

TITLE

AN ENCODER, A DECODER AND CORRESPONDING METHODS FOR ADAPTIVE LOOP FILTERING

5 CROSS-REFERENCE TO RELATED APPLICATIONS

This patent application claims the priority to PCT/EP2020/051788, filed on January 24, 2020.

The disclosure of the aforementioned patent application is hereby incorporated by reference in its entirety.

10 TECHNICAL FIELD

Embodiments of the present application (disclosure) generally relate to the field of picture processing and more particularly to adaptive loop filtering.

BACKGROUND

- 15 Video coding (video encoding and decoding) is used in a wide range of digital video applications, for example broadcast digital TV, video transmission over internet and mobile networks, real-time conversational applications such as video chat, video conferencing, DVD and Blu-ray discs, video content acquisition and editing systems, and camcorders of security applications.
- 20 The amount of video data needed to depict even a relatively short video can be substantial, which may result in difficulties when the data is to be streamed or otherwise communicated across a communications network with limited bandwidth capacity. Thus, video data is generally compressed before being communicated across modern day telecommunications networks. The size of a video could also be an issue when the video is stored on a storage
- 25 device because memory resources may be limited. Video compression devices often use software and/or hardware at the source to code the video data prior to transmission or storage,

thereby decreasing the quantity of data needed to represent digital video images. The compressed data is then received at the destination by a video decompression device that decodes the video data. With limited network resources and ever increasing demands of higher video quality, improved compression and decompression techniques that improve compression ratio with little to no sacrifice in picture quality are desirable.

SUMMARY

Embodiments of the present application provide apparatuses and methods for encoding and decoding according to the independent claims.

The foregoing and other objects are achieved by the subject matter of the independent claims. Further implementation forms are apparent from the dependent claims, the description and the figs.

Particular embodiments are outlined in the attached independent claims, with other embodiments in the dependent claims.

15

The first aspect of the present invention provides a method of filtering a sample value of a picture, wherein the method comprises: obtaining a reconstructed sample value for a block of the picture; obtaining filter coefficients for adaptive loop filtering according to a bitstream; obtaining a sum value according to the filter coefficients and the reconstructed sample value for the block; rounding the sum value according to a vertical position of a sample in the block, to obtain a rounded sum value; obtaining a filter reconstructed sample value for the block according to the rounded sum value.

According to embodiments of the present invention, a rounding value corresponding to the right shift value during the ALF filtering process is used. This solution gives an advantage that the ALF filtering is precise and consistent when compared with the other filtering stages.

In one implementation, the rounded sum value is equal to $(\text{sum} + (1 \ll (\text{alfShiftY} - 1))) \gg \text{alfShiftY}$, wherein sum is the sum value, alfShiftY is a variable which is determined based on the vertical position of the sample.

- 5 In one implementation, alfShiftY is equal to 10 when the vertical position of the sample is satisfied any one of the below conditions:

$y == \text{CtbSizeY} - t - 1$, or $y == \text{CtbSizeY} - t$, or $y == \text{ctbHeightC} - t$, or $y == \text{ctbHeightC} - t - 1$, wherein y is the vertical position of the sample, CtbSizeY is a coding tree block, CTB, size, ctbHeightC is the height of the CTB, t is an integer value.

- 10 In one implementation, alfShiftY is equal to 7 when the vertical position of the sample is not satisfied any one of the below conditions:

$y == \text{CtbSizeY} - t - 1$, or $y == \text{CtbSizeY} - t$, or $y == \text{ctbHeightC} - t$, or $y == \text{ctbHeightC} - t - 1$, wherein y is the vertical position of the sample, CtbSizeY is a coding tree block, CTB, size, ctbHeightC is the height of the CTB, t is an integer value.

- 15 In one implementation, when y is equal to 0, alfShiftY is equal to 10.

In one implementation, when y is not equal to 0, alfShiftY is equal to 7.

In one implementation, the vertical position of the sample in the block is a chroma sample position or is a luma sample position.

- 20 In one implementation, the rounding process comprises: adding a value which is determined based on the vertical position to the sum value in order to obtain an added sum value, and shifting the added sum value based on the vertical position.

In one implementation, obtaining a filter reconstructed sample value for the block according to the rounded sum value comprises: obtaining a filter reconstructed sample value for the block according to the rounded sum value and the reconstructed sample value for the

- 25 block.

In one implementation, the block is a coding tree block.

The second aspect of the present invention provides a video decoding apparatus, the apparatus comprise: a reconstructing module, which is configured to obtain a reconstructed

sample value for a block of a picture; a parsing module, which is configured to obtain filter coefficients for adaptive loop filtering according to a bitstream; a calculating module, which is configured to obtain a sum value according to the filter coefficients and the reconstructed sample value for the block; a rounding module, which is configured to round the sum value according to a vertical position of a sample in the block, to obtain a rounded sum value; a filtering module, which is configured to obtain a filter reconstructed sample value for the block according to the rounded sum value.

According to embodiments of the present invention, a rounding value corresponding to the right shift value during the ALF filtering process is used. This solution gives an advantage that the ALF filtering is precise and consistent when compared with the other filtering stages.

In one implementation, the rounded sum value is equal to $(\text{sum} + (1 \ll (\text{alfShiftY} - 1))) \gg \text{alfShiftY}$, wherein sum is the sum value, alfShiftY is a variable which is determined based on the vertical position of the sample.

In one implementation, alfShiftY is equal to 10 when the vertical position of the sample is satisfied any one of the below conditions:

$y == \text{CtbSizeY} - t - 1$, or $y == \text{CtbSizeY} - t$, or $y == \text{ctbHeightC} - t$, or $y == \text{ctbHeightC} - t - 1$, wherein y is the vertical position of the sample, CtbSizeY is a coding tree block, CTB, size, ctbHeightC is the height of the CTB, t is an integer value.

In one implementation, alfShiftY is equal to 7 when the vertical position of the sample is not satisfied any one of the below conditions:

$y == \text{CtbSizeY} - t - 1$, or $y == \text{CtbSizeY} - t$, or $y == \text{ctbHeightC} - t$, or $y == \text{ctbHeightC} - t - 1$, wherein y is the vertical position of the sample, CtbSizeY is a coding tree block, CTB, size, ctbHeightC is the height of the CTB, t is an integer value.

In one implementation, when y is equal to 0, alfShiftY is equal to 10.

In one implementation, when y is not equal to 0, alfShiftY is equal to 7.

In one implementation, the vertical position of the sample in the block is a chroma sample position or is a luma sample position.

In one implementation, the rounding module is configured to
add value which is determined based on the vertical position to the sum value in order to
5 obtain an added sum value, and shift the added sum value based on the vertical position, to
obtain the rounded sum value.

The third aspect of the present invention provides a decoder comprising processing circuitry
for carrying out the method according to the first aspect and any one of implementation of the
10 first aspect.

The fourth aspect of the present invention provides a computer program product comprising
program code for performing the method according to the first aspect, the third aspect and
any one of implementation of the first aspect, the third aspect when executed on a computer
15 or a processor.

The fifth aspect of the present invention provides a decoder, comprising: one or more
processors; and a non-transitory computer-readable storage medium coupled to the processors
and storing programming for execution by the processors, wherein the programming, when
20 executed by the processors, configures the decoder to carry out the method according to any
one of the first aspect, the third aspect and any one of implementation of the first aspect, the
third aspect.

The sixth aspect of the present invention provides a non-transitory computer-readable
25 medium carrying a program code which, when executed by a computer device, causes the

computer device to perform the method according to any one of the first aspect, the third aspect and any one of implementation of the first aspect, the third aspect.

The seventh aspect of the present invention provides a non-transitory storage medium comprising a bitstream encoded/decoded by the method of any one of the above
5 embodiments.

The eighth aspect of the present invention provides an encoded bitstream for the video signal by including a plurality of syntax elements, wherein the plurality of syntax elements comprises a deblocking control parameter for chroma component, which is conditionally
10 signaled at least based on a value of a syntax element, wherein the value of the syntax element is related to a deblocking control parameter for a chroma component of a slice of a coded picture.

The method according to the first aspect of the invention can be performed by the apparatus
15 according to the second aspect of the invention. Further features and implementation forms of the method according to the first aspect of the invention correspond to the features and implementation forms of the apparatus according to the second aspect of the invention. Details of one or more embodiments are set forth in the accompanying drawings and the description below. Other features, objects, and advantages will be apparent from the
20 description, drawings, and claims.

The ninth aspect of the present invention provides a method of filtering a sample value of a picture, wherein the method comprises: obtaining a sample value for a block of the picture; obtaining filter coefficients for adaptive loop filtering;
25 obtaining a sum value according to the filter coefficients and the sample value for the block; rounding the sum value according to a vertical position of a sample in the block, to obtain a

rounded sum value; obtaining a filter sample value for the block according to the rounded sum value; encoding the filter sample value for the block to obtain a bitstream.

According to embodiments of the present invention, a rounding value corresponding to the right shift value during the ALF filtering process is used. This solution gives an advantage that the ALF filtering is precise and consistent when compared with the other filtering stages.

In one implementation, the rounded sum value is equal to $(\text{sum} + (1 \ll (\text{alfShiftY} - 1))) \gg \text{alfShiftY}$, wherein sum is the sum value, alfShiftY is a variable which is determined based on the vertical position of the sample.

In one implementation, alfShiftY is equal to 10 when the vertical position of the sample is satisfied any one of the below conditions:

$y == \text{CtbSizeY} - t - 1$, or $y == \text{CtbSizeY} - t$, or $y == \text{ctbHeightC} - t$, or $y == \text{ctbHeightC} - t - 1$, wherein y is the vertical position of the sample, CtbSizeY is a coding tree block, CTB, size, ctbHeightC is the height of the CTB, t is an integer value.

In one implementation, alfShiftY is equal to 7 when the vertical position of the sample is not satisfied any one of the below conditions:

$y == \text{CtbSizeY} - t - 1$, or $y == \text{CtbSizeY} - t$, or $y == \text{ctbHeightC} - t$, or $y == \text{ctbHeightC} - t - 1$, wherein y is the vertical position of the sample, CtbSizeY is a coding tree block, CTB, size, ctbHeightC is the height of the CTB, t is an integer value.

In one implementation, when y is equal to 0, alfShiftY is equal to 10.

In one implementation, when y is not equal to 0, alfShiftY is equal to 7.

In one implementation, the vertical position of the sample in the block is a chroma sample position or is a luma sample position.

In one implementation, the rounding process comprises: adding a value which is determined based on the vertical position to the sum value in order to obtain an added sum value, and shifting the added sum value based on the vertical position.

In one implementation, obtaining a filter sample value for the block according to the rounded sum value comprises: obtaining a filter sample value for the block according to the rounded sum value and the sample value for the block.

In one implementation, the block is a coding tree block.

The tenth aspect of the present invention provides a video encoding apparatus, the apparatus comprise: a obtaining module, which is configured to obtain a sample value for a block of a picture; the obtaining module is configured to obtain filter coefficients for adaptive loop filtering; a calculating module, which is configured to obtain a sum value according to the filter coefficients and the sample value for the block; a rounding module, which is configured to round the sum value according to a vertical position of a sample in the block, to obtain a rounded sum value; a filtering module, which is configured to obtain a filter sample value for the block according to the rounded sum value; a bitstream generating module, which is configured to encode the filter sample value to obtain a bitstream.

According to embodiments of the present invention, a rounding value corresponding to the right shift value during the ALF filtering process is used. This solution gives an advantage that the ALF filtering is precise and consistent when compared with the other filtering stages.

In one implementation, the rounded sum value is equal to $(\text{sum} + (1 \ll (\text{alfShiftY} - 1))) \gg \text{alfShiftY}$, wherein sum is the sum value, alfShiftY is a variable which is determined based on the vertical position of the sample.

In one implementation, alfShiftY is equal to 10 when the vertical position of the sample is satisfied any one of the below conditions:

$y = \text{CtbSizeY} - t - 1$, or $y = \text{CtbSizeY} - t$, or $y = \text{ctbHeightC} - t$, or $y = \text{ctbHeightC} - t - 1$, wherein y is the vertical position of the sample, CtbSizeY is a coding tree block, CTB , size, ctbHeightC is the height of the CTB , t is an integer value.

In one implementation, alfShiftY is equal to 7 when the vertical position of the sample is not satisfied any one of the below conditions:

$y = \text{CtbSizeY} - t - 1$, or $y = \text{CtbSizeY} - t$, or $y = \text{ctbHeightC} - t$, or $y = \text{ctbHeightC} - t - 1$, wherein y is the vertical position of the sample, CtbSizeY is a coding tree block, CTB , size, ctbHeightC is the height of the CTB , t is an integer value.

In one implementation, when y is equal to 0, alfShiftY is equal to 10.

In one implementation, when y is not equal to 0, alfShiftY is equal to 7.

In one implementation, the vertical position of the sample in the block is a chroma sample position or is a luma sample position.

In one implementation, the rounding module is configured to add value which is determined based on the vertical position to the sum value in order to obtain an added sum value, and shift the added sum value based on the vertical position, to obtain the rounded sum value.

The eleventh aspect of the present invention provides an encoder comprising processing circuitry for carrying out the method according to the ninth aspect and any one of implementation of the ninth aspect.

The twelfth aspect of the present invention provides a computer program product comprising program code for performing the method according to the ninth aspect, the third aspect and any one of implementation of the ninth aspect when executed on a computer or a processor.

The thirteenth aspect of the present invention provides an encoder, comprising: one or more processors; and a non-transitory computer-readable storage medium coupled to the processors and storing programming for execution by the processors, wherein the programming, when

executed by the processors, configures the encoder to carry out the method according to any one of the ninth aspect, the third aspect and any one of implementation of the ninth aspect.

The fourteenth aspect of the present invention provides a non-transitory computer-readable medium carrying a program code which, when executed by a computer device, causes the

5 computer device to perform the method according to any one of the ninth aspect and any one of implementation of the ninth aspect.

The fifteenth aspect of the present invention provides a non-transitory storage medium comprising a bitstream encoded/decoded by the method of any one of the above
10 embodiments.

BRIEF DESCRIPTION OF THE DRAWINGS

In the following embodiments of the invention are described in more detail with reference to the attached figures and drawings, in which:

15 Fig. 1A is a block diagram showing an example of a video coding system configured to implement embodiments of the invention;

Fig. 1B is a block diagram showing another example of a video coding system configured to implement embodiments of the invention;

Fig. 2 is a block diagram showing an example of a video encoder configured to
20 implement embodiments of the invention;

Fig. 3 is a block diagram showing an example structure of a video decoder configured to implement embodiments of the invention;

Fig. 4 is a block diagram illustrating an example of an encoding apparatus or a decoding apparatus;

25

Figs. 6 to 22 schematically illustrate examples of embodiments of the invention;

Fig. 23 is a block diagram showing an example structure of a content supply system 3100 which realizes a content delivery service;

Fig. 24 is a block diagram showing a structure of an example of a terminal device.

5 Fig. 25 is a flowchart showing a method embodiment according to the present invention.

Fig. 26 is a block diagram showing an apparatus embodiment according to the present invention.

In the following identical reference signs refer to identical or at least functionally equivalent
10 features if not explicitly specified otherwise.

DETAILED DESCRIPTION OF THE EMBODIMENTS

In the following description, reference is made to the accompanying figures, which form part of the disclosure, and which show, by way of illustration, specific aspects of embodiments of
15 the invention or specific aspects in which embodiments of the present invention may be used.

It is understood that embodiments of the invention may be used in other aspects and comprise structural or logical changes not depicted in the figures. The following detailed description, therefore, is not to be taken in a limiting sense, and the scope of the present invention is defined by the appended claims.

20 For instance, it is understood that a disclosure in connection with a described method may also hold true for a corresponding device or system configured to perform the method and vice versa. For example, if one or a plurality of specific method steps are described, a corresponding device may include one or a plurality of units, e.g., functional units, to perform the described one or plurality of method steps (e.g., one unit performing the one or plurality
25 of steps, or a plurality of units each performing one or more of the plurality of steps), even if such one or more units are not explicitly described or illustrated in the figures. On the other

hand, for example, if a specific apparatus is described based on one or a plurality of units, e.g., functional units, a corresponding method may include one step to perform the functionality of the one or plurality of units (e.g., one step performing the functionality of the one or plurality of units, or a plurality of steps each performing the functionality of one or more of the plurality of units), even if such one or plurality of steps are not explicitly described or illustrated in the figures. Further, it is understood that the features of the various exemplary embodiments and/or aspects described herein may be combined with each other, unless specifically noted otherwise.

Video coding typically refers to the processing of a sequence of pictures, which form the video or video sequence. Instead of the term “picture” the term “frame” or “image” may be used as synonyms in the field of video coding. Video coding (or coding in general) comprises two parts video encoding and video decoding. Video encoding is performed at the source side, typically comprising processing (e.g., by compression) the original video pictures to reduce the amount of data required for representing the video pictures (for more efficient storage and/or transmission). Video decoding is performed at the destination side and typically comprises the inverse processing compared to the encoder to reconstruct the video pictures. Embodiments referring to “coding” of video pictures (or pictures in general) shall be understood to relate to “encoding” or “decoding” of video pictures or respective video sequences. The combination of the encoding part and the decoding part is also referred to as CODEC (Coding and Decoding).

In case of lossless video coding, the original video pictures can be reconstructed, i.e. the reconstructed video pictures have the same quality as the original video pictures (assuming no transmission loss or other data loss during storage or transmission). In case of lossy video coding, further compression, e.g., by quantization, is performed, to reduce the amount of data representing the video pictures, which cannot be completely reconstructed at the decoder, i.e.

the quality of the reconstructed video pictures is lower or worse compared to the quality of the original video pictures.

Several video coding standards belong to the group of “lossy hybrid video codecs” (i.e. combine spatial and temporal prediction in the sample domain and 2D transform coding for applying quantization in the transform domain). Each picture of a video sequence is typically partitioned into a set of non-overlapping blocks and the coding is typically performed on a block level. In other words, at the encoder the video is typically processed, i.e. encoded, on a block (video block) level, e.g., by using spatial (intra picture) prediction and/or temporal (inter picture) prediction to generate a prediction block, subtracting the prediction block from the current block (block currently processed/to be processed) to obtain a residual block, transforming the residual block and quantizing the residual block in the transform domain to reduce the amount of data to be transmitted (compression), whereas at the decoder the inverse processing compared to the encoder is applied to the encoded or compressed block to reconstruct the current block for representation. Furthermore, the encoder duplicates the decoder processing loop such that both will generate identical predictions (e.g., intra- and inter predictions) and/or re-constructions for processing, i.e. coding, the subsequent blocks. In the following embodiments of a video coding system 10, a video encoder 20 and a video decoder 30 are described based on Figs. 1 to 3.

Fig. 1A is a schematic block diagram illustrating an example coding system 10, e.g., a video coding system 10 (or short coding system 10) that may utilize techniques of this present application. Video encoder 20 (or short encoder 20) and video decoder 30 (or short decoder 30) of video coding system 10 represent examples of devices that may be configured to perform techniques in accordance with various examples described in the present application. As shown in Fig. 1A, the coding system 10 comprises a source device 12 configured to provide encoded picture data 21 e.g., to a destination device 14 for decoding the encoded picture data 13.

The source device 12 comprises an encoder 20, and may additionally, i.e. optionally, comprise a picture source 16, a pre-processor (or pre-processing unit) 18, e.g., a picture pre-processor 18, and a communication interface or communication unit 22.

The picture source 16 may comprise or be any kind of picture capturing device, for example a camera for capturing a real-world picture, and/or any kind of a picture generating device, for example a computer-graphics processor for generating a computer animated picture, or any kind of other device for obtaining and/or providing a real-world picture, a computer generated picture (e.g., a screen content, a virtual reality (VR) picture) and/or any combination thereof (e.g., an augmented reality (AR) picture). The picture source may be any kind of memory or storage storing any of the aforementioned pictures.

In distinction to the pre-processor 18 and the processing performed by the pre-processing unit 18, the picture or picture data 17 may also be referred to as raw picture or raw picture data 17.

Pre-processor 18 is configured to receive the (raw) picture data 17 and to perform pre-processing on the picture data 17 to obtain a pre-processed picture 19 or pre-processed picture data 19. Pre-processing performed by the pre-processor 18 may, e.g., comprise trimming, color format conversion (e.g., from RGB to YCbCr), color correction, or de-noising. It can be understood that the pre-processing unit 18 may be optional component.

The video encoder 20 is configured to receive the pre-processed picture data 19 and provide encoded picture data 21 (further details will be described below, e.g., based on Fig. 2).

Communication interface 22 of the source device 12 may be configured to receive the encoded picture data 21 and to transmit the encoded picture data 21 (or any further processed version thereof) over communication channel 13 to another device, e.g., the destination device 14 or any other device, for storage or direct reconstruction.

The destination device 14 comprises a decoder 30 (e.g., a video decoder 30), and may additionally, i.e. optionally, comprise a communication interface or communication unit 28, a post-processor 32 (or post-processing unit 32) and a display device 34.

The communication interface 28 of the destination device 14 is configured receive the
5 encoded picture data 21 (or any further processed version thereof), e.g., directly from the source device 12 or from any other source, e.g., a storage device, e.g., an encoded picture data storage device, and provide the encoded picture data 21 to the decoder 30.

The communication interface 22 and the communication interface 28 may be configured to transmit or receive the encoded picture data 21 or encoded data 13 via a direct
10 communication link between the source device 12 and the destination device 14, e.g., a direct wired or wireless connection, or via any kind of network, e.g., a wired or wireless network or any combination thereof, or any kind of private and public network, or any kind of combination thereof.

The communication interface 22 may be, e.g., configured to package the encoded picture data
15 21 into an appropriate format, e.g., packets, and/or process the encoded picture data using any kind of transmission encoding or processing for transmission over a communication link or communication network.

The communication interface 28, forming the counterpart of the communication interface 22, may be, e.g., configured to receive the transmitted data and process the transmission data
20 using any kind of corresponding transmission decoding or processing and/or de-packaging to obtain the encoded picture data 21.

Both, communication interface 22 and communication interface 28 may be configured as unidirectional communication interfaces as indicated by the arrow for the communication channel 13 in Fig. 1A pointing from the source device 12 to the destination device 14, or
25 bi-directional communication interfaces, and may be configured, e.g., to send and receive messages, e.g., to set up a connection, to acknowledge and exchange any other information

related to the communication link and/or data transmission, e.g., encoded picture data transmission.

The decoder 30 is configured to receive the encoded picture data 21 and provide decoded picture data 31 or a decoded picture 31 (further details will be described below, e.g., based on
5 Fig. 3 or Fig. 5).

The post-processor 32 of destination device 14 is configured to post-process the decoded picture data 31 (also called reconstructed picture data), e.g., the decoded picture 31, to obtain post-processed picture data 33, e.g., a post-processed picture 33. The post-processing performed by the post-processing unit 32 may comprise, e.g., color format conversion (e.g.,
10 from YCbCr to RGB), color correction, trimming, or re-sampling, or any other processing, e.g., for preparing the decoded picture data 31 for display, e.g., by display device 34.

The display device 34 of the destination device 14 is configured to receive the post-processed picture data 33 for displaying the picture, e.g., to a user or viewer. The display device 34 may be or comprise any kind of display for representing the reconstructed picture, e.g., an
15 integrated or external display or monitor. The displays may, e.g., comprise liquid crystal displays (LCD), organic light emitting diodes (OLED) displays, plasma displays, projectors, micro LED displays, liquid crystal on silicon (LCoS), digital light processor (DLP) or any kind of other display.

Although Fig. 1A depicts the source device 12 and the destination device 14 as separate
20 devices, embodiments of devices may also comprise both or both functionalities, the source device 12 or corresponding functionality and the destination device 14 or corresponding functionality. In such embodiments the source device 12 or corresponding functionality and the destination device 14 or corresponding functionality may be implemented using the same hardware and/or software or by separate hardware and/or software or any combination
25 thereof.

As will be apparent for the skilled person based on the description, the existence and (exact) split of functionalities of the different units or functionalities within the source device 12 and/or destination device 14 as shown in Fig. 1A may vary depending on the actual device and application.

5 The encoder 20 (e.g., a video encoder 20) or the decoder 30 (e.g., a video decoder 30) or both encoder 20 and decoder 30 may be implemented via processing circuitry as shown in Fig. 1B, such as one or more microprocessors, digital signal processors (DSPs), application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs), discrete logic, hardware, video coding dedicated or any combinations thereof. The encoder 20 may be
10 implemented via processing circuitry 46 to embody the various modules as discussed with respect to encoder 20 of Fig. 2 and/or any other encoder system or subsystem described herein. The decoder 30 may be implemented via processing circuitry 46 to embody the various modules as discussed with respect to decoder 30 of Fig. 3 and/or any other decoder system or subsystem described herein. The processing circuitry may be configured to perform the
15 various operations as discussed later. As shown in Fig. 5, if the techniques are implemented partially in software, a device may store instructions for the software in a suitable, non-transitory computer-readable storage medium and may execute the instructions in hardware using one or more processors to perform the techniques of this disclosure. Either of video encoder 20 and video decoder 30 may be integrated as part of a combined
20 encoder/decoder (CODEC) in a single device, for example, as shown in Fig. 1B. Source device 12 and destination device 14 may comprise any of a wide range of devices, including any kind of handheld or stationary devices, e.g., notebook or laptop computers, mobile phones, smart phones, tablets or tablet computers, cameras, desktop computers, set-top boxes, televisions, display devices, digital media players, video gaming consoles,
25 video streaming devices (such as content services servers or content delivery servers), broadcast receiver device, broadcast transmitter device, or the like and may use no or any

kind of operating system. In some cases, the source device 12 and the destination device 14 may be equipped for wireless communication. Thus, the source device 12 and the destination device 14 may be wireless communication devices.

In some cases, video coding system 10 illustrated in Fig. 1A is merely an example and the techniques of the present application may apply to video coding settings (e.g., video encoding or video decoding) that do not necessarily include any data communication between the encoding and decoding devices. In other examples, data is retrieved from a local memory, streamed over a network, or the like. A video encoding device may encode and store data to memory, and/or a video decoding device may retrieve and decode data from memory. In some examples, the encoding and decoding is performed by devices that do not communicate with one another, but simply encode data to memory and/or retrieve and decode data from memory.

For convenience of description, embodiments of the invention are described herein, for example, by reference to High-Efficiency Video Coding (HEVC) or to the reference software of Versatile Video coding (VVC), the next generation video coding standard developed by the Joint Collaboration Team on Video Coding (JCT-VC) of ITU-T Video Coding Experts Group (VCEG) and ISO/IEC Motion Picture Experts Group (MPEG). One of ordinary skill in the art will understand that embodiments of the invention are not limited to HEVC or VVC.

Encoder and Encoding Method

Fig. 2 shows a schematic block diagram of an example video encoder 20 that is configured to implement the techniques of the present application. In the example of Fig. 2, the video encoder 20 comprises an input 201 (or input interface 201), a residual calculation unit 204, a transform processing unit 206, a quantization unit 208, an inverse quantization unit 210, and inverse transform processing unit 212, a reconstruction unit 214, a loop filter unit 220, a decoded picture buffer (DPB) 230, a mode selection unit 260, an entropy encoding unit 270 and an output 272 (or output interface 272). The mode selection unit 260 may include an inter

prediction unit 244, an intra prediction unit 254 and a partitioning unit 262. Inter prediction unit 244 may include a motion estimation unit and a motion compensation unit (not shown). A video encoder 20 as shown in Fig. 2 may also be referred to as hybrid video encoder or a video encoder according to a hybrid video codec.

5 The residual calculation unit 204, the transform processing unit 206, the quantization unit 208, the mode selection unit 260 may be referred to as forming a forward signal path of the encoder 20, whereas the inverse quantization unit 210, the inverse transform processing unit 212, the reconstruction unit 214, the buffer 216, the loop filter 220, the decoded picture buffer (DPB) 230, the inter prediction unit 244 and the intra-prediction unit 254 may be
10 referred to as forming a backward signal path of the video encoder 20, wherein the backward signal path of the video encoder 20 corresponds to the signal path of the decoder (see video decoder 30 in Fig. 3). The inverse quantization unit 210, the inverse transform processing unit 212, the reconstruction unit 214, the loop filter 220, the decoded picture buffer (DPB) 230, the inter prediction unit 244 and the intra-prediction unit 254 are also referred to forming
15 the “built-in decoder” of video encoder 20.

Pictures & Picture Partitioning (Pictures & Blocks)

The encoder 20 may be configured to receive, e.g., via input 201, a picture 17 (or picture data 17), e.g., picture of a sequence of pictures forming a video or video sequence. The received picture or picture data may also be a pre-processed picture 19 (or pre-processed picture data
20 19). For sake of simplicity the following description refers to the picture 17. The picture 17 may also be referred to as current picture or picture to be coded (in particular in video coding to distinguish the current picture from other pictures, e.g., previously encoded and/or decoded pictures of the same video sequence, i.e. the video sequence which also comprises the current picture).

