US Patent & Trademark Office
Patent Public Search | Text View

United States Patent Application Publication	20250260812
Kind Code	A1
Publication Date	August 14, 2025
Inventor(s)	Yin; Wenbin et al.

ADAPTIVE FILTER REUSING METHODS ON ADAPTIVE LOOP FILTER IN VIDEO CODING

Abstract

A method, apparatus, and system for processing video data is disclosed. In one aspect, the method includes reusing filters of an adaptive loop filter (ALF) for a current ALF processing unit after modification of the filters. A conversion is performed between a visual media data and a bitstream based on the ALF.

Inventors: **Yin; Wenbin (Beijing, CN), Zhang; Kai (San Diego, CA), Zhang; Li (San Diego, CA)**

Applicant: **Douyin Vision Co., Ltd.** (Beijing, CN); **Bytedance Inc.** (Los Angeles, CA)

Family ID: **1000008563224**

Appl. No.: **19/192638**

Filed: **April 29, 2025**

Foreign Application Priority Data

WO	PCT/CN2022/128886	Nov. 01, 2022

Related U.S. Application Data

parent WO continuation PCT/CN2023/129046 20231101 PENDING child US 19192638

Publication Classification

Int. Cl.: **H04N19/117** (20140101); **H04N19/159** (20140101); **H04N19/184** (20140101); **H04N19/70** (20140101)

U.S. Cl.:

CPC **H04N19/117** (20141101); **H04N19/159** (20141101); **H04N19/184** (20141101); **H04N19/70** (20141101);

Background/Summary

CROSS REFERENCE TO RELATED APPLICATIONS [0001] This application is a continuation of International Patent Application No. PCT/CN2023/129046, filed on Nov. 1, 2023, which claims the priority to and benefits of International Patent Application No. PCT/CN2022/128886, filed on Nov. 1, 2022. All the aforementioned patent applications are hereby incorporated by reference in their entireties.

TECHNICAL FIELD
[0002] The present disclosure relates to generation, storage, and consumption of digital audio video media information in a file format.
BACKGROUND
[0003] Digital video accounts for the largest bandwidth used on the Internet and other digital communication networks. As the number of connected user devices capable of receiving and displaying video increases, the bandwidth demand for digital video usage is likely to continue to grow.
SUMMARY
[0004] A first aspect relates to a method for processing video data, comprising: reusing filters of an adaptive loop filter (ALF) for a current ALF processing unit after modification of the filters; and performing a conversion between a visual media data and a bitstream based on the ALF.
[0005] Optionally, in any of the preceding aspects, another implementation of the aspect provides scaling one or more coefficients of a first filter of the filters by a weight (w).
[0006] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the weight is 0.5 or 1.5.
[0007] Optionally, in any of the preceding aspects, another implementation of the aspect provides scaling all coefficients of a first filter of the filters by a weight (w), wherein the weight is identical for all of the coefficients.
[0008] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the filters are included in the bitstream for each ALF processing unit.
[0009] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the weight is included in the bitstream for each ALF processing unit.
[0010] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the weight is pre-defined at an encoder and at a decoder for each ALF processing unit.
[0011] Optionally, in any of the preceding aspects, another implementation of the aspect provides that a weight index of the weight is included in the bitstream for each ALF processing unit.
[0012] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the weight is determined in real time for each ALF processing unit.
[0013] Optionally, in any of the preceding aspects, another implementation of the aspect provides scaling two different coefficients of a first filter from the filters by two different weights.
[0014] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different weights are included in the bitstream for each ALF processing unit.
[0015] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different weights are pre-defined at an encoder and at a decoder for each ALF processing unit.
[0016] Optionally, in any of the preceding aspects, another implementation of the aspect provides that weight indexes are included in the bitstream for each ALF processing unit.

[0017] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different weights are determined in real time for each ALF processing unit.
[0018] Optionally, in any of the preceding aspects, another implementation of the aspect provides adding an offset (k) to one or more coefficients of a first filter of the filters.
[0019] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is 10 or −10.
[0020] Optionally, in any of the preceding aspects, another implementation of the aspect provides adding an offset (k) to all coefficients of a first filter of the filters, wherein the offset is identical for all of the coefficients.
[0021] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is included in the bitstream for each ALF processing unit.
[0022] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is included in the bitstream for each ALF processing unit.
[0023] Optionally, in any of the preceding aspects, another implementation of the aspect provides that an offset index for the offset is included in the bitstream for each ALF processing unit.
[0024] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is determined in real time for each ALF processing unit.
[0025] Optionally, in any of the preceding aspects, another implementation of the aspect provides adding two different offsets to two different coefficients of a first filter from the filters.
[0026] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different offsets are included in the bitstream for each ALF processing unit.
[0027] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different offsets are pre-defined at an encoder and at a decoder for each ALF processing unit.
[0028] Optionally, in any of the preceding aspects, another implementation of the aspect provides that offset indexes for the offsets are included in the bitstream for each ALF processing unit.
[0029] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different offsets are determined in real time for each ALF processing unit.
[0030] Optionally, in any of the preceding aspects, another implementation of the aspect provides adding an offset (t) to one or more clipping parameters of a first filter of the filters.
[0031] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is +1 or −1.
[0032] Optionally, in any of the preceding aspects, another implementation of the aspect provides adding an offset (t) to all clipping parameters of a first filter of the filters, wherein the offset is identical for all of the clipping parameters.
[0033] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is included in the bitstream for each ALF processing unit.
[0034] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is pre-defined at an encoder and at a decoder for each ALF processing unit.
[0035] Optionally, in any of the preceding aspects, another implementation of the aspect provides that an offset index for the offset is included in the bitstream for each ALF processing unit.
[0036] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is determined in real time for each ALF processing unit.
[0037] Optionally, in any of the preceding aspects, another implementation of the aspect provides adding two different offsets to two different clipping parameters of a first filter from the filters.
[0038] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different offsets are included in the bitstream for each ALF processing unit.
[0039] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different offsets are pre-defined at an encoder and at a decoder for each ALF processing unit.

[0040] Optionally, in any of the preceding aspects, another implementation of the aspect provides that offset indexes for the offsets are included in the bitstream for each ALF processing unit.
[0041] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different offsets are determined in real time for each ALF processing unit.
[0042] Optionally, in any of the preceding aspects, another implementation of the aspect provides scaling an output of a first filter of the filters by a weight.
[0043] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the weight is included in the bitstream for each ALF processing unit.
[0044] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the weight is pre-defined at an encoder and at a decoder for each ALF processing unit.
[0045] Optionally, in any of the preceding aspects, another implementation of the aspect provides that two different weight indexes are included in the bitstream for two different ALF processing units.
[0046] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the weight is determined in real time for each ALF processing unit.
[0047] Optionally, in any of the preceding aspects, another implementation of the aspect provides adding an offset to an output of a first filter of the filters.
[0048] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is included in the bitstream for each ALF processing unit.
[0049] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is pre-defined at an encoder and at a decoder for each ALF processing unit.
[0050] Optionally, in any of the preceding aspects, another implementation of the aspect provides that an offset index for the offset is included in the bitstream for each ALF processing unit.
[0051] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is determined in real time for each ALF processing unit.
[0052] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the current ALF processing unit is a luma component.
[0053] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the current ALF processing unit is a chroma component.
[0054] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the ALF comprises a cross component ALF (CCALF) and the current ALF processing unit comprises a CCALF processing unit.
[0055] Optionally, in any of the preceding aspects, another implementation of the aspect provides scaling one or more coefficients of a first filter of the filters by a weight (w).
[0056] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the weight is 0.5 or 1.5.
[0057] Optionally, in any of the preceding aspects, another implementation of the aspect provides scaling all coefficients of a first filter of the filters by a weight (w), wherein the weight is identical for all of the coefficients.
[0058] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the filters are included in the bitstream for each CCALF processing unit.
[0059] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the weight is included in the bitstream for each CCALF processing unit.
[0060] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the weight is pre-defined at an encoder and at a decoder for each CCALF processing unit.
[0061] Optionally, in any of the preceding aspects, another implementation of the aspect provides that a weight index of the weight is included in the bitstream for each CCALF processing unit.
[0062] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the weight is determined in real time for each CCALF processing unit.
[0063] Optionally, in any of the preceding aspects, another implementation of the aspect provides

scaling two different coefficients of a first filter from the filters by two different weights.
[0064] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different weights are included in the bitstream for each CCALF processing unit.
[0065] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different weights are pre-defined at an encoder and at a decoder for each CCALF processing unit.
[0066] Optionally, in any of the preceding aspects, another implementation of the aspect provides that weight indexes are included in the bitstream for each CCALF processing unit.
[0067] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different weights are determined in real time for each CCALF processing unit.
[0068] Optionally, in any of the preceding aspects, another implementation of the aspect provides adding an offset (k) to one or more coefficients of a first filter of the filters.
[0069] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is 10 or −10.
[0070] Optionally, in any of the preceding aspects, another implementation of the aspect provides adding an offset (k) to all coefficients of a first filter of the filters, wherein the offset is identical for all of the coefficients.
[0071] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is included in the bitstream for each CCALF processing unit.
[0072] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is included in the bitstream for each CCALF processing unit.
[0073] Optionally, in any of the preceding aspects, another implementation of the aspect provides that an offset index for the offset is included in the bitstream for each CCALF processing unit.
[0074] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is determined in real time for each CCALF processing unit.
[0075] Optionally, in any of the preceding aspects, another implementation of the aspect provides adding two different offsets to two different coefficients of a first filter from the filters.
[0076] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different offsets are included in the bitstream for each CCALF processing unit.
[0077] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different offsets are pre-defined at an encoder and at a decoder for each CCALF processing unit.
[0078] Optionally, in any of the preceding aspects, another implementation of the aspect provides that offset indexes for the offsets are included in the bitstream for each CCALF processing unit.
[0079] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different offsets are determined in real time for each CCALF processing unit.
[0080] Optionally, in any of the preceding aspects, another implementation of the aspect provides adding an offset (t) to one or more clipping parameters of a first filter of the filters.
[0081] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is +1 or −1.
[0082] Optionally, in any of the preceding aspects, another implementation of the aspect provides adding an offset (t) to all clipping parameters of a first filter of the filters, wherein the offset is identical for all of the clipping parameters.
[0083] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is included in the bitstream for each CCALF processing unit.
[0084] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is pre-defined at an encoder and at a decoder for each CCALF processing unit.
[0085] Optionally, in any of the preceding aspects, another implementation of the aspect provides that an offset index for the offset is included in the bitstream for each CCALF processing unit.
[0086] Optionally, in any of the preceding aspects, another implementation of the aspect provides

that the offset is determined in real time for each CCALF processing unit.
[0087] Optionally, in any of the preceding aspects, another implementation of the aspect provides adding two different offsets to two different clipping parameters of a first filter from the filters.
[0088] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different offsets are included in the bitstream for each CCALF processing unit.
[0089] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different offsets are pre-defined at an encoder and at a decoder for each CCALF processing unit.
[0090] Optionally, in any of the preceding aspects, another implementation of the aspect provides that offset indexes for the offsets are included in the bitstream for each CCALF processing unit.
[0091] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the two different offsets are determined in real time for each CCALF processing unit.
[0092] Optionally, in any of the preceding aspects, another implementation of the aspect provides scaling an output of a first filter of the filters by a weight.
[0093] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the weight is included in the bitstream for each CCALF processing unit.
[0094] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the weight is pre-defined at an encoder and at a decoder for each CCALF processing unit.
[0095] Optionally, in any of the preceding aspects, another implementation of the aspect provides that two different weight indexes are included in the bitstream for two different CCALF processing units.
[0096] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the weight is determined in real time for each ALF processing unit.
[0097] Optionally, in any of the preceding aspects, another implementation of the aspect provides adding an offset to an output of a first filter of the filters.
[0098] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is included in the bitstream for each CCALF processing unit.
[0099] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is pre-defined at an encoder and at a decoder for each CCALF processing unit.
[0100] Optionally, in any of the preceding aspects, another implementation of the aspect provides that an offset index for the offset is included in the bitstream for each CCALF processing unit.
[0101] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the offset is determined in real time for each CCALF processing unit.
[0102] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the current CCALF processing unit is a blue difference chroma (Cb) component.
[0103] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the current CCALF processing unit is a red difference chroma (Cr) component.
[0104] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the method is in implemented in post-processing or in pre-processing.
[0105] Optionally, in any of the preceding aspects, another implementation of the aspect provides that one or more of the methods are used jointly.
[0106] Optionally, in any of the preceding aspects, another implementation of the aspect provides that one or more of the methods are used individually.
[0107] Optionally, in any of the preceding aspects, another implementation of the aspect provides converting a first value to a second value prior to encoding the second value in the bitstream, and converting the second value back to the first value after parting.
[0108] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the conversion comprises quantization and/or dequantization.
[0109] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the conversion comprises implementing a transform and/or an inverse transform.

[0110] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the conversion is used jointly.
[0111] Optionally, in any of the preceding aspects, another implementation of the aspect provides that information for the method is included in the bitstream as one or more syntax elements (SEs).
[0112] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the one or more SEs are predictively coded.
[0113] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the one or more SEs are context coded with at least one context.
[0114] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the one or more SEs are bypass coded.
[0115] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the method is applied to any in-loop filtering tools, a pre-processing filtering method, or post-processing filtering method in video coding including but not limited to ALF/CCALF or any other filtering method.
[0116] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the method is applied to an in-loop filtering method.
[0117] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the method is applied to ALF, CCAPF, a sample adaptive offset (SAO) filter, or another in-loop filtering method.
[0118] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the method is applied to a pre-processing filtering method.
[0119] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the method is applied to a post-processing filtering method.
[0120] Optionally, in any of the preceding aspects, another implementation of the aspect provides that a video unit corresponding to the current ALF processing unit or the CCALF processing unit comprises one of a sequence, a picture, a sub-picture, a slice, a tile, a coding tree unit (CTU), a CTU row, groups of CTUs, a coding unit (CU), a prediction unit (PU), a transform unit (TU), a coding tree block (CTB), a coding block (CB), a prediction block (PB), a transform block (TB), and any other region that contains more than one luma or chroma sample or pixel.
[0121] Optionally, in any of the preceding aspects, another implementation of the aspect provides that an indication of whether and/or how to apply the method is included in the bitstream.
[0122] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the indication is included in the bitstream at a sequence level, at a group of pictures level, at a picture level, at a slice level, at a tile group level, in a sequence header, in a picture header, in a sequence parameter set (SPS), in a video parameter set (VPS), in a dependency parameter set (DPS), in a decoding capability information (DCI), in a picture parameter set (PPS), in an adaptation parameter set (APS), in a slice header, or in a tile group header.
[0123] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the indication is included in the bitstream in a prediction block (PB), a transform block (TB), a coding block (CB), a prediction unit (PU), a transform unit (TU), a coding unit (CU), a virtual pipeline data unit (VPDU), a coding tree unit (CTU), a CTU row, a slice, a tile, a sub-picture, or any other region that contains more than one sample or pixel.
[0124] Optionally, in any of the preceding aspects, another implementation of the aspect provides that whether and/or how to apply any of the methods is dependent on coded information, and wherein the coded information comprises block size, color format, a single or dual tree partitioning, a color component, a slice type, or a picture type.
[0125] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the conversion includes encoding the visual media data into the bitstream.
[0126] Optionally, in any of the preceding aspects, another implementation of the aspect provides that the conversion includes decoding the visual media data from the bitstream.

