Chapter 6
HEVC Transform and Quantization

Madhukar Budagavi, Arild Fuldseth, and Gisle Bjøntegaard

Abstract This chapter provides an overview of the transform and quantization design in HEVC. HEVC specifies two-dimensional transforms of various sizes from 4×4 to 32×32 that are finite precision approximations to the discrete cosine transform (DCT). In addition, HEVC also specifies an alternate 4×4 integer transform based on the discrete sine transform (DST) for use with 4×4 luma Intra prediction residual blocks. During the transform design, special care was taken to allow implementation friendliness, including limited bit depth, preservation of symmetry properties, embedded structure and basis vectors having almost equal norm. The HEVC quantizer design is similar to that of H.264/AVC where a quantization parameter (QP) in the range of 0–51 (for 8-bit video sequences) is mapped to a quantizer step size that doubles each time the QP value increases by 6. A key difference, however, is that the transform basis norm correction factors incorporated into the descaling matrices of H.264/AVC are no longer needed in HEVC simplifying the quantizer design. A QP value can be transmitted (in the form of delta QP) for a quantization group as small as 8×8 samples for rate control and perceptual quantization purposes. The QP predictor used for calculating the delta QP uses a combination of left, above and previous QP values. HEVC also supports frequency-dependent quantization by using quantization matrices for all transform block sizes. This chapter also provides an overview of the three special coding modes in HEVC (I_PCM mode, lossless mode, and transform skip mode) that modify the transform and quantization process by either skipping the transform or by skipping both transform and quantization.

M. Budagavi (✉)
Texas Instruments Inc., Dallas, TX, USA
e-mail: madhu072@yahoo.com

A. Fuldseth • G. Bjøntegaard
Cisco Systems Norway, 1366 Lysaker, Norway
e-mail: arild.fuldseth@cisco.com; gbjonteg@cisco.com

V. Sze et al. (eds.), *High Efficiency Video Coding (HEVC): Algorithms and Architectures*, Integrated Circuits and Systems, DOI 10.1007/978-3-319-06895-4_6,
© Springer International Publishing Switzerland 2014

6.1 Introduction

In the block-based hybrid video coding approach, transforms are applied to the residual signal resulting from inter- or intra-picture prediction as shown in Fig. 6.1. At the encoder, the residual signal of a picture is divided into square blocks of size $N \times N$ where $N = 2^M$ and M is an integer. Each residual block (U) is then input to a two-dimensional $N \times N$ forward transform. The two-dimensional transform can be implemented as a separable transform by applying an N-point one-dimensional transform to each row and each column separately. The resulting $N \times N$ transform coefficients (*coeff*) are then subject to quantization (which is equivalent to division by quantization step size *Qstep* and subsequent rounding) to obtain quantized transform coefficients (*level*). At the decoder, the quantized transform coefficients are then de-quantized (which is equivalent to multiplication by *Qstep*). Finally, a two-dimensional $N \times N$ separable inverse transform is applied to the de-quantized transform coefficients ($coeff_Q$) resulting in a residual block of quantized samples which is then added to the intra- or inter-prediction samples to obtain the reconstructed block.

Typically, the forward- and inverse transform matrices are transposes of each other and are designed to achieve near lossless reconstruction of the input residual block when concatenated without the intermediate quantization and de-quantization steps.

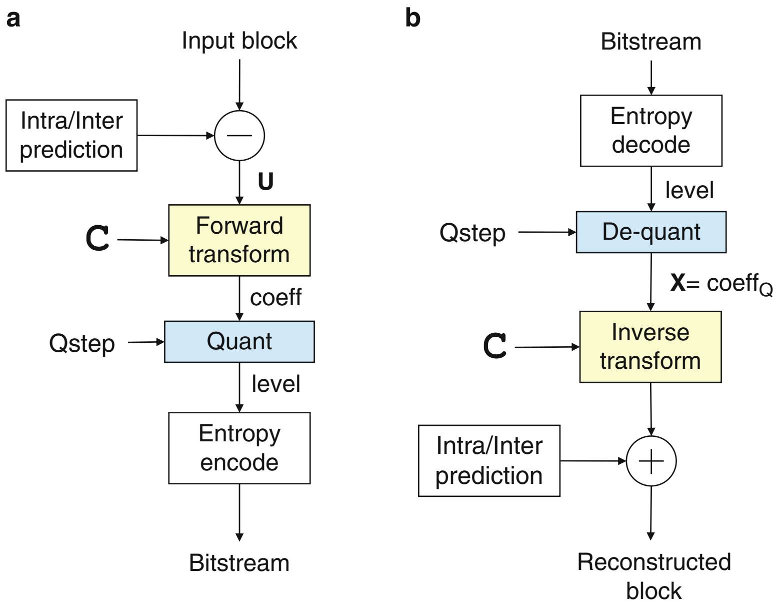

Fig. 6.1 Block-based hybrid video coding. (**a**) Encoder, (**b**) Decoder. **C** is the transform matrix and *Qstep* is the quantization step size. Reproduced with permission from [6]. © IEEE 2013

In video coding standards such as HEVC, the de-quantization process and inverse transforms are specified, while the forward transforms and quantization process are chosen by the implementer (subject to constraints on the bitstream).

This chapter is organized as follows. Section 6.2 describes the two transform types used in HEVC: the *core* transform based on the discrete cosine transform and the *alternate* transform based on the discrete sine transform. Design principles used to develop the transform are also highlighted to provide insight into the transform design process which considered both coding efficiency and complexity. In Sect. 6.3, the HEVC quantization process is described. Topics covered in this section include the actual quantization and de-quantization steps, quantization matrices, and quantization parameter derivation. Section 6.4 provides an overview of the three special coding modes in HEVC (I_PCM mode, Lossless mode, and Transform skip mode) that modify the transform and quantization process by either skipping the transform or by skipping both transform and quantization. Sections 6.5 and 6.6 provide complexity analysis and coding performance results respectively.

6.2 HEVC Transform[1]

The HEVC standard [16] specifies core transform matrices of size 4×4, 8×8, 16×16 and 32×32 to be used for two-dimensional transforms in the context of block-based motion-compensated video compression. Multiple transform sizes improve compression performance, but also increase the implementation complexity. Hence a careful design of the core transforms is needed.

HEVC specifies two-dimensional core transforms that are finite precision approximations to the inverse discrete cosine transform (IDCT) for all transform sizes. Note that because of the approximations, the HEVC core transforms are not the IDCT. The fact that an IDCT is not used does not necessarily make the HEVC core transforms imperfect. In fact, the finite precision approximations are desirable as explained in the next two paragraphs. The main purpose of the transform is to de-correlate the input residual block. The optimal de-correlating transform is the Karhunen–Loeve transform (KLT) [22] and not necessarily the DCT. This is especially true for the coding of 4×4 luma intra-prediction residual blocks where HEVC specifies an alternate 4×4 integer transform based on the discrete sine transform (DST) [24]. Note that only the inverse transforms are specified in the HEVC standard and the forward transforms are not. So an encoder may get additional coding efficiency benefits by using the actual inverse rather than the transpose of the inverse transform.

[1]Portions of this section are © 2013 IEEE. Reprinted, with permission, from M. Budagavi, A. Fuldseth, G. Bjøntegaard, V. Sze, M. Sadafale, "Core Transform Design in the High Efficiency Video Coding (HEVC) Standard," IEEE Journal of Selected Topics in Signal Processing, December 2013.

In the H.261, MPEG-1, H.262/MPEG-2, and H.263 video coding standards, an 8-point IDCT was specified with infinite precision. To ensure interoperability and to minimize drift between encoder and decoder implementations using finite precision, two features were included in the standards. First, block-level periodic intra refresh was mandatory. Second, a conformance test for the accuracy of the IDCT using a pseudo-random test pattern was specified.

In the H.264/MPEG-4 Advanced Video Coding (AVC) standard [15], the problem of encoder–decoder drift was solved by specifying integer valued 4×4 and 8×8 transform matrices. The transforms were designed as approximations to the IDCT with emphasis on minimizing the number of arithmetic operations. These transforms had large variations of the norm of the basis vectors. As a consequence of this, non-flat default de-quantization matrices were specified to compensate for the different norms of the basis vectors [20].

During the development of HEVC, several different approximations of the IDCT were studied for the core transform. The first version of the HEVC Test Model HM1 used the H.264/AVC transforms for 4×4 and 8×8 blocks and integer approximation of Chen's fast IDCT [7] for 16×16 and 32×32 blocks. The HM1 inverse transforms had the following characteristics [23, 28]:

- Non-flat de-quantization matrices for all transform sizes: While acceptable for small transform sizes, the implementation cost of using de-quantization matrices for larger transforms is high because of larger block sizes,
- Different architectures for different transform sizes: This leads to increased area since hardware sharing across different transform sizes is difficult,
- A 20-bit transpose buffer used for storing intermediate results after the first transform stage in 2D transform: An increased transpose buffer size leads to larger memory and memory bandwidth. In hardware, the transpose buffer area can be significant and comparable to transform logic area [30],
- Full factorization architecture requiring cascaded multipliers and intermediate rounding for 16- and 32-point transforms: This increases data path dependencies and impacts parallel processing performance. It also leads to increased bit width for multipliers and accumulators (32 bits and 64 bits respectively in software). In hardware, in addition to area increase, it also leads to increased circuit delay thereby limiting the maximum frequency at which the inverse transform block can operate.

To address the complexity concerns of the HM1 transforms, a matrix multiplication based core transform was proposed in [10] and eventually adopted as the HEVC core transform. The design goal was to develop a transform that was efficient to implement in both software on SIMD machines and in hardware. Alternative proposals to the HEVC core transform design can be found in [1, 9, 17].

