Chapter 12
New Video Coding Standards: H.264 and H.265

12.1 Overview of H.264

- H.264 is also known as MPEG-4 Part 10, AVC (Advanced Video Coding). It is often referred to as the H.264/AVC (or H.264/MPEG-4 AVC) video coding standard.
- H.264 provides a higher video coding efficiency, up to 50% better compression than MPEG-2 and up to 30% better than H.263 and MPEG-4 Advanced Simple Profile, while maintaining the same quality of the compressed video.

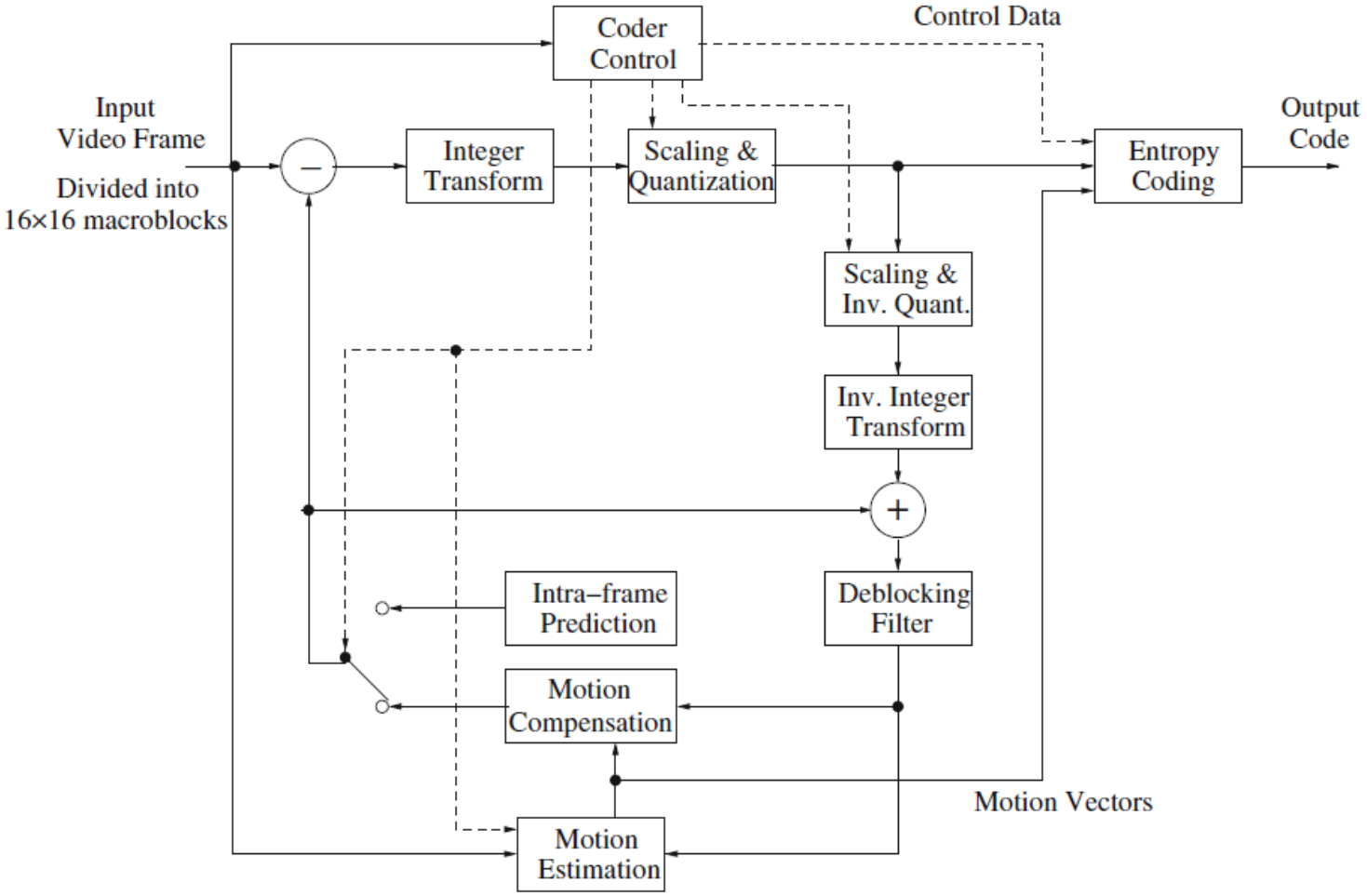


Fig. 12.1: H.264 encoder

12.1 Overview of H.264 (Cont'd)

- Main features of H.264/AVC are:
  - Integer transform in 4 × 4 blocks.
  - Variable block-size motion compensation in luma images.
  - Quarter-pixel accuracy in motion vectors.
  - Multiple reference picture motion compensation.
  - Directional spatial prediction for intra frames.
  - In-loop deblocking filtering.
  - CAVLC and CABAC.
  - Robust to data errors and data losses.

12.1 Overview of H.264 (Cont'd)

- Major blocks in decoder
  - Entropy decoding.
  - Inverse quantization and transform of residual pixels.
  - Motion compensation or intra-prediction.
  - Reconstruction.
  - In-loop deblocking filter on reconstructed pixels.

12.1.1 Motion compensation

- Similar to MPEG-2 and H.263, H.264 employs the technology of *hybrid coding*, i.e., a combination of inter-picture motion predictions and intra-picture spatial prediction, and transform coding on residual errors.

Variable Block-Size Motion Compensation

- Initial macroblock size is 16×16, could be further divided into smaller blocks

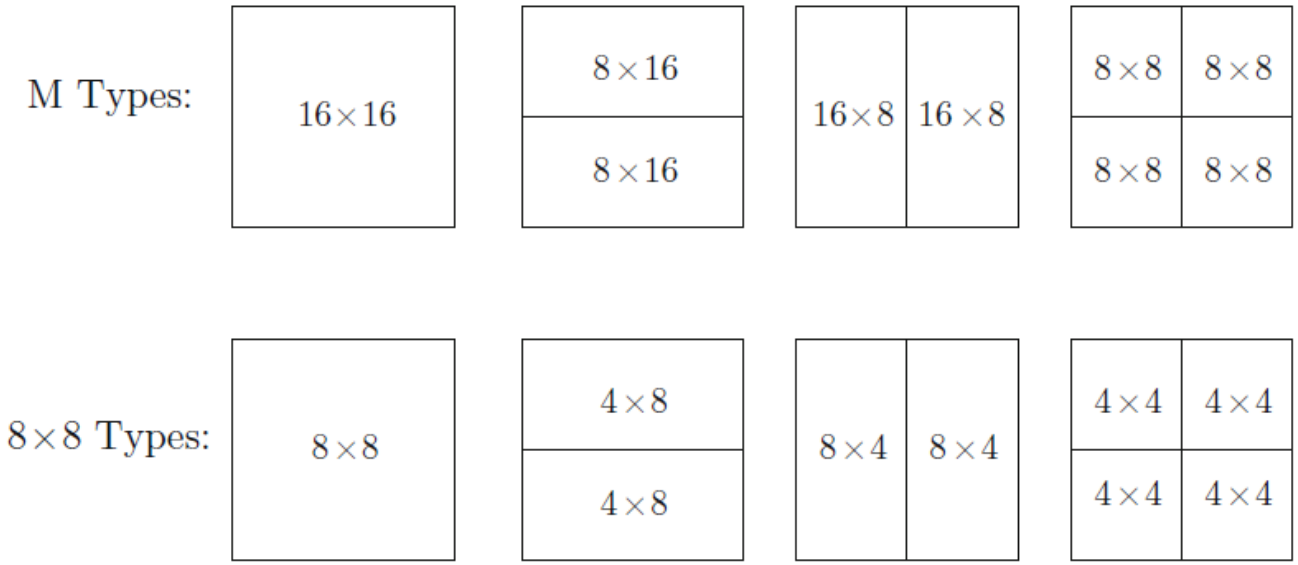


Fig. 12.2: H.264 macroblock segmentation

Quarter-Pixel Precision

- The accuracy of motion compensation is quarter-pixel precision in luma images. Pixel values at half-pixel and quarter-pixel positions can be derived by interpolation