[0127] A second aspect relates to an apparatus for processing media data comprising: one or more processors; and a non-transitory memory with instructions thereon, wherein the instructions upon execution by the processor cause the apparatus to perform the method in any of the disclosed embodiments.
[0128] A third aspect relates to a non-transitory computer readable medium, comprising a computer program product for use by a video coding device, the computer program product comprising computer executable instructions stored on the non-transitory computer readable medium such that when executed by a processor cause the video coding device to perform the method in any of the disclosed embodiments.
[0129] A fourth aspect relates to a non-transitory computer-readable recording medium storing a bitstream of a video which is generated by a method performed by a video processing apparatus, wherein the method comprises the method in any of the disclosed embodiments.
[0130] A fifth aspect relates to a method for storing a bitstream of a video comprising the method in any of the disclosed embodiments.
[0131] A sixth aspect relates to a method, apparatus, or system described in the present disclosure.
[0132] For the purpose of clarity, any one of the foregoing embodiments may be combined with any one or more of the other foregoing embodiments to create a new embodiment within the scope of the present disclosure.
[0133] These and other features will be more clearly understood from the following detailed description taken in conjunction with the accompanying drawings and claims.

Description

BRIEF DESCRIPTION OF THE DRAWINGS
[0134] For a more complete understanding of this disclosure, reference is now made to the following brief description, taken in connection with the accompanying drawings and detailed description, wherein like reference numerals represent like parts.
[0135] FIG. **1** illustrates an example of nominal vertical and horizontal locations of 4:2:2 luma and chroma samples in a picture.
[0136] FIG. **2** illustrates an example encoder block diagram.
[0137] FIG. **3** illustrates an example picture partitioned into raster scan slices.
[0138] FIG. **4** illustrates an example picture partitioned into rectangular scan slices.
[0139] FIG. **5** illustrates an example picture partitioned into bricks.
[0140] FIGS. **6**A-**6**C illustrate examples of coding tree blocks (CTBs) crossing picture borders.
[0141] FIG. **7** illustrates an example of intra prediction modes.
[0142] FIG. **8** illustrates an example of block boundaries in a picture.
[0143] FIG. **9** illustrates an example of pixels involved in filter usage.
[0144] FIG. **10** illustrates an example of filter shapes for adaptive loop filter (ALF).
[0145] FIG. **11** illustrates an example of transformed coefficients for 5×5 diamond filter support.
[0146] FIG. **12** illustrates an example of relative coordinates for 5×5 diamond filter support.
[0147] FIG. **13** is a block diagram showing an example video processing system.
[0148] FIG. **14** is a block diagram of an example video processing apparatus.
[0149] FIG. **15** is a flowchart for an example method of video processing.
[0150] FIG. **16** is a block diagram that illustrates an example video coding system.
[0151] FIG. **17** is a block diagram that illustrates an example encoder.
[0152] FIG. **18** is a block diagram that illustrates an example decoder.
[0153] FIG. **19** is a schematic diagram of an example encoder.
[0154] FIG. **20** is a method for processing video data in accordance with an embodiment of the disclosure.

DETAILED DESCRIPTION

[0155] It should be understood at the outset that although an illustrative implementation of one or more embodiments are provided below, the disclosed systems and/or methods may be implemented using any number of techniques, whether currently known or yet to be developed. The disclosure should in no way be limited to the illustrative implementations, drawings, and techniques illustrated below, including the exemplary designs and implementations illustrated and described herein, but may be modified within the scope of the appended claims along with their full scope of equivalents.

[0156] Section headings are used in the present disclosure for ease of understanding and do not limit the applicability of techniques and embodiments disclosed in each section only to that section. Furthermore, the techniques described herein are applicable to other video codec protocols and designs.

1. Initial Discussion

[0157] This disclosure is related to video coding technologies. Specifically, it is related to in-loop filter and other coding tools in image/video coding. The ideas may be applied individually or in various combinations to video codecs, such as High Efficiency Video Coding (HEVC), Versatile Video Coding (VVC), or other video coding technologies.

2. Abbreviations

[0158] The present disclosure includes the following abbreviations. Advanced video coding (Rec. International Telecommunication Union (ITU) telecommunication standardization sector (ITU-T) H.264|ISO/IEC 14496-10) (AVC), coded picture buffer (CPB), clean random access (CRA), coding tree unit (CTU), coded video sequence (CVS), decoded picture buffer (DPB), decoding parameter set (DPS), general constraints information (GCI), high efficiency video coding, also known as Rec. ITU-T H.265| International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 23008-2, (HEVC), Joint exploration model (JEM), motion constrained tile set (MCTS), network abstraction layer (NAL), output layer set (OLS), picture header (PH), picture parameter set (PPS), profile, tier, and level (PTL), picture unit (PU), reference picture resampling (RPR), raw byte sequence payload (RBSP), supplemental enhancement information (SEI), slice header (SH), sequence parameter set (SPS), video coding layer (VCL), video parameter set (VPS), versatile video coding, also known as Rec. ITU-T H.266|ISO/IEC 23090-3, (VVC), VVC test model (VTM), video usability information (VUI), transform unit (TU), coding unit (CU), deblocking filter (DF), sample adaptive offset (SAO), adaptive loop filter (ALF), coding block flag (CBF), quantization parameter (QP), rate distortion optimization (RDO), and bilateral filter (BF).

3. Video Coding Standards

[0159] Video coding standards have evolved primarily through the development of the ITU-T and ISO/IEC standards. The ITU-T produced H.261 and H.263, ISO/IEC produced Moving Picture Experts Group (MPEG)-1 and MPEG-4 Visual, and the two organizations jointly produced the H.262/MPEG-2 Video and H.264/MPEG-4 Advanced Video Coding (AVC) and H.265/HEVC [1] standards. Since H.262, the video coding standards are based on the hybrid video coding structure wherein temporal prediction plus transform coding are utilized. To explore the future video coding technologies beyond HEVC, the Joint Video Exploration Team (JVET) was founded by Video Coding Experts Group (VCEG) and MPEG jointly. Many methods have been adopted by JVET and put into the reference software named Joint Exploration Model (JEM) [2]. The JVET was renamed to be the Joint Video Experts Team (JVET) when the Versatile Video Coding (VVC) project officially started. VVC is a coding standard, targeting at 50% bitrate reduction as compared to HEVC. The VVC working draft and VVC test model (VTM) are continuously updated.

[0160] An example version of the VVC draft, i.e., Versatile Video Coding (Draft 10) may be found at: https://jvet-experts.org/doc_end_user/documents/19_Teleconference/wg11/JVET-S2001-v17.zip. An example version of the reference software of VVC, named as VTM, could be found at: https://vcgit.hhi.fraunhofer.de/jvet-u-ee2/VVCSoftware_VTM/-/tree/VTM-11.2.

[0161] International Telecommunication Union Telecommunication Standardization Sector (ITU-T) video coding experts group (VCEG) and International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC) Moving Picture Experts Group (MPEG) joint technical committee (JTC) 1/subcommittee (SC) 29/working group (WG) 11 are studying the potential need for standardization of future video coding technology with a compression capability that significantly exceeds that of the current VVC standard. Such future standardization action could either take the form of extended extension(s) of VVC or an entirely new standard. The groups are working together on this exploration activity in a joint-collaboration effort known as the Joint Video Exploration Team (JVET) to evaluate compression technology designs proposed by their experts in this area. The first Exploration Experiments (EE) are established by JVET and reference software named Enhanced Compression Model (ECM) is in use. The test model ECM is updated continuously.

3.1 Color Space and Chroma Subsampling

[0162] Color space, also known as the color model (or color system), is a mathematical model which describes the range of colors as tuples of numbers, for example as 3 or 4 values or color components (e.g., RGB). Generally speaking, a color space is an elaboration of the coordinate system and sub-space. For video compression, the most frequently used color spaces are luma, blue difference chroma, and red difference chroma (YCbCr) and red, green, blue (RGB).

[0163] YCbCr, Y′CbCr, or Y Pb/Cb Pr/Cr, also written as YCBCR or Y′CBCR, is a family of color spaces used as a part of the color image pipeline in video and digital photography systems. Y′ is the luma component and CB and CR are the blue-difference and red-difference chroma components. Y′ (with prime) is distinguished from Y, which is luminance, meaning that light intensity is nonlinearly encoded based on gamma corrected RGB primaries.

[0164] Chroma subsampling is the practice of encoding images by implementing less resolution for chroma information than for luma information, taking advantage of the human visual system's lower acuity for color differences than for luminance.

3.1.1 4:4:4

[0165] In 4:4:4, each of the three Y′CbCr components have the same sample rate. Thus there is no chroma subsampling. This scheme is sometimes used in high-end film scanners and cinematic postproduction.

3.1.2 4:2:2

[0166] In 4:3:2, the two chroma components are sampled at half the sample rate of luma. The horizontal chroma resolution is halved while the vertical chroma resolution is unchanged. This reduces the bandwidth of an uncompressed video signal by one-third with little to no visual difference. An example of nominal vertical and horizontal locations of 4:2:2 color format is depicted in FIG. **1**.

3.1.3 4:2:0

[0167] In 4:2:0, the horizontal sampling is doubled compared to 4:1:1, but as the Cb and Cr channels are only sampled on each alternate line in this scheme, the vertical resolution is halved. The data rate is thus the same. Cb and Cr are each subsampled at a factor of 2 both horizontally and vertically. There are three variants of 4:2:0 schemes, having different horizontal and vertical siting. In MPEG-2, Cb and Cr are cosited horizontally. Cb and Cr are sited between pixels in the vertical direction (sited interstitially). In Joint Photographic Experts Group (JPEG)/JPEG File Interchange Format (JFIF), H.261, and MPEG-1, Cb and Cr are sited interstitially, halfway between alternate luma samples. In 4:2:0 DV, Cb and Cr are co-sited in the horizontal direction. In the vertical direction, they are co-sited on alternating lines.

TABLE-US-00001 TABLE 1 SubWidthC and SubHeightC values derived from chroma_format_idc and separate_colour_plane_flag

chroma_format_idc	separate_colour_plane_flag	Chroma format	SubWidthC	SubHeightC
0	0	Monochrome	1	1
1	0	4:2:0	2	2
2	0	4:2:2	2	1
3	0	4:4:4	1	1
3	1	4:4:4	1	1

3.2 Example Coding Flow of a Video Codec

[0168] FIG. **2** shows an example of encoder block diagram of VVC, which contains three in-loop filtering blocks: deblocking filter (DF), sample adaptive offset (SAO) and ALF. Unlike DF, which uses predefined filters, SAO and ALF utilize the original samples of the current picture to reduce the mean square errors between the original samples and the reconstructed samples by adding an offset and by applying a finite impulse response (FIR) filter, respectively, with coded side information signaling the offsets and filter coefficients. ALF is located at the last processing stage of each picture and can be regarded as a tool trying to catch and fix artifacts created by the previous stages.

3.3 Definitions of Video/Coding Units

[0169] A picture is divided into one or more tile rows and one or more tile columns. A tile is a sequence of CTUs that covers a rectangular region of a picture. A tile may be divided into one or more bricks, each of which includes a number of CTU rows within the tile. A tile that is not partitioned into multiple bricks may also be referred to as a brick. However, a brick that is a true subset of a tile may not be referred to as a tile. A slice either contains several tiles of a picture or several bricks of a tile.

[0170] Two modes of slices are supported, namely the raster-scan slice mode and the rectangular slice mode. In the raster-scan slice mode, a slice contains a sequence of tiles in a tile raster scan of a picture. In the rectangular slice mode, a slice contains a number of bricks of a picture that collectively form a rectangular region of the picture. The bricks within a rectangular slice are in the order of brick raster scan of the slice. FIG. **3** shows an example of raster-scan slice partitioning of a picture, where the picture is divided into 12 tiles and 3 raster-scan slices.

[0171] FIG. **4** shows an example of rectangular slice partitioning of a picture, where the picture is divided into 24 tiles (6 tile columns and 4 tile rows) and 9 rectangular slices.

[0172] FIG. **5** shows an example of a picture partitioned into tiles, bricks, and rectangular slices, where the picture is divided into 4 tiles (2 tile columns and 2 tile rows), 11 bricks (the top-left tile contains 1 brick, the top-right tile contains 5 bricks, the bottom-left tile contains 2 bricks, and the bottom-right tile contain 3 bricks), and 4 rectangular slices.

3.3.1 CTU/CTB Sizes

[0173] In VVC, the CTU size, signaled in a sequence parameter set (SPS) by the syntax element log 2_ctu_size_minus2, could be as small as 4×4.

