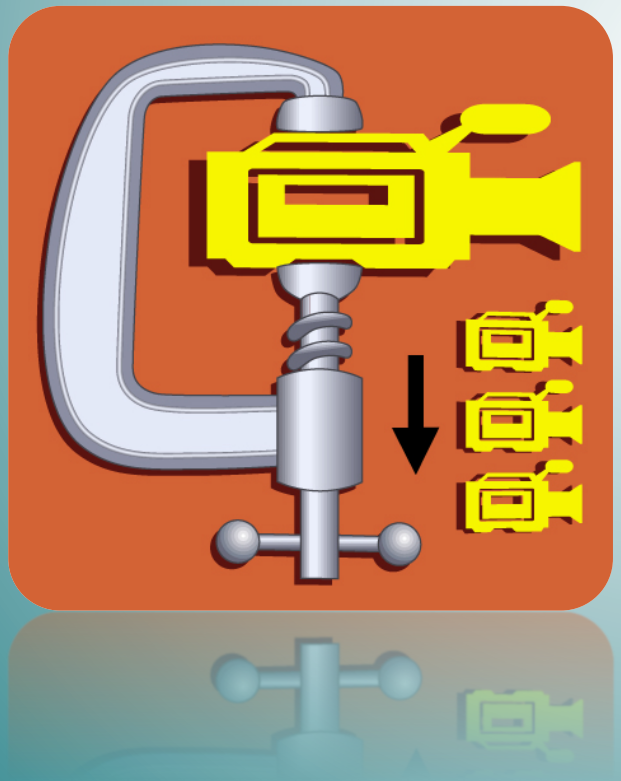
Compresión de Vídeo

Tema 2.7. Transformación y cuantificación



Juan A. Michell Martín
Gustavo A. Ruiz Robredo

Departamento de Electrónica y Computadores

TRANSFORMADA DISCRETA DEL COSENO (DCT)

- La transformada discreta del coseno (DCT) opera con un block **x** de N×N muestras para crear la matriz de coeficientes **X** de dimensión N×N

- La 2D DCT directa (*forward*) viene definida como

$$X = A \cdot x \cdot A^T$$

- La 2D DCT inversa (*inverse*) viene definida como

$$x = A^T \cdot X \cdot A$$

- Los elementos **A** son

$$A(n,m) = C_n \cos\frac{(2m+1)n\pi}{2N}$$

donde

$$C_n = \sqrt{\frac{1}{N}}, \quad n = 0 \qquad C_n = \sqrt{\frac{2}{N}}, \quad n > 0$$

- De forma que la 2D DCT directa e inversa se puede expresar

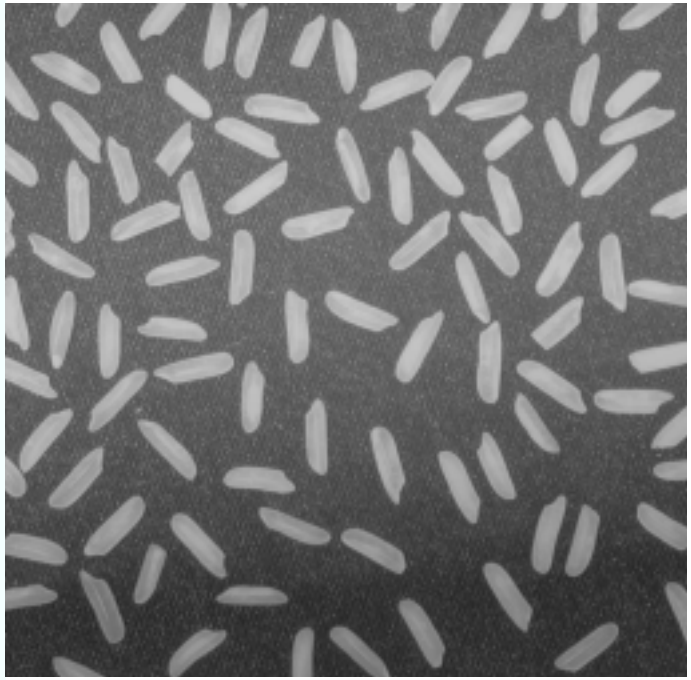
$$X(u,v) = C_u C_v \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x(n,m) \cos\frac{(2n+1)u\pi}{2\times N} \cos\frac{(2m+1)v\pi}{2\times N}$$

$$x(n,m) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C_u C_v X(u,v) \cos\frac{(2n+1)u\pi}{2\times N} \cos\frac{(2m+1)v\pi}{2\times N}$$

- Por ejemplo para N=4 se tiene

$$A = \begin{bmatrix} \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{5\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{7\pi}{8}\right) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{2\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{6\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{10\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{14\pi}{8}\right) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{9\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{15\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{21\pi}{8}\right) \end{bmatrix}$$

Concentración
de la energía



2D DCT

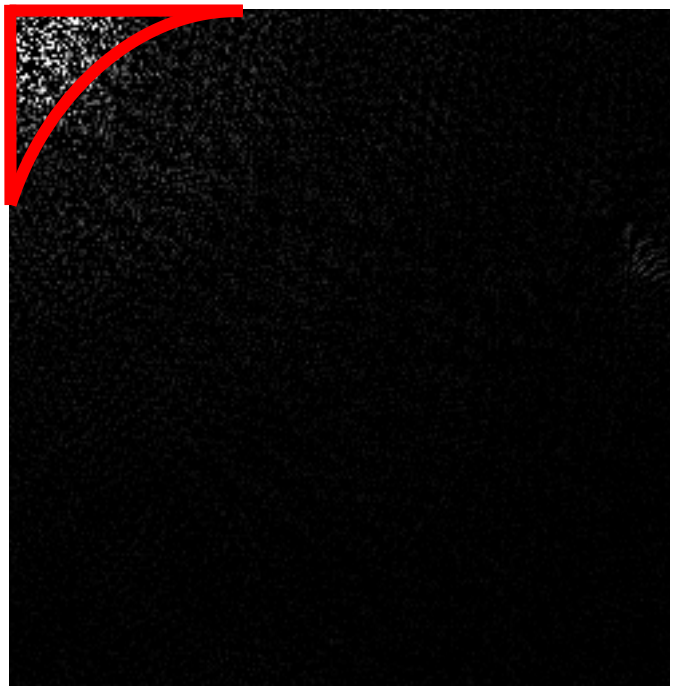


Imagen original

Distribución de coeficientes

¿Por qué la DCT?

- La DCT tiene una buena capacidad de compactación de la energía al dominio transformado, es decir, consigue concentrar la mayor parte de la información en pocos coeficientes transformados.
- La transformación es independiente de los datos. El algoritmo aplicado no varia con los datos que recibe, como sí sucede en otros algoritmos de compresión.
- Hay fórmulas para el cálculo rápido del algoritmo reduciendo el número de operaciones.
- Produce pocos errores en los límites de los bloques imagen. La minimización de los errores a los contornos de los bloques permite reducir el efecto de bloque en la imágenes reconstruidas.
- Tiene una interpretación frecuencial de los componentes transformados. La capacidad de interpretar los coeficientes en el punto de vista frecuencial permite aprovechar al máximo la capacidad de compresión.