- In Figure 12.3, the half pixel b and h could be calculated by

$$b_1 = E - 5F + 20G + 20H - 5I + J$$

$$h_1 = A - 5C + 20G + 20M - 5R + T$$

$$b = (b_1 + 16) \gg 5$$

$$h = (h_1 + 16) \gg 5.$$

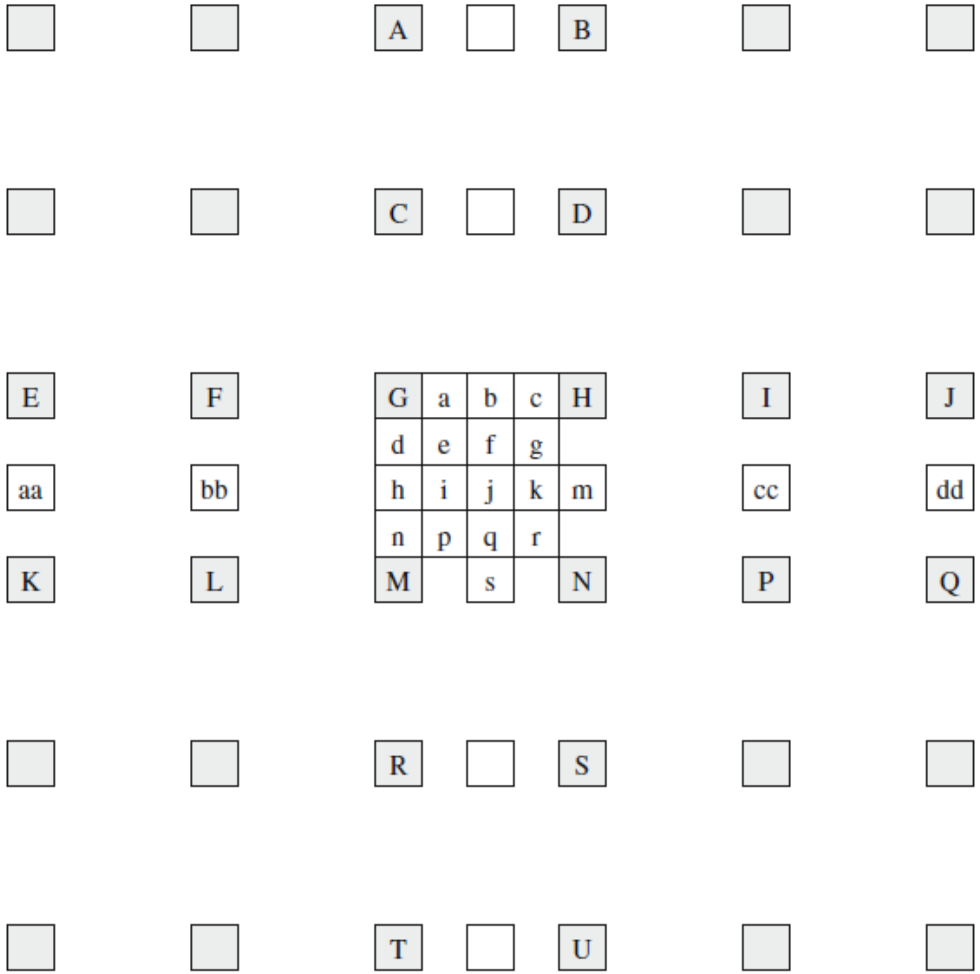
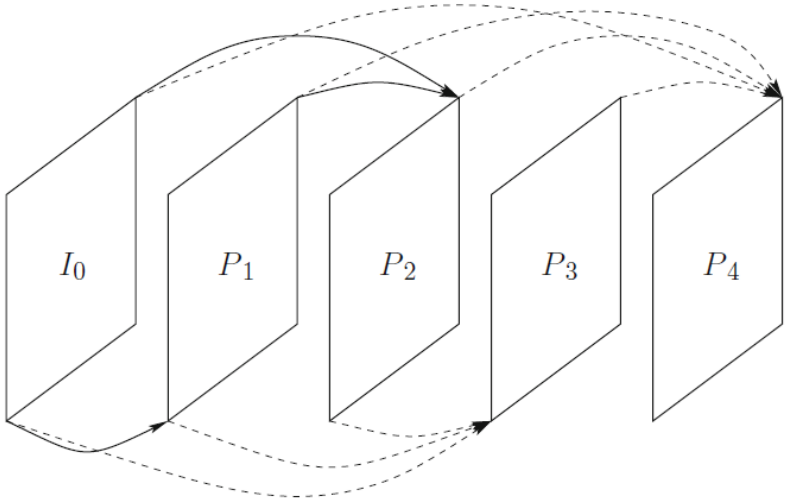


Fig. 12.3: H.264 fractional sample interpolation

H.264 continues to support the classic I-P-B structure. Additional options in Group of Pictures are as follows:

- No B-frames
- Multiple reference frames



- Hierarchical prediction structure

12.1.2 Integer transform

- DCT is known to cause prediction shift because of floating point calculation and rounding errors in the transform and inverse transform. It could be accumulated, causing a large error.
- In H.264, a simple integer-precision 4 × 4 DCT is utilized to eliminates the encoder/decoder mismatch problems.
- H.264 also provides a quantization scheme with nonlinear step-sizes to obtain accurate rate control.

12.1.2 Integer transform (Cont'd)

- 2D DCT can be realized by two consecutive 1D transforms

$$\mathbf{F} = \mathbf{T} \times \mathbf{f} \times \mathbf{T}^T$$

- The 4 x 4 DCT matrix is

$$\mathbf{T_4} = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix}$$

$$a = 1/2$$

$$b = \sqrt{\frac{1}{2}} \cos\frac{\pi}{8}$$

$$c = \sqrt{\frac{1}{2}} \cos\frac{3\pi}{8}$$

12.1.2 Integer transform (Cont'd)

- To derive a scaled 4 × 4 integer transform, we can simply scale the entries of **T4** up and round them to nearest integers

$$\mathbf{H} = \text{round}(\alpha \cdot \mathbf{T_4})$$

- Set $\alpha = 2.5$, it yields

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

12.1.2 Integer transform (Cont'd)

- The H is orthogonal, although its rows no longer have the same norm. The normalization step is postponed into the quantization step.
- The inverse transform is

$$\mathbf{H_{inv}} = \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix}$$

12.1.3 Quantization and Scaling

- Let **f** be the 4 × 4 input matrix, and $\hat{\mathbf{F}}$ the transformed and then quantized output. The forward integer transform, scaling and quantization are

$$\hat{\mathbf{F}} = \text{round}\left[(\mathbf{H} \times \mathbf{f} \times \mathbf{H}^T) \cdot \mathbf{M_f}/2^{15}\right]$$

M_f is the 4 × 4 quantization matrix derived from matrix **m** (see Table 12.1) and quantization parameter *QP*.

12.1.3 Quantization and Scaling (Cont'd)

- $0 \leq QP < 6$,

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

$QP \geq 6$, each element $\mathbf{m}(QP,k)$ is replaced

$$\mathbf{m}(QP\%6,k)/2^{\lfloor QP/6 \rfloor}$$

12.1.3 Quantization and Scaling (Cont'd)

Table 12.1: The matrix **m**

QP	Positions in $\mathbf{M_f}$ (0, 0), (0, 2) (2, 0), (2, 2)	Positions in $\mathbf{M_f}$ (1, 1), (1, 3) (3, 1), (3, 3)	Remaining $\mathbf{M_f}$ positions
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

12.1.3 Quantization and Scaling (Cont'd)

- Let $\tilde{\mathbf{f}}$ be the de-quantized and then inversely transformed result. The scaling, de-quantization, and inverse integer transform are

$$\tilde{\mathbf{f}} = \text{round}\left[(\mathbf{H_{inv}} \times (\hat{\mathbf{F}} \cdot \mathbf{V_i}) \times \mathbf{H_{inv}}^T)/2^6\right]$$

V_i is the 4 × 4 de-quantization matrix derived from matrix **v** (see Table 12.2) and quantization parameter *QP*

12.1.3 Quantization and Scaling (Cont'd)

- $0 \leq QP < 6$

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

$QP \geq 6$, each element $\mathbf{v}(QP,k)$ is replaced by $\mathbf{v}(QP\%6,k) \cdot 2^{\lfloor QP/6 \rfloor}$

12.1.3 Quantization and Scaling (Cont'd)

Table 12.2: The matrix **v**

QP	Positions in $\mathbf{V_i}$ (0, 0), (0, 2) (2, 0), (2, 2)	Positions in $\mathbf{V_i}$ (1, 1), (1, 3) (3, 1), (3, 3)	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

12.1.4 H.264 Examples

72	82	85	79
74	75	86	82
84	73	78	80
77	81	76	84

$\mathbf{f}$

13107	8066	13107	8066
8066	5243	8066	5243
13107	8066	13107	8066
8066	5243	8066	5243

$\mathbf{M_f}$

507	-12	-2	2
0	-7	-14	5
2	0	-8	-11
-1	8	4	3

$\hat{\mathbf{F}}$

10	13	10	13
13	16	13	16
10	13	10	13
13	16	13	16

$\mathbf{V_i}$

72	82	85	79
74	75	86	82
84	73	78	80
77	81	76	84

$\tilde{\mathbf{f}}$

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$\boldsymbol{\epsilon} = \mathbf{f} - \tilde{\mathbf{f}}$

(a) QP = 0, no scale-down of **F** values in the quantization step, so no reconstruction error.

12.1.4 H.264 Examples (Cont'd)

72	82	85	79
74	75	86	82
84	73	78	80
77	81	76	84

$\mathbf{f}$

6554	4033	6554	4033
4033	2622	4033	2622
6554	4033	6554	4033
4033	2622	4033	2622

$\mathbf{M_f}$

254	-6	-1	1
0	-4	-7	3
1	0	-4	-6
0	4	2	1

$\hat{\mathbf{F}}$

20	26	20	26
26	32	26	32
20	26	20	26
26	32	26	32

$\mathbf{V_i}$

72	82	85	79
74	75	86	82
84	74	78	80
77	82	76	84

$\tilde{\mathbf{f}}$

0	0	0	0
0	0	0	0
0	-1	0	0
0	-1	0	0

$\epsilon = \mathbf{f} - \tilde{\mathbf{f}}$

(b) QP = 6. Compared to for *QP* = 0, $\mathbf{M_f}$ are reduced in half, $\mathbf{V_i}$ are about twice. Slight error.

