Chapter 7
In-Loop Filters in HEVC

Andrey Norkin, Chih-Ming Fu, Yu-Wen Huang, and Shawmin Lei

Abstract The HEVC standard specifies two in-loop filters, a deblocking filter and a sample adaptive offset (SAO). The in-loop filters are applied in the encoding and decoding loops, after the inverse quantization and before saving the picture in the decoded picture buffer. The deblocking filter is applied first. It attenuates discontinuities at the prediction and transform block boundaries. The second in-loop filter, SAO, is applied to the output of the deblocking filter and further improves the quality of the decoded picture by attenuating ringing artifacts and changes in sample intensity of some areas of a picture. The most important advantage of the in-loop filters is improved subjective quality of reconstructed pictures. In addition, using the filters in the decoding loop also increases the quality of the reference pictures and hence also the compression efficiency.

7.1 Introduction

The in-loop filters constitute an important part of the HEVC video coding standard. As seen from the name, the in-loop filters are applied in the encoding and decoding loops, after the inverse quantization but before saving the picture to the decoded picture buffer. HEVC standard specifies two in-loop filters, a deblocking filter, which is applied first, and a sample adaptive offset (SAO), which is applied to the output of the [40] deblocking filter. The deblocking filter attenuates discontinuities at prediction and transform block boundaries [36, 37]. The SAO further improves

A. Norkin (✉)
Ericsson Research, Ericsson, Stockholm, Sweden
e-mail: andrey.norkin@ericsson.com

C.-M. Fu • Y.-W. Huang • S. Lei
MediaTek, Hsinchu, Taiwan
e-mail: chihming.fu@mediatek.com; yuwen.huang@mediatek.com; shawmin.lei@mediatek.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_7,

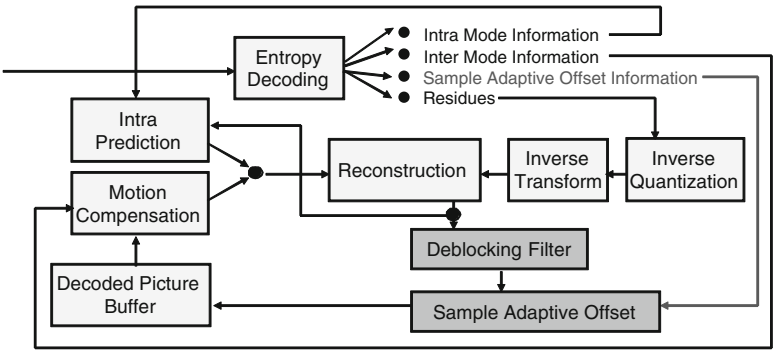

Fig. 7.1 Block diagram of HEVC decoder. Reproduced with permission from [13], © 2012 IEEE

the quality of the decoded picture by reducing the ringing artifacts and changes in the sample intensity of areas of a reconstructed picture. Since the deblocking and SAO attenuate different artifacts, their benefits are additive when used together. An HEVC encoder can turn on and off each of the in-loop filters independently.

Modern video coding standards try to remove as much redundancy from the coded representation of video as possible. One of the sources of redundancy is the temporal redundancy, i.e. similarity between the subsequent pictures in a video sequence. This type of redundancy is effectively removed by motion prediction. Another type of redundancy is spatial redundancy and is removed by intra-prediction from the neighboring samples and spatial transforms. In HEVC, both the motion prediction and transform coding are block-based. The size of motion-predicted blocks varies from 8×4 and 4×8, to 64×64 luma samples, while the size of block transforms and intra-predicted blocks varies from 4×4 to 32×32 samples.

These blocks are coded relatively independently from the neighboring blocks and approximate the original signal with some degree of similarity. Since coded blocks only approximate the original signal, the difference between the approximations may cause discontinuities at the prediction and transform block boundaries [27, 36, 37]. These discontinuities are attenuated by the deblocking filter.

Larger transforms used in HEVC can also introduce more ringing artifacts compared to H.264/AVC that mainly come from quantization error of transform coefficients [17]. In addition to that, HEVC uses 8 or 7-tap fractional luma sample interpolation and 4-tap fractional chroma sample interpolation, while H.264/AVC uses 6-tap and 2-tap for luma and chroma respectively. A higher number of interpolation taps can also lead to more ringing artifacts. These artifacts are corrected by a new filter: sample adaptive offset (SAO). As shown in Fig. 7.1, SAO is applied to the output of the deblocking filter when the deblocking filter is turned on, otherwise, it is applied to the reconstructed picture

There are several reasons for making in-loop filters a part of the standard. In principle, the in-loop filters can also be applied as post-filters. An advantage of

using post-filters is that decoder manufacturers can create post-filters that better suit their needs. However, if the filter is a part of the standard, the encoder has control over the filter and can assure the necessary level of quality by instructing the decoder to enable the filter and specifying the filter parameters. Moreover, since the in-loop filters increase the quality of the reference pictures, they also improve the compression efficiency. A post-filter would also require an additional buffer for filtered pictures, while the output of an in-loop filter can be kept in the decoded picture buffer (DPB). There is also another specific advantage of using the deblocking filter as an in-loop filter compared to the deblocking post-filter. If the deblocking is applied as a post-filter, block artifacts can be copied by motion estimation inside the blocks, which can make the artifacts detection more difficult and increases the deblocking complexity compared to the in-loop filtering, which needs to be applied only to the block boundaries [27, 36, 37].

It is known that the deblocking in-loop filter in H.264/AVC constitutes a significant part of the decoder complexity [27]. Therefore, when designing the in-loop filters in HEVC, efforts have been spent on reducing the loop filters complexity, while still achieving improvements of subjective quality. The HEVC in-loop filters are easily parallelizable, which can bring advantages when running the HEVC decoders and encoders on multi-core architectures.

This chapter is organized as follows. Section 7.2 of the chapter describes the HEVC deblocking filter. Section 7.3 describes SAO, in particular, two types of sample offsets: an edge offset and a band offset. Section 7.4 discusses implementation and parallelization aspects of the HEVC in-loop filters as well as the details of CTU-based implementation of the filter operations. Section 7.5 demonstrates the subjective quality and coding efficiency improvements. Section 7.6 summarizes the main differences between the in-loop filters in HEVC and H.264/AVC while Sect. 7.7 concludes the chapter.

7.2 Deblocking Filter

7.2.1 Block Artifacts in Video Coding

As mentioned in Sect. 7.1, in HEVC both the motion prediction and transform coding are block-based. The size of motion predicted blocks varies from 4×8 and 8×4 to 64×64 luma samples, while the size of block transforms and intra-predicted blocks varies from 4×4 to 32×32 samples. These blocks are coded relatively independently from their neighboring blocks and approximate the original signal with some degree of similarity. Since coded blocks only approximate the original signal, the difference between the approximations may cause discontinuities at the prediction and transform block boundaries. For example, motion prediction of the adjacent blocks may come from the non-adjacent areas of a reference picture (see Fig. 7.2) or even from different reference pictures. In case of non-overlapping block transforms,

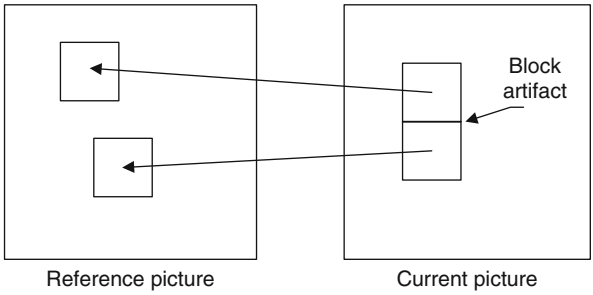

Fig. 7.2 Block artifact may be created when adjacent blocks are predicted from non-adjacent areas in the reference picture

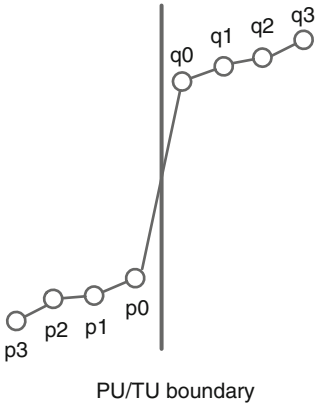

Fig. 7.3 Example of block artifact in one dimension [37]

used in HEVC, coarse quantization can also create discontinuities at the block boundaries. In highly detailed areas with high-frequency content, such artifacts can be masked by the human visual system. However, in the smooth areas, discontinuities between the blocks are easily noticed by a viewer and may cause significant degradation of the perceived video quality. The example of a block artifact in one dimension is shown in Fig. 7.3. The horizontal axis shows the sample positions along a horizontal or vertical 1-D line, and the vertical axis shows the sample values.

Deblocking filter attenuates the artifacts in the areas, where they are mostly visible, i.e. in the smooth, uniform areas. The excessive filtering in the highly detailed areas should be avoided since it can cause undesirable blurring. The artifacts in those areas are rarely noticed by the human eye, while it is also more difficult to determine whether the discontinuity is caused by a block boundary or belongs to the original signal [27]. Therefore, an important part of the deblocking filter is the deblocking filtering decisions, which determine whether a particular part of a block boundary is to be filtered. In these decisions, the HEVC deblocking filter uses the mode and motion information from the decoded bitstream as well as analyses the values of reconstructed samples on the sides of the block boundary. The strength of the deblocking filter can also be adjusted by the encoder on the picture and the slice basis.

Section 7.2.2 provides a description of the HEVC deblocking filer, while Sect. 7.5.1 discusses the coding efficiency and subjective quality improvements

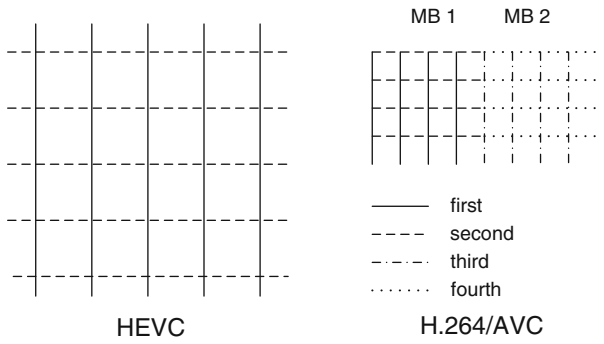

Fig. 7.4 Order of boundaries processing in HEVC and H.264/AVC deblocking. In each group (first to fourth), boundaries are processed from left to right and from top to bottom. In HEVC all vertical (horizontal) boundaries can be processed in parallel

brought by HEVC deblocking filtering. The deblocking filter complexity and parallelization aspects are addressed in Sect. 7.4.1.

7.2.2 HEVC Deblocking Filter Description

7.2.2.1 Decisions to Filter a Block Boundary

As a compromise between the subjective quality and computational complexity, the HEVC deblocking filter in case of 4:2:0 chroma subsampling is only applied to the block boundaries that lie at the luma and chroma sample positions that are multiples of eight (in H.264/AVC the deblocking is applied on the 4×4 luma and chroma sample grid). Since the deblocking filtering is only applied to the boundaries between the coding units (CU), prediction units (PU), or transform units (TU) and not to the inside areas, the average complexity of the HEVC deblocking is further decreased compared to the H.264/AVC since HEVC can use larger block sizes.

In HEVC deblocking, the vertical boundaries in a picture are filtered first, followed by the horizontal boundaries. In a coding unit, the vertical boundaries between the coding blocks are processed starting from the left-most boundary towards the right-hand side. The horizontal boundaries are processed starting from the top-most boundary towards the bottom. In contrast to that, the H.264/AVC deblocking filter operates on a macroblock (MB) basis, where the four vertical boundaries in an MB are processed sequentially starting from the left-most MB boundary, and then the four horizontal MB boundaries are processed starting from the top-most MB boundary. The order of processing of the block boundaries in HEVC and H.264/AVC is compared in Fig. 7.4. One can see that the filtering order in HEVC deblocking is more regular than that in H.264/AVC. Moreover, since the deblocking of one vertical (horizontal) boundary in HEVC does not affect deblocking of other vertical (horizontal) boundaries because of only filtering the boundaries on the 8×8 sample grid, all the vertical (horizontal) boundaries can be

Table 7.1 Boundary strength (*Bs*) derivation [37]

Conditions	Bs
At least one of the adjacent blocks is intra	2
At least one of the adjacent blocks has non-zero transform coefficients	1
Absolute difference between the motion vectors that belong to the adjacent blocks is greater than or equal to one integer luma sample	1
Motion prediction in the adjacent blocks refers to different reference pictures or number of motion vectors is different	1
Otherwise	0

processed in parallel. This allows better parallelization which is addressed in more detail in Sect. 7.4.1. The parallelization of H.264/AVC deblocking filtering is more complicated since filtering of a block boundary affects the deblocking decisions at a parallel block boundary, and the vertical and horizontal filtering operations are alternating.

