# Trabajo Práctico 2
# Compresión de Imágenes

Organización del Computador 2

2do. Cuatrimestre 2008

## 1. Introdución teórica

Los algoritmos usuales de compresión de datos e imágenes constan de tres etapas bien definidas: **transformación**, **cuantización** y **codificación**. La transformación de los datos es un procedimiento genérico que se utiliza para representar la información en una forma alternativa y en la que, en principio, resulta más evidente la redundancia existente en los datos originales. Es necesario que esta transformación sea inversible, es decir, que a partir de los datos transformados podamos recuperar exactamente la información original. La transformación de los datos puede ser de naturaleza muy distinta, o incluso, no resultar necesario si existen procedimientos eficientes para la compresión directa.

La compresión de datos puede ser con pérdidas (*lossy*) o sin pérdidas (*lossless*) en función de si la información que se recupera coincide exactamente con la original o es sólo una aproximación. Los métodos de codificación sin pérdidas se utilizan sobre todo en aplicaciones de codificación de datos binarios de aplicaciones informáticas en los que es absolutamente necesario recuperar la información original. Los formatos más populares son el `ZIP` y el `ARJ`. En procesamiento de imágenes los métodos de compresión sin pérdidas encuentran su aplicación en la codificación de imágenes médicas o científicas en las que puede resultar crítico la pérdida de parte de la información. Un ejemplo de formato de compresión de imágenes sin pérdida de información es el `PNG`.

La compresión con pérdidas es la más habitual en la codificación de señales de vídeo y audio. Evidentemente, las pérdidas son tolerables siempre que la calidad de las señales decodificadas sean aceptables. El principio general sobre el que se sustenta la codificación con pérdidas es que no resulta necesario codificar aquellas componentes de la información que no son observables por los sistemas de percepción humana. Por lo tanto, estos métodos se fundamentan en las características psicofisilógicas de los sistemas auditivo y visual, que son, en última instancia, los que deben evaluar la calidad del algoritmo de compresión. Por ello, es fundamental comprender las limitaciones y características de estos sistemas de percepción para diseñar codificadores en los que las pérdidas de información resulten poco evidentes o incluso inapreciables. La principal ventaja de estas estrategias de codificación es que consiguen unos factores de compresión muy superiores a los que se obtienen con los métodos sin pérdidas. La pérdida se produce en la etapa de cuantización. En ésta se reduce un intervalo de valores a un sólo valor que representa a todos los valores del intervalo.

Por último, la etapa de codificación consiste en establecer una correspondencia entre cada uno de los símbolos que componen los datos que deseamos comprimir y una secuencia de códigos. Como la correspondencia entre los símbolos y sus respectivos códigos es unívoca, esta

estapa resulta también sin pérdida de información. En el Trabajo Práctico 1 presentamos la codificación de Huffman.

## 2. Algoritmo de compresión

`JPEG` es un formato muy utilizado para comprimir imágenes fotográficas (donde hay una alta gama de colores). La compresión se logra a través de la eliminación de información redundante y/o irrelevante. Es un tipo de compresión con pérdida.

En el presente trabajo vamos a estudiar una simplificación de este formato. Para eso utilizaremos el siguiente algoritmo de compresión[1]:

1. Leer los datos de la imagen.
2. Separar los datos de la imagen en sus canales R, G y B.
3. Para cada canal:
    4. Dividir la imagen en bloques (submatrices) de $8 \times 8$.
        5. Para cada bloque:
            6. **Transformar** el bloque.
            7. **Cuantizar** el resultado de la transformación.
            8. **Codificar** el resultado de la cuantización.
            9. Guardar la codificación en un bitstream de salida.
10. Generar el archivo con la imagen comprimida a partir de los bitstreams de todos los bloques.

La etapa de **transformación** consiste en aplicar **DCT** (Tansformada Discreta del Coseno) a cada bloque de la imagen $B$ de $8 \times 8$. Para ello primero se debe contruir la siguiente matriz:

$$DCT_{(i,j)} = c(i)\ cos(\frac{(2\ j+1)\ i\ \pi}{16}) \qquad 0 \leq i, j \leq 7$$

donde

$$c(i) = \begin{cases} \sqrt{\frac{1}{8}} & \text{si } i = 0 \\ \sqrt{\frac{2}{8}} & \text{si } i \neq 0 \end{cases}$$

