Emulador HEVC INTRA en Matlab

Javier Ruiz Atencia, Otoniel López Granado, Manuel Pérez Malumbres,
Miguel O. Martínez-Rach[1]

Resumen— **Se ha diseñado un emulador HEVC Intra en Matlab con el objetivo de proporcionar a la comunidad científica y educativa una herramienta que implementa las diferentes etapas de codificación Intra del estándar HEVC y obtiene resultados mediante curvas rate-distortion y valores BD-Rate para distintas métricas objetivas de calidad. Las etapas de transformación, cuantización uniforme y perceptual, predicción Intra y reconstrucción han sido validadas con el software de referencia HM 14.0. Se utiliza un particionado en CUs de tamaño fijo. El desarrollo en Matlab permite probar modificaciones de diseño del HEVC relacionadas con estas etapas de forma más sencilla que utilizando el software de referencia. Se detalla su funcionalidad y se muestran resultados de la salida obtenida al codificar diferentes secuencias de vídeo.**

Palabras clave— **HEVC, emulador, transformación, cuantización, predicción Intra, Matlab**

I. Introducción

EN el proceso de investigación sobre posibles mejoras de los codificadores de video o imagen, los investigadores suelen trabajar con el software de referencia, para que finalmente el producto de su investigación sea compatible con el estándar para en el que se enfocan. La mayoría de los estándares recientes de codificación de video, H.264 o HEVC por ejemplo, se centran en definir las características que tiene que tener el bitstream para poder ser leído por un software compatible por el estándar.

De esta manera mucha de la investigación relacionada con mejoras en el HEVC se centra en poder modificar, mejorar u optimizar el codificador para que su resultado genere un bitstream compatible pero con mejoras en calidad, rendimiento, coste computacional, etc.

Como hemos dicho los investigadores suelen utilizar el software de referencia del codificador para incluir o probar sus ideas o mejoras. Trabajar sobre este software de referencia es a menudo costoso, puesto que la curva de aprendizaje del mismo y la complejidad del mismo son importantes.

Son pocos los casos en los que el investigador plantea una mejora en alguna de las etapas de codificación/decodificación teniendo la certeza de obtener los resultados esperados tras insertarlo en el estándar. Lo habitual es que se tenga que recurrir a distintas modificaciones iterando varias veces en un bucle de refinamiento de su teoría al observar los resultados obtenidos.

No olvidemos que las imágenes y videos se tratan como matrices de datos sobre los que el software opera para transformarlos, filtrarlos, reducirlos, codificarlos etc. y esta naturaleza matricial de la imagen y el video es ocultada por el lenguaje de programación utilizado en el software de referencia, habitualmente C++, con el uso de punteros y complejas y no siempre bien documentadas, estructuras de datos u objetos. Cuando la mejora que plantea el investigador se basa por ejemplo en procesar la matriz de datos para aplicar nuevos filtros o realizar operaciones entre distintas matrices es mucho más intuitivo aplicar un lenguaje de programación basado en matrices como Matlab.

Resultaría más práctico para el investigador que el bucle de codificación de la idea, ejecución, análisis de resultados, estudio y recodificación de la misma se haga directamente sobre matrices, por ejemplo cuando en sus propuestas se vean involucradas etapas donde la información es intrínsecamente matricial como las de transformación, predicción, cuantización y reconstrucción del HEVC entre otras.

Sin embargo no todas las líneas de investigación relativas al HEVC se beneficiarían de un HEVC en Matlab, por ejemplo aquellas cuyo objetivo fuese la mejora del coste computacional. Además hay que tener en cuenta que Matlab consume muchos recursos de la máquina y la velocidad de ejecución no es comparable a la obtenida por el software compilado en C++.

Pero para aquellos casos como los mencionados y aunque finalmente las propuestas deban ser implementadas en el software de referencia, una herramienta compatible con la ejecución del software de referencia del HEVC implementada en Matlab sería muy interesante. Cuando hablamos aquí de compatibilidad queremos decir que los resultados de cada una de las etapas implementadas son 100 % coincidentes con el resultado (bit a bit y valores enteros o flotantes) con aquellas etapas del software de referencia ante la misma entrada y la misma configuración, es decir, aplicando los mismos procesos. De esta forma se podría asegurar que los resultados obtenidos en Matlab por ejemplo en una variación que mejore de la etapa de predicción Intra, serían los mismos si se implementara esa mejora en el software de referencia. Con la ventaja de haber trabajado sin la interferencia mencionada del lenguaje de programación.

