a text comment.
EOI	0xFFD9	None	End of Image	

There are other *Start Of Frame* markers that introduce other kinds of JPEG.

Since several vendors might use the same APP*n* marker type, application-specific markers often begin with a standard or vendor name (e.g., "Exif" or "Adobe") or some other identifying string.

At a restart marker, block-to-block predictor variables are reset, and the bitstream is synchronized to a byte boundary. Restart markers provide means for recovery after bitstream

error, such as transmission over an unreliable network or file corruption. Since the runs of macroblocks between restart markers may be independently decoded, these runs may be decoded in parallel.

Typical Usage of JPEG codec

1. The representation of the colors in the image is converted from RGB to YCbCr, consisting of one luma component (Y), representing brightness, and two chroma components, (Cb and Cr), representing color. This step is sometimes skipped.
2. The resolution of the chroma data is reduced, usually by a factor of 2. This reflects the fact that the eye is less sensitive to fine color details than to fine brightness details.
3. The image is split into blocks of 8×8 pixels, and for each block, each of the Y, Cb, and Cr data undergoes a discrete cosine transform (DCT). A DCT is similar to a Fourier transform in the sense that it produces a kind of spatial frequency spectrum.
4. The amplitudes of the frequency components are quantized. Human vision is much more sensitive to small variations in color or brightness over large areas than to the strength of high-frequency brightness variations. Therefore, the magnitudes of the high-frequency components are stored with a lower accuracy than the low-frequency components. The quality setting of the encoder (for example 50 or 95 on a scale of 0–100 in the Independent JPEG Group's library[4]) affects to what extent the resolution of each frequency component is reduced. If an excessively low quality setting is used, the high-frequency components are discarded altogether.
5. The resulting data for all 8×8 blocks is further compressed with a loss-less algorithm, a variant of Huffman encoding.

The decoding process reverses these steps. In the remainder of this section, the encoding and decoding processes are described in more detail.

Encoding

Many of the options in the JPEG standard are not commonly used, and as mentioned above, most image software uses the simpler JFIF format when creating a JPEG file, which among other things specifies the encoding method. Here is a brief description of one of the more common methods of encoding when applied to an input that has 24 bits per pixel (eight each of red, green, and blue). This particular option is a lossy data compression method.

Color space transformation

First, the image should be converted from RGB into a different color space called YCbCr. It has three components Y, Cb and Cr: the Y component represents the brightness of a pixel, the Cb and Cr components represent the chrominance (split into blue and red components). This is the same color space as used by digital color television as well as digital video including video DVDs, and is similar to the way color is represented in analog PAL video and MAC but not by analog NTSC, which uses the YIQ color space. The YCbCr color space conversion allows greater compression without a significant effect on perceptual image quality (or greater perceptual image quality for the same compression). The compression is more efficient as the brightness information, which is more important to the eventual perceptual quality of the image, is confined to a single channel, more closely representing the human visual system.

This conversion to YCbCr is specified in the JFIF standard, and should be performed for the resulting JPEG file to have maximum compatibility. However, some JPEG implementations in "highest quality" mode do not apply this step and instead keep the colour information in the RGB color model, where the image is stored in separate channels for red, green and blue luminance. This results in less efficient compression, and would not likely be used if file size was an issue.

Downsampling

Due to the densities of color- and brightness-sensitive receptors in the human eye, humans can see considerably more fine detail in the brightness of an image (the Y component) than in the color of an image (the Cb and Cr components). Using this knowledge, encoders can be designed to compress images more efficiently.

The transformation into the YCbCr color model enables the next step, which is to reduce the spatial resolution of the Cb and Cr components (called "downsampling" or "chroma subsampling"). The ratios at which the downsampling can be done on JPEG are 4:4:4 (no downsampling), 4:2:2 (reduce by factor of 2 in horizontal direction), and most commonly 4:2:0 (reduce by factor of 2 in horizontal and vertical directions). For the rest of the compression process, Y, Cb and Cr are processed separately and in a very similar manner.

Block splitting

After subsampling, each channel must be split into 8×8 blocks of pixels. Depending on chroma subsampling, this yields (Minimum Coded Unit) MCU blocks of size 8×8 (4:4:4 – no subsampling), 16×8 (4:2:2), or most commonly 16×16 (4:2:0).

If the data for a channel does not represent an integer number of blocks then the encoder must fill the remaining area of the incomplete blocks with some form of dummy data. Filling the edge pixels with a fixed color (typically black) creates ringing artifacts along the visible part of the border; repeating the edge pixels is a common technique that reduces the visible border, but it can still create artifacts.

Discrete cosine transform

Next, each component (Y, Cb, Cr) of each 8×8 block is converted to a frequency-domain representation, using a normalized, two-dimensional type-II discrete cosine transform (DCT).

Consider the following as an example of an 8x8 sub image:

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

Before computing the DCT of the subimage, its gray values are shifted from a positive range to one centered around zero. For an 8-bit image each pixel has 256 possible values: [0,255]. To center around zero it is necessary to subtract by half the number of possible values, or 128.

Subtracting 128 from each pixel value yields pixel values on [− 128,127].

$$\overset{x}{\longrightarrow}$$

$$\begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix} \Bigg\downarrow y$$

The next step is to take the two-dimensional DCT, which is given by:

$$G_{u,v} = \alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} g_{x,y} \cos\left[\frac{\pi}{8}\left(x+\frac{1}{2}\right)u\right]\cos\left[\frac{\pi}{8}\left(y+\frac{1}{2}\right)v\right]$$

- u is the horizontal spatial frequency, for the integers, $0 \leq u < 8$
- v is the vertical spatial frequency, for the integers, $0 \leq v < 8$
- $$\alpha_p(n) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } n = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases}$$
- $g_{x,y}$ is the pixel value at coordinates (x,y).
- $G_{u,v}$ is the DCT coefficient at coordinates (u,v)

If this transformation is performed on the above matrix,

$$\begin{array}{c} u \\ \longrightarrow \end{array}$$

$$\begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix} \Big\downarrow v$$

Note the rather large value of the top-left corner. This is the DC coefficient. The remaining 63 coefficients are called the AC coefficients. The advantage of the DCT is its tendency to aggregate most of the signal in one corner of the result, as may be seen above. The quantization step to follow accentuates this effect while simultaneously reducing the overall size of the DCT coefficients, resulting in a signal that is easy to compress efficiently in the entropy stage.

The DCT temporarily increases the bit-depth of the image, since the DCT coefficients of an 8-bit/component image take up to 11 or more bits (depending on fidelity of the DCT calculation) to store. This may force the codec to temporarily use 16-bit bins to hold these coefficients, doubling the size of the image representation at this point; they are typically reduced back to 8-bit values by the quantization step. The temporary increase in size at this stage is not a performance concern for most JPEG implementations, because typically only a very small part of the image is stored in full DCT form at any given time during the image encoding or decoding process.

Quantization

The human eye is good at seeing small differences in brightness over a relatively large area, but not so good at distinguishing the exact strength of a high frequency brightness variation. This allows one to greatly reduce the amount of information in the high frequency components. This is done by simply dividing each component in the frequency domain by a constant for that component, and then rounding to the nearest integer. This is the main lossy operation in the whole process. As a result of this, it is typically the case that many of the higher frequency components are rounded to zero, and many of the rest become small positive or negative numbers, which take many fewer bits to store.

A typical Quantization matrix:

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

The quantized DCT coefficients are computed with:

$$B_{j,k} = \mathrm{round}\left(\frac{G_{j,k}}{Q_{j,k}}\right) \text{ for } j = 0,1,2,\cdots,N_1-1; k = 0,1,2,\cdots,N_2-1$$

where G is the unquantized DCT coefficients; Q is the quantization matrix above; and B is the quantized DCT coefficients.