Luego, para transformar $B$ a través de la **DCT** se debe hacer el siguiente cálculo:

$$B_{dct} = DCT \cdot B^t \cdot DCT^t$$

Para recuperar el bloque original, es decir, realizar la inversa, se debe hacer lo siguiente:

$$B = (DCT^t \cdot B_{dct} \cdot DCT)^t$$

[1] El algortimo de descompresión es análogo, utilizando las funciones inversas

La etapa de **cuantización** consiste en dividir cada elemento de la matriz transformada ($B_{dct}$) de la siguiente manera:

$$B_{q_{(i,j)}} = redondear\Big(\frac{B_{dct_{(i,j)}}}{Q_{(i,j)}}\Big) \qquad 0 \leq i,j \leq 7$$

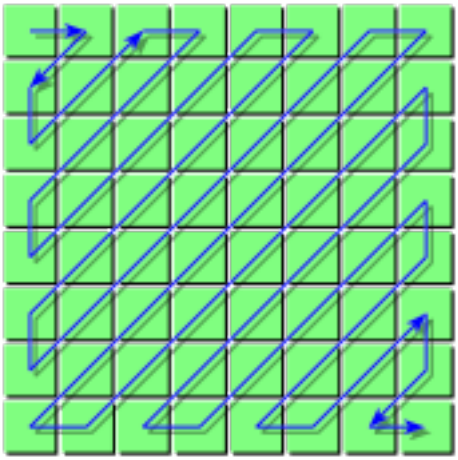
donde

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

La pseudo-inversa de la cuantización se obtiene realizando:

$$\tilde{B}_{dct_{(i,j)}} = (B_{q_{(i,j)}} \cdot Q_{(i,j)}) \qquad 0 \leq i,j \leq 7$$

Finalmente, en la etapa de **codificación** se recorre la matriz cuantizada en forma de *zigzag* como muestra la siguiente figura:



y se codifica cada uno de los valores de la siguiente manera: $B_{q_{(0,0)}}$ se agrega al bitstream codificado tal cual está. Luego, se cuentan la cantidad de apariciones de ceros en la matriz (recorriéndola en forma de zigzag) antes del primer valor no nulo. Se agrega al bitstream codificado esta cantidad y a continuación el valor no nulo. Este procedimiento se repite hasta recorrer toda la matriz. En caso de llegar al final de la matriz y no haber un valor no nulo (es decir, el elemento $B_{q_{(7,7)}}$ también es cero), se agrega la cantidad de ceros contada y luego, un cero. Finalmente, al terminar de codificar la matriz se agrega al bitstream codificado dos ceros seguidos para indicar el fin de la codificación. Veamos un ejemplo. Dada una matriz:

$$M = \begin{pmatrix} 160 & 51 & 0 & 16 & \cdots \\ 12 & 0 & 1 & \cdots & \cdots \\ 0 & 0 & \cdots & \cdots & \cdots \\ 14 & \cdots & \cdots & \cdots & 0 \\ 5 & \cdots & \cdots & 0 & 7 \\ \cdots & \cdots & 1 & 0 & 0 \end{pmatrix}$$

su codificación resulta (los paréntesis están agregados para clarificar la lectura): 160 (0 51) (0 12) (3 16) (0 1) (1 14) (0 5)... (? 1) (2 7) (2 0) (0 0)

## 3. Enunciado

El objetivo de este trabajo práctico es realizar un programa que dado un archivo `.bmp` comprima los datos correspondientes a la imagen utilizando la codificación antes mencionada. El resultado va ser un archivo `.joc2` que va a tener un encabezado (header) que incluya el encabezado del `.bmp`, y los datos de la imagen comprimidos. También se pide realizar un programa que dado un archivo `.joc2` descomprima los datos y lo convierta a un archivo `.bmp`. En este caso el archivo `.bmp` obtenido será ligeramente diferente del original, ya que el método de compresión propuesto es con pérdida de información.

Las funciones de interacción con el usuario y manejo de archivos pueden estar implementadas en lenguaje C. El resto de las funciones deben implementarse en lenguaje ensamblador. El prototipo de las funciones debe ser definido por los alumnos.

Los programas de compresión y descompresión deben soportar sólo el formato `BMP` de color real de 24 bits (cada pixel se representa con una terna de bytes `RGB`). No hay restricciones para el tamaño del archivo. Pueden asumir que las dimensiones de la imagen son múltiplo de 8.

Las funciones que se piden implementar en lenguaje C son las siguientes:

- `bmp2joc2`: programna principal para comprimir. Sintaxis de uso:
  `bmp2joc2 nombreArchivoEntrada.bmp nombreArchivoSalida.joc2`.
- `joc22bmp`: programa para principal descomprimir. Sintaxis de uso:
  `joc22bmp nombreArchivoEntrada.oc2 nombreArchivoSalida.bmp`.
- `readbmp`: levanta las estructuras del archivo `BMP` (Header e InfoHeader) y copia los datos de la imagen (pixels) en tres buffers (uno por cada canal).
- `writebmp`: escribe el Header y el InfoHeader y copia los datos ya descomprimidos de la imagen que están tres buffers (uno por cada canal) en el archivo `.bmp`.
- `readjoc2`: levanta las estructuras del archivo `JOC2` (Header del `.joc2`, Header e InfoHeader del `.bmp`) y copia el bitstream a un buffer.
- `writejoc2`: escribe el Header del `.joc2`, el Header e InfoHeader del `.joc2`, y copia el bitstream comprimido que está en un buffer en el archivo `.joc2`.

