Compresión de Imágenes mediante KLT

Gustavo Brown Rodríguez

Resumen

El presente informe describe la implementación de un codec de imágenes utilizando la transformación de Karhunen-Loève y los resultados obtenidos en la compresión de un conjunto de imágenes de prueba. Asimismo se realiza una comparación de rendimiento entre el codificador implementado y un codificador JPEG sobre el mismo conjunto de imágenes de prueba.

1. Introducción

El presente artículo describe el estudio e implementación de un codificador y decodificador de imágenes utilizando la transformación de Karhunen-Loève(***KLT***).
La sección 2 introduce brevemente la KLT, la manera de calcularla y su aplicación al contexto de compresión de imágenes. La sección 3 comenta la implementación de un codificador/decodificador de imágenes basado en la KLT. La sección 4 describe los resultados obtenidos al comprimir las imágenes de un conjunto de prueba utilizando el codificador descripto en la sección 3 y lo compara con los resultados obtenidos al aplicar una codificación mediante DCT. Por último, la sección 5 concluye el articulo comentando los resultados y trabajos a futuro.

2. Transformación de Karhunen-Loève

La transformación de Karhunen-Loève(de aquí en más ***KLT***) es la transformación lineal cuya base de vectores ortonormales son los vectores propios de la matriz de covarianza de los datos que se desean procesar, y tiene algunas propiedades que la hacen muy útil en la compresión de imágenes. A continuación se comenta brevemente su deducción y propiedades[Dony01].
Se dice que un proceso esta decorrelacionado si dadas dos variables aleatorias x_i, x_j, la esperanza de su producto(o su covarianza) es:

$$E(x_i x_j) = \begin{cases} 0 \, si \, i \neq j \\ \sigma^2 \, si \, i = j \end{cases}$$

Si tomamos como elemento un bloque de N pixels de una imagen, donde a cada píxel se le puede asignar un valor (por ejemplo en el rango 0-255), podemos definir un vector $x=[x_1 \; x_2 \; ... \; x_{N-1} \; x_N]^T$ con una matriz de covarianza definida de la siguiente manera:

$$[C]_x = E\left((x-m)(x-m)^T\right)$$

donde m es la media E(x).

Para poder decorrelacionar los datos debemos buscar la transformación lineal [W] cuya transpuesta gire x para producir una matriz de covarianza diagonalizada:

$$y = [W]^T x$$

Esto se reduce a resolver el sistema de vectores propios de la matriz de covarianza $[C]_x$ de la siguiente manera:

$$[C]_y = E(yy^T) = E\left(([X]^T x)([X]^T x)^T\right) =$$
$$E([W]^T (xx^T)[W]) = [W]^T [C]_x [W] \Rightarrow$$
$$\Rightarrow [C]_X [W] = [W][C]_y$$

La solución para la matriz de covarianza $[C]_y$ definida como

$$[C]_y = \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_N \end{bmatrix}$$

tiene en la diagonal los valores propios de la matriz de covarianza de los datos $[C]_x$ y las columnas de la matriz [W] son los vectores propios.

Propiedades de la KLT

A continuación se describen algunas de las propiedades más importantes de la KLT descriptos en [DCIM03] y [Dony01].

Los coeficientes x_i de la KLT están decorrelacionados por construcción ($E(x_i x_j) = \lambda_j \delta_{ij}$).

La KLT minimiza la entropía definida como

$$\hat{\lambda}_j = \frac{\lambda_j}{\sum_{i=1}^{N} \lambda_i} \qquad H = -\sum_{i=j}^{N} \hat{\lambda}_j \log\left(\hat{\lambda}_j\right)$$

Las transformaciones lineales ortonormales son reversibles y por tanto en sí mismas no incurren en compresión alguna. La KLT, siendo una transformación lineal ortonormal no escapa a esta propiedad. Para reducir la cantidad de bits necesaria para codificar una imagen lo que se hace es cuantificarla. De esta manera al aplicar un proceso de cuantificación sobre los coeficientes de una imagen transformada podremos codificarla con una menor cantidad de bits, dado que parte de la información que los coeficientes tienen sobre imagen se ha perdido irreversiblemente. Posteriormente al aplicarle la antitransformada no obtendremos la imagen original sino una aproximación con mayor o menor grado de error. Otra técnica para reducir el tamaño en la codificación de la imagen es truncar la cantidad de coeficientes a describir en la imagen transformada. Al igual que con la cuantificación esto introduce un error en la reconstrucción de la imagen que queremos que sea lo menor posible. Aquí es donde la KLT es muy util, porque la KLT es la transformación unitaria que minimiza el error de truncamiento. Supongamos que de los N coeficientes solamente codificaremos la información relativa a los primero M de ellos. Por construcción de la KLT estos M coeficientes serán los vectores propios(w_i) con mayores valores propios (λ_i) de la matriz de covarianza de los datos de entrada. El error de truncamiento para la KLT puede entonces deducirse de la siguiente manera:

$$E(\varepsilon^2) = E\left(\frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2\right) = \frac{1}{N}E\left(\sum_{i=1}^{M}(y_i - y_i)^2 + \sum_{i=M+1}^{N}(y_i - 0)^2\right) = \frac{1}{N}E\left(\sum_{i=M+1}^{N} y_i^2\right) =$$

$$= \frac{1}{N}\sum_{i=M+1}^{N}\sigma_i^2 = \frac{1}{N}\sum_{i=M+1}^{N}\lambda_i$$

La ventaja de aplicar la KLT es que los M coeficientes que se codifican en la imagen son los que tienen mayor valor propio y por ende son los que aportan mayor información a la imagen. Dado que como se comento anteriormente la KLT minimiza la entropía, los M coeficientes pueden describirse con la menor cantidad posible de bits al ser codificados mediante un codificador de Huffman o vía codificación aritmética.

