Compresión de vídeo con avconv

Análisis sistemático de rendimiento

Juan Marín Vega Fernando Moral Algaba

17 de noviembre de 2016

1. La notación DEVIL

El comando -codecs nos muestra la lista de *decoders* y/o *encoders*, de audio vídeo y subtítulos, incluida en la librería libavconv. En la lista se observa que en algunas implementaciones solo esta disponible la función de codificar, o por el contrario sólo son capaces de descodificar. La notación DE[VAS]ILS nos indica las capacidades y/o características de las implementaciones.

- **D** Indica que que la implementación en concreto es capaz de descodificar.
- **E** Indica que la implementación codifica.
- **V** Indica que es un códec de vídeo.
- **A** Indica que es un códec de audio
- **S** Indica que es un códec de subtítulos.
- **I** Indica que tan solo trabaja a nivel intra-frame.
- **L** Indica que es una codificación con pérdida (Lossy).
- **S** Indica que es una codificación sin pérdida (Lossless).

2. Soporte en el sistema para mjpeg...

- **mjpeg** o Motion jpeg. Es **DEVIL.**. Incluye soporte para decodificar y codificar. Además se indica que es un códec de vídeo, que sólo trabaja a nivel intra-frame y codifica con pérdida.
- **mpeg1video** o MPEG-1 video. Es **DEV.L.**. Incluye soporte para decodificar y codificar. Es un códec de vídeo, tiene perdida y trabaja a nivel inter e intra cuadro.

- **mpeg4** o MPEG-4 part2. Es **DEV.L.**. Incluye soporte para decodificar y codificar. Es un códec de vídeo, tiene perdida y trabaja a nivel inter e intra cuadro.
- **h264** o H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10. Es **DEV.LS**. Incluye soporte para decodificar y codificar. Es un códec de vídeo que permite codificar con o sin perdida, y también trabaja a nivel inter e intra cuadro.

3. Comparación de los códecs de forma cuantitativa y cualitativa

	Tiempo	**Memoria cpu**	**Tamaño fichero**	**Calidad visual**
mjpeg	1º 229 ms	2º 17 MB	4º 452 KB	2º
mpeg1video	2º 356 ms	3º 18 MB	2º 229 KB	4º
mpeg4	3º 382 ms	1º 16 MB	1º 194 KB	3º
h264	4º 1.7 s	4º 51 MB	3º 409 KB	1º

Cuadro 1: Tabla de códecs

Se han realizado las pruebas en dos sistemas distintos con cpu AMD e INTEL hallando diferentes valores en el empleo de memoria. Los datos de la cuadro precedente corresponden a cpu INTEL y el uso de memoria es inferior al caso de la cpu AMD.

4. Comparación de ratios de compresión respecto mjpeg

Comparamos con mjpeg por que es el único de los 4 que sólo trabaja a nivel intra-frame.

	Ratio
mpeg4	1:2.32
mpeg1video	1:1.97
h264	1:1.1
mjpeg	1:1

Cuadro 2: Comparativa de ratios de compresión respecto a mjpeg

5. Modificación del GOP

La Figura 1 muestra la evolución del ratio con los diferentes valores de GOP. La Figura 2 muestra la evolución del número de frames del tipo I y P en función del GOP.

Vemos que la suma de estos dos tipos de frame siempre da 100, que en este caso es el numero de imágenes que conforman el vídeo original.