Para estas funciones se recomienda utilizar las funciones `fopen`, `fread`, `fwrite` y `fclose` de la librería de C `stdio.h`.

Las funciones que se piden implementar en lenguaje ensamblador son las siguientes:

- `dividirEnBloques`: recibe un canal de los datos de la imagen y devuelve el bloque de la imagen de $8 \times 8$ a partir de las coordenadas pasadas como parámetro.
- `generarDCT`: genera la matriz DCT, utilizando la FPU. Debe llamarse una única vez en todo el programa. Se deben usar números de punto flotante de precisión simple de 32 bits.
- `transformar`: recibe un bloque de la imagen de $8 \times 8$ y aplica la transformación. La multiplicación de matrices se debe implementar utilizando el set de instrucciones SSE. Se deben usar números de punto flotante de precisión simple de 32 bits. Se recomienda utilizar la técnica de *loop unrolling* para optimizar el código de la multiplicación.
- `cuantizar`: recibe un bloque ya transformado y lo cuantiza. Esta función debe ser implementada utilizando el set de instrucciones SSE. El resultado de la cuantización debe devolverse en números enteros de 16 bits de precisión. Se recomienda utilizar la técnica de *loop unrolling* para optimizar el código de la división.
- `codificar`: recibe un bloque cuantizado y devuelve su respectivo bitstream codificado. El tamaño de los datos del bitstream es de 16 bits.
- `decodificar`: recibe un puntero el bitstream comprimido de un determinado canal, decodifica un bloque y actualiza el puntero.
- `decuantizar`: realiza la operación inversa a la cuantización.
- `antitransformar`: realiza la operación inversa a la transformación.
- `unirBloques`: toma un buffer correspondiente a un canal, un bloque de ese canal y una coordenada dentro del buffer y copia el bloque a partir de dicha coordenada.

Y todas las funciones auxiliares que puedan llegar a utilizar.

# 4. Informe y forma de entrega

El informe debe reflejar todo el trabajo realizado, las decisiones tomadas (con el estudio de sus alternativas), las estructuras de datos usadas (con gráficos y/o dibujos si ayudan a clarificar), el testing que hayan hecho para detectar errores y optimizar código y los resultados obtenidos. Debe contar como mínimo con los siguientes capítulos: introducción, desarrollo, resultados y conclusiones. Debe estar estructurado top-down, es decir, leyendo la introducción se debe saber qué se hizo y cuáles son las partes más importantes. En el capítulo de desarrollo se deben detallar las decisiones que se tomaron, las estructuras de datos que se utilizaron y la implementación de cada una de las funciones, particularmente las implementadas en lenguaje ensamblador (se recomienda entregar pseudo-código). El capítulo de resultados debe contener

las pruebas de compresión realizadas con el nuevo método de compresión. Para variar las tasas de compresión (y pérdida de calidad) pueden probar con distintas matrices de cuantización $Q$. Es importante comparar los resultados obtenidos con los del Trabajo Práctico 1. Por último, se presentan las conclusiones del trabajo. Además, el informe debe incluir una carátula (con nombre del grupo y de los integrantes con número de libreta y email), instrucciones para el corrector (por ejemplo, como ensamblar los archivos fuente para obtener el ejecutable) y el detalle de todos los archivos entregados. La fecha de entrega de este trabajo es el martes 2 de diciembre, en el horario de clase (de 17 a 22 hs). No se aceptarán trabajos pasada esa fecha. No se puede entregar una hoja que sea sólo la carátula. Si un grupo decide no entregar nada en la primera fecha, va directo a recuperatorio. No hay segundo recuperatorio. Por eso insistimos en que conviene que entreguen en primera fecha lo que tengan hecho hasta el momento para que les corrijamos esa parte. Documenten a medida que realizan el trabajo, no dejen esto para el final. Se pide entregar una carpeta con el informe del trabajo impreso y un CD con las siguientes directorios:

- src: contiene los archivos fuente.
- exe: contiene los archivos ejecutables.
- enunciado: contiene este documento.
- informe: contiene el informe en formato .pdf.
- resultados: contiene los archivos .bmp y sus correspondientes archivos comprimidos .joc2.