3. Implementación de un codificador de imágenes utilizando KLT

Esta sección describe la implementación en el lenguaje de programación Java de un codificador/decodificador de imágenes utilizando la transformación de Karhunen-Loève[Gusbro04]. El codificador implementado recibe imágenes de entrada de 1 banda (tonos de grises) e imágenes en 3 bandas (RGB). Opcionalmente se permite pasar las imágenes a color al espacio de luminancia y cromas (YCrCb) para ajustar la entrada a la capacidad psicovisual de una persona. En el caso de imágenes color trabajando en el espacio YCrCb se aplica un subsampleo aplicando la media 2x2 para las bandas asociadas a las cromas para optimizar la compresión en el archivo de salida.

Codificador KLT

La codificación de una imagen utilizando la KLT implica la realización de varias fases.
El codificador aplica secuencialmente estas fases a cada banda de la imagen de entrada y de la misma manera produce una imagen codificada en dichas bandas.
La base ortonormal de la transformada de la KLT es dependiente de los datos. Esto quiere decir que para cada imagen a la cual se le desea aplicar la KLT primero se deberá obtener los coeficientes de la base. Para ello se debe aplicar un análisis de componentes principales (Principal Component Analysis, PCA) mediante una descomposición en vectores propios como se explico en la sección 2.

La primer fase del codificador implica convertir los bloques de LxL en vectores v_i para aplicar el PCA, donde L es un parámetro del codificador que indica el lado de los bloques. Si la imagen de entrada no tiene un tamaño múltiplo de L, ésta se agranda temporalmente extendiendo los bordes hasta que el tamaño sea múltiplo de L. De esta manera el codificador siempre trabaja con vectores del mismo tamaño.
Para obtener la base de la KLT primero se obtiene la matriz de covarianza a partir de los vectores v_i de la imagen y luego se realiza una descomposición en vectores propios aplicando los algoritmos implementados en [Jama00].
Esta primera fase es en realidad opcional, porque el codificador permite levantar la base de la KLT a utilizar a partir de un archivo. Esto permite, por ejemplo, utilizar la base de la KLT obtenida a partir de una imagen y aplicar la codificación a otro set de imágenes; o simplemente no recalcular una y otra vez la base si se desea codificar la misma imagen con distintos parámetros de calidad.

En una segunda fase se procede a aplicar la KLT a la imagen de entrada. Para ello nuevamente se parte la imagen a procesar en bloques de LxL y se le aplica la transformada obteniendo los coeficientes del bloque en la nueva base.
El codificador permite definir, además de la calidad de la imagen de salida, la cantidad de coeficientes que se desean mantener en la imagen de salida. Como se ha explicado en la sección anterior, los coeficientes que se codificaran serán aquellos para los cuales los versores de la base corresponden a los vectores propios con mayores valores propios en la matriz de covarianza. Luego del mencionado truncamiento, se procede a cuantificar estos coeficientes aplicando el algoritmo descripto en la figura 1.
El primer coeficiente es tratado diferente y generalmente se codifica con mayor cantidad de bits. Este coeficiente es llamado el coeficiente de continua (DC) porque generalmente expresa el tono base de la banda a codificar. Los demás coeficientes suelen llamarse coeficientes de alterna (AC).
Para el codificador propuesto, la cuantificación del coeficiente de continua es independiente de la calidad de salida de la imagen. La cuantificación de los demás coeficientes (los de AC) sí tienen en cuenta el parámetro de calidad. Para estos coeficientes, la cuantificación es lineal con la calidad y exponencial con el número de coeficiente procesado. De esta manera, los coeficientes asociados a versores con mayor valor propio podrán describirse con mayor cantidad de bits. Adicionalmente, cuando se codifica en el espacio YCrCb, las bandas asociadas a las cromas Cr y Cb se cuantizan aún más.