12.1.4 H.264 Examples (Cont'd)

72	82	85	79
74	75	86	82
84	73	78	80
77	81	76	84

$\mathbf{f}$

1638	1008	1638	1008
1008	655	1008	655
1638	1008	1638	1008
1008	655	1008	655

$\mathbf{M_f}$

63	-2	0	0
0	-1	-2	1
0	0	-1	-1
0	1	0	0

$\hat{\mathbf{F}}$

80	104	80	104
104	128	104	128
80	104	80	104
104	128	104	128

$\mathbf{V_i}$

70	81	86	78
73	73	85	83
82	75	77	82
77	79	74	85

$\tilde{\mathbf{f}}$

2	1	-1	1
1	2	1	-1
2	-2	1	-2
0	2	2	-1

$\boldsymbol{\epsilon} = \mathbf{f} - \tilde{\mathbf{f}}$

(c) QP = 18, larger errors.

12.1.4 H.264 Examples (Cont'd)

$$\mathbf{f} = \begin{bmatrix} 72 & 82 & 85 & 79 \\ 74 & 75 & 86 & 82 \\ 84 & 73 & 78 & 80 \\ 77 & 81 & 76 & 84 \end{bmatrix} \qquad \mathbf{M_f} = \begin{bmatrix} 410 & 252 & 410 & 252 \\ 252 & 164 & 252 & 164 \\ 410 & 252 & 410 & 252 \\ 252 & 164 & 252 & 164 \end{bmatrix} \qquad \hat{\mathbf{F}} = \begin{bmatrix} 16 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{V_i} = \begin{bmatrix} 320 & 416 & 320 & 416 \\ 416 & 512 & 416 & 512 \\ 320 & 416 & 320 & 416 \\ 416 & 512 & 416 & 512 \end{bmatrix} \qquad \tilde{\mathbf{f}} = \begin{bmatrix} 80 & 80 & 80 & 80 \\ 80 & 80 & 80 & 80 \\ 80 & 80 & 80 & 80 \\ 80 & 80 & 80 & 80 \end{bmatrix} \qquad \boldsymbol{\epsilon} = \mathbf{f} - \tilde{\mathbf{f}} = \begin{bmatrix} -8 & 2 & 5 & -1 \\ -6 & -5 & 6 & 2 \\ 4 & -7 & -2 & 0 \\ -3 & 1 & -4 & 4 \end{bmatrix}$$

(d) QP = 30, all AC coefficients become zero, very large errors, unacceptable!

12.1.5 Intra coding

- Intra-coded macroblocks are predicted using some of the neighboring reconstructed pixels
- Different intra prediction block sizes (4 × 4 or 16 × 16) can be chosen
- Nine prediction modes for the 4 × 4 blocks
- Four prediction modes for the 16 × 16 blocks

12.1.5 Intra coding (Cont'd)

- For each prediction mode, the predicted value and the actual value will be compared to produce the prediction error.
- The mode that produces the least prediction error will be chosen as the prediction mode for the block.
- The prediction errors (residuals) are then sent to transform coding where the 4 × 4 integer transform is employed.

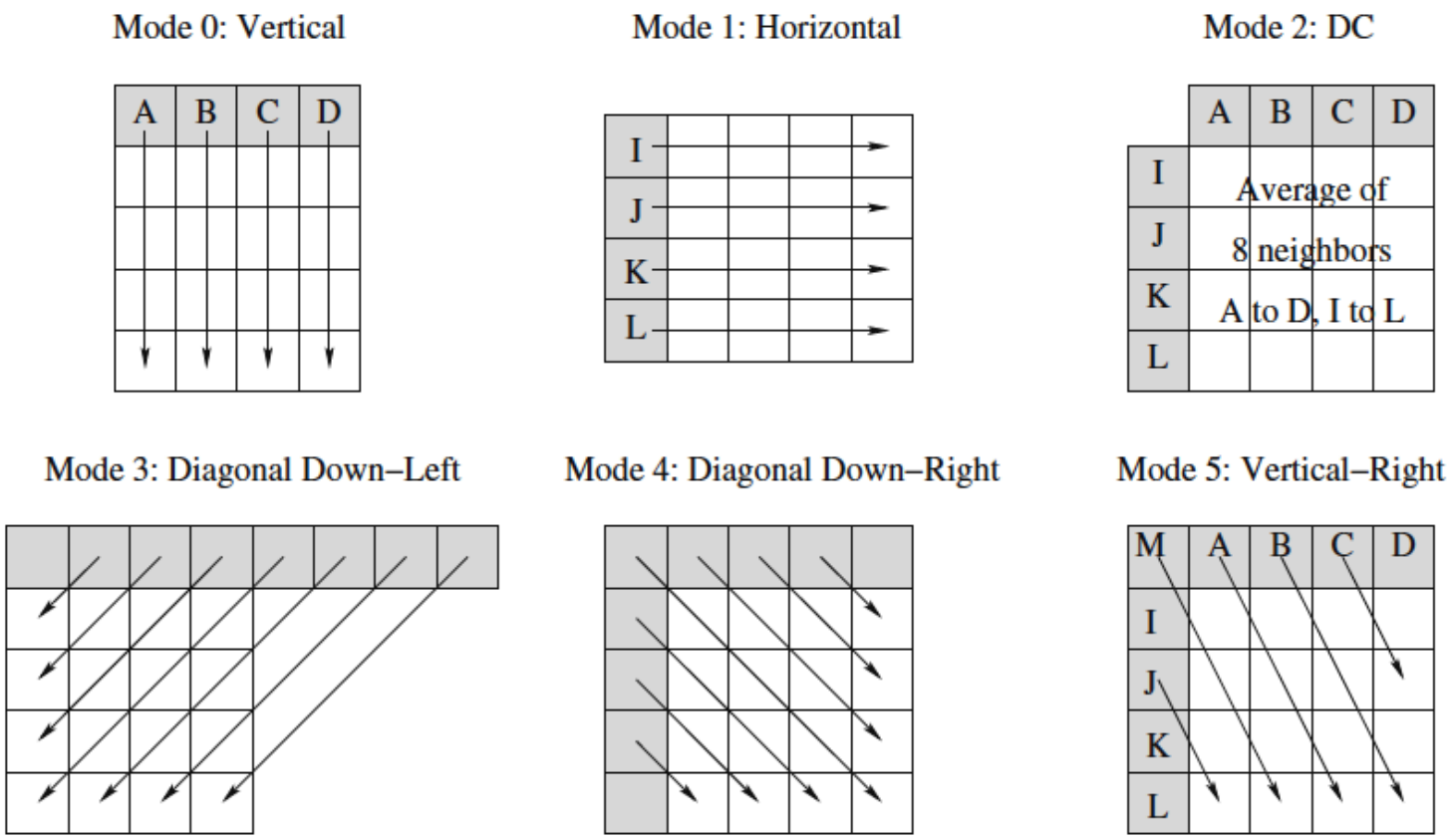


Fig. 12.7: The first six of nine intra_4 × 4 prediction modes in H.264 intra frame prediction

12.1.6 In-Loop Deblocking Filtering

- H.264 specifies a signal-adaptive deblocking filter in which a set of filters is applied on the 4×4 block edges.
- The deblocking filtering takes place in the loop, after the inverse transform in the encoder, before the decoded block data are fed to Motion Estimation.

In-Loop Deblocking Filtering (Cont'd)

- To protect real edges across the block boundary, the deblocking filtering on $p0$ and $q0$ will be applied only if all the following criteria are met:

$$\begin{aligned} |p_0 - q_0| &< \alpha(QP), \\ |p_0 - p_1| &< \beta(QP), \\ |q_0 - q_1| &< \beta(QP), \end{aligned}$$

where α and β are thresholds.

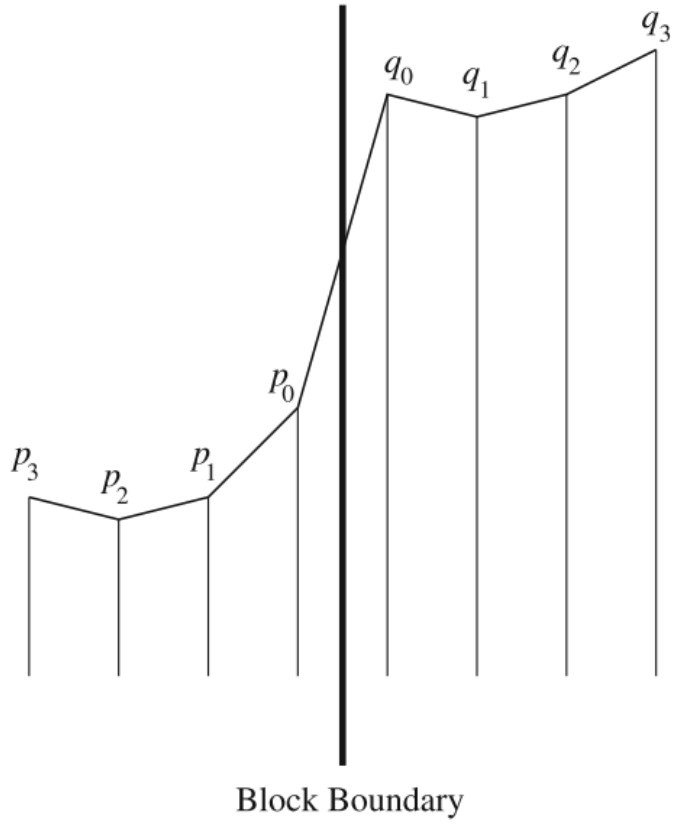


Fig. 12.8: The height for the pixels p0, q0, etc. indicates their value. The function of deblocking filtering is basically smoothing of the block edges.

