Video compression design, analysis, consulting and research

White Paper:

4x4 Transform and Quantization in H.264/AVC

Version 1.1a Revised April 2009

H.264 Transform and Quantization

1 Overview

In an H.264/AVC codec, macroblock data are transformed and quantized prior to coding and rescaled and inverse transformed prior to reconstruction and display (Figure 1). Several transforms are specified in the H.264 standard: a 4x4 "core" transform, 4x4 and 2x2 Hadamard transforms and an 8x8 transform (High profiles only).

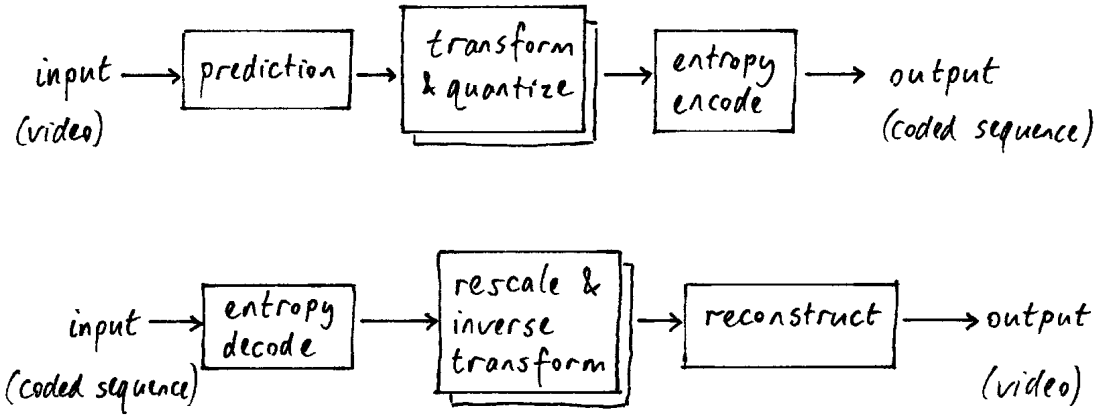

Figure 1 Transform and quantization in an H.264 codec

This paper describes a derivation of the forward and inverse transform and quantization processes applied to 4x4 blocks of luma and chroma samples in an H.264 codec. The transform is a scaled approximation to a 4x4 Discrete Cosine Transform that can be computed using simple integer arithmetic. A normalisation step is incorporated into forward and inverse quantization operations.

2 The H.264 transform and quantization process

The inverse transform and re-scaling processes, shown in Figure 2, are defined in the H.264/AVC standard. Input data (quantized transform coefficients) are re-scaled (a combination of inverse quantization and normalisation, see later). The re-scaled values are transformed using a "core" inverse transform. In certain cases, an inverse transform is applied to the DC coefficients prior to re-scaling. These processes (or their equivalents) must be implemented in every H.264-compliant decoder. The corresponding forward transform and quantization processes are not standardized but suitable processes can be derived from the inverse transform / rescaling processes (Figure 3).

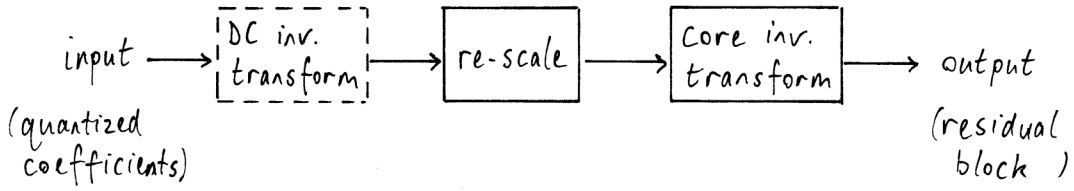

Figure 2 Re-scaling and inverse transform

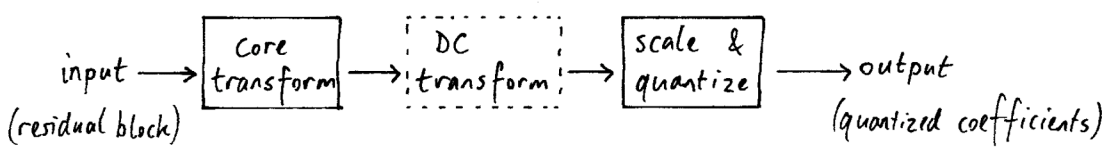

Figure 3 Forward transform and quantization

3 Developing the forward transform and quantization process

The basic 4x4 transform used in H.264 is a scaled approximate Discrete Cosine Transform (DCT). The transform and quantization processes are structured such that computational complexity is minimized. This is achieved by reorganising the processes into a core part and a scaling part.

