# Indication Of One Slice Per SubPicture In Subpicture-Based Video Coding

## CROSS-REFERENCE TO RELATED APPLICATIONS

**[0001]** This patent application claims the benefit of U.S. Provisional Patent Application No. 62/904,481 filed September 23, 2019, by Ye-Kui Wang., et al., and titled "Indication of One Slice Per Subpicture in Subpicture-Based Video Coding," and U.S. Provisional Patent Application No. 62/905,122 filed September 24, 2019 by Ye-Kui Wang. et al., and titled "Indication of One Slice Per Subpicture in Subpicture-Based Video Coding," which are hereby incorporated by reference.

## TECHNICAL FIELD

**[0002]** The present disclosure is generally related to video coding, and is specifically related to signaling an indication in subpicture-based video coding.

## BACKGROUND

**[0003]** The amount of video data needed to depict even a relatively short video can be substantial, which may result in difficulties when the data is to be streamed or otherwise communicated across a communications network with limited bandwidth capacity. Thus, video data is generally compressed before being communicated across modern day telecommunications networks. The size of a video could also be an issue when the video is stored on a storage device because memory resources may be limited. Video compression devices often use software and/or hardware at the source to code the video data prior to transmission or storage, thereby decreasing the quantity of data needed to represent digital video images. The compressed data is then received at the destination by a video decompression device that decodes the video data. With limited network resources and ever increasing demands of higher video quality, improved compression and decompression techniques that improve compression ratio with little to no sacrifice in image quality are desirable.

## SUMMARY

**[0004]** A first aspect relates to a method implemented by a decoder, comprising receiving, by the decoder, a bitstream including a picture parameter set (PPS) including a single slice per subpicture flag; determining, by the decoder, whether the single slice per subpicture flag has a first value or a second value, wherein the first value specifies that each subpicture referring to the PPS includes one and only one rectangular slice, and wherein the second value specifies that each subpicture referring to the PPS may include one or more rectangular slices; and decoding, by the

decoder, the one and only one rectangular slice when the single slice per subpicture flag has the first value and the one or more rectangular slices when the single slice per subpicture flag has the second value to obtain a decoded picture.

**[0005]** The method provides techniques that use a flag in the bitstream to indicate whether each subpicture of a picture contains one, and only one, rectangular slice. When the flag indicates that each subpicture contains only a single rectangular slice, the syntax elements for signaling the number and layout of subpictures (e.g., the position and size of each subpicture) can be omitted from the bitstream because the subpictures are the same as the slices. That is, only the syntax elements for the slices need to be included in the bitstream when the flag indicates each subpicture contains only a single rectangular slice. This reduces the amount of information carried in the bitstream, which decreases memory, processor, and network resource usage and increases coding efficiency. Thus, the coder / decoder (a.k.a., "codec") in video coding is improved relative to current codecs. As a practical matter, the improved video coding process offers the user a better user experience when videos are sent, received, and/or viewed.

**[0006]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that the single slice per subpicture flag is designated as pps_single_slice_per_subpic_flag.

**[0007]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that the first value is one (1) and the second value is zero (0).

**[0008]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that the PPS further comprises a number of slices in picture flag, and wherein a value of the number of slices in picture flag plus one specifies a number of slices in each picture referring to the PPS.

**[0009]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that the number of slices in picture flag is designated as pps_num_slices_in_pic_minus1.

**[0010]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that each subpicture comprises a rectangular region of one or more slices within a picture.

**[0011]** Optionally, in any of the preceding aspects, another implementation of the aspect provides generating an image for display based on the one and only one rectangular slice when the single slice per subpicture flag has the first value and the one or more rectangular slices when the single slice per subpicture flag has the second value.

**[0012]** A second aspect relates to a method implemented by an encoder, the method comprising setting, by the encoder, a single slice per subpicture flag to a first value when each subpicture referring to a picture parameter set (PPS) includes one and only one rectangular slice and to a second value when each subpicture referring to the PPS may include one or more rectangular

slices; encoding, by the encoder, the PPS containing the single slice per subpicture flag into a bitstream; and storing, by the encoder, the bitstream for communication toward a decoder.