12.1.7 Exp-Golomb code

- The simplest Exp-Golomb code (0th order) consists of three parts: **[Prefix] [1] [Suffix]**

  The prefix is a sequence of *l* zeros. Given an unsigned (positive) number N to be coded, $l = \lfloor \log_2(N+1) \rfloor$. The suffix S is the binary number $N + 1 - 2^l$ in *l* bits.

- The unsigned numbers are used to indicate, e.g., macroblock type, reference frame index. For signed numbers, they will simply be squeezed in to produce a new set of table entries (Table 12.3).

Table 12.3: The 0th order Exp-Golomb codewords

Unsigned N	Signed N	Codeword
0	0	1
1	1	010
2	−1	011
3	2	00100
4	−2	00101
5	3	00110
6	−3	00111
7	4	0001000
8	−4	0001001
…	…	…

Exp-Golomb code (Cont'd)

- The kth order Exp-Golomb encoding consists of three parts: **[Prefix] [1] [Suffix]**

  The prefix is a sequence of *l* zeros. Given an unsigned (positive) number N to be coded, $l = \lfloor \log_2(N/2^k + 1) \rfloor$ is the binary number $N + 2^k(1 - 2^l)$ in *l*+*k* bits.

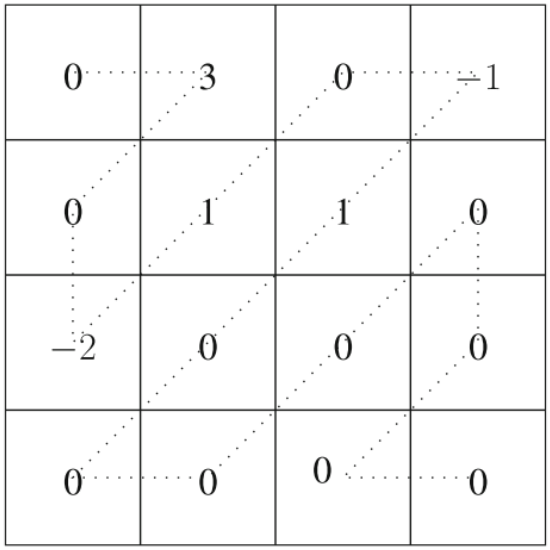
- For example, the *EG*1 code for *N* = 4 is 0110. It is because $l = \lfloor \log_2(4/2^1 + 1) \rfloor = 1,$ the prefix is 0; the suffix is the binary representation of $4 + 2^1(1 - 2^1) = 2,$ in *l*+*k* = 1+1 = 2 bits, it is 10.

Table 12.4: First- and second-order Exp-Golomb codewords

Unsigned N	EG_1 Codeword	EG_2 Codeword
0	10	100
1	11	101
2	0100	110
3	0101	111
4	0110	01000
5	0111	01001
6	001000	01010
7	001001	01011
8	001010	01100
9	001011	01101
10	001100	01110
11	001101	01111
12	001110	0010000
13	001111	0010001
14	00010000	0010010
15	00010001	0010011
...	...	...

12.1.8 CAVLC

- In CAVLC (Context-Adaptive Variable Length Coding), multiple VLC tables are predefined for each data type, and predefined rules predict the optimal VLC table based on the context.
- CAVLC checks several parameters from the block data, including *total number of nonzero coefficients, number of trailing ± 1s, signs of the Trailing 1s*, *sign of magnitude of the other zero coefficients,* … and then generate the coding result.



- After the zigzag scan, the 1-D sequence is:

  0 3 0 –2 1 0 –1 1 0 0 0 0 0 0 0 0.

- The resulting CAVLC is

  0000100 0 1 0 0001 001 0 111 11 10 1 01

following the procedure in Table 12.5.

Table 12.5: CAVLC code generation process

Data	Value	Code
coeff_token	TotalCoeffs = 5, Trailing_1s = 3	0000100
Trailing_1 [4] Sign	+	0
Trailing_1 [3] Sign	−	1
Trailing_1 [2] Sign	+	0
Level [1]	−2 (SuffixLength = 0)	0001 (prefix)
Level [0]	3 (SuffixLength = 1)	001 (prefix) 0 (suffix)
Total zeros	3	111
run_before [4]	zeros_left = 3, run_before = 0	11
run_before [3]	zeros_left = 3, run_before = 1	10
run_before [2]	zeros_left = 2, run_before = 0	1
run_before [1]	zeros_left = 2, run_before = 1	01
run_before [0]	zeros_left = 1, run_before = 1	No code required

12.1.9 CABAC

- For better coding efficiency in H.264 Main and High profiles, Context-Adaptive Binary Arithmetic Coding (CABAC) is used for some data and quantized residual coefficients.
- CABAC has three components, binarization, context modeling, and binary arithmetic coding.
- The implementation of CABAC has a tremendous amount of details. Referred to Marpe's CSVT 03 paper for detailed discussions.

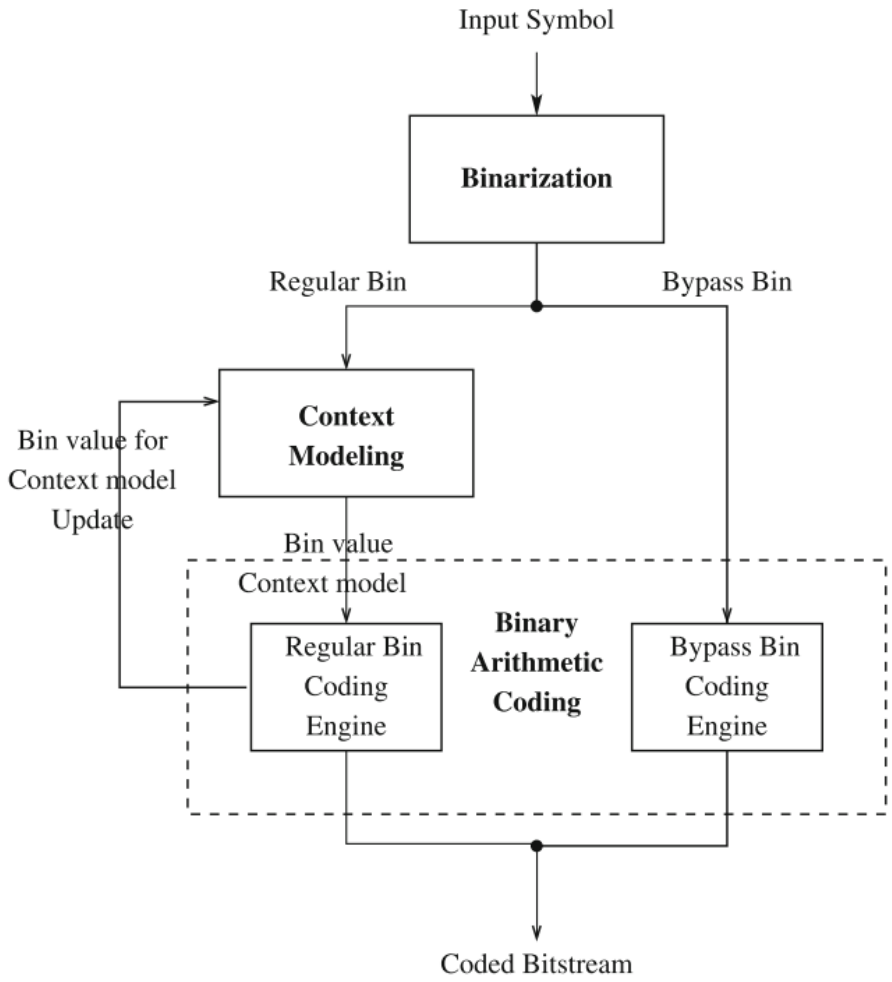


Fig. 12.10: Block diagram of CABAC in H.264

12.1.10 H.264 profiles

- Four profiles for different applications.
- Baseline Profile for real-time conversational applications, e.g., teleconference
  - **Arbitrary slice order (ASO).** The decoding order of slices within a picture may not follow monotonic increasing order.
  - **Flexible macroblock order (FMO).** Macroblocks can be decoded in any order.
  - **Redundant slices.** Redundant copies of the slices can be decoded, to further improve error resilience.