Consider a block of pixel data that is processed by a two-dimensional Discrete Cosine Transform (DCT) followed by quantization (dividing by a quantization step size, Q_{step} , then rounding the result) (Figure 4a).

Rearrange the DCT process into a core transform ($\mathbf{C_f}$) and a scaling matrix ($\mathbf{S_f}$) (Figure 4b).

Scale the quantization process by a constant (2^{15}) and compensate by dividing and rounding the final result (Figure 4c).

Combine $\mathbf{S_f}$ and the quantization process into $\mathbf{M_f}$ (Figure 4d), where:

$$\mathbf{M}_f \approx \frac{\mathbf{S}_f \cdot 2^{15}}{Q_{step}}$$

Equation 1

(The reason for the use of ≈ will be explained later).

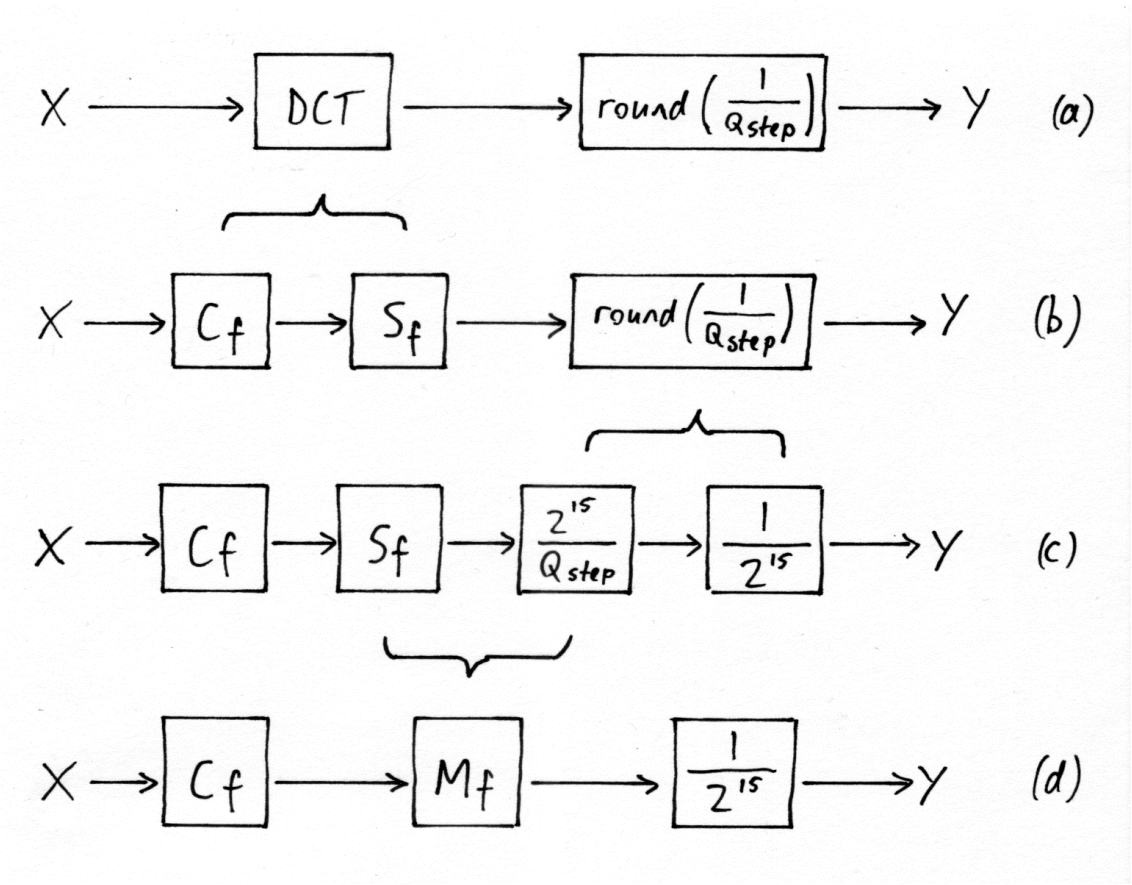

Figure 4 Development of the forward transform and quantization process

4 Developing the rescaling and inverse transform process

Consider a re-scaling (or "inverse quantization") operation followed by a two-dimensional inverse DCT (IDCT) (Figure 5a).

