PROCESSING AND CODIFICATION IMAGES BASED ON JPEG STANDARD

Proyecto Fin de Carrera

UNIVERSIDAD CARLOS III DE MADRID

Departamento de Teoría de la Señal

Autor: Sergio Tallante Pastor

Profesor: Prof. Wolf-Peter Buchwald

Tutor: Dirk Reifenstahl

Wolfenbüttel, January 2011

PROCESSING AND CODIFICATION IMAGES BASED ON JPEG STANDARD

Final Project

FACHHOCHSCHULE BRAUNSCHWEIG/WOLFENBÜTTEL

UNIVERSITY **OF APPLIED SCIENCES**

Department of Electrotechnical Engineering

Author: Sergio Tallante Pastor

Matrikel-Nr: 1009154

Mentor: Prof. Wolf-Peter Buchwald

Tutor: Dirk Reifenstahl

Wolfenbüttel, January 2011

This is to certify that except where specific reference is made, the work described in this project is result of the candidate. Neither this project, nor any part of it, has been presented or is currently submitted in candidature for any degree at another University.

Candidate: Sergio Tallante Pastor

Date: 28-01-2011

ACKNOWLEDGEMENTS

This Final Project is the end of one of the most important stages of my life, and although I have had some bad experiences, I think there have been more good moments, and these are the really important.

I would like to thank Professor Wolf-Peter Buchwald for giving me the opportunity of doing this Final Project and contacting with Dirk Reifenstahl, who has supervised my work with flexibility and wisdom.

I would like to thank my new friends of Wolfenbüttel and Braunschweig for having given me support and advices in the development of this Final Thesis and for having gone with me in my stay in Germany. And also, to thank my UC3M´s classmates who have accompanied me over the years, specially Alex and Julio, who have borne and help me at long evenings of study in the library.

Finally I would like to thank all my family but specially, my father Jesús Manuel, my mother María del Carmen, my sister Lara and my girlfriend Silvia for having been always there for me and because without them all of this effort would not have any sense.

Thank you very much!

AGRADECIMIENTOS

Este Proyecto de Fin de Carrera es el final de una de las etapas más importantes de mi vida, y aunque he tenido algunas malas experiencias, creo que han sido más los buenos momentos, y esto es verdaderamente importante.

Me gustaría agradecer a mi profesor Wolf-Peter Buchwald el haberme dado la oportunidad de hacer este proyecto, y ponerme en contacto con mi tutor Dirk Reifenstahl, quién me ha supervisado con flexibilidad y sabiduría.

Agradecer a mis nuevos amigos de Wolfenbüttel y Braunschweig por haberme apoyado y aconsejado con este proyecto, y por haberme acompañado en mi estancia en Alemania. Y también, cómo no, a todos mis compañeros de carrera que me han acompañado a lo largo de los años, especialmente a Alex y Julio, quienes me han aguantado y aconsejado en las largas tardes de estudio en la biblioteca.

Finalmente me gustaría dar las gracias a toda mi familia, pero especialmente a mi padre Jesús Manuel, mi madre María del Carmen, mi hermana Lara y mi novia Silvia por haber estado siempre ahí cuando les he necesitado, y porque sin ellos todo este esfuerzo no habría tenido ningún sentido.

Gracias a todos.

ABSTRACT

This project raises the necessity to use the image compression currently, and the different methods of compression and codification. Specifically, it will deepen the lossy compression standards with the JPEG [1] standard. The main goal of this project is to implement a Matlab program, which encode and compress an image of any format in a "jpg" format image, through JPEG standard premises.

JPEG compresses images based on their *spatial frequency*, or level of detail in the image. Areas with low levels of detail, like blue sky, are compressed better than areas with high levels of detail, like hair, blades of trees, or hard-edged transitions. The JPEG algorithm takes advantage of the human eye's increased sensitivity to small differences in brightness versus small differences in color, especially at higher frequencies. The JPEG algorithm first transforms the image from RGB to the luminance/chrominance (Y-Cb-Cr) color space, or brightness/grayscale (Y) from the two color components. The algorithm then downsamples the color components and leaves the brightness component alone.

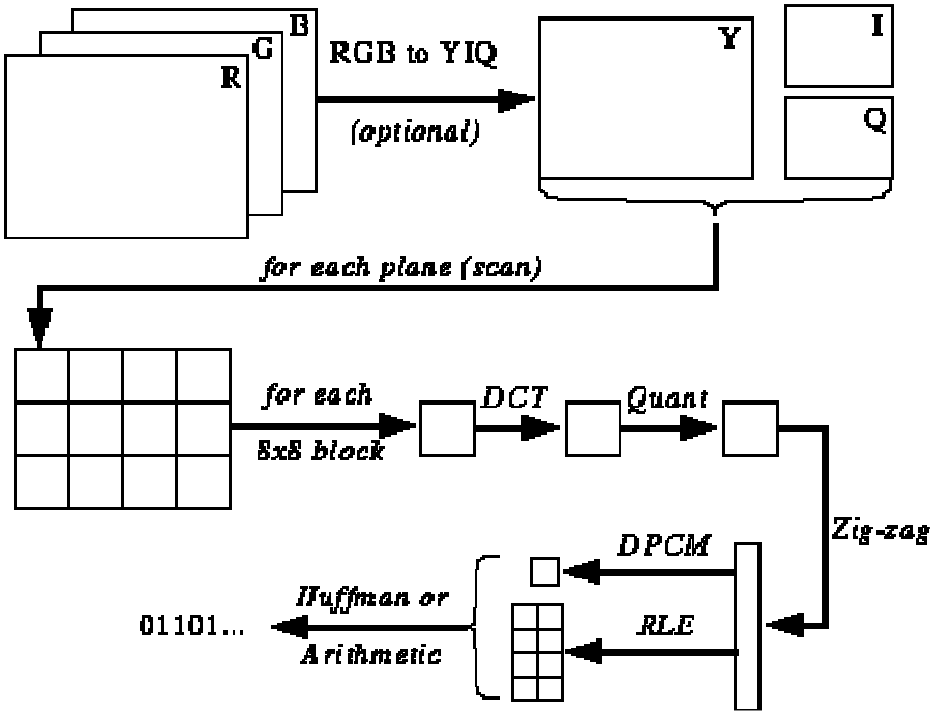


Figure 1. JPEG compression algorithm [2]

Next, the JPEG algorithm approximates 8x8 blocks of pixels with a base value representing the average, plus some frequency coefficients for nearby variations. Quantization, then downsamples these DCT coefficients. Higher frequencies and chroma are quantized by larger coefficients than lower frequencies and luminance. Thus more of the brightness information is kept than the higher frequencies and color values. So the lower the level of detail and the fewer abrupt color or tonal transitions, the more efficient the JPEG algorithm becomes.

RESUMEN

En este proyecto se aborda la necesidad de comprimir las imágenes en la actualidad, además de explicar los diferentes métodos posibles para la compresión y codificación de imágenes. En concreto, se va a profundizar en los estándares de compresión con pérdidas, mediante el estándar JPEG. El pilar central del proyecto será la realización de un programa en Matlab que codifique y comprima una imagen de cualquier formato en una imagen con formato “jpg”, mediante las premisas del estándar JPEG.

La compresión de imágenes con JPEG está basada en la frecuencia espacial, o nivel de detalle, de las imágenes. Las áreas con bajo nivel de detalle, es decir, homogéneas, se pueden comprimir mejor que áreas con gran nivel de detalle o las transiciones de los bordes. El algoritmo JPEG se aprovecha de la sensibilidad del ojo humano a pequeñas diferencias de brillo frente a las de color, especialmente con altas frecuencias. El algoritmo JPEG primero transforma la paleta de colores de la imagen RGB a un espacio de color de luminancia/crominancia (Y-Cb-Cr), o brillo/escala de grises (Y) con las dos componentes del color. El algoritmo a continuación disminuye las componentes del color y deja solo la componente del brillo.

A continuación, se aproxima la imagen en bloques de 8x8 pixeles con un valor base promedio, además de coeficientes de frecuencia de variaciones cercanas. Con la cuantificación, se disminuyen la resolución de los coeficientes de la DCT. Las frecuencias más altas y crominancias se cuantifican con los coeficientes de bajas frecuencias y luminancia. De esta forma, se mantienen mayor información de brillo que de altas frecuencias y colores. Por lo tanto, cuanto más homogénea sea la imagen (menor nivel de detalle y menos transiciones tonales abruptas) más eficiente será el algoritmo JPEG.

RESUMEN DE 10 HOJAS EN ESPAÑOL

En este proyecto se aborda la necesidad de comprimir las imágenes en la actualidad, además de explicar los diferentes métodos posibles para la compresión y codificación de imágenes. En concreto, se va a profundizar en los estándares de compresión con pérdidas, mediante el estándar JPEG. El pilar central del proyecto será la realización de un programa en Matlab que codifique y comprima una imagen de cualquier formato en una imagen con formato “jpg”, mediante las premisas del estándar JPEG.

1.-Prólogo de la compresión de imagen

Una cantidad ingente de datos es producida cuando una función de luz 2-D es muestreada y cuantificada para crear una imagen digital. De hecho, la cantidad de datos generados puede llegar a ser tan grande que no resulte práctico su almacenamiento, procesamiento o comunicación. En tales casos, la representación más allá del simple muestreo 2-D y cuantificación del nivel de gris son necesarios.

La compresión de la imagen aborda el problema de reducción de la cantidad necesaria de datos para representar una imagen digital. La base fundamental del proceso de reducción es la eliminación de datos redundantes. Desde el punto de vista matemático, esto equivale a la transformación de una matriz de pixeles 2-D en un conjunto de datos estadísticamente correlados. La transformación de la imagen se aplica antes de su almacenamiento o transmisión. Posteriormente, la imagen comprimida se descomprimirá para reconstruir la imagen original, o en su defecto una aproximación de la misma.

El interés en la compresión de imágenes se remonta a más de 40 años. El enfoque inicial de los esfuerzos de investigación en este campo fue el desarrollo de métodos para reducir el ancho de banda analógico de transmisión de vídeo, un proceso llamado *bandwidth compression*. Sin embargo, la llegada del ordenador digital y el posterior desarrollo de circuitos integrados avanzados, causó un gran interés en el cambio de lo analógico a los métodos de compresión digital. Gracias a la adopción de diversas normas fundamentales internacionales de compresión de imagen, este campo ha experimentado

una etapa de crecimiento a través de la aplicación práctica del trabajo teórico que se inició en la década de 1940, cuando C.E. Shannon y otros formularon por primera vez el punto de vista probabilístico de la información y su representación, transmisión y compresión.

2.-Estado del arte de la compresión de imagen

Para nuestro propósito, una imagen es una secuencia bidimensional de valores de la muestra con extensión finita. El término “pixel”, usado aquí, se entiende como sinónimo de una muestra de la imagen. La primera coordenada, n1, hace referencia al índice de la fila, mientras que la coordenada n2 hace referencia al índice de la columna de la muestra, o pixel. El valor de la muestra $x[n_1,n_2]$ representa la intensidad (brillo) de la imagen en la posición $[n_1,n_2]$. Los valores de las muestras serán B-bits con o sin signo.

Las imágenes digitales comúnmente se representan mediante un conjunto de 8 bits sin signo, aunque largas cadenas de bits suelen encontrarse en aplicaciones médicas, militares o científicas. En muchos casos, los B-bits valores de las muestras se interpretan como representaciones uniformemente cuantificadas de los valores reales en el intervalo de 0 a 1 (para el caso sin signo) o de $-0´5$ a $0´5$ (con signo).

Las imágenes a color normalmente se suelen representan con tres valores por cada punto de muestreo, correspondientes a las componentes de los colores primarios rojo, verde y azul. Representamos este tipo de imágenes con tres secuencias muestrales por separado, $x_R[n_1,n_2]$, $x_G[n_1,n_2]$, $x_B[n_1,n_2]$. De forma más general, podemos tener una colección arbitraria de componentes de imagen.

Las imágenes destinadas a imprimirse en color a veces tienen cuatro componentes, correspondientes al cian, magenta, amarillo y negro; de hecho, algunas impresoras a color añaden el verde y el violeta creando seis colores de componentes primarios. Las imágenes hiperespectrales de satélite pueden llegar a tener cientos de componentes de imagen, correspondientes a diferentes regiones del espectro. En su mayor parte, centraremos nuestra atención en una única componente de imagen, con el fin de comprender que

siempre es posible aplicar un sistema de compresión para cada componente por separado en cada paso.

El grado con el que una imagen puede ser comprimida depende de su contenido. Por esta razón, a menudo nos referimos a determinados grupos de imágenes. Una posible clasificación es:

- **Naturales:** Imágenes de escenas naturales, incluyendo imágenes fotográficas.
- **Textos:** Imágenes escaneadas o textos generados por ordenador.
- **Gráficos:** Imágenes escaneadas o gráficos generados por ordenador.
- **Compuestas:** Imágenes que contienen una mezcla de las tres clasificaciones anteriores.

A menos que se indique lo contrario, en este proyecto nos centraremos principalmente en las imágenes naturales.

3.- Compresión sin y con pérdidas

El objetivo principal de la compresión sin pérdida es minimizar el número de bits requeridos para representar la imagen original sin ninguna pérdida de información. Todos los B bits de cada muestra deben ser reconstruidos a la perfección durante la descompresión. Para la compresión de una imagen, en ocasiones, cierta pérdida de información es tolerada por las siguientes razones:

- Una pérdida significante de información puede ser tolerada por el sistema visual humano sin que interfiera con la percepción del contenido de la imagen.
- En la mayoría de los casos, la entrada digital del algoritmo de compresión es en sí mismo una representación imperfecta del mundo real. Esto es especialmente cierto cuando los valores de la muestra de la imagen son versiones cuantificadas de la base de valores reales de cuantificación.
- La compresión sin pérdida normalmente es incapaz de alcanzar los altos requerimientos de compresión requeridos por las aplicaciones de almacenaje y distribución.

La compresión sin pérdida se aplica también a menudo en casos en que es difícil determinar cómo introducir una pérdida aceptable que aumente el nivel de compresión. En las imágenes con colores de la paleta cromática, por ejemplo, un pequeño error numérico en el valor de una muestra puede tener un efecto drástico en la representación del color. Por último, la compresión sin pérdida también puede ser apropiada cuando la imagen va a ser editada y recomprimida sucesivas veces, por lo que la acumulación de errores, tras múltiples compresiones con pérdidas, puede llegar a ser inaceptable.

Al permitir la introducción de pequeños errores, es natural esperar que podamos ser capaces de representar de forma aproximada la imagen, usando un menor número de bits dentro de lo posible, con una menor compresión. El objetivo principal de la compresión con pérdida es minimizar el número de bits necesarios para representar una imagen con un nivel permisible de distorsión. La distorsión, por supuesto, deberá ser tratada de forma apropiada.

4.- El comité JPEG y sus estándares

El “Joint Photographic Expert Group” se formó en 1986 como un comité conjunto para la investigación y el desarrollo en la digitalización de imágenes, bajo los auspicios de la ISO y el ITU-T. Este comité ha creado muchos estándares desde que fue creado.

El estándar JPEG

El estándar JPEG se estableció en 1992 como un medio de añadir fotos a los terminales de texto y fue aprobado por el grupo en 1994. En 1993 el grupo desarrolló un JPEG sin pérdidas usando un algoritmo diferente para tratar de preservar la calidad de la imagen al mismo tiempo que se reducía el tamaño de la imagen.

El primer estándar publicado, conocido como JPEG, fue muy popular gracias a su facilidad, rapidez y eficacia. De hecho, tuvo una rápida y amplia difusión gracias a Internet. Además de ser un método de compresión, también se le considera como un formato de archivo de imagen. JPEG/Exif es el formato de imagen más común usado por las cámaras digitales y otros dispositivos de captura de imagen fotográfica, junto con el JPEG/JFIF, formato de almacenamiento y transmisión de imágenes fotográficas más

común en la “World Wide Web” (WWW). Estas variaciones de formatos normalmente no se distinguen, y son llamados simplemente JPEG.

JPEG es un método común de compresión con pérdida de imágenes fotográficas. Este método de codificación de imágenes descarta parte de los datos, dando como resultado un conjunto de datos que, al ser descomprimidos, son diferentes al original, pero lo suficientemente parecidos como para ser útiles. El grado de compresión puede ser ajustado, lo que permite una selección aceptable entre el tamaño de almacenamiento y la calidad de la imagen. Si especificamos un gran nivel de compresión, perderemos una cantidad significante de calidad, pero obtendremos un archivo pequeño. Por el contrario, con una tasa de compresión baja, obtendremos una calidad muy similar a la original, con menor tamaño de archivo. Esta compresión puede ser ejecutada múltiples veces, pero el factor de calidad obtenido irá siendo peor cada vez que se descomprima y se vuelva a comprimir la imagen (ya que la pérdida de información será mayor).

Este sistema de compresión se basa en la reducción de la información tomando un promedio de las zonas degradadas, esto es, el valor de los colores de algunos pixeles son calculados basándose en los colores de los pixeles vecinos. Por esta razón, este formato es muy eficiente en el almacenamiento de imágenes que tienen muchos degradados y matices de color.

Las principales características del primer estándar fueron:

- El uso de la Transformada Discreta del Coseno (DCT), basada en la compresión con pérdidas y la codificación Huffman con restricción de entrada de 8bpp.
- Una extensión para el control de la mejora de las imágenes.
- Un método sin pérdidas con una codificación predictiva y códigos Huffman o aritméticos.