7.3.2.3 Sequence Parameter Set RBSP Syntax

TABLE-US-00002 Descriptor seq_parameter_set_rbsp( ) { sps_decoding_parameter_set_id u(4) sps_video_parameter_set_id u(4) sps_max_sub_layers_minus1 u(3) sps_reserved_zero_5bits u(5) profile_tier_level(sps_max_sub_layers_minus1) gra_enabled_flag u(1) sps_seq_parameter_set_id ue(v) chroma_format_idc ue(v) if(chroma_format_idc = = 3) separate_colour_plane_flag u(1) pic_width_in_luma_samples ue(v) pic_height_in_luma_samples ue(v) conformance_window_flag u(1) if(conformance_window_flag) { conf_win_left_offset ue(v) conf_win_right_offset ue(v) conf_win_top_offset ue(v) conf_win_bottom_offset ue(v) } bit_depth_luma_minus8 ue(v) bit_depth_chroma_minus8 ue(v) log2_max_pic_order_cnt_lsb_minus4 ue(v) sps_sub_layer_ordering_info_present_flag u(1) for(i = (sps_sub_layer_ordering_info_present_flag ? 0 : sps_max_sub_layers_minus1); i <= sps_max_sub_layers_minus1; i++) { sps_max_dec_pic_buffering_minus1[i] ue(v) sps_max_num_reorder_pics[i] ue(v) sps_max_latency_increase_plus1[i] ue(v) } long_term_ref_pics_flag u(1) sps_idr_rpl_present_flag u(1) rpl1_same_as_rp10_flag u(1) for(i = 0; i < !rpl1_same_as_rpl0_flag ? 2 : 1; i++) { num_ref_pic_lists_in_sps[i] ue(v) for(j = 0; j < num_ref_pic_lists_in_sps[i]; j++) ref_pic_list_struct(i, j) } qtbtt_dual_tree_intra_flag u(1) log2_ctu_size_minus2 ue(v) log2_min_luma_coding_block_size_minus2 ue(v) partition_constraints_override_enabled_flag u(1) sps_log2_diff_min_qt_min_cb_intra_slice_luma ue(v) sps_log2_diff_min_qt_min_cb_inter_slice ue(v)   sps_max_mtt_hierarchy_depth_inter_slice ue(v)   sps_max_mtt_hierarchy_depth_intra_slice_luma ue(v)   if(sps_max_mtt_hierarchy_depth_intra_slice_luma != 0) { sps_log2_diff_max_bt_min_qt_intra_slice_luma ue(v) sps_log2_diff_max_tt_min_qt_intra_slice_luma ue(v)   }   if(sps_max_mtt_hierarchy_depth_inter_slices != 0) {   sps_log2_diff_max_bt_min_qt_inter_slice ue(v)   sps_log2_diff_max_tt_min_qt_inter_slice ue(v)   }   if(qtbtt_dual_tree_intra_flag) {   sps_log2_diff_min_qt_min_cb_intra_slice_chroma ue(v) sps_max_mtt_hierarchy_depth_intra_slice_chroma ue(v)   if (sps_max_mtt_hierarchy_depth_intra_slice_chroma != 0) { sps_log2_diff_max_bt_min_qt_intra_slice_chroma ue(v) sps_log2_diff_max_tt_min_qt_intra_slice_chroma ue(v)   }   } . . .   rbsp_trailing_bits( ) }

[0174] log 2_ctu_size_minus2 plus 2 specifies the luma coding tree block size of each CTU. log 2_min_luma_coding_block_size_minus2 plus 2 specifies the minimum luma coding block size. The variables CtbLog2SizeY, CtbSizeY, MinCbLog2SizeY, MinCbSizeY, MinTbLog2SizeY, MaxTbLog2SizeY, MinTbSizeY, MaxTbSizeY, PicWidthInCtbsY, PicHeightInCtbsY, PicSizeInCtbsY, Pic WidthInMinCbsY, PicHeightInMinCbsY, PicSizeInMinCbsY, PicSizeInSamplesY, PicWidthInSamplesC and PicHeightInSamplesC are derived as follows:

[00001] CtbLog2SizeY = log2_ctu_size_minus2 + 2   (7 - 9)

CtbSizeY = 1   CtbLog2SizeY   (7 - 10)

MinCbLog2SizeY = log2_min_luma_coding_block_size_minus2 + 2   (7 - 11)

MinCbSizeY = 1   MinCbLog2SizeY   (7 - 12)   MinTbLog2SizeY = 2   (7 - 13)

MaxTbLog2SizeY = 6   (7 - 14)   MinTbSizeY = 1   MinTbLog2SizeY   (7 - 15)

MaxTbSizeY = 1   MaxTbLog2SizeY   (7 - 16)

PicWidthInCtbsY = Ceil(pic_width_in_luma_samples ÷ CtbSizeY)   (7 - 17)

PicHeightInCtbsY = Ceil(pic_height_in_luma_samples ÷ CtbSizeY)   (7 - 18)

PicSizeInCtbsY = PicWidthInCtbsY * PicHeightInCtbsY   (7 - 19)

PicWidthInMinCbsY = pic_width_in_luma_samples / MinCbSizeY   (7 - 20)

PicHeightInMinCbsY = pic_height_in_luma_samples / MinCbSizeY   (7 - 21)

PicSizeInMinCbsY = PicWidthInMinCbsY * PicHeightInMinCbsY   (7 - 22)

PicSizeInSamplesY = pic_width_in_luma_samples * pic_height_in_luma_samples   (7 - 23)

PicWidthInSamplesC = pic_width_in_luma_samples / SubWidthC   (7 - 24)

PicHeightInSamplesC = pic_height_in_luma_samples / SubHeightC   (7 - 25)

3.3.2 CTUs in One Picture

[0175] Suppose the CTB/largest coding unit (LCU) size indicated by M×N (typically M is equal to N), and for a CTB located at picture border (or tile or slice or other types of borders, picture border is taken as an example) border, K×L samples are within picture border wherein either K<M or L<N. FIGS. **6**A-**6**C illustrate examples of CTBs crossing picture borders. For those CTBs as depicted in FIGS. **6**A-**6**C, the CTB size is still equal to M×N, however, the bottom boundary/right boundary of the CTB is outside the picture.

3.4 Intra Prediction

[0176] To capture the arbitrary edge directions presented in natural video, the number of directional intra modes is extended from 33, as used in HEVC, to 65. FIG. **7** illustrates an example of 67 intra prediction modes. The extended directional modes are depicted in FIG. **7**, and the planar and direct current (DC) modes remain the same. These denser directional intra prediction modes apply for all block sizes and for both luma and chroma intra predictions.

[0177] Angular intra prediction directions may be defined from 45 degrees to −135 degrees in clockwise direction as shown in FIG. **7**. In VTM, several angular intra prediction modes are adaptively replaced with wide-angle intra prediction modes for the non-square blocks. The replaced modes are signaled and remapped to the indexes of wide angular modes after parsing. The total number of intra prediction modes is unchanged, e.g., 67, and the intra mode coding is unchanged.

[0178] In the HEVC, every intra-coded block has a square shape and the length of each of the block's sides is a power of 2. Thus, no division operations are required to generate an intra-predictor using DC mode. In VVC, blocks can have a rectangular shape that necessitates the use of a division operation per block in the general case. To avoid division operations for DC prediction, only the longer side is used to compute the average for non-square blocks.

3.5 Inter Prediction

[0179] For each inter-predicted CU, motion parameters include motion vectors, reference picture indices, reference picture list usage index, and extended information used for the new coding feature of VVC to be used for inter-predicted sample generation. The motion parameters can be signaled in an explicit or implicit manner. When a CU is coded with skip mode, the CU is associated with one PU and has no significant residual coefficients, no coded motion vector delta, and/or reference picture index. A merge mode is specified whereby the motion parameters for the current CU are obtained from neighboring CUs, including spatial and temporal candidates, and extended schedules introduced in VVC. The merge mode can be applied to any inter-predicted CU, not only for skip mode. The alternative to merge mode is the explicit transmission of motion parameters, where motion vector, corresponding reference picture index for each reference picture list, reference picture list usage flag, and other useful information are signaled explicitly per each CU.

3.6 Deblocking Filter

[0180] Deblocking filtering is an example in-loop filter in video codec. In VVC, the deblocking filtering process is applied on CU boundaries, transform subblock boundaries, and prediction subblock boundaries. The prediction subblock boundaries include the prediction unit boundaries introduced by the Subblock based Temporal Motion Vector prediction (SbTMVP) and affine modes. The transform subblock boundaries include the transform unit boundaries introduced by Subblock transform (SBT) and Intra Sub-Partitions (ISP) modes and transforms due to implicit split of large CUs. The processing order of the deblocking filter is defined as horizontal filtering for vertical edges for the entire picture first, followed by vertical filtering for horizontal edges. This specific order enables either multiple horizontal filtering or vertical filtering processes to be applied in parallel threads. Filtering processes can also be implemented on a CTB-by-CTB basis with only a small processing latency.

[0181] The vertical edges in a picture are filtered first. Then the horizontal edges in a picture are filtered with samples modified by the vertical edge filtering process as input. The vertical and horizontal edges in the CTBs of each CTU are processed separately on a coding unit basis. The vertical edges of the coding blocks in a coding unit are filtered starting with the edge on the left-hand side of the coding blocks proceeding through the edges towards the right-hand side of the coding blocks in their geometrical order. The horizontal edges of the coding blocks in a coding unit are filtered starting with the edge on the top of the coding blocks proceeding through the edges towards the bottom of the coding blocks in their geometrical order. FIG. **8** picture samples and horizontal and vertical block boundaries on the 8×8 grid, and the nonoverlapping blocks of the 8×8 samples, which can be deblocked in parallel.

3.6.1 Boundary Decision

[0182] Filtering is applied to 8×8 block boundaries. In addition, such boundaries must be a transform block boundary or a coding subblock boundary, for example due to usage of Affine motion prediction (ATMVP). For other boundaries, deblocking filtering is disabled.

3.6.2 Boundary Strength Calculation
[0183] For a transform block boundary/coding subblock boundary, if the boundary is located in the 8×8 grid, the boundary may be filtered and the setting of bS [xDi][yDj] (wherein [xDi][yDj] denotes the coordinate) for this edge as defined in Table 2 and Table 3, respectively.
TABLE-US-00003 TABLE 2 Boundary strength (when SPS IBC is disabled) Priority Conditions Y U V 5 At least one of the adjacent blocks is intra 2 2 2 4 TU boundary and at least one of the adjacent 1 1 1 blocks has non-zero transform coefficients 3 Reference pictures or number of MVs (1 for 1 N/A N/A uni-prediction, 2 for bi-prediction) of the adjacent blocks are different 2 Absolute difference between the motion 1 N/A N/A vectors of same reference picture that belong to the adjacent blocks is greater than or equal to one integer luma sample 1 Otherwise 0 0 0
TABLE-US-00004 TABLE 3 Boundary strength (when SPS IBC is enabled) Priority Conditions Y U V 8 At least one of the adjacent blocks is intra 2 2 2 7 TU boundary and at least one of the adjacent 1 1 1 blocks has non-zero transform coefficients 6 Prediction mode of adjacent blocks is 1 different (e.g., one is IBC, one is inter) 5 Both IBC and absolute difference between 1 N/A N/A the motion vectors that belong to the adjacent blocks is greater than or equal to one integer luma sample 4 Reference pictures or number of MVs (1 for 1 N/A N/A uni-prediction, 2 for bi-prediction) of the adjacent blocks are different 3 Absolute difference between the motion 1 N/A N/A vectors of same reference picture that belong to the adjacent blocks is greater than or equal to one integer luma sample 1 Otherwise 0 0 0
3.6.3 Deblocking Decision for Luma Component
[0184] FIG. **9** illustrates pixels involved in filter on/off decision and strong/weak filter selection. Wider-stronger luma filter is filters are used only if all the Condition1, Condition2 and Condition 3 are TRUE. The condition 1 is the “large block condition”. This condition detects whether the samples at P-side and Q-side belong to large blocks, which are represented by the variable bSidePisLargeBlk and bSideQisLargeBlk respectively. The bSidePisLargeBlk and bSideQisLargeBlk are defined as follows. [0185] bSidePisLargeBlk=((edge type is vertical and p.sub.0 belongs to CU with width >=32)‖(edge type is horizontal and p.sub.0 belongs to CU with height >=32))? TRUE: FALSE [0186] bSideQisLargeBlk=((edge type is vertical and q.sub.0 belongs to CU with width >=32)‖(edge type is horizontal and q.sub.0 belongs to CU with height >=32))? TRUE: FALSE
[0187] Based on bSidePisLargeBlk and bSideQisLargeBlk, the condition 1 is defined as follows:
[0188] Condition1=(bSidePisLargeBlk‖bSidePisLargeBlk)? TRUE: FALSE
[0189] Next, if Condition 1 is true, the condition 2 will be further checked. First, the following variables are derived: [0190] dp0, dp3, dq0, dq3 are first derived as in HEVC [0191] if (p side is greater than or equal to 32)
[00002] $dp0 = (dp0 + Abs(p50 - 2 * p40 + p30) + 1) \gg 1$
$dp3 = (dp3 + Abs(p53 - 2 * p43 + p33) + 1) \gg 1$ [0192] if (q side is greater than or equal to 32)
[00003] $dq0 = (dq0 + Abs(q50 - 2 * q40 + q30) + 1) \gg 1$
$dq3 = (dq3 + Abs(q53 - 2 * q43 + q33) + 1) \gg 1$ [0193] Condition2=(d<B)? TRUE: FALSE
[0194] where d=dp0+dq0+dp3+dq3.
[0195] If Condition1 and Condition2 are valid, whether any of the blocks uses sub-blocks is further checked:
```
TABLE-US-00005 If (bSidePisLargeBlk)     {               If (mode block P == SUBBLOCKMODE)
                  Sp =5                 else                     Sp =7 } else     Sp = 3 If (bSideQisLargeBlk)
      {               If (mode block Q == SUBBLOCKMODE)                        Sq =5                else
                    Sq =7            } else       Sq = 3
```
[0196] Finally, if both the Condition 1 and Condition 2 are valid, the deblocking method will check the condition 3 (the large block strong filter condition), which is defined as follows. In the Condition3 StrongFilterCondition, the following variables are derived:
TABLE-US-00006 dpq is derived as in HEVC. sp3 = Abs(p3 − p0), derived as in HEVC if (p side

is greater than or equal to 32) if(Sp==5) sp3 = (sp3 + Abs(p5 − p3) + 1) >> 1 else sp3 = (sp3 + Abs(p7 − p3) + 1) >> 1 sq3 = Abs(q0 − q3), derived as in HEVC if (q side is greater than or equal to 32) If(Sq==5) sq3 = (sq3 + Abs(q5 − q3) + 1) >> 1 else sq3 = (sq3 + Abs(q7 − q3) + 1) >> 1

[0197] As in HEVC, StrongFilterCondition=(dpq is less than (β>>2), sp3+sq3 is less than (3*β>>5), and Abs (p0−q0) is less than (5*tC+1)>>1)? TRUE: FALSE.

3.6.4 Stronger Deblocking Filter for Luma

[0198] Bilinear filter is used when samples at either one side of a boundary belong to a large block. A sample belonging to a large block is defined as when the width >=32 for a vertical edge, and when height >=32 for a horizontal edge. The bilinear filter is listed below. Block boundary samples pi for i=0 to Sp-1 and qi for j=0 to Sq-1 (pi and qi are the i-th sample within a row for filtering vertical edge, or the i-th sample within a column for filtering horizontal edge) in HEVC deblocking described above) are then replaced by linear interpolation as follows:

[00004]$p_i' = (f_i * \mathrm{Middle}_{s,t} + (64 - f_i) * P_s + 32) \gg 6), \mathrm{clippedto}\, p_i \pm \mathrm{tcPD}_i$

$q_j' = (g_j * \mathrm{Middle}_{s,t} + (64 - g_j) * Q_s + 32) \gg 6), \mathrm{clippedto}\, q_j \pm \mathrm{tcPD}_j$

where tcPD.sub.i and tcPD.sub.j term is a position dependent clipping described above and g.sub.j, f.sub.i, Middle.sub.s,t, P.sub.s and Q.sub.s are given below:

3.6.5 Deblocking Decision for Chroma

[0199] The chroma strong filters are used on both sides of the block boundary. Here, the chroma filter is selected when both sides of the chroma edge are greater than or equal to 8 (chroma position), and the following decision with three conditions are satisfied: the first one is for decision of boundary strength as well as large block. The filter can be applied when the block width or height which orthogonally crosses the block edge is equal to or larger than 8 in chroma sample domain. The second and third one is basically the same as for HEVC luma deblocking decision, which are on/off decision and strong filter decision, respectively.