25 A (digital) picture is or can be regarded as a two-dimensional array or matrix of samples with intensity values. A sample in the array may also be referred to as pixel (short form of picture

element) or a pel. The number of samples in horizontal and vertical direction (or axis) of the array or picture define the size and/or resolution of the picture. For representation of color, typically three color components are employed, i.e. the picture may be represented or include three sample arrays. In RGB format or color space a picture comprises a corresponding red, green and blue sample array. However, in video coding each pixel is typically represented in a luminance and chrominance format or color space, e.g., YCbCr, which comprises a luminance component indicated by Y (sometimes also L is used instead) and two chrominance components indicated by Cb and Cr. The luminance (or short luma) component Y represents the brightness or grey level intensity (e.g., like in a grey-scale picture), while the two chrominance (or short chroma) components Cb and Cr represent the chromaticity or color information components. Accordingly, a picture in YCbCr format comprises a luminance sample array of luminance sample values (Y), and two chrominance sample arrays of chrominance values (Cb and Cr). Pictures in RGB format may be converted or transformed into YCbCr format and vice versa, the process is also known as color transformation or conversion. If a picture is monochrome, the picture may comprise only a luminance sample array. Accordingly, a picture may be, for example, an array of luma samples in monochrome format or an array of luma samples and two corresponding arrays of chroma samples in 4:2:0, 4:2:2, and 4:4:4 colour format.

Embodiments of the video encoder 20 may comprise a picture partitioning unit (not depicted in Fig. 2) configured to partition the picture 17 into a plurality of (typically non-overlapping) picture blocks 203. These blocks may also be referred to as root blocks, macro blocks (H.264/AVC) or coding tree blocks (CTB) or coding tree units (CTU) (H.265/HEVC and VVC). The picture partitioning unit may be configured to use the same block size for all pictures of a video sequence and the corresponding grid defining the block size, or to change the block size between pictures or subsets or groups of pictures, and partition each picture into the corresponding blocks.

In further embodiments, the video encoder may be configured to receive directly a block 203 of the picture 17, e.g., one, several or all blocks forming the picture 17. The picture block 203 may also be referred to as current picture block or picture block to be coded.

Like the picture 17, the picture block 203 again is or can be regarded as a two-dimensional
5 array or matrix of samples with intensity values (sample values), although of smaller dimension than the picture 17. In other words, the block 203 may comprise, e.g., one sample array (e.g., a luma array in case of a monochrome picture 17, or a luma or chroma array in case of a color picture) or three sample arrays (e.g., a luma and two chroma arrays in case of a color picture 17) or any other number and/or kind of arrays depending on the color format
10 applied. The number of samples in horizontal and vertical direction (or axis) of the block 203 define the size of block 203. Accordingly, a block may, for example, an $M \times N$ (M -column by N -row) array of samples, or an $M \times N$ array of transform coefficients.

Embodiments of the video encoder 20 as shown in Fig. 2 may be configured encode the picture 17 block by block, e.g., the encoding and prediction is performed per block 203.

15 Residual Calculation

The residual calculation unit 204 may be configured to calculate a residual block 205 (also referred to as residual 205) based on the picture block 203 and a prediction block 265 (further details about the prediction block 265 are provided later), e.g., by subtracting sample values of the prediction block 265 from sample values of the picture block 203, sample by sample
20 (pixel by pixel) to obtain the residual block 205 in the sample domain.

Transform

The transform processing unit 206 may be configured to apply a transform, e.g., a discrete cosine transform (DCT) or discrete sine transform (DST), on the sample values of the residual block 205 to obtain transform coefficients 207 in a transform domain. The transform
25 coefficients 207 may also be referred to as transform residual coefficients and represent the residual block 205 in the transform domain.

The transform processing unit 206 may be configured to apply integer approximations of DCT/DST, such as the transforms specified for H.265/HEVC. Compared to an orthogonal DCT transform, such integer approximations are typically scaled by a certain factor. In order to preserve the norm of the residual block which is processed by forward and inverse transforms, additional scaling factors are applied as part of the transform process. The scaling factors are typically chosen based on certain constraints like scaling factors being a power of two for shift operations, bit depth of the transform coefficients, tradeoff between accuracy and implementation costs, etc. Specific scaling factors are, for example, specified for the inverse transform, e.g., by inverse transform processing unit 212 (and the corresponding inverse transform, e.g., by inverse transform processing unit 312 at video decoder 30) and corresponding scaling factors for the forward transform, e.g., by transform processing unit 206, at an encoder 20 may be specified accordingly.

Embodiments of the video encoder 20 (respectively transform processing unit 206) may be configured to output transform parameters, e.g., a type of transform or transforms, e.g., directly or encoded or compressed via the entropy encoding unit 270, so that, e.g., the video decoder 30 may receive and use the transform parameters for decoding.

Quantization

The quantization unit 208 may be configured to quantize the transform coefficients 207 to obtain quantized coefficients 209, e.g., by applying scalar quantization or vector quantization. The quantized coefficients 209 may also be referred to as quantized transform coefficients 209 or quantized residual coefficients 209.

The quantization process may reduce the bit depth associated with some or all of the transform coefficients 207. For example, an n-bit transform coefficient may be rounded down to an m-bit Transform coefficient during quantization, where n is greater than m. The degree of quantization may be modified by adjusting a quantization parameter (QP). For example for scalar quantization, different scaling may be applied to achieve finer or coarser quantization.

Smaller quantization step sizes correspond to finer quantization, whereas larger quantization step sizes correspond to coarser quantization. The applicable quantization step size may be indicated by a quantization parameter (QP). The quantization parameter may for example be an index to a predefined set of applicable quantization step sizes. For example, small

5 quantization parameters may correspond to fine quantization (small quantization step sizes) and large quantization parameters may correspond to coarse quantization (large quantization step sizes) or vice versa. The quantization may include division by a quantization step size and a corresponding and/or the inverse dequantization, e.g., by inverse quantization unit 210, may include multiplication by the quantization step size. Embodiments according to some

10 standards, e.g., HEVC, may be configured to use a quantization parameter to determine the quantization step size. Generally, the quantization step size may be calculated based on a quantization parameter using a fixed point approximation of an equation including division. Additional scaling factors may be introduced for quantization and dequantization to restore the norm of the residual block, which might get modified because of the scaling used in the

15 fixed point approximation of the equation for quantization step size and quantization parameter. In one example implementation, the scaling of the inverse transform and dequantization might be combined. Alternatively, customized quantization tables may be used and signaled from an encoder to a decoder, e.g., in a bitstream. The quantization is a lossy operation, wherein the loss increases with increasing quantization step sizes.

20 Embodiments of the video encoder 20 (respectively quantization unit 208) may be configured to output quantization parameters (QP), e.g., directly or encoded via the entropy encoding unit 270, so that, e.g., the video decoder 30 may receive and apply the quantization parameters for decoding.

Inverse Quantization

25 The inverse quantization unit 210 is configured to apply the inverse quantization of the quantization unit 208 on the quantized coefficients to obtain dequantized coefficients 211,

e.g., by applying the inverse of the quantization scheme applied by the quantization unit 208 based on or using the same quantization step size as the quantization unit 208. The dequantized coefficients 211 may also be referred to as dequantized residual coefficients 211 and correspond - although typically not identical to the transform coefficients due to the loss
5 by quantization - to the transform coefficients 207.

Inverse Transform

The inverse transform processing unit 212 is configured to apply the inverse transform of the transform applied by the transform processing unit 206, e.g., an inverse discrete cosine transform (DCT) or inverse discrete sine transform (DST) or other inverse transforms, to
10 obtain a reconstructed residual block 213 (or corresponding dequantized coefficients 213) in the sample domain. The reconstructed residual block 213 may also be referred to as transform block 213.

Reconstruction

The reconstruction unit 214 (e.g., adder or summer 214) is configured to add the transform
15 block 213 (i.e. reconstructed residual block 213) to the prediction block 265 to obtain a reconstructed block 215 in the sample domain, e.g., by adding – sample by sample - the sample values of the reconstructed residual block 213 and the sample values of the prediction block 265.

Filtering

20 The loop filter unit 220 (or short “loop filter” 220), is configured to filter the reconstructed block 215 to obtain a filtered block 221, or in general, to filter reconstructed samples to obtain filtered samples. The loop filter unit is, e.g., configured to smooth pixel transitions, or otherwise improve the video quality. The loop filter unit 220 may comprise one or more loop filters such as a de-blocking filter, a sample-adaptive offset (SAO) filter or one or more other
25 filters, e.g., a bilateral filter, an adaptive loop filter (ALF), a sharpening, a smoothing filters or a collaborative filters, or any combination thereof. Although the loop filter unit 220 is

shown in Fig. 2 as being an in loop filter, in other configurations, the loop filter unit 220 may be implemented as a post loop filter. The filtered block 221 may also be referred to as filtered reconstructed block 221.

Embodiments of the video encoder 20 (respectively loop filter unit 220) may be configured to
5 output loop filter parameters (such as sample adaptive offset information), e.g., directly or encoded via the entropy encoding unit 270, so that, e.g., a decoder 30 may receive and apply the same loop filter parameters or respective loop filters for decoding.

Decoded Picture Buffer

The decoded picture buffer (DPB) 230 may be a memory that stores reference pictures, or in
10 general reference picture data, for encoding video data by video encoder 20. The DPB 230 may be formed by any of a variety of memory devices, such as dynamic random access memory (DRAM), including synchronous DRAM (SDRAM), magnetoresistive RAM (MRAM), resistive RAM (RRAM), or other types of memory devices. The decoded picture buffer (DPB) 230 may be configured to store one or more filtered blocks 221. The decoded
15 picture buffer 230 may be further configured to store other previously filtered blocks, e.g., previously reconstructed and filtered blocks 221, of the same current picture or of different pictures, e.g., previously reconstructed pictures, and may provide complete previously reconstructed, i.e. decoded, pictures (and corresponding reference blocks and samples) and/or a partially reconstructed current picture (and corresponding reference blocks and samples),
20 for example for inter prediction. The decoded picture buffer (DPB) 230 may be also configured to store one or more unfiltered reconstructed blocks 215, or in general unfiltered reconstructed samples, e.g., if the reconstructed block 215 is not filtered by loop filter unit 220, or any other further processed version of the reconstructed blocks or samples.

Mode Selection (Partitioning & Prediction)

25 The mode selection unit 260 comprises partitioning unit 262, inter-prediction unit 244 and intra-prediction unit 254, and is configured to receive or obtain original picture data, e.g., an

original block 203 (current block 203 of the current picture 17), and reconstructed picture data, e.g., filtered and/or unfiltered reconstructed samples or blocks of the same (current) picture and/or from one or a plurality of previously decoded pictures, e.g., from decoded picture buffer 230 or other buffers (e.g., line buffer, not shown).. The reconstructed picture data is used as reference picture data for prediction, e.g., inter-prediction or intra-prediction, to obtain a prediction block 265 or predictor 265.

Mode selection unit 260 may be configured to determine or select a partitioning for a current block prediction mode (including no partitioning) and a prediction mode (e.g., an intra or inter prediction mode) and generate a corresponding prediction block 265, which is used for the calculation of the residual block 205 and for the reconstruction of the reconstructed block 215.

Embodiments of the mode selection unit 260 may be configured to select the partitioning and the prediction mode (e.g., from those supported by or available for mode selection unit 260), which provide the best match or in other words the minimum residual (minimum residual means better compression for transmission or storage), or a minimum signaling overhead (minimum signaling overhead means better compression for transmission or storage), or which considers or balances both. The mode selection unit 260 may be configured to determine the partitioning and prediction mode based on rate distortion optimization (RDO), i.e. select the prediction mode which provides a minimum rate distortion. Terms like “best”, “minimum”, “optimum” etc. in this context do not necessarily refer to an overall “best”, “minimum”, “optimum”, etc. but may also refer to the fulfillment of a termination or selection criterion like a value exceeding or falling below a threshold or other constraints leading potentially to a “sub-optimum selection” but reducing complexity and processing time.

In other words, the partitioning unit 262 may be configured to partition the block 203 into smaller block partitions or sub-blocks (which form again blocks), e.g., iteratively using

quad-tree-partitioning (QT), binary partitioning (BT) or triple-tree-partitioning (TT) or any combination thereof, and to perform, e.g., the prediction for each of the block partitions or sub-blocks, wherein the mode selection comprises the selection of the tree-structure of the partitioned block 203 and the prediction modes are applied to each of the block partitions or sub-blocks.

In the following the partitioning (e.g., by partitioning unit 260) and prediction processing (by inter-prediction unit 244 and intra-prediction unit 254) performed by an example video encoder 20 will be explained in more detail.

Partitioning

The partitioning unit 262 may partition (or split) a current block 203 into smaller partitions, e.g., smaller blocks of square or rectangular size. These smaller blocks (which may also be referred to as sub-blocks) may be further partitioned into even smaller partitions. This is also referred to as tree-partitioning or hierarchical tree-partitioning, wherein a root block, e.g., at root tree-level 0 (hierarchy-level 0, depth 0), may be recursively partitioned, e.g., partitioned into two or more blocks of a next lower tree-level, e.g., nodes at tree-level 1 (hierarchy-level 1, depth 1), wherein these blocks may be again partitioned into two or more blocks of a next lower level, e.g., tree-level 2 (hierarchy-level 2, depth 2), etc. until the partitioning is terminated, e.g., because a termination criterion is fulfilled, e.g., a maximum tree depth or minimum block size is reached. Blocks which are not further partitioned are also referred to as leaf-blocks or leaf nodes of the tree. A tree using partitioning into two partitions is referred to as binary-tree (BT), a tree using partitioning into three partitions is referred to as ternary-tree (TT), and a tree using partitioning into four partitions is referred to as quad-tree (QT).

As mentioned before, the term “block” as used herein may be a portion, in particular a square or rectangular portion, of a picture. With reference, for example, to HEVC and VVC, the block may be or correspond to a coding tree unit (CTU), a coding unit (CU), prediction unit

(PU), and transform unit (TU) and/or to the corresponding blocks, e.g., a coding tree block (CTB), a coding block (CB), a transform block (TB) or prediction block (PB).

For example, a coding tree unit (CTU) may be or comprise a CTB of luma samples, two corresponding CTBs of chroma samples of a picture that has three sample arrays, or a CTB of

5 samples of a monochrome picture or a picture that is coded using three separate colour planes and syntax structures used to code the samples. Correspondingly, a coding tree block (CTB) may be an $N \times N$ block of samples for some value of N such that the division of a component into CTBs is a partitioning. A coding unit (CU) may be or comprise a coding block of luma samples, two corresponding coding blocks of chroma samples of a picture that has three
10 sample arrays, or a coding block of samples of a monochrome picture or a picture that is coded using three separate colour planes and syntax structures used to code the samples. Correspondingly a coding block (CB) may be an $M \times N$ block of samples for some values of M and N such that the division of a CTB into coding blocks is a partitioning.

In embodiments, e.g., according to HEVC, a coding tree unit (CTU) may be split into CUs by

15 using a quad-tree structure denoted as coding tree. The decision whether to code a picture area using inter-picture (temporal) or intra-picture (spatial) prediction is made at the CU level.

Each CU can be further split into one, two or four PUs according to the PU splitting type.

Inside one PU, the same prediction process is applied and the relevant information is transmitted to the decoder on a PU basis. After obtaining the residual block by applying the

20 prediction process based on the PU splitting type, a CU can be partitioned into transform units (TUs) according to another quadtree structure similar to the coding tree for the CU.

In embodiments, e.g., according to the latest video coding standard currently in development, which is referred to as Versatile Video Coding (VVC), Quad-tree and binary tree (QTBT)

partitioning is used to partition a coding block. In the QTBT block structure, a CU can have
25 either a square or rectangular shape. For example, a coding tree unit (CTU) is first partitioned by a quadtree structure. The quadtree leaf nodes are further partitioned by a binary tree or

ternary (or triple) tree structure. The partitioning tree leaf nodes are called coding units (CUs), and that segmentation is used for prediction and transform processing without any further partitioning. This means that the CU, PU and TU have the same block size in the QTBT coding block structure. In parallel, multiple partition, for example, triple tree partition was
5 also proposed to be used together with the QTBT block structure.

In one example, the mode selection unit 260 of video encoder 20 may be configured to perform any combination of the partitioning techniques described herein.

As described above, the video encoder 20 is configured to determine or select the best or an optimum prediction mode from a set of (pre-determined) prediction modes. The set of
10 prediction modes may comprise, e.g., intra-prediction modes and/or inter-prediction modes.

Intra-Prediction

The set of intra-prediction modes may comprise 35 different intra-prediction modes, e.g., non-directional modes like DC (or mean) mode and planar mode, or directional modes, e.g., as defined in HEVC, or may comprise 67 different intra-prediction modes, e.g.,
15 non-directional modes like DC (or mean) mode and planar mode, or directional modes, e.g., as defined for VVC.

The intra-prediction unit 254 is configured to use reconstructed samples of neighboring blocks of the same current picture to generate an intra-prediction block 265 according to an intra-prediction mode of the set of intra-prediction modes.

20 The intra prediction unit 254 (or in general the mode selection unit 260) is further configured to output intra-prediction parameters (or in general information indicative of the selected intra prediction mode for the block) to the entropy encoding unit 270 in form of syntax elements 266 for inclusion into the encoded picture data 21, so that, e.g., the video decoder 30 may receive and use the prediction parameters for decoding.

Inter-Prediction

The set of (or possible) inter-prediction modes depends on the available reference pictures (i.e. previous at least partially decoded pictures, e.g., stored in DBP 230) and other inter-prediction parameters, e.g., whether the whole reference picture or only a part, e.g., a search window area around the area of the current block, of the reference picture is used for searching for a best matching reference block, and/or e.g., whether pixel interpolation is applied, e.g., half/semi-pel and/or quarter-pel interpolation, or not.

Additional to the above prediction modes, skip mode and/or direct mode may be applied.

The inter prediction unit 244 may include a motion estimation (ME) unit and a motion compensation (MC) unit (both not shown in Fig.2). The motion estimation unit may be configured to receive or obtain the picture block 203 (current picture block 203 of the current picture 17) and a decoded picture 231, or at least one or a plurality of previously reconstructed blocks, e.g., reconstructed blocks of one or a plurality of other/different previously decoded pictures 231, for motion estimation. E.g., a video sequence may comprise the current picture and the previously decoded pictures 231, or in other words, the current picture and the previously decoded pictures 231 may be part of or form a sequence of pictures forming a video sequence.

The encoder 20 may, e.g., be configured to select a reference block from a plurality of reference blocks of the same or different pictures of the plurality of other pictures and provide a reference picture (or reference picture index) and/or an offset (spatial offset) between the position (x, y coordinates) of the reference block and the position of the current block as inter prediction parameters to the motion estimation unit. This offset is also called motion vector (MV).

The motion compensation unit is configured to obtain, e.g., receive, an inter prediction parameter and to perform inter prediction based on or using the inter prediction parameter to obtain an inter prediction block 265. Motion compensation, performed by the motion

compensation unit, may involve fetching or generating the prediction block based on the motion/block vector determined by motion estimation, possibly performing interpolations to sub-pixel precision. Interpolation filtering may generate additional pixel samples from known pixel samples, thus potentially increasing the number of candidate prediction blocks that may
5 be used to code a picture block. Upon receiving the motion vector for the PU of the current picture block, the motion compensation unit may locate the prediction block to which the motion vector points in one of the reference picture lists.

Motion compensation unit may also generate syntax elements associated with the blocks and the video slice for use by video decoder 30 in decoding the picture blocks of the video slice.

10 Entropy Coding

The entropy encoding unit 270 is configured to apply, for example, an entropy encoding algorithm or scheme (e.g., a variable length coding (VLC) scheme, an context adaptive VLC scheme (CAVLC), an arithmetic coding scheme, a binarization, a context adaptive binary arithmetic coding (CABAC), syntax-based context-adaptive binary arithmetic coding
15 (SBAC), probability interval partitioning entropy (PIPE) coding or another entropy encoding methodology or technique) or bypass (no compression) on the quantized coefficients 209, inter prediction parameters, intra prediction parameters, loop filter parameters and/or other syntax elements to obtain encoded picture data 21 which can be output via the output 272, e.g., in the form of an encoded bitstream 21, so that, e.g., the video decoder 30 may receive
20 and use the parameters for decoding, . The encoded bitstream 21 may be transmitted to video decoder 30, or stored in a memory for later transmission or retrieval by video decoder 30.

Other structural variations of the video encoder 20 can be used to encode the video stream. For example, a non-transform based encoder 20 can quantize the residual signal directly without the transform processing unit 206 for certain blocks or frames. In another
25 implementation, an encoder 20 can have the quantization unit 208 and the inverse quantization unit 210 combined into a single unit.

Decoder and Decoding Method

Fig. 3 shows an example of a video decoder 30 that is configured to implement the techniques of this present application. The video decoder 30 is configured to receive encoded picture data 21 (e.g., encoded bitstream 21), e.g., encoded by encoder 20, to obtain a decoded picture 331. The encoded picture data or bitstream comprises information for decoding the encoded picture data, e.g., data that represents picture blocks of an encoded video slice and associated syntax elements.

In the example of Fig. 3, the decoder 30 comprises an entropy decoding unit 304, an inverse quantization unit 310, an inverse transform processing unit 312, a reconstruction unit 314 (e.g., a summer 314), a loop filter 320, a decoded picture buffer (DPB) 330, an inter prediction unit 344 and an intra prediction unit 354. Inter prediction unit 344 may be or include a motion compensation unit. Video decoder 30 may, in some examples, perform a decoding pass generally reciprocal to the encoding pass described with respect to video encoder 100 from Fig. 2.

As explained with regard to the encoder 20, the inverse quantization unit 210, the inverse transform processing unit 212, the reconstruction unit 214 the loop filter 220, the decoded picture buffer (DPB) 230, the inter prediction unit 344 and the intra prediction unit 354 are also referred to as forming the “built-in decoder” of video encoder 20. Accordingly, the inverse quantization unit 310 may be identical in function to the inverse quantization unit 110, the inverse transform processing unit 312 may be identical in function to the inverse transform processing unit 212, the reconstruction unit 314 may be identical in function to reconstruction unit 214, the loop filter 320 may be identical in function to the loop filter 220, and the decoded picture buffer 330 may be identical in function to the decoded picture buffer 230. Therefore, the explanations provided for the respective units and functions of the video 20 encoder apply correspondingly to the respective units and functions of the video decoder 30.

Entropy Decoding

The entropy decoding unit 304 is configured to parse the bitstream 21 (or in general encoded picture data 21) and perform, for example, entropy decoding to the encoded picture data 21 to obtain, e.g., quantized coefficients 309 and/or decoded coding parameters (not shown in Fig.

- 5 3), e.g., any or all of inter prediction parameters (e.g., reference picture index and motion vector), intra prediction parameter (e.g., intra prediction mode or index), transform parameters, quantization parameters, loop filter parameters, and/or other syntax elements.

Entropy decoding unit 304 maybe configured to apply the decoding algorithms or schemes corresponding to the encoding schemes as described with regard to the entropy encoding unit

- 10 270 of the encoder 20. Entropy decoding unit 304 may be further configured to provide inter prediction parameters, intra prediction parameter and/or other syntax elements to the mode selection unit 360 and other parameters to other units of the decoder 30. Video decoder 30 may receive the syntax elements at the video slice level and/or the video block level.

Inverse Quantization

- 15 The inverse quantization unit 310 may be configured to receive quantization parameters (QP) (or in general information related to the inverse quantization) and quantized coefficients from the encoded picture data 21 (e.g., by parsing and/or decoding, e.g., by entropy decoding unit 304) and to apply based on the quantization parameters an inverse quantization on the decoded quantized coefficients 309 to obtain dequantized coefficients 311, which may also
- 20 be referred to as transform coefficients 311. The inverse quantization process may include use of a quantization parameter determined by video encoder 20 for each video block in the video slice to determine a degree of quantization and, likewise, a degree of inverse quantization that should be applied.

Inverse Transform

- 25 Inverse transform processing unit 312 may be configured to receive dequantized coefficients 311, also referred to as transform coefficients 311, and to apply a transform to the

dequantized coefficients 311 in order to obtain reconstructed residual blocks 213 in the sample domain. The reconstructed residual blocks 213 may also be referred to as transform blocks 313. The transform may be an inverse transform, e.g., an inverse DCT, an inverse DST, an inverse integer transform, or a conceptually similar inverse transform process. The
5 inverse transform processing unit 312 may be further configured to receive transform parameters or corresponding information from the encoded picture data 21 (e.g., by parsing and/or decoding, e.g., by entropy decoding unit 304) to determine the transform to be applied to the dequantized coefficients 311.

Reconstruction

10 The reconstruction unit 314 (e.g., adder or summer 314) may be configured to add the reconstructed residual block 313, to the prediction block 365 to obtain a reconstructed block 315 in the sample domain, e.g., by adding the sample values of the reconstructed residual block 313 and the sample values of the prediction block 365.

Filtering

15 The loop filter unit 320 (either in the coding loop or after the coding loop) is configured to filter the reconstructed block 315 to obtain a filtered block 321, e.g., to smooth pixel transitions, or otherwise improve the video quality. The loop filter unit 320 may comprise one or more loop filters such as a de-blocking filter, a sample-adaptive offset (SAO) filter or one or more other filters, e.g., a bilateral filter, an adaptive loop filter (ALF), a sharpening, a
20 smoothing filters or a collaborative filters, or any combination thereof. Although the loop filter unit 320 is shown in Fig. 3 as being an in loop filter, in other configurations, the loop filter unit 320 may be implemented as a post loop filter.

Decoded Picture Buffer

The decoded video blocks 321 of a picture are then stored in decoded picture buffer 330,
25 which stores the decoded pictures 331 as reference pictures for subsequent motion compensation for other pictures and/or for output respectively display.

The decoder 30 is configured to output the decoded picture 311, e.g., via output 312, for presentation or viewing to a user.

Prediction

The inter prediction unit 344 may be identical to the inter prediction unit 244 (in particular to
5 the motion compensation unit) and the intra prediction unit 354 may be identical to the inter prediction unit 254 in function, and performs split or partitioning decisions and prediction based on the partitioning and/or prediction parameters or respective information received from the encoded picture data 21 (e.g., by parsing and/or decoding, e.g., by entropy decoding unit 304). Mode selection unit 360 may be configured to perform the prediction (intra or inter
10 prediction) per block based on reconstructed pictures, blocks or respective samples (filtered or unfiltered) to obtain the prediction block 365.

When the video slice is coded as an intra coded (I) slice, intra prediction unit 354 of mode selection unit 360 is configured to generate prediction block 365 for a picture block of the current video slice based on a signaled intra prediction mode and data from previously
15 decoded blocks of the current picture. When the video picture is coded as an inter coded (i.e., B, or P) slice, inter prediction unit 344 (e.g., motion compensation unit) of mode selection unit 360 is configured to produce prediction blocks 365 for a video block of the current video slice based on the motion vectors and other syntax elements received from entropy decoding unit 304. For inter prediction, the prediction blocks may be produced from one of the
20 reference pictures within one of the reference picture lists. Video decoder 30 may construct the reference frame lists, List 0 and List 1, using default construction techniques based on reference pictures stored in DPB 330.

Mode selection unit 360 is configured to determine the prediction information for a video block of the current video slice by parsing the motion vectors and other syntax elements, and
25 uses the prediction information to produce the prediction blocks for the current video block being decoded. For example, the mode selection unit 360 uses some of the received syntax

elements to determine a prediction mode (e.g., intra or inter prediction) used to code the video blocks of the video slice, an inter prediction slice type (e.g., B slice, P slice, or GPB slice), construction information for one or more of the reference picture lists for the slice, motion vectors for each inter encoded video block of the slice, inter prediction status for each
5 inter coded video block of the slice, and other information to decode the video blocks in the current video slice.

Other variations of the video decoder 30 can be used to decode the encoded picture data 21. For example, the decoder 30 can produce the output video stream without the loop filtering unit 320. For example, a non-transform based decoder 30 can inverse-quantize the residual
10 signal directly without the inverse-transform processing unit 312 for certain blocks or frames. In another implementation, the video decoder 30 can have the inverse-quantization unit 310 and the inverse-transform processing unit 312 combined into a single unit.

It should be understood that, in the encoder 20 and the decoder 30, a processing result of a current step may be further processed and then output to the next step. For example, after
15 interpolation filtering, motion vector derivation or loop filtering, a further operation, such as Clip or shift, may be performed on the processing result of the interpolation filtering, motion vector derivation or loop filtering.