The HEVC core transform matrices were designed to have the following properties [10]:

- Closeness to the IDCT
- Almost orthogonal basis vectors

- Almost equal norm of all basis vectors
- Same symmetry properties as the IDCT basis vectors
- Smaller transform matrices are embedded in larger transform matrices
- Eight-bit representation of transform matrix elements
- Sixteen-bit transpose buffer
- Multipliers can be represented using 16 bits or less with no cascaded multiplications or intermediate rounding
- Accumulators can be implemented using less than 32 bits

6.2.1 Discrete Cosine Transform

The N transform coefficients v_i of an N-point 1D DCT applied to the input samples u_i can be expressed as

$$v_i = \sum_{j=0}^{N-1} u_j c_{ij} \tag{6.1}$$

where $i = 0, \ldots, N-1$. Elements c_{ij} of the DCT transform matrix C are defined as

$$c_{ij} = \frac{P}{\sqrt{N}} \cos\left[\frac{\pi}{N}\left(j + \frac{1}{2}\right)i\right] \tag{6.2}$$

where $i,j = 0, \ldots, N-1$ and where P is equal to 1 and $\sqrt{2}$ for $i = 0$ and $i > 0$, respectively. Furthermore, the basis vectors $\mathbf{c}_i$ of the DCT are defined as $\mathbf{c}_i = [c_{i0}, \ldots, c_{i(N-1)}]^T$ where $i = 0, \ldots, N-1$.

The DCT has several properties that are considered useful both for compression efficiency and for efficient implementation [22].

1. The basis vectors are orthogonal, i.e. $\mathbf{c}_i^T \mathbf{c}_j = 0$ for $i \neq j$. This property is desirable for compression efficiency by achieving transform coefficients that are uncorrelated.
2. The basis vectors of the DCT have been shown to provide good energy compaction which is also desirable for compression efficiency.
3. The basis vectors of the DCT have equal norm, i.e. $\mathbf{c}_i^T \mathbf{c}_i = 1$ for $i = 0, \ldots, N-1$. This property is desirable for simplifying the quantization/de-quantization process. Assuming that equal frequency-weighting of the quantization error is desired, equal norm of the basis vectors eliminates the need for quantization/de-quantization matrices.
4. Let $N = 2^M$. The elements of a DCT matrix of size $2^M \times 2^M$ is a subset of the elements of a DCT matrix of size $2^{M+1} \times 2^{M+1}$. More specifically, the basis vectors of the smaller matrix is equal to the first half of the even basis vectors of

the larger matrix. This property is useful to reduce implementation costs as the same multipliers can be reused for various transform sizes.

5. The DCT matrix can be specified by using a small number of unique elements. By examining the elements c_{ij} of (6.2) it can be shown that the number of unique elements in a DCT matrix of size $2^M \times 2^M$ is equal to $2^M - 1$. As further elaborated in Sect. 6.2.4, this is particularly advantageous in hardware implementations.
6. The even basis vectors of the DCT are symmetric, while the odd basis vectors are anti-symmetric. This property is useful to reduce the number of arithmetic operations.
7. The coefficients of a DCT matrix have certain trigonometric relationships that allows for a reduction of the number of arithmetic operations beyond what is possible by exploiting the (anti-)symmetry properties. These properties can be utilized to implement fast algorithms such as the Chen's fast factorization [7].

6.2.2 Finite Precision DCT Approximations

The core transform matrices of HEVC are finite precision approximations of the DCT matrix. The benefit of using finite precision in a video coding standard is that the approximation to the real-valued DCT matrix is specified in the standard rather than being implementation dependent. This avoids encoder–decoder mismatch and drift caused by manufacturers implementing the IDCT with slightly different floating point representations. On the other hand, a disadvantage of using approximate matrix elements is that some of the properties of the DCT discussed in Sect. 6.2.1 may not be satisfied anymore. More specifically, there is a trade-off between the computational cost associated with using high bit-depth for the matrix elements and the degree to which some of the conditions of Sect. 6.2.1 are satisfied.

A straightforward way of determining integer approximations to the DCT matrix elements is to scale each matrix element with some large number (typically between 2^5 and 2^{16}) and then round to the closest integer. However, this approach does not necessarily result in the best compression performance. As shown in Sect. 6.2.3, for a given bit-depth of the matrix elements, a different strategy for approximating the DCT matrix elements results in a different trade-off between some of the properties of Sect. 6.2.1.

6.2.3 HEVC Core Transform Design Principles

The DCT approximations used for the core transforms of HEVC were chosen according to the following principles. First, properties 4–6 of Sect. 6.2.1 were satisfied without any compromise. This choice ensures that several implementation friendly aspects of the DCT are preserved. Second, for properties 1–3 and 7 of Sect. 6.2.1, there were trade-offs between the number of bits used to represent each matrix element and the degree by which each of the properties were satisfied.

Table 6.1 Comparison of transform design methods

	HEVC core transforms	Scaling and rounding
Orthogonality	$o_{ij} < 0.0029$	$o_{ij} < 0.0037$
Closeness to DCT	$m_{ij} < 0.0213$	$m_{ij} < 0.0077$
Norm measure	$n_i < 0.0014$	$n_i < 0.0109$

To measure the degree of approximation for properties 1–3 of Sect. 6.2.1, the following measures are defined for an integer N-point DCT approximation with scaled matrix elements equal to d_{ij} and basis vectors equal to $\mathbf{d}_i = [d_{i0}, \ldots, d_{i(N-1)}]^T$ where $i = 0, \ldots, N-1$.

1. Orthogonality measure: $o_{ij} = \mathbf{d}_i^T \mathbf{d}_j / \mathbf{d}_0^T \mathbf{d}_0, i \neq j$
2. Closeness to DCT measure: $m_{ij} = |\alpha c_{ij} - d_{ij}| / d_{00}$
3. Norm measure: $n_i = |1 - \mathbf{d}_i^T \mathbf{d}_i / \mathbf{d}_0^T \mathbf{d}_0|$

where $i,j = 0, \ldots, N-1$, c_{ij} are the DCT matrix elements of (6.2), and the scale factor α is defined as $d_{00}N^{1/2}$.

As a result of careful investigation, it was decided to represent each matrix coefficient with 8 bit (including sign bit), and to choose the elements of the first basis vector to be equal to 64 (i.e. $d_{0j} = 64, j = 0, \ldots, N-1$). Note that this results in a scale factor of $2^{6+M/2}$ for the HEVC transform matrix when compared to the orthonormal DCT. The remaining matrix elements were hand-tuned (within the constraints of properties 4–6 of Sect. 6.2.1) to achieve a good balance between properties 1–3 of Sect. 6.2.1. The hand-tuning was performed as follows. First, the real-valued scaled DCT matrix elements, αc_{ij}, were derived. Next, for each unique number in the resulting matrices, each integer value in the interval $[-1.5, 1.5]$ around αc_{ij} was examined and the resulting values of o_{ij}, m_{ij}, and n_i were calculated. Since there are only 31 unique numbers in the transform matrices (see Sect. 6.2.4), various permutations can be examined systematically (although not exhaustively). The final integer matrix elements were chosen to give a good compromise between all measures o_{ij}, m_{ij}, and n_i. The resulting worst case values of o_{ij}, m_{ij}, and n_i are shown in the second column of Table 6.1. The norm was considered to be sufficiently close to 1 (i.e. the norm measure n_i is sufficiently close to 0) to justify not using a non-flat default de-quantization matrix in HEVC (i.e. all transform coefficients scaled equally).

For comparison purposes, the resulting measures when multiplying the real-valued DCT matrix elements with $2^{6+M/2}$ and rounding to the closest integer are listed in the third column of Table 6.1. As can be seen from the table, although the matrix elements of the HEVC transforms are farther from the scaled DCT matrix elements, they have better orthogonality and norm properties.

Finally, by using only 8 bit representation, property 7 of Sect. 6.2.1 (trigonometric relationship between matrix elements) was not easily preserved. The authors are not aware of any trigonometric property of the HEVC core transforms that can be utilized to reduce the number of arithmetic operations below those required when using the (anti-) symmetry properties.

6.2.4 Basis Vectors of the HEVC Core Transforms

The left half of the 32×32 matrix specifying the 32-point forward transform is shown in Fig. 6.2. The right half can be derived by using the (anti-) symmetry properties of the basis vectors (property 6 of Sect. 6.2.1). The inverse transform matrix of HEVC is defined as the transpose of the matrix resulting from the figure. The 32×32 matrix contains up to 31 unique numbers as follows.

$$d^{32}_{i,0}, i = 1,\ldots,31 = \begin{Bmatrix} 90, 90, 90, 89, 88, 87, 85, 83, 82, 80, 78, 75, 73, 70, 67, 64, \\ 61, 57, 54, 50, 46, 43, 38, 36, 31, 25, 22, 18, 13, 9, 4 \end{Bmatrix} \tag{6.3}$$

These unique numbers are elements 1–31 of the first column of the forward transform matrix. Note that although the number 90 occurs three times, this is by accident and not generally true. The unique numbers property was used in [26] to enable 25 % area reduction for hardware designs with practical throughput.

Furthermore, the coefficients d^N_{ij} of the smaller transform matrices ($N = 4, 8, 16$) can be derived from the coefficients d^{32}_{ij} of the 32×32 transform matrix as:

$$d^N_{ij} = d^{32}_{i(32/N),j}, i, j = 0,\ldots,N-1 \tag{6.4}$$

Let $\mathbf{D}_4$ denote the 4×4 transform matrix. By using (6.4) and Fig. 6.2, $\mathbf{D}_4$ can be obtained as:

$$\mathbf{D}_4 = \begin{bmatrix} d^{32}_{0,0} & d^{32}_{0,1} & d^{32}_{0,2} & d^{32}_{0,3} \\ d^{32}_{8,0} & d^{32}_{8,1} & d^{32}_{8,2} & d^{32}_{8,3} \\ d^{32}_{16,0} & d^{32}_{16,1} & d^{32}_{16,2} & d^{32}_{16,3} \\ d^{32}_{24,0} & d^{32}_{24,1} & d^{32}_{24,2} & d^{32}_{24,3} \end{bmatrix} = \begin{bmatrix} 64 & 64 & 64 & 64 \\ 83 & 36 & -36 & -83 \\ 64 & -64 & -64 & 64 \\ 36 & -83 & 83 & -36 \end{bmatrix}$$

The 8×8 transform matrix $\mathbf{D}_8$ and the 16×16 transform matrix $\mathbf{D}_{16}$ can be similarly obtained from the 32×32 transform matrix as shown in Fig. 6.2 where different colors are used to highlight the embedded 16×16, 8×8 and 4×4 forward transform matrices. This property allows for different transform sizes to be implemented using the same architecture thereby facilitating hardware sharing [6].