Using this quantization matrix with the DCT coefficient matrix from above results in:

$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

For example, using −415 (the DC coefficient) and rounding to the nearest integer,

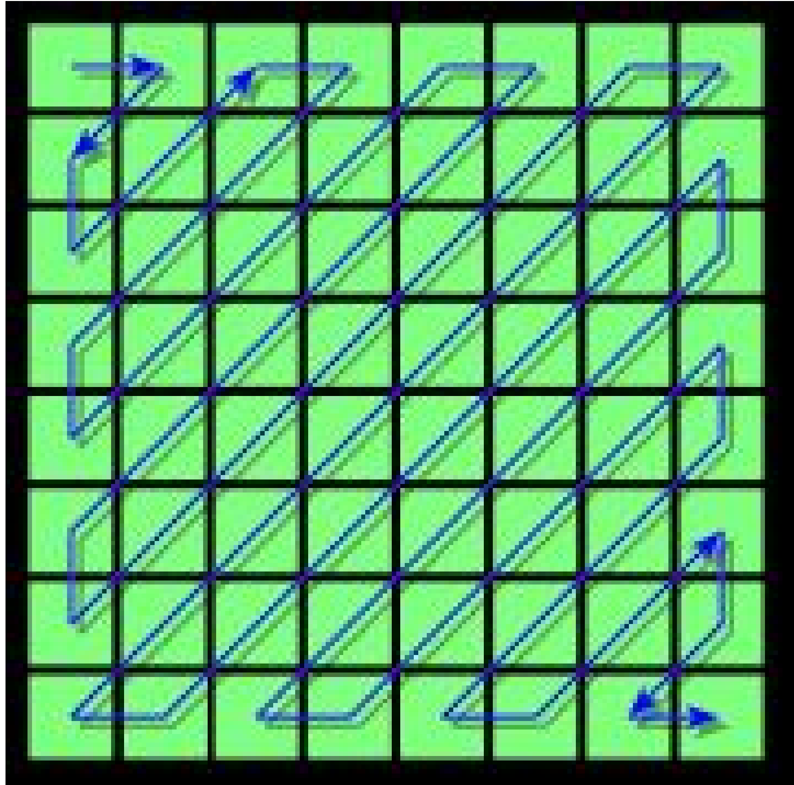
$$\mathrm{round}\left(\frac{-415}{16}\right) = \mathrm{round}(-25.9375) = -26$$

Entropy coding

Entropy coding is a special form of lossless data compression. It involves arranging the image components in a "zigzag" order employing run-length encoding (RLE) algorithm that groups similar frequencies together, inserting length coding zeros, and then using Huffman coding on what is left.

The JPEG standard also allows, but does not require, the use of arithmetic coding, which is mathematically superior to Huffman coding. However, this feature is rarely used as it is covered by patents and because it is much slower to encode and decode compared to Huffman coding. Arithmetic coding typically makes files about 5% smaller.

While using Huffman coding, the zigzag ordering of the JPEG components is done as:



Compression ratio and artifacts

The resulting compression ratio can be varied according to need by being more or less aggressive in the divisors used in the quantization phase. Ten to one compression usually results in an image that cannot be distinguished by eye from the original. 100 to one compression is usually possible, but will look distinctly artifacted compared to the original. The appropriate level of compression depends on the use to which the image will be put.

At many times JPEG images appear with certain irregularities which are due to compression artifacts. These are due to the quantization step of the JPEG algorithm. They are especially noticeable around sharp corners between contrasting colours (text is a good example as it contains many such corners). They can be reduced by choosing a lower level of compression; they may be eliminated by saving an image using a lossless file format, though for photographic images this will usually result in a larger file size. Compression artifacts make low-quality JPEGs unacceptable for storing height maps. The images created with ray-tracing programs have noticeable blocky shapes on the terrain. Compression artifacts are acceptable when the images are used for visualization purpose. Unfortunately subsequent processing of these images usually result in unacceptable artefacts.

Decoding

Decoding to display the image consists of doing all the above in reverse.

Taking the DCT coefficient matrix (after adding the difference of the DC coefficient back in)

$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and taking the entry-for-entry product with the quantization matrix from above results in

$$\begin{bmatrix} -416 & -33 & -60 & 32 & 48 & -40 & 0 & 0 \\ 0 & -24 & -56 & 19 & 26 & 0 & 0 & 0 \\ -42 & 13 & 80 & -24 & -40 & 0 & 0 & 0 \\ -56 & 17 & 44 & -29 & 0 & 0 & 0 & 0 \\ 18 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

which closely resembles the original DCT coefficient matrix for the top-left portion. Taking the inverse DCT (type-III DCT) results in an image with values (still shifted down by 128)

$$\begin{bmatrix} -68 & -65 & -73 & -70 & -58 & -67 & -70 & -48 \\ -70 & -72 & -72 & -45 & -20 & -40 & -65 & -57 \\ -68 & -76 & -66 & -15 & 22 & -12 & -58 & -61 \\ -62 & -72 & -60 & -6 & 28 & -12 & -59 & -56 \\ -59 & -66 & -63 & -28 & -8 & -42 & -69 & -52 \\ -60 & -60 & -67 & -60 & -50 & -68 & -75 & -50 \\ -54 & -46 & -61 & -74 & -65 & -64 & -63 & -45 \\ -45 & -32 & -51 & -72 & -58 & -45 & -45 & -39 \end{bmatrix}$$

and adding 128 to each entry

$$\begin{bmatrix} 60 & 63 & 55 & 58 & 70 & 61 & 58 & 80 \\ 58 & 56 & 56 & 83 & 108 & 88 & 63 & 71 \\ 60 & 52 & 62 & 113 & 150 & 116 & 70 & 67 \\ 66 & 56 & 68 & 122 & 156 & 116 & 69 & 72 \\ 69 & 62 & 65 & 100 & 120 & 86 & 59 & 76 \\ 68 & 68 & 61 & 68 & 78 & 60 & 53 & 78 \\ 74 & 82 & 67 & 54 & 63 & 64 & 65 & 83 \\ 83 & 96 & 77 & 56 & 70 & 83 & 83 & 89 \end{bmatrix}$$

This is the uncompressed subimage and can be compared to the original subimage (also see images to the right) by taking the difference (original − uncompressed) results in error values

$$\begin{bmatrix} -8 & -8 & 6 & 8 & 0 & 0 & 6 & -7 \\ 5 & 3 & -1 & 7 & 1 & -3 & 6 & 1 \\ 2 & 7 & 6 & 0 & -6 & -12 & -4 & 6 \\ -3 & 2 & 3 & 0 & -2 & -10 & 1 & -3 \\ -2 & -1 & 3 & 4 & 6 & 2 & 9 & -6 \\ 11 & -3 & -1 & 2 & -1 & 8 & 5 & -3 \\ 11 & -11 & -3 & 5 & -8 & -3 & 0 & 0 \\ 4 & -17 & -8 & 12 & -5 & -7 & -5 & 5 \end{bmatrix}$$

with an average absolute error of about 5 values per pixels:

$$\frac{1}{64}\sum_{x=1}^{8}\sum_{y=1}^{8}|e(x,y)| = 4.8125$$

The error is most noticeable in the bottom-left corner where the bottom-left pixel becomes darker than the pixel to its immediate right.

Required precision

The JPEG encoding does not fix the precision needed for the output compressed image. On the contrary, the JPEG standard (as well as the derived MPEG standards) have very strict precision requirements for the decoding, including all parts of the decoding process (variable length decoding, inverse DCT, dequantization, renormalization of outputs); the output from the reference algorithm must not exceed:

- a maximum 1 bit of difference for each pixel component
- low mean square error over each 8×8-pixel block
- very low mean error over each 8×8-pixel block
- very low mean square error over the whole image
- extremely low mean error over the whole image

These assertions are tested on a large set of randomized input images, to handle the worst cases. Look at the IEEE 1180-1990 standard for reference. This has a consequence on the implementation of decoders, and it is extremely critical because some encoding processes (notably used for encoding sequences of images like MPEG) need to be able to construct, on the encoder side, a reference decoded image. In order to support 8-bit precision per pixel component output, dequantization and inverse DCT transforms are typically implemented with at least 14-bit precision in optimized decoders.