**[0013]** The method provides techniques that use a flag in the bitstream to indicate whether each subpicture of a picture contains one, and only one, rectangular slice. When the flag indicates that each subpicture contains only a single rectangular slice, the syntax elements for signaling the number and layout of subpictures (e.g., the position and size of each subpicture) can be omitted from the bitstream because the subpictures are the same as the slices. That is, only the syntax elements for the slices need to be included in the bitstream when the flag indicates each subpicture contains only a single rectangular slice. This reduces the amount of information carried in the bitstream, which decreases memory, processor, and network resource usage and increases coding efficiency. Thus, the coder / decoder (a.k.a., "codec") in video coding is improved relative to current codecs. As a practical matter, the improved video coding process offers the user a better user experience when videos are sent, received, and/or viewed.

**[0014]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that the single slice per subpicture flag is designated as pps_single_slice_per_subpic_flag.

**[0015]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that the first value is one (1) and the second value is zero (0).

**[0016]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that the PPS further comprises a number of slices in picture flag, and wherein a value of the number of slices in picture flag plus one specifies a number of slices in each picture referring to the PPS.

**[0017]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that the number of slices in picture flag is designated as pps_num_slices_in_pic_minus1.

**[0018]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that each subpicture comprises a rectangular region of one or more slices within a picture.

**[0019]** Optionally, in any of the preceding aspects, another implementation of the aspect provides transmitting the bitstream toward a video decoder.

**[0020]** A third aspect relates to a decoding device, comprising a receiver configured to receive a bitstream including a picture parameter set (PPS) containing a single slice per subpicture flag; a memory coupled to the receiver, the memory storing instructions; and a processor coupled to the memory, the processor configured to execute the instructions to cause the decoding device to determine whether the single slice per subpicture flag has a first value or a second value, wherein the first value specifies that each subpicture referring to the PPS includes one and only one rectangular slice, and wherein the second value specifies that each subpicture referring to the PPS

may include one or more rectangular slices; and decode the one and only one rectangular slice when the single slice per subpicture flag has the first value and the one or more rectangular slices when the single slice per subpicture flag has the second value to obtain a decoded picture.

**[0021]** The decoding device provides techniques that use a flag in the bitstream to indicate whether each subpicture of a picture contains one, and only one, rectangular slice. When the flag indicates that each subpicture contains only a single rectangular slice, the syntax elements for signaling the number and layout of subpictures (e.g., the position and size of each subpicture) can be omitted from the bitstream because the subpictures are the same as the slices. That is, only the syntax elements for the slices need to be included in the bitstream when the flag indicates each subpicture contains only a single rectangular slice. This reduces the amount of information carried in the bitstream, which decreases memory, processor, and network resource usage and increases coding efficiency. Thus, the coder / decoder (a.k.a., "codec") in video coding is improved relative to current codecs. As a practical matter, the improved video coding process offers the user a better user experience when videos are sent, received, and/or viewed.

**[0022]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that the first value is one (1) and the second value is zero (0).

**[0023]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that the PPS further comprises a number of slices in picture flag, and wherein a value of the number of slices in picture flag plus one specifies a number of slices in each picture referring to the PPS.

**[0024]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that the number of slices in picture flag is designated as pps_num_slices_in_pic_minus1.

**[0025]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that the processor is further configured to generate an image for display based on the one and only one rectangular slice when the single slice per subpicture flag has the first value and the one or more rectangular slices when the single slice per subpicture flag has the second value.

**[0026]** A fourth aspect relates to an encoding device, comprising a memory containing instructions; a processor coupled to the memory, the processor configured to implement the instructions to cause the encoding device to encode a bitstream including a picture parameter set (PPS) containing a single slice per subpicture flag; set the single slice per subpicture flag to a first value when each subpicture referring to the PPS includes one and only one rectangular slice and to a second value when each subpicture referring to the PPS may include one or more rectangular slices; and a transmitter coupled to the processor, the transmitter configured to transmit the bitstream toward a video decoder.