Cuantificación coeficientes DC y AC

```
/** Cuantifica el componente de continua */
private double quantizeDC(double value)
{
   return value / DC_QUANTIZER;
}

/** Cuantifica un componente de alterna */
private double quantizeAC(double value, int curCoef)
{
 if(imageType == IMAGE_YCrCb && currentBand != AXIS_Y)
 { // Si estoy codificando en YCrCb las cromas las
   // codifico más grueso
    return (value / AC_QUANTIZER_CrCb)*quality *
            Math.pow(AC_QUANTIZER_FACTOR, curCoef);
 }
 else
 {
     return (value / AC_QUANTIZER)*quality *
             Math.pow(AC_QUANTIZER_FACTOR, curCoef);
 }
}
```

Figura 1

Una vez cuantizados los coeficientes previo a la codificación se procede a escalarlos para poder trabajar con enteros. Luego se procede a la codificación de los coeficientes para ser almacenados en el archivo de salida. Para la codificación se utilizaron las técnicas descriptas en el estándar JPEG. Para los coeficientes de DC se utliza como predictor el coeficiente de DC del bloque anterior, por lo cual sólo se codifica la diferencia. La codificación de la diferencia se realiza a través de VLI(Variable Length Integres). La misma implica codificar mediante Huffman la cantidad de bits que son necesarios para expresar el entero codificado como VLI, seguido del propio entero. La tabla de la figura 2 describe la cantidad de bits y el rango asociado al VLI con esa cantidad de bits. Los coeficientes de AC se codifican mediante Zero Run Length Encoding. Es decir que se determina la cantidad de coeficientes con valor 0 y para el primer coeficiente distinto de 0 se aplica una codificación similar a la utilizada para el coeficiente de continua. En este caso se utiliza un

Tabla de correspondencia VLI	
Bits	Rango
1	-1, 1
2	-3, -2, 2, 3
3	-7, ..., -4, 4, ..., 7
4	-15, ..., -8, 8, ..., 15
5	-31, ..., -16, 16, ..., 31
6	-63, ..., -32, 32, ..., 63
7	-127, ..., -64, 64, ..., 127
8	-255, ..., -128, 128, ..., 255

Figura 2

codificador de Huffman para indicar la cantidad de ceros seguidos junto con la cantidad de bits necesarias para codificar como VLI el coeficiente de alterna (aquí no se utiliza el predictor como en el caso de DC). Cuando todos los coeficientes de alterna restantes del bloque que estuviera siendo procesado son cero, se indica mediante un símbolo especial EOB (EndOfBlock) y se procede con el siguiente bloque.

Para la implementación del codificador se utilizaron 4 codificadores de Huffman. El primero de ellos va formando una tabla de Huffman a partir de los datos asociados a los coeficientes de continua de cada banda de la imagen. Los 3 restantes codificadores van formando cada uno una tabla de Huffman asociada a los valores de Zero Run Length Encoding y a los coeficientes de alterna de cada una de las tres bandas (salvo en el caso de imágenes en tonos de gris, donde solamente se utiliza una de estas tres bandas). Dado que la distribución de probabilidades de cada uno de los símbolos no se conoce hasta terminar de procesar la imagen, la codificación real se retrasa hasta haber completado el procesamiento de toda la imagen. Luego a partir de los datos recolectados y de la distribución de probabilidades se puede realizar la codificación mediante Huffman.

La tercer y última fase del codificador es embeber los coeficientes de la base de la KLT en el archivo de salida. Esta debe realizarse después de codificar toda la banda porque es recién en este momento cuando se conoce con exactitud la cantidad de coeficientes que se han utilizado. Esta última fase es opcional, porque el codificador da la posibilidad de no embeber los coeficientes dentro del archivo de salida, sino generar un archivo separado. De esta manera varias imágenes pueden utilizar el mismo archivo de coeficientes de la base a la hora de codificar/decodificar. En el caso en que los coeficientes de la base estén embebidos en la imagen de salida, se aplican algunas optimizaciones para reducir el tamaño que ocupan dentro del archivo.

La figura 3 muestra los argumentos que permite recibir el codificador. Cabe destacar los primero 3 parametros. El primero (-bN) indica en N, el tamaño del bloque de NxN utilizado en la KLT. Aquí hay un compromiso entre el valor de N y los recursos del sistema ya que cuando más grande sea este número la cantidad de operaciones involucradas para diagonalizar la matriz de covarianza aumenta considerablemente. Los valores usuales son bloques de 8x8 hasta 16x16. Otro de los argumentos (-cN) indica la cantidad de coeficientes de la base a considerar en la codificación y permite controlar el tamaño/calidad del archivo de salida. El parámetro –qN permite indicar el cuantificador de calidad para los coeficientes de alterna (valores de N=1 indican la peor calidad y de N=100 la mejor calidad). Además existen otros argumentos que permiten especificar el espacio de color en el cual trabajar (RGB, YCrCb, o simplemente escala de grises), o argumentos que indican si calcular la base de la KLT o obtenerla de un archivo externo.

La figura 4 muestra un ejemplo de codificación de una imagen (peppers.ppm) en el espacio YCrCb y la codificación de cada canal por separado. Allí se puede ver como las cromas (Cr y Cb) estan subsampleadas y ocupan un cuarto del tamaño del canal de luminancia (en la figura se muestra el canal de luminancia y la imagen original escaladas a un cuarto para que entren en la figura). En esta imagen de ejemplo, de un tamaño original de 768Kb se produjo un archivo de salida de 50Kb utilizando como parámetros una calidad de 90 y como máximo la utilización de 24 coeficientes de la base.