A decision whether to filter a block boundary uses the bitstream information such as prediction modes and motion vectors. Some coding conditions are more likely to create strong block artifacts, which are represented by a so-called *boundary strength* (*Bs*) variable that is assigned to every block boundary and is determined as in Table 7.1. The deblocking is only applied to the block boundaries with *Bs* greater than zero for a luma component and *Bs* greater than 1 for chroma components. Higher values of *Bs* enable stronger filtering by using higher clipping parameter values. The *Bs* derivation conditions reflect the probability that the strongest block artifacts appear at the intra-predicted block boundaries. The conditions also enable luma deblocking when there is possibility of block artifacts caused by quantization and by prediction from non-adjacent areas in a reference picture. Not filtering block boundaries with *Bs* equal to zero avoids multiple repetitive filtering of static areas where the samples are just copied from one picture to another. In chroma deblocking, only the block boundaries adjacent to intra-predicted blocks are filtered, which reduces the deblocking complexity while still removing the strongest block artifacts. The algorithm for *Bs* derivation is explained in a flowchart in Fig. 7.5.

For luma block boundaries with non-zero *Bs* values, the signal on the sides of the block boundary is evaluated to decide whether the deblocking should be applied. For chroma block boundaries, no further evaluation is performed.

The deblocking decisions are made separately for each four-sample segment of a block boundary (see Fig. 7.6). Since the deblocking needs to attenuate visible artifacts in smooth areas, the deblocking decisions evaluate whether the signal at the sides of the block boundary is smooth, i.e. if the signal is flat or has a shape of an inclined plane (ramp) [36, 37, 43]. The deblocking filtering is applied to a block boundary if the following expression is evaluated to be true:

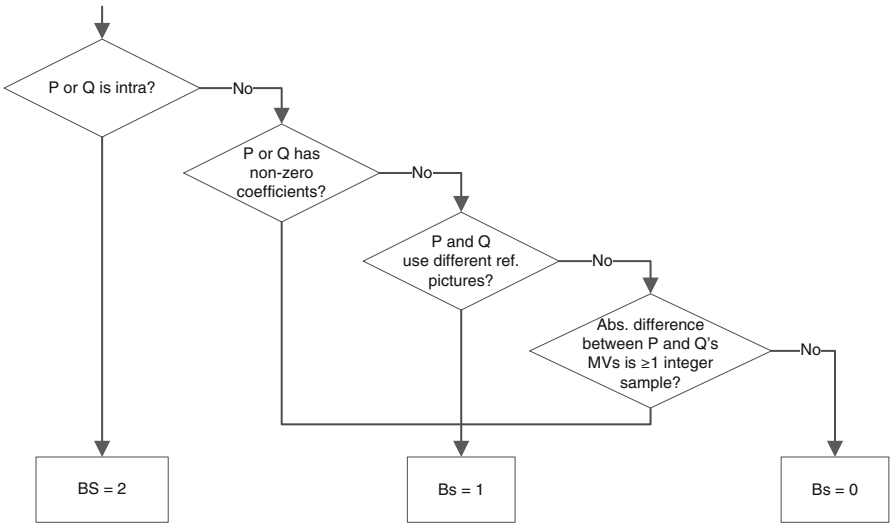

Fig. 7.5 Derivation of boundary strength (*Bs*) for block boundary. P denotes the left (or top) block and Q denotes the right (or bottom) block at the block boundary

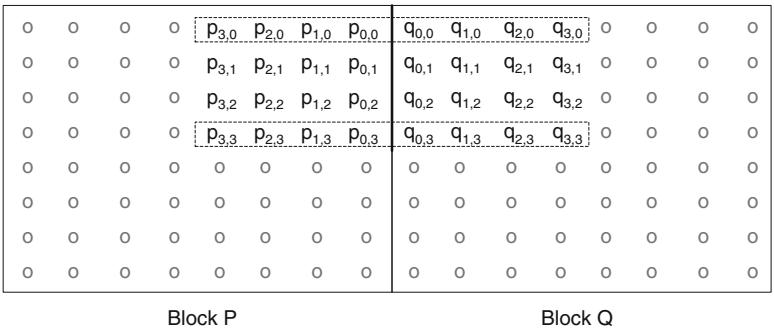

Fig. 7.6 Four-sample segment of vertical block boundary between adjacent blocks P and Q. Deblocking decision are made based on lines 0 and 3 (*dashed*)

$$|p_{2,0} - 2p_{1,0} + p_{0,0}| + |p_{2,3} - 2p_{1,3} + p_{0,3}| + |q_{2,0} - 2q_{1,0} + q_{0,0}| + |q_{2,3} - 2q_{1,3} + q_{0,3}| < \beta, \tag{7.1}$$

where the threshold β depends on the quantization parameters QP and is derived from a look-up table. Equation (7.1) is used to check how much the signal on each side of the block boundary deviates from a straight line (ramp). The equations in this book chapter are given for the case of a vertical block boundary as in Fig. 7.6. The equations for a horizontal block boundary may be obtained from the equations for the vertical boundary by swapping the row and column indices.

The HEVC deblocking has two filtering modes: a normal filtering mode and a strong filtering mode. A decision between these two modes is done for each four-sample segment of a block boundary. The strong filtering is applied to the block boundary if all of the following conditions are true for lines $i = 0$ and $i = 3$ (see Fig. 7.6) [32, 37, 43].

$$|p_{2,\mathrm{i}} - 2p_{1,\mathrm{i}} + p_{0,\mathrm{i}}| + |q_{2,\mathrm{i}} - 2q_{1,\mathrm{i}} + q_{0,\mathrm{i}}| < \beta/8, \tag{7.2}$$

$$|p_{3,\mathrm{i}} - p_{0,\mathrm{i}}| + |q_{0,\mathrm{i}} - q_{3,\mathrm{i}}| < \beta/8, \tag{7.3}$$

$$|p_{0,\mathrm{i}} - q_{0,\mathrm{i}}| < 2.5t_C. \tag{7.4}$$

If all of the (7.2)–(7.4) are true, the strong filtering is applied, otherwise, the normal deblocking filter is applied. The threshold parameter t_C is the clipping parameter described later in this section. Equation (7.4) makes sure that the step between the sample values at the sides of the block boundary is small, while (7.2) checks that there are no significant signal variations at the sides of the boundary, and (7.3) verifies that the signal on both sides is flat.

The deblocking filtering decisions for a block boundary including the decisions between the strong and the normal filtering are summarized in a flowchart in Fig. 7.7.

7.2.2.2 Normal Filtering Mode

When a normal deblocking filtering mode is used, the following conditions are evaluated to decide how many samples are modified at each side of the block boundary. Condition in (7.5) determines how many samples from the block boundary are modified in block P, while condition in (7.6) determines how many samples are modified in block Q (see Fig. 7.6) [36]. The decisions use the same principle as decision (7.1). The smoother the signal on the side of the block boundary, the more filtering is applied [35].

$$|p_{2,0} - 2p_{1,0} + p_{0,0}| + |p_{2,3} - 2p_{1,3} + p_{0,3}| < 3/16\,\beta, \tag{7.5}$$

$$|q_{2,0} - 2q_{1,0} + q_{0,0}| + |q_{2,3} - 2q_{1,3} + q_{0,3}| < 3/16\ \beta. \tag{7.6}$$

If (7.5) is true, two samples from the block boundary are modified in block P, otherwise, one sample is modified. If (7.6) is true, two samples from the block boundary are modified in block Q, otherwise, one sample is modified. The decisions are made for each side of the block boundary independently, i.e. one sample may be filtered on one side of the block boundary, and two samples on the other side.

When condition in (7.1) is true for a four-sample segment of the block boundary, the deblocking filtering operations are subsequently applied to each of the four lines

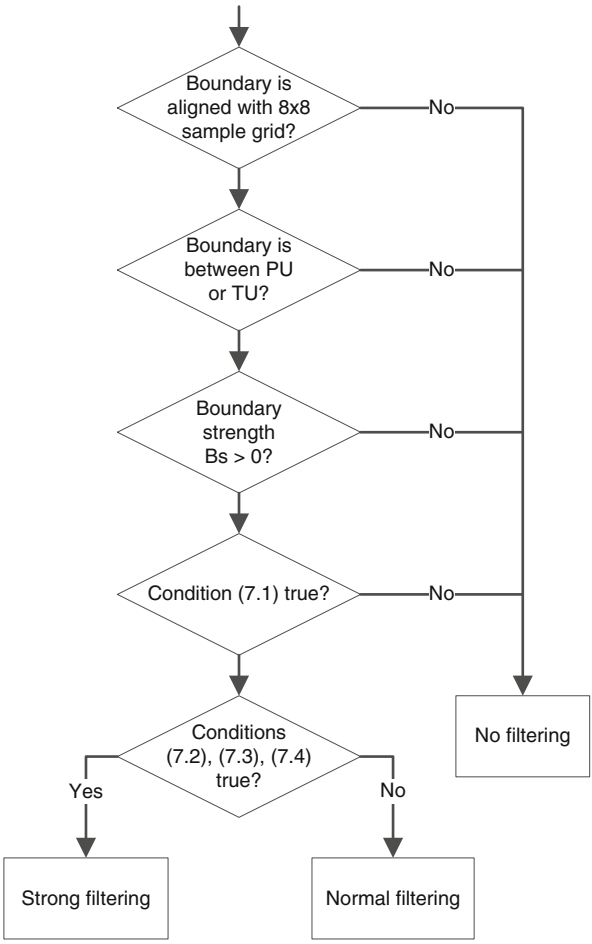

Fig. 7.7 Decisions for each four-sample segment of block boundary. *PU* prediction unit, *TU* transform unit [37]

crossing the block boundary. Since condition in (7.1) is evaluated true for the signal that forms a perfect ramp passing across the block boundary (such as a gradual change in the luma component), the deblocking in the normal filtering mode is designed to not modify the ramp. The filtered sample values p_0' and q_0' (the row index j is omitted for brevity) are determined by adding or subtracting an offset value $\triangle_0$ to each of the sample values:

$$p_0' = p_0 + \Delta_0, \tag{7.7}$$

$$q_0' = q_0 - \Delta_0, \tag{7.8}$$

where the value of $\triangle_0$ is obtained as in

$$\Delta_0 = \text{Clip3}\left(-t_C, t_C, \delta\right), \tag{7.9}$$

where t_C is a clipping parameter dependent on the QP, and Clip3(a, b, x) function clips the variable x to the range (a, b), i.e.

$$\text{Clip3}\,(a, b, x) = \text{Max}\,(a, \text{Min}\,(b, x))\,, \tag{7.10}$$

and δ is determined as

$$\delta = (9 * (q_0 - p_0) - 3 * (q_1 - p_1) + 8) >> 4. \tag{7.11}$$

Neglecting the clipping operation, the impulse response of the filter is (3, 7, 9, −3)/16. The value of δ is proportional to the deviation of the signal at the sides of the block boundary from a ramp and is equal to zero when the signal across the boundary has the form of a perfect ramp across the block boundary [35].

Deblocking filtering is only applied to a line of samples across the block boundary if the absolute value of δ is below t_C, i.e.

$$|\delta| < 10\, t_C\,. \tag{7.12}$$

Expression (7.12) evaluates whether the discontinuity at the block boundary is likely to be a natural edge or caused by a block artifact.