**[0027]** The encoding device provides techniques that use a flag in the bitstream to indicate whether each subpicture of a picture contains one, and only one, rectangular slice. When the flag indicates that each subpicture contains only a single rectangular slice, the syntax elements for signaling the number and layout of subpictures (e.g., the position and size of each subpicture) can be omitted from the bitstream because the subpictures are the same as the slices. That is, only the syntax elements for the slices need to be included in the bitstream when the flag indicates each subpicture contains only a single rectangular slice. This reduces the amount of information carried in the bitstream, which decreases memory, processor, and network resource usage and increases coding efficiency. Thus, the coder / decoder (a.k.a., "codec") in video coding is improved relative to current codecs. As a practical matter, the improved video coding process offers the user a better user experience when videos are sent, received, and/or viewed.

**[0028]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that the first value is one (1) and the second value is zero (0).

**[0029]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that the PPS further comprises a number of slices in picture flag, and wherein a value of the number of slices in picture flag plus one specifies a number of slices in each picture referring to the PPS.

**[0030]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that the number of slices in picture flag is designated as pps_num_slices_in_pic_minus1.

**[0031]** Optionally, in any of the preceding aspects, another implementation of the aspect provides storing the bitstream in the memory prior to transmitting the bitstream toward the decoder.

**[0032]** A fifth aspect relates to a coding apparatus. The coding apparatus includes a receiver configured to receive a picture to encode or to receive a bitstream to decode; a transmitter coupled to the receiver, the transmitter configured to transmit the bitstream to a decoder or to transmit a decoded image to a display; a memory coupled to at least one of the receiver or the transmitter, the memory configured to store instructions; and a processor coupled to the memory, the processor configured to execute the instructions stored in the memory to perform any of the methods disclosed herein.

**[0033]** The coding apparatus provides techniques that use a flag in the bitstream to indicate whether each subpicture of a picture contains one, and only one, rectangular slice. When the flag indicates that each subpicture contains only a single rectangular slice, the syntax elements for signaling the number and layout of subpictures (e.g., the position and size of each subpicture) can be omitted from the bitstream because the subpictures are the same as the slices. That is, only the syntax elements for the slices need to be included in the bitstream when the flag indicates each subpicture

contains only a single rectangular slice. This reduces the amount of information carried in the bitstream, which decreases memory, processor, and network resource usage and increases coding efficiency. Thus, the coder / decoder (a.k.a., "codec") in video coding is improved relative to current codecs. As a practical matter, the improved video coding process offers the user a better user experience when videos are sent, received, and/or viewed.

**[0034]** Optionally, in any of the preceding aspects, another implementation of the aspect provides that a display configured to display a decoded picture.

**[0035]** A sixth aspect relates to a system. The system includes an encoder; and a decoder in communication with the encoder, wherein the encoder or the decoder includes the decoding device, the encoding device, or the coding apparatus disclosed herein.

**[0036]** The system provides techniques that use a flag in the bitstream to indicate whether each subpicture of a picture contains one, and only one, rectangular slice. When the flag indicates that each subpicture contains only a single rectangular slice, the syntax elements for signaling the number and layout of subpictures (e.g., the position and size of each subpicture) can be omitted from the bitstream because the subpictures are the same as the slices. That is, only the syntax elements for the slices need to be included in the bitstream when the flag indicates each subpicture contains only a single rectangular slice. This reduces the amount of information carried in the bitstream, which decreases memory, processor, and network resource usage and increases coding efficiency. Thus, the coder / decoder (a.k.a., "codec") in video coding is improved relative to current codecs. As a practical matter, the improved video coding process offers the user a better user experience when videos are sent, received, and/or viewed.

**[0037]** A seventh aspect relates to a means for coding. The means for coding includes receiving means configured to receive a picture to encode or to receive a bitstream to decode; transmission means coupled to the receiving means, the transmission means configured to transmit the bitstream to a decoding means or to transmit a decoded image to a display means; storage means coupled to at least one of the receiving means or the transmission means, the storage means configured to store instructions; and processing means coupled to the storage means, the processing means configured to execute the instructions stored in the storage means to perform any of the methods disclosed herein.

