Organización del computador II

Segundo Cuatrimestre de 2008

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico 2

Compresión y descompresión de imagenes

Grupo: Usain Bolt

Integrante	LU	Correo electrónico
Ya Lun Wang	623/06	aaron.wang.yl@gmail.com
Agustín Olmedo	679/06	agustinolmedo@gmail.com
Carlos Sebastian Torres	723/06	sebatorres1987@hotmail.com

Índice

1. Introducción **2**

2. Desarrollo **3**
2.1. Algoritmo de división de bloques 3
2.2. algoritmo de traspuesta 3
2.3. matriz DCT 3
2.4. algoritmo de transformación 4
2.5. Cuantización del bloque transformado 4
2.6. Codificación 5
2.7. Proceso de decodificación 6

3. Resultados **6**

4. Conclusiones **6**

1. Introducción

En este nuevo trabajo práctico de la materia, nuevamente tendremos que comprimir el tamaño de una imagen de formato BMP. Sin embargo, en esta ocación, la compresión de datos implicará en una leve pérdida en la definición de la imagen. Dicha pérdida no es significativa a los ojos humanos, pero en compensación permitirá una mayor ahorro de espacio. Recordemos que con la codificación empleando la codificación de Huffman tenía la desventaja de no rendir bien para aquellas imágenes en las cuales había una amplia gama de colores. De esta forma, vamos a implementar un nuevo método en el que se logre salvar esta desventaja que se presentaba en el trabajo anterior.

El tipo de compresión que vamos a emplear en este trabajo corresponde a aquellas denomidas como lossy debido a la casi imperceptible pérdida de calidad que se tiene al comprimir y descomprimir. Este tipo de compresión es muy común en los formatos multimedia de video y de audio, y se aprovechan de ciertas imperceptividades del sentido humano que compensa la pérdida de calidad, con ejemplos como la eliminación de frecuencias de audio, o leve la transformación de colores. El formato empleado para lograr la compresión será el JOC2, del cual su header se da a conocer en la consigna del trabajo.

La implementación ligada a la transformación del formato original BMP al formato JOC2 será basado en los siguientes pasos: Primeramente, se separará la imagen en tres distintos canales que son los colores primarios que componen a la imagen. Para cada uno de esos canales, se les subdividirá en bloques de 8x8 bytes, y en cada uno de estos bloques se les realizará la etapa de transformación, cuantización y codificación. Luego, finalizamos el proceso guardando la codificación en el bitstream de salida en un nuevo archivo.

En la etapa de transformación, emplearemos la matriz denominada Transformada Discreta del Coseno (de aquí en adelante DCT) y le realizaremos la multiplicación de matrices al bloque de 8x8 trabajado en ese momento, que se explicará en el desarrollo.

Al bloque transformado, se lo cuantizará realizandole una división cuyo denominador surgirá de una matriz arbitraria Q. A cada índice de la matriz transformada, se le hará el cociente con aquel elemento del mismo índice de la matriz Q.

Finalmente, mediante un método de recorrido en zig zag, se codificará guardando las sucesiones de 0, indicandose el fin de donde termina dicha sucesión.

2. Desarrollo

En esta sección del informe procederemos a explicar las distintas etapas del algoritmo. Recordemos que previamente habíamos mencionados cómo iban sucediendo las etapas, se puede resumir a partir del siguiente pseudocódigo:
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 x 8.
        5. Para cada bloque:
            6. **Transformar** el bloque.
            7. **Cuantizar** el resultado de la transformacion.
            8. **Codificar** el resultado de la cuantizacion.
            9. Guardar la codificacion en un bitstream de salida.
10. Generar el archivo con la imagen comprimida a partir de los bitstreams de todos los bloques.

A continuación, describiremos los detalles de implementación para los algoritmos que transforman, cuantizan y codifican bloques. Tambíén documentaremos otros algoritmos tales como aquel que separa los bloques, la que genera la matriz traspuesta y la de la decodificación.

2.1. Algoritmo de división de bloques

En este paso procedemos a devolver un bloque dado una imagen en BMP y la coordenada de la posición (1,1) del bloque de la que se desea obtener. La precondición de este algoritmo pide que la coordenada dada como parámetro sea efectivamente el primer elemento (1,1) del bloque elegido. Dado los parámetros, el procedimientos toma la primera fila y la copia a la nueva matriz de 8x8 que es lo que fue reservado como parte de la solución que se quiere devolver. Luego, repito para las 7 filas restantes de tamaño 8, mediante un salto de fila. Cabe destacar que este algoritmo no presenta saltos condicionales, de modo que la copia de filas se da a manera de macros que se repiten en el código implementado, a modo de optimizar el rendimiento del algoritmo.