Parámetros del codificador

```
EncodeKLT v1.0, (u)2004, Gustavo Brown Rodriguez

USO: com.brownsoft.klt.EncodeKLT [Argumentos] Archivo .pgm/.ppm/.jpg/.gif

Argumentos:
        -bN Tamaño de bloque NxN (4-64), por defecto=8
        -cN Cantidad de coeficientes a considerar, por defecto=2*tamaño bloque
        -qN Calidad (1-100)
        -j[Nombre] Guardar base de la klt [en el archivo indicado]
        -kNombre Codificar usando la base de la klt a partir del archivo .kltc
        -y Codificar imagen color en espacio YCrCb (valor por defecto)
        -r Codificar imagen color en espacio RGB
        -g Codificar imagen color en escala de grises
        -oNombre Nombre del archivo de salida
        -m"Msg" Comentario asociado a la imagen
        -n Mostrar codificaci¾n de cada canal
        -s Mostrar imagen de entrada
        -p Mostrar progreso de la codificacion
```

Figura 3

Figura 4

Decodificador KLT

La decodificación de una imagen previamente implica un proceso bastante más fácil y liviano con respecto a los recursos que el que se incurre en la codificación.
El decodificador inicialmente levanta del archivo de la imagen (o de un archivo de coeficientes de base externo) la base de la KLT. Luego itera sobre cada una de las bandas de la imagen codificada y va aplicando la transformación inversa de la KLT para obtener, a partir de los vectores transformados, un conjunto de vectores antitransformados. Previo a la aplicación de la antitransformada se deben decuantizar los coeficientes de continua y alterna de cada uno de los bloques, es decir, se aplica el proceso inverso al ejecutado al momento de codificar la imagen.

Una vez que toda la banda fue decodificada se procede al reordenamiento de los vectores para ir reconstruyendo la imagen. En el caso de imágenes codificadas en el espacio de colores YCrCb hay que realizar un upscaling de las cromas para volver a obtener el tamaño original en dichas bandas. Adicionalmente en este caso también se debe volver a transformar la imagen del espacio YCrCb a RGB para su visualización en pantalla.

4. Resultados codificador KLT

En esta sección se presentan algunos resultados obtenidos al aplicar el codificador KLT a un conjunto de imágenes de prueba y se realizan algunas comparaciones de resultados contra un codificador Jpeg. Las imágenes de prueba pueden descargarse de [wwwCod]. La tabla de la figura 5 describe sus características.

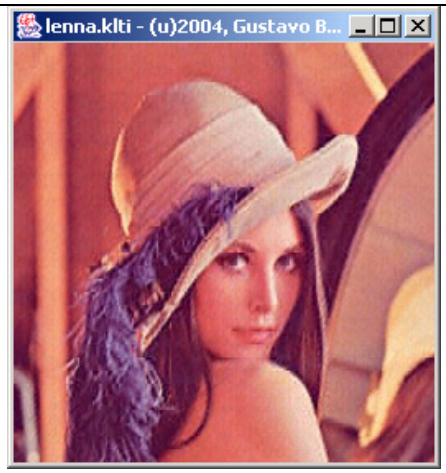
Nombre	Tipo	Tamaño	KBs
baboon.pgm	grises	512x512	256
baboon.ppm	color	512x512	768
house.pgm	grises	256x256	64
house.ppm	color	256x256	192
btfrag.pgm	grises	256x256	64
camera.pgm	grises	256x256	64
lenna.ppm	color	256x256	192
peppers.ppm	color	512x512	768
splash.ppm	color	512x512	768
tulips.ppm	color	768x512	1152

Figura 5

En el primer set de experimentaciones se procedió a codificar las imágenes utilizando distintos valores de calidad, para determinar el impacto del parámetro sobre la compresión de la imagen y sobre los efectos de la compresión con pérdida en la visualización. La tabla de la figura 6 muestra los resultados obtenidos en la compresión de la imagen de prueba lenna.ppm utilizando el espacio de color YCrCb para distintos factores de calidad. En este set de pruebas se consideran en la codificación los primeros 16 coeficientes de la base. La base de la KLT queda incluída en el archivo de salida. Como puede apreciarse muy fácilmente, al ir disminuyendo el factor de calidad y por ende la cantidad de bits por pixels en la imagen codificada, se va viendo una disminución de la calidad en la reconstrucción de la imagen por parte del decodificador. Para una calidad de 60 ya se comienzan a notar los efectos del blocking en la imagen. Con una factor de calidad de 25 en la imagen se distinguen perfectamente los bloques 8x8 utilizados para la codificación.
El siguiente set de pruebas consistió en replicar nuevamente los mismos factores de calidad al set de imágenes pero utilizando un corte de 32 coeficientes de la base de la KLT. En la figura 7 se muestran los resultados obtenidos para la imagen lenna.ppm.