En la búsqueda de una herramienta con las características de la presentada hemos encontrado varias herramientas que proporcionan un análisis completo del bitstream del HEVC con muchas posibilidades y opciones de visualización, algunas de las cuales también de open-source GPL [1]. También existen distintas librerías, módulos e implementaciones del codificador HEVC con código open-source y distintas al software de referencia pero en C++ ninguna

[1]Dpto. de Ingeniería de Computadores, Universidad Miguel Hernández de Elche, e-mail: mmrach@umh.es.

en Matlab. Algunos autores publican el código en Matlab utilizado en su investigación para poder ser reproducido y ampliado [2]. En esta línea en [3] los autores presentan una clase Matlab que utilizando MEX (integración C++ en Matlab) dan acceso al código del motor CABAC de la HM. Esta herramienta es muy interesante para aplicar a nuestro proyecto en futuras versiones. Únicamente hemos encontrado como trabajos MastherThesis implementaciones de la predicción Intra en Matlab [4] y [5] proporcionando ésta última una herramienta que incluye la transformada y la predicción Intra en Matlab bastante cercana a lo que pretendíamos pero no implementa la cuantización y no genera exactamente los mismos resultados que el software de referencia en las etapas implementadas, con lo que la convierten en una herramienta muy interesante para conocer estas etapas desde un punto de vista docente pero no garantizan la compatibilidad que buscamos.

Por tanto en este trabajo hemos desarrollado una herramienta en Matlab que actualmente es ejecutable únicamente desde la línea de comandos para emular al codificador HEVC con las características mencionadas en la sección II. El resto de las secciones se estructuran de la siguiente manera. En el capítulo III se detalla cada uno de los módulos que compone el emulador. Junto a cada módulo se adjunta su diagrama de flujo para facilitar el entendimiento del mismo. En el capítulo IV se exponen a modo de ejemplo resultados obtenidos con esta herramienta. Finalmente, en el capítulo V se da una visión global y resumida del artículo y se definen las líneas futuras de investigación.

II. Características del emulador

El diseño y desarrollo del emulador HEVC en Matlab se ha realizado siguiendo las directrices indicadas en [6] y usando la documentación y el propio código del software de referencia HM [7]. A continuación se muestran los bloques HEVC codificados en Matlab y cuyos resultados están validados con el software de referencia:

- Transformación y cuantización (así como sus inversas) para tamaños de bloque desde 4×4 hasta 32×32.
- Cuantización perceptual (Weighting Matrix) para todos los tamaños del estándar.
- Obtención de las muestras de referencia para las predicciones y sus filtros cuando corresponden.
- Cálculo de las 35 predicciones Intra (modos Planar, DC y 33 modos angulares).
- Obtención del residuo y reconstrucción de imagen.

El software se completa con la implementación de las siguientes características:

- Propuesta de matriz de cuantización perceptual (y su inversa) para tamaños de bloque 4×4 (CSF).
- Selección del mejor modo de predicción mediante:
  - mínimo SATD.
  - mejor R/D donde el Rate se obtiene vía la entropía de primer orden, no usando CABAC.
- Particionado de imágenes en bloques de tamaño fijo (4, 8, 16 y 32 píxeles).
- Procesa solo luminancia para imágenes en Intra mode o secuencias YUV 4:2:0 All Intra.
- Fichero de configuración con parámetros variables y proceso por lotes.
- Obtención de gráficos R/D y estadísticos para las distintas QPs utilizando diferentes métricas de calidad.
  - PSNR, SSIM [8], MSSSIM [9], VIF [10], VIFP [10], PSNRHVS [11] y PSNRHVSM [12].
- Informe de ganancias Bjøntegaard (BD-Rate y BD-Distortion) adaptadas a las distintas métricas de calidad para cada gráfica R/D obtenida.

Con todo esto, la herramienta es capaz de codificar imágenes o vídeos con formato de origen YUV 4:2:0 (únicamente la componentes de luminancia) y 8 bits de profundidad tal y como lo hace el software de referencia HM tras aplicarle el archivo de configuración modificado deshabilitando las funcionalidades no trasladadas al emulador.

III. Esquema y desarrollo del emulador

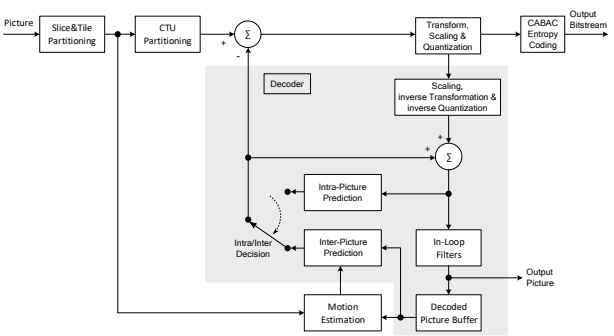


Fig. 1: Diagrama de bloques de un codificador HEVC con decodificador incorporado (*región grisácea*)

El desarrollo del emulador HEVC en Matlab se ha planificado partiendo del esquema básico del codificador HEVC (Figura 1) descartando toda la parte del estimador de movimiento (predicción Inter) así como los filtros In-loop.

La programación del código del emulador se ha realizado de forma modular, lo que permite la paralización de algunas de sus etapas.

A. EmuladorHEVC

En la Figura 2 se muestra el diagrama de flujo del archivo principal del emulador, donde se cargan los parámetros de ejecución, estos son:

- `sequences`: Variable de tipo *cell array* donde se establecen la o las secuencias a procesar.
- `dims`: Variable de tipo *cell array* donde se establecen las dimensiones en píxeles de cada una de las secuencias a procesar.