If two samples are modified in block P, i.e. condition in (7.5) is true, the sample p_1 is modified as

$$p_1{}' = p_1 + \Delta p_1, \tag{7.13}$$

and if condition in (7.6) is true, sample q_1 is modified as

$$q_1{}' = q_1 + \Delta q_1, \tag{7.14}$$

where the $p_1{}'$ and $q_1{}'$ are new values of samples p_1 and q_1 respectively, and the values of $\triangle p_1$ and $\triangle q_1$ are obtained as follows:

$$\Delta p_1 = \text{Clip3}\,(-t_C/2, t_C/2, ((((p_2 + p_0 + 1) >> 1) - p_1 + \Delta_0) >> 1))\,, \tag{7.15}$$

$$\Delta q_1 = \text{Clip3}\,(-t_C/2, t_C/2, ((((q_2 + q_0 + 1) >> 1) - q_1 - \Delta_0) >> 1))\,. \tag{7.16}$$

The impulse response of the filter is (8, 19, –1, 9, –3)/32 if the clipping operation is neglected. One can see that the value of the offset obtained in (7.9) is used in calculation of $\triangle p_1$ and $\triangle q_1$. The filtering operations at positions p_0, p_1, q_0, and q_1 do not modify the signal that has a form of a perfect ramp across the block boundary.

The deblocking filter decisions done for each line of a four-sample segment of a block boundary are summarized in a flowchart in Fig. 7.8.

An example of modifications to the block boundary samples in the normal filtering mode is shown in Fig. 7.9.

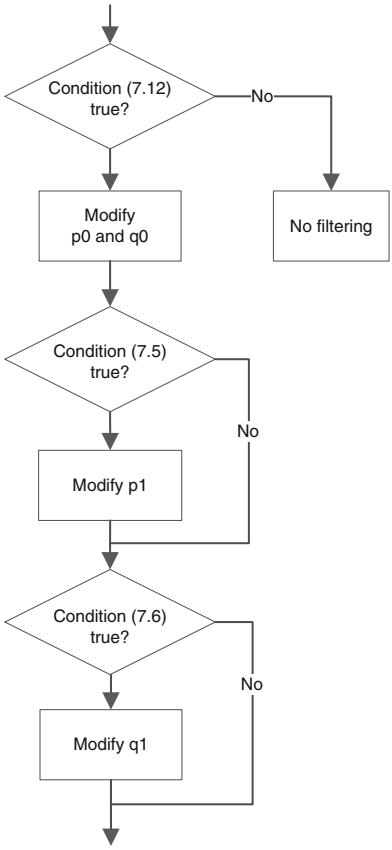

Fig. 7.8 Decisions for normal filter that are applied to each line of four-sample segment of block boundary

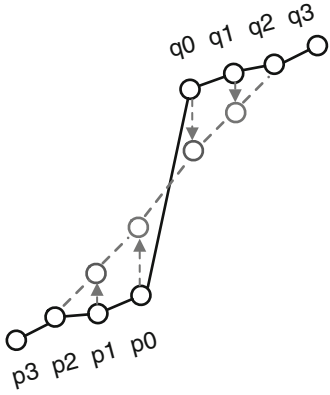

Fig. 7.9 Illustration of normal filtering mode operations: original block boundary (*solid black line*) and modified block boundary (*dashed gray line*)

7.2.2.3 Strong Filtering Mode

The strong deblocking filter in HEVC is applied to smooth flat areas, where block artifacts are more visible. This filtering mode modifies three samples from the block boundary and enables strong low-pass filtering. The HEVC strong filter is similar to the strong filter used by the H.264/AVC video standard [21] except the clipping operation, which is not present in the H.264/AVC strong filter. The reason for the clipping operation in HEVC deblocking filter is that the strong filtering decision is made based on the sample values in only two of the lines in the block, corresponding to $i = 0$ and $i = 3$. The clipping operation limits the amount of filtering in order to make sure that there is no excessive filtering on the lines which were not evaluated in the filtering decisions [19]. The filtering for the samples in block P is performed using the following equations if the clipping operation is neglected:

$$p_0' = (p_2 + 2p_1 + 2p_0 + 2q_0 + q_1 + 4) >> 3, \tag{7.17}$$

$$p_1' = (p_2 + p_1 + p_0 + q_0 + 2) >> 2, \tag{7.18}$$

$$p_2' = (2p_3 + 3p_2 + p_1 + p_0 + q_0 + 4) >> 3, \tag{7.19}$$

where p_0', p_1', and p_2' are modified values of samples p_0, p_1, and p_2 respectively. The modified sample values are then clipped to the range $[p_i - 2t_C, p_i + 2t_C]$. The equations for modification of samples q_0, q_1, and q_2 can be obtained by replacing p with q in (7.17)–(7.19).

7.2.2.4 Chroma Deblocking

As mentioned in Sect. 7.2.2.1, the deblocking is only applied to the chroma block boundaries which have boundary strength *Bs* equal to 2, i.e. when one of the adjacent blocks is intra-predicted. The block boundary should also be a CU, TU or a prediction partition boundary and be aligned with the 8×8 chroma sample grid. No further evaluation on the signal is done for chroma block boundaries. Chroma deblocking only modifies one sample from each side of the block boundary. The following expression is used to obtain the modification offset for the chroma block boundary.

$$\Delta_c = \text{Clip3}\left(-t_C, t_C, (((q_0 - p_0) << 2) + p_1 - q_1 + 4) >> 3\right) \tag{7.20}$$

The value of Δ_c is used for modification of the chroma samples p_0 and q_0 similarly to luma samples as in (7.7) and (7.8).

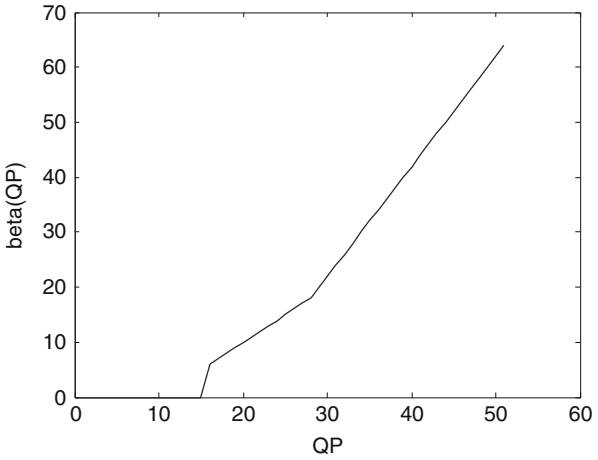

Fig. 7.10 Dependency of β on QP. Reproduced with permission from [37], © 2012 IEEE

7.2.2.5 Deblocking Adaption

Clipping operations are used in deblocking to avoid excessive filtering. The clipping parameter t_C is derived from a look-up table and depends on the average of the quantization parameter QP of the two adjacent blocks [44], which determines how coarse the quantization is. Variable t_C is obtained from a table as t_C (QP) when both adjacent blocks are inter-predicted ($Bs = 1$) and t_C (QP + 2) when at least one of the adjacent blocks is intra-predicted ($Bs = 2$). The deblocking parameter β, which mainly determines which block boundaries are modified by the deblocking filtering, also depends on the quantization parameter QP. One can see the dependency of the parameter β on the QP in Fig. 7.10 and parameter t_C on QP in Fig. 7.11. The higher the QP, the higher are the values of β and t_C and therefore the more often the deblocking filtering is applied, and more samples from the block boundary are modified. In addition, greater modifications to the sample values are allowed for higher QP. For low QP values, when a reconstructed picture has higher quality, the deblocking is essentially turned off by setting β and t_C to zero.

The deblocking strength can be further adjusted on a slice or picture level by sending parameters `tc_offset_div2` and `beta_offset_div2` in the slice header or picture parameter set (PPS) [46]. These parameters specify offsets (divided by two) that are added to a QP value before determining β and t_C from the look-up tables. This gives an encoder a possibility to adapt the deblocking strength depending on the sequence characteristics, the encoding mode, and other factors.

Changing the deblocking parameters on a picture level can be used to improve the subjective quality when using a hierarchical coding structure. An example of a hierarchical-B GOP8 coding structure, similar to the one used in the HM common test conditions random-access mode [6] is shown in Fig. 7.12. In this structure, the encoder can use QP cascading in order to improve the compression

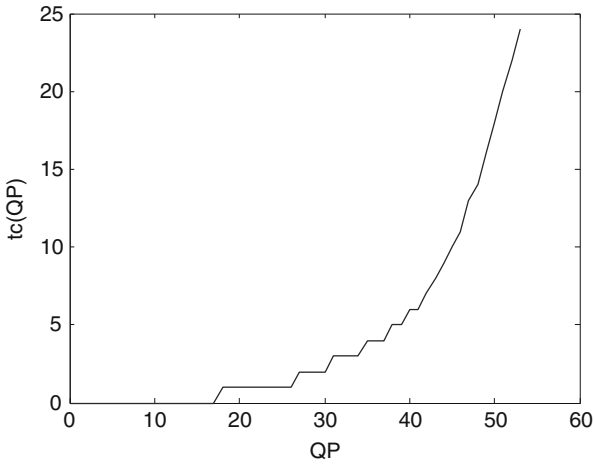

Fig. 7.11 Dependency of t_C on QP. Reproduced with permission from [37], © 2012 IEEE

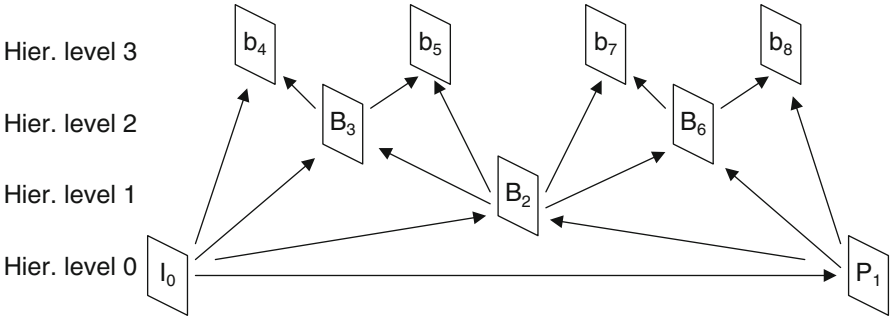

Fig. 7.12 Hierarchical-B coding structure with GOP8 illustrating different depth of the picture in the coding structure [38]

efficiency by applying the base QP to the intra-coded pictures, QP + 1 to the B-pictures at hierarchy level 0 (e.g. picture P_1), QP + 2 to the pictures at hierarchy level 1. Hierarchy level 2 uses QP + 3 and non-reference b-pictures at level 3 use QP + 4. The improvement in compression efficiency is achieved by coding with better quality the pictures at lower hierarchical levels, which are used for prediction of the pictures at higher hierarchy levels. However, in video sequences with chaotic motion, e.g. showing water, smoke, fire, rain, or snow, the QP cascading may cause block artifacts in the pictures at higher hierarchy levels, which is related to using lower bit budget for those pictures because of higher QP values and coarser mode decisions.

When the encoder uses hierarchical coding structure and QP cascading, the deblocking parameters `tc_offset_div2` and `beta_offset_div2` can be used to increase the deblocking strength for the pictures at higher hierarchy levels, which improves the subjective quality on sequences with chaotic motion, while

preserving the subjective quality on other types of video content [33, 38, 39]. An example of deblocking parameters improving the subjective quality on chaotic content in the hierarchical-B GOP8 coding structure is setting the `tc_offset_div2` to values 0, 3, 3, and 5 for pictures at hierarchy levels 0, 1, 2, and 3 respectively, while the `beta_offset_div2` is set to zero for all levels.

7.3 Sample Adaptive Offset (SAO)

7.3.1 Motivation and Overview of SAO

The key function of the sample adaptive offset is to attenuate ringing artifacts, which are more likely to appear when larger transform sizes are used. The SAO reduces sample distortion by first classifying the samples in the region into multiple categories with as selected classifier and adding a specific offset to each sample depending on its category. The classifier index and the offsets for each region are signaled in the bitstream [7, 8]. The SAO encoder is not standardized. It may minimize the average sample rate-distortion cost, as done in the HM reference software, or may use another criterion to generate SAO parameters. SAO can use different sample offsets in a region depending on the sample classification, and SAO parameters can change from one region of a picture to another. The HEVC uses two SAO types: *edge offset* (EO) and *band offset* (BO) [11]. In EO, the classification of a sample is based on its neighborhood, i.e. on the comparison between the current sample and its neighboring samples. In BO, the classification is based on the sample value.

The best coding efficiency could be achieved by a picture-based region partitioning method [11], which would, however, introduce additional encoding latency. In order to achieve low encoding latency and reduce the buffer requirement, the size of the region can be fixed and set as small as one coding tree unit (CTU). Multiple CTUs can share the same SAO parameters by region merging [14] to reduce side information. In HEVC, a CTU consists of three coding tree blocks (CTBs) of color components, and each color component can have its own SAO offsets and share the same EO/BO type for chroma components [10].