Note that from the unique numbers property of (6.3) and the (anti-)symmetry properties, $\mathbf{D}_4$ is also equal to:

$$\mathbf{D}_4 = \begin{bmatrix} d^{32}_{16,0} & d^{32}_{16,0} & d^{32}_{16,0} & d^{32}_{16,0} \\ d^{32}_{8,0} & d^{32}_{24,0} & -d^{32}_{24,0} & -d^{32}_{8,0} \\ d^{32}_{16,0} & -d^{32}_{16,0} & -d^{32}_{16,0} & d^{32}_{16,0} \\ d^{32}_{24,0} & -d^{32}_{8,0} & d^{32}_{8,0} & -d^{32}_{24,0} \end{bmatrix} \tag{6.5}$$

64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64
90	90	88	85	82	78	73	67	61	54	46	38	31	22	13	4
90	87	80	70	57	43	25	9	-9	-25	-43	-57	-70	-80	-87	-90
90	82	67	46	22	-4	-31	-54	-73	-85	-90	-88	-78	-61	-38	-13
89	75	50	18	-18	-50	-75	-89	-89	-75	-50	-18	18	50	75	89
88	67	31	-13	-54	-82	-90	-78	-46	-4	38	73	90	85	61	22
87	57	9	-43	-80	-90	-70	-25	25	70	90	80	43	-9	-57	-87
85	46	-13	-67	-90	-73	-22	38	82	88	54	-4	-61	-90	-78	-31
83	36	-36	-83	-83	-36	36	83	83	36	-36	-83	-83	-36	36	83
82	22	-54	-90	-61	13	78	85	31	-46	-90	-67	4	73	88	38
80	9	-70	-87	-25	57	90	43	-43	-90	-57	25	87	70	-9	-80
78	-4	-82	-73	13	85	67	-22	-88	-61	31	90	54	-38	-90	-46
75	-18	-89	-50	50	89	18	-75	-75	18	89	50	-50	-89	-18	75
73	-31	-90	-22	78	67	-38	-90	-13	82	61	-46	-88	-4	85	54
70	-43	-87	9	90	25	-80	-57	57	80	-25	-90	-9	87	43	-70
67	-54	-78	38	85	-22	-90	4	90	13	-88	-31	82	46	-73	-61
64	-64	-64	64	64	-64	-64	64	64	-64	-64	64	64	-64	-64	64
61	-73	-46	82	31	-88	-13	90	-4	-90	22	85	-38	-78	54	67
57	-80	-25	90	-9	-87	43	70	-70	-43	87	9	-90	25	80	-57
54	-85	-4	88	-46	-61	82	13	-90	38	67	-78	-22	90	-31	-73
50	-89	18	75	-75	-18	89	-50	-50	89	-18	-75	75	18	-89	50
46	-90	38	54	-90	31	61	-88	22	67	-85	13	73	-82	4	78
43	-90	57	25	-87	70	9	-80	80	-9	-70	87	-25	-57	90	-43
38	-88	73	-4	-67	90	-46	-31	85	-78	13	61	-90	54	22	-82
36	-83	83	-36	-36	83	-83	36	36	-83	83	-36	-36	83	-83	36
31	-78	90	-61	4	54	-88	82	-38	-22	73	-90	67	-13	-46	85
25	-70	90	-80	43	9	-57	87	-87	57	-9	-43	80	-90	70	-25
22	-61	85	-90	73	-38	-4	46	-78	90	-82	54	-13	-31	67	-88
18	-50	75	-89	89	-75	50	-18	-18	50	-75	89	-89	75	-50	18
13	-38	61	-78	88	-90	85	-73	54	-31	4	22	-46	67	-82	90
9	-25	43	-57	70	-80	87	-90	90	-87	80	-70	57	-43	25	-9
4	-13	22	-31	38	-46	54	-61	67	-73	78	-82	85	-88	90	-90

Fig. 6.2 Left half of the 32×32 matrix specifying the 32-point forward transform. Embedded 4-point (*green shading*), 8-point (*pink shading*) and 16-point (*yellow shading*) forward transform matrices are also shown in the figure. Reproduced with permission from [6]. © IEEE 2013

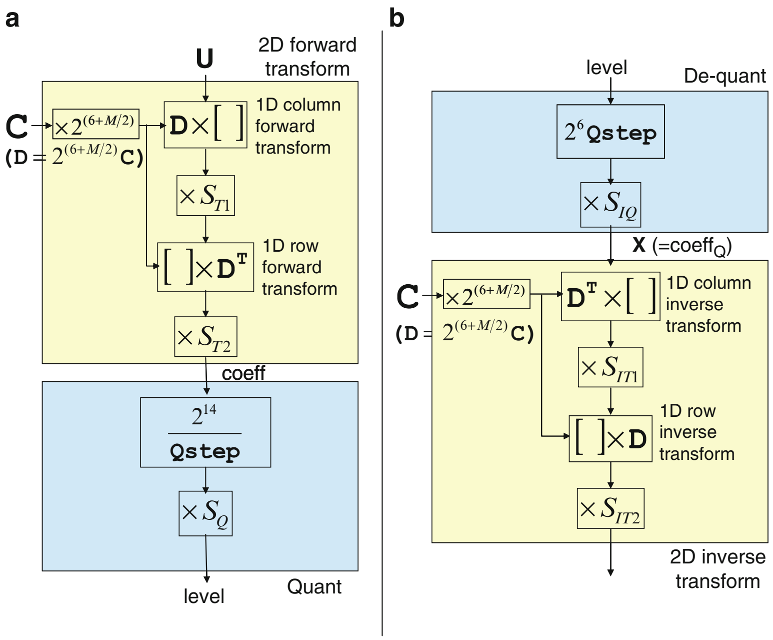

Fig. 6.3 Additional scale factors $S_{T1}, S_{T2}, S_{IT1}, S_{IT2}, S_Q, S_{IQ}$ required to implement HEVC integer transform and quantization. (**a**) Forward transform and quantization, (**b**) inverse transform and quantization. The 2D forward and inverse transform are implemented as separable 1D column and row transforms. **C** is the orthonormal DCT matrix. **D** is the scaled approximation of the DCT matrix. $M = \log_2(N)$ where N is the transform size. Reproduced with permission from [6]. © IEEE 2013

6.2.5 Intermediate Scaling

Since the HEVC matrices are scaled by $2^{(6+M/2)}$ compared to an orthonormal DCT transform, and in order to preserve the norm of the residual block through the forward and inverse two-dimensional transforms, additional scale factors—$S_{T1}, S_{T2}, S_{IT1}, S_{IT2}$ —need to be applied as shown in Fig. 6.3. Note that Fig. 6.3 is basically a fixed point implementation of the transform and quantization in Fig. 6.1. While the HEVC standard specifies the scale factors of the inverse transform (i.e. S_{IT1}, S_{IT2}), the HEVC reference software also specifies corresponding scale factors for the forward transform (i.e. S_{T1}, S_{T2}). The scale factors were chosen with the following constraints:

1. All scale factors shall be a power of two to allow the scaling to be implemented as a right shift.
2. Assuming full range of the input residual block (e.g. a DC block with all samples having maximum amplitude), the bit depth after each transform stage shall be equal to 16 bits (including the sign bit). This was considered a reasonable trade-off between accuracy and implementation costs.

3. Since the HEVC matrices are scaled by $2^{(6+M/2)}$, cascading of the two-dimensional forward and inverse transform will results in a scaling of $2^{(6+M/2)}$ for each of the 1D row forward transform, the 1D column forward transform, the 1D column inverse transform, and the 1D row inverse transform. Consequently to preserve the norm through the two-dimensional forward and inverse transforms, the product of all scale factors shall be equal to $(1/2^{(6+M/2)})^4 = 2^{-24}2^{-2M}$.

The process of selecting the forward transform scale factors is illustrated using the 4×4 forward transform as an example in Fig. 6.4. When video has a bit depth of B bits, the residual will be in the range of $[-2^B + 1, 2^B - 1]$ requiring $(B + 1)$ bits to represent it. In the following worst case bit-depth analysis we will assume a residual block with all samples having maximum amplitude equal to -2^B as input to the first stage of the forward transform. We believe this is a reasonable assumption since all basis vectors have almost the same norm. Note also that we are using -2^B instead of $-2^B + 1$ or $2^B - 1$ in the worst case analysis since it is a power of 2. The scale factor derivation becomes simpler assuming input to be -2^B (which still fits within $(B + 1)$ bits) since all the scale factors are a power of 2. For this worst case input block, the maximum value of an output sample will be $-2^B \times N \times 64$. This corresponds to the dot product of the first basis vector (of length N with all values equal to 64) with an input vector consisting of values equal to -2^B. Therefore, with $N = 2^M$, for the output to fit within 16 bits (i.e., maximum value of -2^{15}) a scaling of $1/(2^B \times 2^M \times 2^6 \times 2^{-15})$ is required. Consequently, the scale factor after the first transform stage is chosen as $S_{T1} = 2^{-(B+M-9)}$.