**[0038]** The means for coding provides techniques that use a flag in the bitstream to indicate whether each subpicture of a picture contains one, and only one, rectangular slice. When the flag indicates that each subpicture contains only a single rectangular slice, the syntax elements for signaling the number and layout of subpictures (e.g., the position and size of each subpicture) can be omitted from the bitstream because the subpictures are the same as the slices. That is, only the syntax elements for the slices need to be included in the bitstream when the flag indicates each subpicture

contains only a single rectangular slice. This reduces the amount of information carried in the bitstream, which decreases memory, processor, and network resource usage and increases coding efficiency. Thus, the coder / decoder (a.k.a., "codec") in video coding is improved relative to current codecs. As a practical matter, the improved video coding process offers the user a better user experience when videos are sent, received, and/or viewed.

**[0039]** 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.

**[0040]** These and other features will be more clearly understood from the following detailed description taken in conjunction with the accompanying drawings and claims.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0041]** 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.

**[0042]** FIG. 1 is a flowchart of an example method of coding a video signal.

**[0043]** FIG. 2 is a schematic diagram of an example coding and decoding (codec) system for video coding.

**[0044]** FIG. 3 is a schematic diagram illustrating an example video encoder.

**[0045]** FIG. 4 is a schematic diagram illustrating an example video decoder.

**[0046]** FIG. 5 is a schematic diagram illustrating an example bitstream containing an encoded video sequence.

**[0047]** FIG. 6 is an embodiment of a method of decoding a coded video bitstream.

**[0048]** FIG. 7 is an embodiment of a method of encoding a coded video bitstream.

**[0049]** FIG. 8 is a schematic diagram of a video coding device.

**[0050]** FIG. 9 is a schematic diagram of an embodiment of a means for coding.

## DETAILED DESCRIPTION

**[0051]** 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 in existence. 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.

**[0052]** The following terms are defined as follows unless used in a contrary context herein. Specifically, the following definitions are intended to provide additional clarity to the present disclosure. However, terms may be described differently in different contexts. Accordingly, the following definitions should be considered as a supplement and should not be considered to limit any other definitions of descriptions provided for such terms herein.

**[0053]** A bitstream is a sequence of bits including video data that is compressed for transmission between an encoder and a decoder. An encoder is a device that is configured to employ encoding processes to compress video data into a bitstream. A decoder is a device that is configured to employ decoding processes to reconstruct video data from a bitstream for display. A picture is an array of luma samples and/or an array of chroma samples that create a frame or a field thereof. A picture that is being encoded or decoded can be referred to as a current picture for clarity of discussion. A reference picture is a picture that contains reference samples that can be used when coding other pictures by reference according to inter-prediction and/or inter-layer prediction. A reference picture list is a list of reference pictures used for inter-prediction and/or inter-layer prediction. Some video coding systems utilize two reference picture lists, which can be denoted as reference picture list one and reference picture list zero. A reference picture list structure is an addressable syntax structure that contains multiple reference picture lists. Inter-prediction is a mechanism of coding samples of a current picture by reference to indicated samples in a reference picture that is different from the current picture where the reference picture and the current picture are in the same layer. A reference picture list structure entry is an addressable location in a reference picture list structure that indicates a reference picture associated with a reference picture list. A slice header is a part of a coded slice containing data elements pertaining to all video data within a tile represented in the slice. A slice includes four sides that are connected at right angles. The four sides include two pair of parallel sides. Further, the sides in a parallel side pair are of equal length. As such, a slice may be any rectangular shape, where a square is a special case of a rectangle where all four sides are of equal length. A rectangular slice is a slice having two pairs of parallel sides, where the sides in each parallel side are of equal length but where one pair of the parallel sides is longer than the other pair of parallel sides. A sequence parameter set (SPS) is a parameter set that contains data related to a sequence of pictures. A picture parameter set (PPS) is a syntax structure containing syntax elements that apply to zero or more entire coded pictures as determined by a syntax element found in each picture header.