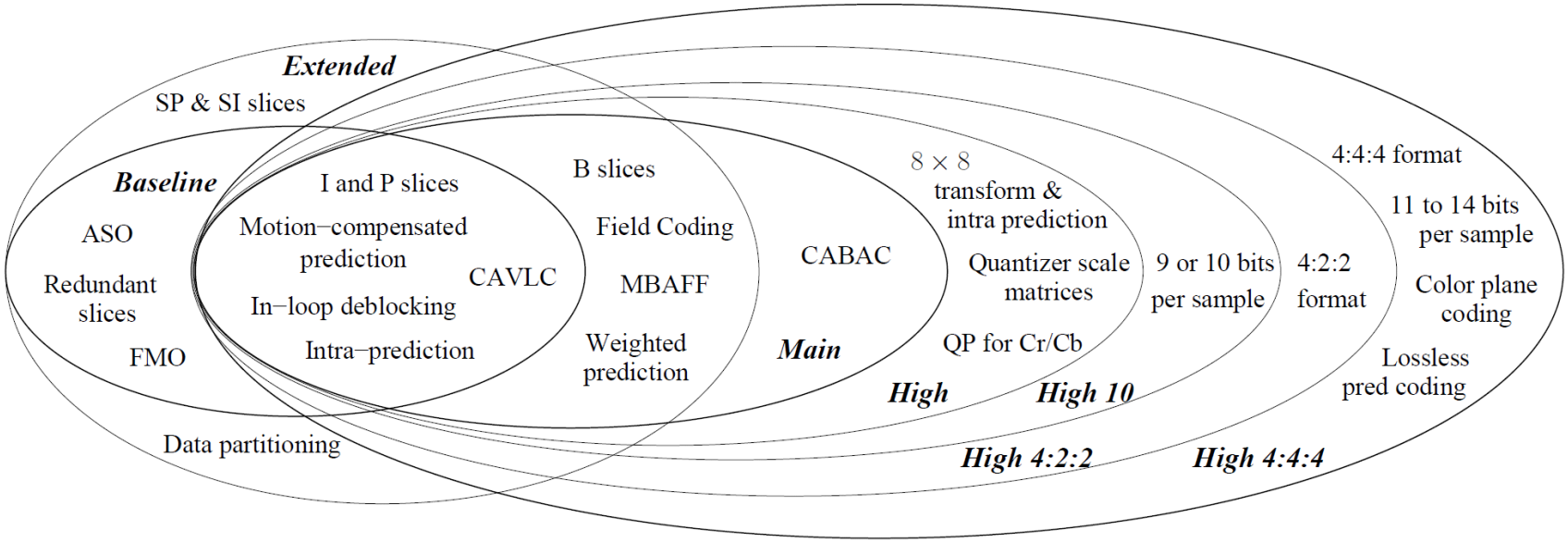


Fig. 12.11: H.264 profiles

12.1.10 H.264 profiles (Cont'd)

- The main profile defined represents non-low-delay applications such as standard definition (SD) digital broadcast TV and stored-medium.
  - **B slices.** Bi-predicted pictures as reference frames
  - **CABAC.** replaces VLC-based entropy coding with binary arithmetic coding with a different adaptive model
  - **Weighted Prediction.** Global weights (multiplier and an offset) for modifying the motion-compensated prediction samples can be specified
- Extended profile and high profile

12.1.11 H.264 Scalable Video Coding

- The Scalable Video Coding (SVC) provides the bit stream scalability which is especially important for multimedia data transmission through various networks that may have very different bandwidths.
- H.264/AVC SVC provides temporal scalability, spatial scalability, quality scalability, and their possible combinations. Compared to previous standards, the coding efficiency is greatly improved.

12.1.12 H.264 Multi-view Video Coding

- Multi-view Video Coding (MVC) is an emerging issue. It has potential applications in some new areas such as Free Viewpoint Video (FVV) where users can specify their preferred views.
- The two most important features in MVC is the **interview prediction**, and **Hierarchical B Pictures.**

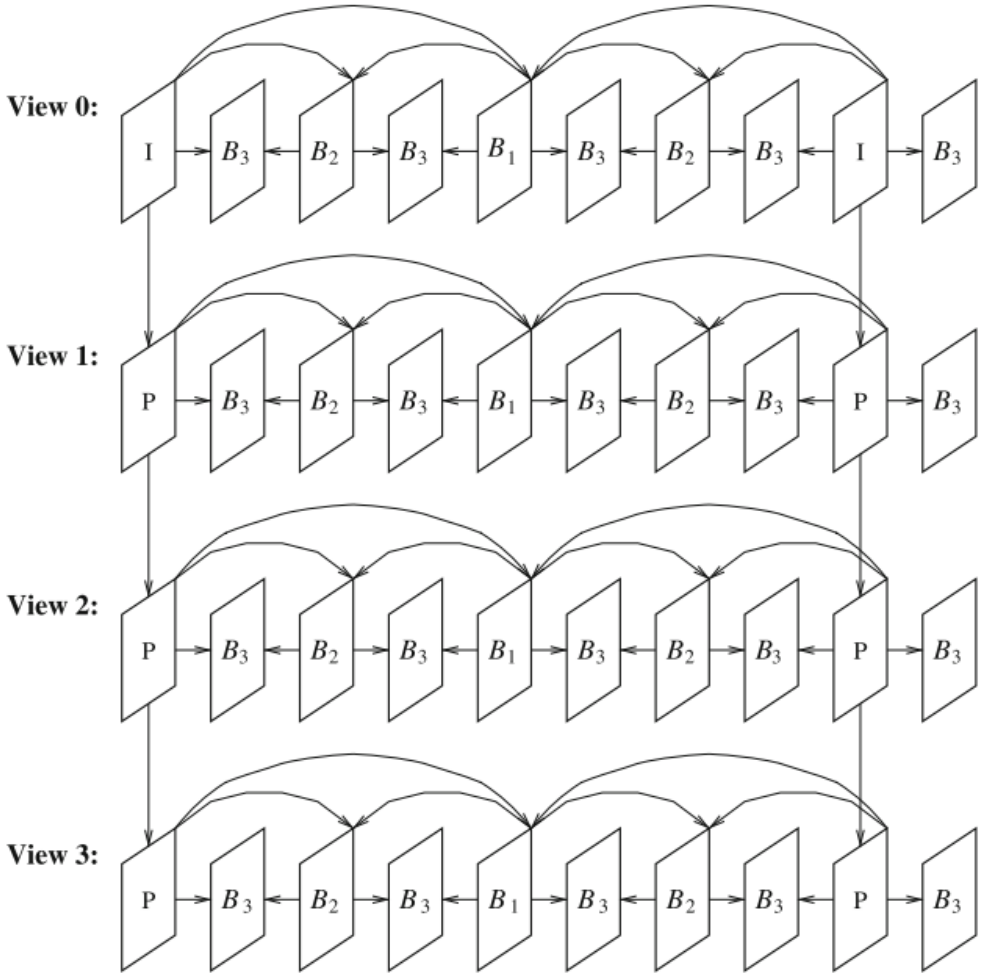


Fig. 12.12: H.264 MVC prediction structure

12.2 Overview of H.265

- HEVC (High Efficiency Video Coding) C became MPEG-H Part 2 (ISO/IEC 23008-2). It is also known as ITU-T Recommendation H.265.
- The development of this new standard was largely motivated by two factors: (a) The need to further improve coding efficiency due to ever increasing video resolution. (b) The need to speed up the more complex coding/decoding methods by exploiting the increasingly available parallel processing devices and algorithms.

12.1 Overview of H.265 (Cont'd)

- Main features of H.265 are:
  - Variable block-size motion compensation, from 4 × 4 up to 64 × 64 in luma images implemented by a quadtree
  - Exploration of parallel processing.
  - Integer transform in various sizes, from 4 × 4 to 32 × 32.
  - Improved interpolation methods for the quarter-pixel
  - Expanded directional spatial prediction for intra coding.
  - The potential use of DST in luma intra coding.
  - In-loop filters including deblocking-filtering and SAO
  - Only CABAC

12.2.1 Motion compensation

- H.265 does not use the simple and fixed structure of macroblocks. Instead, a **quadtree hierarchy** of various blocks is introduced for its efficiency

  - CTB and CTU (Coding Tree Block and Coding Tree Unit). LumaCTB: N×N, N = 16, 32, 64. Chroma CTB is N/2 × N/2. A CTU consists of 1 luma CTB and 2 chroma CTBs.

  - CB (Coding block) is a square block that can be as small as 8 × 8 in luma and 4 × 4 in chroma images. CBs in CTB are coded in Z-order. One luma CB and two chroma CBs form a CU.

12.2.1 Motion compensation (Cont'd)

- Quadtree hierarchy (Cont'd)
  - PB and PU (Prediction Block and Prediction Unit): A CB can be further split into PBs for the purpose of prediction. The prediction mode for a CU can be intra-picture (spatial) or inter-picture (temporal). The PU contains the luma and chroma PBs and their prediction syntax.
  - TB and TU (Transform Block and Transform Unit): A CB can be further split into TBs for the purpose of transform coding of residual errors. A TU consists of TBs from luma and chroma images.