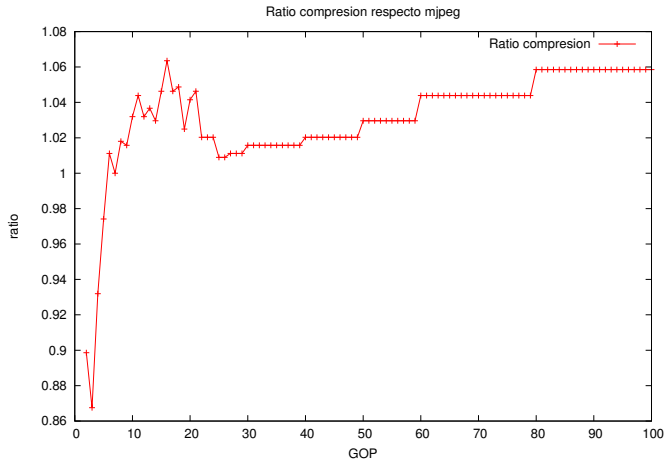


Figura 1: Evolución del ratio de compresión con distintos valores de GOP

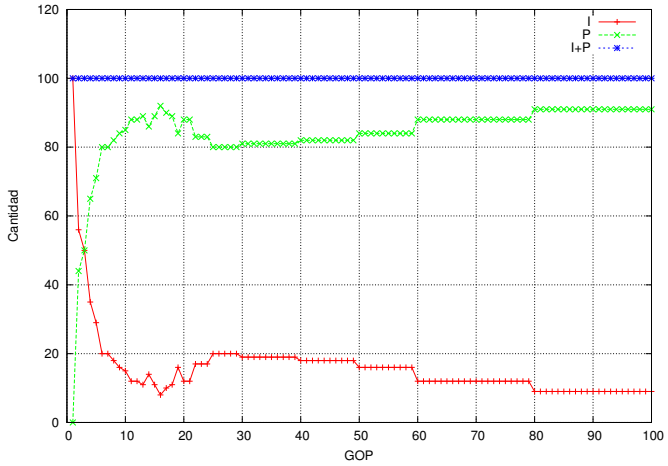


Figura 2: Evolución de numero de frames I, P

Como muestra la Figura 1, cuanto mayor es el GOP, menor es el numero de frames I y menor es el tamaño final del archivo. Esto sucede por que los frames I son los que estan comprimidos de forma intra-cuadro, mientras que los frames P son los que estan comprimidos utilizando el método de predicción de movimiento. Por tanto, cuando usamos GOP 1, el resultado es muy similar al obtenido con **MJPEG**.

6. Modificación del framerate de salida

En la Figura 3 observamos que a mayor framerate, mayor es el tamaño del archivo de salida, debido a que el archivo de salida contiene mas información. Aumentar el framerate de salida implica guardar un mayor número de cuadros por segundo lo cual necesariamente ocupara más espacio.

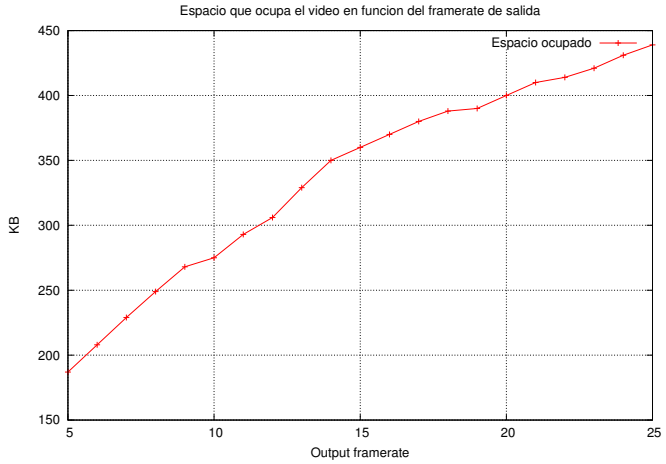


Figura 3: Tamaño del archivo generado en función del framerate de salida

7. Comparación de métodos de comparación de teselas

- **SAD** *Sum of absolute differences*: En este método se comparan las teselas pixel a pixel, haciendo la suma de las diferencias absolutas:

$$SAD = \sum_{i=0}^{n-1} \left| t_a[i,i] - t_b[i,i] \right| \tag{1}$$

- **SATD** *Sum of absolute transformed differences*: Este método se basa en el SAD pero en lugar de usar los valores originales de los píxeles de la tesela, se hace usando la transformada de Hadamard.

- **SSE** *Sum of squared errors*: Este método es similar al SAD pero en lugar de sumar la diferencia absoluta, suma la diferencia al cuadrado.

$$SSE = \sum_{i=0}^{n-1} (t_a[i,i] - t_b[i,i])^2 \tag{2}$$

- **CHROMA** El método **chroma** tiene en cuenta la información de color para la detección de movimiento. Esto hace que el método resulte más preciso aunque más lento.

Según los resultados de la Cuadro 3, **SAD** parece ser uno de los métodos mas rápidos y que generan un archivo de menor tamaño. Para **SSE** suponemos que la operación de realizar el cuadrado de la diferencia practicamente no penaliza en el rendimiento. En el caso de **SATD** creemos que realizar la transformada de Hadamard si que debe resultar algo más costoso. Para el caso de **CHROMA** observamos que el tamaño final del fichero apenas varia pero el tiempo de proceso si aumenta ligeramente.