**[0054]** A flag is a variable or single-bit syntax element that can take one of the two possible values: 0 and 1. A subpicture is a rectangular region of one or more slices within a picture. A

subpicture identifier (ID) is a number, letter, or other indicia that uniquely identifies a subpicture. The subpicture IDs (a.k.a., tile identifiers) are used to identify particular subpictures using a subpicture index, which may be referred to herein as subpicture ID mapping.

**[0055]** An access unit (AU) is a set of one or more coded pictures associated with the same display time (e.g., the same picture order count) for output from a decoded picture buffer (DPB) (e.g., for display to a user). An access unit delimiter (AUD) is an indicator or data structure used to indicate the start of an AU or the boundary between AUs. A decoded video sequence is a sequence of pictures that have been reconstructed by a decoder in preparation for display to a user.

**[0056]** A coded video sequence (CVS) is a sequence of access units (AUs) that include, in decoding order, a coded video sequence start (CVSS) AU, followed by zero or more AUs that are not CVSS AUs, including all subsequent AUs up to but not including any subsequent AU that is a CVSS AU. A CVSS AU is an AU in which there is a prediction unit (PU) for each layer specified by the video parameter set (VPS) and the coded picture in each PU is a coded layer video sequence start (CLVSS) picture. In an embodiment, each picture is within an AU. A PU is a set of network abstraction layer (NAL) units that are associated with each other according to a specified classification rule, are consecutive in decoding order, and contain exactly one coded picture.

**[0057]** The following acronyms are used herein: Adaptive Loop Filter (ALF), Coding Tree Block (CTB), Coding Tree Unit (CTU), Coding Unit (CU), Coded Video Sequence (CVS), Joint Video Experts Team (JVET), Motion-Constrained Tile Set (MCTS), Maximum Transfer Unit (MTU), Network Abstraction Layer (NAL), Picture Order Count (POC), Picture Parameter Set (PPS), Raw Byte Sequence Payload (RBSP), Sample Adaptive Offset (SAO), Sequence Parameter Set (SPS), Temporal Motion Vector Prediction (TMVP), Versatile Video Coding (VVC), and Working Draft (WD).

**[0058]** Many video compression techniques can be employed to reduce the size of video files with minimal loss of data. For example, video compression techniques can include performing spatial (e.g., intra-picture) prediction and/or temporal (e.g., inter-picture) prediction to reduce or remove data redundancy in video sequences. For block-based video coding, a video slice (e.g., a video picture or a portion of a video picture) may be partitioned into video blocks, which may also be referred to as treeblocks, coding tree blocks (CTBs), coding tree units (CTUs), coding units (CUs), and/or coding nodes. Video blocks in an intra-coded (I) slice of a picture are coded using spatial prediction with respect to reference samples in neighboring blocks in the same picture. Video blocks in an inter-coded unidirectional prediction (P) or bidirectional prediction (B) slice of a picture may be coded by employing spatial prediction with respect to reference samples in neighboring blocks in the same picture or temporal prediction with respect to reference samples in other reference pictures. Pictures may be referred to as frames and/or images, and reference pictures may be referred to as

reference frames and/or reference images. Spatial or temporal prediction results in a predictive block representing an image block. Residual data represents pixel differences between the original image block and the predictive block. Accordingly, an inter-coded block is encoded according to a motion vector that points to a block of reference samples forming the predictive block and the residual data indicating the difference between the coded block and the predictive block. An intra-coded block is encoded according to an intra-coding mode and the residual data. For further compression, the residual data may be transformed from the pixel domain to a transform domain. These result in residual transform coefficients, which may be quantized. The quantized transform coefficients may initially be arranged in a two-dimensional array. The quantized transform coefficients may be scanned in order to produce a one-dimensional vector of transform coefficients. Entropy coding may be applied to achieve even more compression. Such video compression techniques are discussed in greater detail below.