7.3.2 Edge Offset

Figure 7.13 shows the Gibbs phenomenon, which can be used to explain the appearance of ringing artifacts in image and video coding. The horizontal axis shows the sample position along a 1-D line and the vertical axis denotes the sample value. The dotted curve represents the original samples while the solid curve represents the reconstructed samples when the highest frequencies in the signal are discarded due to quantization of transform coefficients. Local peaks, convex edges/corners,

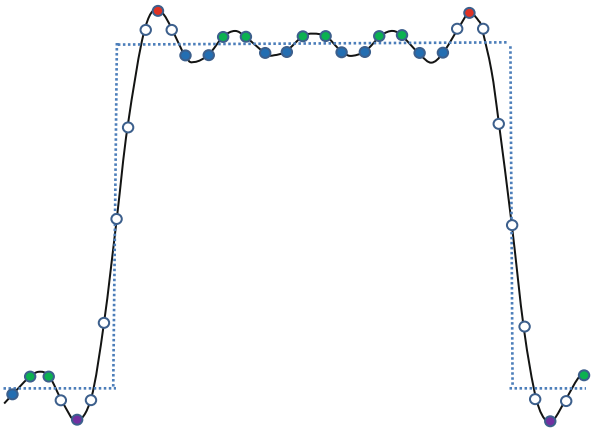

Fig. 7.13 Gibbs phenomenon, where the *dotted curve* is the original samples and the *solid curve* is the reconstructed samples. Local peaks, convex corners, concave corners, and local valleys are marked with *solid circles* and none-of-the-above samples with *empty circles*. Reproduced with permission from [13], © 2012 IEEE

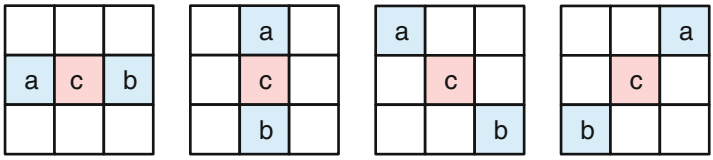

Fig. 7.14 Four 1-D directional patterns for EO sample classification: horizontal (EO class = 0), vertical (EO class = 1), 135° diagonal (EO class = 2), and 45° diagonal (EO class = 3). Reproduced with permission from [13], © 2012 IEEE

concave edges/corners, and local valleys are marked with solid circles and none-of-the-above samples with empty circles. One can observe from the figure that the distortion can be reduced by applying negative offsets to local peaks and convex corners and positive offsets to concave corners and valleys. The operation of the edge offset is based on this observation. EO uses four one-directional patterns for sample classification: horizontal, vertical, 135° diagonal, and 45° diagonal, as shown in Fig. 7.14, where label "c" represents the current sample and labels "a" and "b" represent the two neighboring samples. These four sample patterns form four EO classes. Only one EO class can be selected for each CTB that enables EO. Based on the rate-distortion optimization, one EO class is chosen and an index indicating which EO class is selected is signaled in the bitstream.

For a given EO class with the specific direction, each sample inside the CTB is classified into one of five categories. The current sample value, labeled as "c", is compared with its two neighbors along the selected 1-D pattern. The category classification rules for each sample are summarized in Table 7.2. Categories 1 and 4 are associated with a local valley and a local peak, respectively, along the selected 1-D pattern. Categories 2 and 3 are associated with concave and convex corners, respectively. If the current sample does not belong to any of EO categories 1–4, it is assigned to category 0 and SAO is not applied. Note that the categories are mutually exclusive and a sample can belong only to one category.

Table 7.2 Sample category classification rules for edge offset

Category	Condition
1	c < a && c < b
2	(c < a && c==b) || (c == a && c < b)
3	(c > a && c==b) || (c == a && c > b)
4	c > a && c > b
0	None of the above

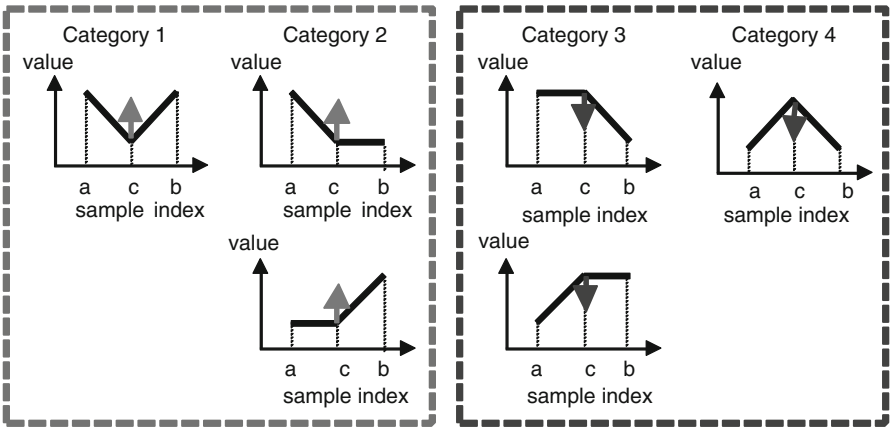

Fig. 7.15 Positive offsets for EO categories 1 and 2 and negative offsets for EO categories 3 and 4 results in smoothing, where the x-axis is sample index and the y-axis is sample value. Reproduced with permission from [13], © 2012 IEEE

The effect of the positive and negative edge offsets is illustrated in Fig. 7.15 and explained as follows. Positive offsets for categories 1 and 2 result in smoothing since local valleys and concave corners become smoother, while negative offsets for these categories result in sharpening. On the contrary, for categories 3 and 4, the negative offsets result in smoothing and positive offsets result in sharpening. In HEVC, sharpening in EO is not allowed. Therefore, the absolute values of four specific offsets are signaled by the encoder—one for each EO category, and the signs of the signaled offsets are implicitly derived from the corresponding EO categories [12, 23, 24]. Both EO and BO use four offsets, which limits the number of offsets to reduce the requirements for a line buffer (the line buffer is explained further in Sect. 7.4.3).

7.3.3 Band Offset

Another offset used by the HEVC SAO tool is band offset (BO). In band offset, one offset is added to all samples whose values belong to the same band (range of

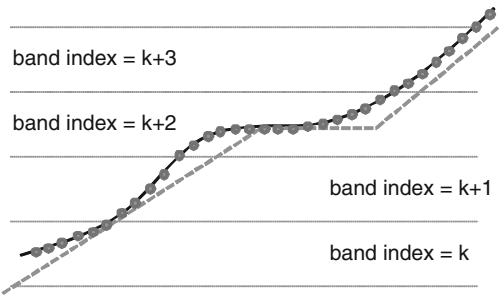

Fig. 7.16 Example of BO, where the *dotted curve* represents original samples and the *solid curve* denotes reconstructed samples. Reproduced with permission from [13], © 2012 IEEE

values). The sample values range is divided into 32 equal bands. For 8-bit samples in the range from 0 to 255, the width of a band is 8. Thus, sample values from $8k$ to $8k + 7$ belong to band k, where k ranges from 0 to 31. The difference between the original samples and reconstructed samples in a band (i.e. the offset of a band) can be signaled to the decoder. There is no constraint on the offset sign for the BO.

Figure 7.16 demonstrates how the BO compensates sample intensity offset of a region. The horizontal axis denotes the sample position and the vertical axis denotes the sample value. The dotted curve represents the original samples, while the solid curve denotes the reconstructed samples, affected by quantization errors of prediction residues and phase shifts because of the coded motion vectors that deviate from the true motion. As shown in Fig. 7.16, if there is a phase shift (difference) between the reconstructed motion vector and the "true" motion vector, a smooth region with a gradient may be offset with a certain value compared to the original signal. In this example, the reconstructed samples are shifted to the left compared to the original samples, which results in a systematic negative error that can be corrected by BO for bands k, $k + 1$, $k + 2$, and $k + 3$, where the samples ranging from $k * 8$ to $((k + 1) * 8) - 1$ are classified as belonging to band k, and can be modified by using the corresponding offset value.

In HEVC, only offsets of four consecutive bands and the starting (or minimum) band position of the current region are signaled to the decoder [25, 29]. Four offsets are signaled in the BO, which is equal to the number of signaled offsets in EO (the number of offsets is limited to reduce the line buffer requirement). The reason for signaling only four bands is that the range of sample values in a region formed by CTBs can be quite limited. Therefore, by signaling the starting band position of current region, BO can identify the minimum sample value to be compensated in the current region so that the decoder can recover it, as shown in the example in Fig. 7.17. This is especially true for chroma CTBs. In natural images, chroma components are often represented by a narrow-band signal, which means that by several band offsets, the encoder can recover most samples in the region.

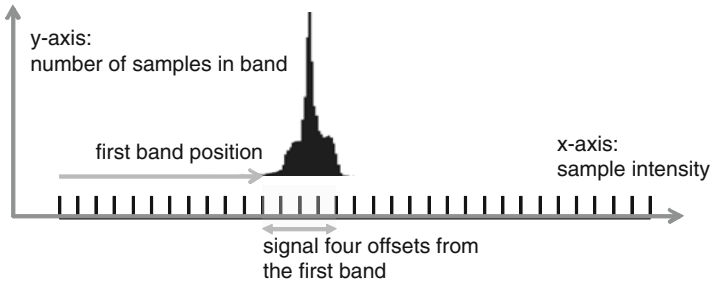

Fig. 7.17 Example of sample distribution in a CTB, where BO sends the offsets of four consecutive bands. Reproduced with permission from [13], © 2012 IEEE

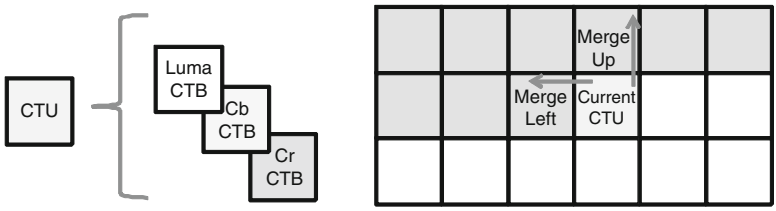

Fig. 7.18 CTU consists of three CTBs of color components; the current CTU can reuse SAO parameters of the left or above CTU. Reproduced with permission from [13], © 2012 IEEE

7.3.4 SAO Parameters Signaling

A syntax element `sample_adaptive_offset_enabled_flag` signaled in the Sequence Parameter Set (SPS) indicates whether SAO is enabled in the current video sequence. In the slice header, two syntax elements, `slice_sao_luma_flag` and `slice_sao_chroma_flag`, indicate if SAO is enabled for luma and chroma, respectively, in the current slice.

Low-delay applications can use the Coding Tree Unit (CTU) based SAO encoding algorithm. As shown in Fig. 7.18, a CTU comprises its corresponding luma CTB, Cb CTB, and Cr CTB. Syntax-wise, the basic unit for SAO parameters adaptation is always one CTU. If SAO is enabled in the current slice, the SAO parameters of each CTU are interleaved into the slice data. The SAO data in the bitstream are signaled in the beginning of each CTU. The CTU-level SAO parameters consist of SAO merging information, type information, and offset information.