Rearrange the IDCT process into a core transform ($\mathbf{C_i}$) and a scaling matrix ($\mathbf{S_i}$) (Figure 5b).

Scale the re-scaling process by a constant (2^6) and compensate by dividing and rounding the final result (Figure 5c)[1].

Combine the re-scaling process and **S** into $\mathbf{V_i}$ (Figure 5d), where:

$$\mathbf{V}_i = \mathbf{S}_i \cdot 2^6 \cdot Q_{step}$$

Equation 2

[1] This rounding operation need not be to the nearest integer.

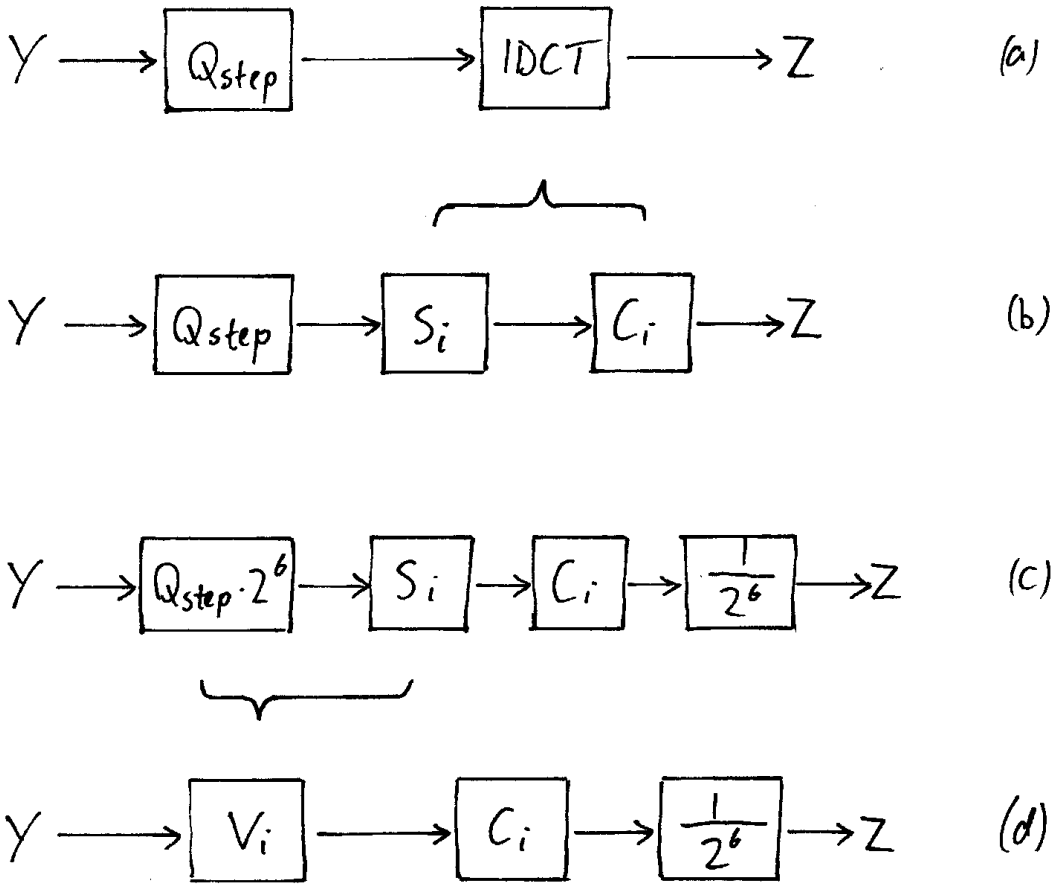

Figure 5 Development of the rescaling and inverse transform process

5 Developing C_f and S_f (4x4 blocks)

Consider a 4x4 two-dimensional DCT of a block **X**:

$$\mathbf{Y} = \mathbf{A} \cdot \mathbf{X} \cdot \mathbf{A}^T$$

Equation 3

Where · indicates matrix multiplication and:

$$\mathbf{A} = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix}, \quad a = \frac{1}{2}, \quad b = \sqrt{\frac{1}{2}}\cos\frac{\pi}{8} = 0.6532\ldots, \quad c = \sqrt{\frac{1}{2}}\cos\frac{3\pi}{8} = 0.2706\ldots$$

The rows of **A** are orthogonal and have unit norms (i.e. the rows are orthonormal). Calculation of Equation 3 on a practical processor requires approximation of the irrational numbers *b* and *c*. A fixed-point approximation is equivalent to scaling each row of **A** and rounding to the nearest integer. Choosing a particular approximation (multiply by 2.5 and round) gives $\mathbf{C_f}$ **:**

$$\mathbf{C}_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}$$

This approximation is chosen to minimise the complexity of implementing the transform (multiplication by $\mathbf{C_f}$ requires only additions and binary shifts) whilst maintaining good compression performance.

The rows of $\mathbf{C_f}$ have different norms. To restore the orthonormal property of the original matrix **A**, multiply all the values c_{ij} in row r by $\frac{1}{\sqrt{\sum_j c_{rj}^2}}$:

$$\mathbf{A_1} = \mathbf{C_f} \bullet \mathbf{R_f} \qquad \text{where } \mathbf{R}_f = \begin{bmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ 1/\sqrt{10} & 1/\sqrt{10} & 1/\sqrt{10} & 1/\sqrt{10} \\ 1/2 & 1/2 & 1/2 & 1/2 \\ 1/\sqrt{10} & 1/\sqrt{10} & 1/\sqrt{10} & 1/\sqrt{10} \end{bmatrix}$$

• denotes element-by-element multiplication (Hadamard-Schur product[2]). Note that the new matrix $\mathbf{A_1}$ is orthonormal.

The two-dimensional transform (Equation 3) becomes:

$$\mathbf{Y} = \mathbf{A_1} \cdot \mathbf{X} \cdot \mathbf{A_1}^T = [\mathbf{C_f} \bullet \mathbf{R_f}] \cdot \mathbf{X} \cdot [\mathbf{C_f}^T \bullet \mathbf{R_f}^T]$$

Rearranging:

$$\begin{aligned} \mathbf{Y} &= [\mathbf{C_f} \cdot \mathbf{X} \cdot \mathbf{C_f}^T] \bullet [\mathbf{R_f} \bullet \mathbf{R_f}^T] \\ &= [\mathbf{C_f} \cdot \mathbf{X} \cdot \mathbf{C_f}^T] \bullet \mathbf{S_f} \end{aligned}$$

Where

$$\mathbf{S}_f = \mathbf{R}_f \bullet \mathbf{R}_f^T = \begin{bmatrix} 1/4 & 1/2\sqrt{10} & 1/4 & 1/2\sqrt{10} \\ 1/2\sqrt{10} & 1/10 & 1/2\sqrt{10} & 1/10 \\ 1/4 & 1/2\sqrt{10} & 1/4 & 1/2\sqrt{10} \\ 1/2\sqrt{10} & 1/10 & 1/2\sqrt{10} & 1/10 \end{bmatrix}$$

[2] $\mathbf{P = Q \bullet R}$ means that each element $p_{ij} = q_{ij} \cdot r_{ij}$

6 Developing C_i and S_i (4x4 blocks)

Consider a 4x4 two-dimensional IDCT of a block **Y**:

$$\mathbf{Z} = \mathbf{A}^T \cdot \mathbf{Y} \cdot \mathbf{A}$$

Equation 4

Where

$$\mathbf{A} = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix}, \quad \begin{aligned} a &= 1/2 \\ b &= \sqrt{1/2}\cos\pi/8 = 0.6532.... \\ c &= \sqrt{1/2}\cos 3\pi/8 = 0.2706.... \end{aligned}$$ as before.