- **Operando resulta**

$$A=\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) \end{bmatrix}$$

- **Otra manera de expresar la anterior matriz**

$$A=\begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & c \end{bmatrix}$$

$$a=\frac{1}{2} \qquad b=\sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) \qquad c=\sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right)$$

TRANSFORMADA ENTERA DEL H.264

- La transformada usada por el H.264 está basada en la DCT 4×4 pero con algunas diferencias fundamentales :
  - Es una transformada de números enteros.
  - Puede ser implementada con solamente operaciones de suma/resta y desplazamientos.
  - Los factores de normalización de la transformada se integran en el cuantificador, reduciendo el número total de multiplicadores.
  - Toda la aritmética se puede realizar en 16 bits, compatible con multitud de procesadores.

- La transformada directa 2D DCT se puede expresar también como

$$X = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix} \begin{bmatrix} \\ x \\ \\ \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix}$$

El operador $\otimes$ significa multiplicación elemento a elemento o multiplicación escalar, no es una multiplicación matricial.

- A nivel matricial se indica como

$$X = \left(C \cdot x \cdot C^T \right) \otimes E$$ Post-escaladado

1. Al término $C \cdot x \cdot C^T$ se denomina cuerpo principal (*core*) de la transformada 2D.
2. E es la matriz de los factores de escalado.

- Para simplificar la anterior transformada y asegurar que permanece ortogonal, los coeficientes a b y d son simplificados a

$$a = \frac{1}{2} \quad b = \sqrt{\frac{2}{5}} \quad d = \frac{1}{2}$$

- Como resultado se obtiene la transformada **directa** usada en el H.264 es

$$X = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} x \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{bmatrix}$$

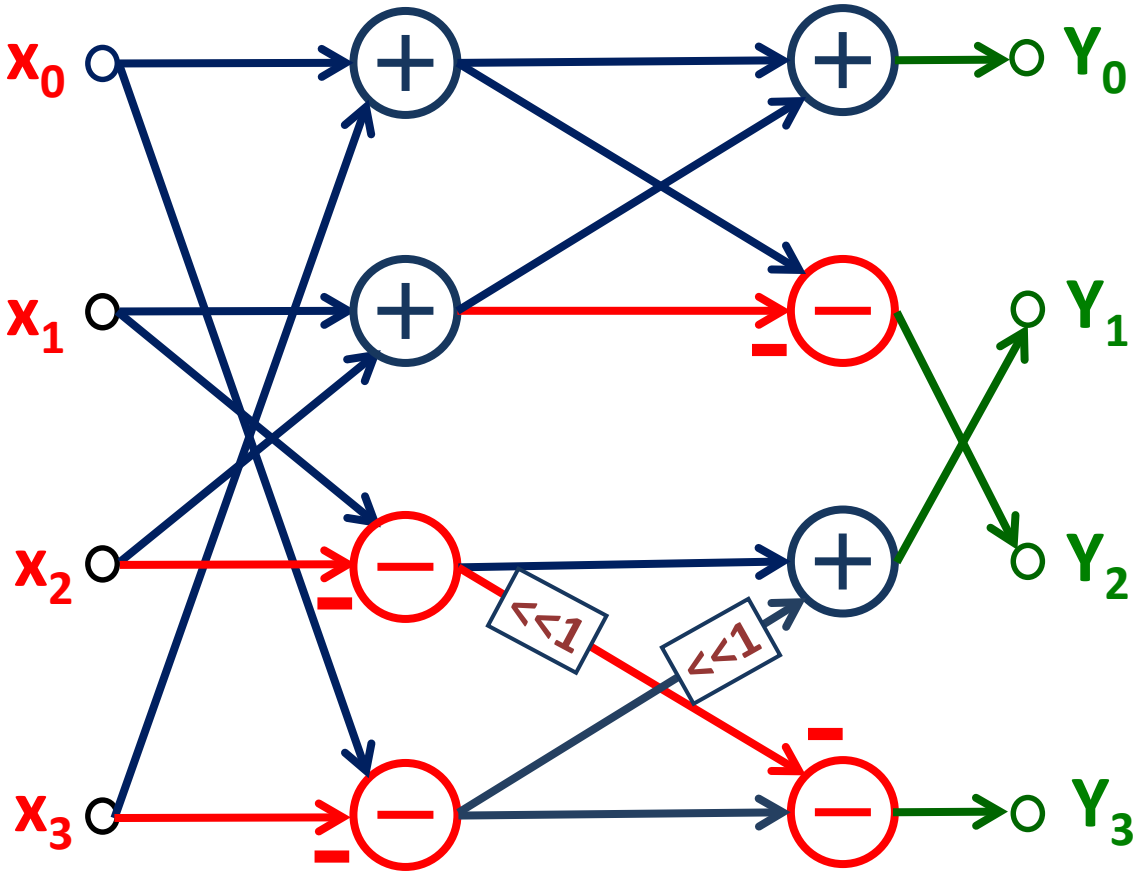
que a nivel matricial se indica como

$$X = \left(C_f \cdot x \cdot C_f^T \right) \otimes E_f$$

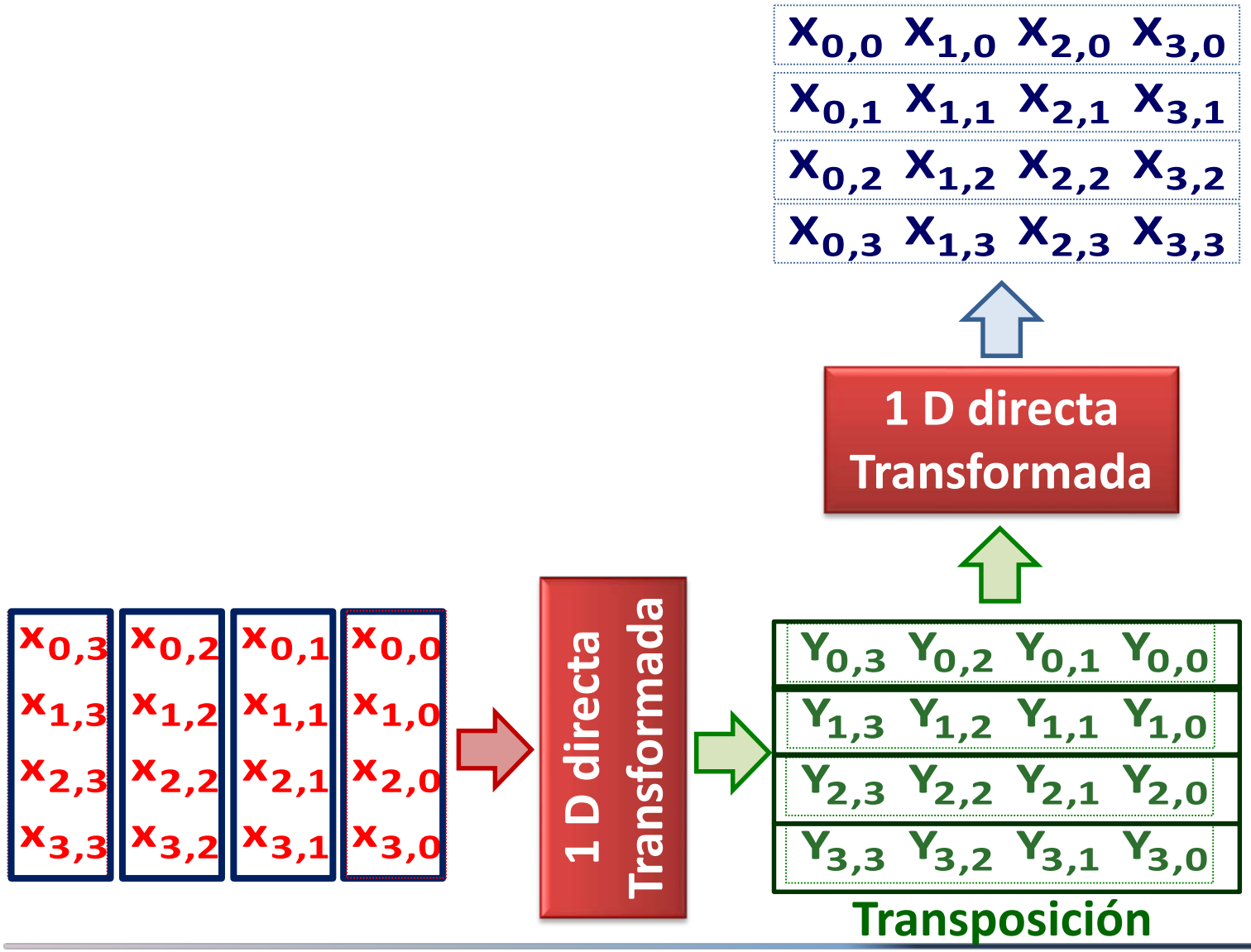
Se puede observar que el *core* de esta transformada $C_f \cdot x \cdot C_f^T$ se calcula a partir de operaciones de suma/resta y de *shift*. No necesita multiplicaciones.