7.3.4.1 SAO Parameters Merging

A CTU can use three options for signaling SAO parameters: reusing SAO parameters of the left CTU (by setting a syntax element `sao_merge_left_flag`

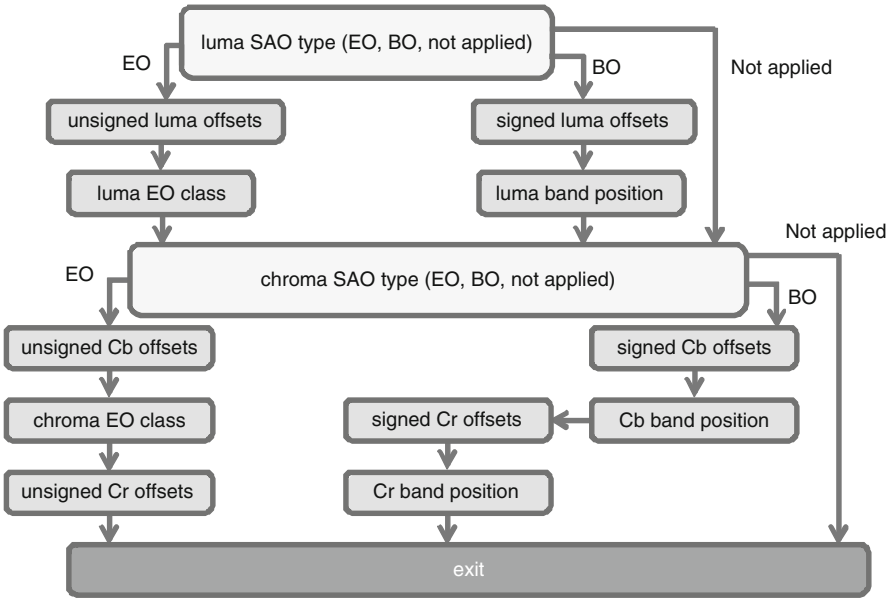

Fig. 7.19 Illustration of CTU-level SAO information coding when the current CTU is not merged with the CTU on the left or above. Reproduced with permission from [13], © 2012 IEEE

to 1), reusing SAO parameters of the above CTU (by setting a syntax element `sao_merge_up_flag` to 1), or by transmitting new SAO parameters. The SAO merging information is shared by all three color components. When SAO merge-left or SAO merge-up mode is indicated, all SAO parameters from the left or above CTU are copied and no more information is signaled for the current CTU. This CTU-based SAO information merging effectively reduces the SAO information that needs to be signaled [31].

7.3.4.2 SAO Type and Offsets Signaling

If merging of SAO information is not used, the information for the current CTU is signaled as shown in Fig. 7.19. Syntax elements for the luma component are first sent, followed by the Cb syntax elements and then the Cr syntax elements. For each color component, the SAO type is transmitted (`sao_type_idx_luma` or `sao_type_idx_chroma`), which indicates EO, BO, or not applied (SAO turned off). If BO or EO is selected, four offsets are transmitted. If BO is selected, the starting band position (`sao_band_position`) is signaled. Otherwise, if EO is selected, the EO class (`sao_eo_class_luma` or `sao_eo_class_chroma`) is signaled. The Cb and Cr components share the SAO type (`sao_type_idx_chroma`) and EO class (`sao_eo_class_chroma`)

syntax elements to reduce the side information and speed-up SAO processing by achieving more efficient memory access on certain platforms [4]. These syntax elements are therefore only coded for the Cb component. The design of the codewords (including "off", "EO class selection index", and "BO band position index") is based on the probability distribution to reduce side information.

7.3.4.3 CABAC Contexts and Bypass Coding

All CTU-level, SAO syntax elements including SAO merging information, SAO type information, and offset information are coded with context-based adaptive binary arithmetic coding (CABAC). Only the first bin of the SAO type, which specifies whether SAO is turned on or off in the current CTU, and the SAO merge-left and merge-up flags use CABAC contexts. All other bins are coded in the bypass mode, which significantly increases the SAO parsing throughput in CABAC without much coding efficiency loss [1, 3, 15, 30, 45].

7.4 Implementation and Parallelization Aspects

7.4.1 Deblocking Filter Complexity and Parallelism

When designing the HEVC deblocking filter, a lot of attention was paid to complexity and parallelization aspects. In H.264/AVC video decoders, deblocking takes a significant part of the computational complexity [27, 28]. Moreover, in H.264/AVC, the deblocking operations at one block boundary may affect the samples used in deblocking of the next block boundary, which complicates parallel processing.

7.4.1.1 HEVC Deblocking Filter Complexity

The complexity of the HEVC deblocking filter has been significantly decreased compared to the H.264/AVC deblocking. First of all, the deblocking is only applied to the block boundaries on the 8×8 luma sample grid. This already decreases the worst-case complexity of the deblocking compared to H.264/AVC, where the deblocking is applied on the 4×4 sample grid. The average complexity of the deblocking operation is also decreased compared to H.264/AVC since the prediction and transform blocks in HEVC are on average larger than those in H.264/AVC, where the maximum size of prediction blocks is 16×16 luma samples and the size of the transform blocks is 8×8 luma samples (if the maximum transform size in HEVC is not restricted to the same limits resulting in the worst-case scenario).

The filtering decisions constitute a significant part of the deblocking filter complexity. In order to reduce the complexity of deblocking decisions, the HEVC

deblocking uses decisions for a four-sample segment of the block boundary based on two lines crossing the block boundary. In contrast, the decisions in H.264/AVC deblocking are done for every line. The complexity of chroma deblocking filtering in HEVC has also been reduced compared to H.264/AVC since only chroma block boundaries with *Bs* equal to 2 are filtered. Therefore, only block boundaries adjacent to the intra-predicted blocks are filtered in the chroma components in contrast to H.264/AVC, where the chroma deblocking is also applied to the block boundaries between the inter-predicted blocks.

7.4.1.2 Deblocking Filter Parallelization Aspects

HEVC deblocking filter allows easy parallelization on several levels. First, parallelization is possible on the color component level. In HEVC, filtering decisions for chroma components are only based on the block boundary strength. Therefore, the only data to be shared between the luma and the chroma deblocking is the *Bs*, which depends on the prediction type [43]. This makes it possible to process chroma components independently of the luma component unlike in H.264/AVC, where chroma deblocking uses the decisions made for luma deblocking.

The vertical and horizontal block boundaries in HEVC are processed in a different order than in H.264/AVC. In HEVC, all the vertical block boundaries in the picture are filtered first, and then all the horizontal block boundaries are filtered [20, 37]. Since the minimum distance between two parallel block boundaries in HEVC is eight samples, and HEVC deblocking modifies at most three samples from the block boundary and uses four samples from the block boundary for deblocking decisions, filtering of one vertical boundary does not affect filtering of any other vertical boundary. This means there are no deblocking dependencies across the block boundaries. In principle, any vertical block boundary can be processed in parallel to any other vertical boundary. The same holds for the horizontal boundaries, although the modified samples from filtering the vertical boundaries are used as the input to filtering the horizontal boundaries.

The deblocking in HEVC can also be performed on the 8×8 block basis [32, 37, 47]. Figure 7.20 illustrates how the deblocking (both for vertical and horizontal boundaries) can be performed independently for each 8×8 block of samples. The deblocking is performed on the 8×8 luma sample grid and decisions are done separately for each four-sample segment of the block boundary, which means that two parts of the eight-sample block boundary are deblocked independently of each other [34]. Therefore, the deblocking of the 8×8 sample square with the crossing of vertical and horizontal lines on the 8×8 sample filtering grid in the middle of the block is not dependent on the deblocking in the other parts of the picture. Basically, the whole picture can be split into such 8×8 sample blocks (4×8 and 8×4 blocks at the picture boundaries), which can all be processed independently of other blocks. Since all vertical block boundaries in HEVC are processed before the horizontal

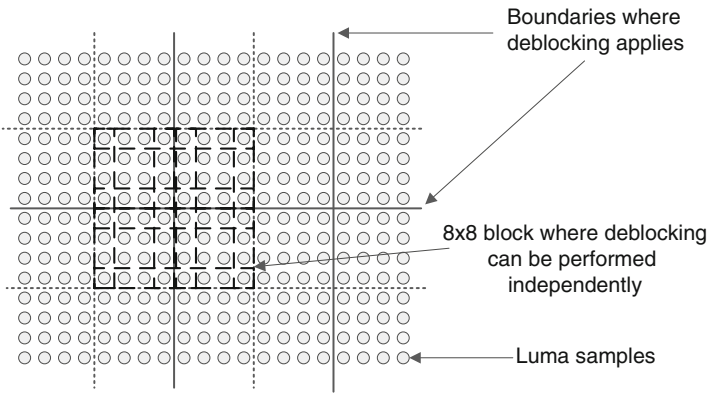

Fig. 7.20 Illustration of picture samples, horizontal and vertical block boundaries on the 8×8 grid, and those non-overlapping blocks of 8×8 samples (marked with *dotted lines*), which can be deblocked in parallel. The *dashed lines* mark samples used in deblocking decisions (vertical and horizontal)

block boundaries, the order of deblocking in each of these 8×8 deblocking units is the same: the vertical block boundary is filtered first, which is followed by the horizontal block boundary.

Since the HEVC deblocking can be easily parallelized, it can be done on a slice or tile [16] basis. In this case, an encoder or decoder can choose the option to first apply deblocking to the inner areas of a tile or slice, while leaving the deblocking on the tile or slice boundaries. When the decoding and deblocking of all tiles or slices is finished, the tile or slice boundaries can be processed as the last step.

Since the deblocking in HEVC is less computationally expensive and more parallelizable than the H.264/AVC deblocking, it can be said that the deblocking in HEVC has a better trade-off between the computational complexity, throughput, subjective and objective quality improvements than the H.264/AVC deblocking and is less of a bottleneck when implementing a decoder.

7.4.2 SAO Implementation Aspects and Parameters Estimation

Since SAO requires sample level operations to classify each sample into bands or categories in both encoder and decoder, the number of operations for each sample needs be reduced as much as possible to reduce the overall computational complexity. At encoder-side, there are many SAO types to be tested to achieve a better rate-distortion performance at reasonable computational complexity. Therefore, some efficient encoder algorithms are discussed in the following sections.

7.4.2.1 Fast Edge Offset Sample Classification

Although the sample classification rules of EO in Table 7.2 seem non-trivial, the EO sample classification can be implemented in a more efficient way by using the following function and equations:

$$sign3(x) = (x > 0)\ ?\ (+1) : (x == 0)\ ?\ (0) : -1, \tag{7.21}$$

$$edgeIdx = 2 + sign3\,(c - a) + sign3\,(c - b)\,, \tag{7.22}$$

$$edgeIdx2category\,[\,] = \{1, 2, 0, 3, 4\}\,, \tag{7.23}$$

$$category = edgeIdx2category\,[edgeIdx] \tag{7.24}$$

where, "c" is the current sample, and "a" and "b" are the two neighboring samples, as shown in Fig. 7.14 and Table 7.2. As a further speed-up, the data obtained in a previous step can be reused in the classification of the next sample. For example, assume that the EO class is 0 (i.e., a 1-D horizontal pattern) and the samples in the CTB are processed in the raster scan order. The "$sign3(c-a)$" of the current sample is equal to "$-sign3(c-b)$" of the neighboring sample to the left. Likewise, the "$sign3(c-b)$" of the current sample can be reused by the neighboring sample to the right. In software implementations, the $sign3(x)$ function can be implemented by using a bitwise operation or a look-up table to avoid using if-else operation, which can be time-consuming on certain platforms.

7.4.2.2 Fast Band Offset Sample Classification

The sample range is equally divided into 32 bands in BO. Since 32 is equal to two to the power of five, the BO sample classification can be implemented as using the five most significant bits of each sample as the classification result. In this way, the complexity of BO decreases, especially in hardware that only needs wire connections without logic gates to obtain the classification result from the sample value. To reduce the software decoding run time, the BO classification can be implemented by using bitwise operation or a look-up table to avoid using if-else operations.

7.4.2.3 Distortion Estimation for Encoder

The rate-distortion optimization process [41] requires multiple calculation of the distortion between the original and reconstructed sample values. A straightforward SAO implementation would add offsets to the samples modified by deblocking and then calculate the distortion between the resulting and the original samples. To reduce the memory access and the number of operations, a fast distortion estimation method [9] can be implemented as follows. Let k, $s(k)$, and $x(k)$ be

sample positions, original samples, and the reconstructed samples, respectively, where k belongs to C, the set of samples inside the CTB that belong to a specific SAO type (i.e., BO or EO), a starting band position or EO class, and a specific band or category. The distortion between original samples and reconstructed samples can be described by the following equation:

$$D_{pre} = \sum_{k \in C} (s(k) - x(k))^2 \tag{7.25}$$

The distortion between the original samples and samples modified by SAO can be described by the following equation

$$D_{post} = \sum_{k \in C} (s(k) - (x(k) + h))^2 \tag{7.26}$$

where h is the offset for the sample set. The distortion change is defined by the following equation:

$$\Delta D = D_{post} - D_{pre} = \sum_{k \in C} \left(h^2 - 2h\,(s(k) - x(k))\right) = Nh^2 - 2hE \tag{7.27}$$

where N is the number of samples in the set, and E is the sum of differences between the original samples and the reconstructed samples (before SAO) as defined by the following equation:

$$E = \sum_{k \in C} (s(k) - x(k)) \tag{7.28}$$

Please note that the sample classification and (7.28) can be calculated right after the input samples become available after the deblocking filtering. Thus, N and E can be calculated only once and stored. Then, the delta rate-distortion cost is defined as follows:

$$\Delta J = \Delta D + \lambda R \tag{7.29}$$

where λ is the Lagrange multiplier, and R represents the estimated bits of side information.