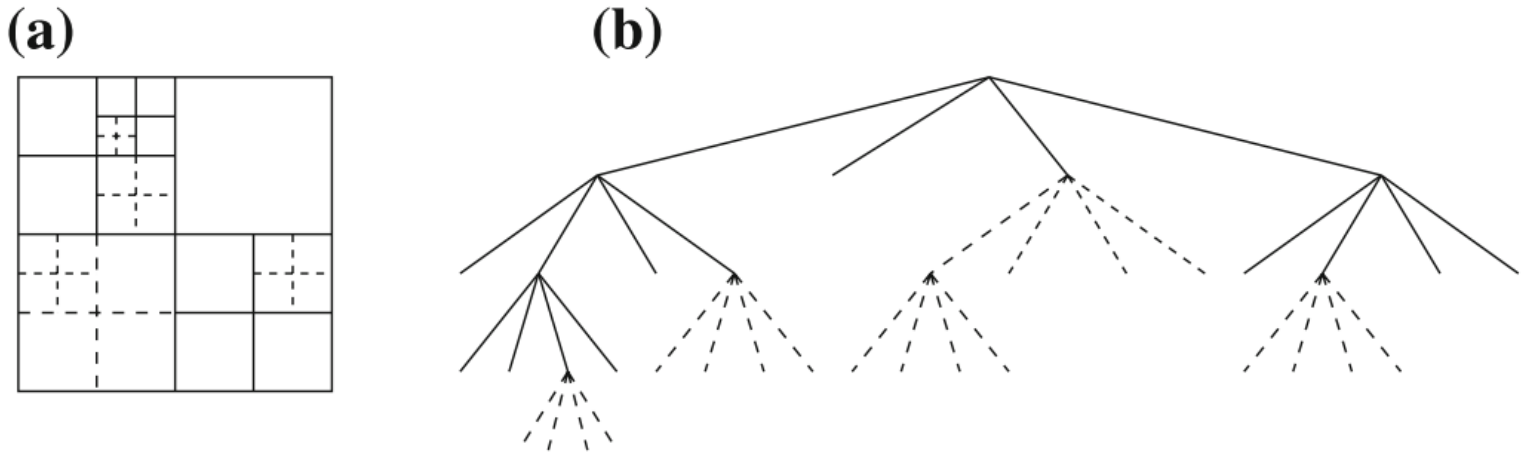


Fig. 12.13: Partitioning of a CTB. (a) The CTB and its partitioning (solid lines CB boundaries, dotted line TB boundaries). (b) The corresponding quadtree, the original CTB is 64 × 64 and the smallest TB is 4 × 4.

Slices and Tiles

- H.265 supports Slices of any length consisting of a sequence of CTUs, I-slices, P-slices, or B-slices.
- A tile is a rectangular structure consisting of CTUs
- An additional feature is the inclusion of the *Wavefront Parallel Processing (WPP)* technology. Basically, rows of CTUs can be processed in parallel, in multiple threads in the wavefront.

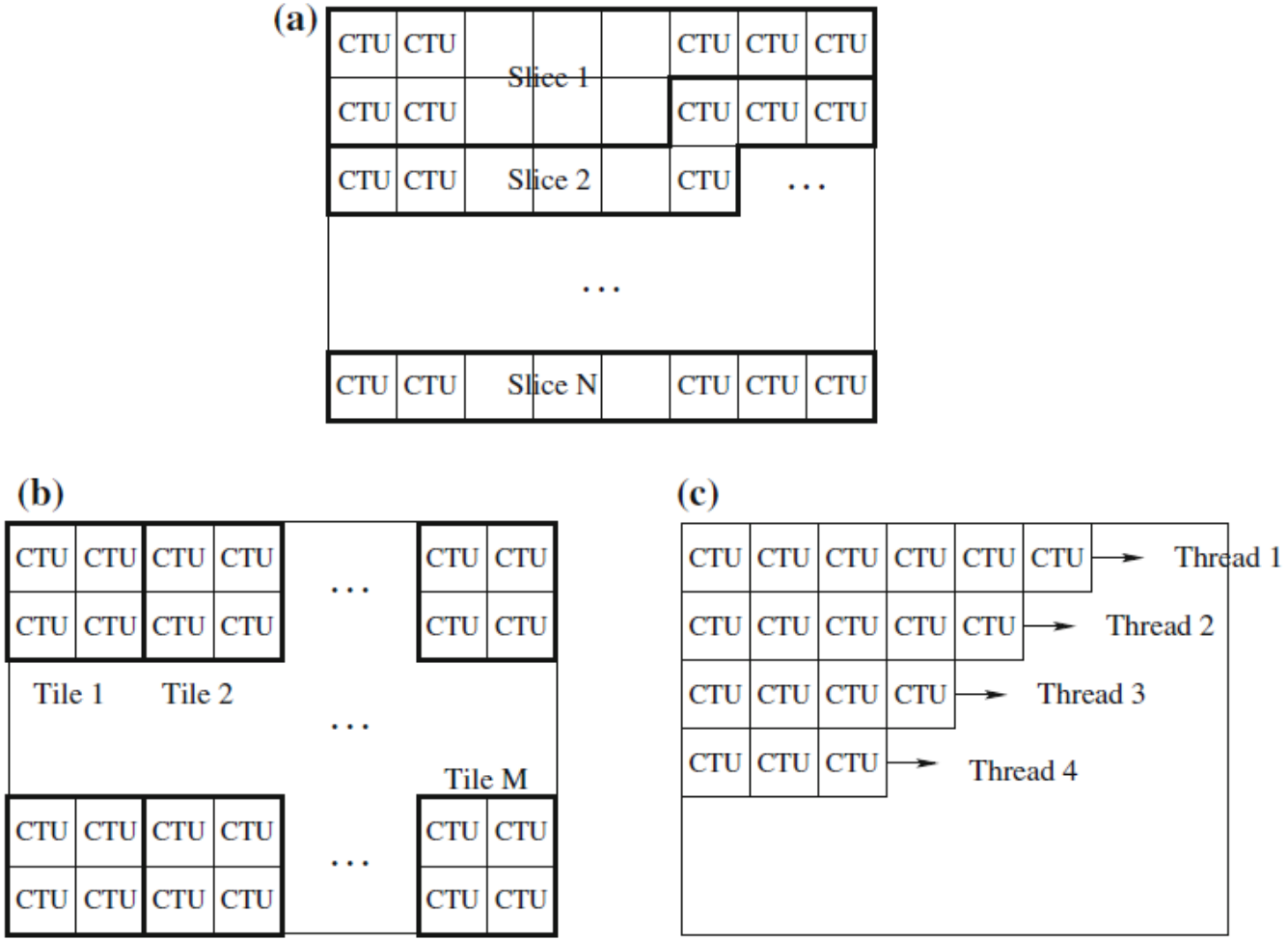


Fig. 12.14: (a) Slices, (**b**) Tiles, (**c**) Wavefronts

Quarter-Pixel in Luma Images

- In inter-picture prediction for luma images, the precision for motion vectors is again at quarter-pixel as in H.264. interpolations.
- An eight-tap filter **hfilter** is used for half-pixel positions and a seven-tap filter **qfilter** is used for quarter-pixel positions.
- All values at subpixel positions are derived through separable filtering steps vertically and horizontally, then averaging to obtain the values at quarter-pixel positions

Fig. 12.15: Interpolation for fractional samples in H.265.

Quarter-Pixel Interpolation

- The value of a, b, c, d, e, f could be derived by

$$a_{i,j} = \sum_{t=-3}^{3} A_{i,j+t} \cdot \text{qfilter}[t] \qquad d_{i,j} = \sum_{t=-3}^{3} A_{i+t,j} \cdot \text{qfilter}[t],$$

$$b_{i,j} = \sum_{t=-3}^{4} A_{i,j+t} \cdot \text{hfilter}[t] \qquad e_{i,j} = \sum_{t=-3}^{4} A_{i+t,j} \cdot \text{hfilter}[t],$$

$$c_{i,j} = \sum_{t=-2}^{4} A_{i,j+t} \cdot \text{qfilter}[1-t] \qquad f_{i,j} = \sum_{t=-2}^{4} A_{i+t,j} \cdot \text{qfilter}[1-t].$$

Quarter-Pixel Interpolation (Cont'd)

- The eight-tap hfilter is symmetric, so it works well for the half-pixel positions which are in the middle of pixels that are on the image grid.
- The seven-tap qfilter is asymmetric, well-suited for the quarter-pixel positions not in the middle.
- The other subpixel samples can be obtained from the vertically nearby *a*, *b*, or *c* pixels as below. To enable 16-bit operations, a right shift of six bits is introduced.

Quarter-Pixel Interpolation (Cont'd)

$$g_{i,j} = \left(\sum_{t=-3}^{3} a_{i+t,j} \cdot \text{qfilter}[t] \right) \gg 6, \qquad p_{i,j} = \left(\sum_{t=-2}^{4} b_{i+t,j} \cdot \text{qfilter}[1-t] \right) \gg 6,$$

$$j_{i,j} = \left(\sum_{t=-3}^{4} a_{i+t,j} \cdot \text{hfilter}[t] \right) \gg 6, \qquad i_{i,j} = \left(\sum_{t=-3}^{3} c_{i+t,j} \cdot \text{qfilter}[t] \right) \gg 6,$$

$$n_{i,j} = \left(\sum_{t=-2}^{4} a_{i+t,j} \cdot \text{qfilter}[1-t] \right) \gg 6,$$

$$h_{i,j} = \left(\sum_{t=-3}^{3} b_{i+t,j} \cdot \text{qfilter}[t] \right) \gg 6, \qquad m_{i,j} = \left(\sum_{t=-3}^{4} c_{i+t,j} \cdot \text{hfilter}[t] \right) \gg 6,$$

$$k_{i,j} = \left(\sum_{t=-3}^{4} b_{i+t,j} \cdot \text{hfilter}[t] \right) \gg 6, \qquad q_{i,j} = \left(\sum_{t=-2}^{4} c_{i+t,j} \cdot \text{qfilter}[1-t] \right) \gg 6.$$

12.2.2 Integer transform

- As in H.264, transform coding is applied to the prediction error residuals.
- The 2-D transform is accomplished by applying a 1-D transform in the vertical and then horizontal direction, where **f** is the input residual data and **F** is the transformed data. **H** is the Integer Transform matrix that approximates the DCT-matrix.

$$\mathbf{F} = \mathbf{H} \times \mathbf{f} \times \mathbf{H}^T$$

- Transform block sizes of 4 × 4, 8× 8, 16 × 16, and 32 × 32 are supported.