Effects of JPEG compression

JPEG compression artifacts blend well into photographs with detailed non-uniform textures, allowing higher compression ratios. Notice how a higher compression ratio first affects the high-frequency textures in the upper-left corner of the image, and how the contrasting lines become more fuzzy. The very high compression ratio severely affects the quality of the image, although the overall colors and image form are still recognizable. However, the precision of colors suffer less (for a human eye) than the precision of contours (based on luminance). This justifies the fact that images should be first transformed in a color model separating the luminance from the chromatic information, before subsampling the chromatic planes (which may also use lower quality quantization) in order to preserve the precision of the luminance plane with more information bits.

Image	Quality	Size (Bytes)	Compression Ratio	Comment
	Full quality (Q = 100)	83,261	2.6:1	Extremely minor artifacts
	Average quality (Q = 50)	15,138	15:1	Initial signs of subimage artifac

	Medium quality (Q = 25)	9,553	23:1	Stronger artifact loss of high resolution information
	Low quality (Q = 10)	4,787	46:1	Severe high frequency loss; artifacts on subimage boundaries ("macroblocking are obvious
	Lowest quality (Q = 1)	1,523	144:1	Extreme loss of color and detail; the leaves are nearly unrecognizable

Further Lossless compression

There is ongoing research on ways to compress further the data in a JPEG image without modifying the represented image. This has applications in scenarios where the original image is only available in JPEG format, and its size needs to be reduced for archival or transmission. Standard general-purpose compression tools cannot significantly compress JPEG files.

Typically, such schemes take advantage of improvements to the naive scheme for coding DCT coefficients, which fails to take into account:

- Correlations between magnitudes of adjacent coefficients in the same block;
- Correlations between magnitudes of the same coefficient in adjacent blocks;
- Correlations between magnitudes of the same coefficient/block in different channels;
- The DC coefficients when taken together resemble a downscale version of the original image multiplied by a scaling factor. Well-known schemes for lossless coding of continuous-tone images can be applied, achieving somewhat better compression than the Huffman coded DPCM used in JPEG.

Some standard but rarely-used options already exist in JPEG to improve the efficiency of coding DCT coefficients: the arithmetic coding option, and the progressive coding option (which produces lower bitrates because values for each coefficient are coded independently, and each coefficient has a significantly different distribution). Modern methods have improved on these techniques by reordering coefficients to group coefficients of larger magnitude together; using adjacent coefficients and blocks to predict new coefficient values; dividing blocks or coefficients up among a small number of independently coded models based on their statistics and adjacent values; and most recently, by decoding blocks, predicting subsequent blocks in the spatial domain, and then encoding these to generate predictions for DCT coefficients.

Typically, such methods can compress existing JPEG files between 15 and 25 percent, and for JPEGs compressed at low-quality settings, can produce improvements of up to 65%.

Common Image Formats

Format	Description	Typical Usage
TIFF	Tagged Image File Format	Scan files, master images and other high quality photographic images. Any situation where versatility and retaining all pixels is more important than file size. Supports the RGB, CMYK, Lab and Grayscale color spaces. Supports layers and alpha channels. Supports 16 bit. Supports text annotations. Offers maximum compatibility between Adobe and non-Adobe products.
PSD	Photoshop Document. Native Adobe Photoshop file format	Scan files, master images and other high quality photographic images. Any situation where versatility and retaining all pixels is more important than file size. Supports the RGB, CMYK, Lab, Grayscale and Multichannel color spaces. Supports layers and alpha channels. Supports 16 bit. Supports text annotations. Offers maximum compatibility between Photoshop versions and other Adobe products.
Raw	The unprocessed data from a digital capture. Contains the actual, unaltered data recorded by a digital camera's sensor. (Technically, a Raw file is a data file and not an image file.) Since Raw is a word and not an acronym, this web site spells it Raw, not RAW. Should not be confused with Adobe's Digital Negative format, DNG.	High quality digital camera captures. Because it gives the photographer unaltered data, many consider it the more flexible format. However, it is not a format the image should remain in. Typically, once opened in Photoshop, a copy is made and saved in the TIFF or PSD format. Supports a broader tonal and color range that JPEG. File size is larger than JPEG, but smaller than uncompressed TIFF. Supports 16 bit. Because it is unprocessed, there is no color space associated with it. Also, Raw is a manufacturer proprietary, closed source format. Therefore, Photoshop's Camera Raw plug-in will need to be able to read the specific manufacturer's Raw format before it can open the file.
XMP	Sidecar file created by Camera Raw	XMP, or Extensible Metadata Platform, files are not actually image files nor do they contain image data. They contain Camera Raw

		adjustment instructions. Photoshop will not update a manufacturer proprietary Raw file. Instead, if adjustments are made to a Raw file in Camera Raw, these adjustments are stored as instructions in a XMP file. When the file is opened again, Camera Raw reapplies the adjustments by processing the instructions in the XMP file.
DNG	Digital Negative. Adobe's generic Raw format.	Since Raw is a manufacturer proprietary format, Adobe created the DNG format to allow a universal, open source, Raw format.
GIF	Graphic Interchange Format	Web graphics with a very limited color gamut and distinct detail, such as web page buttons. Supports very small file sizes and allows an image to have a transparent background. Not a format for photographic images.
PNG	Portable Network Graphics	Web graphics. A competing format to GIF. A license and patent free graphics format. Supports full color. Supports 16 bit. Supports RGB and Grayscale color spaces. Does not support layers. Does not support non-transparency alpha channels. Does not support the CMYK or Lab color spaces. Does not support text annotations. Not a format for master photographic images.
BMP	Bit mapped	A Microsoft Windows graphic file format. More appropriate for graphic work than photographic images. Supports the RGB and Grayscale color spaces. Supports alpha channels. Does not support color profiles. Does not support the CMYK or Lab color spaces. Does not support 16 bit. Does not support layers.
EPS	Encapsulated PostScript	A PostScript file. Not a format used by photographers. Typically used by graphic artists.

Comparison between JPEG and Raw

	Raw	**JPEG**
In-Camera Processing	The only in-camera processing Raw data undergoes is the conversion of an analog signal to digital. The result is a capture than contains 100% of the data recorded by the sensor. This is the main reason the Raw format is considered the most flexible.	In-camera processing applies all color, white balance and tone conversions. The image is complete, unless corrections or adjustments are needed.
Convenience	Raw is not a ready-to-use format. Raw allows more control over the image but requires more work. It must be converted from a data file to an image file before it can be used as an image. Photoshop's Camera Raw performs this conversion.	A JPEG capture is ready to print or email (assuming you do not need to perform any image corrections or enhancements).
White Balance	White balance is not actually applied to a Raw file. It is a value recorded in the file's metadata. This means you can change the white balance inside Camera Raw with absolutely no loss or degradation of quality.	White balance has been permanently applied to the image when saved by the digital camera. To fix any color imbalance, the image must be color corrected.
Bit depth	8 or 16 bit	8 bit
Color and tonal gamut	Since Raw is a 16 bit capture, it can capture more gradations of tone and color than an 8 bit capture.	Up to 16.7 million different colors.
File size	Much larger than JPEG	Small due to lossy compression.
Image quality	Image quality is more a function of the photographer's skills than file format.	More photographers ruin an image because of poor technique than JPEG's lossy compression ever will.

Chapter 2
JPEG 2000

JPEG 2000

JPEG 2000 is a wavelet-based image compression standard. It was created by the Joint Photographic Experts Group committee in the year 2000 with the intention of superseding their original discrete cosine transform-based JPEG standard (created 1992). The standardized filename extension is .jp2 for ISO/IEC 15444-1 conforming files and .jpx for the extended part-2 specifications, published as ISO/IEC 15444-2, while the MIME type is image/jp2.

While there is a modest increase in compression performance of JPEG2000 compared to JPEG, the main advantage offered by JPEG2000 is the significant flexibility of the codestream. The codestream obtained after compression of an image with JPEG2000 is scalable in nature, meaning that it can be decoded in a number of ways; for instance, by truncating the codestream at any point, one may obtain a representation of the image at a lower resolution, or signal-to-noise ratio. By ordering the codestream in various ways, applications can achieve significant performance increases. However, as a consequence of this flexibility, JPEG2000 requires encoders/decoders that are complex and computationally demanding. Another difference, in comparison with JPEG, is in terms of visual artifacts: JPEG 2000 produces ringing artifacts, manifested as blur and rings near edges in the image, while JPEG produces ringing artifacts and 'blocking' artifacts, due to its 8×8 blocks.