El estándar JPEG-LS

JPEG-LS es un simple y eficiente algoritmo base que consiste en dos etapas distintas e independientes llamadas modelado y codificación. Este estándar fue desarrollado con el objetivo de proporcionar un sistema de compresión de imagen de baja complejidad sin pérdidas que pudiera ofrecer una mejor eficiencia de compresión que el JPEG con pérdidas. Fue desarrollado debido a que, en aquel tiempo, el estándar JPEG con

pérdidas basado en la codificación Huffman y otros estándares estában limitados por su rendimiento de compresión. La decorrelación total no podía ser alcanzada por la predicción residual del primer orden de entriopía de los estándares inferiores.

Previo a la codificación, hay dos pasos esenciales que tienen que realizarse en la fase de modelado: “decorrelación” y “modelado de errores”. En la fase de decorrelación (predicción) este algoritmo realiza una detección primitiva de bordes horizontales y verticales, examinando los píxeles vecinos del pixel a tratar. Esta simple predicción se llama predictor de Detección de Bordes de Mediana (MED) o predictor LOCO-I. En el modelado de errorres, el algoritmo JPEG-LS estima el condicional de los errores de predicción usando la correspondiente media de las mestras de cada contexto. El propósito del modelado del contenido es que las estructuras de orden superior, como los patrones de textura y la actividad local de la imagen, puedan ser explotados por el contenido modelado del error predecido.

El estándar JPEG 2000

En 1966, el comité JPEG comenzó a investigar un nuevo estándar de codificación de imagen con los últimos avances. Este proyecto se llamó JPEG 2000, el cuál utilizó las técnicas de compresión basadas en la tecnología Wavelet. Éste fue dividido en seis partes y posteriormente extendido a once partes. La primera parte (el núcleo) está ahora publicado con un Estándar Internacional, y las otras cinco partes (2-6) están completadas o casi completadas. Las últimás cinco partes (8-12) están actualmente bajo desarrollo.

El objetivo de JPEG 2000 no es solo mejorar el rendimiento del estándar JPEG, sino también añadir (o mejorar) características tales como la escalabilidad y la capacidad de edición. De hecho, la mejora de rendimiento en la compresión del JPEG 2000 en relación al estándar original JPEG es en realidad más bien modesto, y no debe ser, por lo general, la principal consideración para la evaluación del diseño. Tasas de compresión, tanto muy altas como bajas, son soportadas por el JPEG 2000.

El diagrama de bloques típico del JPEG 2000 está constituido por una etapa de pretratamiento, una transformada discreta wavelet, una etapa de cuantificación, una codificación aritmética y por último la organización de fotograma.

5.- La codificación JPEG

Como hemos mencionado anteriormente, el principal objetivo de este proyecto es la implementación de un programa con Matlab que codifique y comprima una imagen de cualquier formato en una imagen con formato “jpg”. Aparte de este codificador JPEG, hay un pequeño programa que muestra los efectos de los bloques de la DCT cuando comprimes una imagen en blanco y negro (de una sola componente) con un rudimentario método JPEG.

El proceso de codificación consta de varios pasos:

- Transformación del espacio del color.
- Reducción de la resolución.
- Separación en bloques.
- Transformada Discreta del Coseno.
- Cuantificación.
- Codificación entrópica.

A continuación, se procederá a ver en detalle los pasos del proceso JPEG.

Transformación del espacio del color.

Al principio, la imagen tiene que ser convertida del formato RGB en un espacio de color diferente, llamado $Y'C_BC_R$. Este espacio tiene tres componentes Y', C_B y C_R: la componente Y' representa el brillo (componente de luminancia) del pixel, y las componentes C_B y C_R representan la crominancia (divididas en crominancia azul y crominancia roja).

La conversión en el espacio de color $Y'C_BC_R$ permite una mayor compresión sin un efecto significativo sobre la calidad de la imagen. La compresión es más eficiente debido a que la información de brillo, que es más importante para la calidad final de la percepción de la imagen, se limita a un solo canal. Más concretamente, se corresponde con la percepción del color del sistema visual humano. La transformación del color también mejora la compresión de la decorrelación estadística.

Reducción de la resolución

Debido a la densidad del color (y brillo) para los receptores sensibles de los ojos humanos, los humanos pueden ver muchos más detalles finos en el brillo de una imagen (componente Y') que en el matiz y la saturación de una imagen (componentes C_B and C_R). Usando este conocimiento, los codificadores pueden diseñarse para comprimir imágenes con mayor eficiencia.

La transformación en un modelo de color $Y'C_BC_R$ permite el siguiente paso habitual, destinado a reducir la resolución espacial de las componentes C_B y C_R (llamado "reducción de la resolución" o "submuestreo de crominancia"). Las proporciones con las que normalmente se reduce la resolución de las imágenes para el formato JPEG son 4:4:4 (no hay reducción de la resolución), 4:2:2 (reducción de un factor 2 en la dirección horizontal) o, el más común, 4:2:0 (reducción de un factor 2 en las direcciones horizontal y vertical). Durante el resto del proceso de compresión, se procesan las componentes por separado y de manera muy similar.

Separación en bloques

Después del submuestreo, cada canal (imágenes digitales a color hechas de pixeles) debe ser dividido en bloques de 8x8. La división de bloques es uno de los procesos de separación de la imagen en bloques pequeños. Dependiendo del submuestreo, los bloques podrán ser de tamaño 8x8 (4:4:4 – no submuestreo), 16x8 (4:2:2) o 16x16 (4:2:0).

Si los datos de un canal no repsentan un número entero de bloques, el codificador debe rellenar el area restante de los bloques incompletos con algún tipo de datos de prueba. Rellenando los bordes con un color fijo (por ejemplo, el negro) puede crear objetos a lo largo de la parte visible del borde; repetir los pixeles en los bordes es una técnica común que reduce, aunque no elimina por completo necesariamente, estos objetos, y otras técnicas más sofisticadas pueden ser aplicadas.

Transformada discreta del coseno

Después, cada bolque de 8x8 de cada componente (Y', C_b, C_r) se convierte en una representación domino-frecuencia, usando una normalización bidimensional de tipo II de la transformada discreta del coseno (DCT). La DCT transforma el bloque 8x8 en una combinación lineal de 64 patrones.

La DCT está estrechamente relacionada con la transformada de Fourier discreta. Se trata de una transformación lineal separable, es decir, la transformación de dos dimensiones es equivalente a una DCT unidimensional realizada a lo largo de una sola dimensión, seguido de otra DCT unidimensional en la otra dimensión. La DCT tiende a concentrar la información, lo que es útil para aplicaciones de compresión de imagen.

Antes de calcular la DCT de los bloques 8x8, sus valores se desplazan de un rango positivo a uno centrado en torno a cero. Para una imagen de 8 bits, cada entrada en el bloque original pertenece al rango [0,255]. El punto medio del rango, en este caso es el valor 128, se resta de cada entrada dando como resultado un rango de datos que se centra en torno a cero, por lo que el rango final queda [-128,127]. Este paso reduce en rango dinámico requerido por la etapa de procesamiento DCT.

El siguiente paso es tomar la DCT bidimensional y llevar a cabo esta transformación en la matriz. La entrada de la esquina superior izquierda es la mayor magnitud. Éste es el coeficiente DC. Los 63 coeficientes restantes se llaman coeficientes AC. La ventaja de la DCT es su tendencia a agregar la mayor parte de la información en una esquina. La siguiente etapa de cuantificación acentuará este efecto al mismo tiempo que se reducirá el tamaño total de los coeficientes DCT, dando como resultado una señal fácil de comprimir eficientemente en la etapa de la entropía.

La DCT aumenta temporalmente la profundidad de bits de los datos, ya que los coeficientes DCT de una imagen de 8 bits/componente pueden tomar once o más bits (dependiendo de la fidelidad de los cálculos de la DCT) para almacenar.

Cuantificación

El ojo humano es bueno para ver las pequeñas diferencias de brillo en un área relativamente grande, pero no tan bueno en distinguir de forma exacta variaciones de frecuencia de alto brillo. Esto permite reducir considerablemente la cantidad de información en las componentes de alta frecuencia. Esto se hace simplemente dividiendo cada componente en el dominio de la frecuencia por una constante para esa componente, y redondeando al entero más cercano. Esta operación de redondeo es la única operación con perdidas de todo el proceso, siempre que la DCT se realice con la precisión adecuada.

Como resultado, normalmente muchas de las componentes de altas frecuencias se redondean a cero, y la mayor parte restante son números positivos o negativos pequeños, los cuales representan un menor número de bits.

Codificación entrópica

La codificación entrópica es una forma especial de compresión sin pérdida de datos. Los componentes de la imagen se organizan, mediante un *zig-zag*, en un único vector sobre el que se les aplicará un algoritmo de codificación "run-lenght" (RLE), que agrupará las frecuencias similares, insertando longitudes de ceros de codificación, y luego usando la codificación Huffman en lo restante.

Este modo de codificación se denomina codificación de base secuencial. "Baseline" JPEG también admite la codificación progresiva. La diferencia entre la codificación secuencial y la progresiva es que, mientras la primera codifica los coeficientes de un solo bloque a la vez (de manera zig-zag), el segundo codifica los coeficientes con igual posición de todos los bloques de una sola vez, seguido de los coeficientes posicionados a continuación, y así sucesivamente.

Con el fin de codificar sobre el patrón de coeficientes generado, JPEG usa la codificación Huffman. El código JPEG representa una combinación del número de bits significantes del coeficiente, incluyendo el signo, y el número consecutivo de los coeficientes cero que le preceden. El esquema Huffman usa una tabla de ocurrencia de frecuencia por cada símbolo (o carácter) en la entrada. Esta tabla puede derivar de la propia entrada o de datos que son representativos de la entrada. Por tanto, necesitamos asignar una cadena de bits de longitud variable a cada carácter, que inequívocamente represente cada carácter. Esto significa que la codificación de cada carácter debe tener un único prefijo. Los caracteres que se van a codificar se organizan en un árbol binario. Para cada carácter, se tiene una codificación que se puede encontrar siguiendo la ruta del árbol: la codificación es la cadena de símbolos en cada rama siguiente.

Por último, obtenemos la imagen codificada JPEG escribiendo la secuencia de codificación Huffman, con un gran resultado de calidad y una considerable reducción del tamaño original.

CONTENTS

CHAPTER 1 - INTRODUCTION

1.1 Preface of Image compression

An enormous amount of data is produced when a 2-D light intensity function is sampled and quantized to create a digital image. In fact, the amount of data generated may be so great that it results in impractical storage, processing, and communications requirements. In such cases, representations beyond the simple 2-D sampling and gray-level quantization are needed. For instance, more than 25 Gb (25×10^9 bytes) of data are required to represent the *Encyclopaedia Britannica*[1] in digital form.

Image compression addresses the problem of reducing the amount of data required to represent a digital image. The underlying basis of the reduction process is the removal of redundant data. From a mathematical viewpoint, this amounts to transforming a 2-D pixel array into a statistically uncorrelated data set. The transformation is applied prior to storage or transmission of the image. At some later time, the compressed image is decompressed to reconstruct the original image or an approximation to it.

Interest in image compression dates back more than 40 years. The initial focus of research efforts in this field was on the development of analog methods for reducing video transmission bandwidth, a process called *bandwidth compression*. The advent of the digital computer and subsequent development of advanced integrated circuits, however, caused interest to shift from analog to digital compression approaches. Thanks to the adoption of several key international image compression standards, this field has experimented a poised for significant growth through the practical application of the theoretic work that began in the 1940s, when C. E. Shannon and others first formulated the probabilistic view of information and its representation, transmission, and compression.

[1] The Encyclopaedia Britannica contains about 25,000 pages. A single page scanned at 300 dots per inch and quantized to two levels generates more than 8,000,000 bits (1,000,000 bytes) of data.

Over the years, the need for image compression has grown steadily. Currently, it is recognized as an “enabling technology”. For example, image compression has been and continues to be crucial to the growth of multimedia computing and for handling the increased spatial resolution of imaging sensors. Furthermore, image compression plays a crucial role in many important and diverse applications, including televideoconferencing, remote sensing (the use of satellite imagery for weather and other earth-resource applications), document and medical imaging, even the control of remotely piloted vehicles in military, space, and hazardous waste control applications. In short, an ever-expanding number of applications depend on the efficient manipulation, storage, and transmission of binary, gray-scale or color images.

Since compression is achieved by the removal of one or more of three basic data redundancies: *(1) coding redundancy*, which is present when less than optimal (i.e., the smallest length) code words are used; *(2) interpixel redundancy*, which results from correlations between the pixels of an image; and/or *(3) psychovisual redundancy*, which is due to data that is ignored by the human visual system (i.e., visually nonessential information); we will examine each of these redundancies, and examine the compression standards: JPEG, JPEG-LS and JPEG2000.

1.2 Goals

In this project we will study the main characteristics of the different standard published by the Joint Photographic Expert Group. Specifically, it will deepen the lossy compression standards with the JPEG standard. Before of this, we will remember some elementary concepts about the digital images, and the main characteristics and differences between the lossless compression and the lossy compression.

The main goal of this project will be to implement a Matlab program which encodes and compresses an image of any format in a “jpg” format image, through JPEG standard premises.

CHAPTER 2 – IMAGE COMPRESSION OVERVIEW

2.1 Elementary concepts of digital images

For our purposes an image is a two dimensional sequence of sample values,

$$x[n_1,n_2], \quad 0 \le n_1 < N_1 \quad 0 \le n_2 < N_2$$

having finite extents, N_1 and N_2, in the vertical and horizontal directions, respectively. The term "pixel," where used here, is to be understood as synonymous with an image sample. The first coordinate, n_1 is understood as the row index, while the second coordinate, n_2, is understood as the column index of the sample or pixel. This is illustrated in Figure 2.

The sample value, $x[n_1,n_2]$ represents the intensity (brightness) of the image at location $[n_1,n_2]$. The sample values will usually be B-bits signed or unsigned integers. Thus,

$$x[n_1,n_2] \in \{ 0, 1,\ldots, 2^B - 1 \} \text{ for unsigned imagery}$$
$$x[n_1,n_2] \in \{ -2^{B-1}, -2^{B-1} + 1,\ldots, 2^{B-1} - 1 \} \text{ for signed imagery}$$

Most commonly encountered digital images have an unsigned $B = 8$ bit representation, although larger bit-depths are frequently encountered in medical, military and scientific applications. In many cases, the B-bit sample values are best interpreted as uniformly quantized representations of real-valued quantities, $x'[n_1,n_2]$, in the range 0 to 1 (unsigned) or $-\frac{1}{2}$ to $\frac{1}{2}$ (signed). Letting $\langle \cdot \rangle$ denote rounding to the nearest integer, the relationship between the real-valued and integer sample values may be written as

$$x[n_1,n_2] = \langle 2^B x'[n_1,n_2] \rangle \tag{1.1}$$

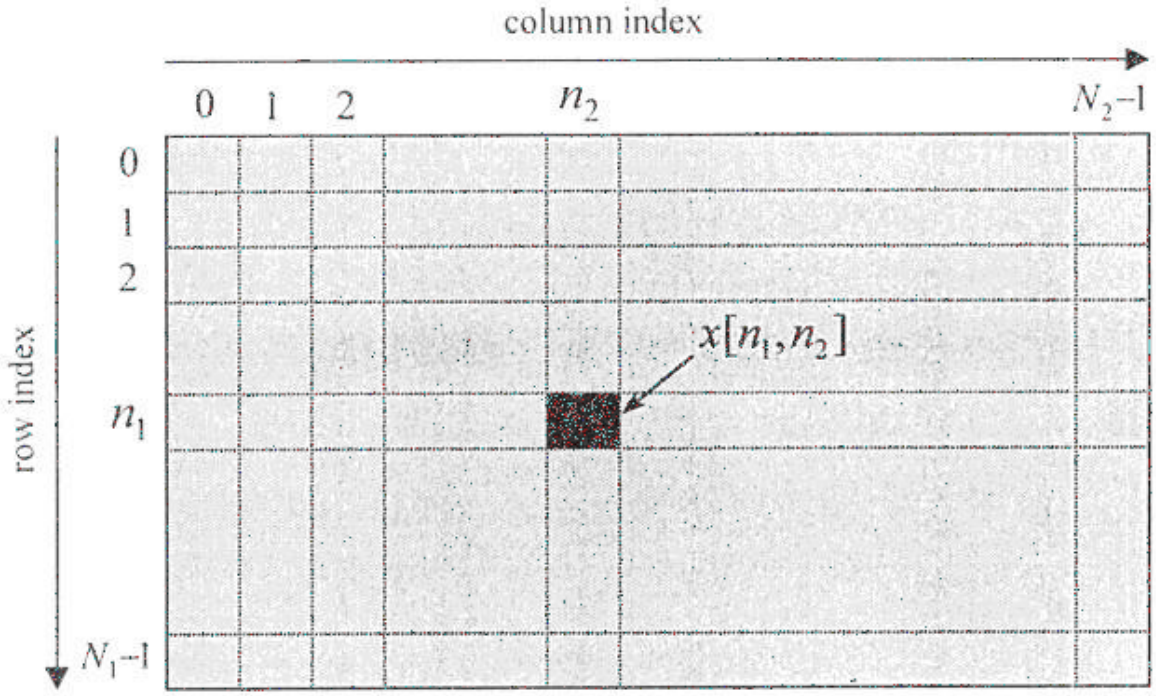


Figure 2. Interpretation of image sample coordinates [4]

Colour images are typically represented with three values per sample location, corresponding to red, green and blue primary colour components. We represent such images with three separate sample sequences, $x_R[n_1,n_2]$, $x_G[n_1,n_2]$, $x_B[n_1,n_2]$. More generally, we may have an arbitrary collection of image components,

$$x_c[n_1,n_2], \quad c = 1, 2, \ldots, C$$

Images prepared for colour printing often have four colour components corresponding to cyan, magenta, yellow and black dyes; in fact, some colour printers add green and violet for six primary colour components. Hyperspectral satellite images can have hundreds of image components, corresponding to different regions of the spectrum. For the most part we shall restrict our attention to a single image component, with the understanding that it is always possible to apply a compression system separately to each component in turn.