Cada bloque será trabajo a partir de los procesos anteriormente mencionados.

2.2. algoritmo de traspuesta

Este algoritmo nos servirá para el algoritmo que se usará en la transformación. Recordemos que una matriz traspuesta es aquel lograda de transformar las filas en columnas. Para dicho algoritmo simplemente tomamos un elemento (i,j) y luego lo reubicamos en la posición (j,i).

2.3. matriz DCT

Recordemos que la matriz DCT tiene los valores definidos de la siguiente forma:

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

donde

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

Esta matriz es igual para cualquier bloque que vayamos a codificar debido a que toma como variables los índices de 0 a 7. Como estos no varían nunca, podemos lograr dicha matriz en un solo

cálculo mediante el módulo de FPU. Un detalle importante a tener en cuenta en todo estos es que para la operación del coseno se tuvo que realizar la transformación a radianes, debido a que esa es la precondición que se tiene dicha operación en FPU.

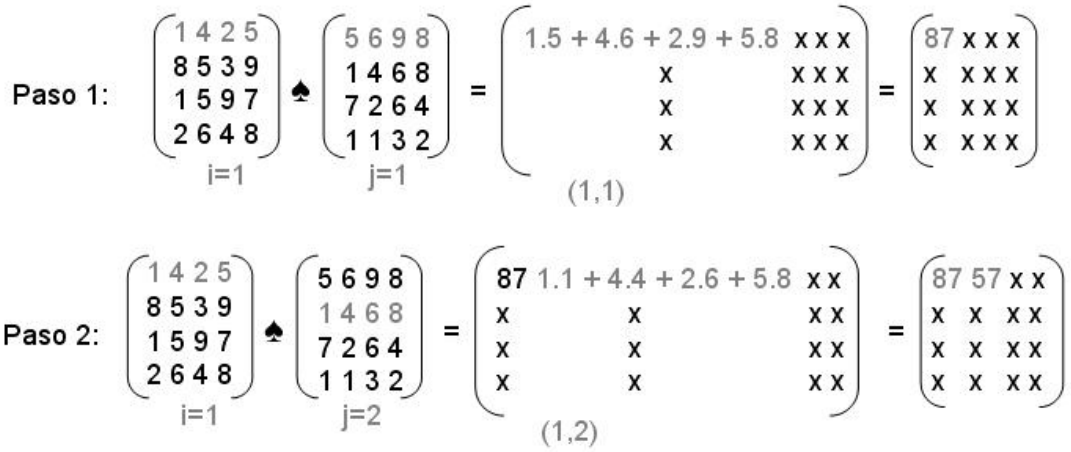
2.4. algoritmo de transformación

El algoritmo de transformación consiste en tomar un bloque, y mediante la matriz DCT aplicarle la siguiente fórmula matemática.

$$B_{dct} = DCT.B^t.DCT^t$$

donde B_{dct} es la nueva matriz resultado.

En principio, vamos a definir la operación ♠ entre dos matrices de iguales dimensiones como el producto de una fila con el de todas sus filas de la forma indicada en la siguiente figura:



Y así sigue hasta recorrer todas los elementos....

Si observamos detenidamente, vemos que la operación ♠ sirve para definir a la operación de producto de matrices:

$$A♠\ B = A.B^t$$

Esta operación ♠ es fácil de implementar en lenguaje ensamblador SSE dado que es fácil aprovechar el producto de filas paralelas. Así, en la fórmula que presentamos anteriormente para B_{dct}, podemos reescribirla de la forma siguiente:

$$B_{dct} = DCT.B^t.DCT^t$$
$$B_{dct} = (DCT♠B).DCT^t$$
$$B_{dct} = (DCT♠B)♠DCT$$

Veamos lo que ocurre cuando queremos realizar el proceso inverso cuando queremos recuperar el bloque original. Conocemos que la fórmula para obtenerlo es:

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

Dado que la traspuesta cumple con la propiedad distributiva con respecto al producto de matrices, podemos reescribir lo anterior para lo siguiente:

$$B = DCT.B_{dct}^t.DCT^t$$

Mirando esta última fórmula podemos deducir que el proceso por el cual queremos obtener lo original, es el mismo con respecto al que empleabamos para lograr la transformación de un bloque.