It should be noted that further operations may be applied to the derived motion vectors of current block (including but not limit to control point motion vectors of affine mode,
20 sub-block motion vectors in affine, planar, ATMVP modes, temporal motion vectors, and so on). For example, the value of motion vector is constrained to a predefined range according to its representing bit. If the representing bit of motion vector is bitDepth, then the range is $-2^{(bitDepth-1)} \sim 2^{(bitDepth-1)}-1$, where “^” means exponentiation. For example, if bitDepth is set equal to 16, the range is -32768 ~ 32767; if bitDepth is set equal to 18, the
25 range is -131072~131071. For example, the value of the derived motion vector (e.g., the MVs of four 4x4 sub-blocks within one 8x8 block) is constrained such that the max difference

between integer parts of the four 4x4 sub-block MVs is no more than N pixels, such as no more than 1 pixel. Here provides two methods for constraining the motion vector according to the bitDepth.

Method 1: remove the overflow MSB (most significant bit) by flowing operations

$$5 \quad ux = (mvx + 2^{bitDepth}) \% 2^{bitDepth} \quad (1)$$

$$mvx = (ux \geq 2^{bitDepth-1}) ? (ux - 2^{bitDepth}) : ux \quad (2)$$

$$uy = (mvy + 2^{bitDepth}) \% 2^{bitDepth} \quad (3)$$

$$mvy = (uy \geq 2^{bitDepth-1}) ? (uy - 2^{bitDepth}) : uy \quad (4)$$

where mvx is a horizontal component of a motion vector of an image block or a sub-block,

10 mvy is a vertical component of a motion vector of an image block or a sub-block, and ux and uy indicates an intermediate value.

For example, if the value of mvx is -32769, after applying formula (1) and (2), the resulting value is 32767. In computer system, decimal numbers are stored as two's complement. The two's complement of -32769 is 1,0111,1111,1111,1111 (17 bits), then the MSB is discarded,

15 so the resulting two's complement is 0111,1111,1111,1111 (decimal number is 32767), which is same as the output by applying formula (1) and (2).

$$ux = (mvp_x + mvd_x + 2^{bitDepth}) \% 2^{bitDepth} \quad (5)$$

$$mvx = (ux \geq 2^{bitDepth-1}) ? (ux - 2^{bitDepth}) : ux \quad (6)$$

$$uy = (mvp_y + mvd_y + 2^{bitDepth}) \% 2^{bitDepth} \quad (7)$$

$$20 \quad mvy = (uy \geq 2^{bitDepth-1}) ? (uy - 2^{bitDepth}) : uy \quad (8)$$

The operations may be applied during the sum of mvp and mvd, as shown in formula (5) to (8).

Method 2: remove the overflow MSB by clipping the value

$$vx = \text{Clip3}(-2^{\text{bitDepth}-1}, 2^{\text{bitDepth}-1} - 1, vx)$$

$$vy = \text{Clip3}(-2^{\text{bitDepth}-1}, 2^{\text{bitDepth}-1} - 1, vy)$$

where vx is a horizontal component of a motion vector of an image block or a sub-block, vy is a vertical component of a motion vector of an image block or a sub-block; x, y and z respectively correspond to three input value of the MV clipping process, and the definition of function Clip3 is as follow:

$$\text{Clip3}(x, y, z) = \begin{cases} x & ; & z < x \\ y & ; & z > y \\ z & ; & \text{otherwise} \end{cases}$$

Fig. 4 is a schematic diagram of a video coding device 400 according to an embodiment of the disclosure. The video coding device 400 is suitable for implementing the disclosed
10 embodiments as described herein. In an embodiment, the video coding device 400 may be a decoder such as video decoder 30 of Fig. 1A or an encoder such as video encoder 20 of Fig. 1A.

The video coding device 400 comprises ingress ports 410 (or input ports 410) and receiver units (Rx) 420 for receiving data; a processor, logic unit, or central processing unit (CPU) 430 to process the data; transmitter units (Tx) 440 and egress ports 450 (or output ports 450)
15 for transmitting the data; and a memory 460 for storing the data. The video coding device 400 may also comprise optical-to-electrical (OE) components and electrical-to-optical (EO) components coupled to the ingress ports 410, the receiver units 420, the transmitter units 440, and the egress ports 450 for egress or ingress of optical or electrical signals.

20 The processor 430 is implemented by hardware and software. The processor 430 may be implemented as one or more CPU chips, cores (e.g., as a multi-core processor), FPGAs, ASICs, and DSPs. The processor 430 is in communication with the ingress ports 410, receiver units 420, transmitter units 440, egress ports 450, and memory 460. The processor 430 comprises a coding module 470. The coding module 470 implements the disclosed
25 embodiments described above. For instance, the coding module 470 implements, processes,

prepares, or provides the various coding operations. The inclusion of the coding module 470 therefore provides a substantial improvement to the functionality of the video coding device 400 and effects a transformation of the video coding device 400 to a different state.

Alternatively, the coding module 470 is implemented as instructions stored in the memory 460 and executed by the processor 430.

The memory 460 may comprise one or more disks, tape drives, and solid-state drives and may be used as an over-flow data storage device, to store programs when such programs are selected for execution, and to store instructions and data that are read during program execution. The memory 460 may be, for example, volatile and/or non-volatile and may be a read-only memory (ROM), random access memory (RAM), ternary content-addressable memory (TCAM), and/or static random-access memory (SRAM).

Fig. 5 is a simplified block diagram of an apparatus 500 that may be used as either or both of the source device 12 and the destination device 14 from Fig. 1A according to an exemplary embodiment.

A processor 502 in the apparatus 500 can be a central processing unit. Alternatively, the processor 502 can be any other type of device, or multiple devices, capable of manipulating or processing information now-existing or hereafter developed. Although the disclosed implementations can be practiced with a single processor as shown, e.g., the processor 502, advantages in speed and efficiency can be achieved using more than one processor.

A memory 504 in the apparatus 500 can be a read only memory (ROM) device or a random access memory (RAM) device in an implementation. Any other suitable type of storage device can be used as the memory 504. The memory 504 can include code and data 506 that is accessed by the processor 502 using a bus 512. The memory 504 can further include an operating system 508 and application programs 510, the application programs 510 including at least one program that permits the processor 502 to perform the methods described here.

For example, the application programs 510 can include applications 1 through N, which further include a video coding application that performs the methods described here.

The apparatus 500 can also include one or more output devices, such as a display 518. The display 518 may be, in one example, a touch sensitive display that combines a display with a touch sensitive element that is operable to sense touch inputs. The display 518 can be coupled to the processor 502 via the bus 512.

Although depicted here as a single bus, the bus 512 of the apparatus 500 can be composed of multiple buses. Further, the secondary storage 514 can be directly coupled to the other components of the apparatus 500 or can be accessed via a network and can comprise a single integrated unit such as a memory card or multiple units such as multiple memory cards. The apparatus 500 can thus be implemented in a wide variety of configurations.

In-Loop filter background

There are in total three types of in loop filters in VTM3. Besides deblocking filter and SAO (the two loop filters in HEVC), adaptive loop filter (ALF) are applied in the VTM3. The order of the filtering process in the VTM3 is the deblocking filter, SAO and ALF.

ALF background

In the VTM3, an adaptive loop filter (ALF) with block-based filter adaption is applied. For luma component, one among 25 filters is selected for each 4×4 block, based on the direction and activity of local gradients.

Filter shape:

In the JEM, two diamond filter shapes (as shown in Fig. 6) are used for luma component. The 7×7 diamond shape is applied for luma component and the 5×5 diamond shape applied for chroma component.

Block classification:

For luma component, each 4×4 block is categorized into one out of 25 classes. The classification index C is derived based on directionality D and a quantized value of activity $\hat{A}$, as follows:

$$C = 5D + \hat{A} \quad (9)$$

To calculate D and $\hat{A}$, gradients of the horizontal, vertical and two diagonal direction are first calculated using 1-D Laplacian:

$$g_v = \sum_{k=i-2}^{i+3} \sum_{l=j-2}^{j+3} V_{k,l}, \quad V_{k,l} = |2R(k,l) - R(k,l-1) - R(k,l+1)| \quad (10)$$

$$g_h = \sum_{k=i-2}^{i+3} \sum_{l=j-2}^{j+3} H_{k,l}, \quad H_{k,l} = |2R(k,l) - R(k-1,l) - R(k+1,l)| \quad (11)$$

$$g_{d1} = \sum_{k=i-2}^{i+3} \sum_{l=j-3}^{j+3} D1_{k,l}, \quad D1_{k,l} = |2R(k,l) - R(k-1,l-1) - R(k+1,l+1)| \quad (12)$$

$$g_{d2} = \sum_{k=i-2}^{i+3} \sum_{l=j-2}^{j+3} D2_{k,l}, \quad D2_{k,l} = |2R(k,l) - R(k-1,l+1) - R(k+1,l-1)| \quad (13)$$

Where indices i and j refer to the coordinates of the upper left sample within the 4×4 block and $R(i,j)$ indicates a reconstructed sample at coordinate (i,j) .

To reduce the complexity of block classification, the subsampled 1-D Laplacian calculation is applied. As shown in Fig.7, the same subsampled positions are used for gradient calculation of all directions.

Then maximum and minimum values of the gradients of horizontal and vertical directions are set as:

$$g_{h,v}^{max} = \max(g_h, g_v), \quad g_{h,v}^{min} = \min(g_h, g_v) \quad (14)$$

The maximum and minimum values of the gradient of two diagonal directions are set as:

$$g_{d0,d1}^{max} = \max(g_{d0}, g_{d1}), \quad g_{d0,d1}^{min} = \min(g_{d0}, g_{d1}) \quad (15)$$

To derive the value of the directionality D , these values are compared against each other and with two thresholds t_1 and t_2 :

Step 1. If both $g_{h,v}^{max} \leq t_1 \cdot g_{h,v}^{min}$ and $g_{d0,d1}^{max} \leq t_1 \cdot g_{d0,d1}^{min}$ are true, D is set to 0.

Step 2. If $g_{h,v}^{max} / g_{h,v}^{min} > g_{d0,d1}^{max} / g_{d0,d1}^{min}$, continue to Step 3; otherwise continue to Step 4.

Step 3. If $g_{h,v}^{max} > t_2 \cdot g_{h,v}^{min}$, D is set to 2; otherwise D is set to 1.

Step 4. If $g_{d0,d1}^{max} > t_2 \cdot g_{d0,d1}^{min}$, D is set to 4; otherwise D is set to 3.

The activity value A is calculated as:

$$A = \sum_{k=i-2}^{i+3} \sum_{l=j-2}^{j+3} (V_{k,l} + H_{k,l}) \quad (16)$$

- 5 A is further quantized to the range of 0 to 4, inclusively, and the quantized value is denoted as $\hat{A}$.

For chroma components in a picture, no classification method is applied, a single set of ALF coefficients is applied for each chroma component.

Geometric transformations of filter coefficients

- 10 Before filtering each 4×4 luma block, geometric transformations such as rotation or diagonal and vertical flipping are applied to the filter coefficients $f(k, l)$, depending on gradient values calculated for that block. This is equivalent to applying these transformations to the samples in the filter support region. The idea is to make different blocks to which ALF is applied more similar by aligning their directionality.
- 15 Three geometric transformations, including diagonal, vertical flip and rotation are introduced:

$$\text{Diagonal: } f_D(k, l) = f(l, k), \quad (17)$$

$$\text{Vertical flip: } f_V(k, l) = f(k, K - l - 1) \quad (18)$$

$$\text{Rotation: } f_R(k, l) = f(K - l - 1, k) \quad (19)$$

- where K is the size of the filter and $0 \leq k, l \leq K - 1$ are coefficients coordinates, such that
- 20 location $(0,0)$ is at the upper left corner and location $(K - 1, K - 1)$ is at the lower right corner. The transformations are applied to the filter coefficients $f(k, l)$ depending on gradient values calculated for that block. The relationship between the transformation and the four gradients of the four directions are summarized in the following table.

Table 1 Mapping of the gradient calculated for one block and the transformations

Gradient values	Transformation
$g_{d2} < g_{d1}$ and $g_h < g_v$	No transformation
$g_{d2} < g_{d1}$ and $g_v < g_h$	Diagonal
$g_{d1} < g_{d2}$ and $g_h < g_v$	Vertical flip
$g_{d1} < g_{d2}$ and $g_v < g_h$	Rotation

Filter parameters signalling

In the VTM3, ALF filter parameters are signalled in the slice header. Up to 25 sets of luma filter coefficients could be signalled. To reduce bits overhead, filter coefficients of different classification can be merged.

The filtering process can be controlled at CTB level. A flag is signalled to indicate whether ALF is applied to a luma CTB. For each chroma CTB, a flag might be signalled to indicate whether ALF is applied to a chroma CTB depends on the value of `alf_chroma_ctb_present_flag`.

The filter coefficients are quantized with norm equal to 128. To further restrict the multiplication complexity, a bitstream conformance is applied that the coefficient value of the central position shall be in the range of 0 to 2^8 and the coefficient values of the remaining positions shall be in the range of -2^7 to $2^7 - 1$, inclusive.

Filtering process

At decoder side, when ALF is enabled for a CTB, each sample $R(i, j)$ within the CU is filtered, resulting in sample value $R'(i, j)$ as shown below, where L denotes filter length, $f_{m,n}$ represents filter coefficient, and $f(k, l)$ denotes the decoded filter coefficients.

$$R'(i, j) = \left(\sum_{k=-\frac{L}{2}}^{\frac{L}{2}} \sum_{l=-\frac{L}{2}}^{\frac{L}{2}} f(k, l) \times R(i + k, j + l) + 64 \right) \gg 7$$

ALF syntax specification according to the VVC specification

Adaptive loop filter process

1.1 General

Inputs of this process are the reconstructed picture sample arrays prior to adaptive loop filter

5 recPictureL, recPictureCb and recPictureCr.

Outputs of this process are the modified reconstructed picture sample arrays after adaptive loop filter alfPictureL, alfPictureCb and alfPictureCr.

The sample values in the modified reconstructed picture sample arrays after adaptive loop filter alfPictureL, alfPictureCb and alfPictureCr, are initially set equal to the sample values in
10 the reconstructed picture sample arrays prior to adaptive loop filter recPictureL, recPictureCb and recPictureCr, respectively.

When a value of tile_group_alf_enabled_flag is equal to 1, for every coding tree unit with luma coding tree block location (rx, ry), where $rx = 0..PicWidthInCtbs - 1$ and $ry = 0..PicHeightInCtbs - 1$, the following process are applied:

15 When a value of alf_ctb_flag[0][rx][ry] is equal to 1, the coding tree block filtering process for luma samples as specified in clause 1.2 is invoked with recPictureL, alfPictureL, and the luma coding tree block location (xCtb, yCtb) set equal to (rx << CtbLog2SizeY, ry << CtbLog2SizeY) as inputs, and the output is the modified filtered picture alfPictureL.

20 When a value of alf_ctb_flag[1][rx][ry] is equal to 1, the coding tree block filtering process for chroma samples as specified in clause 1.1 is invoked with recPicture set equal to recPictureCb, alfPicture set equal to alfPictureCb, and the chroma coding tree block location (xCtbC, yCtbC) set equal to (rx << (CtbLog2SizeY - 1), ry << (CtbLog2SizeY - 1)) as inputs, and the output is the modified filtered picture alfPictureCb.

When a value of $\text{alf_ctb_flag}[2][rx][ry]$ is equal to 1, the coding tree block filtering process for chroma samples as specified in clause 1.4 is invoked with recPicture set equal to recPictureCr , alfPicture set equal to alfPictureCr , and the chroma coding tree block location $(x_{\text{CtbC}}, y_{\text{CtbC}})$ set equal to $(rx \ll (\text{CtbLog2SizeY} - 1), ry \ll (\text{CtbLog2SizeY} - 1))$

5 as inputs, and the output is the modified filtered picture alfPictureCr .

1.2 Coding tree block filtering process for luma samples

Inputs of this process are:

a reconstructed luma picture sample array recPictureL prior to the adaptive loop filtering process,

10 a filtered reconstructed luma picture sample array alfPictureL ,
a luma location $(x_{\text{Ctb}}, y_{\text{Ctb}})$ specifying the top-left sample of the current luma coding tree block relative to the top left sample of the current picture.

Output of this process is the modified filtered reconstructed luma picture sample array alfPictureL .

15 The derivation process for filter index clause 1.3 is invoked with the location $(x_{\text{Ctb}}, y_{\text{Ctb}})$ and the reconstructed luma picture sample array recPictureL as inputs, and $\text{filtIdx}[x][y]$ and $\text{transposeIdx}[x][y]$ with $x, y = 0..\text{CtbSizeY} - 1$ as outputs.

For the derivation of the filtered reconstructed luma samples $\text{alfPictureL}[x][y]$, each reconstructed luma sample inside the current luma coding tree block $\text{recPictureL}[x][y]$ is

20 filtered as follows with $x, y = 0..\text{CtbSizeY} - 1$:

The array of luma filter coefficients $f[j]$ corresponding to the filter specified by $\text{filtIdx}[x][y]$ is derived as follows with $j = 0..12$:

$$f[j] = \text{AlfCoeffL}[\text{filtIdx}[x][y]][j].$$

The luma filter coefficients filterCoeff are derived depending on $\text{transposeIdx}[x][y]$ as

25 follows:

If $\text{transposeIndex}[x][y] = 1$,

filterCoeff[] = { f[9], f[4], f[10], f[8], f[1], f[5], f[11], f[7], f[3], f[0], f[2], f[6], f[12] }

Otherwise, if transposeIndex[x][y] = 2,

filterCoeff[] = { f[0], f[3], f[2], f[1], f[8], f[7], f[6], f[5], f[4], f[9], f[10], f[11], f[12] }

Otherwise, if transposeIndex[x][y] = 3,

5 filterCoeff[] = { f[9], f[8], f[10], f[4], f[3], f[7], f[11], f[5], f[1], f[0], f[2], f[6], f[12] }

Otherwise,

filterCoeff[] = { f[0], f[1], f[2], f[3], f[4], f[5], f[6], f[7], f[8], f[9], f[10], f[11], f[12] }

The locations (hx, vy) for each of the corresponding luma samples (x, y) inside the given array recPicture of luma samples are derived as follows:

10 hx = Clip3(0, pic_width_in_luma_samples - 1, xCtb + x)

vy = Clip3(0, pic_height_in_luma_samples - 1, yCtb + y)

The variable sum is derived as follows:

sum = filterCoeff[0] * (recPictureL[hx, vy + 3] + recPictureL[hx, vy - 3]) +

filterCoeff[1] * (recPictureL[hx + 1, vy + 2] + recPictureL[hx - 1, vy - 2]) +

15 filterCoeff[2] * (recPictureL[hx, vy + 2] + recPictureL[hx, vy - 2]) +

filterCoeff[3] * (recPictureL[hx - 1, vy + 2] + recPictureL[hx + 1, vy - 2]) +

filterCoeff[4] * (recPictureL[hx + 2, vy + 1] + recPictureL[hx - 2, vy - 1]) +

filterCoeff[5] * (recPictureL[hx + 1, vy + 1] + recPictureL[hx - 1, vy - 1]) +

filterCoeff[6] * (recPictureL[hx, vy + 1] + recPictureL[hx, vy - 1]) +

20 filterCoeff[7] * (recPictureL[hx - 1, vy + 1] + recPictureL[hx + 1, vy - 1]) +

filterCoeff[8] * (recPictureL[hx - 2, vy + 1] + recPictureL[hx + 2, vy - 1]) +

filterCoeff[9] * (recPictureL[hx + 3, vy] + recPictureL[hx - 3, vy]) +

filterCoeff[10] * (recPictureL[hx + 2, vy] + recPictureL[hx - 2, vy]) +

filterCoeff[11] * (recPictureL[hx + 1, vy] + recPictureL[hx - 1, vy]) +

25 filterCoeff[12] * recPictureL[hx, vy]

sum = (sum + 64) >> 7

The modified filtered reconstructed luma picture sample $\text{alfPictureL}[x_{\text{Ctb}} + x][y_{\text{Ctb}} + y]$ is derived as follows:

$$\text{alfPictureL}[x_{\text{Ctb}} + x][y_{\text{Ctb}} + y] = \text{Clip3}(0, (1 \ll \text{BitDepthY}) - 1, \text{sum}).$$

1.3 Derivation process for ALF transpose and filter index for luma samples

5 Inputs of this process are:

a luma location $(x_{\text{Ctb}}, y_{\text{Ctb}})$ specifying the top-left sample of the current luma coding tree block relative to the top left sample of the current picture,
a reconstructed luma picture sample array recPictureL prior to the adaptive loop filtering process.

10 Outputs of this process are

the classification filter index array $\text{filtIdx}[x][y]$ with $x, y = 0.. \text{CtbSizeY} - 1$,
the transpose index array $\text{transposeIdx}[x][y]$ with $x, y = 0.. \text{CtbSizeY} - 1$.

The locations (h_x, v_y) for each of the corresponding luma samples (x, y) inside the given array recPicture of luma samples are derived as follows:

15 $h_x = \text{Clip3}(0, \text{pic_width_in_luma_samples} - 1, x)$

$v_y = \text{Clip3}(0, \text{pic_height_in_luma_samples} - 1, y)$

The classification filter index array filtIdx and the transpose index array transposeIdx are derived by the following ordered steps:

The variables $\text{filtH}[x][y]$, $\text{filtV}[x][y]$, $\text{filtD0}[x][y]$ and $\text{filtD1}[x][y]$ with

20 $x, y = -2.. \text{CtbSizeY} + 1$ are derived as follows:

If both x and y are even numbers or both x and y are uneven numbers, the following applies:

$$\text{filtH}[x][y] = \text{Abs}((\text{recPicture}[h_x \text{Ctb} + x, v_y \text{Ctb} + y] \ll 1) - \text{recPicture}[h_x \text{Ctb} + x - 1, v_y \text{Ctb} + y] - \text{recPicture}[h_x \text{Ctb} + x + 1, v_y \text{Ctb} + y])$$

$$\text{filtV}[x][y] = \text{Abs}((\text{recPicture}[h_x \text{Ctb} + x, v_y \text{Ctb} + y] \ll 1) - \text{recPicture}[h_x \text{Ctb} + x, v_y \text{Ctb} + y - 1] - \text{recPicture}[h_x \text{Ctb} + x, v_y \text{Ctb} + y + 1])$$

25 $y - 1] - \text{recPicture}[h_x \text{Ctb} + x, v_y \text{Ctb} + y + 1]$

$\text{filtD0}[x][y] = \text{Abs}((\text{recPicture}[\text{hxCtb}+x, \text{vyCtb}+y] \ll 1) - \text{recPicture}[\text{hxCtb}+x-1, \text{vyCtb}+y-1] - \text{recPicture}[\text{hxCtb}+x+1, \text{vyCtb}+y+1])$

$\text{filtD1}[x][y] = \text{Abs}((\text{recPicture}[\text{hxCtb}+x, \text{vyCtb}+y] \ll 1) - \text{recPicture}[\text{hxCtb}+x+1, \text{vyCtb}+y-1] - \text{recPicture}[\text{hxCtb}+x-1, \text{vyCtb}+y+1])$

- 5 Otherwise, $\text{filtH}[x][y]$, $\text{filtV}[x][y]$, $\text{filtD0}[x][y]$ and $\text{filtD1}[x][y]$ are set equal to 0.

The variables $\text{varTempH1}[x][y]$, $\text{varTempV1}[x][y]$, $\text{varTempD01}[x][y]$, $\text{varTempD11}[x][y]$ and $\text{varTemp}[x][y]$ with $x, y = 0..(\text{CtbSizeY} - 1) \gg 2$ are derived as follows:

$\text{sumH}[x][y] = \sum_i \sum_j \text{filtH}[(x \ll 2) + i][(y \ll 2) + j]$ with $i, j = -2..5$

- 10 $\text{sumV}[x][y] = \sum_i \sum_j \text{filtV}[(x \ll 2) + i][(y \ll 2) + j]$ with $i, j = -2..5$

$\text{sumD0}[x][y] = \sum_i \sum_j \text{filtD0}[(x \ll 2) + i][(y \ll 2) + j]$ with $i, j = -2..5$

$\text{sumD1}[x][y] = \sum_i \sum_j \text{filtD1}[(x \ll 2) + i][(y \ll 2) + j]$ with $i, j = -2..5$

$\text{sumOfHV}[x][y] = \text{sumH}[x][y] + \text{sumV}[x][y]$

The variables $\text{dir1}[x][y]$, $\text{dir2}[x][y]$ and $\text{dirS}[x][y]$ with $x, y = 0..\text{CtbSizeY} - 1$ are

- 15 derived as follows:

The variables hv1 , hv0 and dirHV are derived as follows:

If $\text{sumV}[x \gg 2][y \gg 2]$ is greater than $\text{sumH}[x \gg 2][y \gg 2]$, the following applies:

$\text{hv1} = \text{sumV}[x \gg 2][y \gg 2]$

$\text{hv0} = \text{sumH}[x \gg 2][y \gg 2]$

- 20 $\text{dirHV} = 1$

Otherwise, the following applies:

$\text{hv1} = \text{sumH}[x \gg 2][y \gg 2]$

$\text{hv0} = \text{sumV}[x \gg 2][y \gg 2]$

$\text{dirHV} = 3$

- 25 The variables d1 , d0 and dirD are derived as follows:

If $\text{sumD0}[x \gg 2][y \gg 2]$ is greater than $\text{sumD1}[x \gg 2][y \gg 2]$, the following applies:

d1 = sumD0[x >> 2][y >> 2]

d0 = sumD1[x >> 2][y >> 2]

dirD = 0

Otherwise, the following applies:

5 d1 = sumD1[x >> 2][y >> 2]

d0 = sumD0[x >> 2][y >> 2]

dirD = 2

The variables hvd1, hvd0, are derived as follows:

hvd1 = (d1 * hv0 > hv1 * d0) ? d1 : hv1

10 hvd0 = (d1 * hv0 > hv1 * d0) ? d0 : hv0

The variables dirS[x][y], dir1[x][y] and dir2[x][y] derived as follows:

dir1[x][y] = (d1 * hv0 > hv1 * d0) ? dirD : dirHV

dir2[x][y] = (d1 * hv0 > hv1 * d0) ? dirHV : dirD

dirS[x][y] = (hvd1 > 2 * hvd0) ? 1 : ((hvd1 * 2 > 9 * hvd0) ? 2 : 0)

15 The variable avgVar[x][y] with x, y = 0..CtbSizeY - 1 is derived as follows:

varTab[] = { 0, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 4 }

avgVar[x][y] = varTab[Clip3(0, 15, (sumOfHV[x >> 2][y >> 2] * 64) >> (3 + BitDepthY))]

The classification filter index array filtIdx[x][y] and the transpose index array

20 transposeIdx[x][y] with x = y = 0..CtbSizeY - 1 are derived as follows:

transposeTable[] = { 0, 1, 0, 2, 2, 3, 1, 3 }

transposeIdx[x][y] = transposeTable[dir1[x][y] * 2 + (dir2[x][y] >> 1)]

filtIdx[x][y] = avgVar[x][y]

When dirS[x][y] is not equal 0, filtIdx[x][y] is modified as follows:

25 filtIdx[x][y] += (((dir1[x][y] & 0x1) << 1) + dirS[x][y]) * 5.

1.4 Coding tree block filtering process for chroma samples

Inputs of this process are:

a reconstructed chroma picture sample array `recPicture` prior to the adaptive loop filtering process,

- 5 a filtered reconstructed chroma picture sample array `alfPicture`,
a chroma location (`xCtbC`, `yCtbC`) specifying the top-left sample of the current chroma coding tree block relative to the top left sample of the current picture.

Output of this process is the modified filtered reconstructed chroma picture sample array `alfPicture`.

- 10 The size of the current chroma coding tree block `ctbSizeC` is derived as follows:

$$ctbSizeC = CtbSizeY / SubWidthC$$

For the derivation of the filtered reconstructed chroma samples `alfPicture[ x ][ y ]`, each reconstructed chroma sample inside the current chroma coding tree block `recPicture[ x ][ y ]` is filtered as follows with $x, y = 0..ctbSizeC - 1$:

- 15 The locations (`hx`, `vy`) for each of the corresponding chroma samples (`x`, `y`) inside the given array `recPicture` of chroma samples are derived as follows:

$$hx = Clip3(0, pic_width_in_luma_samples / SubWidthC - 1, xCtbC + x)$$

$$vy = Clip3(0, pic_height_in_luma_samples / SubHeightC - 1, yCtbC + y)$$

The variable `sum` is derived as follows:

- 20 $sum = AlfCoeffC[0] * (recPicture[hx, vy + 2] + recPicture[hx, vy - 2]) +$
 $AlfCoeffC[1] * (recPicture[hx + 1, vy + 1] + recPicture[hx - 1, vy - 1]) +$
 $AlfCoeffC[2] * (recPicture[hx, vy + 1] + recPicture[hx, vy - 1]) +$
 $AlfCoeffC[3] * (recPicture[hx - 1, vy + 1] + recPicture[hx + 1, vy - 1]) +$
 $AlfCoeffC[4] * (recPicture[hx + 2, vy] + recPicture[hx - 2, vy]) +$
25 $AlfCoeffC[5] * (recPicture[hx + 1, vy] + recPicture[hx - 1, vy]) +$
 $AlfCoeffC[6] * recPicture[hx, vy]$

sum = (sum + 64) >> 7

The modified filtered reconstructed chroma picture sample
alfPicture[xCtbC + x][yCtbC + y] is derived as follows:

alfPicture[xCtbC + x][yCtbC + y] = Clip3(0, (1 << BitDepthC) - 1, sum).