The degree to which an image may be compressed depends upon its content. For this reason, we often refer to particular classes of imagery. Some useful classifications are:

- **Natural:** Images representing natural scenes, including photographic images.
- **Text:** Images representing scanned or computer generated text, e.g., fac-simile images.
- **Graphics:** Scanned or computer generated graphics such as line-art and comics.
- **Compound:** Images which typically contain a mixture of above three types of content, e.g., scanned documents.

Unless otherwise stated, we will be primarily concerned with natural images in this project.

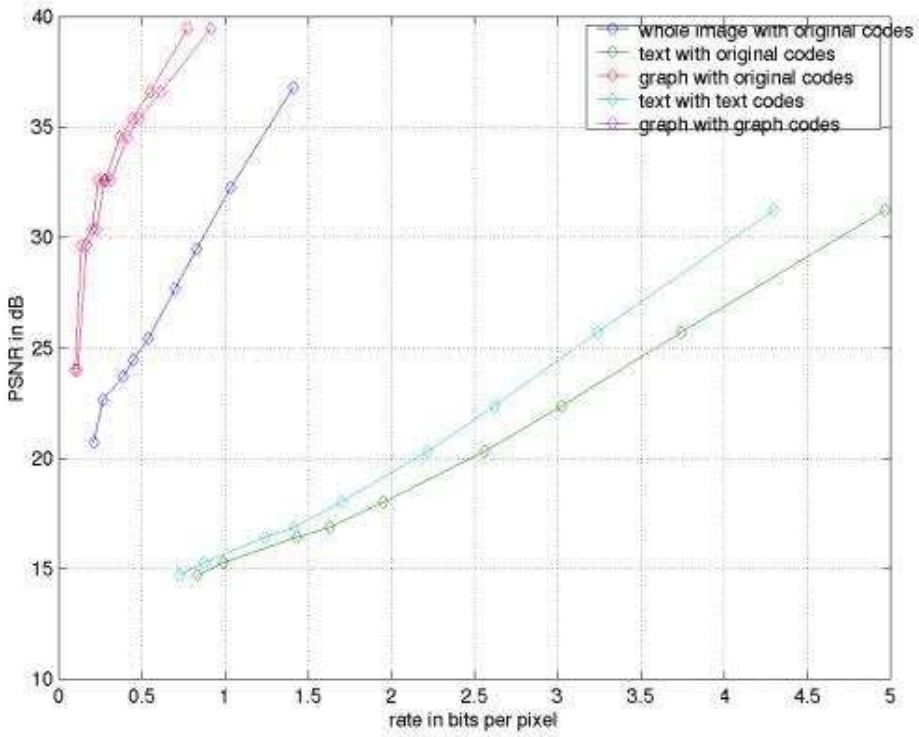


Figure 3. Rate vs. PSNR for segments of a compound image [5]

2.2 Lossless and lossy compression

The primary goal of lossless compression is to minimize the number of bits required to represent the original image samples without any loss of information. All *B* bits of each sample must be reconstructed perfectly during decompression. For image compression, however, some loss of information is usually acceptable for the following three reasons:

- Significant loss can often be tolerated by the human visual system without interfering with perception of the scene content.
- In most cases, digital input to the compression algorithm is itself an imperfect representation of the real-world scene. This is certainly true when the image sample values are quantized versions of underlying real-valued quantise, as expressed in equation (1.1).
- Lossless compression is usually incapable of achieving the high compression requirements of many storage and distribution applications.

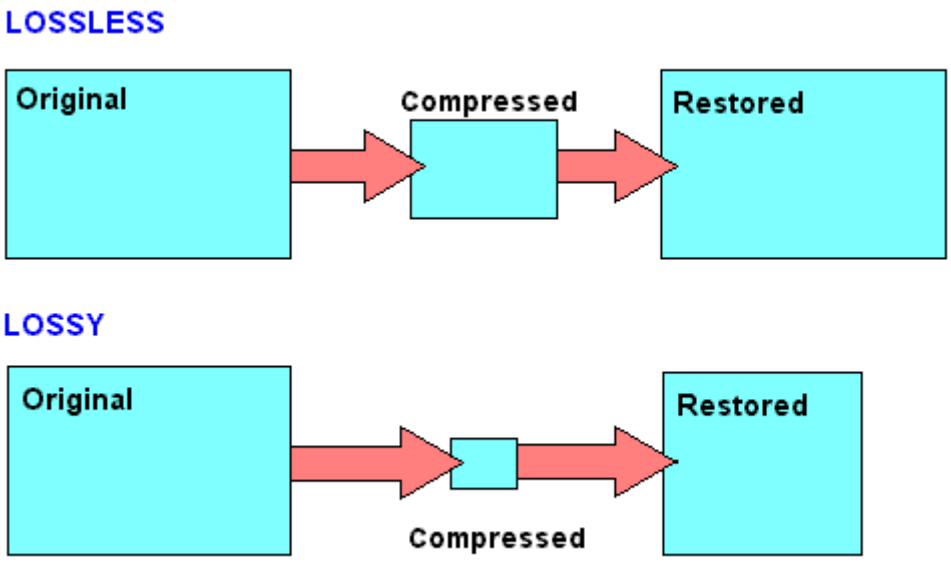


Figure 4. Lossless vs. Lossy compression [6]

Lossless compression is also often applied in cases where it is difficult to determine how to introduce an acceptable loss which will increase compression. In palettized colour images, for example, a small error in the numeric sample value may have a drastic effect upon the colour representation. The highly structured nature of non-natural imagery such as text and graphic usually renders it more amenable to lossless compression. Finally, lossless compression may be appropriate in applications where the image is to be extensively edited and recompressed so that the accumulation of errors from multiple lossy compression operations may become unacceptable.

By allowing the introduction of small errors, it is natural to expect that we should be able to represent the image approximately using a smaller number of bits than is possible within the constraints of less compression. The more distortion we allow, the smaller the compressed representation can be. The primary goal of lossy compression is to minimize the number of bits required to represent an image with an allowable level of distortion. Distortion of course must be assessed in an appropriate manner. Formally, we write $D(x,\hat{x})$, for the distortion between the original image, $x \equiv x[n_1,n_2]$, and the reconstructed image, $\hat{x} \equiv \hat{x}[n_1,n_2]$.

The most commonly employed measure of distortion is MSE (Mean Squared Error), defined by

$$\text{MSE} \equiv \frac{1}{N_1 N_2} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} (x[n_1,n_2] - \hat{x}[n_1,n_2])^2$$

For image compression, the MSE is most commonly quoted in terms of the equivalent reciprocal measure, PSNR (Peak Signal to Noise Ratio), defined by

$$\text{PSNR} \equiv 10\log_{10} \frac{(2^B - 1)^2}{MSE}$$

The PSNR is expressed in dB (decibels). Good reconstructed images typically have PSNR values of 30 dB or more.

The popularity of MSE as a measure for image distortion derives partly from the ease with which it may be calculated and partly from the tractability of linear optimization problems involving squared error metrics.

2.3 Gamma function correction

However, it is worth pointing out the importance of non-linearities in the most commonly encountered image representations, since display devices such as televisions and computer monitors are highly non-linear in that, the excitation power delivered to the phosphor is approximately proportional to v^{γ}, where v is the control voltage applied to the electron gun and γ typically ranges from about 1.8 to 2.8. The image sample values, $x[n_1,n_2]$ are usually assigned so as to compensate for such a non-linearity.

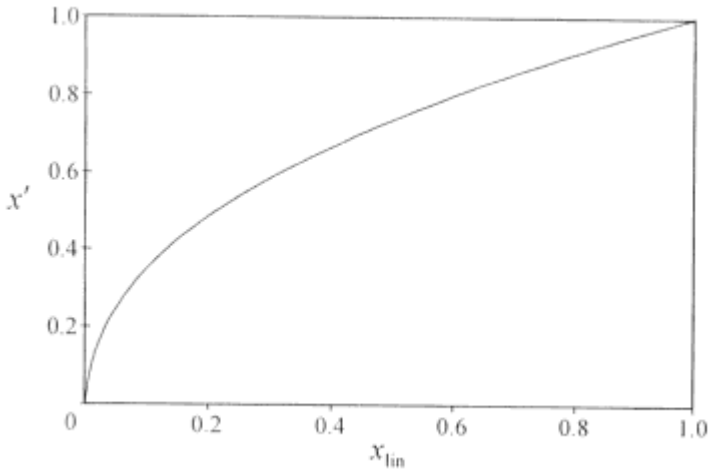


Figure 5. The sRGB gamma function

More specifically, let $x_{\text{lim}}[n_1,n_2]$ denote the normalized scene radiance at image location $[n_1,n_2]$. The normalization is such that $x_{\text{lim}} = 0$ corresponds to the absence of light and $x_{\text{lim}} = 1$ corresponds to the maximum intensity level which we expect to encounter in the scene. The so-called "gamma" function, with parameters γ and β, assigns similarly normalized image sample values, $x'[n_1,n_2]$, in the range 0 to 1, according to

$$x'[n_1,n_2] = \begin{cases} g x_{\text{lim}}[n_1,n_2] & \text{if } 0 \le x_{\text{lim}}[n_1,n_2] \le \varepsilon \\ (1+\beta)(x_{\text{lim}}[n_1,n_2])^{\frac{1}{\gamma}} - \beta & \text{if } \varepsilon \le x_{\text{lim}}[n_1,n_2] \le 1 \end{cases}$$

where the linear breakpoint, ε, and the gradient, g, are defined in terms of γ and β by

$$\varepsilon = \left(\frac{\beta}{(1+\beta)\left(1-\frac{1}{\gamma}\right)} \right)^{\gamma} \quad \text{and} \quad g = \frac{\beta}{\varepsilon(\gamma-1)}$$

These definitions ensure that the gamma function has a continuous derivative at the breakpoint, $x_{\text{lim}} = \varepsilon$.

An emerging standard for the representation of colour images is the sRGB (standard RGB) colour space, in which carefully defined linear red, green and blue primaries are each mapped to non-linear RGB sample values through the gamma function described above with parameters $\gamma = 2.4$ and $\beta = 0.055$. The function is plotted in Figure 5.

It should be noted that most digital images encountered in practice will be gamma corrected, which affects the interpretation of errors introduced in the image sample values during compression. Ignoring the small linear segment in the gamma function (or assuming that $\beta = 0$), so that $x_{\text{lim}} = (x')^{\gamma}$, we see that a small error, dx', in the gamma corrected value correspond to a scene radiance error, dx_{lim}, of

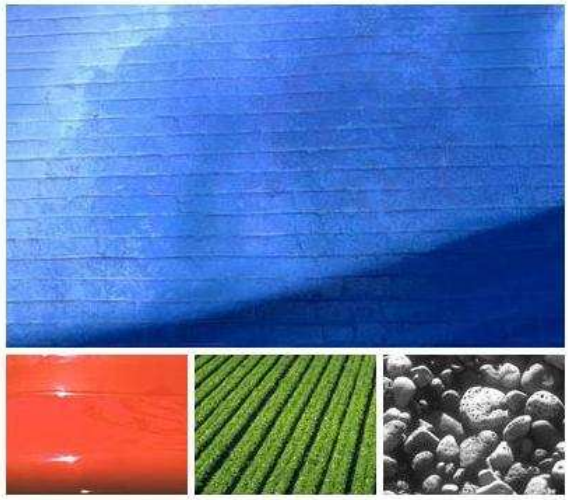
$$dx_{\text{lim}} = \gamma(x')^{\gamma-1} dx' = \gamma(x_{\text{lim}})^{1-\frac{1}{\gamma}} dx' \qquad (1.2)$$

Thus, the scene radiance error will be larger in brighter regions of the image. By a most fortune coincidence (as opposed to design), this behaviour is well matched to a property of the human visual system know as Weber's law. According to Weber's law, the change in scene radiance dx_{lim}, required to effect a just noticeable change in perceived brightness is proportional to x_{lim} itself. For large values of γ, equation (1.2) indicates that $\frac{dx_{\text{lim}}}{x_{\text{lim}}}$ is approximately proportional to dx'. Thus, the gamma corrected values ate more perceptually uniform measures of intensity than linear scene radiance, x_{lim}. In this way, the

effect of Weber's lay is automatically accommodated in simple numerical distortion measures such as MSE, provided they are applied to gamma corrected sample values.

Conversely, MSE turns out to be a much less useful measure of distortion when applied to image samples which have not been gamma corrected. Lossy compression algorithms also yield substantially poorer visual performance when applied to such images.

CHAPTER 3 – THE JPEG COMMITTEE



The Joint Photographic Expert Group [1] was formed in 1986 as a joint committee for the investigation and the development at the images digitalization, under the auspices of the ISO [11] and the ITU-T [12]. This committee has created many standards since it was created. ISO had actually started to work on this 3 years earlier, in April 1983, in an attempt to find methods to add photo quality to the text terminals of the time, but the 'Joint' that the 'J' in JPEG stands for refers to the merger of several groupings in an attempt to share and develop their experience.

The JPEG standard was established in 1992 as a means of adding photos to text terminals and was approved by the group in 1994. In 1993 the group developed Lossless JPEG using a different algorithm to try and preserve image quality while still reducing the size of the image.

3.1 The JPEG Standard

This first standard published known as JPEG, the formal name of the standard that most people refer to it, is ISO/IEC IS 10918-1 | ITU-T Recommendation T.81, as the document was published by both ISO through its national standards bodies, and CCIT, now called ITU-T. JPEG was very popular because was easy, fast and efficient. In fact, it had a quickly and widely spreading thanks to Internet. IS 10918 has actually 4 parts:

- Part 1: The basic JPEG standard, which defines many option and alternatives for the coding of still images of photographic quality.
- Part 2: Which sets rules and checks for making sure software conforms to Part 1.
- Part 3: Set up to add a set extension to improve the standard, including the SPIFF file format.
- Part 4: Defines methods for registering some of the parameters used to extend JPEG.

Besides to be a compression method, it's considered as image file format too. JPEG/Exif is the most common image format used by digital cameras and other photographic image capture devices, along with JPEG/JFIF, it's the most common format for storing and transmitting photographic images on the World Wide Web (WWW). These format variations are often not distinguished, and are simply called JPEG.

JPEG is a commonly used method of "lossy compression" for photographic images. This data encoding method discards some of the data, with the result that decompressing the data yields contents that is different from the original, although similar enough to be useful in some way. The degree of compression can be adjusted, allowing a selectable tradeoff between storage size and image quality. If we specify a very high compression then will lose a significant amount of quality, but we get small file. With a low compression rate, we get a very similar quality to the original, and a smaller file. This compression can be executed multiple times, but the quality factor obtained will usually be worse each time the image is decompressed and recompressed (since the loss information is greater).

This compression system is based on reduce information taking an average of the degraded zones, that is, the color value of some pixels are calculated based on color of pixels around these. For these reasons, this format is very efficient at storing images that have many degrades and shades of color.

The images saved with this format are prone to distorting, pixelling and to form halos around certain sectors of the image. These are more commonly in the areas where there is strong contrast between colours. Generally, if there is more contrast in an image, then needs to be saved with more quality, what will result in a decent-looking final image, opposite a lightly size reduction. This compression is not adequate for images or graphics with texts or lines, and overcoat for files with big areas of solid colours.

The main characteristics of the first standard were:

- The use of Discrete Cosine Transform (DCT), based of “lossy compression” and Huffman coding with entry restriction of 8 bpp.
- An extension for control different enhances of images.
- A lossless method with predictive coding and Huffman codes or arithmetic.

3.2 The JPEG-LS Standard

After the success of the series of JPEG standards, the committee decided it was appropriate to revisit the lossless coding mode within JPEG. This mode had been a late addition to the standard, and in the baseline form of JPEG (not using arithmetic coding) the algorithm used was not really close to 'state of the art' techniques. In addition, it really was not closely related to the block based DCT techniques which characterised the lossy compression that had become the norm. Looking at user requirements, particularly those of the medical imaging business which was concerned about potential large errors being introduced through lossy compression, the scope of the new standard was defined as effective lossless and near lossless compression of continuous-tone, grey scale and colour still images. By near lossless, it was agreed that a scheme was needed that guaranteed a minimum error between the original image data and the reconstructed image data.

A number of contenders for a suitable algorithm were proposed, and the committee carried out an extensive set of measurements to determine the best contender. The winner of these objective tests was agreed to be the *LOCO* [8] algorithm from HP Labs. As well as being the leading contender, HP and Mitsubishi both agreed to offer their patents on a royalty and license fee free basis, ensuring that implementers would hopefully not need to make any payments to implement the new standard. As well as offering effective compression, the algorithm was also relatively easy to implement efficiently in PC software, and produces fast code.

JPEG-LS is a simple and efficient baseline algorithm which consists of two independent and distinct stages called modeling and encoding. JPEG-LS was developed with the aim of providing a low-complexity lossless and near-lossless image compression standard that could offer better compression efficiency than lossless JPEG. It was developed because at the time, the Huffman coding-based JPEG lossless standard and other standards were limited in their compression performance. Total decorrelation cannot be achieved by first order entropy of the prediction residuals employed by these inferior standards.