Algoritmo rápido de la transformada entera directa 1D

$$Y_{1D} = C_f \cdot x_{1D}$$



Implementación de la transformada directa 2D basada en dos 1D



- La transformada inversa 2D usada en el H.264 es

$$x = \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 2 & -1/2 \end{bmatrix} \left(\begin{bmatrix} X \end{bmatrix} \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix}$$

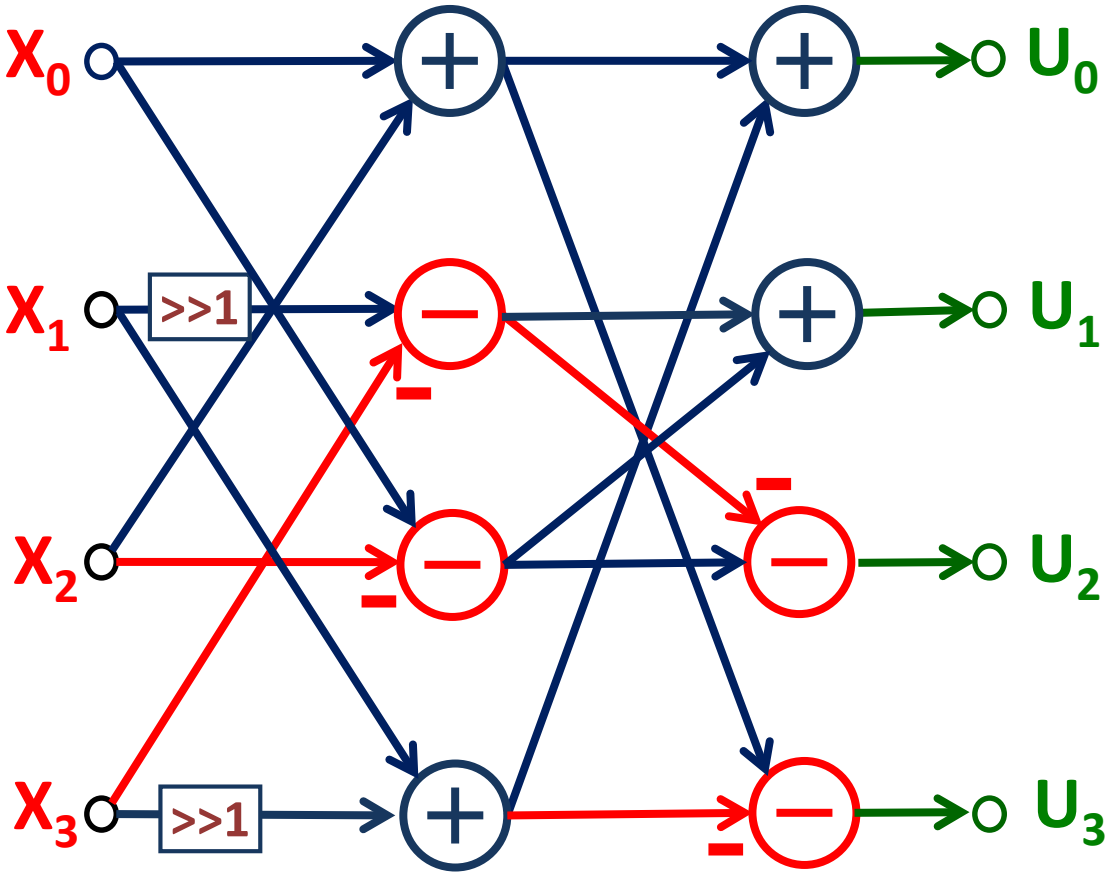
- Matricialmente se puede expresar como

$$x = C_i^T \left(X \otimes E_i \right) C_i$$

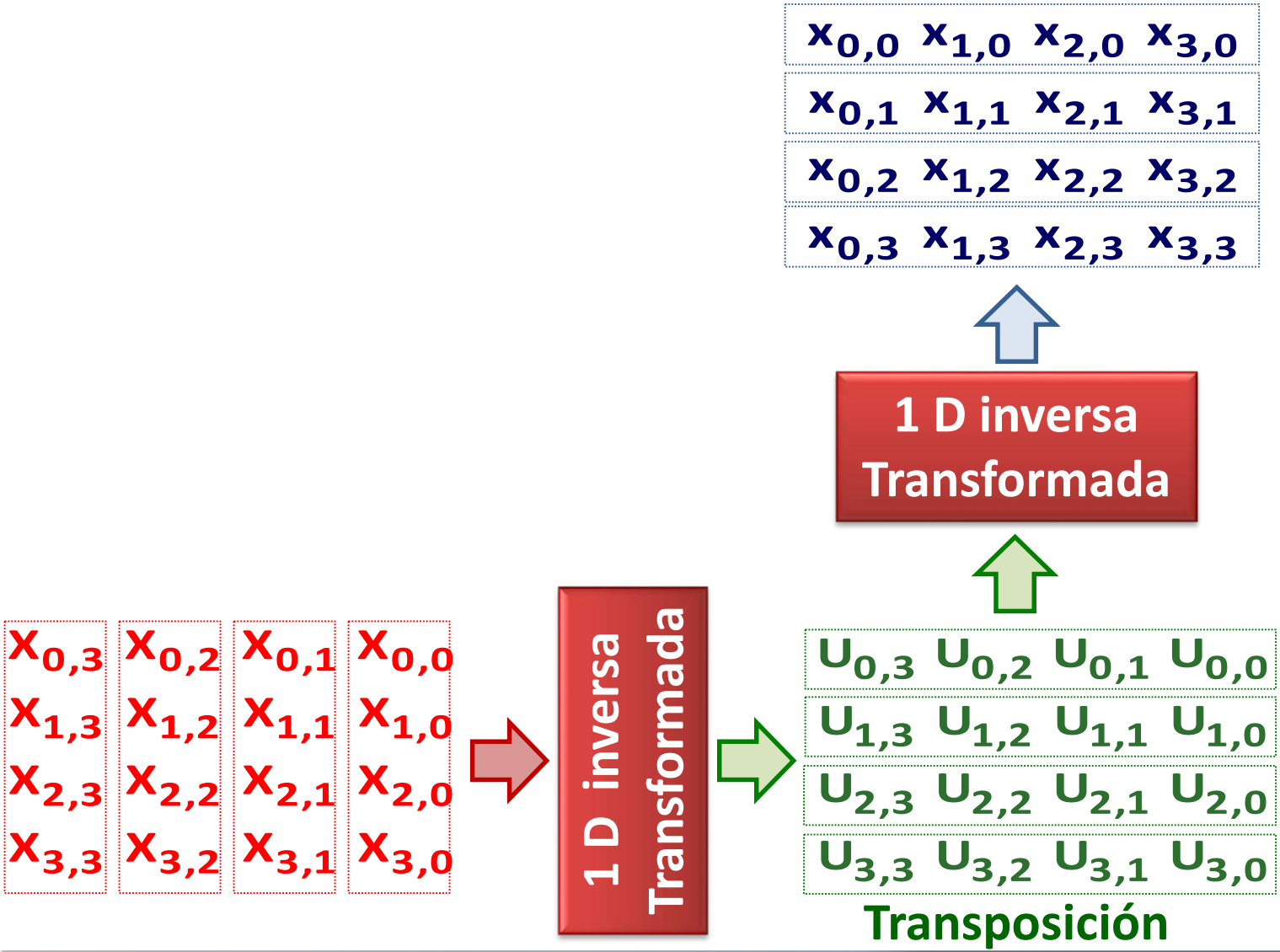
Pre-escaladado

Algoritmo rápido de la transformada inversa 1D

$$U_{1D} = c_i^T \cdot X_{1D}$$



Implementación de la transformada inversa 2D basada en dos 1D



CUANTIFICACIÓN (*QUANTIZATION*)

- El H.264 asume cuantificación escalar que satisface los siguientes requerimientos:
  - Evitar la división y aritmética de punto flotante.
  - Incorporar el post- y pre-escalado de las matrices E_f y E_i.

- La cuantización directa básica se aplica a los coeficientes **X** de la transformada

$$Z(u,v) = \text{round}\left(X(u,v) / QStep\right)$$

donde

  - **X(u,v)** son los coeficientes de la transformada.
  - **Z(u,v)** son los coeficientes cuantificados.
  - **QStep** es el parámetro de cuantificación que indica el tamaño de paso. En el H.264 se define un total de 52 valores estandarizados diferentes.

- Para indexar estos 52 valores se utiliza el parámetro de cuantificación QP (*Quantization Parameter*) que varía de 0 a 51.

Tabla de cuantificación definida en el H.264

QP	0	1	2	3	4	5	
QStep	0.625	0.6875	0.8125	0.875	1	1.125	
QP	6	7	8	9	10	11	12
QStep	1.25	1.375	1.625	1.75	2	2.25	2.5
QP	…	18	…	24	…	30	…
QStep	…	5	…	10	…	20	…
QP	36	…	42	…	48	…	51
QStep	40	…	80	…	160	…	224

- Un incremento de QP en 1 supone un incremento aproximado del 12% en el valor de QStep.