- 5 The current line buffer requirement of three cascaded loop filters (Deblocking, SAO, ALF) of VTM-3.0 for Luma component is derived to be 11.25 lines and for the Chroma component it is derived to be 6.25 lines (see Fig. 8). Accommodating increased line buffer requirements on hardware chips is known to be a concern, as the line buffer needs to be stored on the “on-chip” memory, thereby increasing the hardware chip area.

- 10 To reduce line buffer requirements of VTM-3.0,

ALF block classification is adapted to make sure that the ALF block classification does not need any pixels above or below the virtual boundary (VB). Virtual boundaries (VBs) are upward shifted horizontal CTU boundaries by “N” pixels. For each CTU, SAO and ALF can process samples above the VB before the lower CTU comes, but cannot process samples
15 below the VB until the lower CTU comes, which is mainly caused because of the delay in vertical deblocking filtering.

In VTM-3.0, N can take a positive integer value which is greater than or equal to 4, e.g. N=4 or N=6.

- To reduce line buffer requirements, some examples mainly adapt the ALF block
20 classification in such a way that the subset of samples which are originally used in the block classification are further used.

- A “grid shifting” approach is used, for cases when the ALF block classification window size (P x P) is not an integer multiple of N. In current VTM-3.0, ALF block classification is performed for 4 x 4 blocks. Therefore P is set to 4 for VTM-3.0. Grid shifting approach
25 mainly shifts the ALF block classification window by an integer number of samples, so that a given P x P block classification window would not cross the virtual boundary. Therefore for

the initial samples in the picture which is equivalent to “P- N”, the ALF block classification is done by only using 2 x 2 block window or the first “P-N” lines re-use the block classification decision of the 4 x4 block starting immediately at (“P- N” + 1).

ALF Filtering is adapted to use distorted versions of the filter for both Luma and Chroma

5 components, by either using padding or by adding the un-used filter coefficients to the central coefficient. In such a way additional line buffers are not required for ALF filtering.

Use a modified version of the original ALF block classification such that the block classification will not require any additional line buffers. Also the ALF filtering is modified in such a way that it does not need any additional buffers. The modified version of block

10 classification only uses a subset of samples which are originally used in the original block classification.

As shown in Fig. 8, the Luma line buffer requirements for VVC is 11.25 Luma lines, when the virtual boundary is considered to 4 lines above the CTU boundary.

The explanation of the line buffer requirement is as follows: the deblocking of horizontal

15 edge overlapping with CTU edge cannot be performed as the decisions and filtering require lines K,L,M,M from the first CTU and Lines O,P.. from the bottom CTU. Therefore the deblocking of the horizontal edges overlapping with the CTU boundary is postponed until the lower CTU. Therefore the lines K, L, M, N which are the reconstructed luma samples have to be stored on the line buffer (4 lines). Then the SAO filtering can be performed for lines A till

20 J. The line J can be SAO filtered as deblocking does not change the samples in line K. For SAO filtering of line K, the edge offset classification decision is only stored in the line buffer (which is 0.25 Luma lines). The ALF filtering can only be performed for lines A-F. In an example, the ALF classification is performed for each 4 x 4 block. Each 4 x 4 block classification needs an activity window of size 8 x 8 which in turn needs a 9 x 9 window to

25 compute the 1d Laplacian to determine the gradient.

Therefore, for the block classification of lines G, H, I, J, SAO filtered samples below the Virtual boundary are needed. Also for the lines D, E, F, SAO filtered samples are required for sample classification. Also ALF filtering of Line G needs three SAO filtered lines D, E, F from Above. Therefore the total line buffer requirement is as follows:

- 5 • Lines K-N (Horizontal DF pixels) – 4
- Lines D-J (SAO filtered pixels) – 7
- SAO Edge offset classifier values between line J and line K – 0.25
- Therefore the total number of Luma lines is $7 + 4 + 0.25 = 11.25$

Modified block classification when virtual boundary is 4 lines

- 10 Fig. 9 depicts how the ALF block classification is performed for the lines above and below the virtual boundary.

Solution 1 (Asymmetric truncated ALF block classification)

Fig.9 discloses examples refer to Modified ALF classification to avoid ALF Luma Line Buffers (Asymmetric variant).

- 15 As shown in Fig. 9a and Fig.18a, for the 4 x 4 block immediately above the virtual boundary, only samples which are above the block boundary are used for block classification.

In one implementation, for block classification of 4 x 4 block immediate above VB (4a):

- Only use samples above VB in classification (samples in lines E till I)

For block classification of 4 x 4 block immediate below VB (4b):

- 20 • Only use samples below VB in classification (samples in lines P till L).

In one implementation, for block classification of 4 x 4 block immediate above VB (4a):

Pad line J to line K, L, M.

For block classification of 4 x 4 block immediate below VB (4b):

Pad line K to line J, I, H.

As shown in Fig. 9b, for the block classification of 4 x 4 block immediately below virtual boundary, only the samples which are below the virtual boundary are used.

In Fig. 9a, Fig. 22a (case when 4 x4 block is above VB), the block classification for the block beginning at line G, uses lines D, E, F, F, H, J, J. Since gradient calculation of samples belonging to line J line need one more line below, therefore line J is padded to line K.

In one implementation,

- For 4 x 4 block above VB:
 - Only samples above the VB contribute in the ALF block classification and vice versa
 - For gradient calculation of samples in line J, line K is padded with line J
- For 4 x 4 block below VB:
 - Only samples below the VB contribute in the ALF block classification and vice versa
 - For gradient calculation of samples in line J, line K is padded with line J
- Activity calculation uses a different normalization as less samples are used for block classification.
 - Therefore the following equation is used in calculating the activity:
 - $$\text{avgVar}[x][y] = \text{varTab}[\text{Clip3}(0, 15, (\text{sumOfHV}[x \gg 2][y \gg 2] * \text{Ac}) \gg (3 + \text{BitDepth}_Y))],$$
 where Ac value is 96 for the blocks near the virtual boundary and 64 for the other blocks.

An activity calculation which is used is modified as follows:

$$\text{avgVar}[x][y] = \text{varTab}[\text{Clip3}(0, 15, (\text{sumOfHV}[x \gg 2][y \gg 2] * \text{Ac}) \gg (3 + \text{BitDepth}_Y))]$$

$h_Y))]$, where Ac value is 96 for the blocks near the virtual boundary and 64 for the other blocks.

Alternatively, for the block classification of block beginning at line G, only the lines D, E, F, G are used. Line I could be further used for the classification, in an example, calculates
 5 gradients for every set of 2 lines which makes it cumbersome to include line I in the classification.

The second solution is to use simple padding. The line J is padded to line K, L, M and then the lines beginning from D till M can be used for the block classification.

In Fig. 9b, Fig. 22b and Fig. 18b, (case when 4 x 4 block is below VB), the block
 10 classification for the block beginning at line K uses lines K, L, M, N, O, P, Q. Since gradient calculation of samples belonging to line K line need one more line above, therefore line K is padded to line J. An activity calculation which is used is modified as follows:
 $avgVar[x][y] = varTab[Clip3(0, 15, (sumOfHV[x \gg 2][y \gg 2] * Ac) \gg (3 + BitDept h_Y))]$, where Ac value is 96 for the blocks near the virtual boundary and 64 for the other
 15 blocks.

In one implementation,

For block classification of 4 x 4 block immediate above VB (Fig. 18a):

- Only use samples above VB in classification (samples in lines E till H)
 - Line I and J are considered together in VTM-3.0, therefore only till line H are
 20 used.
 - Pad line J to line K to also use Lines I and J in classification

For block classification of 4 x 4 block immediate below VB (Fig. 18b):

- Only use samples below VB in classification (samples in lines P till M)
 - Line K and L are considered together in VTM-3.0, therefore only till line M
 25 are used.
 - Pad line K to line J to also use Lines I and J in classification.

Alternatively, for the block classification of block beginning at line K, only the lines M, N, O, P, Q are used. Line L could be further used for the classification, in an example, calculates gradients for every set of 2 lines which makes it cumbersome to include line L in the classification.

- 5 The second solution is to use simple padding. The line K is padded to line J, I, H and then the lines beginning from H till Q can be used for the block classification.

Solution 2 (Symmetric truncated ALF block classification)

- As shown in Fig. 10, an implementation refer to modified ALF classification to avoid ALF Luma Line Buffers (Symmetric variant) is disclosed, the symmetric truncated ALF block
10 classification only uses the inner 4 samples (a,b,c,d) to classify the given block.

In an example,

Solution 1:

For block classification of 4 x 4 block immediate above VB (Fig. 10a):

Only use samples above VB in classification. Mainly Samples a, b, c, d are used.

- 15 For block classification of 4 x 4 block immediate below VB (Fig. 10b):

Only use samples below VB in classification. Mainly Samples a, b, c, d are used.

- In Fig. 10a, for the block starting at line G, only the samples belonging to lines H and I (samples a, b, c, d) are used in the gradient computations. No additional padding is required in this case. Similarly in Fig. 10b, for the block starting at line K, only the samples belonging
20 to lines L and M (samples a, b, c, d) are used in the gradient computations. No additional padding is required in this case. This solution has the advantage that the gradient is computed using a symmetric block shape which may be advantageous for subjective picture quality, especially when the picture content changes frequently inside a given local window. Also the solution is computationally less complex as it computes gradient using only 4 samples (a, b, c,
25 d).

Fig. 11 discloses an implementation refer to modified ALF classification to avoid ALF Luma Line Buffers with 6 lines (2 additional lines can be used for ALF block classification), depicts a special case when ALF block classification could use two extra lines of line buffer which can store SAO filtered samples. In this case, the ALF block classification could use
5 two additional lines both above and below the virtual boundary for the blocks starting at line G and line K respectively.

In an example,

Solution 1:

For block classification of 4 x 4 block immediate above VB (left subfigure of Fig. 11):

10 SAO filtered Lines K, L below VB can be further used

For block classification of 4 x 4 block immediate below VB (right subfigure of Fig. 11):

SAO filtered Lines I, J above VB can be further used

In another example,

Solution 2:

15 For block classification of 4 x 4 block immediate above VB (left subfigure of Fig. 11):

Pad line L to line M

For block classification of 4 x 4 block immediate below VB (right subfigure of Fig. 11):

Pad line I to line H.

Modified ALF filtering when virtual boundary is 4 lines

20 Figure. 12a and Figure. 12b discloses an implementation refer to modified ALF filtering to avoid ALF luma line buffers. For filtering of lines H, J, I (above virtual boundary) and lines K, L, M (below virtual boundary), distorted versions of the 7 x 7 luma filter are used.

The filter discards the coefficients which are above and below the virtual boundary in a symmetric fashion. E.g., for the filtering of line M, the coefficient c0 lies above the virtual
25 boundary and vice versa for line H filtering. Therefore in this case for solution 1, the discarded coefficient c0 is added to the neighboring coefficient which is immediately vertical.

Therefore, $2' = c2 + c0$. In solution 2, the discarded coefficient is added to the central coefficient, resulting in $c12' = c12 + 2 * c0$

Similarly for Line L and Line I filtering the modified coefficient values are for solution 1:

$$c5' = c5 + c1$$

$$c6' = c6 + c2 + c0$$

$$c7' = c7 + c3$$

For solution 2: $c12' = c12 + 2 * c0 + 2 * c1 + 2 * c2 + 2 * c3$

- 5 Similarly for Line J and Line K filtering the modified coefficient values are for solution 1:

$$c10' = c10 + c4 + c8$$

$$c11' = c11 + c5 + c7 + c1 + c3$$

$$c12' = c12 + 2 * c6 + 2 * c2 + 2 * c0$$

This solution can also be treated as symmetric padding as shown in Fig. 21. As shown in Fig 21a, for filtering of line M, line K is padded to line J, and line J is padded to line K.

- 10 As shown in Fig. 21b, for filtering of line L, line K is padded to line J and line I. Similarly for filtering of line I, line J is padded to line K and line L.

As shown in Fig. 21c, for filtering of line K, line K is padded to lines J, I and H. Similarly for filtering of line J, line J is padded to lines K, L, and M.

Similar process as described above is also performed for chroma ALF filtering.

- 15 The syntax specification changes for the solution1 is given below:

2.1 General

Inputs of this process are the reconstructed picture sample arrays prior to adaptive loop filter `recPictureL`, `recPictureCb` and `recPictureCr`.

- 20 Outputs of this process are the modified reconstructed picture sample arrays after adaptive loop filter `alfPictureL`, `alfPictureCb` and `alfPictureCr`.

The sample values in the modified reconstructed picture sample arrays after adaptive loop filter `alfPictureL`, `alfPictureCb` and `alfPictureCr`, are initially set equal to the sample values in

the reconstructed picture sample arrays prior to adaptive loop filter `recPictureL`, `recPictureCb` and `recPictureCr`, respectively.

When a value of `tile_group_alf_enabled_flag` is equal to 1, for every coding tree unit with luma coding tree block location (rx, ry) , where $rx = 0..PicWidthInCtbs - 1$ and
5 $ry = 0..PicHeightInCtbs - 1$, the following applies:

When a value of `alf_ctb_flag[ 0 ][ rx ][ ry ]` is equal to 1, the coding tree block filtering process for luma samples as specified in clause 2.2 is invoked with `recPictureL`, `alfPictureL`, and the luma coding tree block location $(xCtb, yCtb)$ set equal to $(rx \ll CtbLog2SizeY, ry \ll CtbLog2SizeY)$ as inputs, and the output is the modified
10 filtered picture `alfPictureL`.

When a value of `alf_ctb_flag[ 1 ][ rx ][ ry ]` is equal to 1, the coding tree block filtering process for chroma samples as specified in clause 2.4 is invoked with `recPicture` set equal to `recPictureCb`, `alfPicture` set equal to `alfPictureCb`, and the chroma coding tree block location $(xCtbC, yCtbC)$ set equal to $(rx \ll (CtbLog2SizeY - 1), ry \ll (CtbLog2SizeY - 1))$
15 as inputs, and the output is the modified filtered picture `alfPictureCb`.

When a value of `alf_ctb_flag[ 2 ][ rx ][ ry ]` is equal to 1, the coding tree block filtering process for chroma samples as specified in clause 2.4 is invoked with `recPicture` set equal to `recPictureCr`, `alfPicture` set equal to `alfPictureCr`, and the chroma coding tree block location $(xCtbC, yCtbC)$ set equal to $(rx \ll (CtbLog2SizeY - 1), ry \ll (CtbLog2SizeY - 1))$
20 as inputs, and the output is the modified filtered picture `alfPictureCr`.

2.2 Coding tree block filtering process for luma samples

Inputs of this process are:

a reconstructed luma picture sample array `recPictureL` prior to the adaptive loop filtering process,
25 a filtered reconstructed luma picture sample array `alfPictureL`,

a luma location (x_{Ctb}, y_{Ctb}) specifying the top-left sample of the current luma coding tree block relative to the top left sample of the current picture.

Output of this process is the modified filtered reconstructed luma picture sample array $alfPictureL$.

- 5 The derivation process for filter index clause 2.3 is invoked with the location (x_{Ctb}, y_{Ctb}) and the reconstructed luma picture sample array $recPictureL$ as inputs, and $filtIdx[x][y]$ and $transposeIdx[x][y]$ with $x, y = 0..CtbSizeY - 1$ as outputs.

For the derivation of the filtered reconstructed luma samples $alfPictureL[x][y]$, each reconstructed luma sample inside the current luma coding tree block $recPictureL[x][y]$ is

- 10 filtered as follows with $x, y = 0..CtbSizeY - 1$:

The array of luma filter coefficients $f[j]$ corresponding to the filter specified by $filtIdx[x][y]$ is derived as follows with $j = 0..12$:

$$f[j] = AlfCoeffL[filtIdx[x][y]][j]$$

The luma filter coefficients $filterCoeff$ are derived depending on $transposeIdx[x][y]$ as

- 15 follows:

If $transposeIndex[x][y] = 1$,

$$filterCoeff[] = \{ f[9], f[4], f[10], f[8], f[1], f[5], f[11], f[7], f[3], f[0], f[2], f[6], f[12] \}$$

Otherwise, if $transposeIndex[x][y] = 2$,

$$filterCoeff[] = \{ f[0], f[3], f[2], f[1], f[8], f[7], f[6], f[5], f[4], f[9], f[10], f[11], f[12] \}$$

- 20 Otherwise, if $transposeIndex[x][y] = 3$,

$$filterCoeff[] = \{ f[9], f[8], f[10], f[4], f[3], f[7], f[11], f[5], f[1], f[0], f[2], f[6], f[12] \}$$

Otherwise,

$$filterCoeff[] = \{ f[0], f[1], f[2], f[3], f[4], f[5], f[6], f[7], f[8], f[9], f[10], f[11], f[12] \}$$

The locations (hx, vy) for each of the corresponding luma samples (x, y) inside the given

- 25 array $recPicture$ of luma samples are derived as follows:

$$hx = Clip3(0, pic_width_in_luma_samples - 1, x_{Ctb} + x)$$

$vy = y$

Filter input sample position offset $r = 0 \dots 6$ are derived according to luma sample position y as shown in the Table 2-2a below

Table 2-2a – Specification of filter input sample position offset $r=0..6$ according to the luma sample position y

y	$r1$	$r2$	$r3$	$r4$	$r5$	$r6$
$y == \text{CtbSizeY} - 5$ (or) $\text{CtbSizeY} - 4$	0	0	0	0	0	0
$y == \text{CtbSizeY} - 6$ (or) $\text{CtbSizeY} - 3$	1	1	1	1	1	1
$y == \text{CtbSizeY} - 7$ (or) $\text{CtbSizeY} - 2$	1	1	2	2	2	2
$y\text{Ctb} + y == \text{pic_height_in_luma_samples} - 1$	0	1	0	2	0	3
$y\text{Ctb} + y == \text{pic_height_in_luma_samples} - 2$	1	1	1	2	1	3
$y\text{Ctb} + y == \text{pic_height_in_luma_samples} - 3$	1	1	2	2	2	3
$y\text{Ctb} + y == 0$	1	0	2	0	3	0
$y\text{Ctb} + y == 1$	1	1	2	1	3	1
$y\text{Ctb} + y == 2$	1	1	2	2	3	2
Other cases	1	1	2	2	3	3

The variable sum is derived as follows:

$$\begin{aligned}
 \text{sum} = & \text{filterCoeff}[0] * (\text{recPictureL}[hx, vy + r5] + \text{recPictureL}[hx, vy - r6]) + \\
 & \text{filterCoeff}[1] * (\text{recPictureL}[hx + 1, vy + r3] + \text{recPictureL}[hx - 1, vy - r4]) + \\
 & \text{filterCoeff}[2] * (\text{recPictureL}[hx, vy + r3] + \text{recPictureL}[hx, vy - r4]) + \\
 10 \quad & \text{filterCoeff}[3] * (\text{recPictureL}[hx - 1, vy + r3] + \text{recPictureL}[hx + 1, vy - r4]) + \\
 & \text{filterCoeff}[4] * (\text{recPictureL}[hx + 2, vy + r1] + \text{recPictureL}[hx - 2, vy - r2]) + \\
 & \text{filterCoeff}[5] * (\text{recPictureL}[hx + 1, vy + r1] + \text{recPictureL}[hx - 1, vy - r2]) + \\
 & \text{filterCoeff}[6] * (\text{recPictureL}[hx, vy + r1] + \text{recPictureL}[hx, vy - r2]) + (8-1155) \\
 & \text{filterCoeff}[7] * (\text{recPictureL}[hx - 1, vy + r1] + \text{recPictureL}[hx + 1, vy - r2]) +
 \end{aligned}$$

```

    filterCoeff[ 8 ] * ( recPictureL[ hx - 2, vy + r1 ] + recPictureL[ hx + 2, vy - r2 ] ) +
    filterCoeff[ 9 ] * ( recPictureL[ hx + 3, vy ] + recPictureL[ hx - 3, vy ] ) +
    filterCoeff[ 10 ] * ( recPictureL[ hx + 2, vy ] + recPictureL[ hx - 2, vy ] ) +
    filterCoeff[ 11 ] * ( recPictureL[ hx + 1, vy ] + recPictureL[ hx - 1, vy ] ) +
5    filterCoeff[ 12 ] * recPictureL[ hx, vy ]

sum = ( sum + 64 ) >> 7

```

The modified filtered reconstructed luma picture sample $\text{alfPictureL}[x_{\text{Ctb}} + x][y_{\text{Ctb}} + y]$ is derived as follows:

$\text{alfPictureL}[x_{\text{Ctb}} + x][y_{\text{Ctb}} + y] = \text{Clip3}(0, (1 \ll \text{BitDepthY}) - 1, \text{sum})$.

10 2.3 Derivation process for ALF transpose and filter index for luma samples

Inputs of this process are:

a luma location $(x_{\text{Ctb}}, y_{\text{Ctb}})$ specifying the top-left sample of the current luma coding tree block relative to the top left sample of the current picture,

a reconstructed luma picture sample array recPictureL prior to the adaptive loop filtering
15 process.

Outputs of this process are

the classification filter index array $\text{filtIdx}[x][y]$ with $x, y = 0..CtbSizeY - 1$,

the transpose index array $\text{transposeIdx}[x][y]$ with $x, y = 0..CtbSizeY - 1$.

The locations (hx, vy) for each of the corresponding luma samples (x, y) inside the given
20 array recPicture of luma samples are derived as follows:

$hx = \text{Clip3}(0, \text{pic_width_in_luma_samples} - 1, x)$

if $y_{\text{Ctb}} + CtbSizeY \geq \text{pic_height_in_luma_samples}$

$vy = \text{Clip3}(0, \text{pic_height_in_luma_samples} - 1, y)$

Otherwise, if $y < (CtbSizeY - 4)$,

25 $vy = \text{Clip3}(0, y_{\text{Ctb}} + CtbSizeY - 5, y_{\text{Ctb}} + y)$

Otherwise,

$$vy = \text{Clip3}(yCtb + CtbSizeY - 4, \text{pic_height_in_luma_samples} - 1, yCtb + y).$$

The classification filter index array `filtIdx` and the transpose index array `transposeIdx` are derived by the following ordered steps:

- 5 The variables `filtH[ x ][ y ]`, `filtV[ x ][ y ]`, `filtD0[ x ][ y ]` and `filtD1[ x ][ y ]` with $x, y = -2..CtbSizeY + 1$ are derived as follows:

If both x and y are even numbers or both x and y are uneven numbers, the following applies:

$$\text{filtH}[x][y] = \text{Abs}((\text{recPicture}[hxCtb+x, vyCtb+y] < 1) - \text{recPicture}[hxCtb+x-1, vyCtb+y] - \text{recPicture}[hxCtb+x+1, vyCtb+y])$$

$$10 \quad \text{filtV}[x][y] = \text{Abs}((\text{recPicture}[hxCtb+x, vyCtb+y] < 1) - \text{recPicture}[hxCtb+x, vyCtb+y-1] - \text{recPicture}[hxCtb+x, vyCtb+y+1])$$

$$\text{filtD0}[x][y] = \text{Abs}((\text{recPicture}[hxCtb+x, vyCtb+y] < 1) - \text{recPicture}[hxCtb+x-1, vyCtb+y-1] - \text{recPicture}[hxCtb+x+1, vyCtb+y+1])$$

$$15 \quad \text{filtD1}[x][y] = \text{Abs}((\text{recPicture}[hxCtb+x, vyCtb+y] < 1) - \text{recPicture}[hxCtb+x-1, vyCtb+y-1] - \text{recPicture}[hxCtb+x+1, vyCtb+y+1])$$

Otherwise, `filtH[ x ][ y ]`, `filtV[ x ][ y ]`, `filtD0[ x ][ y ]` and `filtD1[ x ][ y ]` are set equal to 0.

The variables `varTempH1[ x ][ y ]`, `varTempV1[ x ][ y ]`, `varTempD01[ x ][ y ]`, `varTempD11[ x ][ y ]` and `varTemp[ x ][ y ]` with $x, y = 0..(CtbSizeY - 1) >> 2$ are derived as follows:

- 20 If $(y < 2) == CtbSizeY - 8$ and $yCtb + CtbSizeY < \text{pic_height_in_luma_samples} - 1$ then

$$\text{sumH}[x][y] = \sum_i \sum_j \text{filtH}[(x < 2) + i][(y < 2) + j] \text{ with } i = -2..5 \text{ and } j = -2..3$$

$$\text{sumV}[x][y] = \sum_i \sum_j \text{filtV}[(x < 2) + i][(y < 2) + j] \text{ with } i = -2..5 \text{ and } j = -2..3$$

$$\text{sumD0}[x][y] = \sum_i \sum_j \text{filtD0}[(x < 2) + i][(y < 2) + j] \text{ with } i = -2..5 \text{ and } j =$$

$$25 \quad -2..3$$

$\text{sumD1}[x][y] = \sum_i \sum_j \text{filtD1}[(x \ll 2) + i][(y \ll 2) + j]$ with $i = -2..5$ and $j = -2..3$

variable Ac is set as 96

otherwise, If $(y \ll 2) == \text{CtbSizeY} - 4$ and $y_{\text{Ctb}} + \text{CtbSizeY} <$

5 $\text{pic_height_in_luma_samples} - 1$ then

$\text{sumH}[x][y] = \sum_i \sum_j \text{filtH}[(x \ll 2) + i][(y \ll 2) + j]$ with $i = -2..5$ and $j = 0..5$

$\text{sumV}[x][y] = \sum_i \sum_j \text{filtV}[(x \ll 2) + i][(y \ll 2) + j]$ with $i = -2..5$ and $j = 0..5$

$\text{sumD0}[x][y] = \sum_i \sum_j \text{filtD0}[(x \ll 2) + i][(y \ll 2) + j]$ with $i = -2..5$ and $j = 0..5$

$\text{sumD1}[x][y] = \sum_i \sum_j \text{filtD1}[(x \ll 2) + i][(y \ll 2) + j]$ with $i = -2..5$ and $j = 0..5$

10 variable Ac is set as 96

otherwise

$\text{sumH}[x][y] = \sum_i \sum_j \text{filtH}[(x \ll 2) + i][(y \ll 2) + j]$ with $i, j = -2..5$

$\text{sumV}[x][y] = \sum_i \sum_j \text{filtV}[(x \ll 2) + i][(y \ll 2) + j]$ with $i, j = -2..5$

$\text{sumD0}[x][y] = \sum_i \sum_j \text{filtD0}[(x \ll 2) + i][(y \ll 2) + j]$ with $i, j = -2..5$

15 $\text{sumD1}[x][y] = \sum_i \sum_j \text{filtD1}[(x \ll 2) + i][(y \ll 2) + j]$ with $i, j = -2..5$

$\text{sumOfHV}[x][y] = \text{sumH}[x][y] + \text{sumV}[x][y]$ (8-1168)

variable Ac is set as 64

The variables $\text{dir1}[x][y]$, $\text{dir2}[x][y]$ and $\text{dirS}[x][y]$ with $x, y = 0.. \text{CtbSizeY} - 1$ are derived as follows:

20 The variables hv1 , hv0 and dirHV are derived as follows:

If $\text{sumV}[x \gg 2][y \gg 2]$ is greater than $\text{sumH}[x \gg 2][y \gg 2]$, the following applies:

$\text{hv1} = \text{sumV}[x \gg 2][y \gg 2]$

$\text{hv0} = \text{sumH}[x \gg 2][y \gg 2]$

$\text{dirHV} = 1;$

25 Otherwise, the following applies:

$\text{hv1} = \text{sumH}[x \gg 2][y \gg 2]$

hv0 = sumV[x >> 2][y >> 2]

dirHV = 3.

The variables d1, d0 and dirD are derived as follows:

If sumD0[x >> 2][y >> 2] is greater than sumD1[x >> 2][y >> 2], the following applies:

5 d1 = sumD0[x >> 2][y >> 2]

d0 = sumD1[x >> 2][y >> 2]

dirD = 0;

Otherwise, the following applies:

d1 = sumD1[x >> 2][y >> 2]

10 d0 = sumD0[x >> 2][y >> 2]

dirD = 2.