Choose a particular approximation by scaling each row of **A** and rounding to the nearest 0.5, giving $\mathbf{C_i}$:

$$\mathbf{C}_i = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix}$$

The rows of $\mathbf{C_i}$ are orthogonal but have non-unit norms. To restore orthonormality, multiply all the values c_{ij} in row r by $\dfrac{1}{\sqrt{\sum_j c_{rj}^2}}$:

$$\mathbf{A_2} = \mathbf{C_i} \bullet \mathbf{R_i} \qquad \text{where } \mathbf{R}_i = \begin{bmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ \sqrt{2/5} & \sqrt{2/5} & \sqrt{2/5} & \sqrt{2/5} \\ 1/2 & 1/2 & 1/2 & 1/2 \\ \sqrt{2/5} & \sqrt{2/5} & \sqrt{2/5} & \sqrt{2/5} \end{bmatrix}$$

The two-dimensional inverse transform (Equation 4) becomes:

$$\mathbf{Z} = \mathbf{A_2}^T \cdot \mathbf{Y} \cdot \mathbf{A_2} = [\mathbf{C_i}^T \bullet \mathbf{R_i}^T] \cdot \mathbf{Y} \cdot [\mathbf{C_i} \bullet \mathbf{R_i}]$$

Rearranging:

$$\begin{aligned} \mathbf{Z} &= [\mathbf{C_i}^T] \cdot [\mathbf{Y} \bullet \mathbf{R_i}^T \bullet \mathbf{R_i}] \cdot [\mathbf{C_i}] \\ &= [\mathbf{C_i}^T] \cdot [\mathbf{Y} \bullet \mathbf{S_i}] \cdot [\mathbf{C_i}] \end{aligned}$$

Where

$$\mathbf{S}_i = \mathbf{R}_i^T \bullet \mathbf{R}_i = \begin{bmatrix} 1/4 & 1/\sqrt{10} & 1/4 & 1/\sqrt{10} \\ 1/\sqrt{10} & 2/5 & 1/\sqrt{10} & 2/5 \\ 1/4 & 1/\sqrt{10} & 1/4 & 1/\sqrt{10} \\ 1/\sqrt{10} & 2/5 & 1/\sqrt{10} & 2/5 \end{bmatrix}$$

The core inverse transform $\mathbf{C_i}$ and the rescaling matrix $\mathbf{V_i}$ are defined in the H.264 standard. Hence we now develop $\mathbf{V}_i$ and will then derive $\mathbf{M_f}$.

7 Developing $\mathbf{V_i}$

From Equation 2, $\mathbf{V_i} = \mathbf{S_i} \cdot Q_{step} \cdot 2^6$

H.264 supports a range of quantization step sizes Q_{step}. The precise step sizes are not defined in the standard, rather the scaling matrix $\mathbf{V_i}$ is specified. Q_{step} values corresponding to the entries in $\mathbf{V_i}$ are shown in the following Table.

QP	Q_{step}
0	0.625
1	0.702..
2	0.787..
3	0.884..
4	0.992..
5	1.114..
6	1.250
…	…
12	2.5
…	…
18	5.0
…	…
48	160
…	…
51	224

The ratio between successive Q_{step} values is chosen to be $\sqrt[6]{2} = 1.2246...$ so that Q_{step} doubles in size when QP increases by 6. Any value of Q_{step} can be derived from the first 6 values in the table (QP0 – QP5) as follows:

$$Q_{step}(QP) = Q_{step}(QP\%6) \cdot 2^{floor(QP/6)}$$

The values in the matrix $\mathbf{V_i}$ depend on Q_{step} (hence QP) and on the scaling factor matrix $\mathbf{S_i}$. These are shown for QP 0 to 5 in the following Table.