Prior to encoding, there are two essential steps to be done in the modeling stage: “decorrelation” and “error modeling”. In the decorrelation (prediction) stage this algorithm dose a primitive edge detection of horizontal or vertical edges by examining the neighbouring pixels of the current pixel X as illustrated in Figure 6. The pixel labelled by B is used in the case of vertical edge while the pixel located at A is used in the case of a horizontal edge. This simple prediction is called the Median Edge Detection (MED) predictor or LOCO-I predictor. The pixel X is predicted according the following guesses:

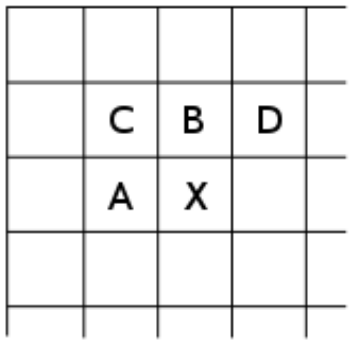


Figure 6. Three neighbouring samples around the sample to be predicted

$$X = \begin{cases} \min(A,B) & \text{if} \quad \mathrm{C} \geq \max(\mathrm{A},\mathrm{B}) \\ \max(A,B) & \text{if} \quad \mathrm{C} \leq \min(\mathrm{A},\mathrm{B}) \\ A+B-C & \text{otherwise} \end{cases}$$

In the error modeling, the JPEG-LS algorithm estimates the conditional of the prediction errors $E\{e|Ctx\}$ using corresponding sample means $\bar{e}(C)$ within each context Ctx. The purpose of context modeling is that the higher order structures like texture patterns and local activity of the image can be exploited by context modeling of the prediction error. Contexts are determined by obtaining the differences of the neighbouring samples which represents the local gradient:

$$g_1 = D - B$$
$$g_2 = B - C$$
$$g_3 = C - A$$

The local gradient reflects the level of activities such as smoothness and edginess of the neighbouring samples. Each one of the differences found in the above equation is then quantized into roughly equiprobable and connected regions. The purpose of the quantization is to maximize the mutual information between the current sample value and its context such that the high-order dependencies can be captured. One can obtain the contexts based on the assumption that

$$P(e|Ctx = [q_1, q_2, q_3]) = P(-e|Ctx = [-q_1, -q_2, -q_3])$$

After merging contexts of both positive and negative signs, the total number of contexts is $\left((2\times 4+1)^3+1\right)/2 = 365$ contexts. A bias estimation could be obtained by dividing cumulative prediction errors within each context by a count of context occurrences. This procedure is modified and improved such that the number of subtractions and additions are reduced. Prediction refinement can then be done by applying these estimates in a feedback mechanism which eliminates prediction biases in different contexts.

Finally, a major side effect of undertaking this standards activity was that some of the other contenders such as CALIC, FELICS, and Ricoh's CREW algorithm in particular had some very attractive features - for example, in the ability to provide a single code stream which could provide lossy and lossless images without additional processing. Although outside the direct scope of JPEG-LS, these features and the discussions they provoked directly led to the development of the architectural approach of the new JPEG 2000 standard. As well, the JPEG standard qualities will be soon insufficient in terms of compression quality, as the typical artefacts of the DCT, such as flexibility in terms of formation of the final frame.

3.3 The JPEG2000 Standard

In 1996, the JPEG committee [9] began to investigate a new standard of image coding with the last advances. This project was named JPEG 2000, which uses states of the art compression techniques based on wavelet technology. It was divided in six parts, subsequently extended to eleven parts. The Part 1 (the core) is now published as an International Standard, and the other five parts (2-6) are complete or nearly complete. The last 5 parts (8-12) are now under development.

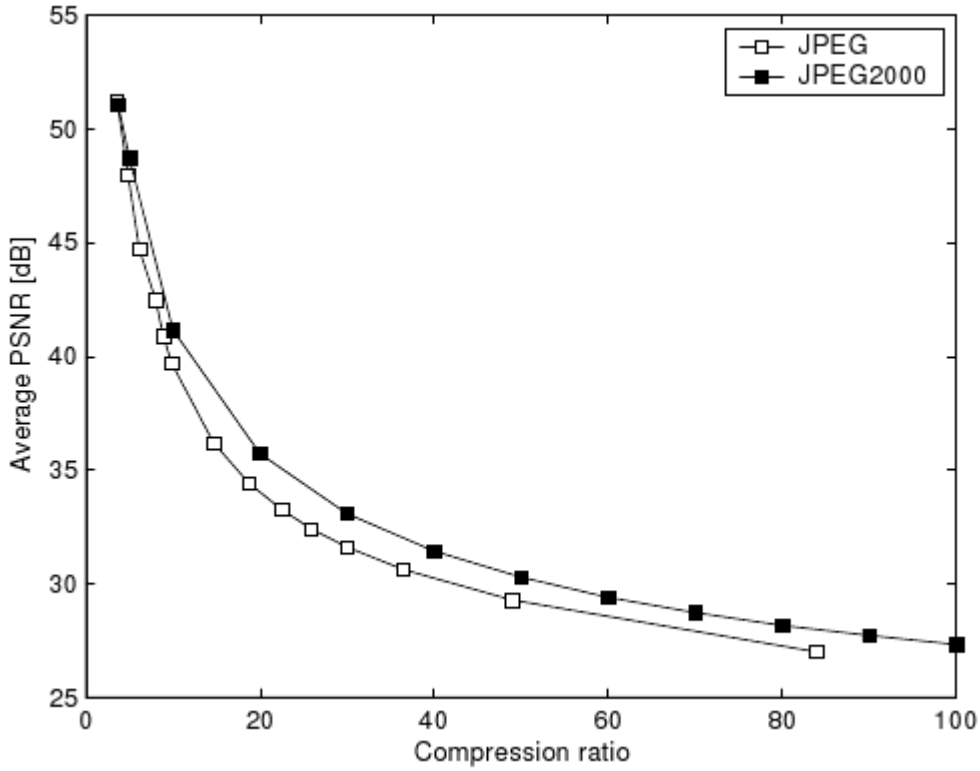


Figure 7. Average PSNR as a function of compression ratio for JPEG and JPEG2000

- Part 1: Defines the core coding system of JPEG 2000 (the syntax of codestream and the necessary steps involved in encoding and decoding JPEG 2000 images). This also defines a basic file format called JP2 and includes guidelines and examples.
- Part 2: Defines various extensions to Part 1, including more flexible forms of wavelet decomposition and coefficient quantization, an alternative way of encoding regions of particular interest, a new file format (JPX) based on JP2, and a rich metadata set for photographic imagery.

- Part 3: Motion JPEG 2000, a file format called MJ2 for motion sequences of JPEG 2000 images. It includes: storing video clips taken using digital still cameras, high-quality frame-based video recording and editing, digital cinema, medical and satellite imagery.
- Part 4: Conformance. It is concerned with testing conformance to Part 1. It specifies test procedures for both encoding and decoding processes, including the definition of a set of decoder compliance classes.
- Part 5: Contains reference software, a short text document, and two source code packages that implement Part 1. One is written in C and the other in Java. They are both available under open-source type licensing.
- Part 6: Defines the JPM file format for document imaging, which uses the Mixed Raster Content (MRC) model of ISO/IEC 16485. JPM is an extension of the JP2 file format defined in Part 1 (for JP2) and Part 2 (for JPX).
- Part 7: Has been abandoned.
- Part 8: JPSEC (security aspect). To avoid illicit uses, this is standardizing tools and solutions in terms of specifications in order to ensure the security of transaction, protection of contents (IPR), and protection of technologies (IP). For example, Encryption, Source authentication, Data integrity, Conditional Access, Ownership protection.
- Part 9: JPIP (interactive protocols and API), main client-server protocol, handles several different formats for the image data returned by the server. This includes ordinary image formats, such as complete JPEG or JPEG 2000 files, and two new types of incremental stream that use JPEG 2000's “tiles” and “precincts” to take full advantage of its scalability properties.
- Part 10: JP3D (volumetric imaging). This part (still at the Working Draft stage) is concerned with the coding of three-dimensional data, the extension of JPEG 2000 from planar to volumetric images, based on the use of non-uniform grids to concentrate the data in the regions where it is most significant.
- Part 11: JPWL (wireless applications) is standardizing tools and methods to achieve the efficient transmission of JPEG 2000 imagery over an error-prone wireless network. More specifically, JPWL extends the elements in the core coding system described in Part 1 with mechanisms for error protection and correction.

- Part 12: ISO Based Media File Format has a common text with Part 12 of the MPEG-4 standard. The format is a general format for timed sequences of media data. It uses the same underlying architecture as Apple's Quick Time file format and the JPEG 2000 file format.

The aim of JPEG 2000 is not only improving compression performance over JPEG but also adding (or improving) features such as scalability and editability. In fact, JPEG 2000's improvement in compression performance relative to the original JPEG standard is actually rather modest and should not ordinarily be the primary consideration for evaluating the design. Very low and very high compression rates are supported in JPEG 2000. For example, to reduce the number of bits for a picture below a certain amount, the advisable thing to do with the first JPEG standard is to reduce the resolution of the input image before encoding it. That's unnecessary when using JPEG 2000, because JPEG 2000 already does this automatically through its multiresolution decomposition structure.

The typical block diagram of JPEG 2000 is constituted by a pre-treatment stage image, a discrete wavelet transform, a quantization stage, an arithmetic encoding and then the frame organization.

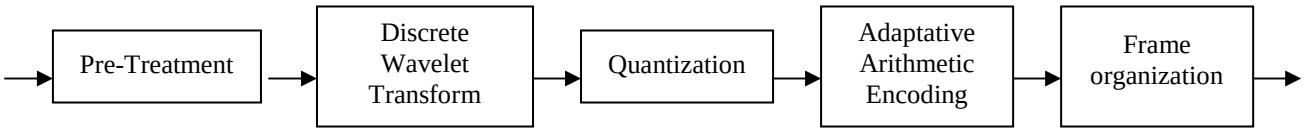


Figure 8. JPEG 2000 Block diagram

In the Pre-Treatment stage, 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, an *Irreversible Color Transform (ICT)*, uses the well known YC_BC_R color space because it has to be implemented in floating or fix-point and causes round-off errors; or the *Reversible Color Transform (RCT)*, uses a modified YUV color space that does not introduce quantization errors, so it is fully reversible.

After the 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. These tiles are then

Wavelet transformed to an arbitrary depth, in contrast to JPEG which uses an 8x8 block-size discrete cosine transform (DCT). Like the color transform, there are two different wavelet transforms: *irreversible*, because it introduces quantization noise that depends on the precision of the decoder; or *reversible*, which uses only integer coefficients. The wavelet transform are implemented by the lifting scheme or by convolution.

After the wavelet, 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).

The quantized sub-bands are split further into precincts, regular regions in the wavelet domain. Then, precincts are split further into code blocks, which are located in a single sub-band and have equal sizes. The encoder has to encode the bits of all quantized coefficients of a code block, starting with the MSB and progressing to LSB by a process called the EBCOT scheme. 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 neighbours 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. When it 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, packets can be reordered almost arbitrarily in the JPEG 200 bit-stream; this gives the encoder as well as image servers a high degree of freedom.

CHAPTER 4 – JPEG ENCODING

As mentioned earlier, the main goal of this project is to implement a Matlab program, which encode and compress an image of any format in a “jpg” format image. Apart from this JPEG encode, there is a little program which shows the DCT blocks effects when compresses a black & white image (only one component) with a rudimentary JPEG method. The program code is attached at the end of this project, in the Annex. The encoding process consists of several steps:

- Color space transformation.
- Downsampling.
- Block splitting.
- Discrete Cosine Transform.
- Quantization.
- Entropy coding.

Next, we will proceed to view in detail all the steps of the JPEG process.

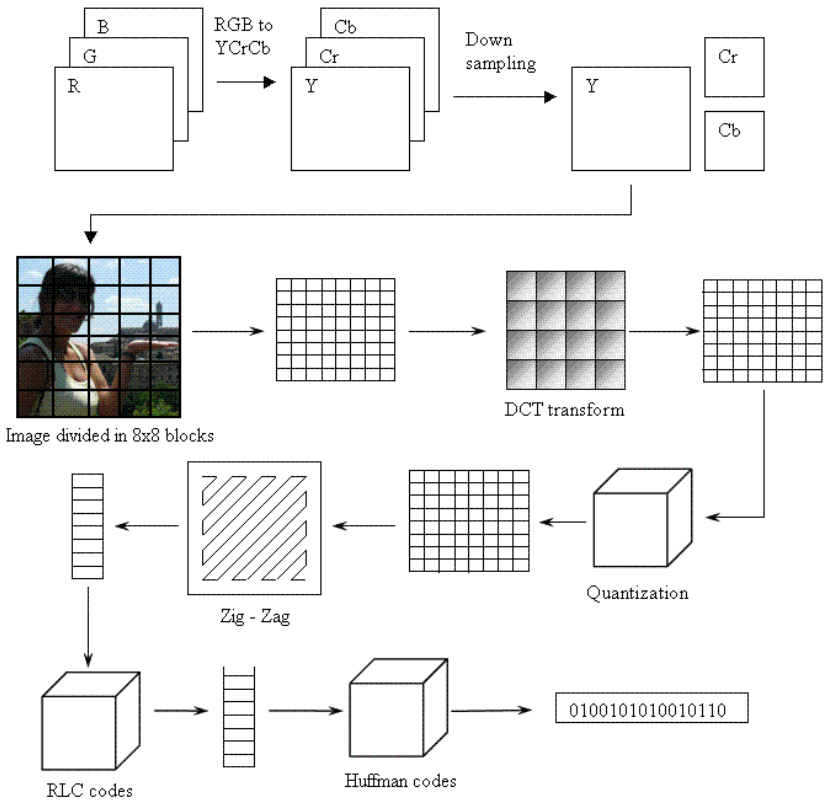


Figure 9. JPEG process diagram

4.1 Color space transformation

At first, the image has to be converted from *RGB* into a different color space called $Y'C_BC_R$. It has three components Y', C_B and C_R: the Y' component represent the brightness (luma component) of a pixel, and the C_B and C_R components represent the chrominance (split into blue-difference and red-difference chroma components).

The $Y'C_BC_R$ 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 because the brightness information, which is more important to the eventual perceptual quality of the image, is confined to a single channel. This more closely corresponds to the perception of color in the human visual system. The color transformation also improves compression by statistical decorrelation.

A particular conversion to $Y'C_BC_R$ 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 color information in the *RGB* color model, where the image is stored in separate channels for red, green and blue brightness components. This results in less efficient compression, and would not likely be used when file size is especially important.

To the JPEG conversion with JFIF, the usage of $Y'C_BC_R$, where Y', C_B and C_R have the full 8-bit range of 0-255, is

$$
\begin{aligned}
Y' &= (0.299\ R'_D) + (0.587\ R'_D) + (0.114\ B'_D) \\
C_B &= 128 - (0.168736\ R'_D) - (0.331264\ R'_D) + (0.5\ B'_D) \\
C_R &= 128 + (0.5\ R'_D) - (0.418688\ R'_D) - (0.081312\ B'_D)
\end{aligned}
$$

4.2 Downsampling

Due 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 hue and color saturation of an image (the C_B and C_R components). Using this knowledge, encoders can be designed to compress images more efficiently.

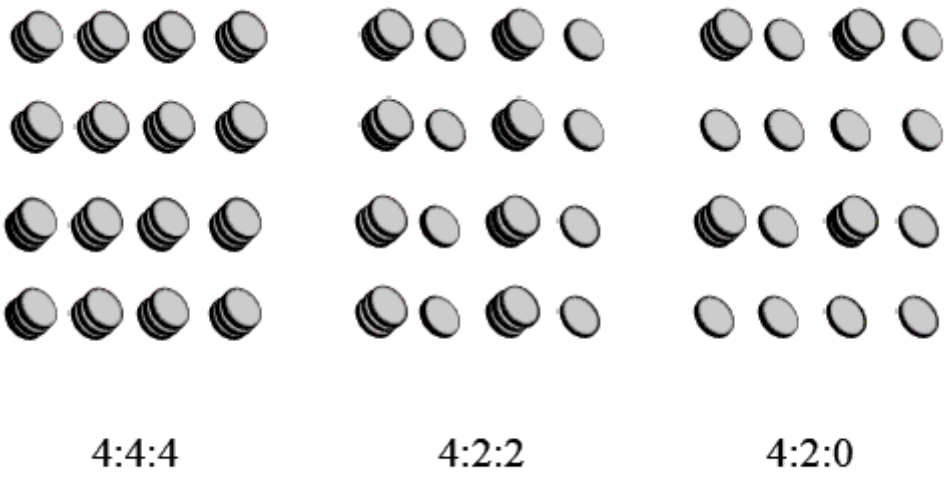


Figure 10. Color compression

The transformation into the $Y'C_BC_R$ color model enables the next usual step, which is to reduce the spatial resolution of the C_B and C_R components (called “downsampling” or “chroma subsampling”). The ratios at which the downsampling is ordinarily done for JPEG images (Figure 10) are 4:4:4 (no downsampling), 4:2:2 (reduction by a factor of 2 in the horizontal direction), or (most commonly) 4:2:0 (reduction by a factor of 2 in both the horizontal and vertical directions). For the rest of the compression process, are processed separately and in a very similar manner.

4.3 Block Splitting

After subsampling, each channel (color digital images made of pixels) must be split into 8×8 blocks. Block splitting is one of the processes that splices the image into smaller blocks. 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 edges with a fixed color (for example, black) can create ringing artifacts along the visible part of the border; repeating the edge pixels is a common technique that reduces (but does not necessarily completely eliminate) such artifacts, and more sophisticated border filling techniques can also be applied.

4.4 Discrete Cosine Transform (DCT)

Next, each 8x8 block of each component (Y', C_b, C_r) is converted to a frequency – domain representation, using a normalized, two-dimensional type-II discrete cosine transform (DCT). The DCT transform an 8x8 block to a linear combination of 64 patterns.