[0200] In the first decision, boundary strength (bS) is modified for chroma filtering and the conditions are checked sequentially. If a condition is satisfied, then the remaining conditions with lower priorities are skipped. Chroma deblocking is performed when bS is equal to 2, or bS is equal to 1 when a large block boundary is detected. The second and third condition is basically the same as HEVC luma strong filter decision as follows.

[0201] In the second condition d is then derived as in HEVC luma deblocking. The second condition will be TRUE when d is less than β. In the third condition StrongFilterCondition is derived as follows: [0202] dpq is derived as in HEVC. [0203] sp3=Abs (p.sub.3-p.sub.0), derived as in HEVC [0204] sq3=Abs (q.sub.0-q.sub.3), derived as in HEVC

[0205] As in HEVC design, StrongFilterCondition=(dpq is less than (β>>2), sp3+sq3 is less than (B>>3), and Abs (p0-q0) is less than (5*tC+1)>>1)

3.6.6 Strong Deblocking Filter for Chroma

[0206] The following strong deblocking filter for chroma is defined:

[00005]$p2' = (3 * p3 + 2 * p2 + p1 + p0 + q0 + 4) \gg 3$

$p1' = (2 * p3 + p2 + 2 * p1 + p0 + q0 + q1 + 4) \gg 3$

$p0' = (p3 + p2 + p1 + 2 * p0 + q0 + q1 + q2 + 4) \gg 3$

[0207] An example chroma filter performs deblocking on a 4×4 chroma sample grid.

3.6.7 Position Dependent Clipping

[0208] The position dependent clipping tcPD is applied to the output samples of the luma filtering process involving strong and long filters that are modifying 7, 5 and 3 samples at the boundary. Assuming quantization error distribution, a clipping value may be increased for samples which are expected to have higher quantization noise, thus expected to have higher deviation of the reconstructed sample value from the true sample value.

[0209] For each P or Q boundary filtered with asymmetrical filter, depending on the result of

decision-making process, position dependent threshold table is selected from two tables (e.g., Tc7 and Tc3 tabulated below) that are provided to decoder as a side information: [0210] Tc7={6, 5, 4, 3, 2, 1, 1}; Tc3={6, 4, 2}; [0211] tcPD=(Sp==3)? Tc3: Tc7; [0212] tcQD=(Sq==3)? Tc3: Tc7;
[0213] For the P or Q boundaries being filtered with a short symmetrical filter, position dependent threshold of lower magnitude is applied: [0214] Tc3={3,2,1};
[0215] Following defining the threshold, filtered p′i and q′i sample values are clipped according to tcP and tcQ clipping values:
[00006]$p''i$ = Clip3($p'i$ + tcPi, $p'i$ - tcPi, $p'i$); $q''j$ = Clip3($q'j$ + tcQj, $q'j$ - tcQj, $q'j$);
where p′i and q′i are filtered sample values, p″i and q″j are output sample value after the clipping and tcPi tcPi are clipping thresholds that are derived from the VVC tc parameter and tcPD and tcQD. The function Clip3 is a clipping function as it is specified in VVC.
3.6.8 Sub-Block Deblocking Adjustment
[0216] To enable parallel friendly deblocking using both long filters and sub-block deblocking the long filters is restricted to modify at most 5 samples on a side that uses sub-block deblocking (AFFINE or ATMVP or decoder-side motion vector refinement (DMVR)) as shown in the luma control for long filters. Extendedly, the sub-block deblocking is adjusted such that that sub-block boundaries on an 8×8 grid that are close to a CU or an implicit TU boundary is restricted to modify at most two samples on each side.
[0217] The following applies to sub-block boundaries that not are aligned with the CU boundary.
TABLE-US-00007
```
If (mode block Q == SUBBLOCKMODE && edge != 0) {
    if (!(implicitTU && (edge == (64 / 4))))
        if (edge == 2 || edge == (orthogonalLength − 2) || edge == (56 / 4) || edge == (72 / 4))
            Sp = Sq = 2;
        else
            Sp = Sq = 3;
    else
        Sp = Sq = bSideQisLargeBlk ? 5:3
}
```
where edge equal to 0 corresponds to CU boundary, edge equal to 2 or equal to orthogonalLength-2 corresponds to sub-block boundary 8 samples from a CU boundary, etc. Where implicit TU is true if implicit split of TU is used.
3.7 Sample Adaptive Offset
[0218] Sample adaptive offset (SAO) is applied to the reconstructed signal after the deblocking filter by using offsets specified for each CTB by the encoder. The video encoder first makes the decision on whether or not the SAO process is to be applied for current slice. If SAO is applied for the slice, each CTB is classified as one of five SAO types as shown in Table 4. The concept of SAO is to classify pixels into categories and reduces the distortion by adding an offset to pixels of each category. SAO operation includes edge offset (EO) which uses edge properties for pixel classification in SAO type 1 to 4 and band offset (BO) which uses pixel intensity for pixel classification in SAO type 5. Each applicable CTB has SAO parameters including sao_merge_left_flag, sao_merge_up_flag, SAO type and four offsets. If sao_merge_left_flag is equal to 1, the current CTB will reuse the SAO type and offsets of the CTB to the left. If sao_merge_up_flag is equal to 1, the current CTB will reuse SAO type and offsets of the CTB above.
TABLE-US-00008 TABLE 4 Specification of SAO type

SAO type	sample adaptive offset type to be used	Number of categories
0	None	0
1	1-D 0-degree pattern edge offset	4
2	1-D 90-degree pattern edge offset	4
3	1-D 135-degree pattern edge offset	4
4	1-D 45-degree pattern edge offset	4
5	band offset	4

3.8 Adaptive Loop Filter
[0219] Adaptive loop filtering for video coding is to minimize the mean square error between original samples and decoded samples by using Wiener-based adaptive filter. The ALF is located at the last processing stage for each picture and can be regarded as a tool to catch and fix artifacts from previous stages. The suitable filter coefficients are determined by the encoder and explicitly signaled to the decoder. To achieve better coding efficiency, especially for high resolution videos, local adaptation is used for luma signals by applying different filters to different regions or blocks

in a picture. In addition to filter adaptation, filter on/off control at coding tree unit (CTU) level is also helpful for improving coding efficiency. Syntax-wise, filter coefficients are sent in a picture level header called adaptation parameter set, and filter on/off flags of CTUs are interleaved at CTU level in the slice data. This syntax design not only supports picture level optimization but also achieves a low encoding latency.

3.8.1 Signaling of Parameters

[0220] According to ALF design in VTM, filter coefficients and clipping indices are carried in ALF APSs. An ALF APS can include up to 8 chroma filters and one luma filter set with up to 25 filters. An index is also included for each of the 25 luma classes. Classes having the same index share the same filter. By merging different classes, the num of bits required to represent the filter coefficients is reduced. The absolute value of a filter coefficient is represented using a 0th order Exp-Golomb code followed by a sign bit for a non-zero coefficient. When clipping is enabled, a clipping index is also signaled for each filter coefficient using a two-bit fixed-length code. Up to 8 ALF APSs can be used by the decoder at the same time.

[0221] Filter control syntax elements of ALF in VTM include two types of information. First, ALF on/off flags are signaled at sequence, picture, slice and CTB levels. Chroma ALF can be enabled at picture and slice level only if luma ALF is enabled at the corresponding level. Second, filter usage information is signaled at picture, slice and CTB level, if ALF is enabled at that level. Referenced ALF APSs IDs are coded at a slice level or at a picture level if all the slices within the picture use the same APSs. Luma component can reference up to 7 ALF APSs and chroma components can reference 1 ALF APS. For a luma CTB, an index is signalled indicating which ALF APS or offline trained luma filter set is used. For a chroma CTB, the index indicates which filter in the referenced APS is used.

[0222] The data syntax elements of ALF associated to LUMA component in VTM are listed as follows:

TABLE-US-00009 Descriptor alf_data( ) { alf_luma_filter_signal_flag u(1) if(alf_luma_filter_signal_flag) { alf_luma_clip_flag u(1) alf_luma_num_filters_signalled_minus1 ue(v) if(alf_luma_num_filters_signalled_minus1 > 0) for(filtIdx = 0; filtIdx < NumAlfFilters; filtIdx++) alf_luma_coeff_delta_idx[filtIdx] u(v) for(sfIdx = 0; sfIdx <= alf_luma_num_filters_signalled_minus1; sfldx++) for(j = 0; j < 12; j++) { alf_luma_coeff_abs[sfIdx][j] ue(v) if(alf_luma_coeff_abs[sfIdx][j]) alf_luma_coeff_sign[sfIdx][j] u(1) } if(alf_luma_clip_flag) for(sfIdx = 0; sfIdx <= alf_luma_num_filters_signalled_minus1; sfIdx++) for(j = 0; j < 12; j++) alf_luma_clip_idx[sfIdx][j] u(2) }

[0223] alf_luma_filter_signal_flag equal to 1 specifies that a luma filter set is signalled. alf_luma_filter_signal_flag equal to 0 specifies that a luma filter set is not signalled. alf_luma_clip_flag equal to 0 specifies that linear adaptive loop filtering is applied to the luma component. alf_luma_clip_flag equal to 1 specifies that non-linear adaptive loop be applied to the luma component. filtering could alf_luma_num_filters_signalled_minus1 plus 1 specifies the number of adaptive loop filter classes for which luma coefficients can be signalled. The value of alf_luma_num_filters_signalled_minus1 shall be in the range of 0 to NumAlfFilters—1, inclusive. alf_luma_coeff_delta_idx [filtIdx] specifies the indices of the signalled adaptive loop filter luma coefficient deltas for the filter class indicated by filtIdx ranging from 0 to NumAlfFilters—1. When alf_luma_coeff_delta_idx [filtIdx] is not present, it is inferred to be equal to 0. The length of alf_luma_coeff_delta_idx [filtIdx] is Ceil (Log2 (alf_luma_num_filters_signalled_minus1+1)) bits. The value of alf_luma_coeff_delta_idx [filtIdx] shall be in the range of 0 to alf_luma_num_filters_signalled_minus1, inclusive. alf_luma_coeff_abs [sfIdx][j] specifies the absolute value of the j-th coefficient of the signalled luma filter indicated by sfIdx. When alf_luma_coeff_abs [sfIdx][j] is not present, it is inferred to be equal 0. The value of

alf_luma_coeff_abs [sfIdx][j] shall be in the range of 0 to 128, inclusive. alf_luma_coeff_sign [sfIdx][j] specifies the sign of the j-th luma coefficient of the filter indicated by sfIdx as follows: [0224] If alf_luma_coeff_sign [sfIdx][j] is equal to 0, the corresponding luma filter coefficient has a positive value. Otherwise (alf_luma_coeff_sign [sfIdx][j] is equal to 1), the corresponding luma filter coefficient has a negative value.
When alf_luma_coeff_sign [sfIdx][j] is not present, it is inferred to be equal to 0.
[0225] alf_luma_clip_idx [sfIdx][j] specifies the clipping index of the clipping value to use before multiplying by the j-th coefficient of the signalled luma filter indicated by sfIdx. When alf_luma_clip_idx [sfIdx][j] is not present, it is inferred to be equal to 0. The coding tree unit syntax elements of ALF associated to LUMA component in VTM are listed as follows:
TABLE-US-00010 Descriptor coding_tree_unit( ) { xCtb = CtbAddrX << CtbLog2SizeY yCtb = CtbAddrY << CtbLog2SizeY if(sh_alf_enabled_flag){ alf_ctb_flag[0][CtbAddrX][CtbAddrY] ae(v) if(alf_ctb_flag[0][CtbAddrX][CtbAddrY]) { if(sh_num_alf_aps_ids_luma > 0) alf_use_aps_flag ae(v) if(alf_use_aps_flag) { if(sh_num_alf_aps_ids_luma > 1) alf_luma_prev_filter_idx ae(v) } else alf_luma_fixed_filter_idx ae(v) } }
[0226] alf_ctb_flag [cIdx][xCtb>>CtbLog2SizeY][yCtb >>CtbLog2SizeY] equal to 1 specifies that the adaptive loop filter is applied to the coding tree block of the colour component indicated by cIdx of the coding tree unit at luma location (xCtb, yCtb). alf_ctb_flag [cldx] [xCtb>>CtbLog2SizeY][yCtb >>CtbLog2Size Y] equal to 0 specifies that the adaptive loop filter is not applied to the coding tree block of the colour component indicated by cIdx of the coding tree unit at luma location (xCtb, yCtb).
[0227] When alf_ctb_flag [cldx][xCtb>>CtbLog2SizeY][yCtb >>CtbLog2SizeY] is not present, it is inferred to be equal to 0. alf_use_aps_flag equal to 0 specifies that one of the fixed filter sets is applied to the luma CTB. alf_use_aps_flag equal to 1 specifies that a filter set from an APS is applied to the luma CTB. When alf_use_aps_flag is not present, it is inferred to be equal to 0. alf_luma_prev_filter_idx specifies the previous filter that is applied to the luma CTB. The value of alf_luma_prev_filter_idx shall be in a range of 0 to sh_num_alf_aps_ids_luma-1, inclusive. When alf_luma_prev_filter_idx is not present, it is inferred to be equal to 0.
[0228] The variable AlfCtbFiltSetIdxY [xCtb>>CtbLog2SizeY][yCtb >>CtbLog2SizeY] specifying the filter set index for the luma CTB at location (xCtb, yCtb) is derived as follows:
[0229] If alf_use_aps_flag is equal to 0, AlfCtbFiltSetIdxY [xCtb>>CtbLog2SizeY][yCtb >>CtbLog2Size Y] is set equal to alf_luma_fixed_filter_idx. [0230] Otherwise, AlfCtbFiltSetIdxY [xCtb>>CtbLog2SizeY][yCtb >>CtbLog2SizeY] is set equal to 16+alf_luma_prev_filter_idx.
[0231] alf_luma_fixed_filter_idx specifies the fixed filter that is applied to the luma CTB. The value of alf_luma_fixed_filter_idx shall be in a range of 0 to 15, inclusive.
[0232] Based on the ALF design of VTM, the ALF design of ECM further introduces the concept of alternative filter sets into luma filters. The luma filters are be trained multiple alternatives/rounds based on the updated luma CTU ALF on/off decisions of each alternative/rounds. In such way, there will be multiple filter sets that associated to each training alternative and the class merging results of each filter set may be different. Each CTU could select the best filter set by RDO and the related alternative information will be signaled. The data syntax elements of ALF associated to LUMA component in ECM are listed as follows:
TABLE-US-00011 Descriptor alf_data( ) { alf_luma_filter_signal_flag u(1) if(alf_luma_filter_signal_flag) { alf_luma_num_alts_minus1 ue(v) for(altIdx = 0; altIdx < alf_luma_num_alts_minus1 +1; altIdx++){ alf_luma_clip_flag[altIdx] u(1) alf_luma_num_filters_signalled_minus1[altIdx] ue(v) if(alf_luma_num_filters_signalled_minus1[altIdx] > 0){ for(filtIdx = 0; filtIdx < NumAlfFilters; filtIdx++) alf_luma_coeff_delta_idx[altIdx][filtIdx] u(v) } for(sfIdx = 0; sfIdx <= alf_luma_num_filters_signalled_minus1[altIdx]; sfIdx++){