For a given CTB with a specific SAO type (i.e., BO or EO), starting band position or EO class, and a specific band or category, several h values (offsets) close to E/N are tested, and the offset that minimizes ΔJ is chosen. After offsets for all bands or categories have been chosen, the ΔJ for each of the 32 bands of BO or each of the five categories of EO are added to obtain the delta (change) of the rate-distortion cost of the entire CTB. The distortion of the BO bands using zero offsets and the EO category 0 can be pre-calculated by (7.25) and stored for subsequent re-use. When SAO decreases the cost for the entire CTB (i.e. the delta cost is negative), SAO is enabled for this CTB. Similarly, the best SAO type and the best starting position or EO class can be found by the fast distortion estimation.

7.4.2.4 Slice-Level On/Off Control

The HM reference software common test conditions [6] use hierarchical quantization parameter (QP) settings. As an example, in the random access condition, the GOP size is eight. Picture can belong to different hierarchy levels depending on their positions in the GOP (see Fig. 7.12). Normally, the picture is only predicted from the pictures with a smaller or the same hierarchy level. A picture with at higher hierarchy level will likely be given a higher QP.

A slice-level on/off decision algorithm [2, 26] is provided as follows. For hierarchy level 0 pictures, SAO is always enabled in the slice header. Given a current picture with a nonzero hierarchy level N, the previous picture is defined as the last picture with hierarchy level $(N-1)$ in the decoding order. If SAO is disabled in more than 75 % of the CTBs in the previous picture, the HM reference encoder will disable SAO in all slice headers of the current picture and skip the SAO encoding process. This encoder technique not only can decrease the number of syntax to be parsed, but also provides 0.5 % BD-rate improvement [2, 26]. Please note that luma and chroma SAO can be enabled or disabled independently in the slice header.

7.4.2.5 SAO Parameters Estimation and Interaction with Deblocking

In the HM reference encoder, SAO parameters are estimated for each CTU. Since SAO is applied to the output of the deblocking filter, the SAO parameters cannot be precisely determined until the deblocked samples are available. However, the deblocked samples of the right columns and the bottom rows in the current coded-tree block (CTB) may be unavailable because the CTU to the right and the CTU below the current CTU may not have been reconstructed yet (in a one-pass encoder). This constraint can be overcome with one of the two options. The first option [18] estimates the SAO parameters on the available CTB samples, i.e. on the CTB samples except the three bottom rows of luma samples, one bottom row of Cb and Cr samples, the rightmost four columns of luma samples, the rightmost two columns of Cb and Cr samples. The proposed approach does not incur a noticeable coding efficiency loss when the 64×64 CTU size is used. However, for smaller CTU sizes, the percentage of samples not used in the SAO parameter estimation is higher, which may cause significant coding efficiency loss. In this case, the second option [22] uses samples before the deblocking instead of unavailable deblocked samples during SAO parameter estimation, which can reduce the loss in coding efficiency for smaller CTU sizes.

7.4.3 CTU-Based Processing and Line Buffer

CTU-based processing is commonly adopted in practical implementations. CTUs are encoded or decoded one by one in a raster scan order and the in-loop filtering is applied right after the encoding/decoding of a CTU. The deblocking filtering of the

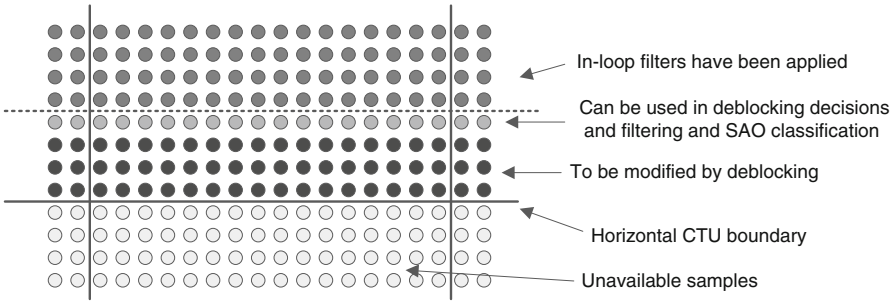

Fig. 7.21 Example of luma samples line buffer. Samples below the *dashed line* and above the horizontal CTU boundary should be kept in the line buffer until the unavailable samples have been reconstructed

bottom horizontal CTU boundary needs samples from the CTU below. Hence, after the current CTU has been processed, the deblocking cannot be applied yet to the bottom rows of samples since the reconstructed samples of the CTB to the bottom of the current CTB are not yet available (see Fig. 7.21). Likewise, since SAO is applied after the deblocking filter, SAO cannot be applied to the bottom sample rows before the deblocking is done. In order to decrease memory bandwidth requirements, the information needed for in-loop filtering over the lower CTU boundary is kept in the fast on-chip memory until the CTU below has been reconstructed and the in-loop filters applied. This on-chip memory is usually called a "line buffer" since the information for horizontal lines of samples typically needs to be kept.

In the deblocking filter, vertical filtering across a horizontal CTU boundary needs four rows of luma samples, two rows of Cb samples, and two rows of Cr samples from the upper CTU to be kept in the line buffer for the filtering decisions and operations. Deblocking can modify up to three rows of luma samples, one row of Cb samples, and one row of Cr samples from the block boundary.

Let us assume that the deblocking filter needs to keep N rows of the above CTBs, where N is equal to four for luma and two for Cb and Cr. Since the N-th row above the horizontal CTB boundary will not to be modified by the vertical deblocking filtering, the SAO can be applied to the $(N+1)$-th row above the horizontal CTB boundary. However, the bottom N rows of the current CTB should be kept in the line buffer before the CTB below is reconstructed and deblocking and SAO are applied (see Fig. 7.21 for luma samples example).

When the CTB below is reconstructed and SAO in the CTB above uses EO with the class greater than zero, applying SAO for the N-th row above the boundary needs the $(N+1)$-th row above the boundary to be available. A straightforward solution is to store reconstructed and deblocked samples of the $(N+1)$-th row in the line buffer. However, using the fast EO sample classification, the "*sign3*" results [the sign of the difference or 0 value between the N-th row and the $(N+1)$-th row] can be stored instead, which reduces the memory requirements to two bits per sample.

In addition to the described four lines on luma samples, two lines of Cb, two lines of Cr samples, three lines of two-bit "*sign3*" values, and some CTU- and PU-level information need to be stored. The deblocking needs information about

Table 7.3 Size of the elements of a 16 × 16 CTU needed to be kept in a line buffer

Values in line buffer	Required bits
Four motion vectors*	128 bits
4 reference picture indices*	16 bits
8 × 8 / 4 × 8 partitions flags*	2 bits
B-/P-prediction flags in 8 × 8 partitions*	2 bits
sign3	64 bits
Luma SAO type	2 bits
Chroma SAO type	2 bits
Starting band positions or EO classes	15 bits
Luma and chroma offsets	48 bits
Total elements	**279 bits**
64 luma samples	512 bits
32 chroma samples	256 bits
Total samples	**768 bits**
Total bits for CTU	**1047 bits**

*This information is also used in motion vector derivation

the motion vectors adjacent to the lower CTU boundary, and SAO needs SAO type and SAO offsets of the upper CTB row. For an 8-bit 4:2:0 video and the smallest CTU size (16 × 16 luma and 8 × 8 chroma samples), the information stored per CTU is 279 bits (this number may depend on the implementation). Among these 279 bits, 128 bits are for four motion vectors (4 × 8 and 8 × 4 partitions can only use one motion vector), 16 bits are for four indices of reference pictures in the decoded picture buffer, two bits are for signaling whether 4 × 8 or 8 × 8 partitions are used, and two bits are for signaling if one or two MVs are used in 8 × 8 partitions. Please note that this information is also needed for other modules, such as motion vector derivation. Alternatively, 8 bits with four Bs values may be stored if the information about motion vectors in the next CTU row is available. Then, 64 bits are for "*sign3*" values of 16 luma and 16 chroma samples, two bits are for luma SAO type, two bits are for chroma SAO type, 15 bits are from starting band positions or EO classes, and 48 bits are from luma and chroma offsets. The sample line buffers for the 16 × 16 CTU keep 64 luma samples and 32 chroma samples, which requires 768 bits (see Table 7.3 for details). Therefore, the total line buffer size is about 15K bytes for full HD (i.e. 1920 × 1080) videos.

The line buffer size would be somewhat smaller when larger CTU sizes are used since the same parameters apply to a larger number of samples. The size of the line buffer can be further reduced by using vertical tiles, hence splitting the horizontal "span" of CTUs to be processed before the bottom CTU is encoded/decoded.

7.4.4 Error Resilience

In order to provide additional error resilience, HEVC allows an encoder to disable in-loop filters over tile and slice boundaries. A flag `slice_loop_`

`filter_across_slices_enabled_flag` equal to 1 indicates that the in-loop filters are applied across the left and upper boundaries of the current slice. When the flag is equal to 0, in-loop filtering operations are not applied across the left and upper boundaries of the current slice. When the flag is not present, it is inferred to be equal to `pps_loop_filter_across_slices_enabled_flag`.

A picture parameters set (PPS) flag `loop_filter_across_tiles_enabled_flag` equal to 1 specifies that in-loop filters are applied across tile boundaries. When the flag is equal to 0, it specifies that in-loop filtering operations are not performed across tile boundaries. When the flag is not present, the value of `loop_filter_across_tiles_enabled_flag` is inferred to be equal to 1.

These flags provide additional error resilience since an error created because of a slice or tile loss will not propagate into neighboring slices or tiles of the same picture (however, it may propagate to other slices or tiles of subsequent pictures because of inter-picture prediction).

7.5 Coding Efficiency and Subjective Quality Improvements

The HEVC in-loop filters improve both the objective and subjective quality. The objective quality improvement is achieved due to increasing the quality of the reconstructed pictures. There is also an additional effect due to better quality of reference pictures, which improves motion prediction and therefore the coding efficiency.

In this section, the compression efficiency improvements have been evaluated on the JCT-VC video sequences test set. The results are provided for several classes of sequences and under different coding conditions defined in the HEVC common test conditions document [6]. These configurations are: All Intra where only intra-prediction is used; Random Access which uses intra pictures over certain time intervals and hierarchical-B coding structure; and two low-delay configurations, which have only one intra-picture, and where motion-compensated prediction uses only temporally preceding pictures. The Low Delay P (LP) configuration does not use bi-directional motion-compensated prediction. The BD-rate is used in the HEVC standardization as a measure for the average bit rate reduction at the same mean squared error [5]. The HEVC reference software HM11.0 was used in all experiments. The reported decoding time has been evaluated by decoding the bitstreams on a Windows 7 (64bit) PC with i7-920 CPU and 8GB of RAM without writing the reconstructed pictures to the disk.