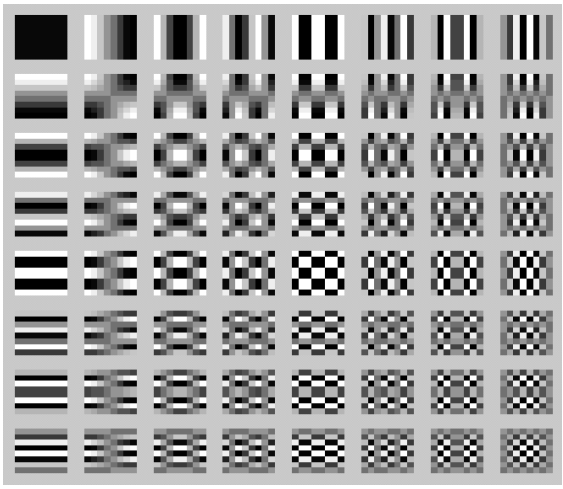


Figure 11. The 64 patterns of DCT [13]

The DCT is closely related to the discrete Fourier transform. It is a separable linear transformation; that is, the two-dimensional transform is equivalent to a one-dimensional DCT performed along a single dimension followed by a one-dimensional DCT in the other dimension. The definition of the two-dimensional DCT for an input image A and output image B is

$$B_{pq} = \alpha_p \alpha_q \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \cos\frac{\pi(2m+1)p}{2M} \cos\frac{\pi(2m+1)q}{2N}, \qquad \begin{matrix} 0 \le p \le M-1 \\ 0 \le q \le N-1 \end{matrix}$$

$$\alpha_p = \begin{cases} 1/\sqrt{M}, & p = 0 \\ \sqrt{2/M}, & 1 \le p \le M-1 \end{cases} \qquad \alpha_q = \begin{cases} 1/\sqrt{N}, & q = 0 \\ \sqrt{2/N}, & 1 \le q \le N-1 \end{cases}$$

where M and N are the row and column size of A, respectively. If you apply the DCT to real data, the result is also real. The DCT tends to concentrate information, making it useful for image compression applications.

As an example, one such 8x8 bit subimage of the program is the Figure 12.

113.2	115.8	116.8	116.2	116.5	118.5	117.4	117.2
113.2	115.8	116.8	116.2	116.5	118.5	117.4	117.2
120.5	122.5	121.9	122.5	125.5	123.5	123.4	121.2
128.4	130.5	130.3	129.6	132.0	129.7	128.2	124.7
139.3	135.3	140.2	140.3	138.4	138.3	134.6	132.1
144.6	145.6	145.6	145.3	147.3	144.9	143.2	138.6
153.9	153.9	157.2	154.7	152.8	153.6	151.6	146.2
160.6	159.3	159.6	159.0	161.2	157.4	154.8	152.6

Figure 12. 8x8 block subimage before DCT

Before computing the DCT of the 8×8 block, its values are shifted from a positive range to one centered around zero. For an 8-bit image, each entry in the original block falls in the range [0,255]. The mid-point of the range (in this case, the value 128) is subtracted from each entry to produce a data range that is centered around zero (like Figure 13), so that the modified range is [−128,127]. This step reduces the dynamic range requirements in the DCT processing stage.

-14.7	-12.1	-11.1	-11.7	-11.4	-9.4	-10.5	-10.7
-14.7	-12.1	-11.1	-11.7	-11.4	-9.4	-10.5	-10.7
-7.4	-5.4	-6.067	-5.4	-2.4	-4.4	-4.5	-6.7
0.4	2.5	2.374	1.6	4.0	1.7	0.2	-3.2
11.3	7.3	12.26	12.3	10.4	10.3	6.6	4.1
16.6	17.6	17.613	17.3	19.3	16.9	15.2	10.6
25.9	25.9	29.2	26.7	24.8	25.6	23.6	18.2
32.6	31.3	31.613	31.0	33.2	29.4	26.8	24.6

Figure 13. 8x8 block subimage centered around zero before DCT

The next step is to take the two-dimensional DCT and perform this transformation on the matrix showed in the Figure 13. We will get the result showed as Figure 14.

53.780	5.552	-9.903	3.205	-3.01	0.098	0.406	1.687
-119.949	-10.041	2.882	-2.266	-0.817	-1.738	-0.402	-1.493
11.747	-1.908	2.115	-1.273	-0.731	-0.727	0.312	-0.637
-2.333	0.111	-0.496	-0.694	-1.011	1.729	1.494	-0.852
4.566	1.778	-0.375	0.302	1.731	0.070	1.034	0.097
2.425	0.620	-0.256	-0.194	-2.113	-0.230	-1.168	0.333
1.174	-0.954	0.030	1.523	1.396	-1.912	-2.016	0.810
2.971	0.261	-0.239	-0.504	0.704	2.172	1.764	-0.352

Figure 14. 8x8 block subimage after DCT

Note the top-left corner entry with the rather magnitude. This is the DC coefficient. The remaining 63 coefficients are called 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 data, 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 effect of DCT operation can be seen as one part of the program (Figure 15). In this case, I use a Black & White image to show this special step. It can see at the right the image after apply the DCT, in 8x8 blocks of DCT coefficients.

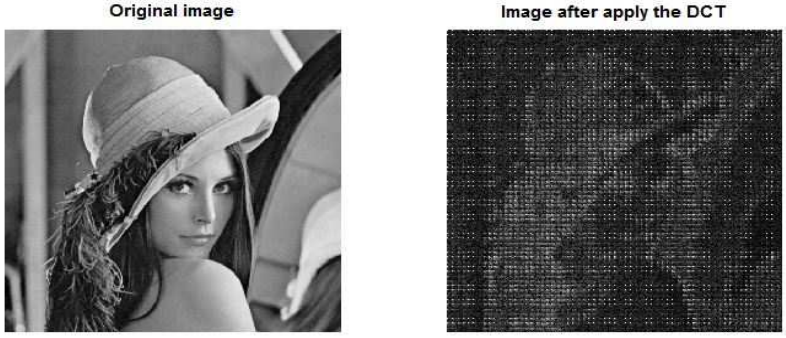


Figure 15. Image after apply the DCT

4.5 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 rounding operation is the only lossy operation in the whole process if the DCT computation is performed with sufficiently high precision. 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 represent.

In the JPEG program, I use these two matrix coefficients / quantized tables, one for luminance component and other for chrominance components.

3	2	2	3	5	8	10	12
2	2	3	4	5	12	12	11
3	3	3	5	8	11	14	11
3	3	4	6	10	17	16	12
4	4	7	11	14	22	21	15
5	7	11	13	16	21	23	18
10	13	16	17	21	24	24	20
14	18	19	20	22	20	21	20

Figure 16. Quantization table: Luminance

3	4	5	9	20	20	20	20
4	4	5	13	20	20	20	20
5	5	11	20	20	20	20	20
9	13	20	20	20	20	20	20
20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20

Figure 17. Quantization table: Chrominance

With these tables and the 8x8 matrix obtained after the DCT, the quantized DCT coefficients are computed with

$$B_{j,k} = round\left(\frac{G_{j,k}}{Q_{j,k}}\right) \quad \text{for} \;\; j = 0,1,2,...,7; \;\; k = 0,1,2,...,7$$

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, result to Figure 18.

18	3	-5	1	-1	0	0	0
-60	-5	1	-1	0	0	0	0
4	-1	1	0	0	0	0	0
-1	0	0	0	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

Figure 18. Quantized DCT coefficients

4.6 Entropy coding

Entropy coding is a special form of lossless data compression. It involves arranging the image components in a *zig-zag* (Figure 20) order employing run-length encoding (RLE) algorithm (Figure 19) that groups similar frequencies together, inserting length coding zeros, and then using *Huffman coding* on what is left.

Figure 19. Example of Run-length encoding (RLE)

The JPEG standard also allows, but does not require decoders to support, the use of arithmetic coding, which is mathematically superior to Huffman coding. However, this feature has rarely been used because it is slower to encode and decode compared to Huffman coding. Arithmetic coding typically makes files about 5-7% smaller.

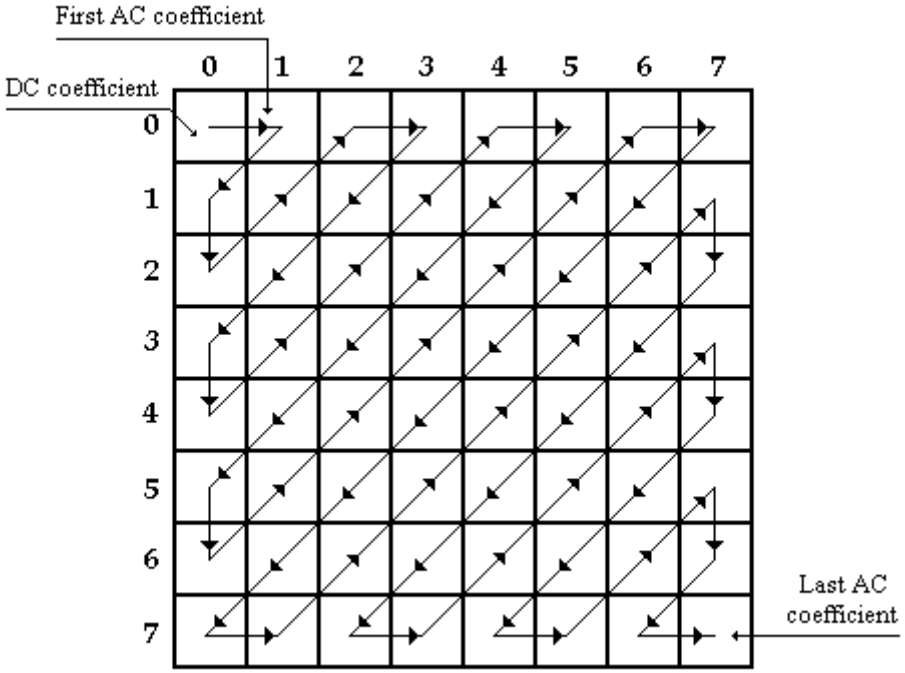


Figure 20. Zig-zag ordering

The previous quantized DC coefficient is used to predict the current quantized DC coefficient. The difference between the two is encoded rather than the actual value. The encoding of the 63 quantized AC coefficients does not use such prediction differencing.

The zig-zag sequence for the above quantized coefficients is

18	3	-60	4	-5	-5	1	1	-1	-1	1	0	1	-1	-1	0	…

Figure 21. Zig-zag ordering in a vector

If the i-th block is represented by B_i and positions within each block are represented by (p,q) where $p = 0, 1, ..., 7$ and $q = 0, 1, ..., 7$, then any coefficient in the DCT image can be represented as $B_i(p,q)$. Thus, in the above scheme, the order of encoding pixels (for the i-th block) is $B_i(0,0)$, $B_i(0,1)$, $B_i(1,0)$, $B_i(2,0)$, $B_i(1,1)$, $B_i(0,2)$, $B_i(0,3)$ and so on.

This encoding mode is called baseline sequential encoding. Baseline JPEG also supports progressive encoding. The different between sequential and progressive encoding is that, while the first encodes coefficients of a single block at a time (in a zig-zag manner), second encodes similar-positioned coefficients of all blocks in one go, followed by the next positioned coefficients of all blocks, and so on.

In order to encode the above generated coefficient pattern, JPEG uses *Huffman* encoding. JPEG's code represent combinations of the number of significant bits of a coefficient, including sign, and the number of consecutive zero coefficients that precede it. Huffman's scheme uses a table of frequency of occurrence for each symbol (or character) in the input. This table may be derived from the input itself or from data which is representative of the input. Then we need to assign a variable-length bit string to each character that unambiguously represents that character. This means that the encoding for each character must have a unique prefix. The characters to be encoded are arranged in a binary tree, like Figure 22.

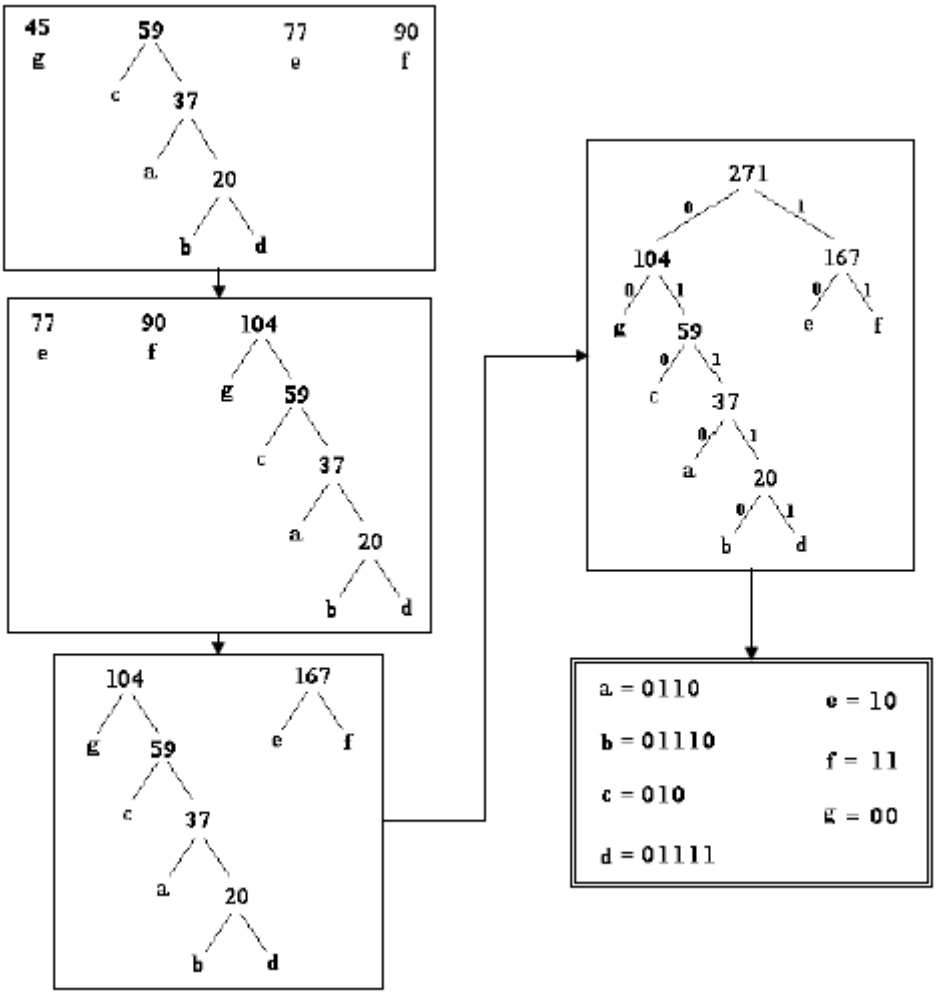


Figure 22. Example of Huffman encoding tree

An encoding for each character is found by following the tree from the route to the character in the leaf: the encoding is the string of symbols on each branch followed.

Finally, we obtain the JPEG encoding image writing the Huffman encoding sequence, with a great result of quality and size. In this program, I obtained the next results shows as Figure 23, Figure 24 and Figure 25.

Original image

Filename:	'silvia.bmp'
FileModDate:	'27-Jan-2011 09:37:22'
FileSize:	**315882**
Format:	'bmp'
FormatVersion:	'Version 3 (Microsoft Windows 3.x)'
Width:	372
Height:	283
BitDepth:	24
ColorType:	'truecolor'
FormatSignature:	'BM'
ImageDataOffset:	54
BitmapHeaderSize:	40
CompressionType:	'none'
BitmapSize:	315828

Figure 23. Information about original image

Compressed image:

Filename:	'save_default.jpg'
FileModDate:	'27-Jan-2011 09:43:21'
FileSize:	**36546**
Format:	'jpg'
Width:	372
Height:	283
BitDepth:	24
ColorType:	'truecolor,
NumberOfSamples:	3
CodingMethod:	'Huffman'
CodingProcess:	'Sequential'

Degree of compression: 88.4305 %

Figure 24. Information about compressed image

Figure 25. Original image vs. compressed image, and their sizes

CHAPTER 5 - GLOSSARY

5.1 Definitions & abbreviations

For the purposes of this project, the following definitions apply.

1- **abbreviated format:** A representation of compressed image data which is missing some or all of the table specifications required for decoding, or a representation of table-specification data without frame headers, scan headers, and entropy-coded segments.

2- **AC coefficient:** Any DCT coefficient for which the frequency is not zero in at least one dimension.

3- **(adaptive) (binary) arithmetic decoding:** An entropy decoding procedure which recovers the sequence of symbols from the sequence of bits produced by the arithmetic encoder.

4- **(adaptive) (binary) arithmetic encoding:** An entropy encoding procedure which codes by means of a recursive subdivision of the probability of the sequence of symbols coded up to that point.

5- **application environment:** The standards for data representation, communication, or storage which have been established for a particular application.

6- **arithmetic decoder:** An embodiment of arithmetic decoding procedure.

7- **arithmetic encoder:** An embodiment of arithmetic encoding procedure.

8- **baseline (sequential):** A particular sequential DCT-based encoding and decoding process specified in this Specification, and which is required for all DCT-based decoding processes.

9- **binary decision:** Choice between two alternatives.

10- **bit stream:** Partially encoded or decoded sequence of bits comprising an entropy-coded segment.

11- **block:** An 8 ´ 8 array of samples or an 8 ´ 8 array of DCT coefficient values of one component.

12- **block-row:** A sequence of eight contiguous component lines which are partitioned into 8 ´ 8 blocks.

13- **byte:** A group of 8 bits.

14- **byte stuffing:** A procedure in which either the Huffman coder or the arithmetic coder inserts a zero byte into the entropy-coded segment following the generation of an encoded hexadecimal X'FF' byte.

15- **carry bit:** A bit in the arithmetic encoder code register which is set if a carry-over in the code register overflows the eight bits reserved for the output byte.

16- **ceiling function:** The mathematical procedure in which the greatest integer value of a real number is obtained by selecting the smallest integer value which is greater than or equal to the real number.

17- **class (of coding process):** Lossy or lossless coding processes.

18- **code register:** The arithmetic encoder register containing the least significant bits of the partially completed entropy-coded segment. Alternatively, the arithmetic decoder register containing the most significant bits of a partially decoded entropy-coded segment.