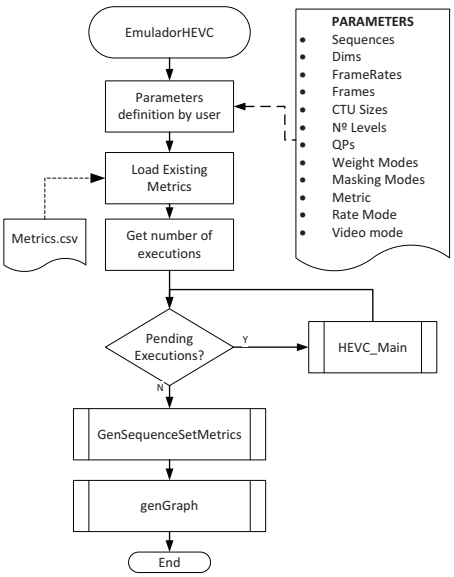


Fig. 2: Diagrama de flujo del archivo `EmuladorHEVC.m`

• `FrameRates`: Variable de tipo *cell array* donde se establece el framerate de cada secuencia.
• `Frames`: Variable de tipo *cell array* donde se especifican los frames a procesar para cada secuencia.
• `CTUSizes` y `NLevels`: Estas dos variables de tipo *double array* establecen el tamaño de CTU y su nivel de división respectivamente. De esta forma, los tamaños fijos de bloque (CU) para el particionado de la imagen vienen determinados por la expresión:

$$\text{CUSize} = \frac{\text{CTUSize}}{2^{\text{NLevel}}} \tag{1}$$

Los valores de CTU permitidos son 16 y 32, mientras que los tamaños de Niveles permitidos van desde 0 hasta 2, con lo que se pueden configurar para tamaños de bloque de 32, 16, 8 y 4.
• `Qps`: Variable de tipo *double array* permite elegir las distintas QPs para cada ejecución. Los valores permitidos corresponden con el estándar HEVC, pudiendo ser desde 0 hasta 51.
• `WeightModes`: Variable de tipo *cell array* donde se determina si se usan o no matrices de cuantización perceptual, basadas en la CSF. Las opciones permitidas son:

- `noCSF`: Utiliza las matrices de cuantización uniformes.
- `staCSF`: Utiliza las matrices de cuantización no-uniformes del estándar HEVC. Para tamaño de bloque 4×4 aplica cuantización uniforme.
- `CSF`: Utiliza las matrices de cuantización no-uniformes del estándar HEVC, excepto para tamaños de bloque 4×4, en cuyo caso se utilizan unas matrices definidas en [13].

• `Metric`: Variable de tipo *char array* donde se especifica la métrica que queremos que muestren las gráficas y para el cálculo del BD-Rate (Bjøntegaard [14]). Cuando se lanza la codificación de las secuencias se calculan todas las métricas (PSNR, SSIM, MSSSIM, VIF, VIFP, PSNRHVS, PSNRHVSM). Se reutilizan estos valores para ejecuciones con la misma configuración.
• `RateMode`: Variable de tipo *char array* donde se especifican las unidades de rate (bits, bpp, bps, kbps, Mbps).
• `videomode`: Variable *booleana* que define si los ficheros de entrada corresponden a imágenes o secuencias, en cuyo caso se promedian los valores de BD-Rate de los frames seleccionados.
• `parallelMode`: El programa permite utilizar esta variable *booleana* para la ejecución paralela en hilos de ejecución independientes.
• `print_to_pdf`: El programa permite utilizar esta variable *booleana* para generar las gráficas resultantes en formato PDF.

El programa genera un archivo `Metrics.csv`, que es la base de datos donde se guarda el resultado de las ejecuciones en formato CSV. Si por los parámetros de entrada se solicita la ejecución de alguna configuración existente en esta base de datos, se omitirá su ejecución y cargará los datos de la misma.

A partir de los parámetros de configuración y descartando los ya realizados se obtiene el número de ejecuciones a realizar. Una ejecución es una combinación única de los parámetros definidos por el usuario, por ejemplo:

```
sequences = 'BlowingBubbles_384x192_50.yuv';
dims = [384,192];
FrameRates = [50];
Frames = [1:50];
CTUSizes = [16];
NLevels = [1];
Qps = [32];
WeightModes = 'noCSF';
```

Después, para cada lanzamiento se ejecuta el proceso HEVCMain, encargado de codificar la secuencia. Una vez se han procesado todos los lanzamientos, se obtienen las métricas y se crea o se añaden nuevas líneas al archivo `Metrics.csv`. Finalmente se lanza la función `genGraph`, que muestra por pantalla las gráficas para todas las secuencias y crea los archivos `Graficas_*.csv` y `Bjontegaard_*.csv`, que contienen los puntos de las curvas y las métricas Bjøntegaard respectivamente.

B. HEVC_Main

`HEVC_Main` es proceso principal encargado de generar la codificación de las secuencias, así como su decodificación para reconstruir la imagen y guardar los archivos necesarios para posteriormente obtener las métricas de calidad. En la Figura 3 se muestra su diagrama de flujo.

El primer paso consiste en calcular el tamaño de los bloques para su particionado utilizando la Ecuación 1. La función `DefineHEVCTables`, obtiene las matrices de transformación y cuantización en función del tamaño de CU.