7.5.1 Deblocking Coding Efficiency and Subjective Quality Improvements

The results for objective performance of the deblocking filter are provided in Table 7.4. The results show that applying HEVC deblocking leads to the 1.3–3.2 %

Table 7.4 Luma BD-rates evaluating objective effects of using deblocking filtering under various coding conditions

Anchor: disable deblocking Test: enable deblocking		Y BD-rate			
		All Intra (AI)	Random Access (RA)	Low Delay B (LB)	Low Delay P (LP)
Class A	Traffic	−2.3 %	−2.7 %	n/a	n/a
Cropped 4K × 2K	PeopleOnStreet	−2.0 %	−4.4 %	n/a	n/a
	Nebuta	−1.1 %	−1.1 %	n/a	n/a
	SteamLocomotive	−2.3 %	−5.4 %	n/a	n/a
Class B	Kimono	−4.1 %	−5.7 %	−5.9 %	−8.0 %
1080p	ParkScene	−1.4 %	−1.9 %	−2.0 %	−2.8 %
	Cactus	−1.3 %	−3.4 %	−3.7 %	−4.5 %
	BasketballDrive	−1.8 %	−3.9 %	−3.7 %	−4.9 %
	BQTerrace	−0.3 %	−1.1 %	−0.6 %	−2.3 %
Class C	BasketballDrill	−0.7 %	−2.1 %	−2.1 %	−2.8 %
WVGA	BQMall	−1.5 %	−2.5 %	−2.6 %	−3.1 %
	PartyScene	−0.4 %	−0.8 %	−0.8 %	−0.9 %
	RaceHorses	−1.2 %	−2.6 %	−2.9 %	−3.2 %
Class D	BasketballPass	−1.4 %	−2.5 %	−2.5 %	−2.9 %
WQVGA	BQSquare	−0.1 %	0.0 %	0.6 %	0.3 %
	BlowingBubbles	−0.4 %	−0.9 %	−0.8 %	−1.1 %
	RaceHorses	−0.9 %	−2.3 %	−2.4 %	−2.7 %
Class E	FourPeople	−2.3 %	n/a	−4.9 %	−6.6 %
720p	Johnny	−2.2 %	n/a	−2.2 %	−5.8 %
	KristenAndSara	−2.1 %	n/a	−4.0 %	−6.0 %
Class F	BasketballDrillText	−0.6 %	−2.0 %	−1.9 %	−2.5 %
	ChinaSpeed	−0.7 %	−1.8 %	−1.9 %	−2.3 %
	SlideEditing	−0.3 %	−0.3 %	−0.5 %	−0.7 %
	SlideShow	−0.8 %	−0.9 %	−0.8 %	−1.1 %
Class summary	Class A	−1.9 %	−3.4 %	n/a	n/a
	Class B	−1.8 %	−3.2 %	−3.2 %	−4.5 %
	Class C	−0.9 %	−2.0 %	−2.1 %	−2.5 %
	Class D	−0.7 %	−1.4 %	−1.3 %	−1.6 %
	Class E	−2.2 %	n/a	−3.7 %	−6.1 %
	Class F	−0.6 %	−1.2 %	−1.3 %	−1.6 %
Overall summary	All	−1.3 %	−2.3 %	−2.3 %	−3.2 %
	Decoding time (%)	107 %	104 %	104 %	105 %

bit rate decrease at the same quality depending on the configuration. For certain classes of sequences, 6 % decrease of bit rate is achieved.

Figure 7.22 compares a part of a decoded picture from the *BasketballDrive* sequence (1080p, 50 fps) in Random Access configuration at QP = 32 where the deblocking was applied with the configuration where the deblocking was turned off. Figure 7.23 shows same comparison for the sequence *KristenAndSara*

Fig. 7.22 Sequence *BasketballDrive*, Random Access, QP32: (**a**) deblocking turned off, (**b**) deblocking turned on

Fig. 7.23 Sequence *KristenAndSara*, Low Delay, QP37: (**a**) deblocking turned off, (**b**) deblocking turned on

(720p@60 fps) coded in Low Delay B configuration at QP = 37. It can be seen that the deblocking filter effectively attenuates block artifacts.

7.5.2 SAO Coding Efficiency and Subjective Quality Improvement

This section illustrates the subjective and objective performance of the SAO tool. Table 7.5 reports the sequence-wise luma BD-rates and the average luma BD-rates and run-times for different encoding structures and CTU sizes equal to 64×64 in luma using the skipping boundary samples algorithm as described in Sect. 7.4.2.5. For *BQTerrace* in the LP condition, the SAO coding gain reaches 18.9 %. It is noted that SAO is particularly effective for Class F sequences, which mostly contain computer graphics and screen content rather than natural video. One could also notice that SAO shows higher coding gains in the LP configuration without bi-directional prediction. Regarding the computational complexity, SAO increases the average decoding time by less than 2–3 %.

The subjective quality improvements due to reduction of ringing artifacts are shown in Figs. 7.24 and 7.25. Figure 7.24 shows an example of the coded computer-generated sequence SlideEditing. SAO significantly improves visual quality by suppressing ringing *artifacts* near objects edges. Figure 7.25 shows examples of natural video sequences *RaceHorses* and *BasketballPass* where the edges of objects are much cleaner when SAO is enabled. According to viewing tests, SAO improves subjective quality [42].

7.5.3 Combined Effect of In-Loop Filters on Coding Efficiency

Table 7.6 demonstrates objective compression efficiency improvements due to both in-loop filters compared to the configuration where both the deblocking filter and SAO are turned off. One can see that the compression efficiency improvements are 2.6–15 % depending on coding configurations. The decoding time increase is about 10 % and depends on the coding conditions. The encoding complexity mostly depends on a particular encoder implementation and is not significant in the HM11.0 encoder operating in common test conditions (on the order of 1 % encoding time increase [13]). These numbers indicate that the in-loop filters are an efficient tool in improving the HEVC compression efficiency.

7.6 Main Differences between HEVC and H.264/AVC In-Loop Filters

This section summarizes key differences between the HEVC and H.264/AVC in-loop filters. There is only a deblocking in-loop filter in H.264/AVC, while the HEVC standard defines two in-loop filters: the deblocking filter and the sample adaptive offset, SAO.

Table 7.5 Luma BD-rates evaluating objective effects of using SAO under various coding conditions

Anchor: disabling SAO Test: enabling SAO CTU Size in luma: 64 × 64 CTU boundary: option 1		Y BD-rate			
		All Intra (AI)	Random Access (RA)	Low Delay B (LB)	Low Delay P (LP)
Class A	Traffic	−0.9 %	−1.2 %	n/a	n/a
Cropped 4K × 2K	PeopleOnStreet	−1.3 %	−2.1 %	n/a	n/a
	Nebuta	−0.1 %	−2.3 %	n/a	n/a
	SteamLocomotive	−0.2 %	−2.6 %	n/a	n/a
Class B	Kimono	−0.5 %	−0.6 %	−0.8 %	−7.8 %
1080p	ParkScene	−0.7 %	−0.8 %	−1.3 %	−9.1 %
	Cactus	−0.4 %	−2.4 %	−2.9 %	−10.4 %
	BasketballDrive	−0.2 %	−1.5 %	−1.5 %	−9.0 %
	BQTerrace	−0.5 %	−4.8 %	−3.8 %	−18.9 %
Class C	BasketballDrill	−1.0 %	−1.7 %	−2.7 %	−6.4 %
WVGA	BQMall	−0.3 %	−0.9 %	−1.7 %	−8.2 %
	PartyScene	−0.1 %	0.2 %	−0.7 %	−4.0 %
	RaceHorses	−0.5 %	−1.9 %	−1.8 %	−9.7 %
Class D	BasketballPass	−0.2 %	−0.6 %	−1.3 %	−4.7 %
WQVGA	BQSquare	−0.5 %	−0.0 %	−0.7 %	−3.9 %
	BlowingBubbles	−0.2 %	0.4 %	0.0 %	−2.9 %
	RaceHorses	−0.5 %	−1.2 %	−1.3 %	−6.4 %
Class E	FourPeople	−0.7 %	n/a	−2.9 %	−9.1 %
720p	Johnny	−0.4 %	n/a	−1.9 %	−13.3 %
	KristenAndSara	−0.6 %	n/a	−2.3 %	−11.4 %
Class F	BasketballDrillText	−1.1 %	−2.0 %	−3.9 %	−6.7 %
	ChinaSpeed	−1.3 %	−3.6 %	−6.9 %	−9.3 %
	SlideEditing	−1.5 %	−3.0 %	−4.5 %	−5.0 %
	SlideShow	−1.9 %	−2.2 %	−5.4 %	−6.4 %
Class summary	Class A	−0.6 %	−2.0 %	n/a	n/a
	Class B	−0.4 %	−2.0 %	−2.0 %	−11.1 %
	Class C	−0.5 %	−1.1 %	−1.7 %	−7.1 %
	Class D	−0.4 %	−0.3 %	−0.8 %	−4.5 %
	Class E	−0.6 %	n/a	−2.6 %	−11.3 %
	Class F	−1.5 %	−2.7 %	−5.2 %	−6.8 %
Overall summary	All	−0.7 %	−1.7 %	−2.4 %	−9.2 %
	Decoding time (%)	103 %	102 %	102 %	103 %

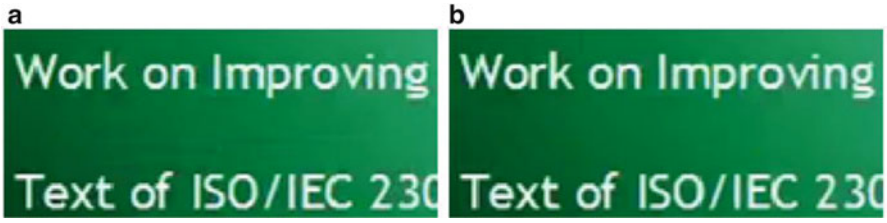

Fig. 7.24 Example of test sequence *SliceEditing* in LP condition, POC = 100, QP = 32: (**a**) SAO is disabled, (**b**) SAO is enabled

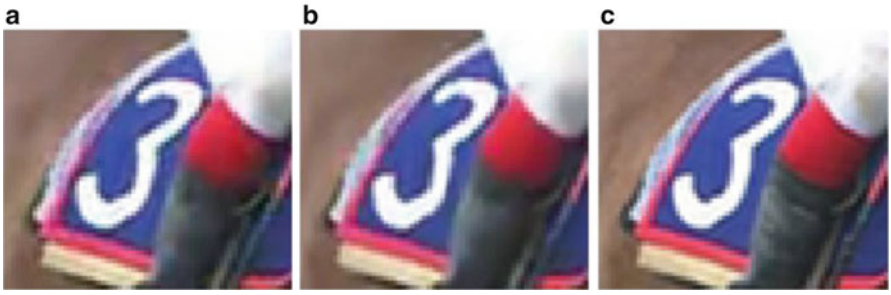

Fig. 7.25 Subjective quality comparison of *RaceHorses* test sequence, POC = 20, QP = 32, LP condition: (**a**) SAO is disabled, (**b**) SAO is enabled, (**c**) original (uncoded) sequence

The computational complexity of the HEVC deblocking is lower than that of the H.264/AVC. Reduction of the HEVC deblocking complexity is achieved by restricting the filtering to the 8×8 sample grid in contrast to the 4×4 sample grid in the H.264/AVC deblocking. Additional complexity reduction in HEVC is achieved by making the sample-based filtering decisions based on a subset of lines across the block boundary in contrast to the line-based decisions in H.264/AVC deblocking. Moreover, the HEVC deblocking of chroma components is only applied to the intra-predicted block boundaries. The HEVC deblocking is also more suitable for parallel implementation than the H.264/AVC deblocking since each 8×8 sample block in HEVC can be deblocked independently of other 8×8 blocks and the order of the vertical and horizontal filtering operations in HEVC deblocking is always the same. The processing order for the horizontal and vertical block boundaries is therefore different in HEVC and H.264/AVC. When applying the HEVC deblocking on the CU basis, right after the CU reconstruction, filtering of four right-most samples of horizontal block boundaries in a CU should be delayed until the next CU to the right is reconstructed and the vertical boundary between the CUs is filtered.