- Un incremento de QP en 6 significa que QStep es incrementado por un factor de 2.

Post-escalado

- La transformada directa 2D

$$X = \left(C_f \cdot x \cdot C_f^{\,T} \right) \otimes E_f = W \otimes \begin{bmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{bmatrix}$$

- En el cuantificador directo, los elementos de E_f se incorporan al cuantificador de forma que cada coeficiente W(u,v) es escalado independientemente:

$$Z(u,v) = \text{round}\left(W(u,v) \cdot E_f(u,v) / QStep \right)$$

- Para simplificar la aritmética y evitar divisiones de número flotantes, se utiliza una matriz de coeficientes enteros MF(u,v)

$$Z(u,v) = \text{round}\left(W(u,v) \cdot MF(u,v) / 2^{qbits} \right)$$

donde $\frac{MF(u,v)}{2^{qbits}} = \frac{E_f(u,v)}{QStep}$ y $qbits = 15 + floor(QP / 6)$

Definición de los valores MF(u,v)

QP%6	Posiciones (0,0), (2,0), (2,2), (0,2)	Posiciones (1,1), (1,3), (3,1), (3,3)	resto
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

Ejemplo: Supongamos QP=4 y (u,v)=(0,0). Entonces, QStep=1.0, $E_f(u,v)=a^2=0.25$ y qbits=15. Entonces

$$\frac{MF(0,0)}{2^{qbits}} = \frac{E_f(0,0)}{QStep} \Rightarrow MF(0,0) = E_f(0,0) \cdot 2^{qbits} \cdot QStep = 0.25 \cdot 2^{15} \cdot 1.0 = 8192$$

- Finalmente, las ecuaciones usadas por el H.264 en el proceso de cuantización directa son

$$|Z(u,v)| = \left(|W(u,v)|\right) \cdot MF(u,v) + f) >> qbits$$

$$sign\left(Z(u,v)\right) = sign\left(W(u,v)\right)$$

$$f = \begin{cases} 2^{qbits} / 3 \text{ para intra} \\ 2^{qbits} / 6 \text{ para inter} \end{cases}$$

$$qbits = 15 + floor(QP / 6)$$

- QP%6 indica que los coeficientes se repiten cada incremento en 6 de QP.
- qbits se incrementa en 1 por cada incremento en 6 de QP. Es decir, qbits=15 para $0 \leq QP \leq 5$, qbits=16 para $6 \leq QP \leq 11$, qbits=17 para $12 \leq QP \leq 17$, etc.