A continuación se extraen del vídeo original los frames a utilizar en formato YUV y BMP. Siguiendo el Esquema 3, para cada Frame de la ejecución se realizan los siguientes procesos:

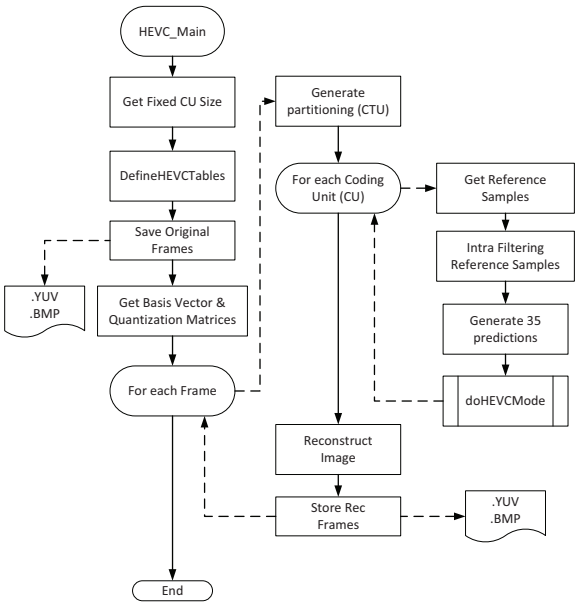


Fig. 3: Diagrama de flujo del archivo `HEVC_Main.m`

- **Particionado del frame:** Se calcula el tamaño del frame para que sea múltiplo de CU y se divide el frame en CUs para su procesamiento.
- **Generación de la matriz de recorrido:** Los CTU de una imagen se recorren en *raster scan order* y su descomposición en CUs utilizando el recorrido *Z-Scan order*(ver Figura 4), que permite tener codificados los bloques necesarios para la predicción Intra.

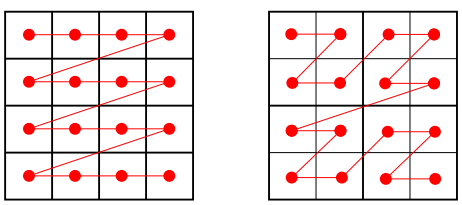


Fig. 4: Algoritmo de recorrido de bloques para un particionado de tamaño fijo. Izquierda: recorrido por CTUs (*raster order*). Derecha: recorrido por CUs (*Z-Scan order*) dentro de cuatro CTUs con un nivel de división.

- **Procesado de cada CU:** El siguiente paso consiste en procesar cada CU en el orden correspondiente, que explicaremos más adelante.
- **Guardado de imagen reconstruida:** Tras procesarse todos los CUs de un frame guardamos la imagen reconstruida en formato YUV y BMP.

El procesamiento que se realiza para cada CU es el siguiente:

- **Obtención de las muestras de referencia:** La función `GetReferenceSamples` genera las muestras de referencia de los bloques adyacentes necesarias para la predicción Intra siguiendo el algoritmo del estándar HEVC.
- **Post-filtrado de muestras de referencia:** Obtenidos los vectores de referencia, a continuación se determina si se debe aplicar el filtro de post-procesado, tal y como está descrito en [6]. En la Tabla I se muestran las diferentes opciones de filtrado según el tamaño de bloque y predicción.

TABLA I: Aplicación de filtro de post-procesado a las muestras de referencia

Tamaño de CU	**Aplicación de filtro**
4×4	No aplicar
8×8	Planar y Angular (2, 18, 34)
16×16	Todos los modos excepto DC, Angular (9, 10, 11, 25, 26, 27)
32×32	Todos los modos excepto DC, Angular (10, 26)

- **Generación de las predicciones:** Se obtienen las 35 predicciones Intra utilizando los vectores de referencia previamente generados.
- **Codificación y decodificación:** Este es el último paso para cada CU y lo lleva a cabo la función `doHEVCMode`, que se encuentra desglosado y explicado más adelante.

C. doHEVCMode

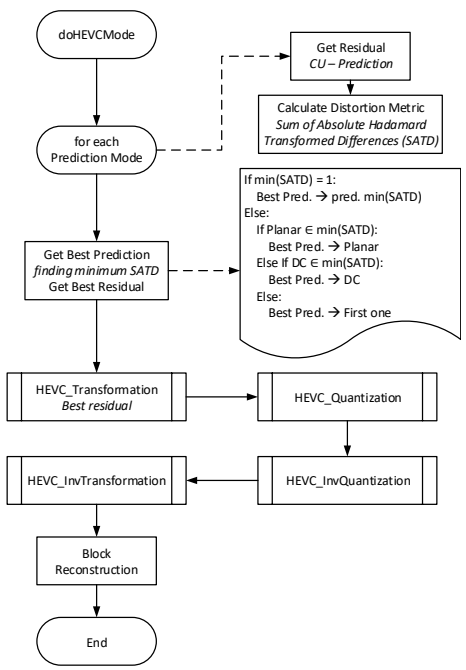


Fig. 5: Diagrama de flujo del archivo `doHEVCMode.m`

En esta función lanzada para cada bloque CU, y cuyo diagrama de flujo se muestra en la Figura 5, se realiza el proceso de cálculo y elección del mejor modo de predicción así como la transformación, cuantización (directa e inversa) del bloque correspondiente.