**[0059]** To ensure an encoded video can be accurately decoded, video is encoded and decoded according to corresponding video coding standards. Video coding standards include International Telecommunication Union (ITU) Standardization Sector (ITU-T) H.261, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) Motion Picture Experts Group (MPEG)-1 Part 2, ITU-T H.262 or ISO/IEC MPEG-2 Part 2, ITU-T H.263, ISO/IEC MPEG-4 Part 2, Advanced Video Coding (AVC), also known as ITU-T H.264 or ISO/IEC MPEG-4 Part 10, and High Efficiency Video Coding (HEVC), also known as ITU-T H.265 or MPEG-H Part 2. AVC includes extensions such as Scalable Video Coding (SVC), Multiview Video Coding (MVC) and Multiview Video Coding plus Depth (MVC+D), and three dimensional (3D) AVC (3D-AVC). HEVC includes extensions such as Scalable HEVC (SHVC), Multiview HEVC (MV-HEVC), and 3D HEVC (3D-HEVC).

**[0060]** There is also a new video coding standard, named Versatile Video Coding (VVC), being developed by the joint video experts team (JVET) of ITU-T and ISO/IEC. While the VVC standard has several working drafts, one Working Draft (WD) of VVC in particular, namely B. Bross, J. Chen, and S. Liu, "Versatile Video Coding (Draft 5)," JVET-N1001-v3, 13th JVET Meeting, March 27, 2019 (VVC Draft 5) is referenced herein.

**[0061]** HEVC includes four different picture partitioning schemes, namely regular slices, dependent slices, tiles, and Wavefront Parallel Processing (WPP), which may be applied for Maximum Transfer Unit (MTU) size matching, parallel processing, and reduced end-to-end delay.

**[0062]** Regular slices are similar as in H.264/AVC. Each regular slice is encapsulated in its own NAL unit, and in-picture prediction (intra sample prediction, motion information prediction, coding mode prediction) and entropy coding dependency across slice boundaries are disabled. Thus a

regular slice can be reconstructed independently from other regular slices within the same picture (though there may still be interdependencies due to loop filtering operations).

**[0063]** The regular slice is the only tool that can be used for parallelization that is also available, in virtually identical form, in H.264/AVC. Regular slice based parallelization does not require much inter-processor or inter-core communication (except for inter-processor or inter-core data sharing for motion compensation when decoding a predictively coded picture, which is typically much heavier than inter-processor or inter-core data sharing due to in-picture prediction). However, for the same reason, the use of regular slices can incur substantial coding overhead due to the bit cost of the slice header and due to the lack of prediction across the slice boundaries. Further, regular slices (in contrast to the other tools mentioned below) also serve as the key mechanism for bitstream partitioning to match MTU size requirements, due to the in-picture independence of regular slices and that each regular slice is encapsulated in its own NAL unit. In many cases, the goal of parallelization and the goal of MTU size matching place contradicting demands to the slice layout in a picture. The realization of this situation led to the development of the parallelization tools mentioned below.

**[0064]** Dependent slices have short slice headers and allow partitioning of the bitstream at treeblock boundaries without breaking any in-picture prediction. Basically, dependent slices provide fragmentation of regular slices into multiple NAL units to provide reduced end-to-end delay by allowing a part of a regular slice to be sent out before the encoding of the entire regular slice is finished.

**[0065]** In WPP, the picture is partitioned into single rows of coding tree blocks (CTBs). Entropy decoding and prediction are allowed to use data from CTBs in other partitions. Parallel processing is possible through parallel decoding of CTB rows, where the start of the decoding of a CTB row is delayed by two CTBs, so to ensure that data related to a CTB above and to the right of the subject CTB is available before the subject CTB is being decoded. Using this staggered start (which appears like a wavefront when represented graphically), parallelization is possible with up to as many processors/cores as the picture contains CTB rows. Because in-picture prediction between neighboring treeblock rows within a picture is permitted, the required inter-processor/inter-core communication to enable in-picture prediction can be substantial. The WPP partitioning does not result in the production of additional NAL units compared to when it is not applied, thus WPP is not a tool for MTU size matching. However, if MTU size matching is needed, regular slices can be used with WPP, with certain coding overhead.