for(j = 0; j < 19; j++){ alf_luma_coeff_abs[altIdx][sfIdx][j] ue(v) if(alf_luma_coeff_abs[altIdx][sfIdx][j]) alf_luma_coeff_sign[altIdx][sfIdx][j] u(1) } } if(alf_luma_clip_flag [altIdx]) for(sfIdx = 0; sfIdx <= alf_luma_num_filters_signalled_minus1[altIdx]; sfIdx++) for(j = 0; j <19; j++) alf_luma_clip_idx[altIdx][sfIdx][j] u(2) } }
[0233] alf_luma_num_alts_minus1 plus 1 specifies the number of alternative filter sets for luma component. The value of alf_luma_num_alts_minus1 shall be in the range of 0 to 3, inclusive. alf_luma_clip_flag [altIdx] equal to 0 specifies that linear adaptive loop filtering is applied to the alternative luma filter set with index altIdxluma component. alf_luma_clip_flag [altIdx] equal to 1 specifies that non-linear adaptive loop filtering could be applied to the alternative luma filter set with index altIdx luma component. alf_luma_num_filters_signalled_minus1 [altIdx] plus 1 specifies the number of adaptive loop filter classes for which luma coefficients can be signalled of the alternative luma filter set with index altIdx. The value of alf_luma_num_filters_signalled_minus1 [altIdx] shall be in the range of 0 to NumAlfFilters—1, inclusive.
[0234] alf_luma_coeff_delta_idx [altIdx][filtIdx] specifies the indices of the signalled adaptive loop filter luma coefficient deltas for the filter class indicated by filtIdx ranging from 0 to NumAlfFilters—1 for the alternative luma filter set with index altIdx. When alf_luma_coeff_delta_idx [filtIdx][altIdx] is not present, it is inferred to be equal to 0. The length of alf_luma_coeff_delta_idx [altIdx][filtIdx] is Ceil (Log2 (alf_luma_num_filters signalled_minus1 [altIdx]+1)) bits. The value of alf_luma_coeff_delta_idx [altIdx][filtIdx] shall be in the range of 0 to alf_luma_num_filters_signalled_minus1 [altIdx], inclusive. alf_luma_coeff_abs [altIdx][sfIdx][j] specifies the absolute value of the j-th coefficient of the signalled luma filter indicated by sfIdx of the alternative luma filter set with index altIdx. When alf_luma_coeff_abs [altIdx][sfIdx][j] is not present, it is inferred to be equal 0. The value of alf_luma_coeff_abs [altIdx][sfIdx][j] shall be in the range of 0 to 128, inclusive.
[0235] alf_luma_coeff_sign [altIdx][sfIdx][j] specifies the sign of the j-th luma coefficient of the filter indicated by sfIdx of the alternative luma filter set with index altIdx as follows: [0236] If alf_luma_coeff_sign [altIdx][sfIdx][j] is equal to 0, the corresponding luma filter coefficient has a positive value. [0237] Otherwise (alf_luma_coeff_sign [altIdx][sfIdx][j] is equal to 1), the corresponding luma filter coefficient has a negative value.
[0238] When alf_luma_coeff_sign [altIdx][sfIdx][j] is not present, it is inferred to be equal to 0. alf_luma_clip_idx [altIdx][sfIdx][j] specifies the clipping index of the clipping value to use before multiplying by the j-th coefficient of the signalled luma filter indicated by sfIdx of the alternative luma filter set with index altIdx. When alf_luma_clip_idx [altIdx][sfIdx][j] is not present, it is inferred to be equal to 0. The coding tree unit syntax elements of ALF associated to LUMA component in ECM are listed as follows:
TABLE-US-00012 Descriptor coding_tree_unit( ) { xCtb = CtbAddrX << CtbLog2SizeY yCtb = CtbAddrY << CtbLog2SizeY if(sh_alf_enabled_flag){ alf_ctb_flag[0][CtbAddrX][CtbAddrY] ae(v) if(alf_ctb_flag[0][CtbAddrX][CtbAddr Y]) { if(sh_num_alf_aps_ids_luma > 0) alf_use_aps_flag ae(v) if(alf_use_aps_flag) { if(sh_num_alf_aps_ids_luma > 1) alt_ctb_luma_filter_alt_idx[CtbAddrX][CtbAddrY] ae(v) alf_luma_prev_filter_idx ae(v) } else alf_luma_fixed_filter_idx ae(v) } }
[0239] alf_ctb_luma_filter_alt_idx [xCtb>>CtbLog2SizeY][yCtb >>CtbLog2SizeY] specifies the index of the alternative luma filters applied to the coding tree block of the luma component, of the coding tree unit at luma location (xCtb, yCtb). When alf_ctb_luma_filter_alt_idx [xCtb>>CtbLog2SizeY][yCtb >>CtbLog2SizeY] is not present, it is inferred to be equal to zero.
3.8.2 Filter Shapes
[0240] FIG. **10** illustrates an example of filter shapes for ALF. In the JEM, up to three diamond

filter shapes (as shown in FIG. **10**) can be selected for the luma component. An index is signalled at the picture level to indicate the filter shape used for the luma component. Each square represents a sample, and Ci (i being 0˜6 (left), 0˜12 (middle), 0˜20 (right)) denotes the coefficient to be applied to the sample. For chroma components in a picture, the 5×5 diamond shape is always used. In VVC, the 7×7 diamond shape is always used for Luma while the 5×5 diamond shape is always used for Chroma.
3.8.3 Classification for ALF
[0241] Each 2×2 (or 4×4) block is categorized into one out of 25 classes. The classification index C is derived based on its directionality D and a quantized value of activity Â, as follows:
[00007]$C = 5D + \hat{A}$.
[0242] To calculate D and Â, gradients of the horizontal, vertical and two diagonal direction are first calculated using 1-D Laplacian:
[00008]$g_v = \sum_{k=i-2}^{i+3} \sum_{l=j-2}^{j+3} V_{k,l}, V_{k,l} = \lvert 2R(k,l) - R(k,l-1) - R(k,l+1) \rvert,$
$g_h = \sum_{k=i-2}^{i+3} \sum_{l=j-2}^{j+3} H_{k,l}, H_{k,l} = \lvert 2R(k,l) - R(k-1,l) - R(k+1,l) \rvert,$
$g_{d1} = \sum_{k=i-2}^{i+3} \sum_{l=j-3}^{j+3} D1_{k,l}, D1_{k,l} = \lvert 2R(k,l) - R(k-1,l-1) - R(k+1,l+1) \rvert$
$g_{d2} = \sum_{k=i-2}^{i+3} \sum_{j=j-2}^{j+3} D2_{k,l}, D2_{k,l} = \lvert 2R(k,l) - R(k-1,l+1) - R(k+1,l-1) \rvert$
[0243] Indices i and j refer to the coordinates of the upper left sample in the 2×2 block and R (i, j) indicates a reconstructed sample at coordinate (i, j). Then D maximum and minimum values of the gradients of horizontal and vertical directions are set as: [0244] g.sub.h,v.sup.max=max(g.sub.h, g.sub.v), g.sub.h,v.sup.min=min (g.sub.h, g.sub.v),
and the maximum and minimum values of the gradient of two diagonal directions are set as: [0245] g.sub.d0,d1.sup.max=max (g.sub.d0, g.sub.d1), g.sub.d0,d1.sup.min=min (g.sub.d0, g.sub.d1),
[0246] To derive the value of the directionality D, these values are compared against each other and with two thresholds t.sub.1 and t.sub.2: [0247] Step 1. If both g.sub.h,v.sup.max≤t.sub.1.Math.g.sub.h,v.sup.min and g.sub.d0,d1.sup.max≤t.sub.1.Math.g.sub.d0,d1.sup.min are true, D is set to 0. [0248] Step 2. If g.sub.h,v.sup.max/g.sub.h,v.sup.min>g.sub.d0,d1.sup.max/g.sub.d0,d1.sup.min, continue from Step 3; otherwise continue from Step 4. [0249] Step 3. If g.sub.h,v.sup.max>t.sub.2.Math.g.sub.h,v.sup.min, D is set to 2; otherwise D is set to 1. [0250] Step 4. If g.sub.d0,d1.sup.max>t.sub.2.Math.g.sub.d0,d1.sup.min, D is set to 4; otherwise D is set to 3.
[0251] The activity value A is calculated as:
[00009]$A = \sum_{k=i-2}^{i+3} \sum_{l=j-2}^{j+3} (V_{k,l} + H_{k,l})$.
[0252] A is further quantized to the range of 0 to 4, inclusively, and the quantized value is denoted as Â. For both chroma components in a picture, no classification method is applied, i.e. a single set of ALF coefficients is applied for each chroma component.
3.8.4 Geometric Transformations of Filter Coefficients
[0253] FIG. **11** illustrates an example of a relative coordinator for the 5×5 diamond filter support. Before filtering each 2×2 block, geometric transformations such as rotation or diagonal and vertical flipping are applied to the filter coefficients f(k, l), which is associated with the coordinate (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.
[0254] Three geometric transformations, including diagonal, vertical flip and rotation are

introduced: [0255] Diagonal: f.sub.D(k, l)=f(l,k), [0256] Vertical flip: f.sub.V(k, l)=f(k,K−l−1), [0257] Rotation: f.sub.R(k,l)=f (K−l−1,k).
where K is the size of the filter and 0≤k, l≤K−1 are coefficients coordinates, such that 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, 1) depending on gradient values calculated for that block. The relationship between the transformation and the four gradients of the four directions are summarized in Table 5. FIG. **11** shows the transformed coefficients for each position based on the 5×5 diamond.
TABLE-US-00013 TABLE 5 Mapping of the gradient calculated for one block and the transformations. Gradient values Transformation g.sub.d2 < g.sub.d1 and g.sub.h < g.sub.v No transformation g.sub.d2 < g.sub.d1 and g.sub.v < g.sub.h Diagonal g.sub.d1 < g.sub.d2 and g.sub.h < g.sub.v Vertical flip g.sub.d1 < g.sub.d2 and g.sub.v < g.sub.h Rotation
3.8.5 Filtering Process
[0258] At decoder side, when ALF is enabled for a block, each sample R(i, j) within the block is filtered, resulting in sample value R′(i, j) as shown below, where L denotes filter length, f.sub.m,n represents filter coefficient, and f(k, l) denotes the decoded filter coefficients.
[00010] $$R'(i,j) = \underset{k=-L/2}{\overset{L/2}{.\text{Math.}}}\ \underset{l=-L/2}{\overset{L/2}{.\text{Math.}}}\ f(k,l) \times R(i+k, j+l)$$
[0259] FIG. **12** shows an example of relative coordinates used for 5×5 diamond filter support supposing the current sample's coordinate (i, j) to be (0, 0). Samples in different coordinates filled with the same color are multiplied with the same filter coefficients.
3.8.6 Non-Linear Filtering Reformulation
[0260] Linear filtering can be reformulated, without coding efficiency impact, in the following expression:
[00011] $$O(x,y) = I(x,y) + \underset{(i,j)\neq(0,0)}{.\text{Math.}}\ w(i,j) . (I(x+i, y+j) - I(x,y))$$
where w(i, j) are the same filter coefficients.
[0261] VVC introduces the non-linearity to make ALF more efficient by using a simple clipping function to reduce the impact of neighbor sample values (I(x+i, y+j)) when they are too different with the current sample value (I(x, y)) being filtered. More specifically, the ALF filter is modified as follows:
[00012] $$O'(x,y) = I(x,y) + \underset{(i,j)\neq(0,0)}{.\text{Math.}}\ w(i,j) . K(I(x+i, y+j) - I(x,y), k(i,j))$$
where K(d, b)=min (b, max (−b,d)) is the clipping function, and k(i, j) are clipping parameters, which depends on the (i, j) filter coefficient. The encoder performs the optimization to find the best k(i, j).
[0262] The clipping parameters k(i, j) are specified for each ALF filter, one clipping value is signaled per filter coefficient. It means that up to 12 clipping values can be signaled in the bitstream per Luma filter and up to 6 clipping values for the Chroma filter. In order to limit the signaling cost and the encoder complexity, only 4 fixed values which are the same for INTER and INTRA slices are used.
[0263] Because the variance of the local differences is often higher for Luma than for Chroma, two different sets for the Luma and Chroma filters are applied. The maximum sample value (here 1024 for 10 bits bit-depth) in each set is also introduced, so that clipping can be disabled if it is not necessary. The 4 values have been selected by roughly equally splitting, in the logarithmic domain, the full range of the sample values (coded on 10 bits) for Luma, and the range from 4 to 1024 for Chroma. More precisely, the Luma table of clipping values have been obtained by the following formula:
[00013] $$\text{AlfClip}_L = \{\text{round}(((M)^{\frac{1}{N}})^{N-n+1})\text{for}\,n \in 1\ .\text{Math.}\ N]\}, \text{with}\,M = 2^{10}\,\text{and}\,N = 4$$
[0264] Similarly, the Chroma tables of clipping values is obtained according to the following

formula:

[00014] $$\text{AlfClip}_C = \{\text{round}(A \,.\, (((\tfrac{M}{A})^{\frac{1}{N+1}})^{N-n})\text{for}\, n \in 1 \text{ .Math. } N]\}, \text{with} M = 2^{10}, N = 4 \text{and} A = 4$$

3.9 Bilateral In-Loop Filter

3.9.1 Bilateral Image Filter

[0265] Bilateral image filter is a nonlinear filter that smooths the noise while preserving edge structures. The bilateral filtering is a technique to make the filter weights decrease not only with the distance between the samples but also with increasing difference in intensity. This way, over-smoothing of edges can be ameliorated. A weight is defined as

[00015] $$w(\quad x, \quad y, \quad I) = e^{-\frac{x^2 + \quad y^2}{2 \quad _d^2} - \frac{I^2}{2 \quad _r^2}}$$

where Δx and Δy is the distance in the vertical and horizontal and ΔI is the difference in intensity between the samples.