REESCALADO (*RESCALING*)

- La transformada inversa 2D

$$x' = C_i^T (Z \otimes E_i) C_i = C_i^T \left(Z \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) C_i$$

- El proceso de pre-escalado se realiza de la siguiente manera:

1.- Se pre-escala los coeficientes a través del factor QStep:

$$X'(u,v) = Z(u,v) \cdot QStep$$

2.- Se incluye las constantes E_i en el proceso de rescalado junto con la constante de normalización, resultando

$$WI(u,v) = Z(u,v) \cdot QStep \cdot E_i(u,v) \cdot 64 = Z(u,v) \cdot MI(u,v) \cdot 2^{floor(QP/6)}$$

3.- El proceso de reescalado finaliza realizando la transformada inversa:

$$xr = C_i^T \cdot WI \cdot C_i$$

Definición de los valores MI(u,v)

QP%6	Posiciones (0,0), (2,0), (2,2), (0,2)	Posiciones (1,1), (1,3), (3,1), (3,3)	resto
0	10	16	13
1	11	18	14
2	13	20	16
3	14	23	18
4	16	25	20
5	18	29	23

Ejemplo: Supongamos QP=3 y (u,v)=(1,2). Entonces, QStep=0.75 y $2^{floor(QP/6)}=1$, $E_i(1,2)=ab=0.3162$. Entonces

$$MI(1,2) = QStep \cdot E_i(1,2) \cdot 64 \cong 18$$

$$W'(1,2) = Z(1,2) \cdot MI(1,2) \cdot 2^{floor(QP/6)} = Z(1,2) \cdot 18 \cdot 1$$

Ejemplo: QP=10

- **Bloque de entrada**

$$x = \begin{bmatrix} 5 & 11 & 8 & 10 \\ 9 & 8 & 4 & 12 \\ 1 & 10 & 11 & 4 \\ 19 & 6 & 15 & 7 \end{bmatrix}$$

```
Matlab: x=[5 11 8 10; 9 8 4 12; 1 10 11 4; 19 6 15 7]
```

- ***Core* de la transformada directa**

$$W = C \cdot x \cdot C^T = \begin{bmatrix} 140 & -1 & -6 & 7 \\ -19 & -39 & 7 & -92 \\ 22 & 17 & 8 & 31 \\ -27 & -32 & -59 & -21 \end{bmatrix}$$

```
Matlab: W=DCT_forward(x)
```

Cuantización.

$$|Z(u,v)| = \left(|W(u,v)|\right) \cdot MF(u,v) + f) >> qbits$$

$$sign\left(Z(u,v)\right) = sign\left(W(u,v)\right)$$

$$f = \begin{cases} 2^{qbits} / 3 \text{ para intra} \\ 2^{qbits} / 6 \text{ para inter} \end{cases}$$

qbits=15+floor(QP/6)=15+floor(10/6)=16

$$f = 2^{qbits} / 3 = 2^{16} / 3$$

$$QP\%6 = 4 \Rightarrow MF = \begin{bmatrix} 8192 & 5243 & 8192 & 3355 \\ 5243 & 3355 & 5243 & 3355 \\ 8192 & 5243 & 8192 & 5243 \\ 3355 & 5243 & 5243 & 3355 \end{bmatrix}$$

Los coeficientes cuantificados

$$Z = \begin{bmatrix} 17 & 0 & -1 & 0 \\ -1 & -2 & 0 & -5 \\ 3 & 1 & 1 & 2 \\ -2 & -1 & -5 & -1 \end{bmatrix}$$

QP

Inter → 0
Intra → 1

```
Matlab: Z=Quantization_forward(W, 10, 0)
```

El proceso de preescalado realiza la operación

$$WI(u,v) = Z(u,v) \cdot MI(u,v) \cdot 2^{floor(QP/6)}$$

$$QP\%6 = 4 \Rightarrow MI = \begin{bmatrix} 16 & 20 & 16 & 20 \\ 20 & 25 & 20 & 25 \\ 16 & 20 & 16 & 20 \\ 25 & 20 & 20 & 25 \end{bmatrix}$$

$$2^{floor(QP/6)} = 2$$

$WI =$

544	0	-32	0
-40	-100	0	-250
96	40	32	80
-80	-50	-200	-50

QP

```
Matlab: WI=Quantization_inverse (Z, 10)
```

- *Core* de la transformada inversa

$$xr = \text{round}(C_i^T \cdot WI \cdot C_i / 64) =$$

4	13	8	10
8	8	4	12
1	10	10	3
18	5	14	7

x=

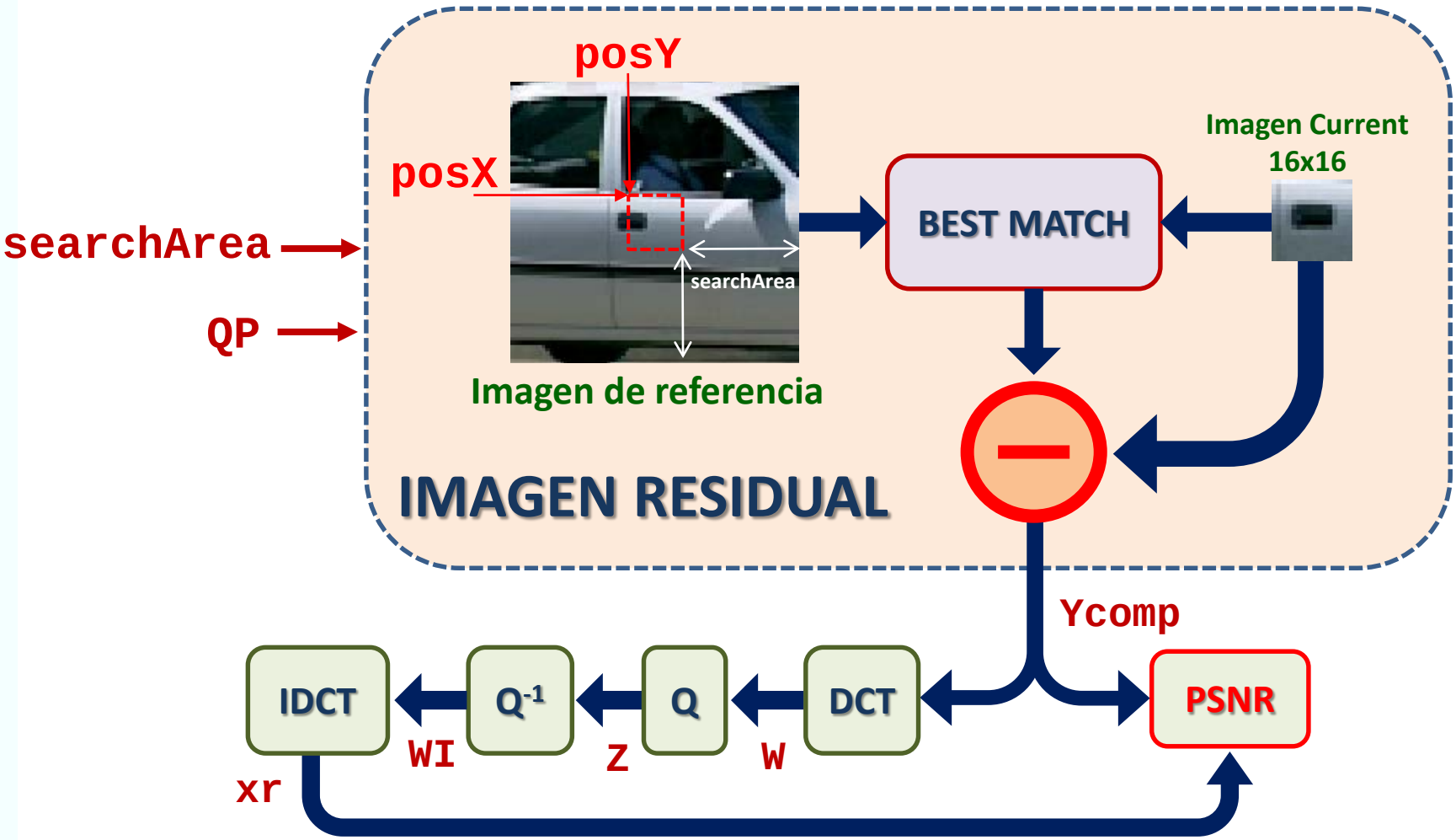
5	11	8	10
9	8	4	12
1	10	11	4
19	6	15	7

```
Matlab: xr=DCT_inverse (WI)
```