The second stage of the forward transform consists of multiplication of the result of the first transform stage with $\mathbf{D}_4^T$. The input into the second stage of the forward transform is the output from the first stage which is a matrix with all elements in the first row having a value of -2^{15}. All other elements will be zero as shown in Fig. 6.4. The output of multiplication with $\mathbf{D}_4^T$ will be a matrix with only a DC value equal to $-2^{15} \times 2^M \times 2^6$ and all remaining values equal to 0. This implies that the scaling required after the second stage of transform is $S_{T2} = 2^{-(M+6)}$ in order for the output to fit within 16 bits.

The first stage of the inverse transform consists of multiplication of the result of the forward transform with $\mathbf{D}_4^T$. In our example, the input into the first stage of the inverse transform is the output matrix from the forward transform which is a matrix with only the DC element equal to -2^{15}. The output of multiplication with $\mathbf{D}_4^T$ will be a matrix with first column elements equal to $-2^{15} \times 2^6$. Consequently, the scaling required after the first stage of the inverse transform for the output to fit within 16 bits is $S_{IT1} = 2^{-6}$.

The second stage of the inverse transform consists of multiplication of the result of the first stage of the inverse transform with $\mathbf{D}_4$. The input into the second stage of the inverse transform is the output matrix from the first stage of inverse transform which is a matrix with first column elements equal to -2^{15}. The output of multiplication with $\mathbf{D}_4$ will be a matrix with all elements equal to $-2^{15} \times 2^6$. So the scaling required after the second stage of inverse transform to get the output values into the original range of $[-2^B, 2^B - 1]$ is $S_{IT2} = 2^{-(21-B)}$.

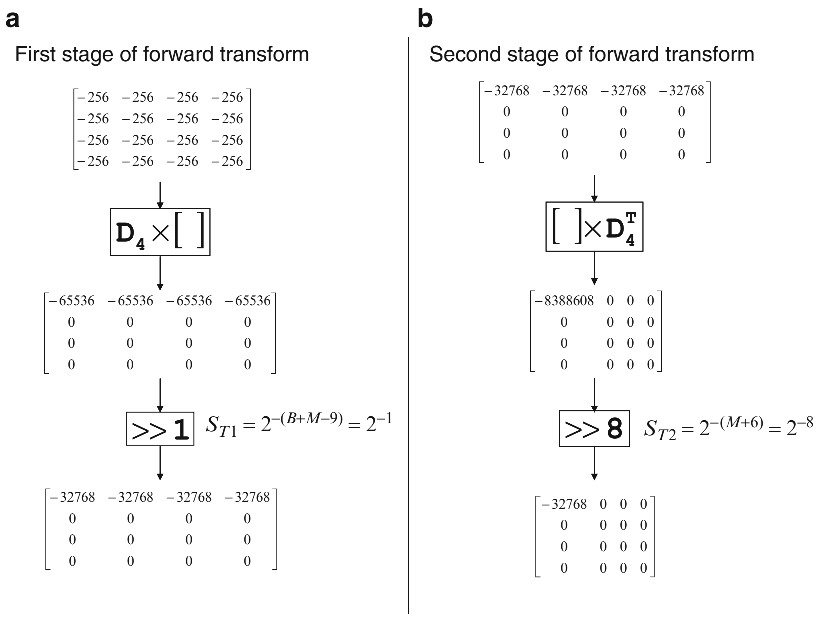

Fig. 6.4 Intermediate scaling factor determination for the forward transform so that the intermediate and output values fit within 16-bits. B is video bit depth and $M = \log_2(N)$ where N is the transform size. Worst case bit-depth analysis is done assuming a residual block with all samples having maximum amplitude equal to -2^B (where $B = 8$ is the video bit depth), as input to the first stage of the forward transform. (**a**) First stage of the forward transform, (**b**) Second stage of the forward transform. Reproduced with permission from [6]. © IEEE 2013

In summary the constraints imposed in this section result in the following scale factors after different transform stages:

- After the first forward transform stage: $S_{T1} = 2^{-(B+M-9)}$
- After the second forward transform stage: $S_{T2} = 2^{-(M+6)}$
- After the first inverse transform stage: $S_{IT1} = 2^{-6}$
- After the second inverse transform stage: $S_{IT2} = 2^{-(21-B)}$

where B is the bit depth of the input/output signal (e.g. 8 bit) and $M = \log_2(N)$.

Without quantization/de-quantization, this choice of scale factors ensures a bit depth of 16 bit after all transform stages. However, quantization errors introduced by the quantization/de-quantization process might increase the dynamic range before each inverse transform stage to more than 16 bit. For example, consider the situation where $B = 8$ and all input samples to the forward transform are equal to 255. In this case, the output of the forward transform will be a DC coefficient with value equal to $255 << 7 = 32640$. For high QP values and with a quantizer rounding upwards, the input to each inverse transform stage can easily exceed the allowed 16 bit dynamic range of $[-32768, 32767]$. While clipping to 16 bit range was considered trivial after the de-quantizer, it was considered undesirable after the first inverse transform stage. In order to allow for quantization error of some reasonable magnitude and at

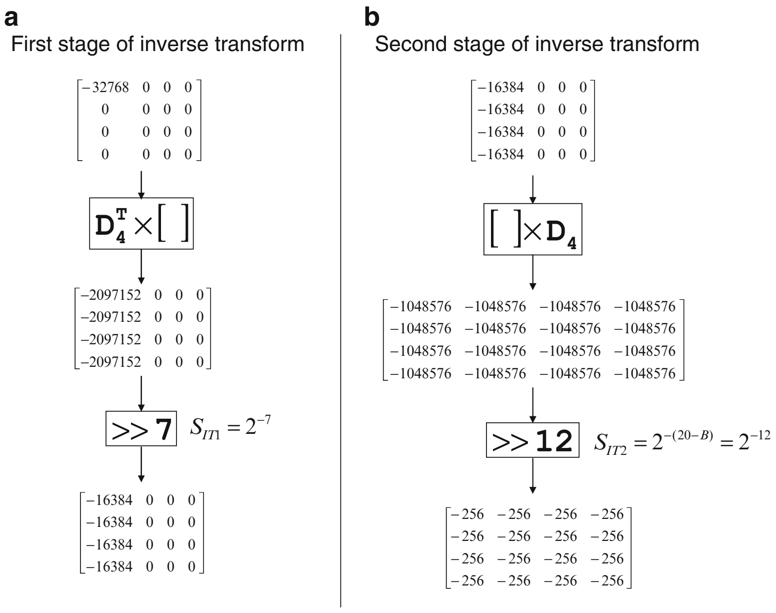
a
First stage of inverse transform
b
Second stage of inverse transform
$D_4^T \times [\ ]$
>>7
$S_{IT1} = 2^{-7}$
$[\ ] \times D_4$
>>12
$S_{IT2} = 2^{-(20-B)} = 2^{-12}$
-32768
-2097152
-16384
-1048576
-256

Fig. 6.5 Use of the inverse transform scale factors assuming the input to be the final output of Fig. 6.4. Video bit depth $B = 8$ (**a**) First stage of the inverse transform, (**b**) Second stage of the inverse transform. Reproduced with permission from [6]. © IEEE 2013

the same time limit the dynamic range between the two inverse transform stages to 16 bits, the choice of scale factors for the inverse transform was finally modified as follows[2]:

- After the first inverse transform stage: $S_{IT1} = 2^{-7}$
- After the second inverse transform stage: $S_{IT2} = 2^{-(20-B)}$

The use of the inverse transform scale factors is illustrated in Fig. 6.5 using the 4×4 inverse transform as an example assuming the input to be the final output of Fig. 6.4.

Tables 6.2 and 6.3 summarize the different scaling factors of the forward and inverse transform, respectively, when compared to the orthonormal DCT.

The HEVC specification specifies an offset value to be added before scaling to carry out rounding. This offset value is equal to the scale factor divided by 2. The offset is not explicitly shown in Figs. 6.3, 6.4, and 6.5.

[2]Note that in the final HEVC specification [16], a clipping operation is introduced after the first inverse transform stage, mainly to allow for random quantization that could be used to create "evil" bitstreams used for stress testing video decoders. With the clipping introduced, the modification to the inverse transform scale factors is not necessary but has been retained in the HEVC specification and Test Model software for maturity reasons.

Table 6.2 Scaling in different stages for the 2D forward transform

	Scale factor
First forward transform stage	$2^{(6+M/2)}$
After the first forward transform stage (S_{T1})	$2^{-(B+M-9)}$
Second forward transform stage	$2^{(6+M/2)}$
After the second forward transform stage (S_{T2})	$2^{-(M+6)}$
Total scaling for the forward transform	$2^{(15-B-M)}$

Table 6.3 Scaling in different stages for the 2D inverse transform

	Scale factor
First inverse transform stage	$2^{(6+M/2)}$
After the first inverse transform stage (S_{IT1})	2^{-7}
Second inverse transform stage	$2^{(6+M/2)}$
After the second inverse transform stage (S_{IT2})	$2^{-(20-B)}$
Total scaling for the inverse transform	$2^{-(15-B-M)}$

Finally, two useful consequences of using 8-bit coefficients and limiting the bit-depth of the intermediate data to 16 bit is that all multiplications can be represented with multipliers having 16 bits or less and that the accumulators before right shift can be implemented with less than 32 bits for all transform stages.

Note also a relevant analysis in [18] that studies the dynamic range of the HEVC inverse transform and provides additional information on the bit depth limits of the intermediate data in the inverse transform.

6.2.6 HEVC Alternate 4 × 4 Transform

The alternate transform is applied to 4 × 4 Luma intra-prediction residual blocks. The forward transform matrix is given by:

$$\mathbf{A}_4 = \begin{bmatrix} 29 & 55 & 74 & 84 \\ 74 & 74 & 0 & -74 \\ 84 & -29 & -74 & 55 \\ 55 & -84 & 74 & -29 \end{bmatrix}$$

The inverse transform matrix is $\mathbf{A}_4^T$. Elements a_{ij} of the alternate transform matrix $\mathbf{A}_4$ are a fixed point representation of Type-7 discrete sine transform (DST) obtained as follows:

$$a_{ij} = \text{round}\left(128 * \frac{2}{\sqrt{2N+1}} \sin\left(\frac{(2i+1)(j+1)\pi}{2N+1}\right)\right)$$

The intermediate scaling and quantization/de-quantization used for the alternate transform is the same as that for the core transform.