[0266] The edge-preserving de-noising bilateral filter adopts a low-pass Gaussian filter for both the domain filter and the range filter. The domain low-pass Gaussian filter gives higher weight to pixels that are spatially close to the center pixel. The range low-pass Gaussian filter gives higher weight to pixels that are similar to the center pixel. Combining the range filter and the domain filter, a bilateral filter at an edge pixel becomes an elongated Gaussian filter that is oriented along the edge and is greatly reduced in gradient direction. This is the reason why the bilateral filter can smooth the noise while preserving edge structures.

3.9.2 Bilateral Filter in Video Coding

[0267] The bilateral filter in video coding is a coding tool for the VVC [2]. The filter acts as a loop filter in parallel with the sample adaptive offset (SAO) filter. Both the bilateral filter and SAO act on the same input samples, each filter produces an offset, and these offsets are then added to the input sample to produce an output sample that, after clipping, goes to the next stage. The spatial filtering strength σ.sub.d is determined by the block size, with smaller blocks filtered more strongly, and the intensity filtering strength σ.sub.r is determined by the quantization parameter, with stronger filtering being used for higher QPs. Only the four closest samples are used, so the filtered sample intensity I.sub.F can be calculated as

[00016] $$I_F = I_C + \frac{w_A \quad I_A + w_B \quad I_B + w_L \quad I_L + w_R \quad I_R}{w_C + w_A + w_B + w_L + w_R}$$

where I.sub.C denotes the intensity of the center sample, ΔI.sub.A=I.sub.A−I.sub.C the intensity difference between the center sample and the sample above. ΔI.sub.B, ΔI.sub.L and ΔI.sub.R denote the intensity difference between the center sample and that of the sample below, to the left and to the right respectively.

4. Technical Problems Solved by Disclosed Technical Solutions

[0268] Example designs for adaptive loop filters (ALF) in video coding have the following problems.

[0269] In an example ALF design, previously signalled filters can only be reused without any modification. However, different pictures and/or slices may use different coding settings and have different characteristics. The performance of ALF may be further improved by using signalled filters with scaling/modification instead of using them directly.

5. A Listing of Solutions and Embodiments

[0270] To solve the above-described problems, methods as summarized below are disclosed. The embodiments should be considered as examples to explain the general concepts and should not be interpreted in a narrow way. Furthermore, these embodiments can be applied individually or combined in any manner.

[0271] It should be noted that the disclosed methods may be used as in-loop filters or post-processing.

[0272] In this disclosure, a video unit may refer to a sequence, a picture, a sub-picture, a slice, a CTU, a block, and/or a region. The video unit may comprise one color component or multiple color components.

[0273] In this disclosure, an ALF and/or cross component ALF (CCALF) processing unit may refer to a sequence, a picture, a sub-picture, a slice, a CTU, a block, a region, or a sample. The ALF and/or CCALF processing unit may comprise one color component or multiple color components.
Example 1
[0274] In an example, the signalled filters of ALF can be reused after modification for a current ALF processing unit.
Example 2
[0275] In one example, one or more coefficients of a signalled filter of ALF may be scaled by a weight w (e.g., w=0.5 or 1.5). In one example, all the coefficients of a signalled filter may be scaled by an identical weight. In one example, the weight may be signalled to the decoder for each ALF processing unit. In one example, the weight may be pre-defined at the encoder and decoder. In one example, the weight index may be signalled for each ALF processing unit. In one example, the weight may be derived on-the-fly for each ALF processing unit.
Example 3
[0276] In one example, two different coefficients of a signalled filter may be scaled by two different weights. In one example, the weights may be signalled to the decoder for each ALF processing unit. In one example, the weights may be pre-defined at the encoder and decoder. In one example, the weight indexes may be signalled for each ALF processing unit. In one example, the weights may be derived on-the-fly for each ALF processing unit.
Example 4
[0277] In one example, one or more coefficients of a signalled filter of ALF may be modified by adding an offset k (e.g., k=10 or −10). In one example, all the coefficients of a signalled filter may be modified by an identical offset. In one example, the offset may be signalled to the decoder for each ALF processing unit. In one example, the offset may be pre-defined at the encoder and decoder. In one example, the offset index may be signalled for each ALF processing unit. In one example, the offset may be derived on-the-fly for each ALF processing unit.
Example 5
[0278] In one example, two different coefficients of a signalled filter may be modified by two different offsets. In one example, the offsets may be signalled to the decoder for each ALF processing unit. In one example, the offsets may be pre-defined at the encoder and decoder. In one example, the offset indexes may be signalled for each ALF processing unit. In one example, the offsets may be derived on-the-fly for each ALF processing unit.
Example 6
[0279] In one example, one or more clipping parameters of a signalled filter of ALF may be modified by adding an offset t (e.g., t=+1 or −1). In one example, all the clipping parameters may be modified by an identical offset. In one example, the offset may be signalled to the decoder side for each ALF processing unit. In one example, the offset may be pre-defined at the encoder and decoder. In one example, the offset index may be signalled for each ALF processing unit. In one example, the offset may be derived on-the-fly for each ALF processing unit.
Example 7
[0280] In one example, two different clipping parameters may be modified by two different offsets. In one example, the offsets may be signalled to the decoder for each ALF processing unit. In one example, the offsets may be pre-defined at the encoder and decoder. In one example, the offset indexes may be signalled for each ALF processing unit. In one example, the offsets may be derived on-the-fly for each ALF processing unit.
Example 8
[0281] In one example, the output of a signalled filter may be scaled by a weight. In one example, the weight may be signaled to the decoder side for each ALF processing unit. In one example, the weight may be pre-defined at the encoder and decoder. In one example, two different weight indices may be signalled for two different ALF processing units. In one example, the weight may

be derived on-the-fly for each ALF processing unit.
Example 9
[0282] In one example, the output of a signalled filter may be modified by adding an offset. In one example, the offset may be signalled to the decoder side for each ALF processing unit. In one example, the offset may be pre-defined at the encoder and decoder. In one example, the offset index may be signalled for each ALF processing unit. In one example, the offset may be derived on-the-fly for each ALF processing unit.
Example 10
[0283] In one example, the signalled filter modification method may be applied to a LUMA component.
Example 11
[0284] In one example, the signalled filter modification method may be applied to a CHROMA component.
Example 12
[0285] In one example, the signalled filters of CCALF can be reused after modification for a current CCALF processing unit.
Example 13
[0286] In one example, the signalled filters of CCALF can be reused after modification.
[0287] In one example, one or more coefficients of a signalled filter of CCALF may be scaled by a weight w (e.g., w=0.5 or 1.5). In one example, all the coefficients of a signalled filter may be scaled by an identical weight. In one example, the weight may be signalled to the decoder for each CCALF processing unit. In one example, the weight may be pre-defined at the encoder and decoder. In one example, the weight index may be signalled for each CCALF processing unit. In one example, the weight may be derived on-the-fly for each CCALF processing unit.
[0288] In one example, two different coefficients of a signalled filter may be scaled by two different weights. In one example, the weights may be signalled to the decoder for each CCALF processing unit. In one example, the weights may be pre-defined at the encoder and decoder. In one example, the weight indexes may be signalled for each CCALF processing unit. In one example, the weights may be derived on-the-fly for each CCALF processing unit.
Example 14
[0289] In one example, one or more coefficients of a signalled filter of CCALF may be modified by adding an offset k (e.g., k=10 or −10). In one example, all the coefficients of a signalled filter may be modified by an identical offset. In one example, the offset may be signalled to the decoder for each CCALF processing unit. In one example, the offset may be pre-defined at the encoder and decoder. In one example, the offset index may be signalled for each CCALF processing unit. In one example, the offset may be derived on-the-fly for each CCALF processing unit.
[0290] In one example, two different coefficients of a signalled filter may be modified by two offsets. In one example, the offsets may be signalled to the decoder for each CCALF processing unit. In one example, the offsets may be pre-defined at the encoder and decoder. In one example, the offset indexes may be signalled for each CCALF processing unit. In one example, the offsets may be derived on-the-fly for each CCALF processing unit.
Example 15
[0291] In one example, one or more clipping parameters of a signalled filter of CCALF may be modified by adding an offset t (e.g., t=+1 or −1). In one example, all the clipping parameters may be modified by an identical offset. In one example, the offset may be signalled to the decoder side for each CCALF processing unit. In one example, the offset may be pre-defined at the encoder and decoder. In one example, the offset index may be signalled for each CCALF processing unit. In one example, the offset may be derived on-the-fly for each CCALF processing unit.
[0292] In one example, two different clipping parameters may be modified by two different offsets. In one example, the offsets may be signalled to the decoder for each CCALF processing unit. In

one example, the offsets may be pre-defined at the encoder and decoder. In one example, the offset indexes may be signalled for each CCALF processing unit. In one example, the offsets may be derived on-the-fly for each CCALF processing unit.
Example 16
[0293] In one example, the output of a signalled filter may be scaled by a weight. In one example, the weight may be signaled to the decoder side for each CCALF processing unit. In one example, the weight may be pre-defined at the encoder and decoder. In one example, the weight index may be signalled for each CCALF processing unit. In one example, the weight may be derived on-the-fly for each CCALF processing unit.
Example 17
[0294] In one example, the output of a signalled filter may be modified by adding an offset. In one example, the offset may be signalled to the decoder side for each CCALF processing unit. In one example, the offset may be pre-defined at the encoder and decoder. In one example, the offset index may be signalled for each CCALF processing unit. In one example, the offset may be derived on-the-fly for each CCALF processing unit.
Example 18
[0295] In one example, the signalled filter modification method may be applied to blue difference chroma (Cb) component. In one example, the signalled filter modification method may be applied to red difference chroma (Cr) component.
Example 19
[0296] In one example, the disclosed methods may be used in post-processing and/or pre-processing.
Example 20
[0297] In one example, the above-mentioned methods may be used jointly.
Example 21
[0298] In one example, the above-mentioned methods may be used individually.
Example 22
[0299] A first value disclosed in this disclosure may be converted to a second value before being signaled, which can be converted back after parting. The conversion may be quantization and/or dequantization. The conversion may be transform/inverse transform. The conversion may be jointly used.
Example 23
[0300] Any information disclosed in the disclosure may be signaled as one or multiple Syntax Element (SE). The SE may be predictively coded. The SE may be coded with at least one context. The SE may be bypass coded.
Example 24
[0301] In one example, the described signalled filter modification method may be applied to any in-loop filtering tools, pre-processing, or post-processing filtering method in video coding (including but not limited to ALF/CCALF or any other filtering method).
[0302] In one example, the example signalled filter modification method may be applied to an in-loop filtering method. In one example, the example signalled filter modification method may be applied to ALF. In one example, the example signalled filter modification method may be applied to CCALF. In one example, the example signalled filter modification method may be applied to SAO. In one example, the example signalled filter modification method may be applied to other in-loop filtering methods.
[0303] In one example, the example signalled filter modification method may be applied to a pre-processing filtering method. In one example, the example signalled filter modification method may be applied to a post-processing filtering method.
Example 25
[0304] In above examples, the video unit may refer to sequence/picture/sub-

picture/slice/tile/coding tree unit (CTU)/CTU row/groups of CTU/coding unit (CU)/prediction unit (PU)/transform unit (TU)/coding tree block (CTB)/coding block (CB)/prediction block (PB)/transform block (TB)/any other region that contains more than one luma or chroma sample/pixel.

Example 26

[0305] Whether to and/or how to apply the disclosed methods above may be signaled in a bitstream.

[0306] In one example, they may be signaled at sequence level/group of pictures level/picture level/slice level/tile group level, such as in a sequence header, picture header, SPS, VPS, DPS, DCI, PPS, APS, slice header, and tile group header.

[0307] In one example, they may be signaled at PB, TB, CB, PU, TU, CU, VPDU, CTU, CTU row, slice, tile, sub-picture, other kinds of region contain more than one sample or pixel.

Example 27

[0308] Whether to and/or how to apply the disclosed methods above may be dependent on coded information, such as block size, color format, single/dual tree partitioning, color component, slice/picture type.

6. References

[0309] [1] J. Strom, P. Wennersten, J. Enhorn, D. Liu, K. Andersson and R. Sjoberg, “Bilateral Loop Filter in Combination with SAO,” in proceeding of IEEE Picture Coding Symposium (PCS), November 2019.

[0310] FIG. **13** is a block diagram showing an example video processing system **4000** in which various techniques disclosed herein may be implemented. Various implementations may include some or all of the components of the system **4000**. The system **4000** may include input **4002** for receiving video content. The video content may be received in a raw or uncompressed format, e.g., 8 or 10 bit multi-component pixel values, or may be in a compressed or encoded format. The input **4002** may represent a network interface, a peripheral bus interface, or a storage interface. Examples of network interface include wired interfaces such as Ethernet, passive optical network (PON), etc. and wireless interfaces such as wireless fidelity (Wi-Fi) or cellular interfaces.

[0311] The system **4000** may include a coding component **4004** that may implement the various coding or encoding methods described in the present disclosure. The coding component **4004** may reduce the average bitrate of video from the input **4002** to the output of the coding component **4004** to produce a coded representation of the video. The coding techniques are therefore sometimes called video compression or video transcoding techniques. The output of the coding component **4004** may be either stored, or transmitted via a communication connected, as represented by the component **4006**. The stored or communicated bitstream (or coded) representation of the video received at the input **4002** may be used by a component **4008** for generating pixel values or displayable video that is sent to a display interface **4010**. The process of generating user-viewable video from the bitstream representation is sometimes called video decompression. Furthermore, while certain video processing operations are referred to as “coding” operations or tools, it will be appreciated that the coding tools or operations are used at an encoder and corresponding decoding tools or operations that reverse the results of the coding will be performed by a decoder.

[0312] Examples of a peripheral bus interface or a display interface may include universal serial bus (USB) or high definition multimedia interface (HDMI) or Displayport, and so on. Examples of storage interfaces include serial advanced technology attachment (SATA), peripheral component interconnect (PCI), integrated drive electronics (IDE) interface, and the like. The techniques described in the present disclosure may be embodied in various electronic devices such as mobile phones, laptops, smartphones or other devices that are capable of performing digital data processing and/or video display.

