[[1]](#footnote-1)

Un Algoritmo Para La Compresión De Imágenes Múltiples Utilizando la Transformada Discreta del Coseno y Block Matching

Alejandro Valdés, María C. Bustos  
Universidad del Valle

*Abstract*—We have developed and algorithm that is capable of compressing multiple successive images using the discrete cosine transformation (DCT) to lower the size of the file, also, it’s used block matching between adjacent frames to reduce even more the quantity of information we are keeping on the file. The algorithm work using a cosine transformation matrix of size 8x8, and applies a simple encoding using tuples.

*Index Terms*—Image Processing, Image Motion Analysis, Image Coding, Image Storage

# INTRODUCTION

l

A compresión de imágenes es un campo muy importante hoy en día, debido a las cantidades enormes de información que se generan diariamente, se deben crear métodos inteligentes para guardar esta información, en nuestro caso, imágenes digitales.

Existen una gran cantidad de formas para guardar la información de las imágenes, usando compresiones *lossless*, o compresiones con perdida. Para nuestro algoritmo estamos usando la transformada discreta del coseno, que nos permite reducir la información contenida en una ventana en unos pocos valores, pero se pierde parte de la información original.

Nuestro algoritmo está basado en dos ideas principalmente, la compresión con pérdida de la transformada discreta del coseno, y el uso de predicción de movimiento (block matching), para imágenes sucesivas. Esto con el fin de reducir la información que conservaremos de cada una de las imágenes o fotogramas.

# Prior and Related Work

Existe una gran cantidad de trabajo en esta área de compresión de imágenes y detección de movimiento por block matching, podemos ver los formatos H.264, o los estándares que se usan en la web, como JPEG (que usa la transformada del coseno), el MPEG-4, etc. Tenemos el trabajo realizado en block matching por Victor Padilla Ramirez, con el Algoritmo de Block-Matching Usando Búsqueda en Árboles.

# Description of the Algorithm

El algoritmo implementado esta dividido en varias fases, a continuación mostraremos cada una de las etapas de este y su funcionamiento.

Decidimos usar el formato PGM en formato ASCII crudo para las imágenes de prueba, debido a que solo contienen los valores de luminancia (escala de grises), para simplificar las pruebas,

Para comprimir imágenes, el algoritmo se divide en varios modulos, cada uno con una función muy especifica:

## Aplicada de la transformada del coseno

Consideremos la imagen como una matriz I de tamaño mxn, donde m y n son múltiplos de 8, entonces, dividiremos esta matriz en matrices más pequeñas, X, que serán de un tamaño 8x8, y a estas le aplicaremos la transformada del coseno, según la siguiente formula:

Y su inversa:

Donde A es la transformada del coseno, y esta dada por la siguiente ecuación:

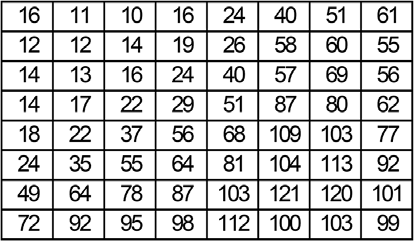
Debemos aclarar que el valor de N en (3) está dado por el tamaño de una dimensión de la matriz X, en este caso, 8.

## Normalización de los valores a través de la cuantización escalar

Cada uno de los valores obtenidos c on la ecuación (1), se deberán normalizar, usando la siguiente formula:

Donde QP es una matriz de tamaño idéntico a X, y está predefinida. Según la elección del QP, podemos aumentar la compresión sacrificando la calidad de la imagen, o viceversa.

Para nuestras pruebas, hemos decidido usar tres matrices QP, definidas a continuación:



(7) QP para luminancia de jpg [1]

El objetivo de esta operación, es llevar gran parte de los valores obtenidos a cero, con el fin de optimizar la compresión de los datos, y reducir el tamaño de la imagen original.

## Recorrido en Zig-Zag de la matriz X

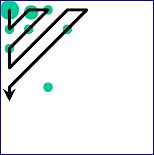


Fig. 1: Ejemplo de un recorrido zig-zag sobre la matriz, observe que el objetivo de hacer un recorrido así, es agrupar el conjunto de ceros (0) que se obtienen en la matriz después de aplicar la cuantización escalar

Obtendremos un vector de tamaño 64, donde los ceros estarán agrupados.

## Cambiar la representación del vector a tuplas

En el paso anterior, se obtuvo un vector de 64 elementos, donde se pueden encontrar muchos ceros seguidos, en estos casos, y es lo esperado por el algoritmo de compresión, usamos una nueva representación de los datos, llamada Tuplas run-level, de la siguiente forma:

(Número de ceros antes del dato, dato)

De esta forma, se agruparán los ceros que se encuentren, y se reducirá el espacio que ocupa la imagen físicamente.

## Aplicar una codificación entrópica (VLC)

Después de aplicar los pasos anteriores, se puede aplicar una codificación de este tipo para reducir aún más el espacio, pero eso esta fuera del alcance de este algoritmo.