19- **coder:** An embodiment of a coding process.

20- **coding:** Encoding or decoding.

21- **coding model:** A procedure used to convert input data into symbols to be coded.

22- **(coding) process:** A general term for referring to an encoding process, a decoding process, or both.

23- **colour image:** A continuous-tone image that has more than one component.

24- **columns:** Samples per line in a component.

25- **component:** One of the two-dimensional arrays which comprise an image.

26- **compressed data:** Either compressed image data or table specification data or both.

27- **compressed image data:** A coded representation of an image, as specified in this Specification.

28- **compression:** Reduction in the number of bits used to represent source image data.

29- **conditional exchange:** The interchange of MPS and LPS probability intervals whenever the size of the LPS interval is greater than the size of the MPS interval (in arithmetic coding).

30- **(conditional) probability estimate:** The probability value assigned to the LPS by the probability estimation state machine (in arithmetic coding).

31- **conditioning table:** The set of parameters which select one of the defined relationships between prior coding decisions and the conditional probability estimates used in arithmetic coding.

32- **context:** The set of previously coded binary decisions which is used to create the index to the probability estimation state machine (in arithmetic coding).

33- **continuous-tone image:** An image whose components have more than one bit per sample.

34- **data unit:** An 8 ´ 8 block of samples of one component in DCT-based processes; a sample in lossless processes.

35- **DC coefficient:** The DCT coefficient for which the frequency is zero in both dimensions.

36- **DC prediction:** The procedure used by DCT-based encoders whereby the quantized DC coefficient from the previously encoded 8 x 8 block of the same component is subtracted from the current quantized DC coefficient.

37- **(DCT) coefficient:** The amplitude of a specific cosine basis function – may refer to an original DCT coefficient, to a quantized DCT coefficient, or to a dequantized DCT coefficient.

38- **decoder:** An embodiment of a decoding process.

39- **decoding process:** A process which takes as its input compressed image data and outputs a continuous-tone image.

40- **default conditioning:** The values defined for the arithmetic coding conditioning tables at the beginning of coding of an image.

41- **dequantization:** The inverse procedure to quantization by which the decoder recovers a representation of the DCT coefficients.

42- **differential component:** The difference between an input component derived from the source image and the corresponding reference component derived from the preceding frame for that component (in hierarchical mode coding).

43- **differential frame:** A frame in a hierarchical process in which differential components are either encoded or decoded.

44- **(digital) reconstructed image (data):** A continuous-tone image which is the output of any decoder defined in this Specification.

45- **(digital) source image (data):** A continuous-tone image used as input to any encoder defined in this Specification.

46- **(digital) (still) image:** A set of two-dimensional arrays of integer data.

47- **discrete cosine transform; DCT:** Either the forward discrete cosine transform or the inverse discrete cosine transform.

48- **downsampling (filter):** A procedure by which the spatial resolution of an image is reduced (in hierarchical mode coding).

49- **encoder:** An embodiment of an encoding process.

50- **encoding process:** A process which takes as its input a continuous-tone image and outputs compressed image data.

51- **entropy-coded (data) segment:** An independently decodable sequence of entropy encoded bytes of compressed image data.

52- **(entropy-coded segment) pointer:** The variable which points to the most recently placed (or fetched) byte in the entropy encoded segment.

53- **entropy decoder:** An embodiment of an entropy decoding procedure.

54- **entropy decoding:** A lossless procedure which recovers the sequence of symbols from the sequence of bits produced by the entropy encoder.

55- **entropy encoder:** An embodiment of an entropy encoding procedure.

56- **entropy encoding:** A lossless procedure which converts a sequence of input symbols into a sequence of bits such that the average number of bits per symbol approaches the entropy of the input symbols.

57- **extended (DCT-based) process:** A descriptive term for DCT-based encoding and decoding processes in which additional capabilities are added to the baseline sequential process.

58- **forward discrete cosine transform; FDCT:** A mathematical transformation using cosine basis functions which converts a block of samples into a corresponding block of original DCT coefficients.

59- **frame:** A group of one or more scans (all using the same DCT-based or lossless process) through the data of one or more of the components in an image.

60- **frame header:** A marker segment that contains a start-of-frame marker and associated frame parameters that are coded at the beginning of a frame.

61- **frequency:** A two-dimensional index into the two-dimensional array of DCT coefficients.

62- **(frequency) band:** A contiguous group of coefficients from the zig-zag sequence (in progressive mode coding).

63- **full progression:** A process which uses both spectral selection and successive approximation (in progressive mode coding).

64- **grayscale image:** A continuous-tone image that has only one component.

65- **hierarchical:** A mode of operation for coding an image in which the first frame for a given component is followed by frames which code the differences between the source data and the reconstructed data from the previous frame for that component. Resolution changes are allowed between frames.

66- **hierarchical decoder:** A sequence of decoder processes in which the first frame for each component is followed by frames which decode an array of differences for each component and adds it to the reconstructed data from the preceding frame for that component.

67- **hierarchical encoder:** The mode of operation in which the first frame for each component is followed by frames which encode the array of differences between the source data and the reconstructed data from the preceding frame for that component.

68- **horizontal sampling factor:** The relative number of horizontal data units of a particular component with respect to the number of horizontal data units in the other components.

69- **Huffman decoder:** An embodiment of a Huffman decoding procedure.

70- **Huffman decoding:** An entropy decoding procedure which recovers the symbol from each variable length code produced by the Huffman encoder.

71- **Huffman encoder:** An embodiment of a Huffman encoding procedure.

72- **Huffman encoding:** An entropy encoding procedure which assigns a variable length code to each input symbol.

73- **Huffman table:** The set of variable length codes required in a Huffman encoder and Huffman decoder.

74- **image data:** Either source image data or reconstructed image data.

75- **interchange format:** The representation of compressed image data for exchange between application environments.

76- **interleaved:** The descriptive term applied to the repetitive multiplexing of small groups of data units from each component in a scan in a specific order.

77- **inverse discrete cosine transform; IDCT:** A mathematical transformation using cosine basis functions which converts a block of dequantized DCT coefficients into a corresponding block of samples.

78- **Joint Photographic Experts Group; JPEG:** The informal name of the committee which created this Specification. The “joint” comes from the CCITT and ISO/IEC collaboration.

79- **latent output:** Output of the arithmetic encoder which is held, pending resolution of carry-over (in arithmetic coding).

80- **less probable symbol; LPS:** For a binary decision, the decision value which has the smaller probability.

81- **level shift:** A procedure used by DCT-based encoders and decoders whereby each input sample is either converted from an unsigned representation to a two's complement representation or from a two's complement representation to an unsigned representation.

82- **lossless:** A descriptive term for encoding and decoding processes and procedures in which the output of the decoding procedure(s) is identical to the input to the encoding procedure(s).

83- **lossless coding:** The mode of operation which refers to any one of the coding processes defined in this Specification in which all of the procedures are lossless (see Annex H).

84- **lossy:** A descriptive term for encoding and decoding processes which are not lossless.

85- **marker:** A two-byte code in which the first byte is hexadecimal FF (X'FF') and the second byte is a value between 1 and hexadecimal FE (X'FE').

86- **marker segment:** A marker and associated set of parameters.

87- **MCU-row:** The smallest sequence of MCU which contains at least one line of samples or one block-row from every component in the scan.

88- **minimum coded unit; MCU:** The smallest group of data units that is coded.

89- **modes (of operation):** The four main categories of image coding processes defined in this Specification.

90- **more probable symbol; MPS:** For a binary decision, the decision value which has the larger probability.

91- **non-differential frame:** The first frame for any components in a hierarchical encoder or decoder. The components are encoded or decoded without subtraction from reference components. The term refers also to any frame in modes other than the hierarchical mode.

92- **non-interleaved:** The descriptive term applied to the data unit processing sequence when the scan has only one component.

93- **parameters:** Fixed length integers 4, 8 or 16 bits in length, used in the compressed data formats.

94- **point transform:** Scaling of a sample or DCT coefficient.

95- **precision:** Number of bits allocated to a particular sample or DCT coefficient.

96- **predictor:** A linear combination of previously reconstructed values (in lossless mode coding).

97- **probability estimation state machine:** An interlinked table of probability values and indices which is used to estimate the probability of the LPS (in arithmetic coding).

98- **probability interval:** The probability of a particular sequence of binary decisions within the ordered set of all possible sequences (in arithmetic coding).

99- **(probability) sub-interval:** A portion of a probability interval allocated to either of the two possible binary decision values (in arithmetic coding).

100- **procedure:** A set of steps which accomplishes one of the tasks which comprise an encoding or decoding process.

101- **process:** See coding process.

102- **progressive (coding):** One of the DCT-based processes defined in this Specification in which each scan typically improves the quality of the reconstructed image.

103- **progressive DCT-based:** The mode of operation which refers to any one of the processes defined in Annex G.

104- **quantization table:** The set of 64 quantization values used to quantize the DCT coefficients.

105- **quantization value:** An integer value used in the quantization procedure.

106- **quantize:** The act of performing the quantization procedure for a DCT coefficient.

107- **reference (reconstructed) component:** Reconstructed component data which is used in a subsequent frame of a hierarchical encoder or decoder process (in hierarchical mode coding).

108- **renormalization:** The doubling of the probability interval and the code register value until the probability interval exceeds a fixed minimum value (in arithmetic coding).

109- **restart interval:** The integer number of MCUs processed as an independent sequence within a scan.

110- **restart marker:** The marker that separates two restart intervals in a scan.

111- **run (length):** Number of consecutive symbols of the same value.

112- **sample:** One element in the two-dimensional array which comprises a component.

113- **sample-interleaved:** The descriptive term applied to the repetitive multiplexing of small groups of samples from each component in a scan in a specific order.

114- **scan:** A single pass through the data for one or more of the components in an image.

115- scan header: A marker segment that contains a start-of-scan marker and associated scan parameters that are coded at the beginning of a scan.

116- sequential (coding): One of the lossless or DCT-based coding processes defined in this Specification in which each component of the image is encoded within a single scan.

117- sequential DCT-based: The mode of operation which refers to any one of the processes defined in Annex F.

118- spectral selection: A progressive coding process in which the zig-zag sequence is divided into bands of one or more contiguous coefficients, and each band is coded in one scan.

119- stack counter: The count of X'FF' bytes which are held, pending resolution of carry-over in the arithmetic encoder.

120- statistical conditioning: The selection, based on prior coding decisions, of one estimate out of a set of conditional probability estimates (in arithmetic coding).

121- statistical model: The assignment of a particular conditional probability estimate to each of the binary arithmetic coding decisions.

122- statistics area: The array of statistics bins required for a coding process which uses arithmetic coding.

123- statistics bin: The storage location where an index is stored which identifies the value of the conditional probability estimate used for a particular arithmetic coding binary decision.

124- successive approximation: A progressive coding process in which the coefficients are coded with reduced precision in the first scan, and precision is increased by one bit with each succeeding scan.

125- table specification data: The coded representation from which the tables used in the encoder and decoder are generated and their destinations specified.

126- transcoder: A procedure for converting compressed image data of one encoder process to compressed image data of another encoder process.

127- (uniform) quantization: The procedure by which DCT coefficients are linearly scaled in order to achieve compression.

128- upsampling (filter): A procedure by which the spatial resolution of an image is increased (in hierarchical mode coding).

129- **vertical sampling factor:** The relative number of vertical data units of a particular component with respect to the number of vertical data units in the other components in the frame.

130- **zero byte:** The X’00’ byte.

131- **zig-zag sequence:** A specific sequential ordering of the DCT coefficients from (approximately) lowest spatial frequency to highest.

REFERENCES

[1] Joint Photographic Experts Group <http://www.jpeg.org/index.html>

[2] <http://www.cs.cf.ac.uk/Dave/Multimedia/node234.html>

[3] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins "Digital Image Processing using MATLAB," Pearson Prentice-Hall Education, Inc. 2004

[4] David S. Taubman and Michael W. Marcellin "JPEG 2000: Image Compression Fundamentals, Standards and Practice," The Kluwer International Series in Engineering and Computer Science, 2004

[5] <http://www.stanford.edu/class/ee368b/Projects/mkalman/report.html>

[6] "Computer Desktop Encyclopedia," The Computer Language Co. Inc., 1998.

[7] <http://en.wikipedia.org/wiki/JPEG#Decoding>

[8] HP Labs LOCO-I / JPEG-LS <http://www.hpl.hp.com/loco/>

[9] P. Schelkens, A. Skodras and T. Ebrahimi "The JPEG 2000 Suite," Wiley Series: Wiley-IS&T Series in Imaging Science and Technology, 2009

[10] Rafael C. Gonzalez, Richard E. Woods "Digital Image Processing, second Edition," Pearson Prentice-Hall Education, Inc. 2002

[11] International Organization for Standardization <www.iso.org>

[12] International Telecommunication Union <www.itu.int>

[13] Subject of "Aplicaciones en el Tratamiento de Señales" Universidad Carlos III, Leganés (Spain) 2010