The variables hvd1, hvd0, are derived as follows:

hvd1 = (d1 * hv0 > hv1 * d0) ? d1 : hv1

hvd0 = (d1 * hv0 > hv1 * d0) ? d0 : hv0.

15 The variables dirS[x][y], dir1[x][y] and dir2[x][y] derived as follows:

dir1[x][y] = (d1 * hv0 > hv1 * d0) ? dirD : dirHV

dir2[x][y] = (d1 * hv0 > hv1 * d0) ? dirHV : dirD

dirS[x][y] = (hvd1 > 2 * hvd0) ? 1 : ((hvd1 * 2 > 9 * hvd0) ? 2 : 0).

The variable avgVar[x][y] with x, y = 0..CtbSizeY - 1 is derived as follows:

20 varTab[] = { 0, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 4 }

avgVar[x][y] = varTab[Clip3(0, 15, (sumOfHV[x >> 2][y >> 2] * Ac) >> (3 + BitDept
hY))].

The classification filter index array filtIdx[x][y] and the transpose index array

transposeIdx[x][y] with x = y = 0..CtbSizeY - 1 are derived as follows:

25 transposeTable[] = { 0, 1, 0, 2, 2, 3, 1, 3 }

transposeIdx[x][y] = transposeTable[dir1[x][y] * 2 + (dir2[x][y] >> 1)]

$\text{filtIdx}[x][y] = \text{avgVar}[x][y]$

When $\text{dirS}[x][y]$ is not equal 0, $\text{filtIdx}[x][y]$ is modified as follows:

$\text{filtIdx}[x][y] += (((\text{dir1}[x][y] \& 0x1) \ll 1) + \text{dirS}[x][y]) * 5.$

2.4 Coding tree block filtering process for chroma samples

5 Inputs of this process are:

a reconstructed chroma picture sample array recPicture prior to the adaptive loop filtering process,

a filtered reconstructed chroma picture sample array alfPicture ,

a chroma location (xCtbC , yCtbC) specifying the top-left sample of the current chroma

10 coding tree block relative to the top left sample of the current picture.

Output of this process is the modified filtered reconstructed chroma picture sample array alfPicture .

The size of the current chroma coding tree block ctbSizeC is derived as follows:

$\text{ctbSizeC} = \text{CtbSizeY} / \text{SubWidthC}.$

15 For the derivation of the filtered reconstructed chroma samples $\text{alfPicture}[x][y]$, each reconstructed chroma sample inside the current chroma coding tree block $\text{recPicture}[x][y]$ is filtered as follows with $x, y = 0.. \text{ctbSizeC} - 1$:

The locations (hx , vy) for each of the corresponding chroma samples (x , y) inside the given array recPicture of chroma samples are derived as follows:

20 $\text{hx} = \text{Clip3}(0, \text{pic_width_in_luma_samples} / \text{SubWidthC} - 1, \text{xCtbC} + x)$

$\text{vy} = y$

filter input sample Position offset $r = 0..4$ are derived according to chroma sample position y as shown in the Table 2.4 below.

Table 2.4b – Specification of filter input sample position offset $r=0..4$ according to the chroma

25 sample position y

y	r1	r2	r3	r4
y == CtbSizeC – 2 (or) CtbSizeC – 3	0	0	0	0
y == CtbSizeC – 1 (or) CtbSizeC – 4	1	1	1	1
y == CtbSizeY – 7 (or) CtbSizeY – 2	1	1	2	2
yCtb + y == (pic_height_in_luma_samples / SubHeightC – 1) - 1	0	1	0	2
yCtb + y == (pic_height_in_luma_samples / SubHeightC – 1) - 2	1	1	1	2
yCtb + y == 0	1	0	2	0
yCtb + y == 1	1	1	2	1
Other cases	1	1	2	2

The variable sum is derived as follows:

$$\begin{aligned}
\text{sum} = & \text{AlfCoeffC}[0] * (\text{recPicture}[hx, vy + r3] + \text{recPicture}[hx, vy - r4]) + \\
& \text{AlfCoeffC}[1] * (\text{recPicture}[hx + 1, vy + r1] + \text{recPicture}[hx - 1, vy - r2]) + \\
& \text{AlfCoeffC}[2] * (\text{recPicture}[hx, vy + r1] + \text{recPicture}[hx, vy - r2]) + \\
5 \quad & \text{AlfCoeffC}[3] * (\text{recPicture}[hx - 1, vy + r1] + \text{recPicture}[hx + 1, vy - r2]) + \\
& \text{AlfCoeffC}[4] * (\text{recPicture}[hx + 2, vy] + \text{recPicture}[hx - 2, vy]) + \\
& \text{AlfCoeffC}[5] * (\text{recPicture}[hx + 1, vy] + \text{recPicture}[hx - 1, vy]) + \\
& \text{AlfCoeffC}[6] * \text{recPicture}[hx, vy] \\
\text{sum} = & (\text{sum} + 64) >> 7
\end{aligned}$$

10 The modified filtered reconstructed chroma picture sample

$\text{alfPicture}[xCtbC + x][yCtbC + y]$ is derived as follows:

$$\text{alfPicture}[xCtbC + x][yCtbC + y] = \text{Clip3}(0, (1 \ll \text{BitDepthC}) - 1, \text{sum}).$$

For solution 2:

$$\begin{aligned}
c_{12}' = & c_{12} + 2 * c_0 + 2 * c_1 + 2 * c_2 + 2 * c_3 + 2 * c_4 + 2 * c_5 + 2 * c_6 + 2 * c_7 + \\
15 \quad & 2 * c_8.
\end{aligned}$$

The ALF filtering process as per ITU JVET-Q0044 is as follows:

8.8.5 Adaptive loop filter process

8.8.5.1 General

Inputs of this process are the reconstructed picture sample array prior to adaptive loop filter recPicture_L and, when ChromaArrayType is not equal to 0, the arrays recPicture_{Cb} and recPicture_{Cr} .

Outputs of this process are the modified reconstructed picture sample array after adaptive loop filter alfPicture_L and, when ChromaArrayType is not equal to 0, the arrays alfPicture_{Cb} and alfPicture_{Cr} .

The sample values in the modified reconstructed picture sample array after adaptive loop filter alfPicture_L and, when ChromaArrayType is not equal to 0, the arrays alfPicture_{Cb} and alfPicture_{Cr} are initially set equal to the sample values in the reconstructed picture sample array prior to adaptive loop filter recPicture_L and, when ChromaArrayType is not equal to 0, the arrays recPicture_{Cb} and recPicture_{Cr} , respectively.

For every coding tree unit with luma coding tree block location (rx, ry) , where $rx = 0..PicWidthInCtbsY - 1$ and $ry = 0..PicHeightInCtbsY - 1$, the following applies:

When $\text{alf_ctb_flag}[0][rx][ry]$ is equal to 1, the coding tree block filtering process for luma samples as specified in clause 8.8.5.2 is invoked with recPicture_L , alfPicture_L , and the luma coding tree block location $(xCtb, yCtb)$ set equal to $(rx \ll CtbLog2SizeY, ry \ll CtbLog2SizeY)$ as inputs, and the output is the modified filtered picture alfPicture_L .

When ChromaArrayType is not equal to 0 and $\text{alf_ctb_flag}[1][rx][ry]$ is equal to 1, the coding tree block filtering process for chroma samples as specified in clause 8.8.5.4 is invoked with recPicture set equal to recPicture_{Cb} , alfPicture set equal to alfPicture_{Cb} , the chroma coding tree block location $(xCtbC, yCtbC)$ set equal to $((rx \ll CtbLog2SizeY) / \text{SubWidthC}, (ry \ll CtbLog2SizeY) / \text{SubHeightC})$, and the alternative chroma filter index altIdx set equal to $\text{alf_ctb_filter_alt_idx}[0][rx][ry]$ as inputs, and the output is the modified filtered picture alfPicture_{Cb} .

When ChromaArrayType is not equal to 0 and $\text{alf_ctb_flag}[2][rx][ry]$ is equal to 1, the coding tree block filtering process for chroma samples as specified in clause 8.8.5.4 is invoked with recPicture set equal to recPicture_{Cr} , alfPicture set equal to alfPicture_{Cr} , the chroma coding tree block location $(xCtbC, yCtbC)$ set equal to $((rx \ll CtbLog2SizeY) / \text{SubWidthC}, (ry \ll CtbLog2SizeY) / \text{SubHeightC})$, and the alternative chroma filter index altIdx set equal to $\text{alf_ctb_filter_alt_idx}[0][rx][ry]$ as inputs, and the output is the modified filtered picture alfPicture_{Cr} .

8.8.5.2 Coding tree block filtering process for luma samples

Inputs of this process are:

a reconstructed luma picture sample array `recPicture` prior to the adaptive loop filtering process,

- a filtered reconstructed luma picture sample array `alfPictureL`,
- a luma location (`xCtb`, `yCtb`) specifying the top-left sample of the current luma coding tree block relative to the top left sample of the current picture.

5

Output of this process is the modified filtered reconstructed luma picture sample array `alfPictureL`.

The derivation process for filter index clause 8.8.5.3 is invoked with the location (`xCtb`, `yCtb`) and the reconstructed luma picture sample array `recPicture` as inputs, and

10

`filtIdx[ x ][ y ]` and `transposeIdx[ x ][ y ]` with $x, y = 0..CtbSizeY - 1$ as outputs.
For the derivation of the filtered reconstructed luma samples `alfPictureL[ x ][ y ]`, each reconstructed luma sample inside the current luma coding tree block `recPicture[ x ][ y ]` is filtered as follows with $x, y = 0..CtbSizeY - 1$:

15

The array of luma filter coefficients `f[ j ]` and the array of luma clipping values `c[ j ]` corresponding to the filter specified by `filtIdx[ x ][ y ]` is derived as follows with $j = 0..11$:

If `AlfCtbFiltSetIdxY[ xCtb >> CtbLog2SizeY ][ yCtb >> CtbLog2SizeY ]` is less than 16, the following applies:

$$i = \text{AlfCtbFiltSetIdxY}[xCtb \gg CtbLog2SizeY][yCtb \gg CtbLog2SizeY] \quad (1411)$$

$$f[j] = \text{AlfFixFiltCoeff}[\text{AlfClassToFiltMap}[i][\text{filtIdx}[x][y]]][j] \quad (1412)$$

20

$$c[j] = 2^{\text{BitDepth}} \quad (1413)$$

Otherwise (`AlfCtbFiltSetIdxY[ xCtb >> CtbLog2SizeY ][ yCtb >> CtbLog2SizeY ]` is greater than or equal to 16, the following applies:

$$i = \text{slice_alf_aps_id_luma}[\text{AlfCtbFiltSetIdxY}[xCtb \gg CtbLog2SizeY][yCtb \gg CtbLog2SizeY] - 16] \quad (1414)$$

25

$$f[j] = \text{AlfCoeff}_L[i][\text{filtIdx}[x][y]][j] \quad (1415)$$

$$c[j] = \text{AlfClip}_L[i][\text{filtIdx}[x][y]][j] \quad (1416)$$

The luma filter coefficients and clipping values index `idx` are derived depending on `transposeIdx[ x ][ y ]` as follows:

If `transposeIndex[ x ][ y ]` is equal to 1, the following applies:

30

$$\text{idx}[] = \{ 9, 4, 10, 8, 1, 5, 11, 7, 3, 0, 2, 6 \} \quad (1417)$$

Otherwise, if transposeIndex[x][y] is equal to 2, the following applies:

$$\text{idx}[] = \{ 0, 3, 2, 1, 8, 7, 6, 5, 4, 9, 10, 11 \} \quad (1418)$$

Otherwise, if transposeIndex[x][y] is equal to 3, the following applies:

$$\text{idx}[] = \{ 9, 8, 10, 4, 3, 7, 11, 5, 1, 0, 2, 6 \} \quad (1419)$$

5 Otherwise, the following applies:

$$\text{idx}[] = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 \} \quad (1420)$$

The locations (h_{x+i} , v_{y+j}) for each of the corresponding luma samples (x , y) inside the given array recPicture of luma samples with $i, j = -3..3$ are derived as follows:

$$h_{x+i} = \text{Clip3}(0, \text{pic_width_in_luma_samples} - 1, \text{xCtb} + x + i) \quad (1421)$$

$$10 \quad v_{y+j} = \text{Clip3}(0, \text{pic_height_in_luma_samples} - 1, \text{yCtb} + y + j) \quad (1422)$$

The variables clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with (xCtb , yCtb) and (x , y) as inputs.

15 The variables h_{x+i} and v_{y+j} are modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with (xCtb , yCtb), (h_{x+i} , v_{y+j}), 0, clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag as input.

The variable applyAlfLineBufBoundary is derived as follows:

20 If the bottom boundary of the current coding tree block is the bottom boundary of current picture and $\text{pic_height_in_luma_samples} - \text{yCtb} \leq \text{CtbSizeY} - 4$, applyAlfLineBufBoundary is set equal to 0:

Otherwise, applyAlfLineBufBoundary is set equal to 1.

The vertical sample position offsets y_1 , y_2 and y_3 are specified in Table 43 according to the vertical luma sample position y and applyAlfLineBufBoundary.

The variable curr is derived as follows:

$$25 \quad \text{curr} = \text{recPicture}[h_x][v_y] \quad (1423)$$

The variable sum is derived as follows:

$$\begin{aligned} \text{sum} = & f[\text{idx}[0]] * (\text{Clip3}(-c[\text{idx}[0]], c[\text{idx}[0]], \text{recPicture}[h_x][v_{y+y_3}] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[0]], c[\text{idx}[0]], \text{recPicture}[h_x][v_{y-y_3}] - \text{curr})) + \\ & f[\text{idx}[1]] * (\text{Clip3}(-c[\text{idx}[1]], c[\text{idx}[1]], \text{recPicture}[h_{x+1}][v_{y+y_2}] - \text{curr}) + \end{aligned}$$

$$\begin{aligned}
& \text{Clip3}(-c[\text{idx}[1]], c[\text{idx}[1]], \text{recPicture}[h_{x-1}][v_y - y_2] - \text{curr})) + \\
& f[\text{idx}[2]] * (\text{Clip3}(-c[\text{idx}[2]], c[\text{idx}[2]], \text{recPicture}[h_x][v_y + y_2] - \text{curr})) + \\
& \text{Clip3}(-c[\text{idx}[2]], c[\text{idx}[2]], \text{recPicture}[h_x][v_y - y_2] - \text{curr})) + \\
& f[\text{idx}[3]] * (\text{Clip3}(-c[\text{idx}[3]], c[\text{idx}[3]], \text{recPicture}[h_{x-1}][v_y + y_2] - \text{curr})) + \\
5 & \text{Clip3}(-c[\text{idx}[3]], c[\text{idx}[3]], \text{recPicture}[h_{x+1}][v_y - y_2] - \text{curr})) + \\
& f[\text{idx}[4]] * (\text{Clip3}(-c[\text{idx}[4]], c[\text{idx}[4]], \text{recPicture}[h_{x+2}][v_y + y_1] - \text{curr})) + \\
& \text{Clip3}(-c[\text{idx}[4]], c[\text{idx}[4]], \text{recPicture}[h_{x-2}][v_y - y_1] - \text{curr})) + \\
& f[\text{idx}[5]] * (\text{Clip3}(-c[\text{idx}[5]], c[\text{idx}[5]], \text{recPicture}[h_{x+1}][v_y + y_1] - \text{curr})) + \\
& \text{Clip3}(-c[\text{idx}[5]], c[\text{idx}[5]], \text{recPicture}[h_{x-1}][v_y - y_1] - \text{curr})) + \\
10 & f[\text{idx}[6]] * (\text{Clip3}(-c[\text{idx}[6]], c[\text{idx}[6]], \text{recPicture}[h_x][v_y + y_1] - \text{curr})) + \\
& \text{Clip3}(-c[\text{idx}[6]], c[\text{idx}[6]], \text{recPicture}[h_x][v_y - y_1] - \text{curr})) + \\
& f[\text{idx}[7]] * (\text{Clip3}(-c[\text{idx}[7]], c[\text{idx}[7]], \text{recPicture}[h_{x-1}][v_y + y_1] - \text{curr})) + \\
& \text{Clip3}(-c[\text{idx}[7]], c[\text{idx}[7]], \text{recPicture}[h_{x+1}][v_y - y_1] - \text{curr})) + \\
& f[\text{idx}[8]] * (\text{Clip3}(-c[\text{idx}[8]], c[\text{idx}[8]], \text{recPicture}[h_{x-2}][v_y + y_1] - \text{curr})) + \\
15 & \text{Clip3}(-c[\text{idx}[8]], c[\text{idx}[8]], \text{recPicture}[h_{x+2}][v_y - y_1] - \text{curr})) + \\
& f[\text{idx}[9]] * (\text{Clip3}(-c[\text{idx}[9]], c[\text{idx}[9]], \text{recPicture}[h_{x+3}][v_y] - \text{curr})) + \\
& \text{Clip3}(-c[\text{idx}[9]], c[\text{idx}[9]], \text{recPicture}[h_{x-3}][v_y] - \text{curr})) + \\
& f[\text{idx}[10]] * (\text{Clip3}(-c[\text{idx}[10]], c[\text{idx}[10]], \text{recPicture}[h_{x+2}][v_y] - \text{curr})) + \\
& \text{Clip3}(-c[\text{idx}[10]], c[\text{idx}[10]], \text{recPicture}[h_{x-2}][v_y] - \text{curr})) + \\
20 & f[\text{idx}[11]] * (\text{Clip3}(-c[\text{idx}[11]], c[\text{idx}[11]], \text{recPicture}[h_{x+1}][v_y] - \text{curr})) + \\
& \text{Clip3}(-c[\text{idx}[11]], c[\text{idx}[11]], \text{recPicture}[h_{x-1}][v_y] - \text{curr})) \\
& \text{sum} = \text{curr} + ((\text{sum} + 64) >> 7) \tag{1425}
\end{aligned}$$

The modified filtered reconstructed luma picture sample $\text{alfPicture}_L[x\text{Ctb} + x][y\text{Ctb} + y]$ is derived as follows:

$$25 \quad \text{alfPicture}_L[x\text{Ctb} + x][y\text{Ctb} + y] = \text{Clip3}(0, (1 \ll \text{BitDepth}) - 1, \text{sum}) \tag{1426}$$

Table 43 – Specification of y1, y2, and y3 according to the vertical luma sample position y and applyAlfLineBufBoundary

Condition	y1	y2	y3
$(y == \text{CtbSizeY} - 5 \mid \mid y == \text{CtbSizeY} - 4) \ \&\&$ $(\text{applyAlfLineBufBoundary} == 1)$	0	0	0
$(y == \text{CtbSizeY} - 6 \mid \mid y == \text{CtbSizeY} - 3) \ \&\&$ $(\text{applyAlfLineBufBoundary} == 1)$	1	1	1
$(y == \text{CtbSizeY} - 7 \mid \mid y == \text{CtbSizeY} - 2) \ \&\&$ $(\text{applyAlfLineBufBoundary} == 1)$	1	2	2
otherwise	1	2	3

8.8.5.3 Derivation process for ALF transpose and filter index for luma samples

Inputs of this process are:

- a luma location (x_{Ctb} , y_{Ctb}) specifying the top-left sample of the current luma coding tree block relative to the top left sample of the current picture,
- 5 a reconstructed luma picture sample array $recPicture$ prior to the adaptive loop filtering process.

Outputs of this process are

- the classification filter index array $filtIdx[x][y]$ with $x, y = 0..CtbSizeY - 1$,
- the transpose index array $transposeIdx[x][y]$ with $x, y = 0..CtbSizeY - 1$.
- 10 The variables $ac[x][y]$, $sumH[x][y]$, $sumV[x][y]$, $sumD0[x][y]$, $sumD1[x][y]$ and $sumOfHV[x][y]$ with $x, y = 0..(CtbSizeY - 1) \gg 2$ are derived as follows:

The variables x_4 and y_4 are set as $(x \ll 2)$ and $(y \ll 2)$, respectively.

The variables $minY$, $maxY$, and $ac[x][y]$ are derived as follows:

- 15 If y_4 is equal to $(CtbSizeY - 8)$ and one of the following condition is true, $minY$ is set equal to -2 , $maxY$ is set equal to 3 , and $ac[x][y]$ is set equal to 3 .

The bottom boundary of the current coding tree block is the bottom boundary of the picture and $pic_height_in_luma_samples - y_{Ctb} > CtbSizeY - 4$.

The bottom boundary of the current coding tree block is not the bottom boundary of the picture.

- 20 Otherwise, if y_4 is equal to $(CtbSizeY - 4)$ and one of the following condition is true, $minY$ is set equal to 0 , $maxY$ is set equal to 5 , and $ac[x][y]$ is set equal to 3 .

The bottom boundary of the current coding tree block is the bottom boundary of the picture and $pic_height_in_luma_samples - y_{Ctb} > CtbSizeY - 4$.

- 25 The bottom boundary of the current coding tree block is not the bottom boundary of the picture

- Otherwise, $minY$ is set equal to -2 and $maxY$ is set equal to 5 , and $ac[x][y]$ is set equal to 2 .

- The variables $clipLeftPos$, $clipRightPos$, $clipTopPos$, $clipBottomPos$, $clipTopLeftFlag$ and $clipBotRightFlag$ are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with (x_{Ctb}, y_{Ctb}) and (x_4, y_4) as inputs.
- 30

- The locations (h_{x_4+i}, v_{y_4+j}) for each of the corresponding luma samples inside the given array $recPicture$ of luma samples with $i, j = -3..6$ are derived as follows:

$$h_{x4+i} = \text{Clip3}(0, \text{pic_width_in_luma_samples} - 1, x\text{Ctb} + x4 + i) \quad (1427)$$

$$v_{y4+j} = \text{Clip3}(0, \text{pic_height_in_luma_samples} - 1, y\text{Ctb} + y4 + j) \quad (1428)$$

5 The variables h_{x4+i} and v_{y4+j} are modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with $(x\text{Ctb}, y\text{Ctb})$, (h_{x4+i}, v_{y4+j}) , the variable isChroma set equal to 0, clipLeftPos , clipRightPos , clipTopPos , clipBottomPos , clipTopLeftFlag and clipBotRightFlag as input.

The variables $\text{filtH}[i][j]$, $\text{filtV}[i][j]$, $\text{filtD0}[i][j]$ and $\text{filtD1}[i][j]$ with $i, j = -2..5$ are derived as follows:

10 If both i and j are even numbers or both i and j are not even numbers, the following applies:

$$\text{filtH}[i][j] = \text{Abs}((\text{recPicture}[h_{x4+i}][v_{y4+j}] \ll 1) - \text{recPicture}[h_{x4+i-1}][v_{y4+j}] - \text{recPicture}[h_{x4+i+1}][v_{y4+j}]) \quad (1429)$$

$$\text{filtV}[i][j] = \text{Abs}((\text{recPicture}[h_{x4+i}][v_{y4+j}] \ll 1) - \text{recPicture}[h_{x4+i}][v_{y4+j-1}] - \text{recPicture}[h_{x4+i}][v_{y4+j+1}]) \quad (1430)$$

$$\begin{aligned} 15 \quad \text{filtD0}[i][j] &= \text{Abs}((\text{recPicture}[h_{x4+i}][v_{y4+j}] \ll 1) - \text{recPicture}[h_{x4+i-1}][v_{y4+j-1}] - \text{recPicture}[h_{x4+i+1}][v_{y4+j-1}] \\ &\quad - \text{recPicture}[h_{x4+i-1}][v_{y4+j+1}]) \quad (1432) \end{aligned}$$

Otherwise, $\text{filtH}[i][j]$, $\text{filtV}[i][j]$, $\text{filtD0}[i][j]$ and $\text{filtD1}[i][j]$ are set equal to 0.

20 The variables $\text{sumH}[x][y]$, $\text{sumV}[x][y]$, $\text{sumD0}[x][y]$, $\text{sumD1}[x][y]$ and $\text{sumOfHV}[x][y]$ are derived as follows:

$$\text{sumH}[x][y] = \sum_i \sum_j \text{filtH}[i][j], \text{ with } i = -2..5, j = \text{minY}.. \text{maxY} \quad (1433)$$

$$\text{sumV}[x][y] = \sum_i \sum_j \text{filtV}[i][j], \text{ with } i = -2..5, j = \text{minY}.. \text{maxY} \quad (1434)$$

$$\text{sumD0}[x][y] = \sum_i \sum_j \text{filtD0}[i][j], \text{ with } i = -2..5, j = \text{minY}.. \text{maxY} \quad (1435)$$

$$\text{sumD1}[x][y] = \sum_i \sum_j \text{filtD1}[i][j], \text{ with } i = -2..5, j = \text{minY}.. \text{maxY} \quad (1436)$$

$$25 \quad \text{sumOfHV}[x][y] = \text{sumH}[x][y] + \text{sumV}[x][y] \quad (1437)$$

The classification filter index array filtIdx and transpose index array transposeIdx are derived by the following steps:

1. The variables $\text{dir1}[x][y]$, $\text{dir2}[x][y]$ and $\text{dirS}[x][y]$ with $x, y = 0.. \text{CtbSizeY} - 1$ are derived as follows:
 - 30 – The variables hv1 , hv0 and dirHV are derived as follows:

- If $\text{sumV}[x \gg 2][y \gg 2]$ is greater than $\text{sumH}[x \gg 2][y \gg 2]$, the following applies:

$$\text{hv1} = \text{sumV}[x \gg 2][y \gg 2] \quad (1438)$$

$$\text{hv0} = \text{sumH}[x \gg 2][y \gg 2] \quad (1439)$$

$$5 \quad \text{dirHV} = 1 \quad (1440)$$

- Otherwise, the following applies:

$$\text{hv1} = \text{sumH}[x \gg 2][y \gg 2] \quad (1441)$$

$$\text{hv0} = \text{sumV}[x \gg 2][y \gg 2] \quad (1442)$$

$$\text{dirHV} = 3 \quad (1443)$$

- $$10 \quad \text{– The variables } d1, d0 \text{ and } \text{dirD} \text{ are derived as follows:}$$

- If $\text{sumD0}[x \gg 2][y \gg 2]$ is greater than $\text{sumD1}[x \gg 2][y \gg 2]$, the following applies:

$$d1 = \text{sumD0}[x \gg 2][y \gg 2] \quad (1444)$$

$$d0 = \text{sumD1}[x \gg 2][y \gg 2] \quad (1445)$$

$$15 \quad \text{dirD} = 0 \quad (1446)$$

- Otherwise, the following applies:

$$d1 = \text{sumD1}[x \gg 2][y \gg 2] \quad (1447)$$

$$d0 = \text{sumD0}[x \gg 2][y \gg 2] \quad (1448)$$

$$\text{dirD} = 2 \quad (1449)$$

- $$20 \quad \text{– The variables } \text{hvd1}, \text{hvd0}, \text{ are derived as follows:}$$

$$\text{hvd1} = (d1 * \text{hv0} > \text{hv1} * d0) ? d1 : \text{hv1} \quad (1450)$$

$$\text{hvd0} = (d1 * \text{hv0} > \text{hv1} * d0) ? d0 : \text{hv0} \quad (1451)$$

- The variables $\text{dirS}[x][y]$, $\text{dir1}[x][y]$ and $\text{dir2}[x][y]$ derived as follows:

$$\text{dir1}[x][y] = (d1 * \text{hv0} > \text{hv1} * d0) ? \text{dirD} : \text{dirHV} \quad (1452)$$

$$25 \quad \text{dir2}[x][y] = (d1 * \text{hv0} > \text{hv1} * d0) ? \text{dirHV} : \text{dirD} \quad (1453)$$

$$\text{dirS}[x][y] = (\text{hvd1} * 2 > 9 * \text{hvd0}) ? 2 : ((\text{hvd1} > 2 * \text{hvd0}) ? 1 : 0) \quad (1454)$$

The variable $\text{avgVar}[x][y]$ with $x, y = 0..CtbSizeY - 1$ is derived as follows:

$$\text{varTab}[] = \{ 0, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 4 \} \quad (1455)$$

$$\begin{aligned} \text{avgVar}[x][y] &= \text{varTab}[\text{Clip3}(0, 15, (\text{sumOfHV}[x \gg 2][y \gg 2] * \\ &\quad \text{ac}[x \gg 2][y \gg 2]) \gg (\text{BitDepth} - 1))] \end{aligned} \quad (1456)$$

5 The classification filter index array $\text{filtIdx}[x][y]$ and the transpose index array $\text{transposeIdx}[x][y]$ with $x = y = 0..\text{CtbSizeY} - 1$ are derived as follows:

$$\text{transposeTable}[] = \{ 0, 1, 0, 2, 2, 3, 1, 3 \}$$

$$\text{transposeIdx}[x][y] = \text{transposeTable}[\text{dir1}[x][y] * 2 + (\text{dir2}[x][y] \gg 1)]$$

$$\text{filtIdx}[x][y] = \text{avgVar}[x][y]$$

When $\text{dirS}[x][y]$ is not equal 0, $\text{filtIdx}[x][y]$ is modified as follows:

$$10 \quad \text{filtIdx}[x][y] += (((\text{dir1}[x][y] \& 0x1) \ll 1) + \text{dirS}[x][y]) * 5 \quad (1457)$$

8.8.5.4 Coding tree block filtering process for chroma samples

Inputs of this process are:

- a reconstructed chroma picture sample array recPicture prior to the adaptive loop filtering process,
- 15 – a filtered reconstructed chroma picture sample array alfPicture ,
- a chroma location $(x\text{CtbC}, y\text{CtbC})$ specifying the top-left sample of the current chroma coding tree block relative to the top left sample of the current picture,
- an alternative chroma filter index altIdx .