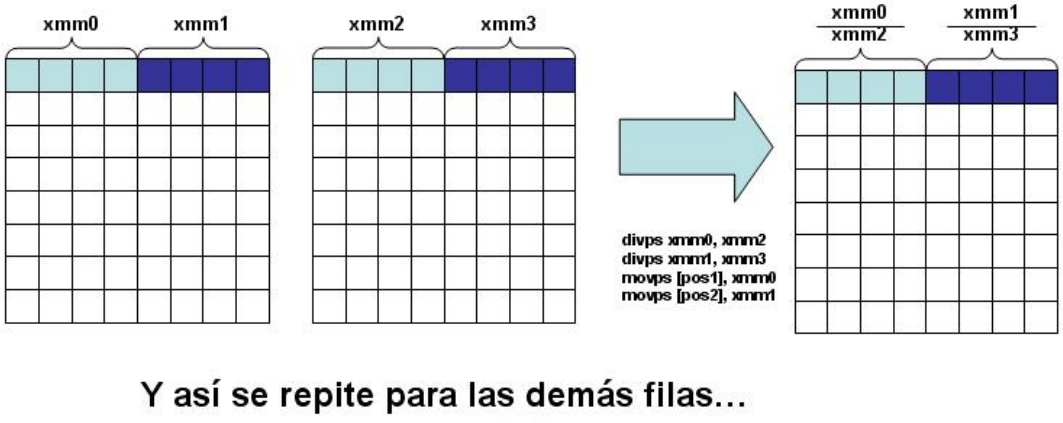
2.5. Cuantización del bloque transformado

La idea general de este algoritmo consiste en hacer una correspondencia univoca entre las coordenadas de amabas matrices y luego, realizarseles la división. Esto es ideal para el procesamiento en paralelo, de modo que se eligió emplear las intrucciones de SSE para la implementación de este procedimiento.

Primeramente, convertimos los números enteros de la matriz Q en puntos flotantes de presición simple mediante operaciones que nos son provistas por SSE. Luego cargamos una fila entera de una matriz, y otra fila de la matriz Q empleando 2 registros de SSE para cada una de las filas, y realizamos la división elemento a elemento. Posteriormente transformamos los resultados de esta división en enteros de 32 bits. Como el prototipo requiere que sean enteros de 16 bits, emplearemos las operaciones de unpack para dicha conversión. Estos pasos los repetimos para las 7 filas restantes que nos falta para procesar.

En este algoritmo se emplea la tecnica de loop unrolling al eliminar cualquier salto condicional en el algoritmo. Debido a que la cantidad de iteraciones en las cuales se procesa una fila para realizarsele la división elemento a elemento con respecto a la otra fila de la otra matriz, es una cantidad finita. Podemos repetir el código que corresponde a la división de una fila por otra, unas 8 veces en total, de manera que evitamos el empleo de saltos condicionales. De esta manera, empleamos macros en las cuales repetimos la secuencia de código en los cuales estamos realizando la suma paralela entre las dos filas.

Aquí podemos observar facilmente que es en esta etapa del algoritmo en el que se produce la pérdida de calidad debido a que el redondeo de la matriz transformada, produce un cambio sobre lo que después será el bloque decodificado. El redondeo del número hace que en cierto valores se pierda la precisión de estos números, de cuya decodificación se hace sentir un leve cambio.



Y así se repite para las demás filas...