**[0066]** Tiles define horizontal and vertical boundaries that partition a picture into tile columns and rows. The scan order of CTBs is changed to be local within a tile (in the order of a CTB raster scan of a tile), before decoding the top-left CTB of the next tile in the order of tile raster scan of a picture. Similar to regular slices, tiles break in-picture prediction dependencies as well as entropy

decoding dependencies. However, they do not need to be included in individual NAL units (same as WPP in this regard); hence tiles cannot be used for MTU size matching. Each tile can be processed by one processor/core, and the inter-processor/inter-core communication required for in-picture prediction between processing units decoding neighboring tiles is limited to conveying the shared slice header in cases where a slice is spanning more than one tile, and loop filtering related sharing of reconstructed samples and metadata. When more than one tile or WPP segment is included in a slice, the entry point byte offset for each tile or WPP segment other than the first one in the slice is signaled in the slice header.

**[0067]** For simplicity, restrictions on the application of the four different picture partitioning schemes have been specified in HEVC. A given coded video sequence cannot include both tiles and wavefronts for most of the profiles specified in HEVC. For each slice and tile, either or both of the following conditions must be fulfilled: 1) all coded treeblocks in a slice belong to the same tile; 2) all coded treeblocks in a tile belong to the same slice. Finally, a wavefront segment contains exactly one CTB row, and when WPP is in use, if a slice starts within a CTB row, it must end in the same CTB row.

**[0068]** A recent amendment to HEVC is specified in the JCT-VC output document JCTVC-AC1005, J. Boyce, A. Ramasubramonian, R. Skupin, G. J. Sullivan, A. Tourapis, Y.-K. Wang (editors), "HEVC Additional Supplemental Enhancement Information (Draft 4)," Oct. 24, 2017, publically available at: http://phenix.int-evry.fr/jct/doc_end_user/documents/29_Macau/wg11/JCTVC-AC1005-v2.zip. With this amendment included, HEVC specifies three MCTS-related supplemental enhancement information (SEI) messages, namely temporal MCTSs SEI message, MCTSs extraction information set SEI message, and MCTSs extraction information nesting SEI message.

**[0069]** The temporal MCTSs SEI message indicates existence of MCTSs in the bitstream and signals the MCTSs. For each MCTS, motion vectors are restricted to point to full-sample locations inside the MCTS and to fractional-sample locations that require only full-sample locations inside the MCTS for interpolation, and the usage of motion vector candidates for temporal motion vector prediction derived from blocks outside the MCTS is disallowed. This way, each MCTS may be independently decoded without the existence of tiles not included in the MCTS.

**[0070]** The MCTSs extraction information sets SEI message provides supplemental information that can be used in the MCTS sub-bitstream extraction (specified as part of the semantics of the SEI message) to generate a conforming bitstream for an MCTS set. The information consists of a number of extraction information sets, each defining a number of MCTS sets and containing RBSP bytes of the replacement video parameter sets (VPSs), SPSs, and PPSs to be used during the MCTS sub-bitstream extraction process. When extracting a sub-bitstream according to the MCTS sub-bitstream

extraction process, parameter sets (VPSs, SPSs, and PPSs) need to be rewritten or replaced, slice headers need to be slightly updated because one or all of the slice address related syntax elements (including first_slice_segment_in_pic_flag and slice_segment_address) typically would need to have different values.

**[0071]** In the latest VVC draft specification, a picture may be partitioned into multiple subpictures, each covering a rectangular region and containing an integer number of complete slices. The subpicture partitioning persists across all pictures within a coded video sequence (CVS), and the partitioning information (i.e., the subpicture positions and sizes) is signaled in the SPS. A subpicture may be indicated as being coded without using sample values from any other subpicture for motion compensation.

**[0072]** In JVET contribution JVET-O0141, which is publicly available at: http://phenix.it-sudparis.eu/jvet/doc_end_user/documents/13_Marrakech/wg11/JVET-M0261-v1.zip, the subpicture design is similar as in the latest VVC draft specification with some differences, one of which is the explicit signaling of a subpicture ID for each subpicture in the SPS as well as the signaling of the subpicture ID in each slice header to enable subpicture-based sub-bitstream extraction without the need to change coded slice NAL units. In this approach, the subpicture ID for each subpicture keeps unchanged across all pictures in a CVS.