12.2.2 Integer transform (Cont'd)

$$\mathbf{H}_{16\times16} = \begin{bmatrix}
64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\
90 & 87 & 80 & 70 & 57 & 43 & 25 & 9 & -9 & -25 & -43 & -57 & -70 & -80 & -87 & -90 \\
89 & 75 & 50 & 18 & -18 & -50 & -75 & -89 & -89 & -75 & -50 & -18 & 18 & 50 & 75 & 89 \\
87 & 57 & 9 & -43 & -80 & -90 & -70 & -25 & 25 & 70 & 90 & 80 & 43 & -9 & -57 & -87 \\
83 & 36 & -36 & -83 & -83 & -36 & 36 & 83 & 83 & 36 & -36 & -83 & -83 & -36 & 36 & 83 \\
80 & 9 & -70 & -87 & -25 & 57 & 90 & 43 & -43 & -90 & -57 & 25 & 87 & 70 & -9 & -80 \\
75 & -18 & -89 & -50 & 50 & 89 & 18 & -75 & -75 & 18 & 89 & 50 & -50 & -89 & -18 & 75 \\
70 & -43 & -87 & 9 & 90 & 25 & -80 & -57 & 57 & 80 & -25 & -90 & -9 & 87 & 43 & -70 \\
64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 \\
57 & -80 & -25 & 90 & -9 & -87 & 43 & 70 & -70 & -43 & 87 & 9 & -90 & 25 & 80 & -57 \\
50 & -89 & 18 & 75 & -75 & -18 & 89 & -50 & -50 & 89 & -18 & -75 & 75 & 18 & -89 & 50 \\
43 & -90 & 57 & 25 & -87 & 70 & 9 & -80 & 80 & -9 & -70 & 87 & -25 & -57 & 90 & -43 \\
36 & -83 & 83 & -36 & -36 & 83 & -83 & 36 & 36 & -83 & 83 & -36 & -36 & 83 & -83 & 36 \\
25 & -70 & 90 & -80 & 43 & 9 & -57 & 87 & -87 & 57 & -9 & -43 & 80 & -90 & 70 & -25 \\
18 & -50 & 75 & -89 & 89 & -75 & 50 & -18 & -18 & 50 & -75 & 89 & -89 & 75 & -50 & 18 \\
9 & -25 & 43 & -57 & 70 & -80 & 87 & -90 & 90 & -87 & 80 & -70 & 57 & -43 & 25 & -9
\end{bmatrix}$$

- For example, $\mathbf{H_{8x8}}$ can be obtained by using the first 8 entries of Rows 0, 2, 4, 6, … of $\mathbf{H_{16x16}}$. For $\mathbf{H_{4\times4}}$, use the first 4 entries of Rows 0, 4, 8, and 12.

12.2.3 Quantization and Scaling

- Unlike the **H** matrix in H.264, the numbers in the H.265 integer transform matrices, are proportionally very close to the actual values of the DCT basis functions. Hence, the ad hoc scaling factors are no longer needed.
- For quantization, the quantization matrix and the same parameter QP as in H.264 are employed. The range of QP is [0, 51]. Similarly, the quantization step size doubles when the QP value is increased by 6.

12.2.4 Intra Coding

- As in H.264, spatial predictions are used in intra coding in H.265. The neighboring boundary samples from the blocks at the top and/or left of the current block are used for the predictions.
- The transform block ranges from 4×4 to 32×32.
- The possible number of prediction modes is increased from 9 in H.264 to 35 in H.265 because of (a) the potentially larger TB size, and (b) the effort to reduce prediction errors.

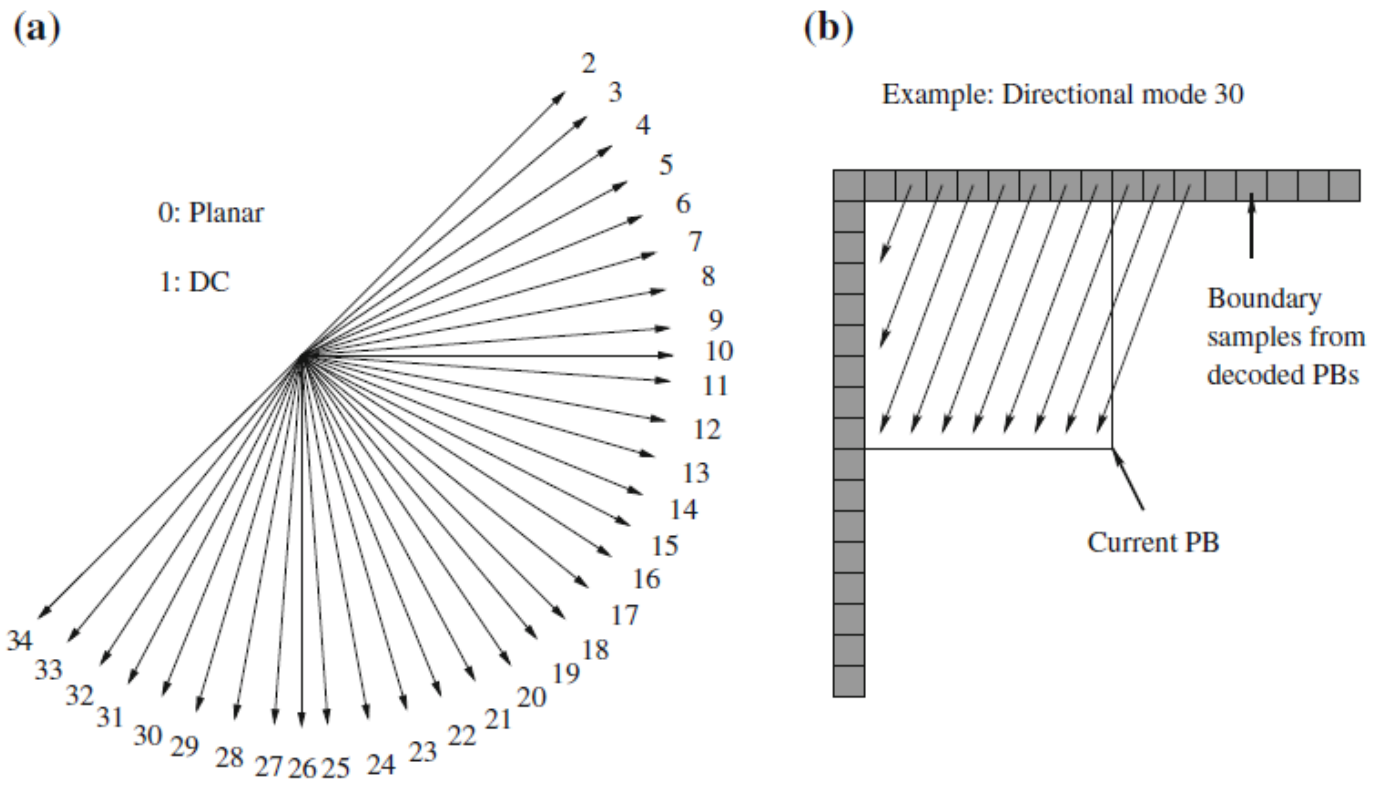


Fig. 12.16: H.265 intra prediction. **a** Modes and intra prediction directions, **b** Intra prediction for an 8 × 8 block

12.2.5 Discrete Sine Transform

- In Intra_4 × 4, for luma residual blocks, HEVC introduced an alternative transform based on one of the variants of the *Discrete Sine Transform (DST)* (the so-called DST-VII). DST is found to cope with this situation better than DCT at the transform coding step.
- The integer matrix for DST can be described by,

$$\mathbf{H_{DST}}[i, j] = \text{round}\left(128 \times \frac{2}{\sqrt{2N+1}} \sin\frac{(2i-1)j\pi}{2N+1}\right)$$

where i and j are indices of row and column

12.2.5 Discrete Sine Transform (Cont'd)

- If N = 4, the H_{DST} is $\mathbf{H_{DST}} = \begin{bmatrix} 29 & 55 & 74 & 84 \\ 74 & 74 & 0 & -74 \\ 84 & -29 & -74 & 55 \\ 55 & -84 & 74 & -29 \end{bmatrix}$

Category 1: the samples for prediction are either all from the left neighbors, or all from the top neighbors of the block.

Category 2: the samples for prediction are from both the top and left neighbors of the current block.

DC: a special prediction mode in which the average of a fixed set of neighboring samples is used.