El primer paso consiste en obtener la mejor predicción posible para el bloque. Como ya hemos visto, en la función `HEVC_Main` se han calculado las 35 predicciones a partir de los vectores de referencia. En esta función se obtiene el residuo de cada modo siguiendo la expresión $CU_{residual} = CU - CU_{predicted}$ y después se calcula su coste o medida de error en base a la suma de las diferencias absolutas transformadas

(en inglés, Sum of Absolute Transformed Differences o SATD).

Este algoritmo, a diferencia de la suma de las diferencias absolutas (SAD), trabaja en el dominio frecuencial aplicando la transformada de Hadamard, computacionalmente más sencilla que la DCT. Debido a que la transformada de Hadamard elimina la redundancia espacial, el uso del SATD es un indicador claro para la distorsión D en el problema de optimización del Rate-Distortion (RDO). El cálculo del SATD para el residuo de un bloque CU de tamaño 4×4 se realiza según la Ecuación 2:

$$\mathrm{SATD} = \frac{1}{2} \sum_{i,j} \left| \mathrm{H} \cdot \mathrm{CU}_{residual} \cdot \mathrm{H}^T \right| \qquad (2)$$

donde H corresponde a la matriz de transformación de Hadamard.

El cálculo del SATD en el emulador lo realiza la función `CalcHad`, donde tiene especifica la transformada para bloques de 4×4 y 8×8 solamente. Para bloques de tamaño superior el cálculo se realiza particionando el bloque en bloques de 8×8, calculando para cada uno el SATD y sumando el resultado de todos.

Una vez se han calculado los SATD para cada modo se selecciona el mejor modo de predicción, que será el que tenga el menor valor de distorsión. Si hay más de un modo con el valor mínimo de SATD la elección del mejor modo se determina en función de los siguientes aspectos:

1. Si el modo Planar se encuentra entre los candidatos se selecciona el modo Planar.
2. Si el modo DC se encuentra entre los candidatos pero no el Planar, se selecciona el modo DC.
3. Si solamente hay modos angulares, se selecciona el de menor índice.

Una vez se ha obtenido el mejor modo de predicción, aplicamos la transformada (función `HEVC_Transformation`) al bloque residuo $\mathrm{CU}_{residual}$, obteniendo la matriz de coeficientes transformados T. Después se aplica la cuantización en la función `HEVC_Quantization`, al cual se le pasa por parámetros T, la matriz de cuantización, el valor QP y la profundidad de bits. Los coeficientes transformados y cuantizados, Q, se guardan para más adelante obtener la entropía del bloque.

Para la reconstrucción del bloque se realiza la cuantización inversa a la matriz Q (función `HEVC_InvQuantization`), obteniendo la matriz IQ. Finalmente, se le aplica la transformada inversa (función `HEVC_InvTransformation`) obteniendo el bloque reconstruido IT.

D. GenSequenceSetMetrics

Realiza el cálculo de las métricas de calidad a partir de las imágenes originales y reconstruidas así como la entropía y guarda estos valores en un fichero de texto. En la Figura 6 se muestra su diagrama de flujo.

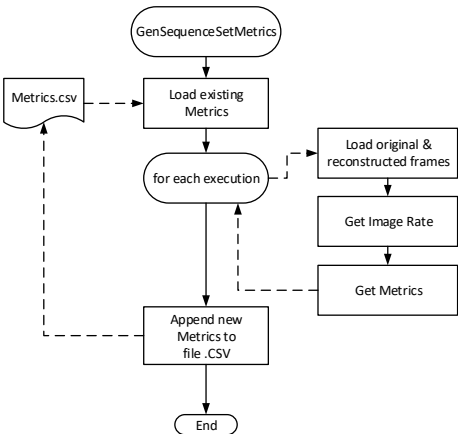


Fig. 6: Diagrama de flujo del archivo `GenSequenceSetMetrics.m`

El primer paso consiste en cargar en memoria el archivo de métricas `Metricas.csv`, si existe, almacenado en el directorio de salida. Después, por cada ejecución, comprueba si ya existe el cálculo de la entropía y de las métricas, en cuyo caso pasa a la siguiente ejecución. Si la ejecución no tuviese el valor previamente calculado, la función realizaría las siguientes acciones:

1. Cargar en memoria el frame original, el reconstruido y la matriz con los coeficientes cuantizados.
2. Obtener la entropía del bloque.
3. Calcular y guardar métricas.

La entropía (el rate) y la calidad nos permitirán obtener las curvas rate-distortion que se obtienen como resultado de la ejecución. El estándar HEVC incluye la codificación entrópica binaria CABAC, pero en este trabajo se ha utilizado el entrópico de Shannon de orden cero por simplicidad para estimar el rate. Para obtener el número de bits totales necesarios para codificar el frame se multiplica la entropía por el número de píxeles del frame. Finalmente, para obtener la tasa de bits multiplicamos los bits por el *framerate* de la secuencia. Se calculan los valores de calidad para las distintas métricas mencionadas anteriormente.