QP	$Q_{step} \cdot 2^6$	$\mathbf{V_i}$ = round ($\mathbf{S_i} \cdot Q_{step} \cdot 2^6$)
0	40	$\begin{bmatrix} 10 & 13 & 10 & 13 \\ 13 & 16 & 13 & 16 \\ 10 & 13 & 10 & 13 \\ 13 & 16 & 13 & 16 \end{bmatrix}$
1	44.898	$\begin{bmatrix} 11 & 14 & 11 & 14 \\ 14 & 18 & 14 & 18 \\ 11 & 14 & 11 & 14 \\ 14 & 18 & 14 & 18 \end{bmatrix}$
2	50.397	$\begin{bmatrix} 13 & 16 & 13 & 16 \\ 16 & 20 & 16 & 20 \\ 13 & 16 & 13 & 16 \\ 16 & 20 & 16 & 20 \end{bmatrix}$
3	56.569	$\begin{bmatrix} 14 & 18 & 14 & 18 \\ 18 & 23 & 18 & 23 \\ 14 & 18 & 14 & 18 \\ 18 & 23 & 18 & 23 \end{bmatrix}$
4	63.496	$\begin{bmatrix} 16 & 20 & 16 & 20 \\ 20 & 25 & 20 & 25 \\ 16 & 20 & 16 & 20 \\ 20 & 25 & 20 & 25 \end{bmatrix}$
5	71.272	$\begin{bmatrix} 18 & 23 & 18 & 23 \\ 23 & 29 & 23 & 29 \\ 18 & 23 & 18 & 23 \\ 23 & 29 & 23 & 29 \end{bmatrix}$

For higher values of QP, the corresponding values in $\mathbf{V_i}$ are doubled (i.e. $\mathbf{V_i}$ (QP=6) = 2$\mathbf{V_i}$(QP=0) , etc).

Note that there are only three unique values in each matrix $\mathbf{V_i}$. These three values are defined as a table of values v in the H.264 standard, for QP=0 to QP=5 :

Table 1 Matrix *v* defined in H.264 standard

QP	*v* (r, 0): $\mathbf{V_i}$ positions (0,0), (0,2), (2,0), (2,2)	*v* (r, 1): $\mathbf{V_i}$ positions (1,1), (1,3), (3,1), (3,3)	*v* (r, 2): Remaining $\mathbf{V_i}$ positions
0	10	16	13
1	11	18	14
2	13	20	16
3	14	23	18
4	16	25	20
5	18	29	23

Hence for QP values from 0 to 5, $\mathbf{V_i}$ is obtained as:

$$V_i = \begin{bmatrix} v(QP,0) & v(QP,2) & v(QP,0) & v(QP,2) \\ v(QP,2) & v(QP,1) & v(QP,2) & v(QP,1) \\ v(QP,0) & v(QP,2) & v(QP,0) & v(QP,2) \\ v(QP,2) & v(QP,1) & v(QP,2) & v(QP,1) \end{bmatrix}$$

Denote this as:

$\mathbf{V_i} = v(\text{QP, n})$
Where *v* (r,n) is row r, column n of *v*.

For larger values of QP (QP>5), index the row of array *v* by QP%6 and then multiply by $2^{\text{floor(QP/6)}}$. In general:

$$\mathbf{V_i} = v\,(\text{QP\%6,n}) \cdot 2^{\text{floor(QP/6)}}$$

The complete inverse transform and scaling process (for 4x4 blocks in macroblocks excluding 16x16-Intra mode) becomes:

$$\mathbf{Z} = \text{round}\left([\mathbf{C_i}^{\mathbf{T}}] \cdot [\mathbf{Y} \bullet v\,(\text{QP\%6,n}) \cdot 2^{\text{floor(QP/6)}}] \cdot [\mathbf{C_i}] \cdot \frac{1}{2^6} \right)$$

(Note: rounded division by 2^6 can be carried out by adding an offset and right-shifting by 6 bit positions).

8 Deriving $\mathbf{M_f}$

Combining Equation 1 and Equation 2:

$$\mathbf{M}_f \approx \frac{\mathbf{S}_i \cdot \mathbf{S}_f \cdot 2^{21}}{\mathbf{V}_i}$$

$\mathbf{S_i}$, $\mathbf{S_f}$ are known and $\mathbf{V}_i$ is defined as described in the previous section. Define $\mathbf{M}_f$ as:

$$\mathbf{M}_f = \text{round}\left(\frac{\mathbf{S}_i \cdot \mathbf{S}_f \cdot 2^{21}}{\mathbf{V}_i}\right)$$

$$\mathbf{S}_i \cdot \mathbf{S}_f \cdot 2^{21} = \begin{bmatrix} 131072 & 104857.6 & 131072 & 104857.6 \\ 104857.6 & 83886.1 & 104857.6 & 83886.1 \\ 131072 & 104857.6 & 131072 & 104857.6 \\ 104857.6 & 83886.1 & 104857.6 & 83886.1 \end{bmatrix}$$