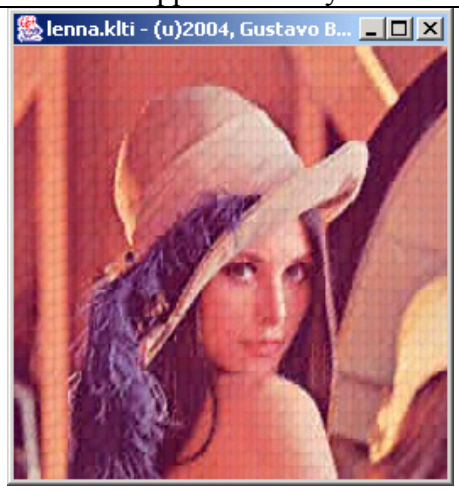
Aquí lo que se puede apreciar es una mayor nitidez de la imágenes. Sin embargo para factores de calidad bajos, el blocking sigue persistiendo.

Imagen original
24bpp – 196623 bytes



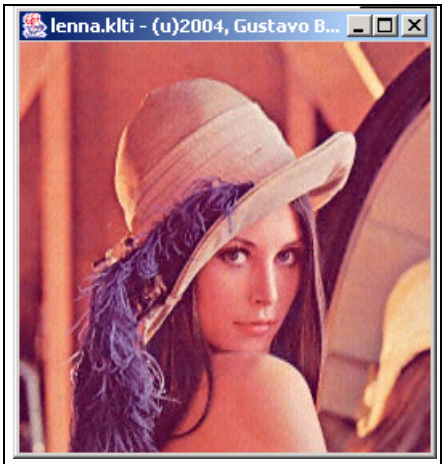
Calidad: 100 – Coefs: 16
2.34bpp – 19185 bytes

Calidad: 60 – Coefs: 16
1.95bpp – 16018 bytes

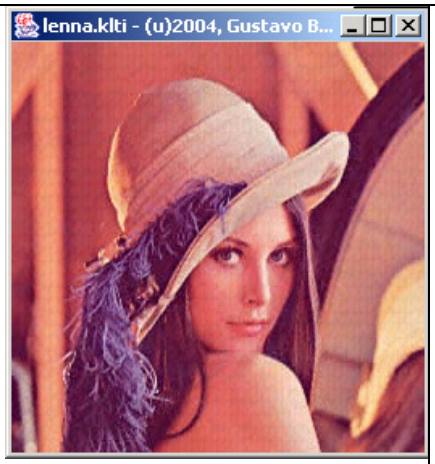


Calidad: 25 – Coefs: 16
1.45bpp – 11847 bytes

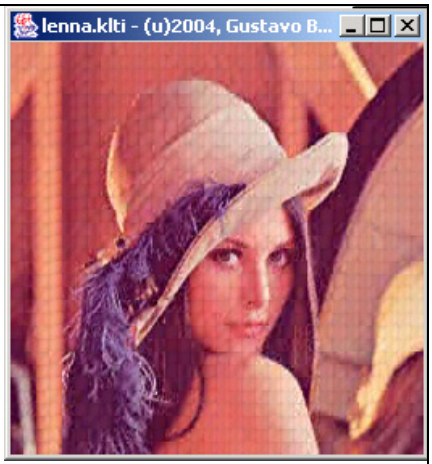
Figura 6



Calidad: 100 – Coefs: 32
3.54bpp – 29016 bytes



Calidad: 60 – Coefs: 16
2.86bpp – 23424 bytes



Calidad: 25 – Coefs: 16
1.81bpp – 14812 bytes

Figura 7

En una segunda experimentación se trabajó con bloques de 16x16 para ver el efecto que tenían sobre las imágenes. Aquí el peso de embeber los coeficientes en la base puede ser grande en relación a la propia imagen, por lo que se optó por codificar el set embebiendo la base de la KLT y sin embeber. Nuevamente se muestran los resultados obtenidos en la figura 8 solamente para la imagen lenna.ppm, aunque ahora se indica el tamaño del archivo con y sin la base de la KLT embebida.

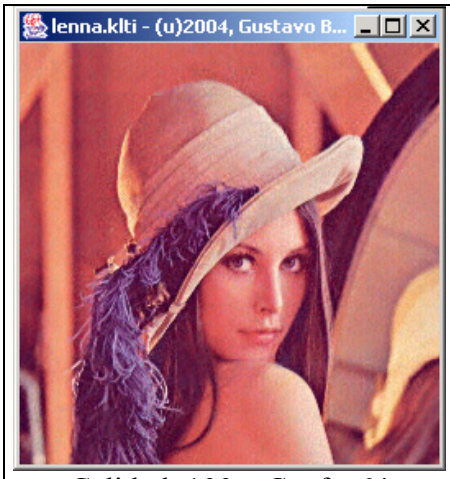
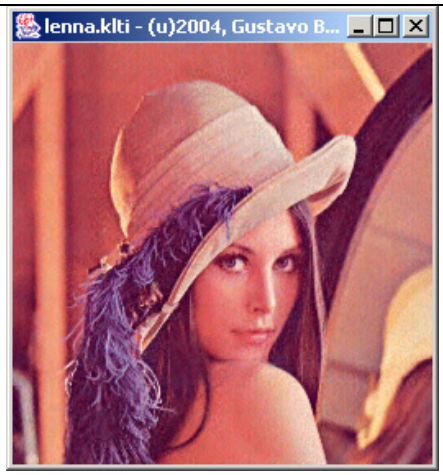
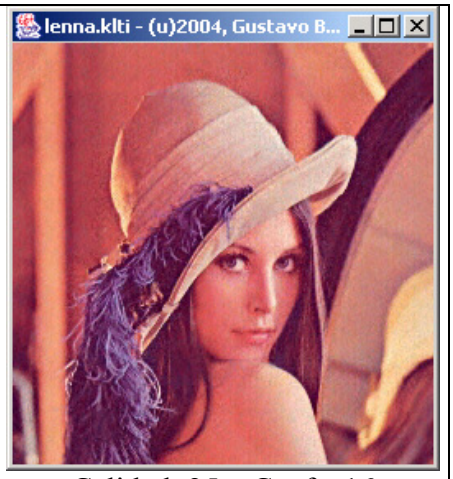
		