## Descompresión de la imagen

La descompresión se lleva a cabo a través de aplicar cada la operación inversa de cada uno de los pasos mencionados anteriormente, llevando cada uno de los cuadros transformados a la forma de matriz I.

## Compresión de imágenes multiples

Usando cada uno de los pasos mencionados anteriormente, podemos aplicar esto a la compresión de multiples imágenes (videocompresión), usando algunas operaciones extras que mencionaremos a continuación.

### Residual de las imágenes

Consideremos I como la imagen actual, e I’ como la imagen anterior a esta, entonces, el residual está definido como:

### Block-Matching

El Block-Matching es una operación de predicción de movimiento, y esta definido también sobre la imagen I e I’, existen diversos algoritmos de block-matching, en nuesta aproximación, lo hicimos de la siguiente forma:

El Block-Matching de I sobre I’ se obtiene separando la imagen I en bloques de 8x8, y para cada uno de estos bloques, buscar el más parecido en la imagen I’, siguiendo los siguientes pasos:

1. Considere el bloque base W como el bloque (i,j) de la imagen I.
2. Tomar la posición, definida como (i,j), del bloque actual en I ir a ese mismo bloque en la imagen I’, definido como V, calcular su MSE respecto a W y guardarlo.
3. Comparar cada uno de los bloques que se puedan crear en el recuadro (i-8, j-8), esquina superior izquierda, hasta (i+8, j+8), variando solo un pixel en el tamaño del bloque, sea en i o en j, con el bloque que se toma como base; si se encuentra un bloque con un MSE menor al bloque inicial, tomaremos ese bloque como V, y guardaremos su MSE.
4. Si encontramos un bloque con un MSE menor al inicial definido por el (i,j), moveremos nuestro recuadro de búsqueda, y pondremos como centro el nuevo (i,j)
5. Repetir los pasos 3 y 4 hasta que A no varie en una iteración

De esta forma obtenemos los vectores de dirección del bloque (i,j).

Con estas dos nuevas operaciones, podremos comprimir los fotogramas, relacionándolos con el fotograma anterior, al crear una imagen con los bloques encontrados, para crear una imagen más parecida a la actual, con el objetivo de reducir la información que se queda en la imagen residual, para aumentar la compresión del video. Debemos aclarar que a la primera imagen del conjunto no se le puede aplicar el residual, porque no tiene una imagen que la preceda, así que esta imagen es guardada como una normal, ya que será la base para las siguientes imágenes.

## Descompresión de imágenes multiples

Para Realizar la descompresión de imágenes multiples (fotogramas), se aplicará la operación inversa a (8), definida como:

Donde I es la imagen que queremos conocer, e I’ es la imagen inmediatamente anterior, que ya conocemos.

# Experimental Results

Se realizaron varias pruebas al algoritmo de compresión de imágenes, usando tres imágenes imágenes diferentes y tres QP distintos como los definimos en (5), (6) y (7).

Mostraremos primero los resultados para el algoritmo de compresión de imágenes, usamos tres imágenes, de distintos tamaños y carateristicas diferentes, calculamos su tamaño final, y su PSNR, además de los promedios de todos los resultados:

TABLE I

Resultados de la Aplicación Del Algoritmo de Compresión de Imágenes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Imagen:** | **Adela** |  |  |
|  | **Original** | **QP de 12** | **QP de 50** | **QP de jpg** |
| **Tamaño** | 2877519 | 678404 | 222860 | 518160 |
| **%** | 100,0% | 23,6% | 7,7% | 18,0% |
| **PSNR** | - | 41,4734 | 33,4192 | 39,1904 |
|  | **Imagen:** | **Wonito** |  |  |
|  | **Original** | **QP de 12** | **QP de 50** | **QP de jpg** |
| **Tamaño** | 2526397 | 623422 | 293418 | 336103 |
| **%** | 100,0% | 24,7% | 11,6% | 13,3% |
| **PSNR** | - | 43,7424 | 33,0493 | 34,0481 |
|  | **Imagen:** | **Lenna** |  |  |
|  | **Original** | **QP de 12** | **QP de 50** | **QP de jpg** |
| **Tamaño** | 951770 | 309151 | 88687 | 172273 |
| **%** | 100,0% | 32,5% | 9,3% | 18,1% |
| **PSNR** | - | 38,7277 | 32,0815 | 35,4678 |
|  | **Promedios** | |  |  |
| **Tamaño** | **Original** | **QP de 12** | **QP de 50** | **QP de jpg** |
| **%** | 100,0% | 26,9% | 9,6% | 16,5% |
| **PSNR** | - | 41,3145 | 32,85 | 36,235433 |

Las unidades del Tamaño están en Bytes.