E. genGraph

Este módulo, cuyo diagrama de flujo se muestra en la Figura 7, imprime por pantalla las gráficas para las ejecuciones definidas en los parámetros de la función `EmuladorHEVC`. También genera un archivo CSV con los puntos de las curvas así como un reporte con las métricas Bjøntegaard.

La función carga en memoria el archivo `Metrics.csv` que contiene los parámetros de rate y calidad previamente obtenidos para cada ejecución. Después, dependiendo del valor del parámetro `videomode`, realiza el promedio de todos los *rates* y métricas de calidad para todos los frames de cada secuencia. Para el caso en que `videomode` no esté activo, esta función devolverá una gráfica

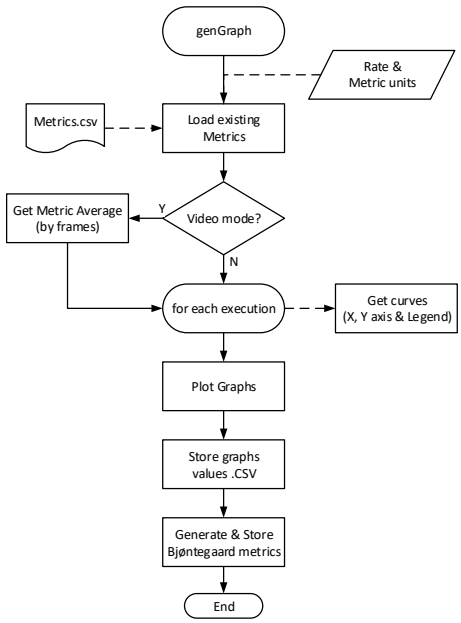


Fig. 7: Diagrama de flujo del archivo `genGraph.m`

para cada uno de los frames que tenga la secuencia.

Las gráficas vienen estructuradas de la siguiente manera: en el eje horizontal se representa el *rate* definido en el parámetro `RateMode`, mientras que en el eje vertical se representa la métrica de calidad, definida en el parámetro `Metric`. Cada tamaño de bloque CU de cada secuencia tiene su propia gráfica independiente, y dentro de cada gráfica se representa mediante curvas las diferentes combinaciones de parámetros restantes (con o sin CSF), donde cada punto de la gráfica corresponde a cada valor del factor de calidad `QPs`.

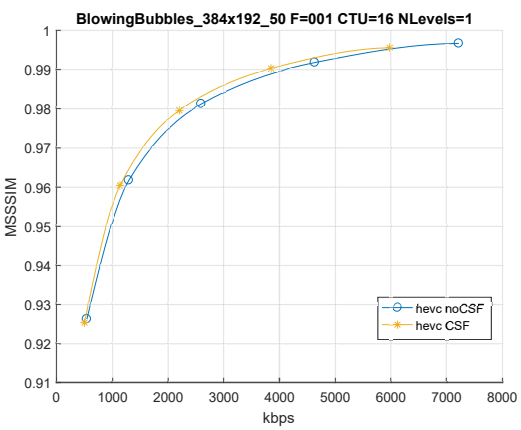


Fig. 8: Gráfica de ejemplo.

La Figura 8 es un ejemplo de ejecución para el frame 1 de la secuencia BlowingBubbles. En el título se muestra el nombre de la secuencia, el número de frame, el valor de CTU y el Nivel de particionado. Esta ejecución se ha lanzado con la opción `videomode` a *false*, en caso de haber sido *true* no aparecería el valor del frame. La leyenda identifica las curvas a partir de los siguientes parámetros: `hevc ␣WeightMode`.

Si se tiene establecido a *true* el valor del campo `print_to_pdf` la ejecución exportará cada una de las gráficas a archivos PDF, asignándoles a cada una un nombre que identifique los parámetros únicos de la misma. Siguiendo el ejemplo de la Figura 8, el nombre del archivo PDF exportado para esa gráfica sería *BlowingBubbles_384x192_50 F=001 CTU=16 NLevels=1 MSSSIM-kbps.pdf*

Con respecto a las curvas, los puntos indican el lugar exacto que relaciona la métrica de calidad y el *rate*, mientras que para la unión de los puntos no se ha trazado una línea recta sino que se ha suavizado mediante interpolación PCHIP (*Piecewise Cubic Hermite Interpolating Polynomial*), algoritmo que incluye Matlab.

Esta función también genera un archivo .CSV con los datos tabulados de una forma específica con el objetivo de ser abierto con una hoja de cálculos (por ejemplo Microsoft Excel) y realizar fácilmente las mismas gráficas con facilidad. En la figura 9 se muestra este archivo para el ejemplo realizado en este apartado.