ANNEX1 – INDEX OF FIGURES

ANNEX2 – CODE OF JPEG PROGRAM

This is the program code done with Matlab:

```
% JPEG COMPRESS PROGRAM
%
%    This program has two parts. The main part of this program is to
%    convert any format image into a ".jpg" image, with the standard JPEG
%    encoding.
%    The operation of this program is detailed in the project.
%
%    The other part shows the DCT blocks effects when compresses a black
%    & white image (only one component) with a rudimentary JPEG method.
%
%
%    Sergio Tallante Pastor
%    Wolfenbüttel, Germany 2010-2011
%
% Resset and clear all previous variables
clc
clear all
close all

% Quantization matrix for Brightness component
qy=[...
    3   2   2   3   5   8   10  12
    2   2   3   4   5   12  12  11
    3   3   3   5   8   11  14  11
    3   3   4   6   10  17  16  12
    4   4   7   11  14  22  21  15
    5   7   11  13  16  21  23  18
    10  13  16  17  21  24  24  20
    14  18  19  20  22  20  21  20];

 % Quantization matrix for Chrominance components
 qc=[...
    3   4   5   9   20  20  20  20
    4   4   5   13  20  20  20  20
    5   5   11  20  20  20  20  20
    9   13  20  20  20  20  20  20
    20  20  20  20  20  20  20  20
    20  20  20  20  20  20  20  20
    20  20  20  20  20  20  20  20
    20  20  20  20  20  20  20  20];

% Entropy coding: we use 'row' and 'col' to do the zig-zag ordering
row= [1 1 2 3 2 1 1 2 3 4 5 4 3 2 1 1 2 3 4 5 6 7 6 5 4 3 2 1 1 2 3 4 5 6
7 8 8 7 6 5 4 3 2 3 4 5 6 7 8 8 7 6 5 4 5 6 7 8 8 7 6 7 8 8];
col= [1 2 1 1 2 3 4 3 2 1 1 2 3 4 5 6 5 4 3 2 1 1 2 3 4 5 6 7 8 7 6 5 4 3
2 1 2 3 4 5 6 7 8 8 7 6 5 4 3 4 5 6 7 8 8 7 6 5 6 7 8 8 7 8];

% "Table" for DC coefficient
table_dc=[...
     3 0 1 0 0 0 0 0 0 0
     3 0 1 1 0 0 0 0 0 0
```

```
     3 1 0 0 0 0 0 0 0 0
     3 1 0 1 0 0 0 0 0 0
     3 1 1 0 0 0 0 0 0 0
     4 1 1 1 0 0 0 0 0 0
     5 1 1 1 1 0 0 0 0 0
     6 1 1 1 1 1 0 0 0 0
     7 1 1 1 1 1 1 0 0 0
     8 1 1 1 1 1 1 1 0 0
     9 1 1 1 1 1 1 1 1 0];

% run - no of bits required to encode data- total length - base code
% length(imp) -  base code use
% for both luminance and chrominance as in hex file...
table_ac=[...
 0  1   3  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0  2   4  2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0  3   6  3 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0  4   8  4 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0
 0  5  10  5 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0
 0  6  13  7 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
 0  7  15  8 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
 0  8  18 10 1 1 1 1 1 1 0 1 1 0 0 0 0 0 0
 0  9  25 16 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 0
 0 10  26 16 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1
 1  1   5  4 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1  2   7  5 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
 1  3  10  7 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0
 1  4  13  9 1 1 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0
 1  5  16 11 1 1 1 1 1 1 1 1 0 1 1 0 0 0 0 0 0
 1  6  22 16 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0 0
 1  7  23 16 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0 1
 1  8  24 16 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0
 1  9  25 16 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1
 1 10  26 16 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0
 2  1   6  5 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 2  2  10  8 1 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0
 2  3  13 10 1 1 1 1 1 1 1 0 1 1 1 0 0 0 0 0 0
 2  4  16 12 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0
 2  5  21 16 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 1
 2  6  22 16 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 1 0
 2  7  23 16 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 1 1
 2  8  24 16 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0
 2  9  25 16 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 1
 2 10  26 16 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 0
 3  1   7  6 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 3  2  11  9 1 1 1 1 1 1 0 1 1 1 0 0 0 0 0 0 0
 3  3  15 12 1 1 1 1 1 1 1 1 1 0 1 0 1 0 0 0 0
 3  4  20 16 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1
 3  5  21 16 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0 0
 3  6  22 16 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0 1
 3  7  23 16 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0
 3  8  24 16 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1
 3  9  25 16 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 0
 3 10  26 16 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 1
 4  1   7  6 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0
 4  2  12 10 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
 4  3  19 16 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1 0
 4  4  20 16 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1 1
 4  5  21 16 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 0 0
 4  6  22 16 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 0 1
 4  7  23 16 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 0
```

```
 4  8  24 16 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 1
 4  9  25 16 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 0
 4 10  26 16 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1
 5  1   8  7 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0
 5  2  13 11 1 1 1 1 1 1 1 0 1 1 1 0 0 0 0 0 0
 5  3  19 16 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0
 5  4  20 16 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1
 5  5  21 16 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0
 5  6  22 16 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 1
 5  7  23 16 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 0
 5  8  24 16 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 1
 5  9  25 16 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 0
 5 10  26 16 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1
 6  1   8  7 1 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0
 6  2  14 12 1 1 1 1 1 1 1 1 1 0 1 1 0 0 0 0 0
 6  3  19 16 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 0
 6  4  20 16 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1
 6  5  21 16 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 0 0
 6  6  22 16 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 0 1
 6  7  23 16 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0
 6  8  24 16 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 1
 6  9  25 16 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 0 0
 6 10  26 16 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 0 1
 7  1   9  8 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0
 7  2  14 12 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0 0
 7  3  19 16 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 0
 7  4  20 16 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1
 7  5  21 16 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 0 0
 7  6  22 16 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 0 1
 7  7  23 16 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 0
 7  8  24 16 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1
 7  9  25 16 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 0
 7 10  26 16 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1
 8  1  10  9 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
 8  2  17 15 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
 8  3  19 16 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0
 8  4  20 16 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1
 8  5  21 16 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0
 8  6  22 16 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1
 8  7  23 16 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0
 8  8  24 16 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1
 8  9  25 16 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0
 8 10  26 16 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1
 9  1  10  9 1 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 0
 9  2  18 16 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0
 9  3  19 16 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
 9  4  20 16 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
 9  5  21 16 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1
 9  6  22 16 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0
 9  7  23 16 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1
 9  8  24 16 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0
 9  9  25 16 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 1
 9 10  26 16 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0
10  1  10  9 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0
10  2  18 16 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1
10  3  19 16 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0
10  4  20 16 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1
10  5  21 16 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0
10  6  22 16 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1
10  7  23 16 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 0
10  8  24 16 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1
```

```
10  9  25 16 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0
10 10  26 16 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1
11  1  11 10 1 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0
11  2  18 16 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0
11  3  19 16 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1
11  4  20 16 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0
11  5  21 16 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1
11  6  22 16 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 0
11  7  23 16 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1
11  8  24 16 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 0
11  9  25 16 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1
11 10  26 16 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 0
12  1  11 10 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0
12  2  18 16 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1
12  3  19 16 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0
12  4  20 16 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1
12  5  21 16 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0
12  6  22 16 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1
12  7  23 16 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0
12  8  24 16 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
12  9  25 16 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
12 10  26 16 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1
13  1  12 11 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
13  2  18 16 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0
13  3  19 16 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1
13  4  20 16 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
13  5  21 16 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1
13  6  22 16 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0
13  7  23 16 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1
13  8  24 16 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0
13  9  25 16 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1
13 10  26 16 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0
14  1  17 16 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1
14  2  18 16 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0
14  3  19 16 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1
14  4  20 16 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0
14  5  21 16 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
14  6  22 16 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
14  7  23 16 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1
14  8  24 16 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0
14  9  25 16 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1
14 10  26 16 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0
15  1  17 16 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1
15  2  18 16 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0
15  3  19 16 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1
15  4  20 16 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
15  5  21 16 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1
15  6  22 16 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0
15  7  23 16 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
15  8  24 16 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
15  9  25 16 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
15 10  26 16 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0];

correct = 0;
component=3;
% Load data
while correct==0
   disp('Enter ...')
   disp('0 - Load an image to compress to .jpg (default)')
   disp('1 - Load a black & white image to change its quality ')
   chos=input('Enter your choice: ');
```

```
   disp(' ')
   if isempty(chos) % Default choice
      chos=0;
      correct=1;
   elseif ~(chos == 1)&&~(chos==0)
      display('Invalid character, use 0 or 1.')
   else
       correct=1;
   end
end
correct=0;

%*********************************************************************%
% If is a Black & White image...
if chos==1
   while(correct==0)
      image21=input('Enter Black & White Image to compress:');
      image21=num2str(image21);
      if ~isempty(image21)
         fid = fopen(image21, 'r');
         if (fid == -1)
            if ~isempty(dir(image21))
               display(['--> Can''t open file "',image21,'" for reading;
                        you may not have read permission.']);
               display('--> Try with another image.')
               disp(' ')
            else
               display(['--> File "',image21,'" does not exist.']);
               display('--> Try putting the name and the format
                        extension.')
               disp(' ')
            end
         else
            % File exists. Get full filename
            image21 = fopen(fid);
            fclose(fid);
            img=imread(image21);
            if size(img,3) ~= 1
               display('--> The image entered is not a valid "Black &
                        White" image.')
               display('--> The image must be of only one component.')
               disp(' ')
            else
               correct=1;
            end
         end
      else
         display('--> Please, enter the name of "Black & White" image.')
      end
   end  % End of `Starter while´

   % Enter the correct compress factor
   while correct==1
      compress = input('Enter the compress factor(%): ');
      if isempty(compress)
         disp('--> You must enter a number between 0 to 100.')
      elseif (compress<100) && (compress>=0)
         correct=0;
      elseif compress==100
         disp('--> Is impossible compress 100%.')
      else
```

```
      disp('--> The number must be between 0 to 100.')
   end
end  % End of `compress while´

display('Processing Black & White Image...')
dataImg=imfinfo(image21);
h=ceil((dataImg.Height)/8)-1; % Height
w=ceil((dataImg.Width)/8)-1; % Width

img1=double(img(1:dataImg.Height,1:dataImg.Width));

if compress==0
   % The final image it's the same of the original image
   imgresglobal = img1;
else
   % Quantization matrix (8x8 block)
   Q=qy.*compress;
   Fglobal=sparse(zeros(size(img1)));
   % Preparing for the calculation of the DCT
   i=1:8;u=i;
   [U,I]=meshgrid(u,i);
   B=sqrt(2/8)*cos(pi*(U-1).*(2*I-1)/(2*8));
   B(:,1)=B(:,1)/sqrt(2);

   % Calculation of the quantized DCT coeficients
   for x=0:h
      for y=0:w
         simg=img1((x*8)+1:x*8+8,(y*8)+1:y*8+8);
         F=B'*simg*B;
         % Compression
         F=sparse(round(F./Q));
         Fglobal((x*8)+1:x*8+8,(y*8)+1:y*8+8)=F;
      end;
   end;
   % Calculation of the compression image
   imgresglobal=zeros(size(img1));
   for x=0:h
      for y=0:w
         F=Fglobal((x*8)+1:x*8+8,(y*8)+1:y*8+8);
         imgres=B*(F.*Q)*B';
         imgres=uint8(imgres);
         imgresglobal((x*8)+1:x*8+8,(y*8)+1:y*8+8)=imgres;
      end;
   end;
end
set(gcf,'units','normalized','menubar','none','position',[0 0 1 0.978])
set(gcf,'name',dataImg.Filename,'numberTitle','off','color','w')
subplot 131 % Original image
img1=double(img(1:dataImg.Height,1:dataImg.Width));
imshow(img1,[min(img1(:)),max(img1(:))])
title('Original image','fontsize',11,'fontweight','bold')
set(gca,'xcolor','k','ycolor','k')
subplot 132 % Image after apply the DCT
DCT=@dct2;
img_DCT=blkproc(img1,[8 8],DCT);
imshow(log10(1+abs(img_DCT)),[])
title('Image after apply the DCT','fontsize',11,'fontweight','bold')
subplot 133 % Compressed image
imshow(imgresglobal,[min(imgresglobal(:)),max(imgresglobal(:))])
title('Compressed image','fontsize',11,'fontweight','bold')
set(gca,'xcolor','k','ycolor','k')
```

```
%*************************************************************************%

% Or if you want to compress and encoding the image to a JPEG format...
else
while(correct==0)
   image21=input('Enter an Image to compress:');
   image21=num2str(image21);
   if ~isempty(image21)
      fid = fopen(image21, 'r');
      if (fid == -1)
         if ~isempty(dir(image21))
            display(['--> Can''t open file "',image21,'" for reading; you
                     may not have read permission.']);
            display('--> Try with another image.')
            disp(' ')
         else
            display(['--> File "',image21,'" does not exist.']);
            display('--> Try putting the name and the format extension.')
            disp(' ')
         end
      else
         % File exists. Get full filename
         fclose(fid);
         img=imread(image21);
         if size(img,3) ~= 3
            display('--> The image entered is not valid.')
            disp(' ')
         else
            correct=1;
            name_comp=input('Enter name of final image (without extension):');
            if isempty(name_comp),
               name_comp='save_default';
            else
               name_comp=num2str(name_comp);
            end
         end
      end
   else
      display('--> Please, enter the name of color image.')
   end
end % End of `Starter while.´

display('Processing Image...')
dataImg=imfinfo(image21);
h=ceil((dataImg.Height)/8); % Height
w=ceil((dataImg.Width)/8); % Width

ycbcr_map=zeros(h*8,w*8,3);
% ycbcr_map(1:dataImg.Height,1:dataImg.Width,1:3) = rgb2ycbcr(img); % Is
% aprox. the same, but it has less quality than colorspace
ycbcr_map(1:dataImg.Height,1:dataImg.Width,1:3) = colorspace('jpegycbcr<-
RGB',img);

fid1 = fopen([name_comp,'.jpg'],'w');
fid2 = fopen('header','r');
hread=fread(fid2);
fwrite(fid1,hread);
fclose(fid2);
buf=[];
buf_str='';
ac_code=[];
```

```
for i=1:h % For cover the rows
   for j=1:w % For cover the columns
      temp=zeros(8,8); % 8x8 block
      for y=1:3 % For the 3 components (Y',Cb,Cr)
         for l=1:8 % Rows
            for k=1:8 % Columns
               temp(l,k)=ycbcr_map(((i-1)*8)+l,((j-1)*8)+k,y);
            end
         end
         % Center the range [0,255] arround zero: [-128,127]
         temp=temp-128;
         % 2-D discrete cosine transform of the 8x8 block
         temp=dct2(temp);
         % Quantization
         if(y==1)
            % Brightness component (Y')
            temp=temp./qy;
         else
            % Chrominance components (Cb,Cr)
            temp=temp./qc;
         end

         temp=round(temp);
         % Entropy coding: zig-zag ordering
         for ct=1:64
            str(ct)=temp(row(ct),col(ct));
         end
        % DC entropy coding
         if j==1 & i==1
            % The top-left corner entry is the rather large magnitude.
            % This is the DC coefficient
            dc=temp(1);
            dc_l(y)=temp(1);
         else
            dc=temp(1)-dc_l(y);
            dc_l(y)=temp(1);
         end

         if dc==0
            dc_code=[0 0];
         else
            dc_abs=abs(dc);
            % Convert decimal number into binary
            dc_bin=dec2bin(dc_abs);
            dc_len=length(dc_bin);
            % Create a vector with replicates
            if dc<0
               comp=repmat(49,[1 dc_len]);
               dc_bin=comp-dc_bin;
            else
               comp=repmat(48,[1 dc_len]);
               dc_bin=dc_bin-comp;
            end
            % Generate the Huffman sequence from the table_dc
            dc_code=table_dc(dc_len,2:1+table_dc(dc_len,1));
            dc_code=[dc_code dc_bin];
         end

         % AC entropy coding
         str=str(2:64); % Rest of entropy coding
```

```
noz_ac=find(str~=0); % Look for non zeros
ac_no=length(noz_ac);

for o=1:ac_no
   ac=str(noz_ac(o));
   ac_abs=abs(ac);
   % Convert decimal number into binary
   ac_bin=dec2bin(ac_abs);
   ac_len=length(ac_bin);
   % Create a vector with replicates
   if ac<0
      comp=repmat(49,[1 ac_len]);
      ac_bin=comp-ac_bin;
   else
      comp=repmat(48,[1 ac_len]);
      ac_bin=ac_bin-comp;
   end

   if(noz_ac(o)==1) % First position of str is not a zero
      % Search the corresponding number of table_ac
      ac_seq=table_ac(ac_len,5:4+table_ac(ac_len,4));
      ac_code=[ac_code ac_seq ac_bin];

   else % First position of str is a zero
      if o==1
         z=noz_ac(o)-1;
      else
         z=noz_ac(o)-noz_ac(o-1)-1;
      end

      % Check 4 if z>=15
      noz=floor(z/16);
      zrl=[1 1 1 1 1 1 1 1 0 0 1];
      ac_code=[ac_code repmat(zrl,[1,noz])];
      z=rem(z,16);
      % Generate the Huffman sequence from the table_ac
      ac_seq=table_ac((z*10)+ac_len,5:4+table_ac
                      ((z*10)+ac_len,4));
      ac_code=[ac_code ac_seq ac_bin];
   end
end

if (str(63)==0),
   ac_code=[ac_code 1 0 1 0];
end

ac_code=[dc_code ac_code];
% The last position of 8x8 blocks
if(i==h)
   if(j==w)
      ac_code=[ac_code 0 0 1 0 1 0];
   end
end

% Write the compressed image into file
q=1;
while(q<=length(ac_code))
   if length(buf)~=8
      buf=[buf ac_code(q)];
      q=q+1;
```

```
            else
               % When the buffer is full (8 bits), convert binary to
               % decimal number, write into fid1, and reset buf & buf_str
               for e=1:8
                  if buf(e)==0
                     buf_str=[buf_str '0'];
                  else
                     buf_str=[buf_str '1'];
                  end
               end
               kk=bin2dec(buf_str);
               fwrite(fid1,kk,'uint8');
               if kk==255
                  fwrite(fid1,0,'uint8');
               end
               buf=[];
               buf_str=[];
            end
         end
         % Reset for others components
         temp=[];
         ac_code=[];
         dc_code=[];
      end
   end
end  % End of [for i=1:h]

% When the buffer has more bits
if ~isempty(buf)
   buf=[buf zeros(1,8-length(buf))];
   for e=1:8
      if buf(e)==0
         buf_str=[buf_str '0'];
      else
         buf_str=[buf_str '1'];
      end
   end
   kk=bin2dec(buf_str);
   fwrite(fid1,kk,'uint8');
   if kk==255
      fwrite(fid1,0,'uint8');
   end
end

% Corresponding to ff
fwrite(fid1,255,'uint8');
% Corresponding to d9
fwrite(fid1,217,'uint8');

fseek(fid1,163,-1);
if(dataImg.Height<=255)
   fwrite(fid1,0,'uint8');
   fwrite(fid1,dataImg.Height,'uint8');
else
   fwrite(fid1,floor(dataImg.Height/256),'uint8');
   fwrite(fid1,rem(dataImg.Height,256),'uint8');
end
fseek(fid1,165,-1);
% Specify height at an offset of a4
if(dataImg.Width<=255)
   fwrite(fid1,0,'uint8');
```

```
   fwrite(fid1,dataImg.Width,'uint8');
else
   fwrite(fid1,floor(dataImg.Width/256),'uint8');
   fwrite(fid1,rem(dataImg.Width,256),'uint8');
end
% End of the function

% Print information about images
display('Info about Original Image:')
imfinfo(image21)
display('Info about Compressed Image:')
dataImgComp=imfinfo([name_comp,'.jpg']);

% Print both images
display(['Compress factor: ',num2str((1-
(dataImgComp.FileSize/dataImg.FileSize))*100),' %'])
warning off all
set(gcf,'units','normalized','menubar','none','position',[0 0 1 0.978])
set(gcf,'name',dataImg.Filename,'numberTitle','off','color','w')
subplot 121 % Original image
imshow(img,[min(img(:)),max(img(:))])
title('Original image','fontsize',11,'fontweight','bold')
set(gca,'xcolor','k','ycolor','k')
xlabel(['File size: ',num2str(dataImg.FileSize),'
bits'],'fontsize',10,'fontweight','bold')
subplot 122 % Compressed image
img_comp=imread([name_comp,'.jpg']);
imshow(img_comp,[min(img_comp(:)),max(img_comp(:))])
title('Compressed image','fontsize',11,'fontweight','bold')
set(gca,'xcolor','k','ycolor','k')
xlabel(['File size: ',num2str(dataImgComp.FileSize),'
bits'],'fontsize',10,'fontweight','bold')
fclose(fid1);
end
```