The alternate transform provides around 1 % bit-rate reduction while coding intra pictures [25]. In intra-picture prediction, a block is predicted from left and/or top neighboring samples. The prediction quality is better near the left and/or top boundary resulting in an intra-prediction residual that tends to have lower amplitude near the boundary samples and higher amplitudes away from the boundary samples. The DST basis functions are better than the DCT basis functions in modeling this spatial characteristic of the intra prediction residual. This can be seen from the first row (basis function) of the alternate transform matrix which increases from left to right as opposed to the DCT transform matrix that has a flat first row. A theoretical analysis of the optimality of DST for intra-prediction residual is provided in [25].

During the course of the development of HEVC, alternate transforms for transform block sizes of 8×8 and higher were also studied. However, only the 4×4 alternate transform was adopted in HEVC since the additional coding gain from using the larger alternate transforms was not significant (also, their complexity is higher since there is no symmetry in the transform matrix and a full matrix multiplication is needed to implement them for transform sizes 8×8 and larger).

6.3 Quantization and De-quantization

Quantization consists of division by a quantization step size (*Qstep*) and subsequent rounding while inverse quantization consists of multiplication by the quantization step size. Here, *Qstep* refers to the equivalent step size for an orthonormal transform, i.e. without the scaling factors of Tables 6.2 and 6.3. Similar to H.264/AVC [27], a quantization parameter (*QP*) is used to determine the quantization step size in HEVC. *QP* can take 52 values from 0 to 51 for 8-bit video sequences. An increase of 1 in *QP* means an increase of the quantization step size by approximately 12 % (i.e., $2^{1/6}$). An increase of 6 leads to an increase in the quantization step size by a factor of 2. In addition to specifying the *relative* difference between the step-sizes of two consecutive *QP* values, there is also a need to define the *absolute* step-size associated with the range of *QP* values. This was done by selecting $Qstep = 1$ for $QP = 4$.

The resulting relationship between *QP* and the equivalent quantization step size for an orthonormal transform is now given by:

$$Qstep(QP) = \left(2^{1/6}\right)^{QP-4} \tag{6.6}$$

Figure 6.6 shows how the quantization step size increases non-linearly with *QP*. Equation (6.6) can be also be expressed as:

$$Qstep(QP) = G_{QP\%6} << \frac{QP}{6} \tag{6.7}$$

where

$$\mathbf{G} = [G_0, G_1, G_2, G_3, G_4, G_5]^T = \left[2^{-4/6}, 2^{-3/6}, 2^{-2/6}, 2^{-1/6}, 2^0, 2^{1/6}\right]^T$$

The fixed point approximation of (6.7) in HEVC is given by

$$g_{QP\%6} = \text{round}\left(2^6 \times G_{QP\%6}\right)$$

This results in

$$\mathbf{g} = [g_0, g_1, g_2, g_3, g_4, g_5]^T = [40, 45, 51, 57, 64, 72]^T$$

HEVC supports frequency-dependent quantization by using quantization matrices for all transform block sizes. Let $W[x][y]$ denote the quantization matrix weight for the transform coefficients at location (x, y) in a transform block. A value of $W[x][y] = 1$ indicates that there is no weighting. The fixed point representation of $W[x][y]$ is given by:

$$w[x][y] = \text{round}(16 \times W[x][y])$$

where $w[x][y]$ is represented using 8-bit values.

For a quantizer output, $level[x][y]$, the de-quantizer is specified in the HEVC standard as

$$coeff_Q[x][y] = \left(\left(level[x][y] \times w[x][y] \times \left(g_{QP\%6} << \frac{QP}{6}\right)\right) + offset_{IQ}\right) >> shift1 \tag{6.8}$$

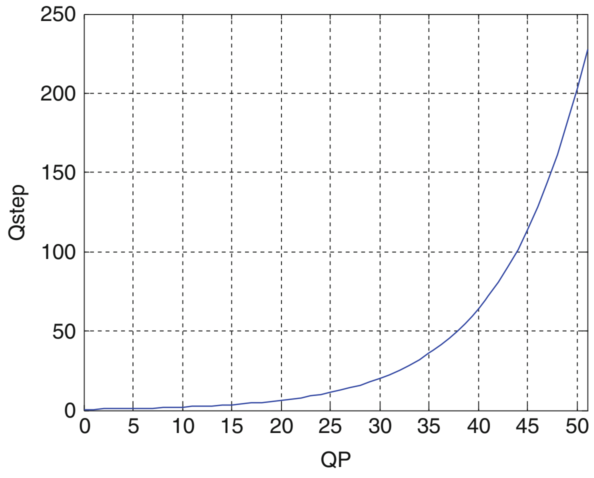

Fig. 6.6 Relationship between quantization step size (*Qstep*) and quantization parameter (*QP*)

where $shift1 = (M - 5 + B)$ and $offset_{IQ} = 1 << (M - 6 + B)$. Note that the quantization matrix weights $w[x][y]$ modulate the quantization step size used for *level* at different positions in the transform block leading to a frequency-dependent quantization.

The scale factor S_{IQ} of Fig. 6.3 is equal to $2^{-shift1}$ and is obtained as follows: When $QP = 4$ (i.e., $Qstep = 1$) and there is no frequency dependent scaling (i.e., $w[x][y] = 16$), the combined scaling of the inverse transform and de-quantization in Fig. 6.3 when multiplied together should result in a product of 1 to maintain the norm of the residual block through inverse transform and inverse quantization, i.e.,

$$S_{IQ} \times g_4 \times 16 \times 2^{-(15-B-M)} = 1 \tag{6.9}$$

This results in $S_{IQ} = 2^{-(M-5+B)}$ leading to $shift1$ being equal to right shift by $(M - 5 + B)$. The scale factor $2^{-(15-B-M)}$ in (6.9) is obtained from Table 6.3.

For the output sample of the forward transform, $coeff[x][y]$, a straightforward quantization scheme can be implemented as follows:

$$\begin{aligned} &level\,[x]\,[y] = sign\,(coeff\,[x]\,[y]) \\ &* \left(\left(\left(abs\,(coeff\,[x]\,[y]) \times f_{QP\%6} \times \tfrac{16}{w[x][y]} + offset_Q\right) >> \tfrac{QP}{6}\right) >> shift2\right) \end{aligned} \tag{6.10}$$

where $shift2 = 29 - M - B$, and

$$\mathbf{f} = [f_0, f_1, f_2, f_3, f_4, f_5]^T = [26214, 23302, 20560, 18396, 16384, 14564]^T$$

Note that $f_{QP\%6} \approx 2^{14}/G_{QP\%6}$. The value of $shift2$ is obtained by imposing similar constraints on the combined scaling in the forward transform and the quantizater as in (6.9), i.e., $S_Q \times f_4 \times 2^{15-B-M} = 1$, where $S_Q = 2^{-shift2}$.

Finally, $offset_Q$ is chosen to achieve the desired rounding.

To summarize, the quantizer multipliers, f_i, and dequantizer multipliers, g_i, were chosen to satisfy the following conditions

- Ensure that g_i can be represented with signed 8 bit data type (i.e., $g_i < 2^7, i = 0, \ldots, 5$)
- Ensure an almost equal increase in step size from one QP value to the next (approximately 12 %) (i.e., $g_{i+1}/g_i \approx 2^{1/6},\ i = 0, \ldots, 4$ and $2g_0/g_5 \approx 2^{1/6}$)
- Ensure approximately unity gain through the quantization and de-quantization processes (i.e.,$f_i \times g_i \times 16 \approx 1 << (shift1 + shift2) = 2^6 \times 2^{14} \times 16, i = 0, \ldots, 5$)
- Provide the desired absolute value of the quantization step size for $QP = 4$ (i.e. $Qstep(4) = 1$, or equivalently, $level = coeff \times 2^{-(15-B-M)}$ for $QP = 4$).

Note that the quantization equation in (6.10) is not specified in the HEVC standard and the encoder has flexibility to implement more sophisticated quantization schemes such as the rate-distortion optimized quantization (RDOQ) scheme implemented in the HEVC Test Model [13]. The idea behind RDOQ is briefly described in Chap. 9.