JPEG 2000 has been published as an ISO standard, ISO/IEC 15444. As of 2008, JPEG 2000 is not widely supported in web browsers, and hence is not generally used on the World Wide Web.

For traditional JPEG, additional metadata, e.g. lighting and exposure conditions, is kept in an application marker in the Exif format specified by the JEITA. JPEG2000 chooses a different route, encoding the same metadata in XML form. The reference between the Exif tags and the XML elements is standardized by the ISO TC42 committee in the standard 12234-1.

Features

- Superior compression performance: At high bit rates, where artifacts become nearly imperceptible, JPEG 2000 has a small machine-measured fidelity advantage over JPEG. At lower bit rates (e.g., less than 0.25 bits/pixel for gray-scale images), JPEG 2000 has a much more significant advantage over certain modes of JPEG: artifacts are less visible and there is almost no blocking. The compression gains over JPEG are attributed to the use of DWT and a more sophisticated entropy encoding scheme.

- Multiple resolution representation: JPEG2000 decomposes the image into a multiple resolution representation in the course of its compression process. This representation can be put to use for other image presentation purposes beyond compression as such.

- Progressive transmission by pixel and resolution accuracy, commonly referred to as progressive decoding and signal-to-noise ratio (SNR) scalability: JPEG2000 provides efficient code-stream organizations which are progressive by pixel accuracy and by image resolution (or by image size). This way, after a smaller part of the whole file has been received, the viewer can see a lower quality version of the final picture. The quality then improves progressively through downloading more data bits from the

source. The 1991 JPEG standard also has a progressive transmission feature but it's rarely used.

- Lossless and lossy compression: Like JPEG 1991,[1] the JPEG2000 standard provides both lossless and lossy compression in a single compression architecture. Lossless compression is provided by the use of a reversible integer wavelet transform in JPEG 2000.

- Random code-stream access and processing, also referred as Region Of Interest (ROI): JPEG2000 code streams offer several mechanisms to support spatial random access or region of interest access at varying degrees of granularity. This way it is possible to store different parts of the same picture using different quality.

- Error resilience: Like JPEG 1991, JPEG2000 is robust to bit errors introduced by noisy communication channels, due to the coding of data in relatively small independent blocks.

- Flexible file format: The JP2 and JPX file formats allow for handling of color-space information, metadata, and for interactivity in networked applications as developed in the JPEG Part 9 JPIP protocol.

- Side channel spatial information: it fully supports transparency and alpha planes.

Color components transformation

Initially, images have to be transformed from the RGB color space to another color space, leading to three *components* that are handled separately. There are two possible choices:

1. Irreversible Color Transform (ICT) uses the well known YCBCR color space. It is called "irreversible" because it has to be implemented in floating or fix-point and causes round-off errors.
2. Reversible Color Transform (RCT) uses a modified YUV color space that does not introduce quantization errors, so it is fully reversible. Proper implementation of the RCT requires that numbers are rounded as specified that cannot be expressed exactly in matrix form. The transformation is:

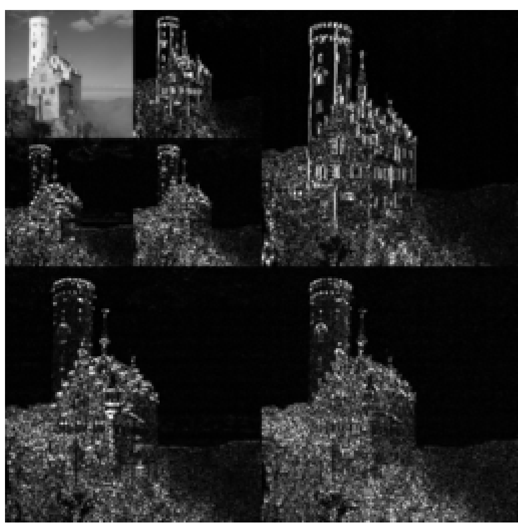
$$Y_r = \left\lfloor \frac{R + 2G + B}{4} \right\rfloor ; C_b = B - G; C_r = R - G; \qquad G = Y - \left\lfloor \frac{C_b + C_r}{4} \right\rfloor ; R = C_r + G; B = C_b + G.$$

Tiling

After color transformation, the image is split into so-called *tiles*, rectangular regions of the image that are transformed and encoded separately. Tiles can be any size, and it is also possible to consider the whole image as one single tile. Once the size is chosen, all the tiles will have the same size (except optionally those on the right and bottom borders). Dividing the image into tiles is advantageous in that the decoder will need less memory to decode the image and it can opt to decode only selected tiles to achieve a partial decoding of the image.

The disadvantage of this approach is that the quality of the picture decreases due to a lower peak signal-to-noise ratio. Using many tiles can create a blocking effect similar to the older JPEG 1992 standard.

Wavelet transform



An example of the wavelet transform that is used in JPEG2000. This is a 2nd-level CDF 9/7 wavelet transform.

These tiles are then wavelet transformed to an arbitrary depth, in contrast to JPEG 1992 which uses an 8×8 block-size discrete cosine transform. JPEG 2000 uses two different wavelet transforms:

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

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

Quantization

After the wavelet transform, the coefficients are scalar-quantized to reduce the amount of bits to represent them, at the expense of a loss of quality. The output is a set of integer numbers which have to be encoded bit-by-bit. The parameter that can be changed to set the final quality is the quantization step: the greater the step, the greater is the compression and the loss of quality. With a quantization step that equals 1, no quantization is performed (it is used in lossless compression).

Coding

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

The quantized sub-bands are split further into *precincts*, rectangular regions in the wavelet domain. They are typically selected in a way that the coefficients within them across the sub-bands form approximately spatial blocks in the (reconstructed) image domain, though this is not a requirement.

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

Clearly, in lossless mode all bit planes have to be encoded by the EBCOT, and no bit planes can be dropped.

The bits selected by these coding passes then get encoded by a context-driven binary arithmetic coder, namely the binary MQ-coder. The context of a coefficient is formed by the state of its nine neighbors in the code block.

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

Packets from all sub-bands are then collected in so-called *layers*. The way the packets are built up from the code-block coding passes, and thus which packets a layer will contain, is not defined by the JPEG 2000 standard, but in general a codec will try to build layers in such a way that the image quality will increase monotonically with each layer, and the image distortion will shrink from layer to layer. Thus, layers define the progression by image quality within the code stream.

The problem is now to find the optimal packet length for all code blocks which minimizes the overall distortion in a way that the generated target bitrate equals the demanded bit rate.

While the standard does not define a procedure as to how to perform this form of rate–distortion optimization, the general outline is given in one of its many appendices: For each bit encoded by the EBCOT coder, the improvement in image quality, defined as mean square error, gets measured; this can be implemented by an easy table-lookup algorithm. Furthermore, the length of the resulting code stream gets measured. This forms for each code

block a graph in the rate–distortion plane, giving image quality over bitstream length. The optimal selection for the truncation points, thus for the packet-build-up points is then given by defining critical *slopes* of these curves, and picking all those coding passes whose curve in the rate–distortion graph is steeper than the given critical slope. This method can be seen as a special application of the method of *Lagrange multiplier* which is used for optimization problems under constraints. The Lagrange multiplier, typically denoted by λ, turns out to be the critical slope, the constraint is the demanded target bitrate, and the value to optimize is the overall distortion.

Packets can be reordered almost arbitrarily in the JPEG 2000 bit-stream; this gives the encoder as well as image servers a high degree of freedom.

Already encoded images can be sent over networks with arbitrary bit rates by using a layer-progressive encoding order. On the other hand, color components can be moved back in the bit-stream; lower resolutions (corresponding to low-frequency sub-bands) could be sent first for image previewing. Finally, spatial browsing of large images is possible through appropriate tile and/or partition selection. All these operations do not require any re-encoding but only byte-wise copy operations.