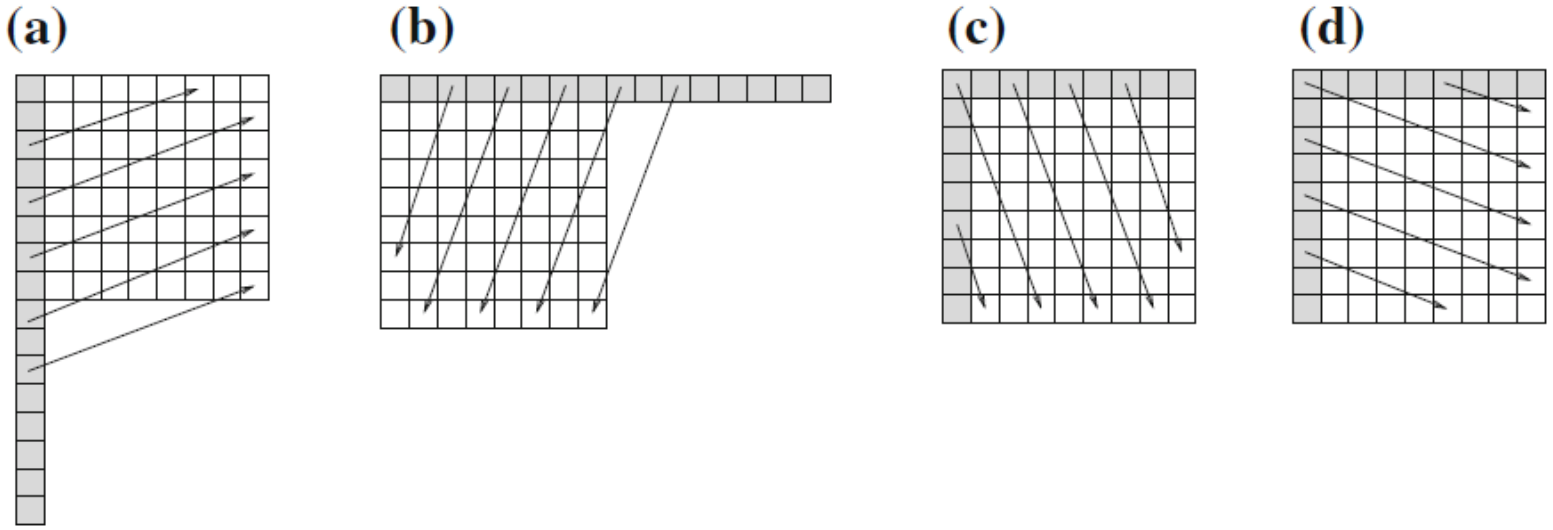


Fig. 12.17: Intra prediction directions in H.265.
(a) and (b): Category 1 – from left or top only,
(c) and (d): Category 2 – from both left and top.

- The prediction error tends to increase for the nodes in the block that are farther away from the top or left neighboring samples.
- In general, DST is better than DCT for this.
- The following could be adopted to alleviate this problem.

Table 12.8: Combining DCT and DST for Intra Coding.

Intra Prediction Category	Neighboring Samples Used	Vertical (Column) Transform	Horizontal (Row) Transform
Category 1	from left only	DCT	DST
Category 1	from top only	DST	DCT
Category 2	from both top and left	DST	DST
DC	special (from a fixed set)	DCT	DCT

12.2.6 In-Loop Filtering

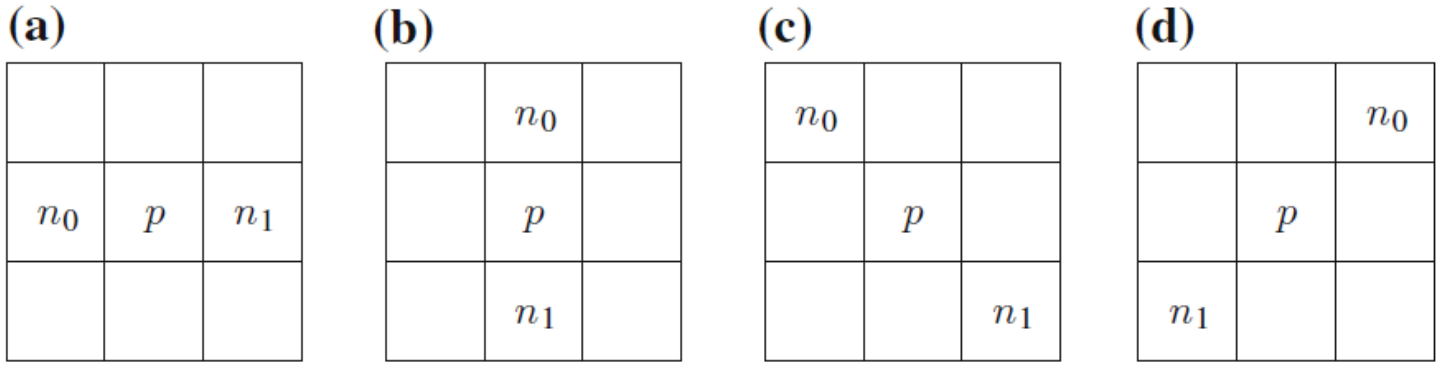
- Similar to H.264, in-loop filtering processes are applied in order to remove the blocky artifacts.
- Instead of applying deblocking filtering to 4×4 blocks as in H.264, it is applied only to edges that are on the 8×8 image grid in H. 265.
- A *Sample Adaptive Offset (SAO)* process is introduced.

12.2.6 In-Loop Filtering (Cont'd)

- Two modes are defined for applying the SAO: *Band offset mode* and *Edge offset mode*.
- In the band offset mode, the range of the sample amplitudes is split into 32 bands. A band offset can be added to the sample values in four of the consecutive bands simultaneously.
- In the Edge Offset Mode, the gradient (edge) information is analyzed first. A positive or negative offset, or zero offset can be added to the sample.

12.2.6 In-Loop Filtering (Cont'd)

- Positive: local minimum ($p < n_0$ & $p < n_1$), or edge pixel ($p < n_0$ & $p = n_1$ or $p = n_0$ & $p < n_1$).
- Negative: local maximum ($p > n_0$ & $p > n_1$), or edge pixel ($p > n_0$ & $p = n_1$ or $p = n_0$ & $p > n_1$).
- Zero: None of the above.



12.2.7 Entropy Coding

- H.265 only uses CABAC in entropy coding, i.e., CAVLC is no longer used.
- Three simple scanning methods are defined to read in the transform coefficients. i.e., *Diagonal up-right*, *Horizontal*, and *Vertical*.
- The goal is still to maximize the length of zero-runs. For different TB size, the scanning method is different.

12.2.8 Special Coding Modes

- **I_PCM** As in H.264, the prediction, transform coding, quantization, and entropy coding steps are bypassed. The PCM-coded (fixed-length) samples are sent directly.
- **Lossless** The residual errors from inter- or intra-predictions are sent to entropy coding directly.
- **Transform skipping** Only the transform step is bypassed. This works for certain data (e.g., computer-generated images or graphics). It can only be applied to 4×4 TBs.

12.2.9 H.265 profiles

- At this point, only three profiles are defined: *Main profile*, *Main 10 profile*, and *Main Still Picture profile*. The sample video formats supported by H.265 main profile is illustrated.

Level(s)	Max Luma picture width × height	Max Luma picture size (samples)	Frame rate (fps)	Main tier max bitrate (Mb/s)
1	176 × 144	36864	15	0.128
2	352 × 288	122880	30	1.5
2.1	640 × 360	245760	30	3.0
3	960 × 540	552960	30	6.0
3.1	1280 × 720	983040	30	10
4 / 4.1	2048 × 1080	2228224	30 / 60	12 / 20
5 / 5.1 / 5.2	4096 × 2160	8912896	30 / 60 / 120	25 / 40 / 60
6 / 6.1 / 6.2	8192 × 4320	35651584	30 / 60 / 120	60 / 120 / 240

12.3 Video Coding Efficiency

- When comparing the coding efficiency of different video compression methods, a common practice is to compare the bitrates of the coded video bitstreams at the same *quality*.
- The video quality assessment approaches can be *objective* or *subjective*: the former is done automatically by computers and the latter requires human judgment.

12.3.1 Objective Assessment

- The most common criterion used for the objective assessment is peak signal-to-noise ratio (PSNR), where I_{max} is the maximum intensity value, e.g., 255 for 8-bit images, *MSE* is the *Mean Squared Error* between the original image I and the compressed image I'.

$$PSNR = 10 \log_{10} \frac{I_{max}^2}{MSE}$$

- For videos, the PSNR is usually the average of the PSNRs of the images in the video sequence.

12.3.1 Objective Assessment

Table 12.10: Average bitrate reductions under equal PSNR. MP—Main Profile, HP—High Profile, ASP—Advanced Simple Profile. For example, when H.265 MP is compared with H264/MPEG-4 AVC HP, a saving of 35.4% is realized.

Video Compression Method	H.264/MPEG-4 AVC HP (%)	MPEG-4 ASP (%)	MPEG-2/H.262 MP (%)
H.265 MP	35.4	63.7	70.8
H.264/MPEG-4 AVC HP	-	44.5	55.4
MPEG-4 ASP	-	-	19.7

12.3.2 Subjective Assessment

- The main advantage of PSNR is that it is easy to calculate. However, it does not necessarily reflect the quality as perceived by humans, i.e., visual quality.
- For subjective assessment, the original and compressed video clips are shown in succession to the human subjects. The subjects are asked to grade the videos by their quality, 0–lowest, 10–highest. The Mean Opinion Score (MOS) is used as the measure for the subjective quality.