- Proceso completo:

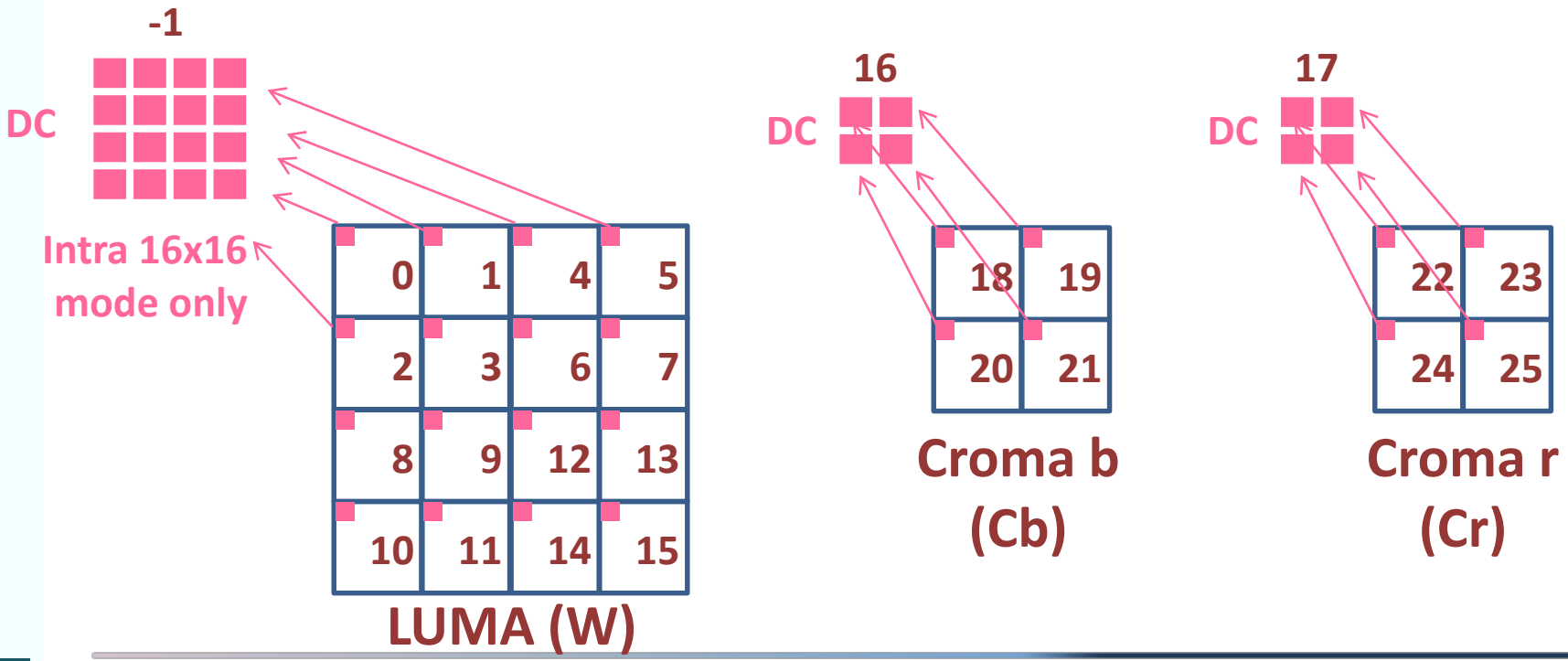
```
Matlab: xr=quantization_complete(x,10,0)
```

EjemploQuantificacionInter.m



TRANSFORMADAS DEL H.264

- El H.264 usa tres tipos de transformada para codificar el bloque residual:
  1. Transformada entera DCT-based de bloques 4x4.
  2. Transformada Hadamard de 2x2 de los coeficientes DC de las cromas.
  3. Solo para intra 16x16, transformada Hadamard de 4x4 de los coeficientes DC de la luma.