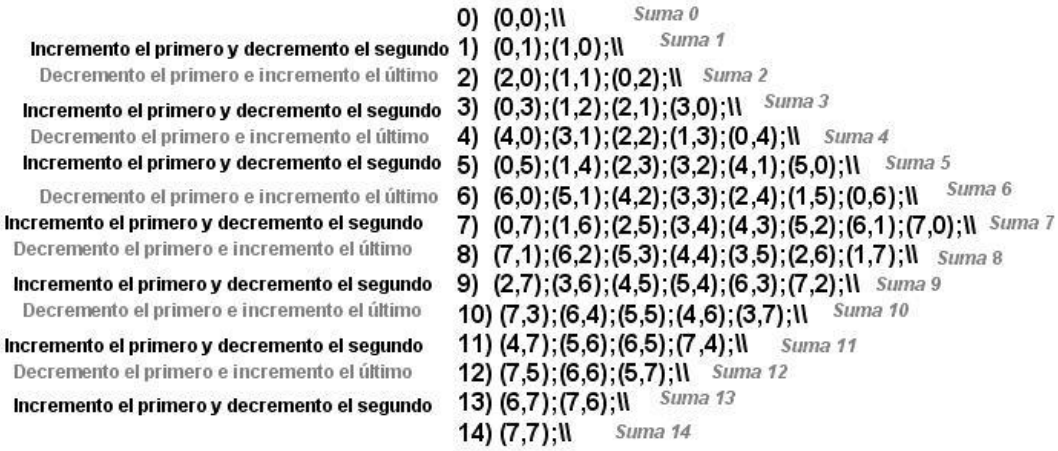
2.6. Codificación

Una vez conseguido el bloque cuantizado, vamos a codificar para obtener efectivamente el bitstream que nos posibilita la compresión de la imagen. La codificación consiste en agregar en la posición (0,09 tal cual está y luego recorrer en zigzag como se mostró anteriormente, contando la cantidad de apariciones de ceros antes de toparse con un valor no nulo. Se escribirá entonces sobre el bitstream dicha cantidad, más el valor no lulo al cual se encontró antes de terminarse el conteo.

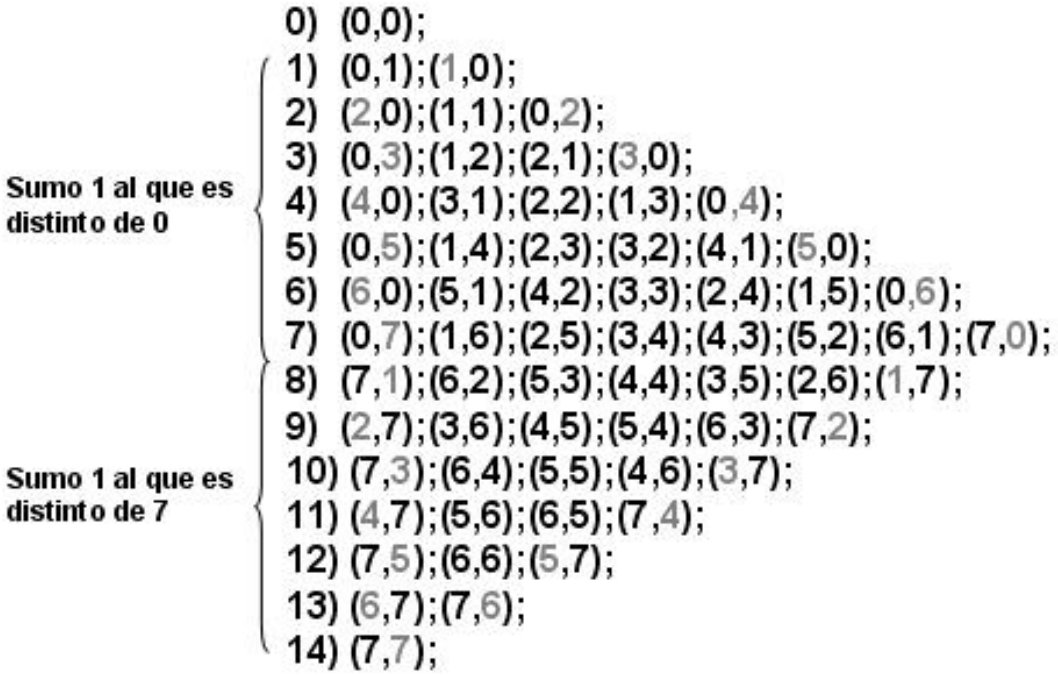
A partir de una tupla de dos números podemos indicar cómo se van recorriendo los elementos de la matriz. Esta tupla (i,j) puede interpretarse en el mundo de la implementación como avanzar i filas más j columnas. Si vemos la figura anterior, podemos resumir el recorrido a partir de la siguiente secuencia de tuplas:
0) (0,0);
1) (0,1);(1,0);

2) (2,0);(1,1);(0,2);
3) (0,3);(1,2);(2,1);(3,0);
4) (4,0);(3,1);(2,2);(1,3);(0,4);
5) (0,5);(1,4);(2,3);(3,2);(4,1);(5,0);
6) (6,0);(5,1);(4,2);(3,3);(2,4);(1,5);(0,6);
7) (0,7);(1,6);(2,5);(3,4);(4,3);(5,2);(6,1);(7,0);
8) (7,1);(6,2);(5,3);(4,4);(3,5);(2,6);(1,7);
9) (2,7);(3,6);(4,5);(5,4);(6,3);(7,2);
10) (7,3);(6,4);(5,5);(4,6);(3,7);
11) (4,7);(5,6);(6,5);(7,4);
12) (7,5);(6,6);(5,7);
13) (6,7);(7,6);
14) (7,7);