Performance

JPEG 2000 gains up to about 20% compression performance for medium compression rates in comparison to the first JPEG standard. For lower or higher compression rates, the improvement can be somewhat greater (especially if altering the input resolution to the codec is not considered as a technique for effective use of the older JPEG standard). Good applications for JPEG 2000 are large images, images with low-contrast edges — e.g., medical images.

It has, however, notably higher computational and memory demands.

Finally, JPEG2000 has in some research been found to perform inferior to the intra-frame coding mode of H.264.

Applications of JPEG 2000

Some markets and applications intended to be served by this standard are listed below:

- Consumer applications such as multimedia devices (e.g., digital cameras, personal digital assistants, 3G mobile phones, color facsimile, printers, scanners, etc.)
- Client/server communication (e.g., the Internet, Image database, Video streaming, video server, Second Life, etc.)
- Military/surveillance (e.g., HD satellite images, Motion detection, network distribution and storage, etc.)
- Medical imagery, esp. the DICOM specifications for medical data interchange.
- Remote sensing
- High-quality frame-based video recording, editing and storage.
- Digital cinema
- JPEG 2000 has many design commonalities with the ICER image compression format that is used to send images back from the Mars rovers.

- World Meteorological Organization has built JPEG 2000 Compression into the new GRIB2 file format. The GRIB file structure is designed for global distribution of meteorological data. The implementation of JPEG 2000 compression in GRIB2 has reduced file sizes up to 80%.

Comparison of JPEG with JPEG 2000

Chapter 3
Program

MATLAB Program

```
clc;
clear all;
close all;

q=1;
w=1;
row=1;
column=1;


a=imread('gs.bmp');
b=double(zeros(8,8,4096));
c=double(zeros(8,8,4096));
for k=1:4096
    for i=row:(row+7)
        for j=column:(column+7)
            b(q,w,k)=a(i,j);
            w=w+1;
            if(w==9)
                w=1;
            end
        end
        q=q+1;
        if(q==9)
            q=1;
        end
    end
    row=i+1;

        if(row==513)
            row=1;
            column=j+1;
        end
end
figure, imshow(a);
b=b-128;

for k=1:4096
    c(:,:,k)=dct2(b(:,:,k));
end

s1=double(zeros(512,512));
row=1;
q=1;
w=1;
column=1;
for k=1:4096
    for i=row:(row+7)
        for j=column:(column+7)
            s1(i,j)=c(q,w,k);
            w=w+1;
            if(w==9)
                w=1;
            end
        end
        q=q+1;
        if(q==9)
```

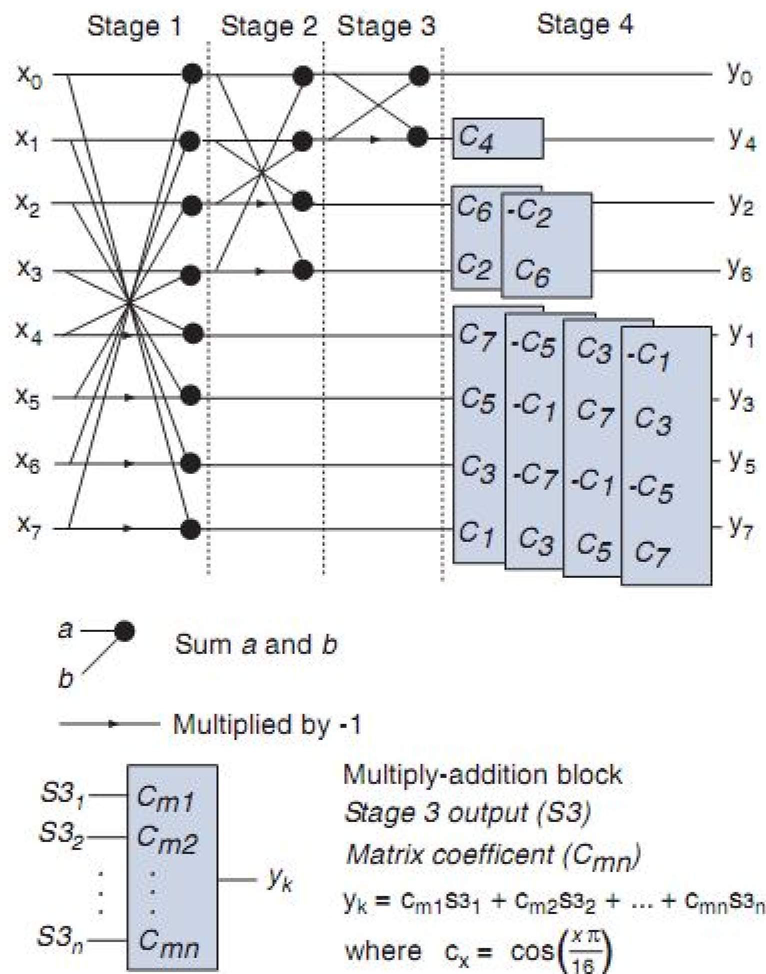
```
                q=1;
            end
        end
        row=i+1;
            if(row==513)
                row=1;
                column=j+1;
            end
end
figure,imshow(s1);

m=[16 11 10 16 24 40 51 61; 12 12 14 19 26 58 60 55; 14 13 16 24 40 57 69
56; 14 17 22 29 51 87 80 62; 18 22 37 56 68 109 103 77; 24 35 55 64 81 104
113 92; 49 64 78 87 103 121 120 101; 72 92 95 98 112 100 103 99];
m=double(m);
d=double(zeros(8,8,4096));
d=double(c);
for k=1:4096
    d(:,:,k)=round(c(:,:,k)./m);
 end
 c=d;
 e=d;



 for k=1:4096
    e(:,:,k)=round(d(:,:,k).*m);
 end



 f=double(zeros(8,8,4096));

 for k=1:4096
     f(:,:,k)=idct2(e(:,:,k));
 end

 f=f+128;

 s2=double(zeros(512,512));
row=1;
q=1;
w=1;
column=1;
for k=1:4096
    for i=row:(row+7)
        for j=column:(column+7)
            s2(i,j)=round(f(q,w,k));
            w=w+1;
            if(w==9)
                w=1;
            end
        end
        q=q+1;
        if(q==9)
            q=1;
        end
    end
    row=i+1;
        if(row==513)
            row=1;
            column=j+1;
```

```
        end
end
figure,imshow(s2);
s3=double(zeros(512,512));
s3=s2+128;
s3=uint8(s3);
figure, imshow(s3);
figure, imshow(s3-a);
```

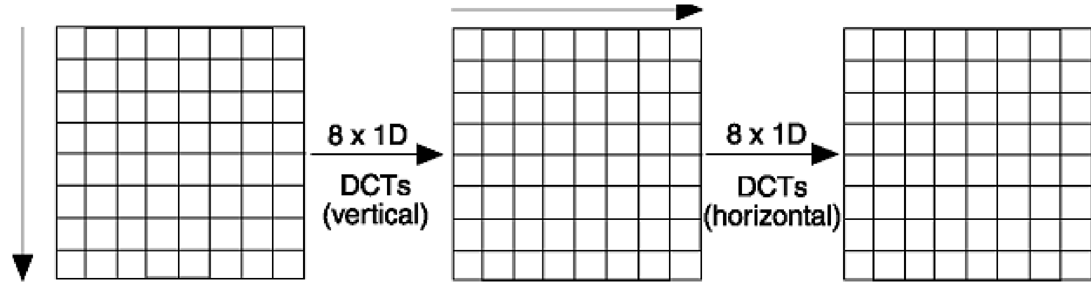
This program was written in MATLAB to take an image as input and output the quantized DCT version of the given image for the purpose of JPEG compression.

