Álvaro Ortega

Tecnologías Multimedia

PROYECTO FINAL

MEMORIA

1. CONCEPTOS TEÓRICOS

Compresión temporal: compara la información de cuadros (frames) sucesivos para encontrar similitudes que se puedan expresar de forma más eficiente. Precisamente por este hecho, la compresión temporal también se conoce con el nombre de **codificación intercuadro**, ya que opera con una secuencia de imágenes en movimiento.
Este tipo de compresión se utiliza habitualmente en difusión, y es eficiente para la transmisión de video, pero su problemática radica en la edición de imágenes, puesto que no existe una descripción completa de cada uno de los frames que componen la secuencia.
La compresión temporal funciona con paquetes de frames, llamados GOP.
Se utiliza la información contenida en la tesela equivalente de la imagen previa (frame P o I) más próximo para realizar la codificación. La codificación intercuadro, es, por tanto, un método predictivo, en el que se intenta predecir para la siguiente imagen los píxeles que habrá en el marco, con tal de poder eliminar zonas de imágenes que pudieran ser reconstruidas a partir de otras.

GOP (Group Of Pictures): el GOP es la estructura que especifica cómo se juntan las imágenes en el método interframe. En una codificación de imágenes, se agrupan las diferentes imágenes en paquetes del tamaño del GOP. Con estos paquetes de imágenes se comparan las imágenes equivalentes para crear la codificación. El GOP contiene los siguientes elementos:

- **Frame o Imagen I:** imagen codificada de forma independiente al resto de imágenes. Cada GOP empieza con este tipo de imagen, a partir de la cual se generará la codificación del resto de imágenes del GOP.
- **Frame o Imagen P:** cada una de las imágenes generadas a partir de la codificación de la diferencia de compensación de movimiento relativa a las imágenes previamente decodificadas.

Compensación de movimiento: en los videos, lo normal es que diferentes zonas de la imagen se desplacen entre imagen e imagen, resultado del movimiento de los objetos de video. La técnica de compensación de movimiento permite encontrar, a partir del parámetro de seek range, teselas equivalentes entre diferentes imágenes del GOP, de tal forma que se puedan eliminar las repetidas en imágenes sucesivas, y, guardando previamente las sustituciones y eliminaciones, recomponer la imagen cuando se decodifica.

2. IMPLEMENTACIÓN DETALLADA

En esta sección de la memoria se comentan los algoritmos implementados y su funcionamiento. No he incluido código fuente ni especificación de los métodos más allá de las cabeceras, ya que ambas se pueden encontrar en el código y en el javadoc respectivamente, con lo cual resulta redundante. Sí incluyo las cabeceras de los métodos para que resulte más sencillo seguir la ejecución del código.

Codificación (Clase Codificador.java):
Pasos seguidos:

1. Rellenar el GOP con imágenes. (*ompleGOp()*).
2. Recorrer el GOP para buscar las teselas candidatas y realizar la sustitución. (*recorreGOP()*).
    a. Subdividir la imagen en teselas. (*subdividirImgTesseles(BufferedImage)*).
    b. Buscar teselas iguales (coincidencias). (*buscarTesselesIguals(Marc iFrame, BufferedImage pFrame)*).
        i. Calcular el ratio PSNR. (*calcularRatioPSNR(Tesseles tesela, BufferedImage pframe)).*
        ii. En caso que el ratio coincida, generar la media de color. (*medianaColor(BufferedImage im)*).
        iii. Aplicar el color. (*setColorPFrames(ArrayList<Tesseles> teseles, BufferedImage pFrame)*).
    c. Generar un fichero con las coordenadas de las teselas cambiadas.
3. Guardar las imágenes generadas por la codificación y generar un ZIP.

Decodificación (Clase Decodificador.java):
Pasos seguidos:

1. Leer el ZIP codificado y sacar las imágenes y el fichero de codificación (*readZip())*.
2. Guardar los datos del fichero de codificación en una array de parámetros. (*readZIP()*).
3. Iterar las imágenes descomprimidas y reconstruirlas a partir de la array. (*IterateImages()*).
    a. Generar las teselas para el I frame de cada GOP. (*genarateMacroBlocs())*
    b. A partir de las coordenadas de la array, encontrar para los sucesivos P frames las teselas eliminadas y aplicar la media de color. (*buildPFrames()*).