20 Output of this process is the modified filtered reconstructed chroma picture sample array alfPicture .

The width and height of the current chroma coding tree block ctbWidthC and ctbHeightC is derived as follows:

$$\text{ctbWidthC} = \text{CtbSizeY} / \text{SubWidthC} \quad (1458)$$

$$\text{ctbHeightC} = \text{CtbSizeY} / \text{SubHeightC} \quad (1459)$$

25 For the derivation of the filtered reconstructed chroma samples $\text{alfPicture}[x][y]$, each reconstructed chroma sample inside the current chroma coding tree block $\text{recPicture}[x][y]$ is filtered as follows with $x = 0..\text{ctbWidthC} - 1$, $y = 0..\text{ctbHeightC} - 1$:

- The locations (h_{x+i}, v_{y+j}) for each of the corresponding chroma samples (x, y) inside the given array recPicture of chroma samples with $i, j = -2..2$ are derived as follows:

$$30 \quad h_{x+i} = \text{Clip3}(0, \text{pic_width_in_luma_samples} / \text{SubWidthC} - 1, x\text{CtbC} + x + i) \quad (1460)$$

$$v_{y+j} = \text{Clip3}(0, \text{pic_height_in_luma_samples} / \text{SubHeightC} - 1, y\text{CtbC} + y + j) \quad (1461)$$

- The variables clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with (xCtbC * SubWidthC, yCtbC * SubHeightC) and (x * SubWidthC, y * SubHeightC) as inputs.
- The variables h_{x+i} and v_{y+j} are modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with (xCtb, yCtb), (h_{x+i} , v_{y+j}), the variable isChroma set equal to 1, clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag as input.
- The variable applyAlfLineBufBoundary is derived as follows:
 - If the bottom boundary of the current coding tree block is the bottom boundary of the picture and $\text{pic_height_in_luma_samples} - (y\text{CtbC} * \text{SubHeightC}) < \text{CtbSizeY} - 4$, applyAlfLineBufBoundary is set equal to 0.
 - Otherwise, applyAlfLineBufBoundary is set equal to 1.
- The vertical sample position offsets y1 and y2 are specified in table 43 according to the vertical chroma sample position y and applyAlfLineBufBoundary.
- The variable curr is derived as follows:

$$\text{curr} = \text{recPicture}[h_x][v_y] \quad (1462)$$

- The array of chroma filter coefficients f[j] and the array of chroma clipping values c[j] is derived as follows with $j = 0..5$:

$$f[j] = \text{AlfCoeffC}[\text{slice_alf_aps_id_chroma}][\text{altIdx}][j] \quad (1463)$$

$$c[j] = \text{AlfClipC}[\text{slice_alf_aps_id_chroma}][\text{altIdx}][j] \quad (1464)$$

- The variable sum is derived as follows:

$$\begin{aligned} \text{sum} = & f[0] * (\text{Clip3}(-c[0], c[0], \text{recPicture}[h_x][v_{y+y2}] - \text{curr}) + \\ & \text{Clip3}(-c[0], c[0], \text{recPicture}[h_x][v_{y-y2}] - \text{curr})) + \\ & f[1] * (\text{Clip3}(-c[1], c[1], \text{recPicture}[h_{x+1}][v_{y+y1}] - \text{curr}) + \\ & \text{Clip3}(-c[1], c[1], \text{recPicture}[h_{x-1}][v_{y-y1}] - \text{curr})) + \\ & f[2] * (\text{Clip3}(-c[2], c[2], \text{recPicture}[h_x][v_{y+y1}] - \text{curr}) + \\ & \text{Clip3}(-c[2], c[2], \text{recPicture}[h_x][v_{y-y1}] - \text{curr})) + \\ & f[3] * (\text{Clip3}(-c[3], c[3], \text{recPicture}[h_{x-1}][v_{y+y1}] - \text{curr}) + \\ & \text{Clip3}(-c[3], c[3], \text{recPicture}[h_{x+1}][v_{y-y1}] - \text{curr})) + \\ & f[4] * (\text{Clip3}(-c[4], c[4], \text{recPicture}[h_{x+2}][v_y] - \text{curr}) + \end{aligned} \quad (1465)$$

$$\begin{aligned} & \text{Clip3}(-c[4], c[4], \text{recPicture}[h_{x-2}][v_y] - \text{curr}) + \\ & f[5] * (\text{Clip3}(-c[5], c[5], \text{recPicture}[h_{x+1}][v_y] - \text{curr}) + \\ & \text{Clip3}(-c[5], c[5], \text{recPicture}[h_{x-1}][v_y] - \text{curr})) \end{aligned}$$

$$\text{sum} = \text{curr} + ((\text{sum} + 64) >> 7) \quad (1466)$$

- 5 – The modified filtered reconstructed chroma picture sample $\text{alfPicture}[x_{\text{CtbC}} + x][y_{\text{CtbC}} + y]$ is derived as follows:

$$\text{alfPicture}[x_{\text{CtbC}} + x][y_{\text{CtbC}} + y] = \text{Clip3}(0, (1 << \text{BitDepth}) - 1, \text{sum}) \quad (1467)$$

Table 44 – Specification of y1 and y2 according to the vertical chroma sample position y and applyAlfLineBufBoundary

Condition	y1	y2
$(y == \text{ctbHeightC} - 2 \mid \mid y == \text{ctbHeightC} - 3) \ \&\& \ (\text{applyAlfLineBufBoundary} == 1)$	0	0
$(y == \text{ctbHeightC} - 1 \mid \mid y == \text{ctbHeightC} - 4) \ \&\& \ (\text{applyAlfLineBufBoundary} == 1)$	1	1
Otherwise	1	2

10

8.8.5.5 ALF boundary position derivation process

Inputs of this process are:

- a luma location $(x_{\text{Ctb}}, y_{\text{Ctb}})$ specifying the top-left sample of the current luma coding tree block relative to the top left sample of the current picture,
- 15 – a luma location (x, y) specifying the current sample relative to the top-left sample of the current luma coding tree block.

Output of this process are:

- the left vertical boundary position clipLeftPos,
- the right vertical boundary position clipRightPos,
- 20 – the above horizontal boundary position clipTopPos,
- the below horizontal boundary position clipBottomPos,
- the top left boundary flag clipTopLeftFlag,
- the bottom right boundary flag clipBotRightFlag.

The variables clipLeftPos, clipRightPos, clipTopPos and clipBottomPos are set equal to
25 –128.

The variables clipTopLeftFlag and clipBotRightFlag are both set equal to 0.

The variable clipTopPos is modified as follows:

- If $y - (CtbSizeY - 4)$ is greater than or equal to 0, the variable clipTopPos is set equal to $yCtb + CtbSizeY - 4$.
- Otherwise, if VirtualBoundariesDisabledFlag is equal to 1, and
5 $yCtb + y - VirtualBoundariesPosY[n]$ is greater than or equal to 0 and less than 3 for any $n = 0..VirtualBoundariesNumHor - 1$, the following applies:

$$clipTopPos = VirtualBoundariesPosY[n] \quad (1468)$$

- Otherwise, if y is less than 3 and one or more of the following conditions are true, the variable clipTopPos is set equal to yCtb:
- 10 – The top boundary of the current coding tree block is the top boundary of the tile, and loop_filter_across_tiles_enabled_flag is equal to 0.
- The top boundary of the current coding tree block is the top boundary of the slice, and loop_filter_across_slices_enabled_flag is equal to 0.
- The top boundary of the current coding tree block is the top boundary of the
15 subpicture, and loop_filter_across_subpic_enabled_flag[SubPicIdx] is equal to 0.

The variable clipBottomPos is modified as follows:

- If VirtualBoundariesDisabledFlag is equal to 1, VirtualBoundariesPosY[n] is not equal to $pic_height_in_luma_samples - 1$ or 0, and $VirtualBoundariesPosY[n] - yCtb - y$ is greater than 0 and less than 5 for any $n = 0..VirtualBoundariesNumHor - 1$, the following
20 applies:

$$clipBottomPos = VirtualBoundariesPosY[n] \quad (1469)$$

- Otherwise, if $CtbSizeY - 4 - y$ is greater than 0 and is less than 5, the variable clipBottomPos is set equal to $yCtb + CtbSizeY - 4$.
- Otherwise, if $CtbSizeY - y$ is less than 5, and one or more of the following conditions are
25 true, the variable clipBottomPos is set equal to $yCtb + CtbSizeY$:
- The bottom boundary of the current coding tree block is the bottom boundary of the tile, and loop_filter_across_tiles_enabled_flag is equal to 0.
- The bottom boundary of the current coding tree block is the bottom boundary of the slice, and loop_filter_across_slices_enabled_flag is equal to 0.
- 30 – The bottom boundary of the current coding tree block is the bottom boundary of the subpicture, and loop_filter_across_subpic_enabled_flag[SubPicIdx] is equal to 0.

The variable clipLeftPos is modified as follows:

- If `VirtualBoundariesDisabledFlag` is equal to 1, and $x_{Ctb} + x - \text{VirtualBoundariesPosX}[n]$ is greater than or equal to 0 and less than 3 for any $n = 0..VirtualBoundariesNumVer - 1$, the following applies:

`clipLeftPos = VirtualBoundariesPosX[ n ]` (1470)

- 5 – Otherwise, if x is less than 3, and one or more of the following conditions are true, the variable `clipLeftPos` is set equal to x_{Ctb} :
- The left boundary of the current coding tree block is the left boundary of the tile, and `loop_filter_across_tiles_enabled_flag` is equal to 0.
 - The left boundary of the current coding tree block is the left boundary of the slice, and
10 `loop_filter_across_slices_enabled_flag` is equal to 0.
 - The left boundary of the current coding tree block is the left boundary of the subpicture, and `loop_filter_across_subpic_enabled_flag[ SubPicIdx ]` is equal to 0.

The variable `clipRightPos` is modified as follows:

- 15 If `VirtualBoundariesDisabledFlag` is equal to 1, and $\text{VirtualBoundariesPosX}[n] - x_{Ctb} - x$ is greater than 0 and less than 5 for any $n = 0..VirtualBoundariesNumVer - 1$, the following applies:

`clipRightPos = VirtualBoundariesPosX[ n ]` (1471)

Otherwise, if $CtbSizeY - x$ is less than 5, and one or more of the following conditions are true, the variable `clipRightPos` is set equal to $x_{Ctb} + CtbSizeY$:

- 20 The right boundary of the current coding tree block is the right boundary of the tile, and `loop_filter_across_tiles_enabled_flag` is equal to 0.

The right boundary of the current coding tree block is the right boundary of the slice, and `loop_filter_across_slices_enabled_flag` is equal to 0.

- 25 The right boundary of the current coding tree block is the right boundary of the subpicture, and `loop_filter_across_subpic_enabled_flag[ SubPicIdx ]` is equal to 0.

The variable `clipTopLeftFlag` and `clipBotRightFlag` are modified as following:

- 30 If the coding tree block covering the luma position (x_{Ctb}, y_{Ctb}) and the coding tree block covering the luma position $(x_{Ctb} - CtbSizeY, y_{Ctb} - CtbSizeY)$ belong to different slices, and `loop_filter_across_slices_enabled_flag` is equal to 0, `clipTopLeftFlag` is set equal to 1.

If the coding tree block covering the luma position (x_{Ctb}, y_{Ctb}) and the coding tree block covering the luma position $(x_{Ctb} + CtbSizeY, y_{Ctb} + CtbSizeY)$ belong to

different slices, and loop_filter_across_slices_enabled_flag is equal to 0, clipBotRightFlag is set equal to 1.

8.8.5.6 ALF sample padding process

Inputs of this process are:

- 5 a luma location (xCtb, yCtb) specifying the top-left sample of the current luma coding tree block relative to the top left sample of the current picture,
- a luma location (x, y) specifying the neighboring sample relative to the top-left sample of the current picture,
- a flag isChroma specifying whether the colour component is chroma component or not,
- 10 the left vertical boundary position clipLeftPos,
- the right vertical boundary position clipRightPos,
- the above horizontal boundary position clipTopPos,
- the below horizontal boundary position clipBottomPos,
- the top left boundary flag clipTopLeftFlag,
- 15 the bottom right boundary flag clipBotRightFlag.

Outputs of this process are:

modified luma location (x, y) specifying the neighboring sample relative to the top-left sample of the current picture,

- The variables picWidth, picHeight, xCtbCur, yCtbCur, CtbSizeHor, CtbSizeVer, topBry, botBry, leftBry and rightBry are derived as follows:
- 20

picWidth = isChroma ? pic_width_in_luma_samples / SubWidthC :
pic_width_in_luma_samples
(1472)

picHeight = isChroma ? pic_height_in_luma_samples / SubHeightC :
25 pic_height_in_luma_samples
(1473)

xCtbCur = isChroma ? xCtb / SubWidthC : xCtb (1474)

yCtbCur = isChroma ? yCtb / SubHeightC : yCtb (1475)

ctbSizeHor = isChroma ? CtbSizeY / SubWidthC : CtbSizeY (1476)

30 ctbSizeVer = isChroma ? CtbSizeY / SubHeightC : CtbSizeY (1477)

topBryPos = isChroma ? clipTopPos / SubHeightC : clipTopPos (1478)

botBryPos = isChroma ? clipBottomPos / SubHeightC : clipBottomPos (1479)

leftBryPos = isChroma ? clipLeftPos / SubWidthC : clipLeftPos (1480)

rightBryPos = isChroma ? clipRightPos / SubWidthC : clipRightPos (1481)

5 The variables (x , y) is modified as follows:

When topBryPos is not less than 0, the following applies:

y = Clip3(topBryPos, picHeight - 1, y) (1482)

When botBryPos is not less than 0, the following applies:

y = Clip3(0, botBryPos - 1, y) (1483)

10 When leftBryPos is not less than 0, the following applies:

x = Clip3(leftBryPos, picWidth - 1, x) (1484)

When rightBryPos is not less than 0, the following applies:

x = Clip3(0, rightBryPos - 1, x) (1485)

(x, y) is set equal to (xCtbCur, y) if all of the followig conditions are true:

15 clipTopLeftFlag is equal to true

topBryPos is less than 0 and leftBryPos is less than 0

x is less than xCtbCur and y is less than yCtbCur

(x, y) is set equal to (xCtbCur + CtbSizeHor - 1, y) if all of the followig conditions are true:

20 clipBotRightFlag is equal to true

botBryPos is less than 0 and rightBryPos is less than 0

x is greater than xCtbCur + CtbSizeHor - 1 and y is greater than yCtbCur + CtbSizeVer - 1.

In an example, ITU JVET-Q0150 proposes to modify the ALF filtering process, especially at

25 the CTU boundaries.

The main changes are to changes to the equations 1425, 1467 (see equations above) and tables 43 and 44 (see tables above).

As disclosed in JVET-Q0150, the ALF virtual boundary processing is specified as follows:

$$1. \text{sum} = \text{curr} + ((\text{sum} + 64) \gg \text{alfShiftY}). \quad (1425)$$

- 5 Normally, Curr means the center pixel value which is currently being filtered. y1, y2 and y3 are vertical Luma sample positions (y) used in ALF filtering. CtbSizeY is the CTB size.

Table 43 – Specification of y1, y2, y3 and alfShiftY according to the vertical luma sample position y and applyAlfLineBufBoundary

Condition	alfShiftY	y1	y2	y3
(y == CtbSizeY - 5 y == CtbSizeY - 4) && (applyAlfLineBufBoundary == 1)	10	0	0	0
(y == CtbSizeY - 6 y == CtbSizeY - 3) && (applyAlfLineBufBoundary == 1)	7	1	1	1
(y == CtbSizeY - 7 y == CtbSizeY - 2) && (applyAlfLineBufBoundary == 1)	7	1	2	2
Otherwise	7	1	2	3

10

$$2. \text{sum} = \text{curr} + ((\text{sum} + 64) \gg \text{alfShiftC}). \quad (1466)$$

Table 44 – Specification of y1, y2 and alfShiftC according to the vertical chroma sample position y and applyAlfLineBufBoundary

Condition	alfShiftC	y1	y2
(y == ctbHeightC - 2 y == ctbHeightC - 3) && (applyAlfLineBufBoundary == 1)	10	0	0
(y == ctbHeightC - 1 y == ctbHeightC - 4) && (applyAlfLineBufBoundary == 1)	7	1	1
Otherwise	7	1	2

- In equations 1425 and 1466, the filtered value during the ALF filter process is divided or
 15 (right shifted) by a value (alfShiftY or alfShiftC) which is dependent on how far the current center pixel (specified by value y) is from the ALF virtual boundary. Normally, alfShiftY is

used for Luma and alfShiftC is used for Chroma component. Therefore, the value of alfShiftY or alfShiftC may be either 10 or 7, depending on the value of y as per table 43, table 44 for Luma and Chroma ALF filtering respectively.

However, according to the above two equations, the rounding value which is used in the ALF filtering process is always the same (64) even if the right shift value is 7 or 10.

Embodiments of the present invention proposed to use a rounding value corresponding to the right shift value during the ALF filtering process. This solution gives an advantage that the ALF filtering is precise and consistent when compared with the other filtering stages.

The specification changes are as follows:

$$10 \quad 1.\text{sum} = \text{curr} + ((\text{sum} + \text{roundY}) \gg \text{alfShiftY}) \quad (1425)$$

Table 43 – Specification of y1, y2, y3, alfShiftY and roundY according to the vertical luma sample position y and applyAlfLineBufBoundary

Condition	roundY	alfShiftY	y1	y2	y3
(y == CtbSizeY - 5 y == CtbSizeY - 4) && (applyAlfLineBufBoundary == 1)	512	10	0	0	0
(y == CtbSizeY - 6 y == CtbSizeY - 3) && (applyAlfLineBufBoundary == 1)	64	7	1	1	1
(y == CtbSizeY - 7 y == CtbSizeY - 2) && (applyAlfLineBufBoundary == 1)	64	7	1	2	2
Otherwise	64	7	1	2	3

$$2.\text{sum} = \text{curr} + ((\text{sum} + \text{roundC}) \gg \text{alfShiftC}) \quad (1466)$$

15 Table 44 – Specification of y1, y2, alfShiftC and roundC according to the vertical chroma sample position y and applyAlfLineBufBoundary

Condition	roundC	alfShiftC	y1	y2
-----------	--------	-----------	----	----

(y == ctbHeightC - 2 y == ctbHeightC - 3) && (applyAlfLineBufBoundary == 1)	512	10	0	0
(y == ctbHeightC - 1 y == ctbHeightC - 4) && (applyAlfLineBufBoundary == 1)	64	7	1	1
Otherwise	64	7	1	2

In some implementations, the samples are processed in units of CTBs. The array size for each luma CTB in both width and height is CtbSizeY in units of samples. The width and height of the array for each chroma CTB are CtbWidthC and CtbHeightC, respectively, in units of samples.

- 5 Two new variables **roundY** and **roundC** are used in equations 1425 and 1466 respectively. The values are specified in tables 43 and 44 respectively.

In an implementation as shown in Fig.25, a method of filtering a sample value of a picture is disclosed, the method comprising:

S2501: obtaining a reconstructed sample value for a block of the picture.

- 10 In an example, inputs of adaptive loop filter process are the reconstructed picture sample array prior to adaptive loop filter recPictureL and, when sps_chroma_format_idc is not equal to 0, the arrays recPictureCb and recPictureCr.

The sample values in the modified reconstructed picture sample array after adaptive loop filter alfPictureL and, when sps_chroma_format_idc is not equal to 0, the arrays alfPictureCb

- 15 and alfPictureCr are initially set equal to the sample values in the reconstructed picture sample array prior to adaptive loop filter recPictureL and, when sps_chroma_format_idc is not equal to 0, the arrays recPictureCb and recPictureCr, respectively.

sps_chroma_format_idc specifies the chroma sampling relative to the luma sampling.

sps_chroma_format_idc	Chroma format
0	Monochrome
1	4:2:0
2	4:2:2
3	4:4:4

S2502: obtaining filter coefficients for adaptive loop filtering according to a bitstream.

adaptive loop filter (ALF) is a filtering process that is applied as part of the decoding process and is controlled by parameters conveyed in an APS.

- 5 adaptation parameter set (APS) is a syntax structure containing syntax elements that apply to zero or more slices as determined by zero or more syntax elements found in slice headers.

The derivation process for filter index is invoked with the location (x_{Ctb} , y_{Ctb}) and the reconstructed luma picture sample array $recPicture$ as inputs, and $filtIdx[x][y]$ and $transposeIdx[x][y]$ with $x, y = 0..CtbSizeY - 1$ as outputs.

- 10 The bitstream may be obtained according to wireless network or wired network. The bitstream may be transmitted from a website, server, or other remote source using a coaxial cable, fiber optic cable, twisted pair, digital subscriber line (DSL), or wireless technologies such as infrared, radio, microwave, WIFI, Bluetooth, LTE or 5G.

In an embodiment, a bitstream are a sequence of bits, in the form of a network abstraction

- 15 layer (NAL) unit stream or a byte stream, that forms the representation of a sequence of access units (AUs) forming one or more coded video sequences (CVSs).

In some embodiments, for a decoding process, decoder side reads a bitstream and derives decoded pictures from the bitstream; for an encoding process, encoder side produces a bitstream.

- 20 Normally, a bitstream will comprise syntax elements that are formed by a syntax structure.
syntax element: An element of data represented in the bitstream.

syntax structure: Zero or more syntax elements present together in the bitstream in a specified order.

In a specific example, bitstream formats specifies the relationship between the network abstraction layer (NAL) unit stream and byte stream, either of which are referred to as the
5 bitstream.

The bitstream can be in one of two formats: the NAL unit stream format or the byte stream format. The NAL unit stream format is conceptually the more "basic" type. The NAL unit stream format comprises a sequence of syntax structures called NAL units. This sequence is ordered in decoding order. There are constraints imposed on the decoding order (and contents)
10 of the NAL units in the NAL unit stream.

The byte stream format can be constructed from the NAL unit stream format by ordering the NAL units in decoding order and prefixing each NAL unit with a start code prefix and zero or more zero-valued bytes to form a stream of bytes. The NAL unit stream format can be extracted from the byte stream format by searching for the location of the unique start code
15 prefix pattern within this stream of bytes.

This clause specifies the relationship between source and decoded pictures that is given via the bitstream.

The video source that is represented by the bitstream is a sequence of pictures in decoding order.

20 The source and decoded pictures are each comprised of one or more sample arrays:

- Luma (Y) only (monochrome).
- Luma and two chroma (YCbCr or YCgCo).
- Green, blue, and red (GBR, also known as RGB).
- Arrays representing other unspecified monochrome or tri-stimulus colour samplings (for
25 example, YZX, also known as XYZ).

The variables and terms associated with these arrays are referred to as luma (or L or Y) and chroma, where the two chroma arrays are referred to as Cb and Cr; regardless of the actual colour representation method in use. The actual colour representation method in use can be indicated in syntax that is specified in VUI parameters as specified in ITU-T H.265 | ISO/IEC 23002-7.

S2503: obtaining a sum value according to the filter coefficients and the reconstructed sample value for the block.

In an example,

- 10 – The variable curr is derived as follows:

$$\text{curr} = \text{recPicture}[h_x][v_y]$$

The variable sum is derived as follows:

$$\begin{aligned} \text{sum} = & f[\text{idx}[0]] * (\text{Clip3}(-c[\text{idx}[0]], c[\text{idx}[0]], \text{recPicture}[h_x][v_y + y_3] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[0]], c[\text{idx}[0]], \text{recPicture}[h_x][v_y - y_3] - \text{curr})) + \\ 15 \quad & f[\text{idx}[1]] * (\text{Clip3}(-c[\text{idx}[1]], c[\text{idx}[1]], \text{recPicture}[h_x + 1][v_y + y_2] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[1]], c[\text{idx}[1]], \text{recPicture}[h_x - 1][v_y - y_2] - \text{curr})) + \\ & f[\text{idx}[2]] * (\text{Clip3}(-c[\text{idx}[2]], c[\text{idx}[2]], \text{recPicture}[h_x][v_y + y_2] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[2]], c[\text{idx}[2]], \text{recPicture}[h_x][v_y - y_2] - \text{curr})) + \\ & f[\text{idx}[3]] * (\text{Clip3}(-c[\text{idx}[3]], c[\text{idx}[3]], \text{recPicture}[h_x - 1][v_y + y_2] - \text{curr}) + \\ 20 \quad & \text{Clip3}(-c[\text{idx}[3]], c[\text{idx}[3]], \text{recPicture}[h_x + 1][v_y - y_2] - \text{curr})) + \\ & f[\text{idx}[4]] * (\text{Clip3}(-c[\text{idx}[4]], c[\text{idx}[4]], \text{recPicture}[h_x + 2][v_y + y_1] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[4]], c[\text{idx}[4]], \text{recPicture}[h_x - 2][v_y - y_1] - \text{curr})) + \\ & f[\text{idx}[5]] * (\text{Clip3}(-c[\text{idx}[5]], c[\text{idx}[5]], \text{recPicture}[h_x + 1][v_y + y_1] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[5]], c[\text{idx}[5]], \text{recPicture}[h_x - 1][v_y - y_1] - \text{curr})) + \\ 25 \quad & f[\text{idx}[6]] * (\text{Clip3}(-c[\text{idx}[6]], c[\text{idx}[6]], \text{recPicture}[h_x][v_y + y_1] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[6]], c[\text{idx}[6]], \text{recPicture}[h_x][v_y - y_1] - \text{curr})) + \\ & f[\text{idx}[7]] * (\text{Clip3}(-c[\text{idx}[7]], c[\text{idx}[7]], \text{recPicture}[h_x - 1][v_y + y_1] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[7]], c[\text{idx}[7]], \text{recPicture}[h_x + 1][v_y - y_1] - \text{curr})) + \\ & f[\text{idx}[8]] * (\text{Clip3}(-c[\text{idx}[8]], c[\text{idx}[8]], \text{recPicture}[h_x - 2][v_y + y_1] - \text{curr}) + \\ 30 \quad & \text{Clip3}(-c[\text{idx}[8]], c[\text{idx}[8]], \text{recPicture}[h_x + 2][v_y - y_1] - \text{curr})) + \\ & f[\text{idx}[9]] * (\text{Clip3}(-c[\text{idx}[9]], c[\text{idx}[9]], \text{recPicture}[h_x + 3][v_y] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[9]], c[\text{idx}[9]], \text{recPicture}[h_x - 3][v_y] - \text{curr})) + \end{aligned}$$

$$\begin{aligned}
& f[idx[10]] * (Clip3(-c[idx[10]], c[idx[10]], recPicture[h_{x+2}][v_y] - curr) + \\
& Clip3(-c[idx[10]], c[idx[10]], recPicture[h_{x-2}][v_y] - curr)) + \\
& f[idx[11]] * (Clip3(-c[idx[11]], c[idx[11]], recPicture[h_{x+1}][v_y] - curr) + \\
& Clip3(-c[idx[11]], c[idx[11]], recPicture[h_{x-1}][v_y] - curr))
\end{aligned}$$

5 In another example,

- The variable curr is derived as follows:

$$curr = recPicture[h_x][v_y]$$

- The array of chroma filter coefficients $f[j]$ and the array of chroma clipping values $c[j]$ is derived as follows with $j = 0..5$:

$$10 \quad f[j] = AlfCoeffC[slice_alf_aps_id_chroma][altIdx][j]$$

$$c[j] = AlfClipC[slice_alf_aps_id_chroma][altIdx][j]$$

- The variable sum is derived as follows:

$$\begin{aligned}
& sum = f[0] * (Clip3(-c[0], c[0], recPicture[h_x][v_{y+y2}] - curr) + \\
& \quad Clip3(-c[0], c[0], recPicture[h_x][v_{y-y2}] - curr)) + \\
15 \quad & f[1] * (Clip3(-c[1], c[1], recPicture[h_{x+1}][v_{y+y1}] - curr) + \\
& \quad Clip3(-c[1], c[1], recPicture[h_{x-1}][v_{y-y1}] - curr)) + \\
& f[2] * (Clip3(-c[2], c[2], recPicture[h_x][v_{y+y1}] - curr) + \\
& \quad Clip3(-c[2], c[2], recPicture[h_x][v_{y-y1}] - curr)) + \\
& f[3] * (Clip3(-c[3], c[3], recPicture[h_{x-1}][v_{y+y1}] - curr) + \\
20 \quad & \quad Clip3(-c[3], c[3], recPicture[h_{x+1}][v_{y-y1}] - curr)) + \\
& f[4] * (Clip3(-c[4], c[4], recPicture[h_{x+2}][v_y] - curr) + \\
& \quad Clip3(-c[4], c[4], recPicture[h_{x-2}][v_y] - curr)) + \\
& f[5] * (Clip3(-c[5], c[5], recPicture[h_{x+1}][v_y] - curr) + \\
& \quad Clip3(-c[5], c[5], recPicture[h_{x-1}][v_y] - curr))
\end{aligned}$$

25 S2504: rounding the sum value according to a vertical position of a sample in the block, to obtain a rounded sum value.