HEVC and H.264/AVC deblocking filters are also different in terms of criteria that evaluate the signal (reconstructed sample values) at the sides of a block boundary to decide whether the deblocking is applied to this block boundary. In H.264/AVC, the deblocking is typically applied to the block boundary when the

Table 7.6 Luma BD-rates evaluating the objective effects of using deblocking filter and SAO under various coding conditions

Anchor: disable deblocking and SAO Test: enable deblocking and SAO		Y BD-rate			
		All Intra (AI)	Random Access (RA)	Low Delay B (LB)	Low Delay P (LP)
Class A	Traffic	−4.4 %	−5.0 %	n/a	n/a
Cropped 4K × 2K	PeopleOnStreet	−4.4 %	−8.1 %	n/a	n/a
	Nebuta	−1.5 %	−4.9 %	n/a	n/a
	SteamLocomotive	−3.3 %	−9.5 %	n/a	n/a
Class B	Kimono	−6.1 %	−7.7 %	−8.1 %	−22.0 %
1080p	ParkScene	−2.9 %	−3.5 %	−4.2 %	−18.1 %
	Cactus	−2.3 %	−6.7 %	−7.2 %	−18.9 %
	BasketballDrive	−2.7 %	−6.5 %	−6.1 %	−18.2 %
	BQTerrace	−1.0 %	−6.7 %	−5.3 %	−25.7 %
Class C	BasketballDrill	−2.4 %	−4.5 %	−5.4 %	−11.9 %
WVGA	BQMall	−2.3 %	−4.0 %	−5.2 %	−15.2 %
	PartyScene	−0.7 %	−1.0 %	−2.0 %	−7.7 %
	RaceHorses	−2.4 %	−5.8 %	−5.9 %	−16.0 %
Class D	BasketballPass	−2.3 %	−3.7 %	−4.8 %	−10.4 %
WQVGA	BQSquare	−0.7 %	−0.1 %	−0.3 %	−5.2 %
	BlowingBubbles	−0.9 %	−0.7 %	−1.2 %	−6.3 %
	RaceHorses	−2.1 %	−4.4 %	−4.9 %	−12.0 %
Class E	FourPeople	−3.8 %	n/a	−8.4 %	−20.6 %
720p	Johnny	−3.4 %	n/a	−5.3 %	−28.2 %
	KristenAndSara	−3.4 %	n/a	−6.9 %	−23.9 %
Class F	BasketballDrillText	−2.3 %	−4.6 %	−6.2 %	−11.6 %
	ChinaSpeed	−2.3 %	−5.9 %	−9.6 %	−13.2 %
	SlideEditing	−1.8 %	−3.4 %	−5.5 %	−5.6 %
	SlideShow	−2.8 %	−3.3 %	−6.9 %	−8.4 %
Class summary	Class A	−3.4 %	−6.9 %	n/a	n/a
	Class B	−3.0 %	−6.2 %	−6.2 %	−20.6 %
	Class C	−2.0 %	−3.8 %	−4.6 %	−12.7 %
	Class D	−1.5 %	−2.2 %	−2.8 %	−8.5 %
	Class E	−3.5 %	n/a	−6.9 %	−24.2 %
	Class F	−2.3 %	−4.3 %	−7.1 %	−9.7 %
Overall summary	All	−2.6 %	−4.8 %	−5.5 %	−15.0 %
	Decoding time (%)	113 %	107 %	107 %	109 %

signal on both sides of the boundary is flat. Therefore in HEVC, the deblocking is also applied when the signal on each side of the block boundary approximates a ramp or a slope, which can happen in smooth areas with changing luma intensity.

The SAO in-loop filter attenuates ringing artifacts, which can be more pronounced in HEVC when larger transform sizes are used by the encoder. Moreover, SAO can also be applied to the inside samples of the large blocks, which cannot be corrected by the deblocking filter. This is especially important for HEVC because of the large transform sizes allowed in the standard.

In case of a CTU-based processing, four lines of samples for the luma component and two lines for the chroma components should be kept in a line buffer for in a line buffer both the HEVC and H.264/AVC deblocking. The fourth line of samples would also be modified by SAO in HEVC and is therefore delayed to be written to the memory compared to H.264/AVC. Some additional memory is required for keeping the SAO parameters (see Table 7.3).

7.7 Conclusions

HEVC defines two in-loop filters, deblocking and sample adaptive offset (SAO), which significantly improve the subjective quality of decoded video sequences as well as compression efficiency by increasing the quality of the reconstructed/reference pictures. The deblocking filter attenuates discontinuities on the block boundaries, while SAO mainly corrects ringing artifacts caused by large transforms and quantization and sample value offsets in certain regions of a picture caused by coding of motion vectors. The complexity of the HEVC deblocking has been significantly reduced compared to the H.264/AVC. Moreover, the HEVC deblocking filter is highly parallelizable with parallelization down to 8×8 sample blocks. Having lower computational complexity and being highly parallelizable, the HEVC deblocking is less of a bottleneck in the decoder implementation than the H.264/AVC deblocking and provides better trade-off between the computational complexity and coding efficiency (subjective and objective quality). SAO is a new in-loop filter, not present in H.264/AVC, which provides significant reduction of ringing artifacts at relatively low decoding complexity. The deblocking and SAO can also be implemented in the same processing unit, which simplifies CTU-based encoding and decoding and reduces cost in hardware implementations.

References

1. Alshina E, Alshin A, Park JH (2012) AHG5: on bypass coding for SAO syntax elements, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-J0043, Stockholm, July 2012
2. Alshina E, Alshin A, Park JH (2012) Encoder modification for SAO, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-J0044, Stockholm, July 2012
3. Alshina E, Alshin A, Park JH, Fu C-M, Huang Y-W, Lei S (2012) AHG5/AHG6: on reducing context models for SAO merge syntax, Joint Collaborative Team on Video Coding Coding (JCT-VC), Document JCTVC-J0041, Stockholm, July 2012
4. Alshina E, Alshin A, Park JH, Laroche G, Gisquet C, Onno P (2012) AHG6: on SAO type sharing between U and V components, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-J0045, Stockholm, July 2012
5. Bjøntegaard G (2001) Calculation of average PSNR differences between RD-curves, ITU-T SG16 Q6 Video Coding Experts Group (VCEG), Document VCEG-M33, Austin, Apr. 2001

6. Bossen F (2013) Common test conditions and software reference configurations, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-L1100, Geneva, Jan. 2013
7. Fu C-M, Chen C-Y, Huang Y-W, Lei S (2010) TE10 Subtest 3: Quadtree-based adaptive offset, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-C147, Guangzhou, Oct. 2010
8. Fu C-M, C-Y Chen, Huang Y-W, Lei S (2011) CE8 Subset 3: picture quadtree adaptive offset, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-D122, Daegu, Jan. 2011
9. Fu C-M, Chen C-Y, Huang Y-W, Lei S (2011) Sample adaptive offset for HEVC. In: IEEE 13th international workshop on multimedia signal processing (MMSP) 2011
10. Fu C-M, Chen C-Y, Huang Y-W, Lei S, Park S, Jeon B, Alshin A, Alshina E (2011) Sample adaptive offset for chroma, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-F057, Torino, July 2011
11. Fu C-M, Chen C-Y, Tsai C-Y, Huang Y-W, Lei S (2011) CE13: sample adaptive offset with LCU-independent decoding, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-E049, Geneva, Mar. 2011
12. Fu C-M, Huang Y-W, Lei S, Chong IS, Karczewicz M (2011) Non-CE8: offset coding in SAO, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G222, Geneva, Nov. 2011
13. Fu C-M, Alshina E, Alshin A, Huang Y-W, Chen C-Y, Tsai C-Y, Hsu C-W, Lei S, Park JH, Han W-J (2012) Sample adaptive offset in the HEVC standard. IEEE Trans Circuits Syst Video Technol 22(12):1755–1764
14. Fu C-M, Chen C-Y, Tsai C-Y, Huang Y-W, Lei S, Chong IS, Karczewicz M, Alshina E, Alshin A (2012) E8.a.3: SAO with LCU-based syntax, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-H0273, San Jose, Feb. 2012
15. Fu C-M, Huang Y-W, Lei S (2012) Non-CE1: bug-fix of offset coding in SAO interleaving mode, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-I0168, Geneva, Apr. 2012
16. Fuldseth A, Horowitz M, Xu S, Segall A, Zhou M (2011) Tiles, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-F335, Torino, July 2011
17. Han W-J, Min J, Kim IK, Alshina E, Alshin A, Lee T, Chen J, Seregin V, Lee S, Hong YM, Cheon MS, Sklyakhov N, McCann K, Davies T, Park JH (2010) Improved video compression efficiency through flexible unit representation and corresponding extension of coding tools. IEEE Trans Circuits Syst Video Technol 20(12):1709–1720
18. Huang Y-W, Alshina E, Chong IS, Wan W, Zhou M (2012) Description of core experiment 1 (CE1): sample adaptive offset filtering, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-H1101, San Jose, Feb. 2012
19. Ikeda M, Suzuki T (2012) Non-CE10: introduction of strong filter, Joint Collaborative Team on Video Coding, Document JCTVC-H0275, San Jose, Feb. 2012
20. Ikeda M, Tanaka J, Suzuki T (2011) CE12 Subset2: parallel deblocking filter, Joint Collaborative Team on video coding (JCT-VC), Document JCTVC-E181, Geneva, Mar. 2011
21. ITU-T Rec. H.264 and ISO/IEC 14496-10 (2003) Advanced Video Coding
22. Kim W-S (2012) AHG6: SAO parameter estimation using non-deblocked pixels, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-J0139, Stockholm, July 2012
23. Kim W-S, Kwon D-K (2012) Non-CE8: method of visual coding artifact removal for SAO, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G680, Geneva, Nov. 2012
24. Kim W-S, Kwon D-K (2012) CE8 Subset c: necessity of sign bits for SAO offsets, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-H0434, San Jose, Feb. 2012
25. Laroche G, Poirier T, Onno P (2011) On additional SAO band offset classifications, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G246, Geneva, Nov. 2011
26. Laroche G, Poirier T, Onno P (2012) Non-CE1: encoder modification for SAO interleaving mode, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-I0184, Geneva, Apr. 2012

27. List P, Josh A, Lainema J, Bjøntegaard G, Karczewicz M (2003) Adaptive loop filter. IEEE Trans Circuits Syst Video Technol 13:614 -619
28. Lou J, Jagmohan A, He D, Lu L, Sun M-T (2009) H.264 deblocking speedup. IEEE Trans Circuits Syst Video Technol 19(8):1178–1182
29. Maani E, Nakagami O (2012) Flexible band offset mode in SAO, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-H0406, San Jose, Feb. 2012
30. Minezawa A, Sugimoto K, Sekiguchi S (2012) Non-CE1: improved edge offset coding for SAO, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-I0066, Geneva, Apr. 2012
31. Minoo K, Baylon D (2012) AHG6: coding of SAO merge left and merge up flags, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-J0355, Stockholm, July 2012
32. Narroschke M, Esenlik S, Wedi T (2011) CE12 Subtest 1: results for modified decisions for deblocking, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G590, Geneva, Nov. 2011
33. Norkin A (2012) Non-CE1: non-normative improvement to deblocking filtering, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-K0289, Shanghai, Oct. 2012
34. Norkin A (2012) CE10.3: deblocking filter simplifications: Bs computation and strong filtering decision, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-H0473, San Jose, Feb. 2012
35. Norkin A, Andersson K, Sjöberg R, Huang Q, An J, Guo X, Lei S (2011) CE12: Ericsson's and MediaTek's deblocking filter, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-F118, Torino, July 2011
36. Norkin A, Andersson K, Fuldseth A, Bjøntegaard G (2012) HEVC deblocking filtering and decisions. In: *Proc. SPIE.* 8499, Applications of Digital Image Processing XXXV, no. 849912, Oct. 2012
37. Norkin A, Bjøntegaard G, Fuldseth A, Narroschke M, Ikeda M, Andersson K, Zhou M, Van der Auwera G (2012) HEVC deblocking filter. IEEE Trans Circuits Syst Video Technol 22(11):1746–1754
38. Norkin A, Andersson K, Kulyk V (2013) Two HEVC encoder methods for block artifact reduction. In: Proceedings of the IEEE international conference on visual communications and image processing (VCIP) 2013, Kuching, Sarawak, 17–20 Nov. 2013
39. Norkin A, Andersson K, Sjöberg R (2013) AHG6: on deblocking filter and parameters signaling, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-L0232, Geneva, Jan. 2013
40. ITU-T Rec. H.265 and ISO/IEC 23008-2 (2013) High efficiency video coding
41. Sullivan GJ, Wiegand T (1998) Rate-distortion optimization for video compression. IEEE Signal Processing Magazine, pp 74–90
42. Tan TK, Fujibayashi A, Suzuki Y, Takiue J (2012) AHG8: objective and subjective evaluation of HM5.0, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-H0116, San Jose, Feb. 2012
43. Ugur K, Andersson KR, Fuldseth A (2010) Video coding technology proposal by Tandberg, Nokia, and Ericsson, Joint Collaborative Team on Video Coding, Document JCTVC-A119, Dresden, Apr. 2010
44. Van der Auwera G, Wang X, Karczewicz M, Narroschke M, Kotra A, Wedi T (2011) Support of varying QP in deblocking, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G1031, Geneva, Nov. 2011
45. Xu J, Tabatabai A (2012) AHG6: on SAO signaling, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-J0268, Stockholm, July 2012
46. Yamakage T, Asaka S, Chujoh T, Karczewicz M, Chong IS (2011) CE12: deblocking filter parameter adjustment in slice level, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G174, Geneva, Nov. 2011
47. Zhou M, Sezer O, Sze V (2011) CE12 subset 2: test results and architectural study on deblocking filter without parallel on/off filter decision, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G088, Geneva, Nov. 2011