Default 4x4 for IntraLuma, IntraCb, IntraCr, InterLuma, InterCb, InterCr

$$\begin{bmatrix} 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 \end{bmatrix}$$

(Flat matrix)

Default 8x8 for IntraLuma, IntraCb, IntraCr

$$\begin{bmatrix} 16 & 16 & 16 & 16 & 17 & 18 & 21 & 24 \\ 16 & 16 & 16 & 16 & 17 & 19 & 22 & 25 \\ 16 & 16 & 17 & 18 & 20 & 22 & 25 & 29 \\ 16 & 16 & 18 & 21 & 24 & 27 & 31 & 36 \\ 17 & 17 & 20 & 24 & 30 & 35 & 41 & 47 \\ 18 & 19 & 22 & 27 & 35 & 44 & 54 & 65 \\ 21 & 22 & 25 & 31 & 41 & 54 & 70 & 88 \\ 24 & 25 & 29 & 36 & 47 & 65 & 88 & 115 \end{bmatrix}$$

Default 8x8 for InterLuma, InterCb, InterCr

$$\begin{bmatrix} 16 & 16 & 16 & 16 & 17 & 18 & 20 & 24 \\ 16 & 16 & 16 & 17 & 18 & 20 & 24 & 25 \\ 16 & 16 & 17 & 18 & 20 & 24 & 25 & 28 \\ 16 & 17 & 18 & 20 & 24 & 25 & 28 & 33 \\ 17 & 18 & 20 & 24 & 25 & 28 & 33 & 41 \\ 18 & 20 & 24 & 25 & 28 & 33 & 41 & 54 \\ 20 & 24 & 25 & 28 & 33 & 41 & 54 & 71 \\ 24 & 25 & 28 & 33 & 41 & 54 & 71 & 91 \end{bmatrix}$$

Fig. 6.7 Default quantization matrices for transform blocks of size 4×4 and 8×8

6.3.1 Quantization Matrix

In HEVC, the encoder can signal whether or not to use quantization matrices enabling frequency dependent scaling. Frequency dependent scaling is useful to carry out human visual system (HVS)-based quantization where low frequency coefficients are quantized with a finer quantization step size when compared to high frequency coefficients in the transform block [12]. HVS-based quantization can provide better visual quality than frequency independent quantization on some video sequences. HEVC uses the following 20 quantization matrices that depend on the size and type of the transform block:

- Luma: Intra 4×4, Inter 4×4, Intra 8×8, Inter 8×8, Intra 16×16, Inter 16×16, Intra 32×32, Inter 32×32
- Cb: Intra 4×4, Inter 4×4, Intra 8×8, Inter 8×8, Intra 16×16, Inter 16×16
- Cr: Intra 4×4, Inter 4×4, Intra 8×8, Inter 8×8, Intra 16×16, Inter 16×16

When frequency dependent scaling is enabled by using the syntax element `scaling_list_enabled_flag`, the quantization matrices of sizes 4×4 and 8×8 have default values as shown in Fig. 6.7. The default quantization matrices for transform blocks of size 16×16 and 32×32 are obtained from the default 8×8 quantization matrices of the same type by upsampling using replication as shown in Fig. 6.8. The red colored blocks in the figure indicate that a quantization matrix entry in the 8×8 quantization matrix is replicated into a 2×2 region in the 16×16 quantization matrix and into a 4×4 region in the 32×32 quantization matrix. 8×8 matrices are used to represent 16×16 and 32×32 quantization matrices in order to reduce the memory needed to store the quantization matrices.

Non-default quantization matrices can also be optionally transmitted in the bitstream in sequence parameter sets (SPS) or picture parameter sets (PPS). Quantization matrix entries are scanned using an up-right diagonal scan and DPCM coded and transmitted. For 16×16 and 32×32 quantization matrices, only size 8×8 matrices (which then get upsampled to the correct size in the

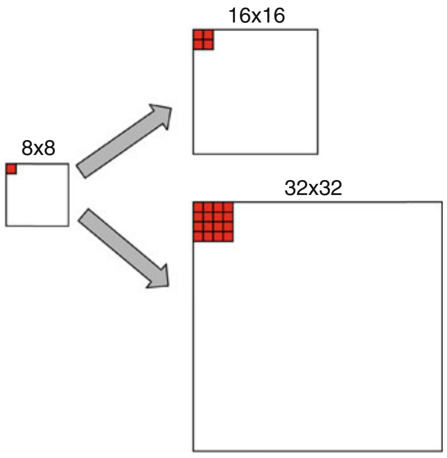

Fig. 6.8 Construction of default quantization matrices for transform block sizes 16 × 16 and 32 × 32 by using the default quantization matrix of size 8 × 8

Table 6.4 Quantization group size for different coding tree unit sizes

`diff_cu_qp_delta_depth`	Quantization group size for 64 × 64 CTU	Quantization group size for 32 × 32 CTU	Quantization group size for 16 × 16 CTU
0	64 × 64	32 × 32	16 × 16
1	32 × 32	16 × 16	8 × 8
2	16 × 16	8 × 8	–
3	8 × 8	–	–

decoder as shown in Fig. 6.8) and the quantization matrix entry at the DC (zero-frequency) position are transmitted. HEVC also allows for prediction of a quantization matrix from another quantization matrix of the same size. The use of quantization matrix (termed as scaling matrix in HEVC) is enabled by setting the flag `scaling_list_enabled_flag` in SPS. When this flag is enabled, additional flags in SPS and PPS control whether the default quantization matrices or non-default quantization matrices are used.

6.3.2 QP Parameter Derivation

The quantization step size (and therefore the *QP* value) may need to be changed within a picture for e.g. rate control and perceptual quantization purposes. HEVC allows for transmission of a delta *QP* value at a *quantization group* (QG) level to allow for *QP* changes within a picture. This is similar to H.264/AVC that allows for modification of QP values at a macroblock level. The QG size is a multiple of coding unit size that can vary from 8 × 8 to 64 × 64 depending on the coding tree unit (CTU) size and the syntax element `diff_cu_qp_delta_depth` as shown in Table 6.4.

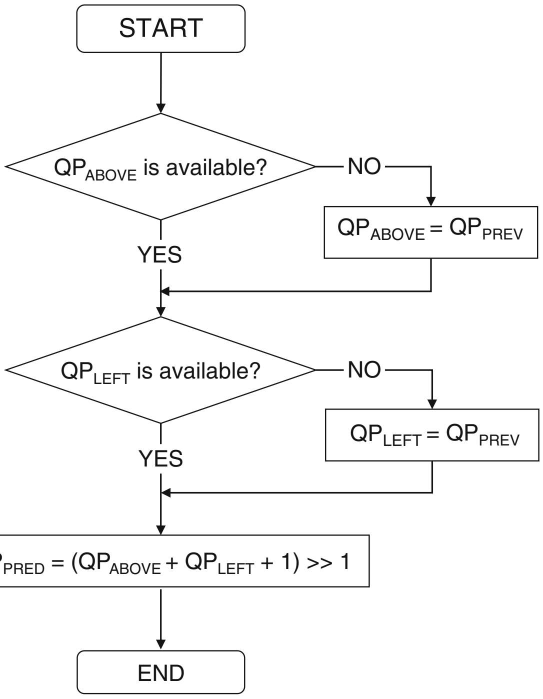
START
QP_{ABOVE} is available?
NO
$QP_{ABOVE} = QP_{PREV}$
YES
QP_{LEFT} is available?
NO
$QP_{LEFT} = QP_{PREV}$
YES
$QP_{PRED} = (QP_{ABOVE} + QP_{LEFT} + 1) >> 1$
END

Fig. 6.9 QP predictor calculation using QP values from the left, above and previous QGs [21]

The delta *QP* is transmitted only in coding units with non-zero transform coefficients. If the CTU is split into coding units that are greater than the QG size, then delta *QP* is signaled at a coding unit (with non-zero transform coefficients) that is greater than the QG size. If the CTU is split into coding units that are smaller than the QG size, then the delta *QP* is signaled in the first coding unit with non-zero transform coefficients in the QG. If a QG has coding units with all zero transform coefficients (e.g. if the merge mode is used in all the coding units of the QG), then delta *QP* will not be signaled.

The QP predictor used for calculating the delta QP uses a combination of QP values from the left, above and the previous QG in decoding order as shown in Fig. 6.9 [21]. The QP predictor uses a combination of two predictive techniques: spatial QP prediction (from left and above QGs) and previous QP prediction. It uses spatial prediction from left and above within a CTU and uses the previous QP as predictor at the CTU boundary. This is shown in Fig. 6.9. The spatially adjacent QP values, QP_{LEFT} and QP_{ABOVE} are considered to be not available when they are in a different CTU or if the current QG is at a slice/tile/picture boundary. When a spatially adjacent QP value is not available, it is replaced with the previous QP value, QP_{PREV}, in decoding order. The previous QP, QP_{PREV}, is initialized to the slice QP value at the beginning of the slice, tile or wavefront.

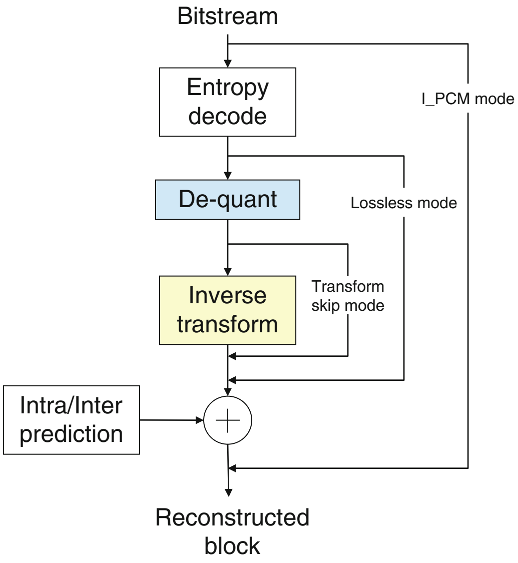

Fig. 6.10 I_PCM, lossless and transform skip modes in decoder

The QP derivation process described in this subsection is used for calculating the luma QP value. The chroma QP values (one for the Cr component and one for the Cb component) are derived from the luma QP by using picture level and slice level offsets and a table lookup.

6.4 HEVC Special Coding Modes

HEVC has three special modes that modify the transform and quantization process: (a) I_PCM mode [8], (b) lossless mode [31], and (c) transform skip mode [19]. These modes skip either the transform or both the transform and quantization. Figure 6.10 shows these modes on top of the generic video decoder data flow of Fig. 6.1.

- In the I_PCM mode, both transform and transform-domain quantization are skipped. In addition, entropy coding and prediction are skipped too and the video samples are directly coded with the specified PCM bit depth. The I_PCM mode is designed for use when there is data expansion during coding e.g. when random noise is input to the video codec. By directly coding the video samples, the data expansion can be avoided for such extreme video sequences. The IPCM mode is signaled at the coding unit level using the syntax element `pcm_flag`.
- In the lossless mode, both transform and quantization are skipped. (The in-loop filter which is not shown in Fig. 6.1 is skipped too.) Mathematically lossless reconstruction is possible since the residual from inter- or intra-picture prediction is directly coded. The lossless mode is signaled at a coding unit level (using the syntax element `cu_transquant_bypass_flag`) in order to enable

mixed lossy/lossless coding of pictures. Such a feature is useful in coding video sequences with mixed content, e.g. natural video with overlaid text and graphics. The text and graphics regions can be coded losslessly to maximize readability whereas the natural content can be coded in a lossy fashion.