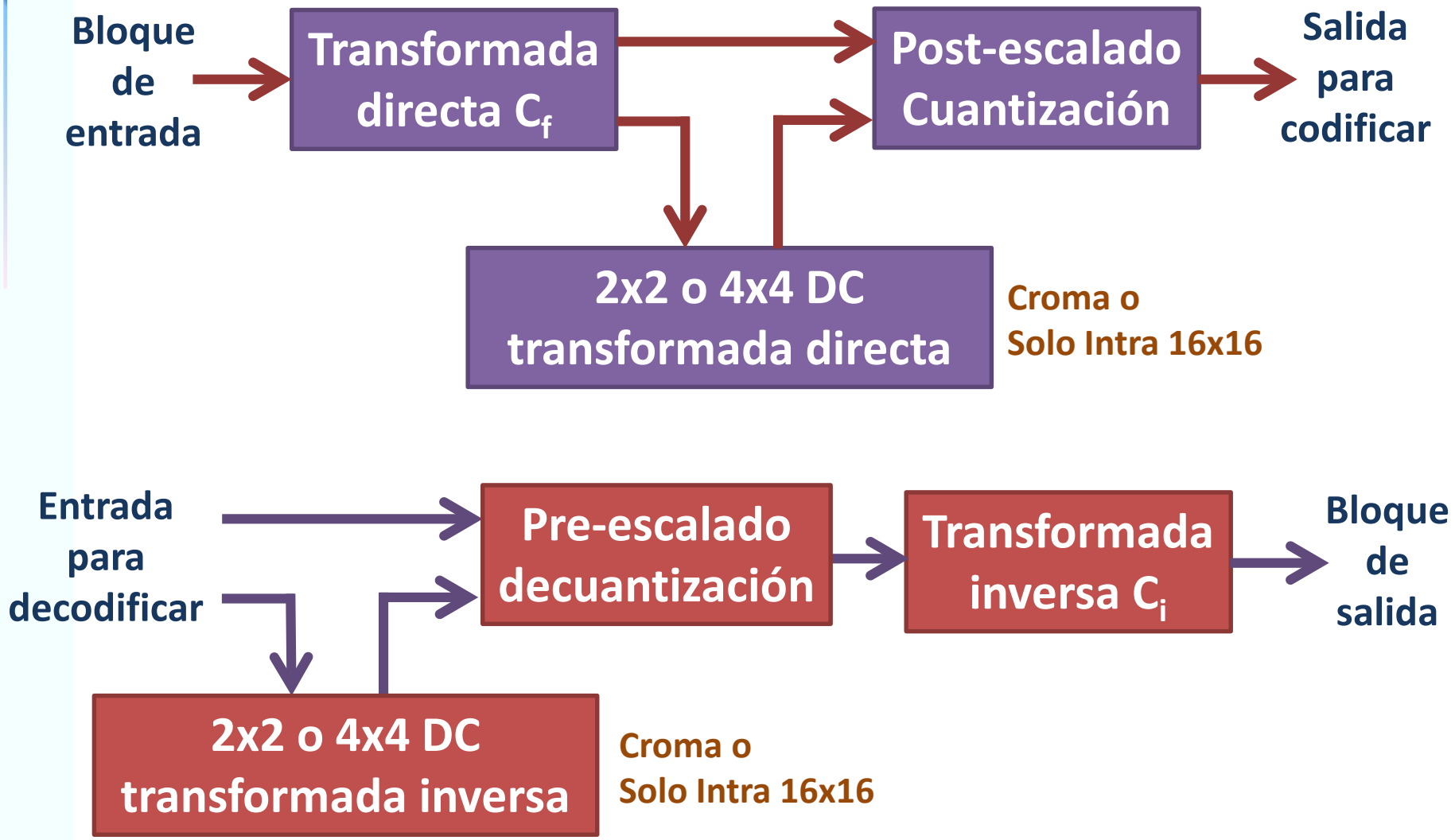
- En intra 16x16

$$\text{Luma DC } Y_{DC} = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} W_{DC} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) / 2$$

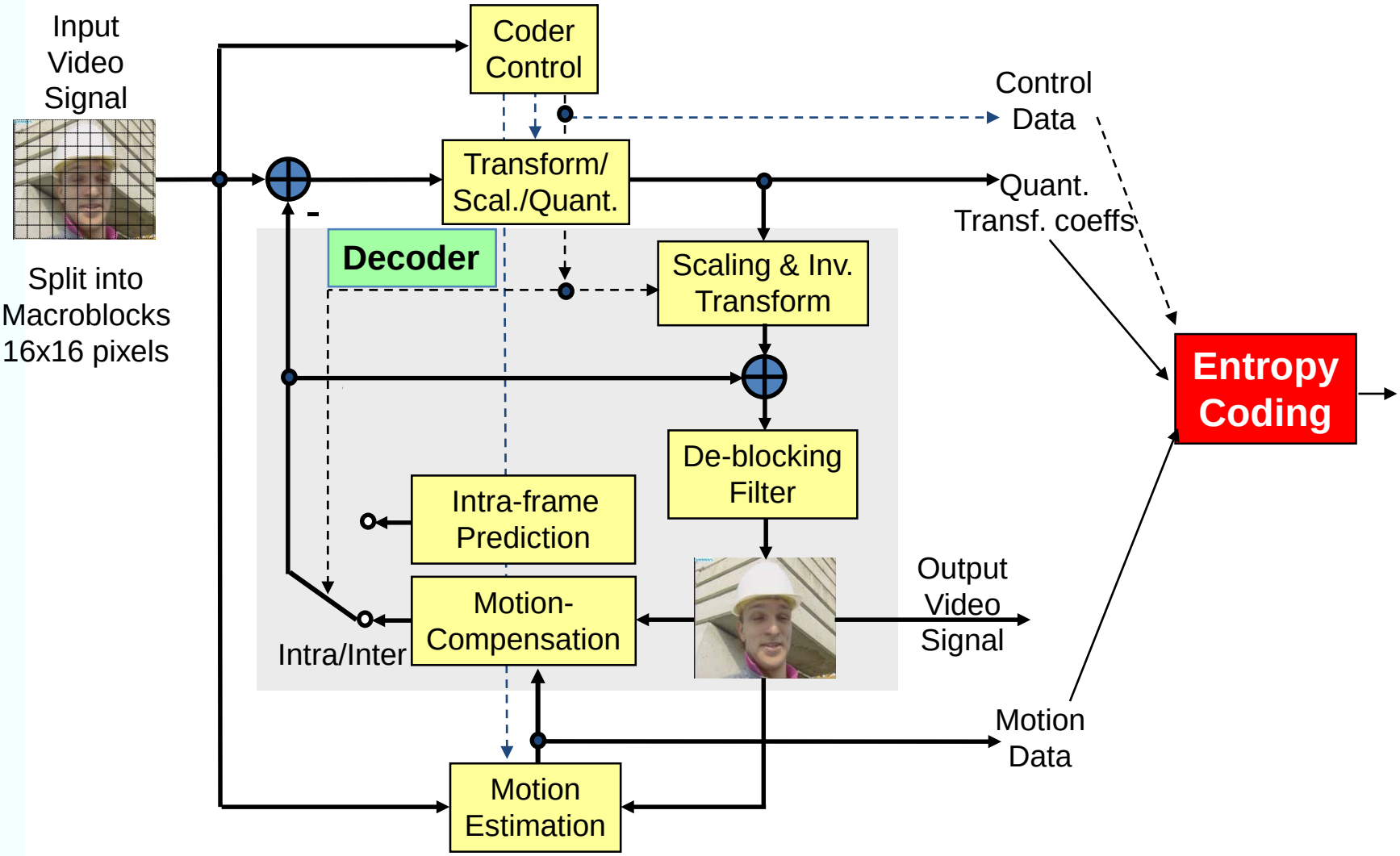
$$\text{Croma DC } Cbr_{DC} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} Cb \text{ o } Cr \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

- Orden de transmisión de coeficientes
  1. En caso intra 16x16, coeficientes Y_{DC}. Etiquetados como -1.
  2. Coeficientes luma de acuerdo al orden indicado (0 a 15).
  3. Coeficientes DC de las cromas (16 y 17).
  4. Resto de coeficientes de las cromas (10 a 25).

Diagrama de flujo de codificación de transformada



CODIFICACIÓN ENTRÓPICA



- **El H.264 utiliza para codificar los símbolos (datos) y generar una secuencia de bits que pueden ser almacenados o transmitidos, diferentes métodos:**