Como podemos ver en la tabla anterior, usando la compresión, en el peor de los casos obtuvimos un 43% de compresión respecto al tamaño original, usando a (5) como QP, y en el mejor de los casos, obtuvimos una compresión del 7,7% del tamaño original, usando el QP definido en (6). También vemos que la imagen más parecida a la original es la que tiene el QP (6), pero también es la más tiene tamaño después de la compresión.

Resultados de la Aplicación Del Algoritmo de Compresión de Imágenes: Comparación de Calidad (Visual)

Imagen Original (Fragmento):



Imagen Comprimida QP (5) (Fragmento):



Imagen Comprimida QP (6) (Fragmento):



Imagen Comprimida QP (7) (Fragmento):



Fig. 2: Ejemplo de la aplicación del algoritmo de compresión de imágenes sobre una imagen de muestra y multiples QP diferentes, observe la diferencia entre cada una de las imágenes.

Vemos en las imágenes anteriores los resultados, tomando un cuadro de muestra de 100x100 pixels, de Lenna, en este caso, vemos que la imagen que muestra más degradación visual, es la comprimida con el QP definido en (6), luego la imagen con QP (7), y por ultimo (5), todo de acuerdo a los resultados obtenidos en la Table 1 del calculo del PSNR respecto a la imagen original.

TABLE II

Resultados de la Aplicación Del Algoritmo de Compresión de Imágenes Sucesivas

|  |  |  |
| --- | --- | --- |
| **Video** | |  |
|  | **Tamaño** | **%** |
| **Original** | 3578332 | 100,0% |
| **Block-Matching** | 1485435 | 41,5% |
| **Compresion solo** | 2039723 | 57,0% |

Las unidades del Tamaño están en Bytes

En el resultado de la anterior tabla, vemos que usando la compresión del video sin block-matching obtenemos una relación de compresión alta (57%), y usando el block-matching obtenemos un 41,5%, esto es un 16,5% menor que la compresión sin aplicar block-matching, obteniendo un mejor resultado.

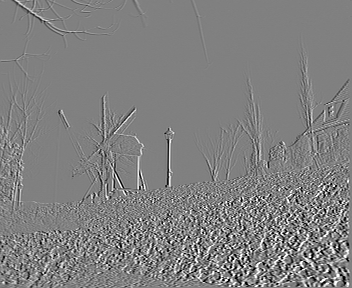


Fig. 2: Residual de dos imagenes de compresión de video sin usar block-matching

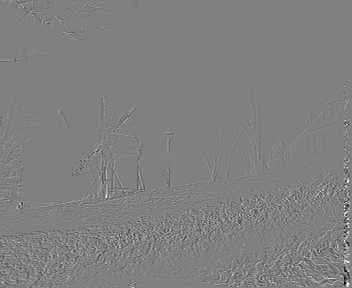


Fig. 3: Residual de dos imagenes de compresión de video usando block-matching

En las imágenes anteriores, vemos la diferencia entre usar el block-matching y no usarlo, como vemos, la cantidad de información que guardamos se reduce drásticamente al usar block-matching, ya que estamos restando las imágenes, pero forzamos que sean más parecidas.

# Conclusions

- En el algoritmo de Block Matching, una decisión fundamental es la elección del tamaño del bloque de búsqueda, debido a que, desafortunadamente, entre mas crezca el bloque de búsqueda, la complejidad del algoritmo aumenta a su vez. Se eligio 8 como el tamaño del bloque porque con este tamaño el tiempo de ejecución del algoritmo es aceptable y la calidad de la imagen no se degrada tanto.

- La compresión de imágenes con pérdida puede llegar a reducir mucho el tamaño original de una imagen, pero esta reducción es inversamente proporcional a la calidad de la imagen, llevando a imágenes con tamaños muy pequeños, pero con una calidad muy baja.

- Una elección muy importante en el momento de comprimir imágenes usando la transformada discreta del coseno, es el QP, ya que según la elección, este podría reducir la información en mucha o poca cantidad, sacrificando la calidad de la imagen

- Al aplicar Block-Matching sobre las imágenes multiples, la perdida de calidad se hizo muy evidente con las imágenes sucesivas, debido a que el block-matching se calculaba sobre la imagen original, y no sobre la comprimida, que es la que conoce el algoritmo, entonces, se acumula ese error en cada una de las iteraciones para rearmar el video

References

1. H.R. Wu and K.R. Rao, “*Digital Video Image Quality and Perceptual Coding*” (Book)” Ed. New York: Taylor-Francis Group, 2006, pp. 33.
2. Iain E. Richardson, “*The H.264 Advanced Video Compression Standard”* (Book style) Second Edition, Ed.Wiley, 2010, pp. 50–51.

1. Manuscript received June 24, 2011.

   Alejandro Valdés Villada, Autor, Vinculado con la Universidad del Valle como estudiante de Pregrado de Ingenieria de Sistemas.

   María Cristina Bustos Rodríguez, Autora, Vinculada con la Universidad del Valle como estudiante de Pregrado de Ingeniería de Sistemas [↑](#footnote-ref-1)