Podemos distinguir un cierto patron en estas tuplas de numeros. Podemos ver que del índice 0 al 14 la suma interior de los números dentro de cada tupla, es igual al índice que se representa. Otro elemento interesante que podemos notar es que las tuplas se generan restando su primera componente y luego sumandosela a su segundo hasta que la primera componente sea 0, de manera inversa según la paridad o imparidad del número de índice. También podemos observar que las sumas ocurren alternadamente. Esto quiere decir que para el indice 1 se le incrementa al primer termino de la tupla mientras se le decrementa al segundo; y posteriormente al segundo ocurre lo inverso, donde se le decrementa al primer término y luego se le incrementa al segundo, y así repitiendose este patrón hacia el final. Una última observación no menos importante consiste en que a partir del séptimo índice, el valor de uno de los términos no puede seguir incrementandose a más de 7. De esta forma, se le suma al otro indice.



En el algoritmo esto se traduce en un ciclo externo que recorre del índice 1 al 14 (el caso 0 está fuera del ciclo) y para lo que hay dentro de estas iteraciones haremos la aumento del primer índice y el decremento del segundo, o el decremento del primer indice y el aumento del segundo. Esto dependerá, como vemos en la figura, de la paridad del índice con el que estamos trabajando. Habrá entonces dos ciclos internos que harán alguna de estas dos operaciones. Otro detalle relacionado a la implementación es que cuando pasamos de un índice al otro, del 1 al 7, le sumamos un uno al índice distinto del 0, y para el 8 al 14 estamos sumando al índice distinto del 7, con respecto al número dejado por la última tupla generada por la iteración previa (el caso (0,0) representa un caso aparte dado que tiene dos 0, de modo que se agrega arbitrariamente para que cumpla con la regla que determina el recorrido)



Sin embargo, el recorrido es apenas la primera parte de este algoritmo. A medida que se va recorriendo los elementos de la matriz. Habiamos mencionado brevemente que el algoritmo cuenta la cantidad de 0, antes de llegar a encontrarse con el primer número que no es 0. De esta forma, dentro del algoritmo, existe un contador de 0 que irá incrementandose siempre y cuando el número que aparece sea un 0. De manera contraria a esto último, se interrumpe dicho conteo y se escribe el valor acumulado hasta el momento, más el valor encontrado no nulo, en forma de una tupla de dos números donde el primero es el valor acumulado de ceros, y el segundo como el número encontrado no nulo que interrumpe el conteo. El caso aislado de esta forma de codificación es el primer valor que se escribe tal cual está.

2.7. Proceso de decodificación

Para el algoritmo de decodificación tenemos que interpretar la información que se obtuvo en la codificación. Esto no es ningún inconveniente grande debido a que debemos escribir sobre el buffer de la imagen, la cantidad de ceros indicados en la primera parte, y luego escribir el número que interrumpe el conteo. Esto nos permitirá volver a los valores de la matriz en el bloque cuantizado.

Una vez conseguido el bloque cuantizado, debemos realizar la multiplicación elemento a elemento, que es el proceso inverso hecho en la cuantización previamente obtenida. Esto nos dará un valor cercano al original (debido al redondeo de cifras).

La antitransformación, como fue explicado anteriormente aprovecha la propiedad distributiva de la traspuesta con respecto al producto, de modo que el mismo algoritmo de la transformación, sirve para lograr la antitransformación del bloque.

3. Resultados

4. Conclusiones

Nos parecio interesante aprender otros métodos de compresión en las cuales se permitieran pequeñas pérdidas en la calidad de la imagen, por el costo de un mayor ahorro logrado en el espacio de almacenamiento. Como hemos mencionado antes, este método es ampliamente utilizado en la vida cotidiana, y es sumamente aplicado al ámbito multimedia, para formatos tales como MP3 y JPEG. Si bien nos basamos en ciertas precondiciones para los cuales se limita su uso para ciertos casos (tales como imágenes con múltiplo de 8x8) el método que empleamos en la implementación de este trabajo nos ha sido muy fructífero.