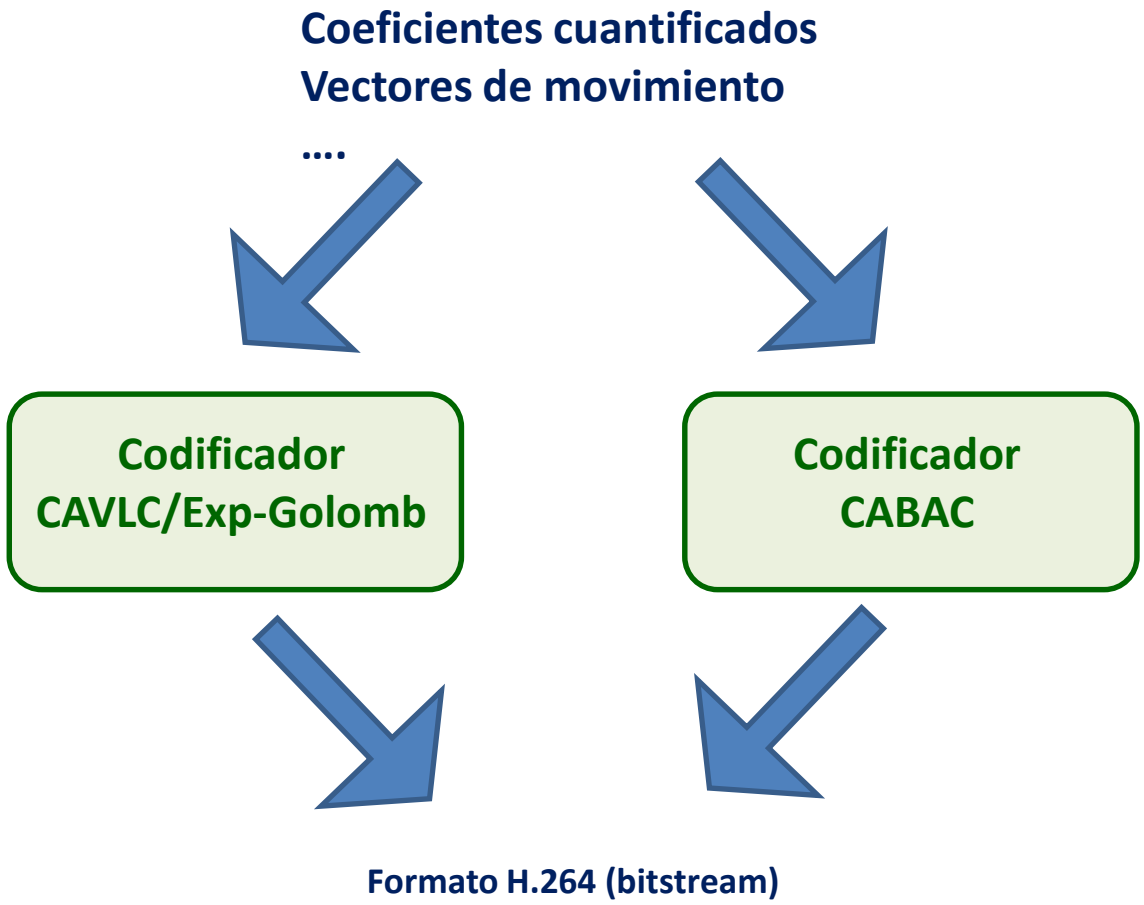
  - **Código de longitud fija. Cualquier símbolo es convertido hacia un código binario de longitud especificada (n bits).**

  - **Código de longitud variable Exponential-Golomb . Cada símbolo es representado por un código de palabra (*codeword*) de longitud variable. El código de palabra es más corto según la frecuencia de ocurrencia de un símbolo.**

  - ***Context-adaptative variable lenght coding* o CAVLC. Ha sido especialmente diseñado para codificar coeficientes de la transformada y la longitud del código es elegido dependiendo de la frecuencia de aparición de los coeficientes siguiendo un proceso adaptativo.**

  - **CABAC o *context-adaptative binary arithmetic coding*. Los modelos de probabilidad son actualizados de acuerdo a las estadísticas de los símbolos previamente codificados.**

OPCIONES DE CODIFICACIÓN ENTRÓPICA



Codificación Exp-Golomb

- Permite codificar eficientemente datos con probabilidad variables. Se asigna un *codewords* corto a los símbolos con mayor frecuencia de aparición y largos *codewords* a los menos comunes.

Exp-Golomb codewords

Code_num	Codeword
0	1
1	010
2	011
3	00100
4	00101
5	00110
6	00111
7	0001000
8	0001001
…	……

Simetría

- El *Exp-Golomp codeword* tiene la siguiente estructura:

[prefijo de M ceros] 1 [INFO]

$$M = \text{floor}\left(\log_2\left(\text{code\_num}+1\right)\right)$$

$$INFO = \text{code\_num} + 1 - 2^M$$

Ejemplo: code_num=107

$$\log_2\left(107+1\right) = 6.754 \Rightarrow M = \text{floor}\left(6.754\right) = 6$$

$$INFO = 107 + 1 - 2^6 = 44_{10} = 101100_2$$

$$\text{codeword} = 00000\,1\,101100$$

Ejemplo: codeword=0000000 1 1100011

$$M = 7$$

$$INFO = 1100011_2 = 99_{10}$$

$$\text{code\_num} = 2^7 + 99 - 1 = 226$$

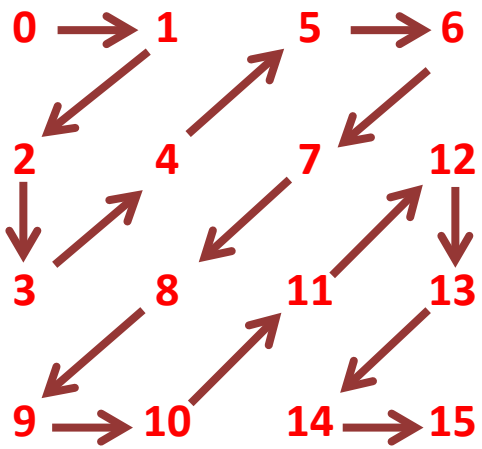
Codificación CAVLC

- La codificación CAVLC es diseñada para aprovechar las características de los bloques 4x4 cuantificados:
  - Después de realizar la transformada y cuantificación a un bloque 4x4, la mayoría de los coeficientes son ceros.
  - El valor de estos coeficientes no-ceros tienden a ser de mayor a menor valor si son ordenados en zig-zag.

❖ Ejemplo de codificación CAVLC de un bloque de coeficientes cuantificados:

0	3	-1	0
0	-1	1	0
1	0	0	0
0	0	0	0

➢ Secuencia de ordenación de los coeficientes en zig-zag:



Secuencia re-ordenada para codificar:

0 3 0 1 -1 -1 0 1 0 0 0 0 0 0 0 0

Cola de ceros

Value	Code	Comments
NumCoeff=5, T1s=3	0000100	Use Num-VLC0
sign of T1 (1)	0	Starting at highest frequency
sign of T1(-1)	1	
sign of T1(-1)	1	
Level= +1	1	Use Lev-VLC0
Level= +3	0010	Use Lev-VLC1
TotZeros=3	0111	Also depends on NumCoeff
ZerosLeft=3;RunBefore=1	10	RunBefore of the 1st Coeff
ZerosLeft=2;RunBefore=0	1	RunBefore of the 2nd Coeff
ZerosLeft=2;RunBefore=0	1	RunBefore of the 3rd Coeff
ZerosLeft=2;RunBefore=1	01	RunBefore of the 4th Coeff
ZerosLeft=1;RunBefore=1		No code required; last coeff

El bitstream transmitido para este bloque es: 000010001110010111101101

Secuencia para decodificar: 000010001110010111101101

Code	Value	Output Array	Comments
0000100	NumCoeff=5, T1s=3	Empty	
0	+	1	T1 sign
1	-	-1,1	T1 sign
1	-	-1,-1,1	T1 sign
1	+1	1,-1,-1,1	level value
0010	+3	+3,1,-1,-1,1	level value
0111	TotZeros=3	+3,1,-1,-1,1	
10	RunBefore=1	+3,1,-1,-1,0,1	RunBefore of the 1st Coeff
1	RunBefore=0	+3,1,-1,-1,0,1	RunBefore of the 2nd Coeff
1	RunBefore=0	+3,1,-1,-1,0,1	RunBefore of the 3rd Coeff
01	RunBefore=1	+3,0,1,-1,-1,0,1	RunBefore of the 4th Coeff
		0,+3,0,1,-1,-1,0,1	ZeroLeft=1

Ejemplo de codificación CAVLC