The image used in the program:

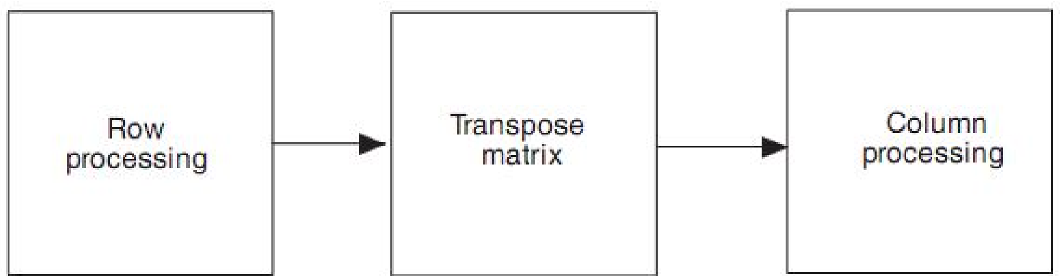
It is observed that when DCT definition is used to calculate the DCT coefficients a lot of time is taken. However this time can be reduced by using an algorithmic approach towards the computation of DCT. Most dedicated computing platforms for Image processing use fast DCT algorithm which is similar to Fast FFT algorithm.



Computing DCT:



3 stages in implementing 2D DCT:



The code for such a DCT Verilog core is from Altera DSP Example Code Library as:

```
module dct (clk, aclr, clken, col_en, row_en,
            x0,x1,x2,x3,x4,x5,x6,x7,
            y0,y1,y2,y3, y4,y5,y6,y7);


    input clk, aclr;
    input clken;
    input col_en, row_en;
    input [21:0] x0, x1, x2, x3, x4, x5, x6, x7;
    output [21:0] y0, y1, y2, y3, y4, y5, y6, y7;


  //Parameter Declaration
  //Row-based DCT
    parameter constant_09808_14_8 = 251;         //FB     C(1)
    parameter constant_m09808_14_8 = 4194053;  //3FFF05 -C(1)
    parameter constant_092388 = 236;             //EC     C(2)
    parameter constant_m092388 = 4194068;     //3FFF14 -C(2)
    parameter constant_08315_14_8 = 212;         //D4     C(3)
    parameter constant_070711 = 181;             //B5     C(4)
    parameter constant_05556_14_8 = 142;         //8E     C(5)
```

```
parameter constant_m05556_14_8 = 4194161; //3FFF71 -C(5)

parameter constant_038268 = 98;                    //62    C(6)

parameter constant_01951_14_8 = 50;                //32    C(7)

parameter constant_m01951_14_8 = 4194254; //3FFFCE -C(7)


//Reg Declaration

reg [21:0] stage1_0, stage1_1, stage1_2, stage1_3;

reg [21:0] stage1_4, stage1_5, stage1_6, stage1_7;

reg [21:0] stage2_0, stage2_1, stage2_2, stage2_3;

reg [21:0] stage3_0, stage3_1;

reg [21:0] stage4_0_0p, stage4_0_1p, stage4_0_2p, stage4_0_3p;

reg [21:0] stage4_0_4p;

reg [21:0] stage4_4_0p, stage4_5_0p, stage4_6_0p, stage4_7_0p;

reg [21:0] stage4_4_1p, stage4_5_1p, stage4_6_1p, stage4_7_1p;

reg [21:0] stage4_4_2p, stage4_5_2p, stage4_6_2p, stage4_7_2p;

reg [21:0] stage4_4_sel, stage4_5_sel, stage4_6_sel, stage4_7_sel;

reg [21:0] stage5_sel_0p, stage5_sel_1p, stage5_sel_2p;

reg [21:0] stage6_2_0p, stage6_3_0p, stage6_2_1p, stage6_3_1p;

reg [21:0] stage6_2_2p, stage6_3_2p;

reg [21:0] stage6_4, stage6_5, stage6_6, stage6_7;

reg [1:0] mux_sel;

reg [7:0] mux_clken_cnt;


//Wire Declaration

wire [21:0] X0, X1, X2, X3, X4, X5, X6, X7;
```

```
wire [21:0] stage2_4_w, stage2_5_w, stage2_6_w, stage2_7_w;

wire [21:0] stage3_2_w, stage3_3_w, stage3_4_w, stage3_5_w, stage3_6_w,
stage3_7_w;

wire [21:0] stage4_1_w, stage4_2_w, stage4_3_w;

wire [21:0] stage5_0_w;

wire [35:0] stage5_1_w;

wire [36:0] stage5_2, stage5_3;

wire [37:0] stage5_sel;

wire [21:0] stage5_2_w, stage5_3_w, stage5_4_w, stage5_5_w, stage5_6_w,
stage5_7_w;

wire [21:0] stage6_0_w, stage6_1_w, stage6_4_w, stage6_5_w, stage6_6_w,
stage6_7_w;

wire [21:0] stage4_4_w, stage4_5_w, stage4_6_w, stage4_7_w;

wire [21:0] stage5_sel_w;

wire mux_clken, mux_clken_w;

wire mux_cnt_en;



//Feeding adders in stage1

assign X0=   x0 ;

assign X1=   x1 ;

assign X2=   x2 ;

assign X3=   x3 ;

assign X4=   x4 ;

assign X5=   x5 ;

assign X6=   x6 ;

assign X7=   x7 ;
```

```
//Connect output transform (switch order)
assign y0 = stage6_0_w;
assign y1 = stage6_4_w;
assign y2 = stage6_2_2p;
assign y3 = stage6_5_w;
assign y4 = stage6_1_w;
assign y5 = stage6_6_w;
assign y6 = stage6_3_2p;
assign y7 = stage6_7_w;

//Stage 1

adder   adder_s1_0 (
.dataa ( X0 ),
.datab ( X7 ),
.clock ( clk ),
.aclr ( aclr),
.clken ( clken ),
.result ( stage1_0 ),
.cout ( ),
.overflow ( )
);

adder   adder_s1_1 (
```

```
.dataa ( X1 ),
.datab ( X6 ),
.clock ( clk ),
.aclr ( aclr),
.clken ( clken ),
.result ( stage1_1 ),
.cout ( ),
.overflow ( )
);

adder   adder_s1_2 (
.dataa ( X2 ),
.datab ( X5 ),
.clock ( clk ),
.aclr ( aclr),
.clken ( clken ),
.result ( stage1_2 ),
.cout ( ),
.overflow ( )
);

adder   adder_s1_3 (
.dataa ( X3 ),
.datab ( X4 ),
.clock ( clk ),
```

```
        .aclr ( aclr),
        .clken ( clken ),
        .result ( stage1_3 ),
        .cout ( ),
        .overflow ( )
        );

subtractor    subtractor_s1_4 (
        .dataa ( X3 ),
        .datab ( X4 ),
        .clock ( clk ),
        .aclr ( aclr ),
        .clken ( clken ),
        .result ( stage1_4 ),
        .cout ( ),
        .overflow ( )
        );

        subtractor      subtractor_s1_5 (
        .dataa ( X2 ),
        .datab ( X5 ),
        .clock ( clk ),
        .aclr ( aclr ),
        .clken ( clken ),
        .result ( stage1_5 ),
```

```
.cout ( ),
.overflow ( )
);

 subtractor       subtractor_s1_6 (
.dataa ( X1 ),
.datab ( X6 ),
.clock ( clk ),
.aclr ( aclr ),
.clken ( clken ),
.result ( stage1_6 ),
.cout ( ),
.overflow ( )
);

 subtractor       subtractor_s1_7 (
.dataa ( X0 ),
.datab ( X7 ),
.clock ( clk ),
.aclr ( aclr ),
.clken ( clken ),
.result ( stage1_7 ),
.cout ( ),
.overflow ( )
);
```

```
//Stage 2

adder   adder_s2_0 (
.dataa ( stage1_0 ),
.datab ( stage1_3 ),
.clock ( clk ),
.aclr ( aclr),
.clken ( clken ),
.result ( stage2_0 ),
.cout ( ),
.overflow ( )
);

adder   adder_s2_1 (
.dataa ( stage1_1 ),
.datab ( stage1_2 ),
.clock ( clk ),
.aclr ( aclr),
.clken ( clken ),
.result ( stage2_1 ),
.cout ( ),
.overflow ( )
);
```

```
subtractor    subtractor_s2_2 (
        .dataa ( stage1_1 ),
        .datab ( stage1_2 ),
        .clock ( clk ),
        .aclr ( aclr ),
        .clken ( clken ),
        .result ( stage2_2 ),
        .cout ( ),
        .overflow ( )
        );

         subtractor       subtractor_s2_3 (
        .dataa ( stage1_0 ),
        .datab ( stage1_3 ),
        .clock ( clk ),
        .aclr ( aclr ),
        .clken ( clken ),
        .result ( stage2_3 ),
        .cout ( ),
        .overflow ( )
        );

        assign stage2_4_w = stage1_4;
        assign stage2_5_w = stage1_5;
        assign stage2_6_w = stage1_6;
```

```
assign stage2_7_w = stage1_7;

//Stage 3

adder   adder_s3_0 (
.dataa ( stage2_0 ),
.datab ( stage2_1 ),
.clock ( clk ),
.aclr ( aclr),
.clken ( clken ),
.result ( stage3_0 ),
.cout ( ),
.overflow ( )
);

subtractor    subtractor_s3_1 (
.dataa ( stage2_0 ),
.datab ( stage2_1 ),
.clock ( clk ),
.aclr ( aclr ),
.clken ( clken ),
.result ( stage3_1 ),
.cout ( ),
```

```
        .overflow ( )
        );

assign stage3_2_w = stage2_2;
assign stage3_3_w = stage2_3;
        assign stage3_4_w = stage2_4_w;
        assign stage3_5_w = stage2_5_w;
        assign stage3_6_w = stage2_6_w;
        assign stage3_7_w = stage2_7_w;


        //Stage 4

        always @(posedge clk or posedge aclr)
        begin
                if (aclr)
                  begin
                        stage4_0_0p <= 0;
                        stage4_0_1p <= 0;
                        stage4_0_2p <= 0;
                        stage4_0_3p <= 0;
                        stage4_0_4p <= 0;
                        stage4_4_0p <= 0;
                        stage4_5_0p <= 0;
                        stage4_6_0p <= 0;
```

```
            stage4_7_0p <= 0;
            stage4_4_1p <= 0;
            stage4_5_1p <= 0;
            stage4_6_1p <= 0;
            stage4_7_1p <= 0;
            stage4_4_2p <= 0;
            stage4_5_2p <= 0;
            stage4_6_2p <= 0;
            stage4_7_2p <= 0;
   end
else if (clken)
   begin
    stage4_0_0p <= stage3_0;
            stage4_0_1p <= stage4_0_0p;
            stage4_0_2p <= stage4_0_1p;
            stage4_0_3p <= stage4_0_2p;
            stage4_0_4p <= stage4_0_3p;
            stage4_4_0p <= stage3_4_w;
            stage4_5_0p <= stage3_5_w;
            stage4_6_0p <= stage3_6_w;
            stage4_7_0p <= stage3_7_w;
            stage4_4_1p <= stage4_4_0p;
            stage4_5_1p <= stage4_5_0p;
            stage4_6_1p <= stage4_6_0p;
            stage4_7_1p <= stage4_7_0p;
```

```
                stage4_4_2p <= stage4_4_1p;

                stage4_5_2p <= stage4_5_1p;

                stage4_6_2p <= stage4_6_1p;

                stage4_7_2p <= stage4_7_1p;

            end

    end


    assign stage4_1_w = stage3_1;

assign stage4_2_w = stage3_2_w;

assign stage4_3_w = stage3_3_w;




    //Stage5

assign stage5_0_w = stage4_0_4p;


    assign stage4_4_w =  stage4_4_sel;

    assign stage4_5_w =  stage4_5_sel;

    assign stage4_6_w =  stage4_6_sel;

    assign stage4_7_w =  stage4_7_sel;


    assign mux_cnt_en = clken;

assign mux_clken = mux_clken_w ? 1'b1 : 1'b0;

    assign mux_clken_w = mux_clken_cnt >=8 && clken == 1'b1 ;


    always @ (posedge clk or posedge aclr)
```

```
        begin
          if (aclr)
            begin
              mux_clken_cnt <= 0;
            end
                else if (mux_cnt_en)
                  begin
                    mux_clken_cnt <= mux_clken_cnt + 1;
                  end
                else
                  begin
                        mux_clken_cnt <= 0;
                  end
        end

//Generate mux select signal
always @ (posedge clk or posedge aclr)
        begin
          if (aclr)
            begin
              mux_sel <= 0;
            end
                else if (mux_clken)
                  begin
                    mux_sel <= mux_sel + 1;
```

```
            end
          else
            begin
              mux_sel <= 0;
            end
      end

// Mux control for data feeding the four_mul_add block
always @ (posedge clk)
      begin
          case (mux_sel)
                  0:  stage4_4_sel = stage3_4_w;
                  1:      stage4_4_sel = -stage4_6_0p;
                  2:      stage4_4_sel = stage4_5_1p;
                  3:  stage4_4_sel = stage4_7_2p;
          endcase
     end

always @ (posedge clk)
      begin
          case (mux_sel)
                  0:  stage4_5_sel = stage3_5_w;
                  1:      stage4_5_sel = -stage4_4_0p;
                  2:      stage4_5_sel = stage4_7_1p;
                  3:  stage4_5_sel = -stage4_6_2p;
```

```
            endcase
        end



always @ (posedge clk)
         begin
            case (mux_sel)
                  0:  stage4_6_sel = stage3_6_w;
                  1:        stage4_6_sel = stage4_7_0p;
                  2:        stage4_6_sel = stage4_4_1p;
                  3:  stage4_6_sel = stage4_5_2p;
            endcase
        end

        always @ (posedge clk)
         begin
            case (mux_sel)
                  0:  stage4_7_sel = stage3_7_w;
                  1:        stage4_7_sel = -stage4_5_0p;
                  2:        stage4_7_sel = -stage4_6_1p;
                  3:  stage4_7_sel = -stage4_4_2p;
            endcase
        end

        one_mult_blk  one_mult_blk_inst0 (
```

```
.dataa ( stage4_1_w[17:0] ),
.datab ( constant_070711 ),     //C5
.clock ( clk ),
.aclr ( aclr ),
.clken ( clken ),
.result ( stage5_1_w )
);

two_mult_add_blk two_mult_add_blk_inst0(
.aclr ( aclr ),
.clken ( clken ),
.clk ( clk ),
.constant_one ( constant_038268 ),  //C6
.constant_two ( constant_092388 ),  //C2
.X ( stage4_2_w[17:0]),
.Y ( stage4_3_w[17:0]),
.result ( stage5_2 )
);

two_mult_add_blk two_mult_add_blk_inst1(
.aclr ( aclr ),
.clken ( clken ),
.clk ( clk ),
.constant_one ( constant_m092388 ),  //-C2
.constant_two ( constant_038268 ),   //C6
```

```
.X ( stage4_2_w[17:0]),

.Y ( stage4_3_w[17:0]),

.result ( stage5_3 )

);


four_mult_add_blk four_mult_add_blk_inst0(

.aclr ( aclr ),

.clken ( clken ),

.clk ( clk ),

.constant_one ( constant_01951_14_8 ),        //C7

.constant_two ( constant_05556_14_8 ),  //C5

.constant_three ( constant_08315_14_8 ),//C3

.constant_four ( constant_09808_14_8 ), //C1

.W ( stage4_4_w[17:0] ),

.X ( stage4_5_w[17:0] ),

.Y ( stage4_6_w[17:0] ),

.Z ( stage4_7_w[17:0] ),

.result ( stage5_sel )

);


assign stage5_sel_w = stage5_sel[29:8];


always @(posedge clk or posedge aclr)

begin

        if (aclr)
```