- In the transform skip mode, only the transform is skipped. This mode was found to improve compression of screen-content video sequences generated in applications such as remote desktop, slideshows etc. These video sequences predominantly contain text and graphics. Transform skip is restricted to only 4×4 transform blocks and its use is signaled at the transform unit level by the `transform_skip_flag` syntax element.

6.5 Complexity Analysis

With straightforward matrix multiplication, the number of operations for the 1D inverse transform is N^2 multiplications and $N(N-1)$ additions. For the 2D transform, the number of multiplications required is $2N^3$ and the number of additions required is $2N^2(N-1)$. However, by utilizing the (anti-) symmetry properties of each basis vector inherited from DCT, the number of arithmetic operations can be significantly reduced. We refer to the algorithm that does this as the Even–Odd decomposition in this paper (it was also referred to as partial butterfly during HEVC development) [14]. Even–Odd decomposition is illustrated below using the 4- and 8-point inverse transform.

Consider the 4-point forward transform matrix defined in (6.5). For notational simplicity the constants $d_{i,0}^{32}$ of Eq. (6.5) will be denoted by d_i. Using the new notation (6.5) becomes

$$\mathbf{D}_4 = \begin{bmatrix} d_{16} & d_{16} & d_{16} & d_{16} \\ d_8 & d_{24} & -d_{24} & -d_8 \\ d_{16} & -d_{16} & -d_{16} & d_{16} \\ d_{24} & -d_8 & d_8 & -d_{24} \end{bmatrix} \tag{6.11}$$

The inverse transform matrix is given by $\mathbf{D}_4^T$. Let $\mathbf{x} = [x_0, x_1, x_2, x_3]^T$ be the input vector and $\mathbf{y} = [y_0, y_1, y_2, y_3]^T$ denote the output. The 1D 4-point inverse transform is given by the following equation:

$$\mathbf{y} = \mathbf{D}_4^T \mathbf{x} \tag{6.12}$$

The Even–Odd decomposition of the inverse transform of an N-point input consists of the following three steps:

1. Calculate the even part using a $N/2 \times N/2$ subset matrix obtained from the even columns of the inverse transform matrix (6.13 shows an example).

2. Calculate the odd part using a $N/2 \times N/2$ subset matrix obtained from the odd columns of the inverse transform matrix (6.15 shows an example).
3. Add/subtract the odd and even parts to generate N-point output (6.16 shows an example).

Even–odd decomposition of the inverse 4-point transform is given by (6.14–6.16):

Even part:

$$\begin{bmatrix} z_0 \\ z_1 \end{bmatrix} = \begin{bmatrix} d_{16} & d_{16} \\ d_{16} & -d_{16} \end{bmatrix} \begin{bmatrix} x_0 \\ x_2 \end{bmatrix} \tag{6.13}$$

The even part can be further simplified as:

$$\begin{aligned} t_0 &= d_{16} x_0 \\ t_1 &= d_{16} x_2 \\ \begin{bmatrix} z_0 \\ z_1 \end{bmatrix} &= \begin{bmatrix} t_0 + t_1 \\ t_0 - t_1 \end{bmatrix} \end{aligned} \tag{6.14}$$

Odd part:

$$\begin{bmatrix} z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} -d_{24} & d_8 \\ -d_8 & -d_{24} \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} \tag{6.15}$$

Add/sub:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} z_0 - z_3 \\ z_1 - z_2 \\ z_1 + z_2 \\ z_0 + z_3 \end{bmatrix} \tag{6.16}$$

The direct 1D 4-point transform using (6.12) would require 16 multiplications and 12 additions. The 2D transform will require 128 multiplications and 96 additions. Even–Odd decomposition on the other hand requires a total of six multiplications and eight additions for 1D transform using (6.14–6.16). The 2D transform using Even–Odd decomposition will require a total of 48 multiplications and 64 additions which is 62.5 % savings in number of multiplications and 33.3 % savings in number of additions when compared to direct matrix multiplication.

The 8-point 1D inverse transform is defined by the following equation:

$$\mathbf{y} = \mathbf{D}_8^T \mathbf{x} \tag{6.17}$$

where $\mathbf{x} = [x_0, x_1, \ldots, x_7]^T$ is input and $\mathbf{y} = [y_0, y_1, \ldots, y_7]^T$ is output, and $\mathbf{D}_8$ is given by:

$$\mathbf{D}_8 = \begin{bmatrix} d_{16} & d_{16} & d_{16} & d_{16} & d_{16} & d_{16} & d_{16} & d_{16} \\ d_4 & d_{12} & d_{20} & d_{28} & -d_{28} & -d_{20} & -d_{12} & -d_4 \\ d_8 & d_{24} & -d_{24} & -d_8 & -d_8 & -d_{24} & d_{24} & d_8 \\ d_{12} & -d_{28} & -d_4 & -d_{20} & d_{20} & d_4 & d_{28} & -d_{12} \\ d_{16} & -d_{16} & -d_{16} & d_{16} & d_{16} & -d_{16} & -d_{16} & d_{16} \\ d_{20} & -d_4 & d_{28} & d_{12} & -d_{12} & -d_{28} & d_4 & -d_{20} \\ d_{24} & -d_8 & d_8 & -d_{24} & -d_{24} & d_8 & -d_8 & d_{24} \\ d_{28} & -d_{20} & d_{12} & -d_4 & d_4 & -d_{12} & d_{20} & -d_{28} \end{bmatrix} \tag{6.18}$$

Even–Odd decomposition for the 8-point inverse transform is given by (6.19–6.21).

Even part:

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} d_{16} & d_8 & d_{16} & d_{24} \\ d_{16} & d_{24} & -d_{16} & -d_8 \\ d_{16} & -d_{24} & -d_{16} & d_8 \\ d_{16} & -d_8 & d_{16} & -d_{24} \end{bmatrix} \begin{bmatrix} x_0 \\ x_2 \\ x_4 \\ x_6 \end{bmatrix} \tag{6.19}$$

Odd part:

$$\begin{bmatrix} z_4 \\ z_5 \\ z_6 \\ z_7 \end{bmatrix} = \begin{bmatrix} -d_{28} & d_{20} & -d_{12} & d_4 \\ -d_{20} & d_4 & -d_{28} & -d_{12} \\ -d_{12} & d_{28} & d_4 & d_{20} \\ -d_4 & -d_{12} & -d_{20} & -d_{28} \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \\ x_5 \\ x_7 \end{bmatrix} \tag{6.20}$$

Add/sub:

$$\mathbf{y} = [z_0 - z_7, z_1 - z_6, z_2 - z_5, z_3 - z_4, z_3 + z_4, z_2 + z_5, z_1 + z_6, z_0 + z_7]^T \tag{6.21}$$

Note that the even part of the 8-point inverse transform is actually a 4-point inverse transform (by comparing 6.19 with transpose of $\mathbf{D}_4$ in 6.11) i.e.,

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \mathbf{D}_4^T \begin{bmatrix} x_0 \\ x_2 \\ x_4 \\ x_6 \end{bmatrix} \tag{6.22}$$

So the Even–Odd decomposition of the 4-point inverse transform (6.14–6.16) can be used to further reduce computational complexity of the even part of the 8-point transform in (6.19).

The direct 1D 8-point transform using (6.17) would require 64 multiplications and 56 additions. The 2D transform will require 1,024 multiplications and 896 additions. An even–odd decomposition on the other hand requires 6 multiplications for (6.22) and 16 multiplications for (6.20) resulting in a total of 22 multiplications. It requires 8 additions for (6.22), 12 additions for (6.20) and 8 additions for

(6.21) resulting in a total of 28 additions. The 2D transform using Even–Odd decomposition will require a total of 352 multiplications and 448 additions.

The computational complexity calculation for the 4-point and 8-point inverse transform can be extended to inverse transforms of larger sizes. In general, the resulting number of multiplications and additions (excluding the rounding operations associated with the shift operations) for the two-dimensional N-point inverse transform can be shown to be

$$O_{mult} = 2N\left(1 + \sum_{k=1}^{\log_2 N} 2^{2k-2}\right)$$

$$O_{add} = 2N\left(\sum_{k=1}^{\log_2 N} 2^{k-1}\left(2^{k-1}+1\right)\right)$$

The number of arithmetic operations for the inverse transform can be further reduced if knowledge about zero-valued input transform coefficients is assumed. In an HEVC decoder, this information can be obtained from the entropy decoding or de-quantization process. For typical video content many blocks of size $N \times N$ will have non-zero coefficients only in a $K \times K$ low frequency sub-block. For example in [5] it was found that on average around 75 % of the transform blocks had non-zero coefficients only in $K \times K$ low frequency sub-blocks. Computations can be saved in two ways for such transform blocks. Figure 6.11 shows the first way. Columns that are completely zero need not be inverse transformed. So only K 1D IDCTs along columns needs to be carried out. However, all N rows will need to be transformed subsequently. The second way to reduce computations is by exploiting the fact that each of the column and row IDCT is on a vector that has non-zero values only in the first K locations. For example with $K = N/2$, $x_4 = x_5 = x_6 = x_7 = 0$, roughly half the computations for the inverse transformation can be eliminated by simplifying Eqs. (6.19–6.20) to

Even part:

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} d_{16} & d_8 \\ d_{16} & d_{24} \\ d_{16} & -d_{24} \\ d_{16} & -d_8 \end{bmatrix} \begin{bmatrix} x_0 \\ x_2 \end{bmatrix}$$

Odd part:

$$\begin{bmatrix} z_4 \\ z_5 \\ z_6 \\ z_7 \end{bmatrix} = \begin{bmatrix} -d_{28} & d_{20} \\ -d_{20} & d_4 \\ -d_{12} & d_{28} \\ -d_4 & -d_{12} \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \end{bmatrix}$$