In an example, the rounding process comprises: adding value which is determined based on the vertical position into the sum value to obtain an added sum value, and shifting the added sum value based on the vertical position.

In an example, the rounded sum value is equal to $(\text{sum} + (1 \ll (\text{alfShiftY} - 1))) \gg \text{alfShiftY}$, wherein sum is the sum value, alfShiftY is a variable which is determined based on the vertical position of the sample.

In an example, the rounded sum value is equal to $(\text{sum} + (1 \ll (\text{alfShiftC} - 1))) \gg \text{alfShiftC}$, wherein sum is the sum value, alfShiftC is a variable which is determined based on the vertical position of the sample.

It could be understood that, alfShiftC is a variable for chroma sample position, alfShiftY is a variable for luma sample position, in the present disclosure, these two variables can be used interchangeably.

In an example, wherein alfShiftY is equal to 10 when the vertical position of the sample is satisfied any one of the below conditions:

$y == \text{CtbSizeY} - t - 1$, or $y == \text{CtbSizeY} - t$, or $y == \text{ctbHeightC} - t$, or $y == \text{ctbHeightC} - t - 1$, wherein y is the vertical position of the sample, CtbSizeY is a coding tree block, CTB, size, ctbHeightC is the height of the CTB, t is an integer value. In some examples, t is 2, 3, 4, 5, 6, and so on.

In an example, alfShiftY is equal to 7 when the vertical position of the sample is not satisfied any one of the below conditions:

$y == \text{CtbSizeY} - t - 1$, or $y == \text{CtbSizeY} - t$, or $y == \text{ctbHeightC} - t$, or $y == \text{ctbHeightC} - t - 1$, wherein y is the vertical position of the sample, CtbSizeY is a coding tree block, CTB, size, ctbHeightC is the height of the CTB, t is an integer value.

In an example, when y is equal to 0, alfShiftY is equal to 10.

In an example, wherein when y is not equal to 0, alfShiftY is equal to 7.

In an example, the vertical position of the sample in the block is a chroma sample position or is a luma sample position.

Table 43 – Specification of y1, y2, y3 and alfShiftY according to the vertical luma sample position y and applyAlfLineBufBoundary

Condition	alfShiftY	y1	y2	y3
(y == CtbSizeY - 5 y == CtbSizeY - 4) && (applyAlfLineBufBoundary == 1)	10	0	0	0
(y == CtbSizeY - 6 y == CtbSizeY - 3) && (applyAlfLineBufBoundary == 1)	7	1	1	1
(y == CtbSizeY - 7 y == CtbSizeY - 2) && (applyAlfLineBufBoundary == 1)	7	1	2	2
Otherwise	7	1	2	3

Or

Table 44 – Specification of y1, y2 and alfShiftC according to the vertical chroma sample position y and applyAlfLineBufBoundary

Condition	alfShiftC	y1	y2
(y == ctbHeightC - 2 y == ctbHeightC - 3) && (applyAlfLineBufBoundary == 1)	10	0	0
(y == ctbHeightC - 1 y == ctbHeightC - 4) && (applyAlfLineBufBoundary == 1)	7	1	1
Otherwise	7	1	2

- 5 S2605: obtaining a filter reconstructed sample value for the block according to the rounded sum value.

Outputs of ALF process are the modified reconstructed picture sample array after adaptive loop filter `alfPictureL` and, when `sps_chroma_format_idc` is not equal to 0, the arrays `ccAlfPictureCb` and `ccAlfPictureCr`.

- 10 In an example,

Coding tree block filtering process for luma samples

Inputs of this process are:

- a reconstructed luma picture sample array `recPicture` prior to the adaptive loop filtering process,
- 15 – a filtered reconstructed luma picture sample array `alfPictureL`,

- a luma location (xCtb, yCtb) specifying the top-left sample of the current luma coding tree block relative to the top-left sample of the current picture.

Output of this process is the modified filtered reconstructed luma picture sample array alfPicture_L .

- 5 The derivation process for filter index is invoked with the location (xCtb, yCtb) and the reconstructed luma picture sample array recPicture as inputs, and $\text{filtIdx}[x][y]$ and $\text{transposeIdx}[x][y]$ with $x, y = 0..CtbSizeY - 1$ as outputs.

For the derivation of the filtered reconstructed luma samples $\text{alfPicture}_L[xCtb + x][yCtb + y]$, each reconstructed luma sample inside the current luma coding tree block $\text{recPicture}[xCtb + x][yCtb + y]$ is filtered as follows with $x, y = 0..CtbSizeY - 1$:

- The array of luma filter coefficients $f[j]$ and the array of luma clipping values $c[j]$ corresponding to the filter specified by $\text{filtIdx}[x][y]$ is derived as follows with $j = 0..11$:
 - If $\text{AlfCtbFiltSetIdxY}[xCtb \gg CtbLog2SizeY][yCtb \gg CtbLog2SizeY]$ is less than 16, the following applies:

$i = \text{AlfCtbFiltSetIdxY}[xCtb \gg CtbLog2SizeY][yCtb \gg CtbLog2SizeY]$

$f[j] = \text{AlfFixFiltCoeff}[\text{AlfClassToFiltMap}[i][\text{filtIdx}[x][y]]][j]$

$c[j] = 2^{\text{BitDepth}}$

- Otherwise $(\text{AlfCtbFiltSetIdxY}[xCtb \gg CtbLog2SizeY][yCtb \gg CtbLog2SizeY])$ is greater than or equal to 16, the following applies:

$i = \text{sh_alf_aps_id_luma}[\text{AlfCtbFiltSetIdxY}[xCtb \gg CtbLog2SizeY][yCtb \gg CtbLog2SizeY] - 16]$

$f[j] = \text{AlfCoeff}_L[i][\text{filtIdx}[x][y]][j]$

$c[j] = \text{AlfClip}_L[i][\text{filtIdx}[x][y]][j]$

- 25 – The luma filter coefficients and clipping values index idx are derived depending on $\text{transposeIdx}[x][y]$ as follows:

- If $\text{transposeIdx}[x][y]$ is equal to 1, the following applies:

$\text{idx}[] = \{ 9, 4, 10, 8, 1, 5, 11, 7, 3, 0, 2, 6 \}$

- Otherwise, if $\text{transposeIdx}[x][y]$ is equal to 2, the following applies:

30 $\text{idx}[] = \{ 0, 3, 2, 1, 8, 7, 6, 5, 4, 9, 10, 11 \}$

- Otherwise, if $\text{transposeIdx}[x][y]$ is equal to 3, the following applies:

$\text{idx}[] = \{ 9, 8, 10, 4, 3, 7, 11, 5, 1, 0, 2, 6 \}$

- Otherwise, the following applies:

$\text{idx}[] = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 \}$

- 5 – The locations (h_{x+i}, v_{y+j}) for each of the corresponding luma samples (x, y) inside the given array recPicture of luma samples with $i, j = -3..3$ are derived as follows:

$$h_{x+i} = \text{Clip3}(0, \text{pps_pic_width_in_luma_samples} - 1, x_{\text{Ctb}} + x + i)$$

$$v_{y+j} = \text{Clip3}(0, \text{pps_pic_height_in_luma_samples} - 1, y_{\text{Ctb}} + y + j)$$

- 10 – The variables clipLeftPos , clipRightPos , clipTopPos , clipBottomPos , clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process with $(x_{\text{Ctb}}, y_{\text{Ctb}})$, (x, y) and the variable vbOffset set equal to 4 as inputs.

- 15 – The variables h_{x+i} and v_{y+j} are modified by invoking the ALF sample padding process with $(x_{\text{Ctb}}, y_{\text{Ctb}})$, (h_{x+i}, v_{y+j}) , the variable isChroma set equal to 0, clipLeftPos , clipRightPos , clipTopPos , clipBottomPos , clipTopLeftFlag and clipBotRightFlag as inputs.

- The variable $\text{applyAlfLineBufBoundary}$ is derived as follows:

- If the bottom boundary of the current coding tree block is the bottom boundary of current picture and $\text{pps_pic_height_in_luma_samples} - y_{\text{Ctb}} \leq \text{CtbSizeY} - 4$, $\text{applyAlfLineBufBoundary}$ is set equal to 0:

- 20 – Otherwise, $\text{applyAlfLineBufBoundary}$ is set equal to 1.

- The vertical sample position offsets y_1, y_2, y_3 and the variable alfShiftY are specified in Table 45 according to the vertical luma sample position y and $\text{applyAlfLineBufBoundary}$.
- The variable curr is derived as follows:

$$\text{curr} = \text{recPicture}[h_x][v_y]$$

- The variable sum is derived as follows:

```

sum = f[ idx[ 0 ] ] * ( Clip3( -c[ idx[ 0 ] ], c[ idx[ 0 ] ],    recPicture[ hx ][ vy+y3 ] -
curr ) +

Clip3( -c[ idx[ 0 ] ], c[ idx[ 0 ] ],    recPicture[ hx ][ vy-y3 ] - curr ) ) +
5      f[ idx[ 1 ] ] * ( Clip3( -c[ idx[ 1 ] ], c[ idx[ 1 ] ],    recPicture[ hx+1 ][ vy+y2 ]
- curr ) +

Clip3( -c[ idx[ 1 ] ], c[ idx[ 1 ] ],    recPicture[ hx-1 ][ vy-y2 ] - curr ) ) +
10      f[ idx[ 2 ] ] * ( Clip3( -c[ idx[ 2 ] ], c[ idx[ 2 ] ],    recPicture[ hx ][ vy+y2 ] - c
urr ) +

Clip3( -c[ idx[ 2 ] ], c[ idx[ 2 ] ],    recPicture[ hx ][ vy-y2 ] - curr ) ) +
15      f[ idx[ 3 ] ] * ( Clip3( -c[ idx[ 3 ] ], c[ idx[ 3 ] ],    recPicture[ hx-1 ][ vy+y2 ]
- curr ) +

Clip3( -c[ idx[ 3 ] ], c[ idx[ 3 ] ],    recPicture[ hx+1 ][ vy-y2 ] - curr ) ) +
20      f[ idx[ 4 ] ] * ( Clip3( -c[ idx[ 4 ] ], c[ idx[ 4 ] ],    recPicture[ hx+2 ][ vy+y1 ]
- curr ) +

Clip3( -c[ idx[ 4 ] ], c[ idx[ 4 ] ],    recPicture[ hx-2 ][ vy-y1 ] - curr ) ) +
25      f[ idx[ 5 ] ] * ( Clip3( -c[ idx[ 5 ] ], c[ idx[ 5 ] ],    recPicture[ hx+1 ][ vy+y1 ]
- curr ) +

Clip3( -c[ idx[ 5 ] ], c[ idx[ 5 ] ],    recPicture[ hx-1 ][ vy-y1 ] - curr ) ) +
30      f[ idx[ 6 ] ] * ( Clip3( -c[ idx[ 6 ] ], c[ idx[ 6 ] ],    recPicture[ hx ][ vy+y1 ] - c
urr ) +

Clip3( -c[ idx[ 6 ] ], c[ idx[ 6 ] ],    recPicture[ hx ][ vy-y1 ] - curr ) ) +
35      f[ idx[ 7 ] ] * ( Clip3( -c[ idx[ 7 ] ], c[ idx[ 7 ] ],    recPicture[ hx-1 ][ vy+y1 ]
- curr ) +

Clip3( -c[ idx[ 7 ] ], c[ idx[ 7 ] ],    recPicture[ hx+1 ][ vy-y1 ] - curr ) ) +
      f[ idx[ 8 ] ] * ( Clip3( -c[ idx[ 8 ] ], c[ idx[ 8 ] ],    recPicture[ hx-2 ][ vy+y1 ]
- curr ) +

Clip3( -c[ idx[ 8 ] ], c[ idx[ 8 ] ],    recPicture[ hx+2 ][ vy-y1 ] - curr ) ) +
      f[ idx[ 9 ] ] * ( Clip3( -c[ idx[ 9 ] ], c[ idx[ 9 ] ],    recPicture[ hx+3 ][ vy ] - cu
rr ) +

```

$$\text{Clip3}(-c[\text{idx}[9]], c[\text{idx}[9]], \text{recPicture}[\text{h}_{x-3}][v_y] - \text{curr})) +$$

$$f[\text{idx}[10]] * (\text{Clip3}(-c[\text{idx}[10]], c[\text{idx}[10]], \text{recPicture}[\text{h}_{x+2}][v_y] - \text{curr})) +$$
5
$$\text{Clip3}(-c[\text{idx}[10]], c[\text{idx}[10]], \text{recPicture}[\text{h}_{x-2}][v_y] - \text{curr})) +$$

$$f[\text{idx}[11]] * (\text{Clip3}(-c[\text{idx}[11]], c[\text{idx}[11]], \text{recPicture}[\text{h}_{x+1}][v_y] - \text{curr})) +$$
10
$$\text{Clip3}(-c[\text{idx}[11]], c[\text{idx}[11]], \text{recPicture}[\text{h}_{x-1}][v_y] - \text{curr}))$$

$$\text{sum} = \text{curr} + ((\text{sum} + (1 \ll (\text{alfShiftY} - 1))) \gg \text{alfShiftY})$$

- The modified filtered reconstructed luma picture sample $\text{alfPicture}_L[\text{xCtb} + \text{x}][\text{yCtb} + \text{y}]$ is derived as follows:

$$\text{alfPicture}_L[\text{xCtb} + \text{x}][\text{yCtb} + \text{y}] = \text{Clip3}(0, (1 \ll \text{BitDepth}) - 1, \text{sum})$$

15 Table 45 – Specification of y1, y2, y3 and alfShiftY according to the vertical luma sample position y and applyAlfLineBufBoundary

Condition	alfShiftY	y1	y2	y3
$(y == \text{CtbSizeY} - 5 \mid \mid y == \text{CtbSizeY} - 4) \ \&\& \ (\text{applyAlfLineBufBoundary} == 1)$	10	0	0	0
$(y == \text{CtbSizeY} - 6 \mid \mid y == \text{CtbSizeY} - 3) \ \&\& \ (\text{applyAlfLineBufBoundary} == 1)$	7	1	1	1
$(y == \text{CtbSizeY} - 7 \mid \mid y == \text{CtbSizeY} - 2) \ \&\& \ (\text{applyAlfLineBufBoundary} == 1)$	7	1	2	2
Otherwise	7	1	2	3

In another example,

Coding tree block filtering process for chroma samples

- 20 Inputs of this process are:
- a reconstructed chroma picture sample array recPicture prior to the adaptive loop filtering process,
 - a filtered reconstructed chroma picture sample array alfPicture ,
 - a chroma location $(\text{xCtbC}, \text{yCtbC})$ specifying the top-left sample of the current chroma coding tree block relative to the top-left sample of the current picture,
- 25

- an alternative chroma filter index altIdx.

Output of this process is the modified filtered reconstructed chroma picture sample array alfPicture.

The width and height of the current chroma coding tree block ctbWidthC and ctbHeightC is derived as follows:

$$ctbWidthC = CtbSizeY / SubWidthC$$

$$ctbHeightC = CtbSizeY / SubHeightC$$

For the derivation of the filtered reconstructed chroma samples alfPicture[xCtbC + x][yCtbC + y], each reconstructed chroma sample inside the current chroma coding tree block recPicture[xCtbC + x][yCtbC + y] is filtered as follows with $x = 0..ctbWidthC - 1, y = 0..ctbHeightC - 1$:

- The locations (h_{x+i}, v_{y+j}) for each of the corresponding chroma samples (x, y) inside the given array recPicture of chroma samples with $i, j = -2..2$ are derived as follows:

$$h_{x+i} = \text{Clip3}(0, \text{pps_pic_width_in_luma_samples} / \text{SubWidthC} - 1, xCtbC + x + i)$$

$$v_{y+j} = \text{Clip3}(0, \text{pps_pic_height_in_luma_samples} / \text{SubHeightC} - 1, yCtbC + y + j)$$

- The variables clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process with $(xCtbC * \text{SubWidthC}, yCtbC * \text{SubHeightC}), (x * \text{SubWidthC}, y * \text{SubHeightC})$ and the variable vbOffset set equal to $2 * \text{SubHeightC}$ as inputs.
- The variables h_{x+i} and v_{y+j} are modified by invoking the ALF sample padding process with $(xCtbC * \text{SubWidthC}, yCtbC * \text{SubHeightC}), (h_{x+i}, v_{y+j})$, the variable isChroma set equal to 1, clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag as inputs.
- The variable applyAlfLineBufBoundary is derived as follows:
 - If the bottom boundary of the current coding tree block is the bottom boundary of the picture and $\text{pps_pic_height_in_luma_samples} - (yCtbC * \text{SubHeightC}) < \text{CtbSizeY} - 4$, applyAlfLineBufBoundary is set equal to 0.
 - Otherwise, applyAlfLineBufBoundary is set equal to 1.

- The vertical sample position offsets y_1 , y_2 and the variable alfShiftC are specified in Table 46 according to the vertical chroma sample position y and $\text{applyAlfLineBufBoundary}$.

- The variable curr is derived as follows:

$$5 \quad \text{curr} = \text{recPicture}[h_x][v_y]$$

- The array of chroma filter coefficients $f[j]$ and the array of chroma clipping values $c[j]$ is derived as follows with $j = 0..5$:

$$f[j] = \text{AlfCoeffC}[sh_alf_aps_id_chroma][altIdx][j]$$

$$c[j] = \text{AlfClipc}[sh_alf_aps_id_chroma][altIdx][j]$$

- $$10 \quad \text{– The variable sum is derived as follows:}$$

$$\begin{aligned} \text{sum} = & f[0] * (\text{Clip3}(-c[0], c[0], \text{recPicture}[h_x][v_{y+y_2}] - \text{curr}) + \\ & \text{Clip3}(-c[0], c[0], \text{recPicture}[h_x][v_{y-y_2}] - \text{curr})) + \\ & f[1] * (\text{Clip3}(-c[1], c[1], \text{recPicture}[h_{x+1}][v_{y+y_1}] - \text{curr}) + \\ & \text{Clip3}(-c[1], c[1], \text{recPicture}[h_{x-1}][v_{y-y_1}] - \text{curr})) + \\ 15 \quad & f[2] * (\text{Clip3}(-c[2], c[2], \text{recPicture}[h_x][v_{y+y_1}] - \text{curr}) + \\ & \text{Clip3}(-c[2], c[2], \text{recPicture}[h_x][v_{y-y_1}] - \text{curr})) + \\ & f[3] * (\text{Clip3}(-c[3], c[3], \text{recPicture}[h_{x-1}][v_{y+y_1}] - \text{curr}) + \\ & \text{Clip3}(-c[3], c[3], \text{recPicture}[h_{x+1}][v_{y-y_1}] - \text{curr})) + \\ & f[4] * (\text{Clip3}(-c[4], c[4], \text{recPicture}[h_{x+2}][v_y] - \text{curr}) + \\ 20 \quad & \text{Clip3}(-c[4], c[4], \text{recPicture}[h_{x-2}][v_y] - \text{curr})) + \\ & f[5] * (\text{Clip3}(-c[5], c[5], \text{recPicture}[h_{x+1}][v_y] - \text{curr}) + \\ & \text{Clip3}(-c[5], c[5], \text{recPicture}[h_{x-1}][v_y] - \text{curr})) \end{aligned}$$

$$\text{sum} = \text{curr} + ((\text{sum} + (1 \ll (\text{alfShiftC} - 1))) \gg \text{alfShiftC})$$

- The modified filtered reconstructed chroma picture sample $\text{alfPicture}[x_{\text{CtbC}} + x][y_{\text{CtbC}} + y]$ is derived as follows:

$$\text{alfPicture}[x_{\text{CtbC}} + x][y_{\text{CtbC}} + y] = \text{Clip3}(0, (1 \ll \text{BitDepth}) - 1, \text{sum})$$

Table 46 – Specification of y_1 , y_2 and alfShiftC according to the vertical chroma sample position y and $\text{applyAlfLineBufBoundary}$

Condition	alfShiftC	y1	y2
$(y == \text{ctbHeightC} - 2 \mid \mid y == \text{ctbHeightC} - 3) \ \&\&$ $(\text{applyAlfLineBufBoundary} == 1)$	10	0	0
$(y == \text{ctbHeightC} - 1 \mid \mid y == \text{ctbHeightC} - 4) \ \&\&$ $(\text{applyAlfLineBufBoundary} == 1)$	7	1	1
Otherwise	7	1	2

In an implementation as shown in Fig. 26, a video decoding apparatus 2600 is disclosed, the apparatus 2600 comprise: a reconstructing module 2601, which is configured to obtain a reconstructed sample value for a block of a picture; a parsing module 2602, which is configured to obtain filter coefficients for adaptive loop filtering according to a bitstream; a calculating module 2603, which is configured to obtain a sum value according to the filter coefficients and the reconstructed sample value for the block; a rounding module 2604, which is configured to round the sum value according to a vertical position of a sample in the block, to obtain a rounded sum value; a filtering module 2605, which is configured to obtain a filter reconstructed sample value for the block according to the rounded sum value.

In an example, the rounded sum value is equal to $(\text{sum} + (1 \ll (\text{alfShiftY} - 1))) \gg \text{alfShiftY}$, wherein sum is the sum value, alfShiftY is a variable which is determined based on the vertical position of the sample.

In an example, alfShiftY is equal to 10 when the vertical position of the sample is satisfied any one of the below conditions: $y == \text{CtbSizeY} - t - 1$, or $y == \text{CtbSizeY} - t$, or $y == \text{ctbHeightC} - t$, or $y == \text{ctbHeightC} - t - 1$, wherein y is the vertical position of the sample, CtbSizeY is a coding tree block, CTB, size, ctbHeightC is the height of the CTB, t is an integer value.

In an example, alfShiftY is equal to 7 when the vertical position of the sample is not satisfied any one of the below conditions:

$y = \text{CtbSizeY} - t - 1$, or $y = \text{CtbSizeY} - t$, or $y = \text{ctbHeightC} - t$, or $y = \text{ctbHeightC} - t - 1$, wherein y is the vertical position of the sample, CtbSizeY is a coding tree block, CTB, size, ctbHeightC is the height of the CTB, t is an integer value.

In an example, when y is equal to 0, alfShiftY is equal to 10.

5 In an example, when y is not equal to 0, alfShiftY is equal to 7.

In an example, the vertical position of the sample in the block is a chroma sample position or is a luma sample position.

In an example, the rounding module is configured to

add value which is determined based on the vertical position to the sum value in order to
10 obtain an added sum value, and shift the added sum value based on the vertical position, to obtain the rounded sum value.

The further details for modules in video decoding apparatus could refer to the above method examples and implementations.

15 An additional solution (solution 3) which is depicted in Fig. 12c can also be used to perform filtering of the lines H, J, I (above virtual boundary) and lines K, L, M (below virtual boundary). Based on the dominant edge direction for a given block, the ALF filter is modified.

It has to be noted that any algorithm which can derive the dominant edge direction for the
20 given block can be used and is not limited to the algorithm explained below. The dominant edge direction for the given block can be computed using the gradient values computed for the given block using the gradient values g_v, g_h, g_{d1} and g_{d2} and directionality value D computed during the ALF block classification. If the directionality value D is 0, 1, 2 then as depicted in Fig. 12c, subfigure c, the solution 1 (adding coefficients vertically) is chosen. If
25 the directionality value D is 3 or 4 (case when the directionality of the block has a strong

“diagonal” edge), then the following further condition g_{d1} is greater than or equal to g_{d2} is checked. If true, then the 135 degree gradient is chosen as the dominant edge direction and therefore as depicted in Fig. 12c, subfigure b, the modified filtering is performed as follows:

For lines M and H $c3' = c3 + c0$

5 For lines L and I $c6' = c6 + c1$

$$c7' = c7 + c2$$

$$c8' = c3 + c8 + c0$$

For lines J and K:

$$c9' = c9 + c8 + c3 + c0$$

$$c10' = c10 + c7 + c2$$

$$c11' = c1 + c6 + c11 + c4$$

$$c12' = c12 + 2 * c5$$

10 On the other hand if g_{d2} is greater than g_{d1} then as depicted in Fig. 12c, subfigure a, the modified filtering is performed as follows:

For lines M and H $c1' = c1 + c0$

For lines L and I $c4' = c1 + c4 + c0$

$$c5' = c5 + c2$$

15 $c6' = c6 + c3$

For lines J and K:

$$c9' = c9 + c4 + c1 + c0$$

$$c10' = c10 + c5 + c2$$

$$c11' = c8 + c7 + c6 + c3 + c11$$

$$c12' = c12 + 2 * c7$$

If a given filter is transformed as given in table 1, then all the above solutions can still be
20 used. The equations in the above solutions are given for the case when no transformation is

applied. For a transformed filter, the coefficients in the equations can be replaced based on the equations 17, 18, 19 respectively.

An additional solution (solution 4) which is depicted in Figs. 19a and 19b, disables filtering when the absolute sum of the padded coefficients is greater than a threshold (T) times the absolute sum of the non padded coefficients. For example, for

lines M and H, $if (2 * abs(c0) > (T * (2 * (abs(c1) + abs(c2) + abs(c3) + abs(c4) + abs(c5) + abs(c6) + abs(c7) + abs(c8) + abs(c9) + abs(c10) + abs(c11)) + abs(c12))))$, then the ALF filtering is disabled for lines M and H.

Similarly for lines L and I, $if (2 * (abs(c0) + abs(c1) + abs(c2) + abs(c3)) > (T * (2 * (abs(c4) + abs(c5) + abs(c6) + abs(c7) + abs(c8) + abs(c9) + abs(c10) + abs(c11)) + abs(c12))))$, then the ALF filtering is disabled for lines L and I.

Similarly for lines J and K, $if (2 * (abs(c0) + abs(c1) + abs(c2) + abs(c3) + abs(c4) + abs(c5) + abs(c6) + abs(c7) + abs(c8)) > (T * (2 * (abs(c9) + abs(c10) + abs(c11)) + abs(c12))))$, then the ALF filtering is

disabled for lines J and K.

Where T is generally a threshold which can take a fractional values ranging between 0 and 1. For example, T value can be 0.5, 0.6 or 0.7

Similarly for Chroma, as depicted in Fig. 20, for lines N and K, $if (2 * abs(c0) > (T * (2 * (abs(c1) + abs(c2) + abs(c3) + abs(c4) + abs(c5) + abs(c6))))$, then the ALF

filtering is disabled for lines N and K.

for lines M and L, if $((2 * (abs(c0) + abs(c1) + abs(c2) + abs(c3)) > (T * (2 * (abs(c4) + abs(c5)) + abs(c6))))$, then the ALF filtering is disabled for lines M and L.

The function $abs(x)$ is defined as follows:

$$5 \quad Abs(x) = \begin{cases} x & ; \quad x \geq 0 \\ -x & ; \quad x < 0 \end{cases}$$

According to an alternative solution, the filtering operation is disabled if based on the determined geometric transformation and the horizontal proximity of the filtered pixel sample to the virtual boundary. In other words assume that the pixel sample to be filtered is N samples above or below the virtual boundary (horizontal distance is N to horizontal boundary)
10 and the determined geometric filter transformation is transformation_X. The filtering operation is disabled for a pixel sample if;

1. If N is smaller than a predetermined threshold, namely thr,
2. And if transformation_X is among a set of transformations that are predetermined or indicated in the bitstream.