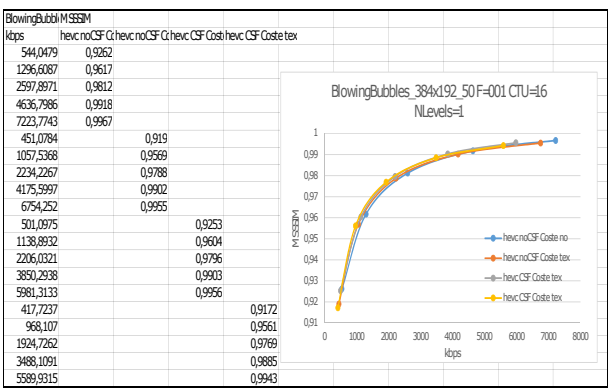


Fig. 9: Archivo CSV generado con los puntos de las gráficas y su representación en Microsoft Excel

Por último, esta función genera otro archivo CSV con el cálculo de las métricas Bjøntegaard, que nos permiten comparar las curvas generadas de forma numérica.

IV. Ejemplos de ejecución

En esta sección mostramos en las figuras 10 a 12 algunos resultados de ejecuciones para distintas secuencias en gráficas Rate-Distortion y valores BD-Rate.

Adicionalmente a las características incluidas en la propia herramienta se ha utilizado la misma para incorporar un modelo de texture masking aplicado al HEVC y presentado en [15], de modo que en las gráficas (ver Figura 13) se puede analizar las mejoras introducidas en el rendimiento del codificador. Así mismo este rendimiento nos lo ofrece la herramienta en formato BD-Rate como vemos en la Tabla II.

V. Conclusiones y trabajo futuro

En este artículo hemos presentado una herramienta emuladora de HEVC en Matlab que facilita la investigación de distintos esquemas de cuantización, transformación, etc. debido a que la Matlab es un

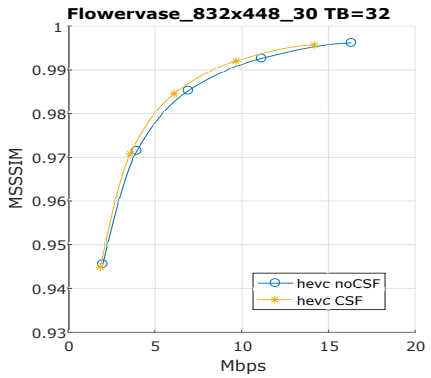


Fig. 10: Gráfica MSSSIM - Mbps para la secuencia *Flowervase* y tamaño de bloque 32×32.

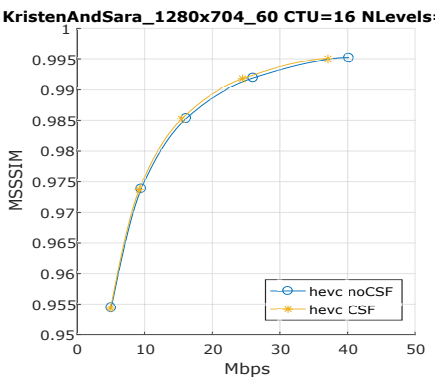


Fig. 11: Gráfica MSSSIM - Mbps para la secuencia *KristenAndSara* y tamaño de bloque 16×16.

lenguaje orientado a matrices y todas las operaciones son más sencillas de analizar e implementar que utilizando el software de referencia. Para ello, esta herramienta ha sido validada contra el software de referencia HM 14.0, garantizando que las etapas de transformación, cuantización y predicción Intra generan los mismos resultados que dicho software de referencia. Así el investigador podrá intercalar y modificar estos esquemas, obteniendo resultados que podrán ser integrados en el software de referencia una vez validados.

La versión actual de la herramienta realiza un particionado de bloques de tamaño fijo, obtiene gráficas

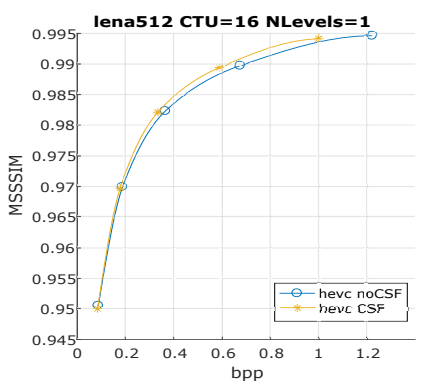
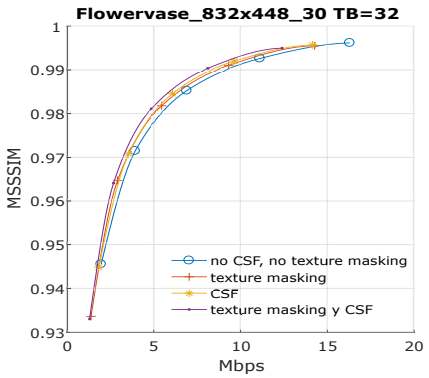
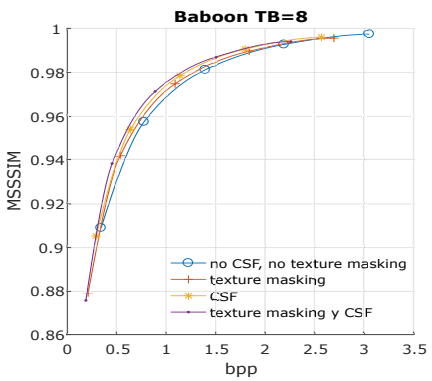


Fig. 12: Gráfica MSSSIM - bpp para la imagen *Lena* y tamaño de bloque 8×8.