Calidad: 100 – Coefs: 64 No emb.: 2.77bpp – 22697 bytes Embeb:12.02bpp – 98730 bytes	Calidad: 60 – Coefs: 64 No emb.: 2.45bpp – 20090 bytes Embeb:11.73bpp – 96123 bytes	Calidad: 25 – Coefs: 16 No emb.: 2.45bpp – 15285 bytes Embeb:11.14bpp – 91317 bytes

Figura 8

En este nuevo set de imágenes podemos ver que no se distinguen los efectos de blocking pero a costa de un archivo más grande. Si no embebemos los coeficientes de la KLT cada una de las imágenes ocupa entre 15 y 20Kb, pero al embeber los coeficientes de la base utilizados vemos que el tamaño se dispara a cercad e 90Kb.

Hasta ahora las imágenes codificadas fueron tratadas en el espacio YCrCb, con la cual aprovechando las características psicovisuales de los hombres se puede comprimir aún más las cromas subsampleando dichas bandas y cuantificándolas más grueso. En el proximo set de experimentaciones se muestra los resultados de aplicar la codificación a la imagen house.ppm utilizando los espacios YCrCb, RGB y una codificación en escalas de grises. De los resultados obtenidos podemos ver en la figura 9 que la codificación en el espacio YCrCb es mucho más eficiente que la codificación en el espacio RGB. La calidad de ambas imágenes a la vista del usuario es prácticamente idéntica, pero gracias al submuestreo y a la cuantificación más gruesa de las cromas en el espacio YCrCb, se obtuvo un ahorro de casi un 40% con respecto a RGB. Por supuesto que la codificación en escala de grises lleva a una imagen de menor tamaño, pero en ese caso solamente se codifica una banda con lo cual es lógico que el tamaño sea mucho menor.