[0313] FIG. **14** is a block diagram of an example video processing apparatus **4100**. The apparatus **4100** may be used to implement one or more of the methods described herein. The apparatus **4100**

may be embodied in a smartphone, tablet, computer, Internet of Things (IoT) receiver, and so on. The apparatus **4100** may include one or more processors **4102**, one or more memories **4104** and video processing circuitry **4106**. The processor(s) **4102** may be configured to implement one or more methods described in the present disclosure. The memory (memories) **4104** may be used for storing data and code used for implementing the methods and techniques described herein. The video processing circuitry **4106** may be used to implement, in hardware circuitry, some techniques described in the present disclosure. In some embodiments, the video processing circuitry **4106** may be at least partly included in the processor **4102**, e.g., a graphics co-processor.
[0314] FIG. **15** is a flowchart for an example method **4200** of video processing. The method **4200** includes determining to reuse filters of an ALF with modification after use for a current ALF processing unit at step **4202**. A conversion is performed between a visual media data and a bitstream based on the ALF at step **4204**. The conversion of step **4204** may include encoding at an encoder or decoding at a decoder, depending on the example.
[0315] It should be noted that the method **4200** can be implemented in an apparatus for processing video data comprising a processor and a non-transitory memory with instructions thereon, such as video encoder **4400**, video decoder **4500**, and/or encoder **4600**. In such a case, the instructions upon execution by the processor, cause the processor to perform the method **4200**. Further, the method **4200** can be performed by a non-transitory computer readable medium comprising a computer program product for use by a video coding device. The computer program product comprises computer executable instructions stored on the non-transitory computer readable medium such that when executed by a processor cause the video coding device to perform the method **4200**.
[0316] FIG. **16** is a block diagram that illustrates an example video coding system **4300** that may utilize the techniques of this disclosure. The video coding system **4300** may include a source device **4310** and a destination device **4320**. Source device **4310** generates encoded video data which may be referred to as a video encoding device. Destination device **4320** may decode the encoded video data generated by source device **4310** which may be referred to as a video decoding device.
[0317] Source device **4310** may include a video source **4312**, a video encoder **4314**, and an input/output (I/O) interface **4316**. Video source **4312** may include a source such as a video capture device, an interface to receive video data from a video content provider, and/or a computer graphics system for generating video data, or a combination of such sources. The video data may comprise one or more pictures. Video encoder **4314** encodes the video data from video source **4312** to generate a bitstream. The bitstream may include a sequence of bits that form a coded representation of the video data. The bitstream may include coded pictures and associated data. The coded picture is a coded representation of a picture. The associated data may include sequence parameter sets, picture parameter sets, and other syntax structures. I/O interface **4316** may include a modulator/demodulator (modem) and/or a transmitter. The encoded video data may be transmitted directly to destination device **4320** via I/O interface **4316** through network **4330**. The encoded video data may also be stored onto a storage medium/server **4340** for access by destination device **4320**.
[0318] Destination device **4320** may include an I/O interface **4326**, a video decoder **4324**, and a display device **4322**. I/O interface **4326** may include a receiver and/or a modem. I/O interface **4326** may acquire encoded video data from the source device **4310** or the storage medium/server **4340**. Video decoder **4324** may decode the encoded video data. Display device **4322** may display the decoded video data to a user. Display device **4322** may be integrated with the destination device **4320**, or may be external to destination device **4320**, which can be configured to interface with an external display device.
[0319] Video encoder **4314** and video decoder **4324** may operate according to a video compression standard, such as the High Efficiency Video Coding (HEVC) standard, Versatile Video Coding (VVC) standard and other current and/or further standards.

[0320] FIG. **17** is a block diagram illustrating an example of video encoder **4400**, which may be video encoder **4314** in the system **4300** illustrated in FIG. **16**. Video encoder **4400** may be configured to perform any or all of the techniques of this disclosure. The video encoder **4400** includes a plurality of functional components. The techniques described in this disclosure may be shared among the various components of video encoder **4400**. In some examples, a processor may be configured to perform any or all of the techniques described in this disclosure.
[0321] The functional components of video encoder **4400** may include a partition unit **4401**, a prediction unit **4402** which may include a mode select unit **4403**, a motion estimation unit **4404**, a motion compensation unit **4405**, an intra prediction unit **4406**, a residual generation unit **4407**, a transform processing unit **4408**, a quantization unit **4409**, an inverse quantization unit **4410**, an inverse transform unit **4411**, a reconstruction unit **4412**, a buffer **4413**, and an entropy encoding unit **4414**.
[0322] In other examples, video encoder **4400** may include more, fewer, or different functional components. In an example, prediction unit **4402** may include an intra block copy (IBC) unit. The IBC unit may perform prediction in an IBC mode in which at least one reference picture is a picture where the current video block is located.
[0323] Furthermore, some components, such as motion estimation unit **4404** and motion compensation unit **4405** may be highly integrated, but are represented in the example of video encoder **4400** separately for purposes of explanation.
[0324] Partition unit **4401** may partition a picture into one or more video blocks. Video encoder **4400** and video decoder **4500** may support various video block sizes.
[0325] Mode select unit **4403** may select one of the coding modes, intra or inter, e.g., based on error results, and provide the resulting intra or inter coded block to a residual generation unit **4407** to generate residual block data and to a reconstruction unit **4412** to reconstruct the encoded block for use as a reference picture. In some examples, mode select unit **4403** may select a combination of intra and inter prediction (CIIP) mode in which the prediction is based on an inter prediction signal and an intra prediction signal. Mode select unit **4403** may also select a resolution for a motion vector (e.g., a sub-pixel or integer pixel precision) for the block in the case of inter prediction.
[0326] To perform inter prediction on a current video block, motion estimation unit **4404** may generate motion information for the current video block by comparing one or more reference frames from buffer **4413** to the current video block. Motion compensation unit **4405** may determine a predicted video block for the current video block based on the motion information and decoded samples of pictures from buffer **4413** other than the picture associated with the current video block.
[0327] Motion estimation unit **4404** and motion compensation unit **4405** may perform different operations for a current video block, for example, depending on whether the current video block is in an I slice, a P slice, or a B slice.
[0328] In some examples, motion estimation unit **4404** may perform uni-directional prediction for the current video block, and motion estimation unit **4404** may search reference pictures of list 0 or list 1 for a reference video block for the current video block. Motion estimation unit **4404** may then generate a reference index that indicates the reference picture in list 0 or list 1 that contains the reference video block and a motion vector that indicates a spatial displacement between the current video block and the reference video block. Motion estimation unit **4404** may output the reference index, a prediction direction indicator, and the motion vector as the motion information of the current video block. Motion compensation unit **4405** may generate the predicted video block of the current block based on the reference video block indicated by the motion information of the current video block.
[0329] In other examples, motion estimation unit **4404** may perform bi-directional prediction for the current video block, motion estimation unit **4404** may search the reference pictures in list 0 for a reference video block for the current video block and may also search the reference pictures in

list 1 for another reference video block for the current video block. Motion estimation unit **4404** may then generate reference indexes that indicate the reference pictures in list 0 and list 1 containing the reference video blocks and motion vectors that indicate spatial displacements between the reference video blocks and the current video block. Motion estimation unit **4404** may output the reference indexes and the motion vectors of the current video block as the motion information of the current video block. Motion compensation unit **4405** may generate the predicted video block of the current video block based on the reference video blocks indicated by the motion information of the current video block.
[0330] In some examples, motion estimation unit **4404** may output a full set of motion information for decoding processing of a decoder. In some examples, motion estimation unit **4404** may not output a full set of motion information for the current video. Rather, motion estimation unit **4404** may signal the motion information of the current video block with reference to the motion information of another video block. For example, motion estimation unit **4404** may determine that the motion information of the current video block is sufficiently similar to the motion information of a neighboring video block.
[0331] In one example, motion estimation unit **4404** may indicate, in a syntax structure associated with the current video block, a value that indicates to the video decoder **4500** that the current video block has the same motion information as another video block.
[0332] In another example, motion estimation unit **4404** may identify, in a syntax structure associated with the current video block, another video block and a motion vector difference (MVD). The motion vector difference indicates a difference between the motion vector of the current video block and the motion vector of the indicated video block. The video decoder **4500** may use the motion vector of the indicated video block and the motion vector difference to determine the motion vector of the current video block.
[0333] As discussed above, video encoder **4400** may predictively signal the motion vector. Two examples of predictive signaling techniques that may be implemented by video encoder **4400** include advanced motion vector prediction (AMVP) and merge mode signaling.
[0334] Intra prediction unit **4406** may perform intra prediction on the current video block. When intra prediction unit **4406** performs intra prediction on the current video block, intra prediction unit **4406** may generate prediction data for the current video block based on decoded samples of other video blocks in the same picture. The prediction data for the current video block may include a predicted video block and various syntax elements.
[0335] Residual generation unit **4407** may generate residual data for the current video block by subtracting the predicted video block(s) of the current video block from the current video block. The residual data of the current video block may include residual video blocks that correspond to different sample components of the samples in the current video block.
[0336] In other examples, there may be no residual data for the current video block for the current video block, for example in a skip mode, and residual generation unit **4407** may not perform the subtracting operation.
[0337] Transform processing unit **4408** may generate one or more transform coefficient video blocks for the current video block by applying one or more transforms to a residual video block associated with the current video block.
[0338] After transform processing unit **4408** generates a transform coefficient video block associated with the current video block, quantization unit **4409** may quantize the transform coefficient video block associated with the current video block based on one or more quantization parameter (QP) values associated with the current video block.
[0339] Inverse quantization unit **4410** and inverse transform unit **4411** may apply inverse quantization and inverse transforms to the transform coefficient video block, respectively, to reconstruct a residual video block from the transform coefficient video block. Reconstruction unit **4412** may add the reconstructed residual video block to corresponding samples from one or more

predicted video blocks generated by the prediction unit **4402** to produce a reconstructed video block associated with the current block for storage in the buffer **4413**.
[0340] After reconstruction unit **4412** reconstructs the video block, the loop filtering operation may be performed to reduce video blocking artifacts in the video block.
[0341] Entropy encoding unit **4414** may receive data from other functional components of the video encoder **4400**. When entropy encoding unit **4414** receives the data, entropy encoding unit **4414** may perform one or more entropy encoding operations to generate entropy encoded data and output a bitstream that includes the entropy encoded data.
[0342] FIG. **18** is a block diagram illustrating an example of video decoder **4500** which may be video decoder **4324** in the system **4300** illustrated in FIG. **16**. The video decoder **4500** may be configured to perform any or all of the techniques of this disclosure. In the example shown, the video decoder **4500** includes a plurality of functional components. The techniques described in this disclosure may be shared among the various components of the video decoder **4500**. In some examples, a processor may be configured to perform any or all of the techniques described in this disclosure.
[0343] In the example shown, video decoder **4500** includes an entropy decoding unit **4501**, a motion compensation unit **4502**, an intra prediction unit **4503**, an inverse quantization unit **4504**, an inverse transformation unit **4505**, a reconstruction unit **4506**, and a buffer **4507**. Video decoder **4500** may, in some examples, perform a decoding pass generally reciprocal to the encoding pass described with respect to video encoder **4400**.
[0344] Entropy decoding unit **4501** may retrieve an encoded bitstream. The encoded bitstream may include entropy coded video data (e.g., encoded blocks of video data). Entropy decoding unit **4501** may decode the entropy coded video data, and from the entropy decoded video data, motion compensation unit **4502** may determine motion information including motion vectors, motion vector precision, reference picture list indexes, and other motion information. Motion compensation unit **4502** may, for example, determine such information by performing the AMVP and merge mode.
[0345] Motion compensation unit **4502** may produce motion compensated blocks, possibly performing interpolation based on interpolation filters. Identifiers for interpolation filters to be used with sub-pixel precision may be included in the syntax elements.
[0346] Motion compensation unit **4502** may use interpolation filters as used by video encoder **4400** during encoding of the video block to calculate interpolated values for sub-integer pixels of a reference block. Motion compensation unit **4502** may determine the interpolation filters used by video encoder **4400** according to received syntax information and use the interpolation filters to produce predictive blocks.
[0347] Motion compensation unit **4502** may use some of the syntax information to determine sizes of blocks used to encode frame(s) and/or slice(s) of the encoded video sequence, partition information that describes how each macroblock of a picture of the encoded video sequence is partitioned, modes indicating how each partition is encoded, one or more reference frames (and reference frame lists) for each inter coded block, and other information to decode the encoded video sequence.
[0348] Intra prediction unit **4503** may use intra prediction modes for example received in the bitstream to form a prediction block from spatially adjacent blocks. Inverse quantization unit **4504** inverse quantizes, i.e., de-quantizes, the quantized video block coefficients provided in the bitstream and decoded by entropy decoding unit **4501**. Inverse transform unit **4505** applies an inverse transform.
[0349] Reconstruction unit **4506** may sum the residual blocks with the corresponding prediction blocks generated by motion compensation unit **4502** or intra prediction unit **4503** to form decoded blocks. If desired, a deblocking filter may also be applied to filter the decoded blocks in order to remove blockiness artifacts. The decoded video blocks are then stored in buffer **4507**, which