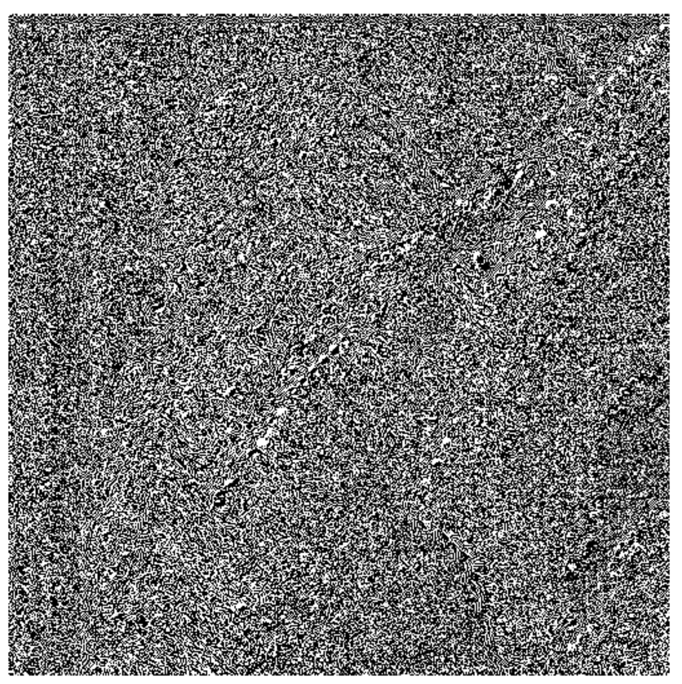
```
        begin
            stage5_sel_0p <= 0;
            stage5_sel_1p <= 0;
            stage5_sel_2p <= 0;
        end
      else if (clken)
        begin
         stage5_sel_0p <= stage5_sel_w;  // s6
            stage5_sel_1p <= stage5_sel_0p; // s5
            stage5_sel_2p <= stage5_sel_1p; // s4
        end
end



//Stage6
assign stage5_2_w = stage5_2[29:8];
assign stage5_3_w = stage5_3[29:8];

assign stage5_4_w = stage5_sel_2p;
assign stage5_5_w = stage5_sel_1p;
assign stage5_6_w = stage5_sel_0p;
assign stage5_7_w = stage5_sel_w;

always @(posedge clk or posedge aclr)
begin
```

```
        if (aclr)
          begin
                stage6_2_0p <= 0;
                stage6_3_0p <= 0;
                stage6_2_1p <= 0;
                stage6_3_1p <= 0;
                stage6_2_2p <= 0;
                stage6_3_2p <= 0;
          end
        else if (clken)
          begin
           stage6_2_0p <= stage5_2_w;
                stage6_3_0p <= stage5_3_w;
                stage6_2_1p <= stage6_2_0p;
                stage6_3_1p <= stage6_3_0p;
                stage6_2_2p <= stage6_2_1p;
                stage6_3_2p <= stage6_3_1p;
          end
end


assign stage6_0_w = stage5_0_w;
assign stage6_1_w = stage5_1_w[29:8];
assign stage6_4_w = stage5_4_w;
assign stage6_5_w = stage5_5_w;
assign stage6_6_w = stage5_6_w;
```

```
    assign stage6_7_w = stage5_7_w;

endmodule
```

Chapter 4
MATLAB Results:

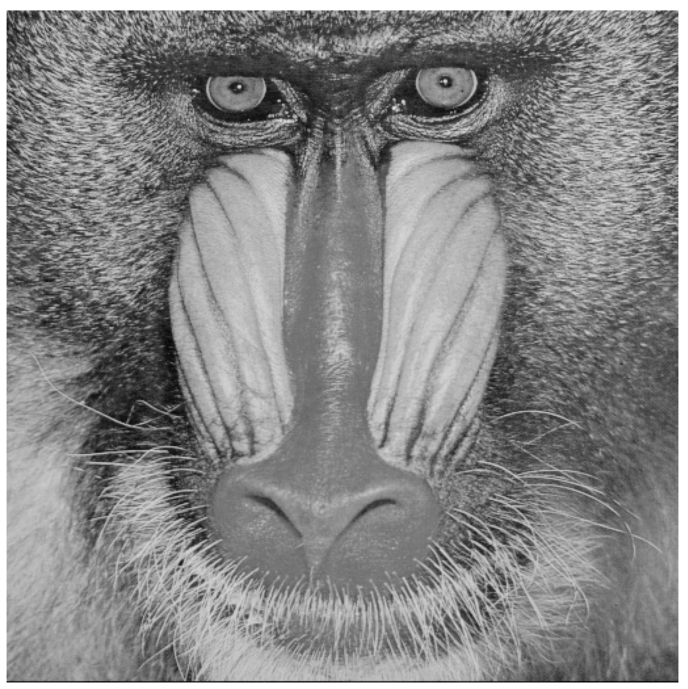
Original Input Image (Lenna)



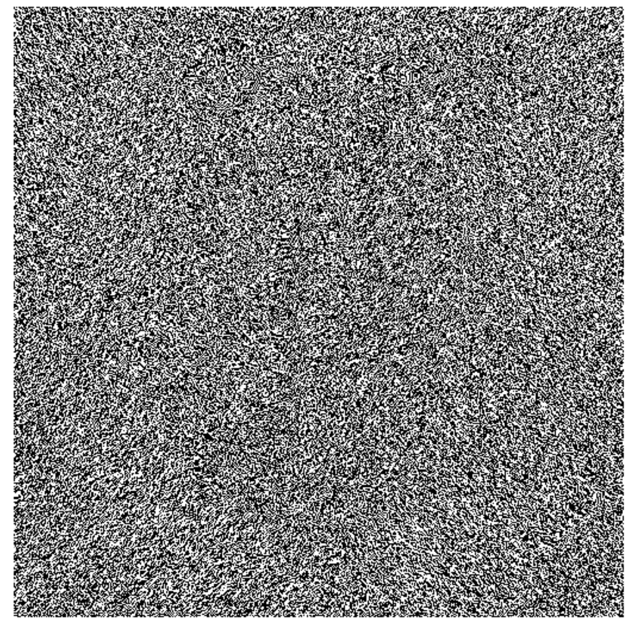
Lenna DCT

Recovered Lenna Image

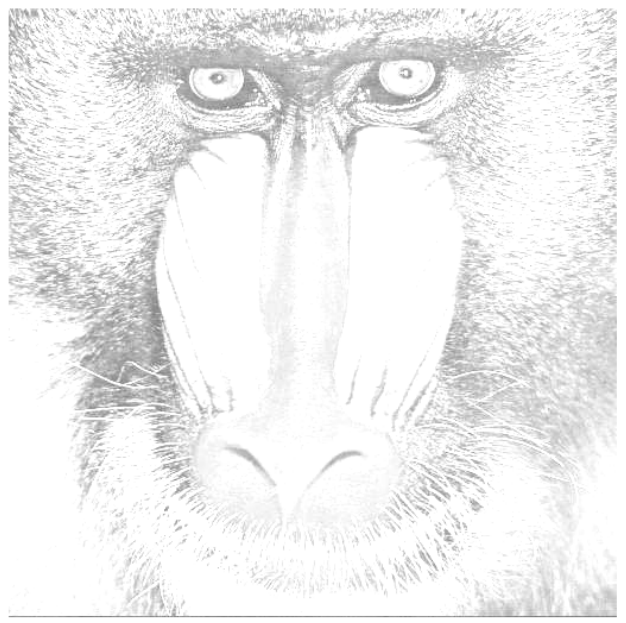
Lenna Error Image



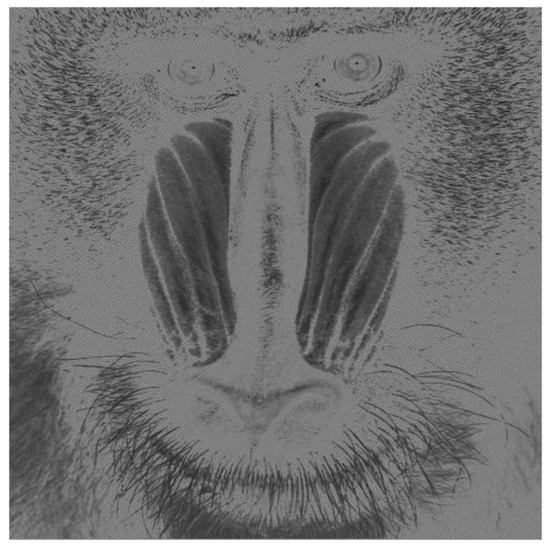
Original Baboon Image



Baboon DCT



Recovered Baboon



Baboon Error Image

REFERENCES:

- Wikipedia
  en.wikipedia.org
- Digital Image Processing by Gonzalez and Woods
- ALTERA DSP Example Code