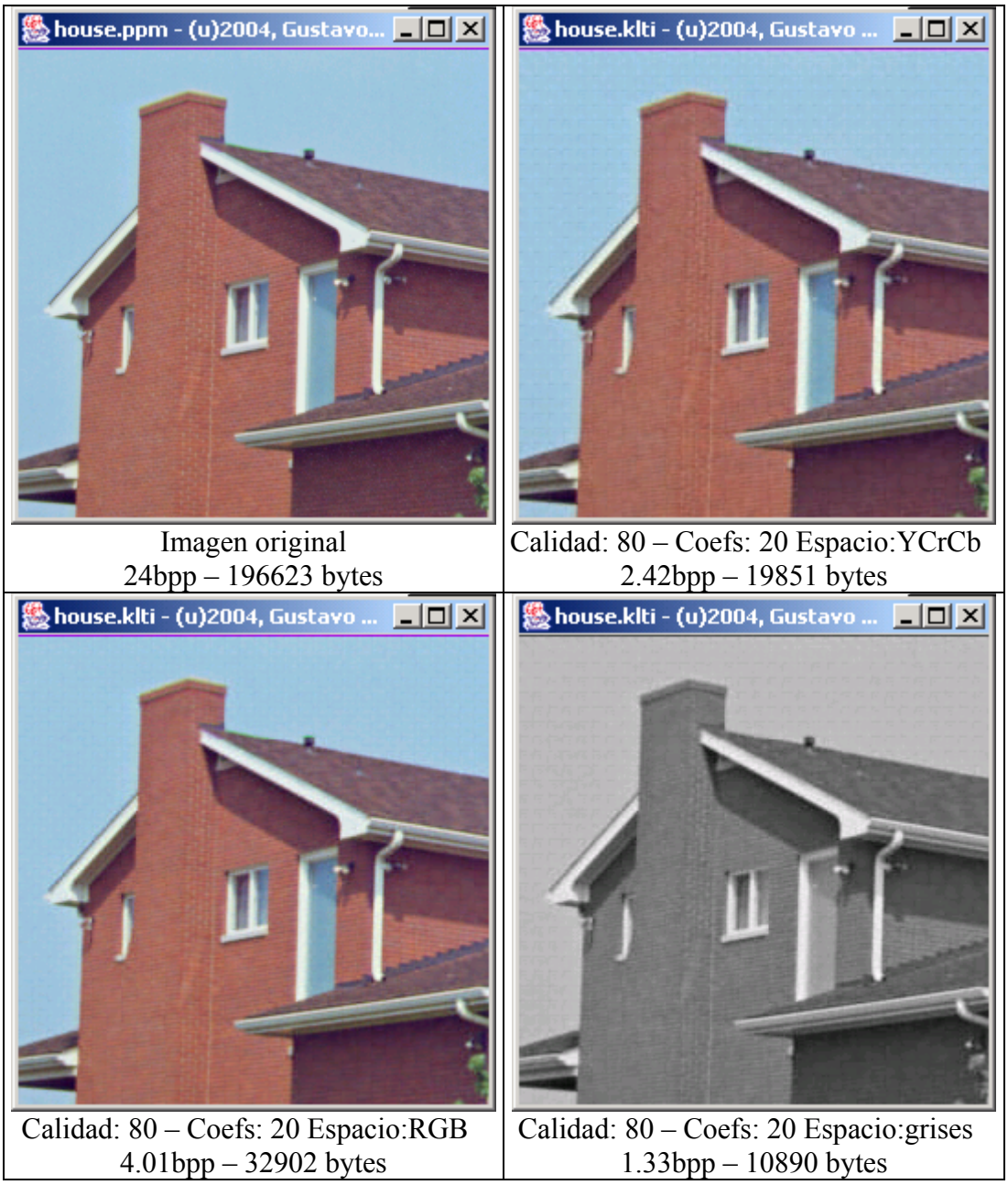


Imagen original
24bpp – 196623 bytes

Calidad: 80 – Coefs: 20 Espacio:YCrCb
2.42bpp – 19851 bytes

Calidad: 80 – Coefs: 20 Espacio:RGB
4.01bpp – 32902 bytes

Calidad: 80 – Coefs: 20 Espacio:grises
1.33bpp – 10890 bytes

Figura 9

Luego de haber realizado varias experimentaciones variando distintos parámetros del codificador, se procedió a realizar comparaciones de performance contra un codificador Jpeg. Para la comparación se tomo algunas imágenes del set de prueba.

Se comparo la calidad de las imágenes al comprimirlas hasta un mismo tamaño dado. La figura 10 muestra los resultados obtenidos para las imágenes baboon.ppm y lenna.ppm. Para el caso de la imagen baboon se codificó la imagen Jpeg hasta con un parámetro de calidad de 68 lo cual llevo a tener un archivo de 86.417 bytes. Luego variando los parámetros de calidad y cantidad de coeficientes se codifico la misma imagen con el codificador KLT. Para una calidad de 90 y 24 coeficientes se llego a tener una imagen de 85.235 bytes. Comparando ambas imágenes no se pueden distinguir diferencias de calidad entre ambas.

En cambio, al comprimir mediante Jpeg la imagen lenna.ppm con valores de calidad muy pequeños (factor de calidad 15), la calidad de la imagen sigue siendo aceptable ocupando 8226 bytes, mientras que con el codificador KLT aún sin embeber los coeficientes de la KLT en la imagen se puede ver una degradación de la calidad en comparación con la vista con Jpeg.

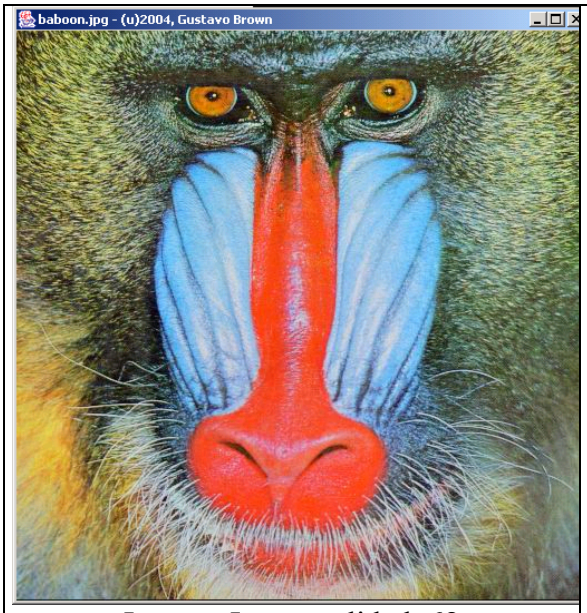
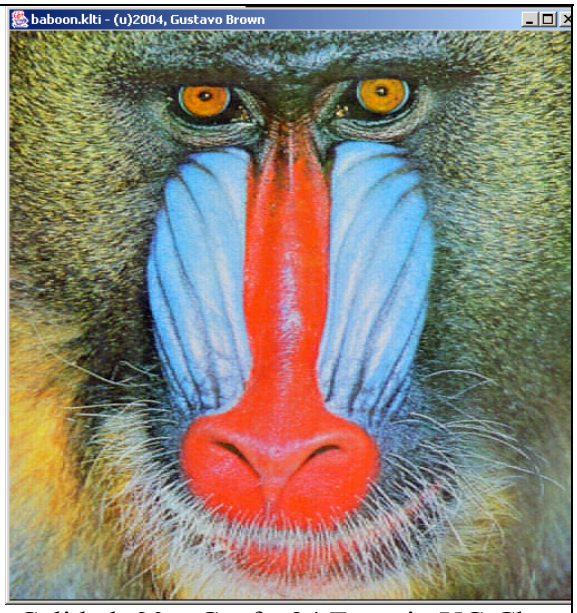
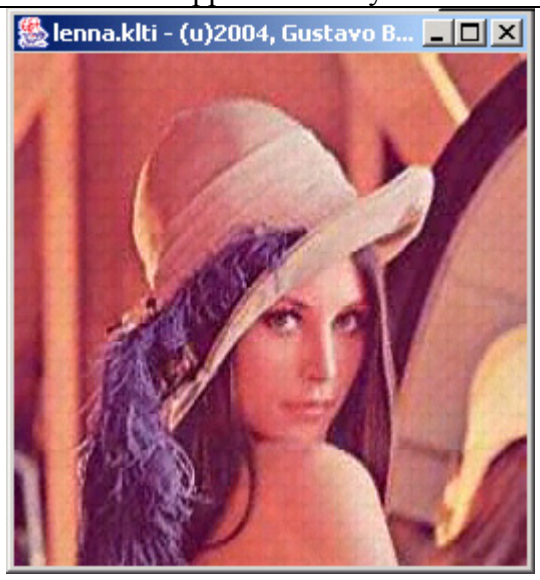