And this is the function “colorspace” used to convert from RGB color image to $Y'C_BC_R$.

```
function varargout = colorspace(Conversion,varargin)
% COLORSPACE  Convert a color image between color representations.
%   B = COLORSPACE(S,A) converts the color representation of image A
%   where S is a string specifying the conversion.  S tells the
%   source and destination color spaces, S = 'dest<-src', or
%   alternatively, S = 'src->dest'.  Supported color spaces are
%
%     'RGB'              R'G'B' Red Green Blue (ITU-R BT.709 gamma-
%                         corrected)
%     'YPbPr'            Luma (ITU-R BT.601) + Chroma
%     'YCbCr'/'YCC'      Luma + Chroma ("digitized" version of Y'PbPr)
%     'YUV'              NTSC PAL Y'UV Luma + Chroma
%     'YIQ'              NTSC Y'IQ Luma + Chroma
%     'YDbDr'            SECAM Y'DbDr Luma + Chroma
%     'JPEGYCbCr'        JPEG-Y'CbCr Luma + Chroma
%     'HSV'/'HSB'        Hue Saturation Value/Brightness
%     'HSL'/'HLS'/'HSI'  Hue Saturation Luminance/Intensity
%     'XYZ'              CIE XYZ
%     'Lab'              CIE L*a*b* (CIELAB)
%     'Luv'              CIE L*u*v* (CIELUV)
```

```
%     'Lch'             CIE L*ch (CIELCH)
%
%  All conversions assume 2 degree observer and D65 illuminant.  Color
%  space names are case insensitive.  When R'G'B' is the source or
%  destination, it can be omitted. For example 'yuv<-' is short for
%  'yuv<-rgb'.
%
%  MATLAB uses two standard data formats for R'G'B': double data with
%  intensities in the range 0 to 1, and uint8 data with integer-valued
%  intensities from 0 to 255.  As MATLAB's native datatype, double data is
%  the natural choice, and the R'G'B' format used by colorspace.  However,
%  for memory and computational performance, some functions also operate
%  with uint8 R'G'B'.  Given uint8 R'G'B' color data, colorspace will
%  first cast it to double R'G'B' before processing.
%
%  If A is an Mx3 array, like a colormap, B will also have size Mx3.
%
%  [B1,B2,B3] = COLORSPACE(S,A) specifies separate output channels.
%  COLORSPACE(S,A1,A2,A3) specifies separate input channels.
%
%   Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins
%   Digital Image Processing Using MATLAB, Prentice-Hall, 2004
%


%%% Input parsing %%%
if nargin < 2, error('Not enough input arguments.'); end
[SrcSpace,DestSpace] = parse(Conversion);
if nargin == 2
   Image = varargin{1};
elseif nargin >= 3
   Image = cat(3,varargin{:});
else
   error('Invalid number of input arguments.');
end
FlipDims = (size(Image,3) == 1);
if FlipDims, Image = permute(Image,[1,3,2]); end
if ~isa(Image,'double'), Image = double(Image)/255; end
if size(Image,3) ~= 3, error('Invalid input size.'); end
SrcT = gettransform(SrcSpace);
DestT = gettransform(DestSpace);
if ~ischar(SrcT) & ~ischar(DestT)
   % Both source and destination transforms are affine, so they
   % can be composed into one affine operation
   T = [DestT(:,1:3)*SrcT(:,1:3),DestT(:,1:3)*SrcT(:,4)+DestT(:,4)];
   Temp = zeros(size(Image));
   Temp(:,:,1) = T(1)*Image(:,:,1) + T(4)*Image(:,:,2) +
T(7)*Image(:,:,3) + T(10);
   Temp(:,:,2) = T(2)*Image(:,:,1) + T(5)*Image(:,:,2) +
T(8)*Image(:,:,3) + T(11);
   Temp(:,:,3) = T(3)*Image(:,:,1) + T(6)*Image(:,:,2) +
T(9)*Image(:,:,3) + T(12);
   Image = Temp;
elseif ~ischar(DestT)
   Image = rgb(Image,SrcSpace);
   Temp = zeros(size(Image));
   Temp(:,:,1) = DestT(1)*Image(:,:,1) + DestT(4)*Image(:,:,2) +
DestT(7)*Image(:,:,3) + DestT(10);
   Temp(:,:,2) = DestT(2)*Image(:,:,1) + DestT(5)*Image(:,:,2) +
DestT(8)*Image(:,:,3) + DestT(11);
```

```
   Temp(:,:,3) = DestT(3)*Image(:,:,1) + DestT(6)*Image(:,:,2) +
DestT(9)*Image(:,:,3) + DestT(12);
   Image = Temp;
else
   Image = feval(DestT,Image,SrcSpace);
end
%%% Output format %%%
if nargout > 1
   varargout = {Image(:,:,1),Image(:,:,2),Image(:,:,3)};
else
   if FlipDims, Image = permute(Image,[1,3,2]); end
   varargout = {Image};
end
return;

function [SrcSpace,DestSpace] = parse(Str)
% Parse conversion argument
if isstr(Str)
   Str = lower(strrep(strrep(Str,'-',''),' ',''));
   k = find(Str == '>');

   if length(k) == 1         % Interpret the form 'src->dest'
      SrcSpace = Str(1:k-1);
      DestSpace = Str(k+1:end);
   else
      k = find(Str == '<');

      if length(k) == 1      % Interpret the form 'dest<-src'
         DestSpace = Str(1:k-1);
         SrcSpace = Str(k+1:end);
      else
         error(['Invalid conversion, ''',Str,'''.']);
      end
   end
   SrcSpace = alias(SrcSpace);
   DestSpace = alias(DestSpace);
else
   SrcSpace = 1;             % No source pre-transform
   DestSpace = Conversion;
   if any(size(Conversion) ~= 3), error('Transformation matrix must be
3x3.'); end
end
return;

function Space = alias(Space)
Space = strrep(Space,'cie','');
if isempty(Space)
   Space = 'rgb';
end
switch Space
case {'ycbcr','ycc'}
   Space = 'ycbcr';
case {'hsv','hsb'}
   Space = 'hsv';
case {'hsl','hsi','hls'}
   Space = 'hsl';
case
{'rgb','yuv','yiq','ydbdr','ycbcr','jpegycbcr','xyz','lab','luv','lch'}
   return;
end
return;
```

```
function T = gettransform(Space)
% Get a colorspace transform: either a matrix describing an affine transform,
% or a string referring to a conversion subroutine
switch Space
case 'ypbpr'
   T = [0.299,0.587,0.114,0;-0.1687367,-0.331264,0.5,0;0.5,-0.418688,-
0.081312,0];
case 'yuv'
   % R'G'B' to NTSC/PAL YUV
   % Wikipedia: http://en.wikipedia.org/wiki/YUV
   T = [0.299,0.587,0.114,0;-0.147,-0.289,0.436,0;0.615,-0.515,-0.100,0];
case 'ydbdr'
   % R'G'B' to SECAM YDbDr
   % Wikipedia: http://en.wikipedia.org/wiki/YDbDr
   T = [0.299,0.587,0.114,0;-0.450,-0.883,1.333,0;-1.333,1.116,0.217,0];
case 'yiq'
   % R'G'B' in [0,1] to NTSC YIQ in [0,1];[-0.595716,0.595716];[-
0.522591,0.522591];
   % Wikipedia: http://en.wikipedia.org/wiki/YIQ
   T = [0.299,0.587,0.114,0;0.595716,-0.274453,-0.321263,0;0.211456,-
0.522591,0.311135,0];
case 'ycbcr'
   % R'G'B' (range [0,1]) to ITU-R BRT.601 (CCIR 601) Y'CbCr
   % Wikipedia: http://en.wikipedia.org/wiki/YCbCr
   % Poynton, Equation 3, scaling of R'G'B to Y'PbPr conversion
   T = [65.481,128.553,24.966,16;-37.797,-74.203,112.0,128;112.0,-
93.786,-18.214,128];
case 'jpegycbcr'
   % Wikipedia: http://en.wikipedia.org/wiki/YCbCr
   T = [0.299,0.587,0.114,0;-0.168736,-0.331264,0.5,0.5;0.5,-0.418688,-
0.081312,0.5]*255;
case {'rgb','xyz','hsv','hsl','lab','luv','lch'}
   T = Space;
otherwise
   error(['Unknown color space, ''',Space,'''.']);
end
return;

function Image = rgb(Image,SrcSpace)
% Convert to Rec. 709 R'G'B' from 'SrcSpace'
switch SrcSpace
case 'rgb'
   return;
case 'hsv'
   % Convert HSV to R'G'B'
   Image = huetorgb((1 -
Image(:,:,2)).*Image(:,:,3),Image(:,:,3),Image(:,:,1));
case 'hsl'
   % Convert HSL to R'G'B'
   L = Image(:,:,3);
   Delta = Image(:,:,2).*min(L,1-L);
   Image = huetorgb(L-Delta,L+Delta,Image(:,:,1));
case {'xyz','lab','luv','lch'}
   % Convert to CIE XYZ
   Image = xyz(Image,SrcSpace);
   % Convert XYZ to RGB
   T = [3.240479,-1.53715,-0.498535;-
0.969256,1.875992,0.041556;0.055648,-0.204043,1.057311];
   R = T(1)*Image(:,:,1) + T(4)*Image(:,:,2) + T(7)*Image(:,:,3);  % R
   G = T(2)*Image(:,:,1) + T(5)*Image(:,:,2) + T(8)*Image(:,:,3);  % G
   B = T(3)*Image(:,:,1) + T(6)*Image(:,:,2) + T(9)*Image(:,:,3);  % B
```

```
   % Desaturate and rescale to constrain resulting RGB values to [0,1]
   AddWhite = -min(min(min(R,G),B),0);
   Scale = max(max(max(R,G),B)+AddWhite,1);
   R = (R + AddWhite)./Scale;
   G = (G + AddWhite)./Scale;
   B = (B + AddWhite)./Scale;
   % Apply gamma correction to convert RGB to Rec. 709 R'G'B'
   Image(:,:,1) = gammacorrection(R);  % R'
   Image(:,:,2) = gammacorrection(G);  % G'
   Image(:,:,3) = gammacorrection(B);  % B'
otherwise  % Conversion is through an affine transform
   T = gettransform(SrcSpace);
   temp = inv(T(:,1:3));
   T = [temp,-temp*T(:,4)];
   R = T(1)*Image(:,:,1) + T(4)*Image(:,:,2) + T(7)*Image(:,:,3) + T(10);
   G = T(2)*Image(:,:,1) + T(5)*Image(:,:,2) + T(8)*Image(:,:,3) + T(11);
   B = T(3)*Image(:,:,1) + T(6)*Image(:,:,2) + T(9)*Image(:,:,3) + T(12);
   AddWhite = -min(min(min(R,G),B),0);
   Scale = max(max(max(R,G),B)+AddWhite,1);
   R = (R + AddWhite)./Scale;
   G = (G + AddWhite)./Scale;
   B = (B + AddWhite)./Scale;
   Image(:,:,1) = R;
   Image(:,:,2) = G;
   Image(:,:,3) = B;
end
% Clip to [0,1]
Image = min(max(Image,0),1);
return;

function Image = xyz(Image,SrcSpace)
% Convert to CIE XYZ from 'SrcSpace'
WhitePoint = [0.950456,1,1.088754];
switch SrcSpace
case 'xyz'
   return;
case 'luv'
   % Convert CIE L*uv to XYZ
   WhitePointU = (4*WhitePoint(1))./(WhitePoint(1) + 15*WhitePoint(2) +
3*WhitePoint(3));
   WhitePointV = (9*WhitePoint(2))./(WhitePoint(1) + 15*WhitePoint(2) +
3*WhitePoint(3));
   L = Image(:,:,1);
   Y = (L + 16)/116;
   Y = invf(Y)*WhitePoint(2);
   U = Image(:,:,2)./(13*L + 1e-6*(L==0)) + WhitePointU;
   V = Image(:,:,3)./(13*L + 1e-6*(L==0)) + WhitePointV;
   Image(:,:,1) = -(9*Y.*U)./((U-4).*V - U.*V);                    % X
   Image(:,:,2) = Y;                                               % Y
   Image(:,:,3) = (9*Y - (15*V.*Y) - (V.*Image(:,:,1)))./(3*V);  % Z
case {'lab','lch'}
   Image = lab(Image,SrcSpace);
   % Convert CIE L*ab to XYZ
   fY = (Image(:,:,1) + 16)/116;
   fX = fY + Image(:,:,2)/500;
   fZ = fY - Image(:,:,3)/200;
   Image(:,:,1) = WhitePoint(1)*invf(fX);  % X
   Image(:,:,2) = WhitePoint(2)*invf(fY);  % Y
   Image(:,:,3) = WhitePoint(3)*invf(fZ);  % Z
otherwise   % Convert from some gamma-corrected space
   % Convert to Rec. 701 R'G'B'
```

```
   Image = rgb(Image,SrcSpace);
   % Undo gamma correction
   R = invgammacorrection(Image(:,:,1));
   G = invgammacorrection(Image(:,:,2));
   B = invgammacorrection(Image(:,:,3));
   % Convert RGB to XYZ
   T = inv([3.240479,-1.53715,-0.498535;-
0.969256,1.875992,0.041556;0.055648,-0.204043,1.057311]);
   Image(:,:,1) = T(1)*R + T(4)*G + T(7)*B;  % X
   Image(:,:,2) = T(2)*R + T(5)*G + T(8)*B;  % Y
   Image(:,:,3) = T(3)*R + T(6)*G + T(9)*B;  % Z
end
return;

function Image = hsv(Image,SrcSpace)
% Convert to HSV
Image = rgb(Image,SrcSpace);
V = max(Image,[],3);
S = (V - min(Image,[],3))./(V + (V == 0));
Image(:,:,1) = rgbtohue(Image);
Image(:,:,2) = S;
Image(:,:,3) = V;
return;

function Image = hsl(Image,SrcSpace)
% Convert to HSL
switch SrcSpace
case 'hsv'
   % Convert HSV to HSL
   MaxVal = Image(:,:,3);
   MinVal = (1 - Image(:,:,2)).*MaxVal;
   L = 0.5*(MaxVal + MinVal);
   temp = min(L,1-L);
   Image(:,:,2) = 0.5*(MaxVal - MinVal)./(temp + (temp == 0));
   Image(:,:,3) = L;
otherwise
   Image = rgb(Image,SrcSpace);  % Convert to Rec. 701 R'G'B'
   % Convert R'G'B' to HSL
   MinVal = min(Image,[],3);
   MaxVal = max(Image,[],3);
   L = 0.5*(MaxVal + MinVal);
   temp = min(L,1-L);
   S = 0.5*(MaxVal - MinVal)./(temp + (temp == 0));
   Image(:,:,1) = rgbtohue(Image);
   Image(:,:,2) = S;
   Image(:,:,3) = L;
end
return;

function Image = lab(Image,SrcSpace)
% Convert to CIE L*a*b* (CIELAB)
WhitePoint = [0.950456,1,1.088754];
switch SrcSpace
case 'lab'
   return;
case 'lch'
   % Convert CIE L*CH to CIE L*ab
   C = Image(:,:,2);
   Image(:,:,2) = cos(Image(:,:,3)*pi/180).*C;  % a*
   Image(:,:,3) = sin(Image(:,:,3)*pi/180).*C;  % b*
otherwise
```

```
   Image = xyz(Image,SrcSpace);  % Convert to XYZ
   % Convert XYZ to CIE L*a*b*
   X = Image(:,:,1)/WhitePoint(1);
   Y = Image(:,:,2)/WhitePoint(2);
   Z = Image(:,:,3)/WhitePoint(3);
   fX = f(X);
   fY = f(Y);
   fZ = f(Z);
   Image(:,:,1) = 116*fY - 16;    % L*
   Image(:,:,2) = 500*(fX - fY);  % a*
   Image(:,:,3) = 200*(fY - fZ);  % b*
end
return;

function Image = luv(Image,SrcSpace)
% Convert to CIE L*u*v* (CIELUV)
WhitePoint = [0.950456,1,1.088754];
WhitePointU = (4*WhitePoint(1))./(WhitePoint(1) + 15*WhitePoint(2) +
3*WhitePoint(3));
WhitePointV = (9*WhitePoint(2))./(WhitePoint(1) + 15*WhitePoint(2) +
3*WhitePoint(3));
Image = xyz(Image,SrcSpace); % Convert to XYZ
U = (4*Image(:,:,1))./(Image(:,:,1) + 15*Image(:,:,2) + 3*Image(:,:,3));
V = (9*Image(:,:,2))./(Image(:,:,1) + 15*Image(:,:,2) + 3*Image(:,:,3));
Y = Image(:,:,2)/WhitePoint(2);
L = 116*f(Y) - 16;
Image(:,:,1) = L;                          % L*
Image(:,:,2) = 13*L.*(U - WhitePointU);  % u*
Image(:,:,3) = 13*L.*(V - WhitePointV);  % v*
return;

function Image = lch(Image,SrcSpace)
% Convert to CIE L*ch
Image = lab(Image,SrcSpace);  % Convert to CIE L*ab
H = atan2(Image(:,:,3),Image(:,:,2));
H = H*180/pi + 360*(H < 0);
Image(:,:,2) = sqrt(Image(:,:,2).^2 + Image(:,:,3).^2);  % C
Image(:,:,3) = H;                                        % H
return;

function Image = huetorgb(m0,m2,H)
% Convert HSV or HSL hue to RGB
N = size(H);
H = min(max(H(:),0),360)/60;
m0 = m0(:);
m2 = m2(:);
F = H - round(H/2)*2;
M = [m0, m0 + (m2-m0).*abs(F), m2];
Num = length(m0);
j = [2 1 0;1 2 0;0 2 1;0 1 2;1 0 2;2 0 1;2 1 0]*Num;
k = floor(H) + 1;
Image =
reshape([M(j(k,1)+(1:Num).'),M(j(k,2)+(1:Num).'),M(j(k,3)+(1:Num).')],[N,
3]);
return;

function H = rgbtohue(Image)
% Convert RGB to HSV or HSL hue
[M,i] = sort(Image,3);
i = i(:,:,3);
Delta = M(:,:,3) - M(:,:,1);
```

```
Delta = Delta + (Delta == 0);
R = Image(:,:,1);
G = Image(:,:,2);
B = Image(:,:,3);
H = zeros(size(R));
k = (i == 1);
H(k) = (G(k) - B(k))./Delta(k);
k = (i == 2);
H(k) = 2 + (B(k) - R(k))./Delta(k);
k = (i == 3);
H(k) = 4 + (R(k) - G(k))./Delta(k);
H = 60*H + 360*(H < 0);
H(Delta == 0) = nan;
return;

function Rp = gammacorrection(R)
Rp = real(1.099*R.^0.45 - 0.099);
i = (R < 0.018);
Rp(i) = 4.5138*R(i);
return;

function R = invgammacorrection(Rp)
R = real(((Rp + 0.099)/1.099).^(1/0.45));
i = (R < 0.018);
R(i) = Rp(i)/4.5138;
return;


function fY = f(Y)
fY = real(Y.^(1/3));
i = (Y < 0.008856);
fY(i) = Y(i)*(841/108) + (4/29);
return;

function Y = invf(fY)
Y = fY.^3;
i = (Y < 0.008856);
Y(i) = (fY(i) - 4/29)*(108/841);
return;
```