15 According to one specific example thr is equal to 1 (indicating the row of samples that are one line above and one line below the virtual boundary). According to another example the set of transformations include at least one of 'No transformation', 'Diagonal', 'Vertical flip', 'Rotation'. According to another example thr is equal to 1 and the set of transformations include at least one of 'No transformation', 'Diagonal', 'Vertical flip', 'Rotation'. In another
20 example the set of transformations include Diagonal and Rotation.

It is noted that the set of transformations is a subset of all possible filter transformations. According to document JVET1001_v7 "Versatile Video Coding (Draft 3)" document (publicly accessible at <http://phenix.it-sudparis.eu/jvet/>), there are 4 possible filter transformations. In the document the filter transformation is determined according to

subsection “8.5.4.3 Derivation process for ALF transpose and filter index for luma samples”, and the determined transformation is indicated by variable `transposeIdx`. The values that can be assumed by `transposeIdx` are 0, 1, 2 and 3.

According to one example the filtering operation is disabled for a pixel sample if the `transposeIdx` is determined to be equal to a predetermined value. It is noted that the set of predetermined values includes at least one value between 0 and 3. It is further noted that the set of predetermined values does not include all of the possible values (all of 0, 1, 2 and 3).

In another example the filtering operation is disabled if the pixel to be filtered is one sample above or below the virtual boundary and the determined filter transformation is among a set of predetermined values.

Line buffer requirements for chroma samples:

Fig. 13 depicts the line buffer requirements for the chroma samples. As deblocking only uses 2 samples and modifies a maximum of 1 sample, the chroma ALF line buffer requirements comes down to 6.25 lines. Since no explicit block classification is performed for the chroma samples, only the filtering has to be modified at the virtual boundaries (VBs). Fig. 14 refer to modified ALF filter for Chroma, which shows the distorted version of the filter generated by either using padding (solution 1) or by adding the discarded coefficient to the center coefficient (solution 2).

Modified block classification when virtual boundary is 6 lines

Fig. 15 shows the ALF line buffer requirements when the deblocking filter uses 6 lines of line buffer. Therefore the virtual boundary is 6 lines above the CTU boundary. For this case, the total loop filter line buffer requirement increases to 15.25 lines.

To reduce the line buffer requirements, Fig. 16 discloses re-use block classification decision at virtual boundary and use truncated version for filtering, and Fig. 16 depicts one solution which is explained below:

The block classification of the 4 x 4 block starting at line C is achieved by padding the line H to line I to ensure the block classification does not require any line below the virtual boundary. In Fig. 16, Line H is padded to line I to facilitate the block classification decision of 4 x 4 block starting at line C

Lines G, H re-use block classification decision from block C, D, E, F

lines I, J use block classification decision from the block K, L, M, N

Truncated versions of the filters (slides 8 and 9) are used to filter lines F, G, H, I, J, K.

Lines G, H re-use block classification decision from block C, D, E, F. Lines I, J re-use block classification decision from lines K, L, M, N (or) lines I, J, K, L can be considered as one 4x 4 block segment and classification is performed for these 4 lines, and then lines M, N re-use the block classification decision from block K, L, M, N.

Fig. 17 uses a grid-shifting approach where the first two lines at the top picture boundary and last two lines at the bottom picture boundary are classified and filtered separately and therefore the ALF classification and filtering grid is moved by 2 samples down in the Y axis. This solution has an advantage that a given 4 x 4 block never overlaps or crosses across a virtual boundary. In general when the virtual boundary size is not a multiple of 4 (ALF block classification size), then the ALF filtering grid can be shifted by using the following formula:

$((\text{Virtual boundary size}) / 4) - 4$. For the actual filtering itself, the distorted version of the filter explained in Figures 12a, 12b and 12c can be used. Refer to Fig. 17, when virtual boundary is 6 lines above the CTU boundary, the solution uses a “Grid

Shifting” approach.

The ALF filter grid is shifted by two lines in the Y-axis to ensure that a given 4 x 4 block classification window does not cross a given virtual boundary

In general when the virtual boundary size is not a multiple of 4 (ALF block classification size), the grid can be shifted by the following value on the Y-Axis :

$$(((\text{Virtual boundary size}) / 4) - 4).$$

At the frame boundaries,

At the top frame boundary lines **A'**, **B'** re-use the block classification decision from the 4 x 4 block starting at the third line.

Similarly for the last 2 lines at the bottom frame boundary the block classification re-uses the decision from the immediately above 4 x4 block classification window.

Although embodiments of the invention have been primarily described based on video coding, it should be noted that embodiments of the coding system 10, encoder 20 and decoder 30 (and correspondingly the system 10) and the other embodiments described herein may also be configured for still picture processing or coding, i.e. the processing or coding of an individual picture independent of any preceding or consecutive picture as in video coding. In general only inter-prediction units 244 (encoder) and 344 (decoder) may not be available in case the picture processing coding is limited to a single picture 17. All other functionalities (also referred to as tools or technologies) of the video encoder 20 and video decoder 30 may equally be used for still picture processing, e.g., residual calculation 204/304, transform 206, quantization 208, inverse quantization 210/310, (inverse) transform 212/312, partitioning 262/362, intra-prediction 254/354, and/or loop filtering 220, 320, and entropy coding 270 and entropy decoding 304.

Following is an explanation of the applications of the encoding method as well as the decoding method as shown in the above-mentioned embodiments, and a system using them.

Fig. 23 is a block diagram showing a content supply system 3100 for realizing content distribution service. This content supply system 3100 includes capture device 3102, terminal device 3106, and optionally includes display 3126. The capture device 3102 communicates with the terminal device 3106 over communication link 3104. The communication link may include the communication channel 13 described above. The communication link 3104 includes but not limited to WIFI, Ethernet, Cable, wireless (3G/4G/5G), USB, or any kind of combination thereof, or the like.

The capture device 3102 generates data, and may encode the data by the encoding method as shown in the above embodiments. Alternatively, the capture device 3102 may distribute the data to a streaming server (not shown in the Figures), and the server encodes the data and transmits the encoded data to the terminal device 3106. The capture device 3102 includes but not limited to camera, smart phone or Pad, computer or laptop, video conference system, PDA, vehicle mounted device, or a combination of any of them, or the like. For example, the capture device 3102 may include the source device 12 as described above. When the data includes video, the video encoder 20 included in the capture device 3102 may actually perform video encoding processing. When the data includes audio (i.e., voice), an audio encoder included in the capture device 3102 may actually perform audio encoding processing. For some practical scenarios, the capture device 3102 distributes the encoded video and audio data by multiplexing them together. For other practical scenarios, for example in the video conference system, the encoded audio data and the encoded video data are not multiplexed. Capture device 3102 distributes the encoded audio data and the encoded video data to the terminal device 3106 separately.

In the content supply system 3100, the terminal device 310 receives and reproduces the encoded data. The terminal device 3106 could be a device with data receiving and recovering capability, such as smart phone or Pad 3108, computer or laptop 3110, network video recorder (NVR)/ digital video recorder (DVR) 3112, TV 3114, set top box (STB) 3116, video

conference system 3118, video surveillance system 3120, personal digital assistant (PDA) 3122, vehicle mounted device 3124, or a combination of any of them, or the like capable of decoding the above-mentioned encoded data. For example, the terminal device 3106 may include the destination device 14 as described above. When the encoded data includes video, 5 the video decoder 30 included in the terminal device is prioritized to perform video decoding. When the encoded data includes audio, an audio decoder included in the terminal device is prioritized to perform audio decoding processing.

For a terminal device with its display, for example, smart phone or Pad 3108, computer or laptop 3110, network video recorder (NVR)/ digital video recorder (DVR) 3112, TV 3114, 10 personal digital assistant (PDA) 3122, or vehicle mounted device 3124, the terminal device can feed the decoded data to its display. For a terminal device equipped with no display, for example, STB 3116, video conference system 3118, or video surveillance system 3120, an external display 3126 is contacted therein to receive and show the decoded data.

When each device in this system performs encoding or decoding, the picture encoding device 15 or the picture decoding device, as shown in the above-mentioned embodiments, can be used.

Fig. 24 is a diagram showing a structure of an example of the terminal device 3106. After the terminal device 3106 receives stream from the capture device 3102, the protocol proceeding unit 3202 analyzes the transmission protocol of the stream. The protocol includes but not limited to Real Time Streaming Protocol (RTSP), Hyper Text Transfer Protocol (HTTP), 20 HTTP Live streaming protocol (HLS), MPEG-DASH, Real-time Transport protocol (RTP), Real Time Messaging Protocol (RTMP), or any kind of combination thereof, or the like.

After the protocol proceeding unit 3202 processes the stream, stream file is generated. The file is outputted to a demultiplexing unit 3204. The demultiplexing unit 3204 can separate the multiplexed data into the encoded audio data and the encoded video data. As described above, 25 for some practical scenarios, for example in the video conference system, the encoded audio data and the encoded video data are not multiplexed. In this situation, the encoded data is

transmitted to video decoder 3206 and audio decoder 3208 without through the demultiplexing unit 3204.

Via the demultiplexing processing, video elementary stream (ES), audio ES, and optionally subtitle are generated. The video decoder 3206, which includes the video decoder 30 as explained in the above mentioned embodiments, decodes the video ES by the decoding method as shown in the above-mentioned embodiments to generate video frame, and feeds this data to the synchronous unit 3212. The audio decoder 3208, decodes the audio ES to generate audio frame, and feeds this data to the synchronous unit 3212. Alternatively, the video frame may store in a buffer (not shown in Fig. 24) before feeding it to the synchronous unit 3212. Similarly, the audio frame may store in a buffer (not shown in Fig. 24) before feeding it to the synchronous unit 3212.

The synchronous unit 3212 synchronizes the video frame and the audio frame, and supplies the video/audio to a video/audio display 3214. For example, the synchronous unit 3212 synchronizes the presentation of the video and audio information. Information may code in the syntax using time stamps concerning the presentation of coded audio and visual data and time stamps concerning the delivery of the data stream itself.

If subtitle is included in the stream, the subtitle decoder 3210 decodes the subtitle, and synchronizes it with the video frame and the audio frame, and supplies the video/audio/subtitle to a video/audio/subtitle display 3216.

The present invention is not limited to the above-mentioned system, and either the picture encoding device or the picture decoding device in the above-mentioned embodiments can be incorporated into other system, for example, a car system.

Embodiments, e.g., of the encoder 20 and the decoder 30, and functions described herein, e.g., with reference to the encoder 20 and the decoder 30, may be implemented in hardware, software, firmware, or any combination thereof. If implemented in software, the functions may be stored on a computer-readable medium or transmitted over communication media as

one or more instructions or code and executed by a hardware-based processing unit.

Computer-readable media may include computer-readable storage media, which corresponds to a tangible medium such as data storage media, or communication media including any medium that facilitates transfer of a computer program from one place to another, e.g.,

5 according to a communication protocol. In this manner, computer-readable media generally may correspond to (1) tangible computer-readable storage media which is non-transitory or (2) a communication medium such as a signal or carrier wave. Data storage media may be any available media that can be accessed by one or more computers or one or more processors to retrieve instructions, code and/or data structures for implementation of the techniques
10 described in this disclosure. A computer program product may include a computer-readable medium.

By way of example, and not limiting, such computer-readable storage media can comprise RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage, or other magnetic storage devices, flash memory, or any other medium that can be used to store
15 desired program code in the form of instructions or data structures and that can be accessed by a computer. Also, any connection is properly termed a computer-readable medium. For example, if instructions are transmitted from a website, server, or other remote source using a coaxial cable, fiber optic cable, twisted pair, digital subscriber line (DSL), or wireless technologies such as infrared, radio, and microwave, then the coaxial cable, fiber optic cable,
20 twisted pair, DSL, or wireless technologies such as infrared, radio, and microwave are included in the definition of medium. It should be understood, however, that computer-readable storage media and data storage media do not include connections, carrier waves, signals, or other transitory media, but are instead directed to non-transitory, tangible storage media. Disk and disc, as used herein, includes compact disc (CD), laser disc, optical
25 disc, digital versatile disc (DVD), floppy disk and Blu-ray disc, where disks usually

reproduce data magnetically, while discs reproduce data optically with lasers. Combinations of the above should also be included within the scope of computer-readable media.

Instructions may be executed by one or more processors, such as one or more digital signal processors (DSPs), general purpose microprocessors, application specific integrated circuits (ASICs), field programmable logic arrays (FPGAs), or other equivalent integrated or discrete logic circuitry. Accordingly, the term “processor,” as used herein may refer to any of the foregoing structure or any other structure suitable for implementation of the techniques described herein. In addition, in some aspects, the functionality described herein may be provided within dedicated hardware and/or software modules configured for encoding and decoding, or incorporated in a combined codec. Also, the techniques could be fully implemented in one or more circuits or logic elements.

The techniques of this disclosure may be implemented in a wide variety of devices or apparatuses, including a wireless handset, an integrated circuit (IC) or a set of ICs (e.g., a chip set). Various components, modules, or units are described in this disclosure to emphasize functional aspects of devices configured to perform the disclosed techniques, but do not necessarily require realization by different hardware units. Rather, as described above, various units may be combined in a codec hardware unit or provided by a collection of interoperative hardware units, including one or more processors as described above, in conjunction with suitable software and/or firmware.

The present disclosure discloses the following further aspects:

Example 1. A method of filtering a sample value of a picture, the picture comprising a first block and a second block, the first block and the second block adjoining each other in a block boundary, wherein the picture comprises at least $n \times m$ samples $p(i,j)$ arranged in n rows and m columns, wherein i is a row index that runs from 1 to n and j is a column index that runs from 1 to m , wherein samples $p(i,j)$ with i from 1 to k belong to the first block and samples with i from $k+1$ to n belong to the second block, wherein k is an integer number less than n ,

wherein the method comprises, with an integer t that is less than k and with an integer j in the range of 1 to m :

generating, with an integer i in the range of 1 to $k-t-1$, a filtered sample value by applying a filter to the value of the sample $p(i,j)$, wherein the filter is supported by a set of filter support samples, wherein the set of filter support samples comprises the sample $p(i,j)$ and one or more further samples, wherein the set of filter support samples does not comprise any samples $p(i',j')$ with row index i' greater than $k-t-1$, and/or

generating, with an integer i in the range of $k-t$ to k , a filtered sample value by applying a filter to the value of the sample $p(i,j)$, wherein the filter is supported by a set of filter support samples, wherein the set of filter support samples comprises the sample $p(i,j)$ and one or more further samples, wherein the set of filter support samples does not comprise any samples $p(i',j')$ with row index i' less than $k-t$.

Example 2. A method of filtering sample values of a picture, the picture comprising a first block and a second block, the first block and the second block adjoining each other in a block boundary,

wherein the picture comprises at least $n \times m$ samples $p(i,j)$ arranged in n rows and m columns, wherein i is a row index that runs from 1 to n and j is a column index that runs from 1 to m , wherein samples $p(i,j)$ with i from 1 to k belong to the first block and samples with i from $k+1$ to n belong to the second block, wherein k is an integer number less than n ,

wherein the method comprises, with an integer t that is less than k and for each i from 1 to k , and each j from 1 to m :

generating a filtered sample value by applying a filter to the value of sample $p(i,j)$, wherein the filter is supported by a set of filter support samples, wherein the set of filter support samples comprises the sample $p(i,j)$ and one or more further samples, wherein if i is in the range of 1 to $k-t-1$, the set of filter support samples does not comprise any samples $p(i',j')$ with row index i' greater than $k-t-1$, and wherein if i is in the range of $k-t$ to k , the set

of filter support samples does not comprise any samples $p(i',j')$ with row index i' less than $k-t$.

Example 3. The method of example 1 or 2, wherein the first block is a first coding tree unit, CTU, and the second block is a second CTU.

5 Example 4. The method of any one of examples 1 to 3, wherein t is 2, 3, 4, 5 or 6.

Example 5. The method of any one of examples 1 to 4,

wherein for each i in the range of 1 to $k-t-1$ and for each j from 1 to m , the set of filter support samples comprises multiple samples $p(i',j')$ with i' in the range of 1 to $k-t-1$; and

10 wherein for each i in the range of $k-t$ to k and for each j from 1 to m , the set of filter support samples comprises multiple samples $p(i',j')$ with i' equal to or greater than $k-t$.

Example 6. The method of example 5, wherein for each i in the range of $k-t$ to k and for each j from 1 to m , the set of filter support samples further comprises one or more samples $p(i',j')$ with i' greater than k .

Example 7. The method of any one of examples 1 to 6, wherein the filter is an in-loop filter.

15 Example 8. The method of any one of examples 1 to 7, wherein the filter comprises a deblocking filter.

Example 9. The method of any one of examples 1 to 8, wherein the filter comprises a deblocking filter, a sample adaptive offset filter, SAO, an adaptive loop filter, ALF, or cross component adaptive loop filter (CCALF).

20 Example 10. The method of any one of examples 1 to 9, comprising:

determining sample values of the first block;

applying the filter to the sample values of the first block in rows 1 to $k-t-1$;

storing the sample values of the first block from rows $k-t$ to k in a buffer;

determining sample values of the second block; and

25 applying the filter to the sample values of the first block in rows $k-t$ that have been stored in the buffer.

Example 11. The method of any one of examples 1 to 10, wherein the filter is an adaptive loop filter or a cross component adaptive loop filter, or the adaptive filter is determined by using a classification step which accesses samples of the current block and also samples surrounding the block.

- 5 Example 12. The method of example 11, wherein when samples surrounding the block are not available, the adaptive filter is determined by using a normalization value, wherein the normalization value is a positive integer value and is dependent on the number of accessible samples surrounding a given block.

- Example 13. The method of example 11 or 12, wherein the normalization value has an
10 inverse relationship with the number of accessible samples.

Example 14. The method of any of the examples 10 to 13, where in the normalization value is 96, when not all the samples surrounding the block are available.

Example 15. The method of any of the examples 10 to 14, wherein the normalization value is 64, when all the samples surrounding the block are available.

- 15 Example 16. The method of any one of examples 1 to 10, wherein the filter is an adaptive loop filter.

Example 17. The method of any one of the examples 1 to 10, wherein a symmetric padding operation is applied when required samples are not accessible for filtering a given sample.

- Example 18. The method of the example 17, wherein the symmetric padding operation is
20 defined as replacing the unavailable samples by immediate neighboring samples which are available for filtering operation.

- Example 19. The method of any of example 17 or 18, wherein the symmetric padding operation is defined as a method that the filter is made symmetric by applying the padding operation on both sides of the filter even though on one side, the filter samples can still be
25 accessible.

Example 20. The method of any one of examples 1 to 19, wherein the picture is included in a frame of a video sequence.

Example 21. The method of any one of examples 1 to 20, performed to encode or decode the picture, or performed to encode or decode a video sequence that comprises the picture.

- 5 Example 22. The method of any one of examples 1 to 21, wherein a rounding value used in the filtering process is dependent on the vertical position (y coordinate) of the sample $p(i,j)$.

Example 23. The method of any one of examples 1 to 22, wherein a rounding value used in the filter process is dependent on the vertical position (y coordinate) of the sample $p(i,j)$ and a quantization value (e.g. $\gg$ right shift value) used in the filtering process.

- 10 Example 24. The method of example 23, wherein when the quantization value is indicated by a variable Q , the rounding value is equal to $pow(2, Q-1)$, the variable is obtained according to the vertical position (y coordinate) of the sample $p(i,j)$.

Example 25. The method of any of examples 22 to 24, wherein the rounding value is equal to 512 when the vertical position (y coordinate) of the sample $p(i,j)$ is satisfied any one of the

- 15 below conditions:

$y == CtbSizeY - t - 1$, or $y == CtbSizeY - t$, or $y == ctbHeightC - t$, or $y == ctbHeightC - t - 1$ (In some examples, for Chroma, t is equal to 2; for Luma, t is equal to 4).

Example 26. The method of any of examples 22 to 25, wherein the rounding value is equal to 64 when the vertical position (y coordinate) of the sample $p(i,j)$ is not satisfied any one of the

- 20 below conditions:

$y == CtbSizeY - t - 1$, or $y == CtbSizeY - t$, or $y == ctbHeightC - t$, or $y == ctbHeightC - t - 1$.

Example 27. An encoder (20) comprising processing circuitry for carrying out the method according to any one of examples 1 to 26.

- 25 Example 28. A decoder (30) comprising processing circuitry for carrying out the method according to any one of examples 1 to 26.

Example 29. A computer program comprising a program code for performing the method according to any one of examples 1 to 26.

Example 30. A decoder, comprising:

- one or more processors; and
- 5 a non-transitory computer-readable storage medium coupled to the processors and storing programming for execution by the processors, wherein the programming, when executed by the processors, configures the decoder to carry out the method according to any one of examples 1 to 26.

Example 31. An encoder, comprising:

- 10 one or more processors; and
- a non-transitory computer-readable storage medium coupled to the processors and storing programming for execution by the processors, wherein the programming, when executed by the processors, configures the encoder to carry out the method according to any one of examples 1 to 26.

15

CLAIMS

1. A method of filtering a sample value of a picture,

wherein the method comprises:

5 obtaining a reconstructed sample value for a block of the picture;

obtaining filter coefficients for adaptive loop filtering according to a bitstream;

obtaining a sum value according to the filter coefficients and the reconstructed sample value for the block;

rounding the sum value according to a vertical position of a sample in the block, to obtain

10 a rounded sum value, wherein the vertical position of the sample in the block is a luma sample position, and wherein the rounded sum value equals $\text{curr} + ((\text{sum} + \text{roundY}) \gg \text{alfShiftY})$

and satisfies the following table:

Condition	roundY	alfShiftY	y1	y2	y3
$(y == \text{CtbSizeY} - 5 \quad \quad y == \text{CtbSizeY} - 4)$ && $(\text{applyAlfLineBufBoundary} == 1)$	512	10	0	0	0
$(y == \text{CtbSizeY} - 6 \quad \quad y == \text{CtbSizeY} - 3)$ && $(\text{applyAlfLineBufBoundary} == 1)$	64	7	1	1	1
$(y == \text{CtbSizeY} - 7 \quad \quad y == \text{CtbSizeY} - 2)$ && $(\text{applyAlfLineBufBoundary} == 1)$	64	7	1	2	2
Otherwise	64	7	1	2	3

wherein

15 y represents the vertical position of the sample;

curr represents the reconstructed sample value for the block;

sum represents the sum value;

CtbSizeY represents a size of a coding tree block (CTB) corresponding to the block;

y1, y2 and y3 represent vertical sample position offsets indicating other reconstructed sample values involved in the obtaining the sum value;

roundY and alfShiftY are variables that are determined based on the vertical position of the sample;

apply AlfLineBufBoundary is an index and derived as follows:

if a bottom boundary of the coding tree block is a bottom boundary of the picture and $\text{pic_height_in_luma_samples} - y_{\text{Ctb}} \leq \text{CtbSizeY} - 4$,

apply AlfLineBufBoundary is set equal to 0, otherwise,

apply AlfLineBufBoundary is set equal to 1,

wherein the $\text{pic_height_in_luma_samples}$ represents a height of the picture in luma samples, and $(x_{\text{Ctb}}, y_{\text{Ctb}})$ represents a location of the CTB; and

obtaining a filter reconstructed sample value for the block according to the rounded sum value.

2. The method of claim 1, wherein alfShiftY is equal to 10 when the vertical position of the sample is satisfied any one of the below conditions:

$y = \text{CtbSizeY} - t - 1$, or

$y = \text{CtbSizeY} - t$, or

$y = \text{ctbHeightC} - t$, or

$y = \text{ctbHeightC} - t - 1$,

wherein y is the vertical position of the sample, CtbSizeY is a coding tree block, CTB, size, ctbHeightC is the height of the CTB, t is an integer value.

3. The method of claim 1 or 2, wherein alfShiftY is equal to 7 when the vertical position of the sample is not satisfied any one of the below conditions:

$y = \text{CtbSizeY} - t - 1$, or

$y = \text{CtbSizeY} - t$, or

$y = \text{ctbHeightC} - t$, or

$y = \text{ctbHeightC} - t - 1$,

5 wherein y is the vertical position of the sample, CtbSizeY is a coding tree block, CTB, size, ctbHeightC is the height of the CTB, t is an integer value.

4. The method of any one of claims 1 to 3, wherein when y is equal to 0, alfShiftY is equal to 10.

10

5. The method of any one of claims 1 to 4, wherein when y is not equal to 0, alfShiftY is equal to 7.

6. The method of any one of claims 1 to 5, wherein the rounding process comprises:

15 adding a value which is determined based on the vertical position to the sum value in order to obtain an added sum value, and
 shifting the added sum value based on the vertical position.

7. A video decoding apparatus, comprising:

20 a reconstructing module, which is configured to obtain a reconstructed sample value for a block of a picture;

 a parsing module, which is configured to obtain filter coefficients for adaptive loop filtering according to a bitstream;

 a calculating module, which is configured to obtain a sum value according to the filter
25 coefficients and the reconstructed sample value for the block;

a rounding module, which is configured to round the sum value according to a vertical position of a sample in the block, to obtain a rounded sum value, wherein the vertical position of the sample in the block is a luma sample position, and wherein the rounded sum value equals $\text{curr} + ((\text{sum} + \text{roundY}) \gg \text{alfShiftY})$ and satisfies the following table:

Condition	roundY	alfShiftY	y1	y2	y3
$(y == \text{CtbSizeY} - 5 \quad \quad y == \text{CtbSizeY} - 4)$ && $(\text{applyAlfLineBufBoundary} == 1)$	512	10	0	0	0
$(y == \text{CtbSizeY} - 6 \quad \quad y == \text{CtbSizeY} - 3)$ && $(\text{applyAlfLineBufBoundary} == 1)$	64	7	1	1	1
$(y == \text{CtbSizeY} - 7 \quad \quad y == \text{CtbSizeY} - 2)$ && $(\text{applyAlfLineBufBoundary} == 1)$	64	7	1	2	2
Otherwise	64	7	1	2	3

5

wherein

y represents the vertical position of the sample;

curr represents the reconstructed sample value for the block;

sum represents the sum value;

10 CtbSizeY represents a size of a coding tree block (CTB) corresponding to the block;

y1, y2 and y3 represent vertical sample position offsets indicating other reconstructed sample values involved in the obtaining the sum value;

roundY and alfShiftY are variables that are determined based on the vertical position of the sample;

15 apply AlfLineBufBoundary is an index and derived as follows:

if a bottom boundary of the coding tree block is a bottom boundary of the picture and $\text{pic_height_in_luma_samples} - y_{\text{Ctb}} \leq \text{CtbSizeY} - 4$,

apply $\text{AlfLineBufBoundary}$ is set equal to 0, otherwise,

apply $\text{AlfLineBufBoundary}$ is set equal to 1,

wherein the $\text{pic_height_in_luma_samples}$ represents a height of the picture in luma samples, and $(x_{\text{Ctb}}, y_{\text{Ctb}})$ represents a location of the CTB; and

5 a filtering module, which is configured to obtain a filter reconstructed sample value for the block according to the rounded sum value.

8. The video decoding apparatus of claim 7, wherein alfShiftY is equal to 10 when the vertical position of the sample is satisfied any one of the below conditions:

10 $y = \text{CtbSizeY} - t - 1$, or

$y = \text{CtbSizeY} - t$, or

$y = \text{ctbHeightC} - t$, or

$y = \text{ctbHeightC} - t - 1$,

wherein y is the vertical position of the sample, CtbSizeY is a coding tree block, CTB,

15 size, ctbHeightC is the height of the CTB, t is an integer value.

9. The video decoding apparatus of claim 7 or 8, wherein alfShiftY is equal to 7 when the vertical position of the sample is not satisfied any one of the below conditions:

$y = \text{CtbSizeY} - t - 1$, or

20 $y = \text{CtbSizeY} - t$, or

$y = \text{ctbHeightC} - t$, or

$y = \text{ctbHeightC} - t - 1$,

wherein y is the vertical position of the sample, CtbSizeY is a coding tree block, CTB, size, ctbHeightC is the height of the CTB, t is an integer value.

25

10. The video decoding apparatus of any one of claims 7 to 9, wherein when y is equal to 0, alfShiftY is equal to 10.

11. The video decoding apparatus of any one of claims 7 to 10, wherein when y is not equal to 0, alfShiftY is equal to 7.

12. The video decoding apparatus of any one of claims 7 to 11, wherein the rounding module is configured to

add a value which is determined based on the vertical position to the sum value in order to obtain an added sum value, and shift the added sum value based on the vertical position, to obtain the rounded sum value.

13. A decoder, comprising:

one or more processors; and

a non-transitory computer-readable storage medium coupled to the processors and storing programming for execution by the processors, wherein the programming, when executed by the processors, configures the decoder to carry out the method according to any one of the claims 1 to 6.

14. A non-transitory computer-readable medium comprising an encoded bitstream for a video signal decoded by performing the method of any one of the claims 1 to 6.