Imagen Jpeg – calidad: 68
86.417 bytes



Calidad: 90 – Coefs: 24 Espacio:YCrCb
2.60bpp – 85.237 bytes

Imagen Jpeg – calidad: 15
8.226 bytes



Calidad: 45 – Coefs: 20 Espacio:YCrCb
Coefs sin embeber: 8.579 bytes
Coefs embebidos: 19.320 bytes

Figura 10

De los resultados obtenidos en estas y otras imágenes se puede ver que cuando las imágenes son suficientemente grandes el codificador KLT tiene un desempeño superior a Jpeg, mientras que para imágenes pequeñas o con factores de calidad muy chicos, el overhead de tener que embeber los coeficientes de la base hacen que Jpeg tenga un rendimiento superior.

Otro punto a favor para Jpeg es que el orden de complejidad de los algoritmos involucrados es menor. Los tiempos de ejecución y la cantidad de recursos que el codificador KLT necesita son muy superiores a los que se necesitan para codificar una imagen utilizando la DCT como transformada.

5. Conclusiones y trabajos a futuro

El proyecto permitió experimentar con varias técnicas asociadas a compresión de imágenes. Por un lado se implementaron los algoritmos que aplican la transformación de Karhunen-Loève sobre varios tipos de imágenes (en escalas de grises, a colores en el espacio RGB y a colores en el espacio YCrCb). Para las imágenes a colores en el espacio YCrCb se optimiza por dos vías la codificación, por un lado submuestreando los canales de las cromas y por otro cuantificando más grueso los coeficientes de alterna de dichos canales. A la hora de codificar, la utilización de codificadores de Huffman, VLI y Zero RLE permiten obtener grandes ratios de compresión incluso con calidades de imagen de salida muy buenas.

El desarrollo del codificador implico la implementación codificadores VLI y de multiplexores de codificadores de Huffman, de tal manera que no sea necesario recorrer dos veces una imagen para poder codificarla, dado que el diccionario de Huffman recién se conoce luego de terminar la aplicación de la transformación sobre la imagen.
De las pruebas realizadas sobre el codificador puede deducirse que tiene un desempeño aceptable sobre imágenes grandes, en las cuales el tamaño de la base de la KLT no es lo suficientemente grande como para pesar demasiado en el archivo de salida. Otra opción es utilizar un archivo externo como base de la KLT, aunque esto solamente es útil si la distribución de probabilidades es similar en un conjunto suficientemente grande de imágenes; de esta manera se puede dividir el peso de la base de la KLT entre la codificación de todas las imágenes del conjunto.

Como trabajo a futuro más importante esta rever el algoritmo de cuantificación de coeficientes de continua y alterna. Estos algoritmos tienen gran incidencia sobre la performance del codificador en la compresión de imágenes. Un ajuste en los parámetros del algoritmo actual pueden llevar a mejoras en este sentido, aunque probablemente lo mejor se implementar un nuevo algoritmo.

Otro posible trabajo a futuro es investigar la interdependencia de los coeficientes de alterna entre bloques. Actualmente el codificador utiliza, al igual que el estándar Jpeg, un predictor para los coeficientes de alterna. Se podría investigar sobre la eficacia de aplicar un predictor a los primeros coeficientes de alterna. El código fuente comentado del codificador se encuentra disponible en [Gusbro04].

Referencias

- [Dony01] R. D. Dony: "Karhunen-Loève Transform", *The Transform and Data Compression Handbook,* CRC Press, 2001
- [DCIM03] Docentes curso Codificación de Imágenes y Video: "Diapositivas powerpoint del curso", 2003
- [Jama00] Bruce Miller: 'Java Math Package', http://www.jama.org/index.html
- [Gusbro04] Pagina web del proyecto, http://www.fing.edu.uy/~gbrown/klt/index.html
- [wwwCod] Pagina web del curso Codificación de Imágenes y Video, http://iie.fing.edu.uy/ense/asign/codif/