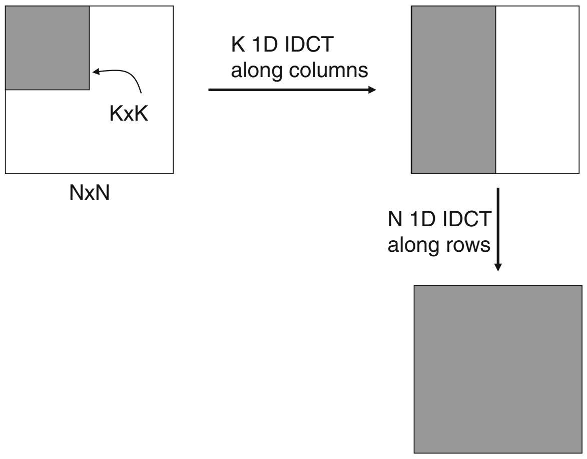

Fig. 6.11 Efficient implementation of inverse transform of a block with non-zero coefficients in only the $K \times K$ low frequency sub-block. Shaded regions denote the regions that can contain non-zero coefficients. Only K 1D IDCTs are required along columns

Table 6.5 Arithmetic operation counts for HEVC two-dimensional inverse transforms

N	K	O_{mult}	O_{add}
4	4	48	64
8	8	352	448
8	4	132	228
16	16	2,752	3,200
16	8	1,032	1,512
16	4	420	820
32	32	21,888	23,808
32	16	8,208	10,320
32	8	3,400	5,320
32	4	1,476	3,060

In general, the number of multiplications can be reduced approximately by a factor of $(N/K)^2$ for the first stage and a factor of (N/K) for the second stage. Table 6.5 shows the number of arithmetic operations for various values of N and K.

Note that the majority of the arithmetic operations listed in Table 6.5 can be efficiently implemented using SIMD instructions since the operations are matrix multiply operations. For example, for an 8×8 inverse transform implementation, (6.20) can be efficiently implemented on a 4-way SIMD processor in 4 cycles v/s 16 cycles on a processor without SIMD acceleration. Software performance using SIMD acceleration on various Intel processor architectures for the 8×8, 16×16, and 32×32 transform sizes are provided in [3, 11].

Only the Even–Odd decomposition of the inverse transform has been described in this subsection. However, the Even–Odd decomposition idea can be used to reduce the complexity of the forward transform too. The article [6] presents analysis of both the forward and the inverse core transform in more details. It also describes hardware sharing by the application of property 4 of Sect. 6.2.1 (smaller transforms being embedded in larger transforms).

6.6 Coding Performance

The different transform sizes used in a coding block in HEVC are signaled in a quadtree structure [29]. The maximum transform size to use in a coding block is signaled in the sequence parameter set. Table 6.6 compares the coding performance of HEVC when all transform sizes (up to 32×32) are used to the coding performance when only 4×4 and 8×8 transforms are used as in H.264/AVC. The standard Bjøntegaard Delta-Rate (BD-Rate) metric [2] is used for comparison. Table 6.6 shows that there is a bit rate savings in the range of 5.6–6.8 % on average because of the introduction of larger transform sizes (16×16 and 32×32) in HEVC. The bit rate savings are higher at larger resolution video such as 4K (2560×1600) and 1080p (1920×1080). The HEVC Test Model, HM-9.0.1 [13] was used for the simulations and the video sequences and coding conditions used were as described in [4].

Table 6.7 compares the coding performance of the HEVC 4×4 and 8×8 transforms to that of the corresponding H.264/AVC transforms. The H.264/AVC 4×4 and 8×8 transforms were converted to 8-bit precision and implemented in the HM-9.0.1 Test Model. Only the 4×4 and 8×8 transform sizes were enabled in the simulations. It can be seen from Table 6.7 that the HEVC 4×4 and 8×8 transforms perform better than the corresponding H.264/AVC transforms in terms of coding performance.

Table 6.6 BD-rate savings of using larger transform sizes (16×16 and 32×32) on top of the smaller transform sizes (4×4 and 8×8)

	All Intra (%)	Random access (%)	Low delay B (%)
4K	−9.1	−10.1	n/a
1080p	−6.7	−8.0	−9.1
WVGA	−2.5	−4.3	−6.0
WQVA	−2.2	−2.8	−3.7
720p	−7.7	n/a	−8.4
Overall	−5.6	−6.4	−6.8

Table 6.7 BD-Rate savings of the HEVC 4×4 and 8×8 transforms versus the H.264/AVC 4×4 and 8×8 transforms

	All Intra (%)	Random access (%)	Low delay B (%)
4K	−1.2	−0.7	n/a
1080p	−0.6	−0.4	−0.3
WVGA	−0.2	−0.2	−0.1
WQVA	−0.1	0.0	−0.1
720p	−0.5	n/a	−0.2
Overall	−0.5	−0.3	−0.2

References

1. Alshina E, Alshin A, Lee W, Park J (2011) Full factorization core transforms for HEVC, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G737, Geneva, Nov. 2011.
2. Bjøntegaard G (2001) VCEG-M33: calculation of average PSNR differences between RD curves, ITU-T SG16 Q6 Video Coding Experts Group (VCEG), Document VCEG-M33, Austin, Apr. 2001.
3. Bossen F (2011) On software complexity, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G757, Geneva, Nov. 2011.
4. Bossen F (2012) Common HM test conditions and software reference configurations, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-K1100, Shanghai, Oct. 2012.
5. Budagavi M (2011) IDCT pruning, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-E386, Geneva, Mar. 2011
6. Budagavi M, Fuldseth A, Bjøntegaard G, Sze V, Sadafale M (2013) Core transform design in the High Efficiency Video Coding (HEVC) standard. IEEE Trans Circuits Syst Video Technol 7(6):1029–1041
7. Chen W-H, Smith CH, Fralick S (1977) A fast computational algorithm for the discrete cosine transform. IEEE Trans Commun COM-25(9):1004–1009
8. Chono K, Aoki H, Wahadaniah V, Lim CS (2011) Proposal of enhanced PCM coding in HEVC, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-E192, Geneva, Mar. 2011
9. Dai W, Krishnan M, Topiwala J, Topiwala P, Alshina E (2011) Lossless core transforms for HEVC, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G266, Geneva, Nov. 2011
10. Fuldseth A, Bjøntegaard G, Sadafale M, Budagavi M (2011) Core transform design for HEVC, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G495, Geneva, Nov. 2011
11. Fuldseth A, Endresen LP, Selnes S, Arbatov V, Franchetti F, Puschel M (2011) SIMD Optimization of proposed HEVC transforms, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G497, Geneva, Nov. 2011
12. Haque M, Tabatabai A, Morigami Y (2011) HVS model based default quantization matrices, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G880, Geneva, Nov. 2011
13. HEVC Test Model HM-9.0.1 Nov. 2012 [Online]. Available https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-9.0.1/
14. Hung C-Y, Landman P (1997) Compact inverse discrete cosine transform circuit for MPEG video decoding. In: Proceedings of IEEE SIPS, Nov. 1997, pp 364–373
15. ITU-T Rec. H.264 and ISO/IEC 14496-10 (2003) Advanced video coding
16. ITU-T Rec. H.265 and ISO/IEC 23008-2 (2013) High efficiency video coding
17. Joshi R, Sole J, Karczewicz M (2011) Scaled integer transform supporting recursive factorization structure, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G579, Geneva, Nov. 2011
18. Kerofsky L, Riabtsev S (2012) Dynamic range analysis of HEVC/H.265 inverse transform operations, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-L0332, Geneva, Jan. 2013
19. Lan C, Xu J, Sullivan GJ, Wu F (2012) Intra transform skipping, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-I0408, Geneva, Apr. 2012
20. Malvar HS, Hallapuro A, Karczewicz M, Kerofsky L (2003) Low complexity transform and quantization in H.264/AVC. IEEE Trans Circuits Syst Video Technol 13(7):598–603
21. Nakamura H, Nishitani M, Fukushima S (2012) Non-CE4: compatible QP prediction with RC and AQ, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-H0204, San Jose, Feb. 2012

22. Rao KR, Yip P (1990) Discrete cosine transform: algorithms, advantages, applications. Academic, Boston
23. Sadafale M, Budagavi M (2010) Low-complexity configurable transform architecture for HEVC, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-C226, Guangzhou, Oct. 2010
24. Saxena A, Fernandes FC (2011) CE7: mode-dependent DCT/DST without 4×4 full matrix multiplication for intra prediction, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-E125, Geneva, Mar. 2011
25. Saxena A, Fernandes FC (2013) DCT/DST-based transform coding for intra prediction in image/video coding. IEEE Trans Image Proc 22(10):3974–3981
26. Tikekar M, Huang C-T, Juvekar C, Chandrakasan A (2011) Core transform property for practical throughput hardware design. Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G265, Geneva, Nov. 2011
27. Wiegand T, Sullivan GJ, Bjøntegaard G, Luthra A (2003) Overview of the H.264/AVC video coding standard. IEEE Trans Circuits Syst Video Technol 13(7):560–570
28. Wiegand T, Han W-J, Ohm J-R, Sullivan GJ (2010) High Efficiency Video Coding (HEVC) text specification working draft 1, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-C403, Guangzhou, Oct. 2010
29. Winken M, Helle P, Marpe D, Schwarz H, Wiegand T (2011) Transform coding in the HEVC Test Model. In: Proceedings of the IEEE international conference image processing, pp 3693–3696
30. Zhou M, Sze V (2010) TE 12: evaluation of transform unit (TU) size, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-C056, Guangzhou, Oct. 2010
31. Zhou M, Gao W, Jiang M, Yu H (2012) HEVC lossless coding and improvements. IEEE Trans Circuits Syst Video Technol 22(12):1839–1843