The entries in matrix $\mathbf{M_f}$ may be calculated as follows (Table 2):

Table 2 Tables *v* and *m*

QP	*v* (r, 0): $\mathbf{V_i}$ positions (0,0), (0,2), (2,0), (2,2)	*v* (r, 1): $\mathbf{V_i}$ positions (1,1), (1,3), (3,1), (3,3)	*v* (r, 2): Remaining $\mathbf{V_i}$ positions	*m* (r, 0): $\mathbf{M_f}$ positions (0,0), (0,2), (2,0), (2,2)	*m* (r, 1): $\mathbf{M_f}$ positions (1,1), (1,3), (3,1), (3,3)	*m* (r,2): Remaining $\mathbf{M_f}$ positions
0	10	16	13	13107	5243	8066
1	11	18	14	11916	4660	7490
2	13	20	16	10082	4194	6554
3	14	23	18	9362	3647	5825
4	16	25	20	8192	3355	5243
5	18	29	23	7282	2893	4559

Hence for QP values from 0 to 5, $\mathbf{M_f}$ can be obtained from *m* , the last three columns of Table 2:

$$Mf = \begin{bmatrix} m(QP,0) & m(QP,2) & m(QP,0) & m(QP,2) \\ m(QP,2) & m(QP,1) & m(QP,2) & m(QP,1) \\ m(QP,0) & m(QP,2) & m(QP,0) & m(QP,2) \\ m(QP,2) & m(QP,1) & m(QP,2) & m(QP,1) \end{bmatrix}$$

Denote this as:

$\mathbf{M_f}$ = *m*(QP, n)
Where *m* (r,n) is row r, column n of *m*.

For larger values of QP (QP>5), index the row of array *m* by QP%6 and then divide by $2^{\text{floor}(QP/6)}$. In general:

$$\mathbf{M_f} = m\,(QP\%6,n)/\ 2^{\text{floor}(QP/6)}$$

Where *m* (r,n) is row r, column n of *m*.

The complete forward transform, scaling and quantization process (for 4x4 blocks and for modes excluding 16x16-Intra) becomes:

$$\mathbf{Y} = \text{round}\left([\mathbf{C_f}]\cdot[\mathbf{Y}]\cdot[\mathbf{C_f}^{\mathbf{T}}] \quad \bullet\ m\,(\text{QP}\%6,\text{n})/\ 2^{\text{floor}(\text{QP}/6)} \right] \cdot \frac{1}{2^{15}} \ )$$

(Note: rounded division by 2^{15} may be carried out by adding an offset and right-shifting by 15 bit positions).

9 Further reading

ITU-T Recommendation H.264, *Advanced Video Coding for Generic Audio-Visual Services*, November 2007.

H. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, *Low-complexity transform and quantization in H.264/AVC*, IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, pp. 598–603, July 2003.

T. Wiegand, G.J. Sullivan, G. Bjontegaard, A. Luthra, *Overview of the H.264/AVC video coding standard*, IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, No. 7. (2003), pp. 560-576.

I. Richardson, *The H.264 Advanced Video Compression Standard*, to be published in late 2009.

See *http://www.vcodex.com/links.html* for links to further resources on H.264 and video compression.

Acknowledgement

I would like to thank Gary Sullivan for suggesting a treatment of the H.264 transform and quantization processes along these lines and for his helpful comments on earlier drafts of this document.

About the author

As a researcher, consultant and author working in the field of video compression (video coding), my books on video codec design and the MPEG-4 and H.264 standards are widely read by engineers, academics and managers. I advise companies on video coding standards, design and intellectual property and lead the Centre for Video Communications Research at The Robert Gordon University in Aberdeen, UK and the Fully Configurable Video Coding research initiative.

Using the material in this document

Iain Richardson, 4x4 *Transform and Quantization in H.264/AVC*, VCodex Ltd White Paper, April 2009, http://www.vcodex.com/