(a) Curva MSSSIM - Mbps para tamaño de bloque 32×32



(b) Curva MSSSIM - bpp para tamaño de bloque 16×16

Fig. 13: Ejemplo de curvas Rate-Distortion para diferentes imágenes y secuencias de vídeo.

TABLA II: Resultados BD-Rate (%) para métrica perceptual MS-SSIM y tamaño de bloque 32×32.

Secuencia test	**Texture masking**	**Contrast masking**	**Texture y contrast masking**
Baboon (imagen)	-2.06	-8.90	-10.39
Lena (imagen)	-5.25	-3.81	-9.49
BlowingBubbles	-2.03	-5.01	-6.57
Flowervase	-7.56	-7.56	-13.92
KristenAndSara	-9.09	-2.41	-10.91
Cactus	-6.90	-2.55	-8.84

Rate-Distortion como resultado de su ejecución y calcula valores BD-Rate para las distintas curvas de estas gráficas. También permite habilitar o deshabilitar el uso de matrices de cuantización perceptual.

Como trabajos futuros, planteamos la ampliación de la herramienta con la integración del particionado Quad-Tree del estándar HEVC, la integración del algoritmo CABAC e implementación del RDO en base a él, implementación de los filtros de de-blocking y SAO, la inclusion de nuevas métricas y la implementación de una interfaz de usuario con GUIDE de Matlab.

Esta herramienta estará disponible en breve en su versión actual (y futuras) en la web del grupo de investigación GATCOM [16].

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades con referencia RTI2018-098156-B-C54 cofinanciado con fondos FEDER (MINECO/FEDER/UE).

Referencias

[1] D. Springer, W. Schnurrer, A. Weinlich, A. Heindel, J. Seiler, and A. Kaup, "Open Source HEVC Analyzer for Rapid Prototyping (HARP)," in *IEEE Int. Conf. on Image Processing (ICIP)*, Paris, France, October 2014.

[2] Haoming Chen, Tao Zhang, Ming-Ting Sun, Ankur Saxena, and Madhukar Budagavi, "Improving intra prediction in high-efficiency video coding," *IEEE Transactions on Image Processing*, vol. 25, no. 8, pp. 3671–3682, 2016.

[3] Blaeser M., Rohlfing C., Gao Y., and Wien M., "Adaptive coding of non-negative factorization parameters with application to informed source separation," in *43rd International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Calgary, Alberta, Canada, 2018, Proceedings of the 43rd International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE.

[4] Vishal Deep, "Realization of State-of-the-art Intra-Prediction High Efficiency Video Coding (HEVC)," M.S. thesis, Lyles College of Engineering California State University, 12 2016.

[5] Masoud Nouripayam and Nima Sheikhipoor, "HEVC (H.265) Intra-Frame prediction implementation using MATLAB," 2014.

[6] Vivienne Sze, Madhukar Budagavi, and Gary J. Sullivan, *High Efficiency Video Coding (HEVC): Algorithms and Architectures*, Springer Publishing Company, Incorporated, 2014.

[7] Fraunhofer Institute for Telecommunications, "HM Reference Software Version 14.0," `https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-14.0/`, 2014.

[8] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *Image Processing, IEEE Transactions on*, vol. 13, no. 4, pp. 600–612, 2004.

[9] Z. Wang, E.P. Simoncelli, and A.C. Bovik, "Multiscale structural similarity for image quality assessment," in *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, 2003, vol. 2, pp. 1398–1402 Vol.2.

[10] H.R. Sheikh and A.C. Bovik, "Image information and visual quality," *Image Processing, IEEE Transactions on*, vol. 15, no. 2, pp. 430–444, 2006.

[11] Nikolay Ponomarenko, Flavia Silvestri, Karen Egiazarian, Marco Carli, Jaakko Astola, and Vladimir Lukin, "On between-coefficient contrast masking of dct basis functions," in *Proceedings of the third international workshop on video processing and quality metrics*, 2007, vol. 4.

[12] Nikolay Ponomarenko, Federica Battisti, Karen Egiazarian, Jaakko Astola, and Vladimir Lukin, "Metrics performance comparison for color image database," in *Fourth international workshop on video processing and quality metrics for consumer electronics*, 2009, vol. 27.

[13] Miguel Onofre Martínez-Rach, *Perceptual image coding for wavelet based encoders*, Ph.D. thesis, Universidad Miguel Hernández, Dec. 2014.

[14] G Bjontegaard, "Calculation of average psnr differences between rd-curves," *Proceedings of the ITU-T Video Coding Experts Group (VCEG) Thirteenth Meeting*, 01 2001.

[15] Javier Ruiz, Otoniel López Granado, Manuel Malumbres, and Miguel Martínez-Rach, "Análisis combinado de texture y contrast masking en hevc," in *Jornadas SARTECO*, 2019.

[16] Universidad Miguel Hernández de Elche, "Grupo de Arquitectura y Tecnología de Computadores," `http://atc.umh.es/gatcom/`.