Conversión de otros formatos a JPEG (Clase JPEGCompress.java):
Pasos seguidos (*convertToJPG(String inputPath, String outputPath)*):

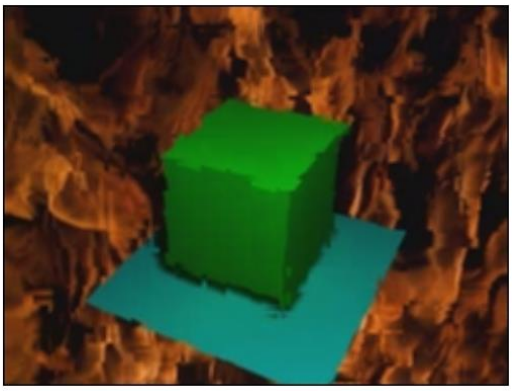
1. Comprobar la extensión de la imagen para encontrar aquellas que no sean JPEG, y pasar por parámetro la ruta de la imagen.
2. Crear un file Input Stream para leer la imagen pasada por parámetro.
3. Crear un file Output Stream para generar la nueva imagen.
4. Aplicar el método write de la clase built in de java ImageIO para generar una nueva imagen con extensión JPEG a partir de la información de la imagen inicial almacenada en el file Input Stream.
5. Guardar la imagen en la ruta de destino parametrizada.

3. RESULTADOS

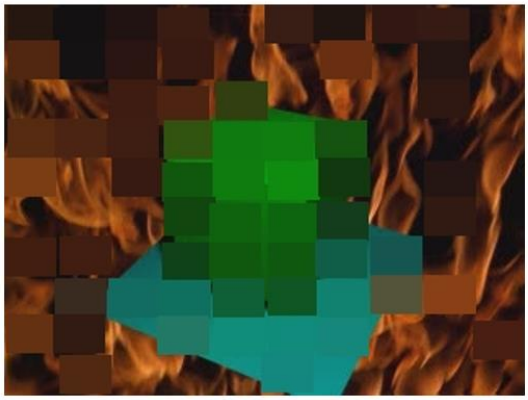
- RESULTAT 1
  - Comanda: -i "material/Cubo.zip" -e -d -o "comprimido.zip" --fps 4 --GOP 4 --nTiles 20 --seekRange 3 --quality 22
  - Paràmetres: GOP: 4 nTiles: 20 quality: 22 seekRange: 3
  - Encode: 30213 milisegons
  - Decode: 36778 milisegons
  - Zip inicial: 10344228 Comprimit: 4275311
  - El rati de compresió és el següent: 2.4195263

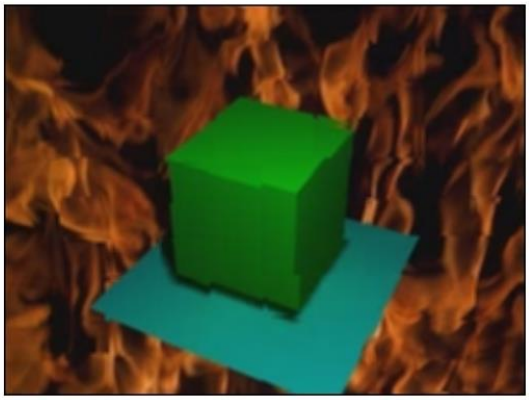
Codificada

Decodificada



- RESULTAT 2
  - Comanda: -i "material/Cubo.zip" -e -d -o "comprimido.zip" --fps 4 --GOP 10 --nTiles 10 --seekRange 2 --quality 10
  - Paràmetres: GOP: 10 nTiles: 10 quality: 20 seekRange: 2
  - Encode: 17735 milisegons
  - Decode: 41935 milisegons
  - Zip inicial: 10344228 Comprimit: 3558599
  - El rati de compresió és el següent: 2.906826





4. CONCLUSIONES Y REFLEXIONES

La primera conclusión que salta a la vista es que la calidad de la descompresión no es la mejor, incluso con parámetros relativamente restrictivos. Discutiendo con el compañero, creemos que parte de la problemática viene de cómo decidimos implementar la selección de teselas y del seek range. En nuestro algoritmo, el seek range busca tesela a tesela, y no por píxeles, con lo cual eso ya afecta la calidad. Además, al calcular la media de color de las teselas del I frame y cambiarlas por las de los P frames, también perdemos cierto nivel de calidad.

También es digno de mención el hecho de que nos tarde más el decodificador que el codificador. Por las pruebas realizadas con el debugger, he llegado a la conclusión de que lo que hace que el debugger tarde tanto es el leer el archivo de coordenadas, ya que lo lee de forma lineal, lo que aumenta de forma muy notoria el tiempo.

Podría comentar las reflexiones vistas en clase sobre que aumentar las teselas aumenta la calidad, o que reducir el GOP también permite mayor calidad ya que las imágenes no se deforman tanto, pero todas estas conclusiones ya se han visto en clase. Sí que cabría comentar que todo lo comentado en clase se ha probado y ha dado los resultados esperados, relación inversa entre calidad y compresión cuando se aumenta o disminuye el GOP, y el número de teselas.

Finalmente, añadiendo este punto tras la presentación que realizamos en la clase de teoría, es cierto que los resultados acerca de factor de compresión no son correctos, ya que en el ZIP de imágenes que se ha usado como modelo (cubo.zip) las imágenes se encuentran en PNG, mientras que en el comprimido con codificación se encuentran en JPEG.