provides reference blocks for subsequent motion compensation/intra prediction and also produces decoded video for presentation on a display device.
[0350] FIG. **19** is a schematic diagram of an example encoder **4600**. The encoder **4600** is suitable for implementing the techniques of VVC. The encoder **4600** includes three in-loop filters, namely a deblocking filter (DF) **4602**, a sample adaptive offset (SAO) **4604**, and an adaptive loop filter (ALF) **4606**. Unlike the DF **4602**, which uses predefined filters, the SAO **4604** and the ALF **4606** utilize the original samples of the current picture to reduce the mean square errors between the original samples and the reconstructed samples by adding an offset and by applying a finite impulse response (FIR) filter, respectively, with coded side information signaling the offsets and filter coefficients. The ALF **4606** is located at the last processing stage of each picture and can be regarded as a tool trying to catch and fix artifacts created by the previous stages.
[0351] The encoder **4600** further includes an intra prediction component **4608** and a motion estimation/compensation (ME/MC) component **4610** configured to receive input video. The intra prediction component **4608** is configured to perform intra prediction, while the ME/MC component **4610** is configured to utilize reference pictures obtained from a reference picture buffer **4612** to perform inter prediction. Residual blocks from inter prediction or intra prediction are fed into a transform (T) component **4614** and a quantization (Q) component **4616** to generate quantized residual transform coefficients, which are fed into an entropy coding component **4618**. The entropy coding component **4618** entropy codes the prediction results and the quantized transform coefficients and transmits the same toward a video decoder (not shown). Quantization components output from the quantization component **4616** may be fed into an inverse quantization (IQ) components **4620**, an inverse transform component **4622**, and a reconstruction (REC) component **4624**. The REC component **4624** is able to output images to the DF **4602**, the SAO **4604**, and the ALF **4606** for filtering prior to those images being stored in the reference picture buffer **4612**.
[0352] FIG. **20** is a method for processing video data in accordance with an embodiment of the disclosure. In block **2002**, the method includes reusing filters of an adaptive loop filter (ALF) for a current ALF processing unit after modification of the filters. In block **2004**, the method includes performing a conversion between a visual media data and a bitstream based on the ALF.
[0353] A listing of solutions preferred by some examples is provided next.
[0354] The following solutions show examples of techniques discussed herein. [0355] 1. A method for processing video data comprising: determining to reuse filters of an adaptive loop filter (ALF) with modification after use for a current ALF processing unit; and performing a conversion between a visual media data and a bitstream based on the ALF. [0356] 2. The method of solution 1, wherein all coefficients of a filter are scaled by an identical weight, and wherein the weight is signaled, is predefined, is signaled by a weight index, is derived, or combinations thereof. [0357] 3. The method of any of solutions 1-2, wherein two different coefficients of a filter are scaled by two different weights, and wherein the weights are signaled, are predefined, are signaled by weight indices, are derived, or combinations thereof. [0358] 4. The method of any of solutions 1-3, wherein one or more coefficients of a filter of the ALF are modified by adding an offset k. [0359] 5. The method of any of solutions 1-4, wherein all the coefficients of a filter are modified by an identical offset, and wherein the offset is signaled, is predefined, is signaled by a weight index, is derived, or combinations thereof. [0360] 6. The method of any of solutions 1-5, wherein two different coefficients of a filter are modified by two different offsets, and wherein the offsets are signaled, are predefined, are signaled by offset indices, are derived, or combinations thereof. [0361] 7. The method of any of solutions 1-6, wherein one or more clipping parameters of a filter of the ALF are modified by adding an offset t. [0362] 8. The method of any of solutions 1-7, wherein all clipping parameters are modified by an identical offset, and wherein the offset is signaled, is predefined, is signaled as an offset index for each ALF processing unit, is derived, or combinations thereof. [0363] 9. The method of any of solutions 1-8, wherein two different clipping parameters are modified by two different offsets, and wherein the offsets are signaled, are predefined, are

signaled as offset indices for each ALF processing unit, are derived, or combinations thereof. [0364] 10. The method of any of solutions 1-9, wherein an output of a filter is scaled by a weight, and wherein the weight is signaled, is predefined, is signaled by one or more weight indices, is derived, or combinations thereof. [0365] 11. The method of any of solutions 1-10, wherein an output of a filter is modified by adding an offset, and wherein the offset is signaled, is predefined, is signaled by one or more offset indices, is derived, or combinations thereof. [0366] 12. The method of any of solutions 1-11, wherein a filter modification is applied to a luma component or a chroma component. [0367] 13. The method of any of solutions 1-12, wherein the ALF is a cross component ALF (CCALF). [0368] 14. The method of any of solutions 1-13, wherein a filter modification is applied to a blue difference chroma (Cb) component or a red difference chroma (Cr) component. [0369] 15. The method of any of solutions 1-14, wherein the method is used in pre-processing or post-processing of a video. [0370] 16. The method of any of solutions 1-15, wherein a first value is converted to a second value according to quantization, dequantization, transform, inverse transform, or combinations thereof. [0371] 17. The method of any of solutions 1-16, wherein the bitstream includes one or more syntax elements that are predictively coded, context coded, bypass coded, or combinations thereof. [0372] 18. The method of any of solutions 1-17, wherein the method is applied as part of an in-loop filter, and wherein the method is applied to an ALF, a CCALF, a sample adaptive offset (SAO) filter, or combinations thereof. [0373] 19. The method of any of solutions 1-18, wherein the processing unit is a sequence, picture, sub-picture, slice, tile, coding tree unit (CTU), CTU row, group of CTUs, coding unit (CU), prediction unit (PU), transform unit (TU), coding tree block (CTB), coding block (CB), prediction block (PB), transform block (TB), or any other region that contains more than one luma or chroma sample or pixel. [0374] 20. The method of any of solutions 1-19, wherein application of the method is signaled in the bitstream. [0375] 21. The method of any of solutions 1-20, wherein application of the method is dependent on coded information. [0376] 22. An apparatus for processing video data comprising: a processor; and a non-transitory memory with instructions thereon, wherein the instructions upon execution by the processor, cause the processor to perform the method of any of solutions 1-21. [0377] 23. A non-transitory computer readable medium comprising a computer program product for use by a video coding device, the computer program product comprising computer executable instructions stored on the non-transitory computer readable medium such that when executed by a processor cause the video coding device to perform the method of any of solutions 1-21. [0378] 24. A non-transitory computer-readable recording medium storing a bitstream of a video which is generated by a method performed by a video processing apparatus, wherein the method comprises: determining to reuse filters of an adaptive loop filter (ALF) with modification after use for a current ALF processing unit; and generating the bitstream based on the determining. [0379] 25. A method for storing bitstream of a video comprising: determining to reuse filters of an adaptive loop filter (ALF) with modification after use for a current ALF processing unit; generating the bitstream based on the determining; and storing the bitstream in a non-transitory computer-readable recording medium. [0380] 26. A method, apparatus, or system described in the present disclosure.

[0381] In the solutions described herein, an encoder may conform to the format rule by producing a coded representation according to the format rule. In the solutions described herein, a decoder may use the format rule to parse syntax elements in the coded representation with the knowledge of presence and absence of syntax elements according to the format rule to produce decoded video.

[0382] In the present disclosure, the term “video processing” may refer to video encoding, video decoding, video compression or video decompression. For example, video compression algorithms may be applied during conversion from pixel representation of a video to a corresponding bitstream representation or vice versa. The bitstream representation of a current video block may, for example, correspond to bits that are either co-located or spread in different places within the bitstream, as is defined by the syntax. For example, a macroblock may be encoded in terms of transformed and coded error residual values and also using bits in headers and other fields in the bitstream. Furthermore, during conversion, a decoder may parse a bitstream with the knowledge that some fields may be present, or absent, based on the determination, as is described in the above solutions. Similarly, an encoder may determine that certain syntax fields are or are not to be included and generate the coded representation accordingly by including or excluding the syntax fields from the coded representation.

[0383] The disclosed and other solutions, examples, embodiments, modules and the functional operations described in this disclosure can be implemented in digital electronic circuitry, or in computer software, firmware, or hardware, including the structures disclosed in this disclosure and their structural equivalents, or in combinations of one or more of them. The disclosed and other embodiments can be implemented as one or more computer program products, i.e., one or more modules of computer program instructions encoded on a computer readable medium for execution by, or to control the operation of, data processing apparatus. The computer readable medium can be a machine-readable storage device, a machine-readable storage substrate, a memory device, a composition of matter effecting a machine-readable propagated signal, or a combination of one or more them. The term “data processing apparatus” encompasses all apparatus, devices, and machines for processing data, including by way of example a programmable processor, a computer, or multiple processors or computers. The apparatus can include, in addition to hardware, code that creates an execution environment for the computer program in question, e.g., code that constitutes processor firmware, a protocol stack, a database management system, an operating system, or a combination of one or more of them. A propagated signal is an artificially generated signal, e.g., a machine-generated electrical, optical, or electromagnetic signal, that is generated to encode information for transmission to suitable receiver apparatus.

[0384] A computer program (also known as a program, software, software application, script, or code) can be written in any form of programming language, including compiled or interpreted languages, and it can be deployed in any form, including as a stand-alone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A computer program does not necessarily correspond to a file in a file system. A program can be stored in a portion of a file that holds other programs or data (e.g., one or more scripts stored in a markup language document), in a single file dedicated to the program in question, or in multiple coordinated files (e.g., files that store one or more modules, sub programs, or portions of code). A computer program can be deployed to be executed on one computer or on multiple computers that are located at one site or distributed across multiple sites and interconnected by a communication network.

[0385] The processes and logic flows described in this disclosure can be performed by one or more programmable processors executing one or more computer programs to perform functions by operating on input data and generating output. The processes and logic flows can also be performed by, and apparatus can also be implemented as, special purpose logic circuitry, e.g., a field programmable gate array (FPGA) or an application specific integrated circuit (ASIC).

[0386] Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and data from a read only memory or a random-access memory or both. The essential elements of a computer are a processor for performing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto optical disks, or optical disks. However, a computer need not have such devices. Computer readable media suitable for storing computer program instructions and data include all forms of non-volatile memory, media and memory devices, including by way of example semiconductor memory devices, e.g., erasable programmable read-only memory (EPROM), electrically erasable

programmable read-only memory (EEPROM), and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto optical disks; and compact disc read-only memory (CD ROM) and Digital versatile disc-read only memory (DVD-ROM) disks. The processor and the memory can be supplemented by, or incorporated in, special purpose logic circuitry.
[0387] While the present disclosure contains many specifics, these should not be construed as limitations on the scope of any subject matter or of what may be claimed, but rather as descriptions of features that may be specific to particular embodiments of particular techniques. Certain features that are described in the present disclosure in the context of separate embodiments can also be implemented in combination in a single embodiment. Conversely, various features that are described in the context of a single embodiment can also be implemented in multiple embodiments separately or in any suitable subcombination. Moreover, although features may be described above as acting in certain combinations and even initially claimed as such, one or more features from a claimed combination can in some cases be excised from the combination, and the claimed combination may be directed to a subcombination or variation of a subcombination.
[0388] Similarly, while operations are depicted in the drawings in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. Moreover, the separation of various system components in the embodiments described in the present disclosure should not be understood as requiring such separation in all embodiments.
[0389] Only a few implementations and examples are described and other implementations, enhancements and variations can be made based on what is described and illustrated in the present disclosure.
[0390] A first component is directly coupled to a second component when there are no intervening components, except for a line, a trace, or another medium between the first component and the second component. The first component is indirectly coupled to the second component when there are intervening components other than a line, a trace, or another medium between the first component and the second component. The term “coupled” and its variants include both directly coupled and indirectly coupled. The use of the term “about” means a range including ±10% of the subsequent number unless otherwise stated.
[0391] While several embodiments have been provided in the present disclosure, it should be understood that the disclosed systems and methods might be embodied in many other specific forms without departing from the spirit or scope of the present disclosure. The present examples are to be considered as illustrative and not restrictive, and the intention is not to be limited to the details given herein. For example, the various elements or components may be combined or integrated in another system or certain features may be omitted, or not implemented.
[0392] In addition, techniques, systems, subsystems, and methods described and illustrated in the various embodiments as discrete or separate may be combined or integrated with other systems, modules, techniques, or methods without departing from the scope of the present disclosure. Other items shown or discussed as coupled may be directly connected or may be indirectly coupled or communicating through some interface, device, or intermediate component whether electrically, mechanically, or otherwise. Other examples of changes, substitutions, and alterations are ascertainable by one skilled in the art and could be made without departing from the spirit and scope disclosed herein.

Claims

1. A method for processing video data, comprising: reusing, during a conversion between a video comprising a current ALF processing unit and a bitstream of the video, filters of an adaptive loop filter (ALF), which are signaled in the bitstream, for the current ALF processing unit after modification of the filters; and performing the conversion based on the ALF.

2. The method of claim 1, further comprising scaling one or more coefficients of a first filter of the filters by a weight (w).
3. The method of claim 2, wherein the weight w is 0.5 or 1.5.
4. The method of claim 1, further comprising scaling all coefficients of a first filter of the filters by a weight (w), wherein the weight is identical for all of the one or more coefficients.
5. The method of claim 1, further comprising adding an offset (k) to one or more coefficients of a first filter of the filters.
6. The method of claim 5, wherein the offset k is 10 or −10.
7. The method of claim 1, further comprising adding an offset (k) to all coefficients of a first filter of the filters, wherein the offset is identical for all of the coefficients.
8. The method of claim 1, further comprising adding an offset (t) to one or more clipping parameters of a first filter of the filters.
9. The method of claim 8, wherein the offset t is +1 or −1.
10. The method of claim 1, further comprising adding an offset (t) to all clipping parameters of a first filter of the filters, wherein the offset is identical for all of the clipping parameters.
11. The method of claim 1, further comprising scaling an output of a first filter of the filters by a weight.
12. The method of claim 1, wherein the ALF comprises a cross component ALF (CCALF) and the current ALF processing unit comprises a CCALF processing unit.
13. The method of claim 12, further comprising adding an offset to an output of a first filter of the CCALF.
14. The method of claim 1, wherein information for the method is included in the bitstream as one or more syntax elements (SEs).
15. The method of claim 1, wherein the conversion includes encoding the video into the bitstream.
16. The method of claim 1, wherein the conversion includes decoding the video from the bitstream.
17. An apparatus for processing media data comprising: one or more processors; and a non-transitory memory with instructions thereon, wherein the instructions upon execution by the one or more processors cause the apparatus to: reuse, during a conversion between a video comprising a current ALF processing unit and a bitstream of the video, filters of an adaptive loop filter (ALF), which are signaled in the bitstream, for the current ALF processing unit after modification of the filters; and perform the conversion based on the ALF.
18. The apparatus of claim 17, wherein the one or more processors are further caused to: scale one or more coefficients of a first filter of the filters by a weight (w), wherein the weight w is 0.5 or 1.5; or scale all coefficients of a first filter of the filters by a weight (w), wherein the weight is identical for all of the one or more coefficients; add an offset (k) to one or more coefficients of a first filter of the filters, wherein the offset k is 10 or **−10**; or add an offset (k) to all coefficients of a first filter of the filters, wherein the offset is identical for all of the coefficients; add an offset (t) to one or more clipping parameters of a first filter of the filters, wherein the offset t is +1 or **−1**; or add an offset (t) to all clipping parameters of a first filter of the filters, wherein the offset is identical for all of the clipping parameters; scale an output of a first filter of the filters by a weight, wherein the ALF comprises a cross component ALF (CCALF) and the current ALF processing unit comprises a CCALF processing unit, and wherein the one or more processors are further caused to add an offset to an output of a first filter of the CCALF, and wherein information for reuse of the filters is included in the bitstream as one or more syntax elements (SEs).
19. A non-transitory computer readable storage medium storing instructions that cause a processor to: reuse, during a conversion between a video comprising a current ALF processing unit and a bitstream of the video, filters of an adaptive loop filter (ALF), which are signaled in the bitstream, for the current ALF processing unit after modification of the filters; and perform the conversion based on the ALF
20. A non-transitory computer-readable recording medium storing a bitstream of a video which is

generated by a method performed by a video processing apparatus, wherein the method comprises: reusing filters of an adaptive loop filter (ALF), which are signaled in the bitstream, for a current ALF processing unit of the video after modification of the filters; and generating the bitstream based on the ALF.