Observando el Cuadro 3 se aprecia que las diferencias son muy pequeñas, hecho que atribuimos a la pequeña duración del vídeo utilizado.

	Tiempo	**Tamaño fichero**
sad	1.136 s	439 KB
satd	1.146 s	445 KB
sse	1.125 s	445 KB
chroma	1.224 s	445 KB

Cuadro 3: Comparativa de los diferentes métodos de comparación de teselas.

8. Comparación de los algoritmos de búsqueda de desplazamiento

Los algoritmos de búsqueda de desplazamiento siguen un patrón de búsqueda determinado:

- En el caso del algoritmo **DIA** *Diamond*, se consultan las posiciones adyacentes horizontal y verticalmente, indicadas con **1**, en busca de una tesela con un valor de coincidencia mejor que **0**.

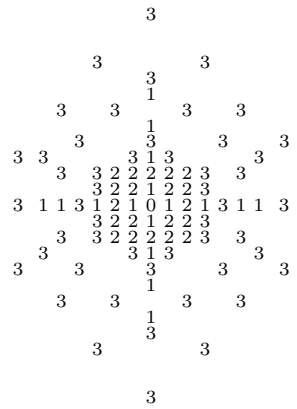
$$\begin{array}{ccc} & 1 & \\ 1 & 0 & 1 \\ & 1 & \end{array} \tag{3}$$

- En el caso del algoritmo **HEX** *Hexagon* se hace una primera iteración en los puntos indicados con **1** y si no hay una tesela mejor que **0** se pasa a una segunda iteración marcada con **2**.

$$\begin{array}{ccccccc} & & 1 & & 1 & & \\ & 2 & & 2 & & 2 & \\ 1 & & 2 & & 2 & & 1 \\ & 2 & & 2 & & 2 & \\ & & 1 & & 1 & & \end{array} \tag{4}$$

- Para el algoritmo **UMH** *Unheven Multy Hexagon* se sigue el patrón especificado,

en el que el parámetro *-me_range* define el tamaño del hexágono.



(5)

- En el caso del algoritmo **FULL** se recorren todos los posibles bloques.

(6)

	Tiempo	**Tamaño fichero**
dia	1.188 s	444 KB
hex	1.226 s	434 KB
umh	1.500 s	439 KB
full	1.772 s	434 KB

Cuadro 4: Comparativa de los diferentes algoritmos de búsqueda de desplazamiento.

9. Comparativa del rango de búsqueda de desplazamiento usando el método umh

Observando los resultados expuestos en la **figura 4** vemos que un valor de 10 es apropiado. El tamaño del archivo de salida es similar al obtenido con valores superiores y el tiempo de ejecución es razonablemente bajo.

10. Mejor conjunto de parámetros

Después de las pruebas realizadas decidimos usar los siguientes parámetros. Son bastantes exigentes pero teniendo en cuenta que es un video de reducido tamaño, los resultados obtenidos han sido bastante buenos.

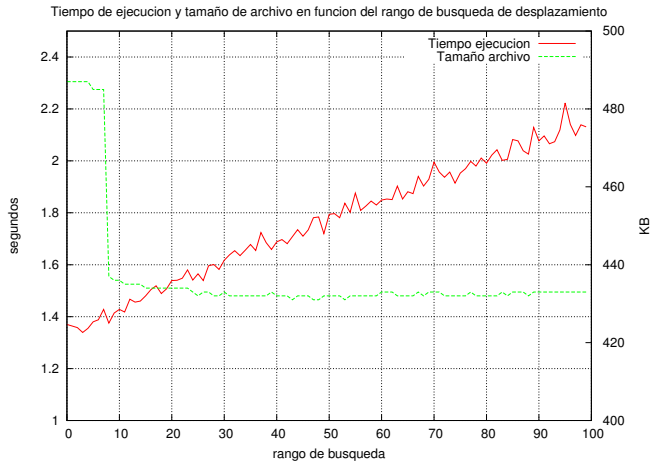


Figura 4: Tamaño del archivo y tiempo de procesado en función del rango de búsqueda

- GOP: *15*
- FPS: *25*
- Método comparación de teselas: *chroma*
- Algoritmo de búsqueda de desplazamiento: *umh*
- Rango de búsqueda de desplazamiento: *10*

Los resultados obtenidos son:

- Tiempo de ejecución: 2.144 segundos.
- Tamaño del archivo resultante: 421 KB.
- Memoria utilizada: 42648 KB.