**[0073]** In a common use case, each subpicture contains only one slice. However, the ability to indicate the common use case is lacking or missing, which is unfortunate. Indeed, knowing that each subpicture contains only one slice could prove useful. For example, when the layout of slices in pictures is known, the bits for signaling the subpicture layout could be completely omitted from the bitstream as the layout of the subpictures would be exactly the same as the layout of the slices.

**[0074]** Disclosed herein are techniques that use a flag in the bitstream to indicate whether each subpicture of a picture contains one, and only one, rectangular slice. When the flag indicates that each subpicture contains only a single rectangular slice, the syntax elements for signaling the number and layout of subpictures (e.g., the position and size of each subpicture) can be omitted from the bitstream because the subpictures are the same as the slices. That is, only the syntax elements for the slices need to be included in the bitstream when the flag indicates each subpicture contains only a single rectangular slice. This reduces the amount of information carried in the bitstream, which decreases memory, processor, and network resource usage and increases coding efficiency. Thus, the coder / decoder (a.k.a., “codec”) in video coding is improved relative to current codecs. As a practical matter, the improved video coding process offers the user a better user experience when videos are sent, received, and/or viewed.

**[0075]** FIG. 1 is a flowchart of an example operating method 100 of coding a video signal. Specifically, a video signal is encoded at an encoder. The encoding process compresses the video

signal by employing various mechanisms to reduce the video file size. A smaller file size allows the compressed video file to be transmitted toward a user, while reducing associated bandwidth overhead. The decoder then decodes the compressed video file to reconstruct the original video signal for display to an end user. The decoding process generally mirrors the encoding process to allow the decoder to consistently reconstruct the video signal.

**[0076]** At step 101, the video signal is input into the encoder. For example, the video signal may be an uncompressed video file stored in memory. As another example, the video file may be captured by a video capture device, such as a video camera, and encoded to support live streaming of the video. The video file may include both an audio component and a video component. The video component contains a series of image frames that, when viewed in a sequence, gives the visual impression of motion. The frames contain pixels that are expressed in terms of light, referred to herein as luma components (or luma samples), and color, which is referred to as chroma components (or color samples). In some examples, the frames may also contain depth values to support three dimensional viewing.

**[0077]** At step 103, the video is partitioned into blocks. Partitioning includes subdividing the pixels in each frame into square and/or rectangular blocks for compression. For example, in High Efficiency Video Coding (HEVC) (also known as H.265 and MPEG-H Part 2) the frame can first be divided into coding tree units (CTUs), which are blocks of a predefined size (e.g., sixty-four pixels by sixty-four pixels). The CTUs contain both luma and chroma samples. Coding trees may be employed to divide the CTUs into blocks and then recursively subdivide the blocks until configurations are achieved that support further encoding. For example, luma components of a frame may be subdivided until the individual blocks contain relatively homogenous lighting values. Further, chroma components of a frame may be subdivided until the individual blocks contain relatively homogenous color values. Accordingly, partitioning mechanisms vary depending on the content of the video frames.

**[0078]** At step 105, various compression mechanisms are employed to compress the image blocks partitioned at step 103. For example, inter-prediction and/or intra-prediction may be employed. Inter-prediction is designed to take advantage of the fact that objects in a common scene tend to appear in successive frames. Accordingly, a block depicting an object in a reference frame need not be repeatedly described in adjacent frames. Specifically, an object, such as a table, may remain in a constant position over multiple frames. Hence the table is described once and adjacent frames can refer back to the reference frame. Pattern matching mechanisms may be employed to match objects over multiple frames. Further, moving objects may be represented across multiple frames, for example due to object movement or camera movement. As a particular example, a video may show an automobile that moves across the screen over multiple frames. Motion vectors can be

employed to describe such movement. A motion vector is a two-dimensional vector that provides an offset from the coordinates of an object in a frame to the coordinates of the object in a reference frame. As such, inter-prediction can encode an image block in a current frame as a set of motion vectors indicating an offset from a corresponding block in a reference frame.

**[0079]** Intra-prediction encodes blocks in a common frame. Intra-prediction takes advantage of the fact that luma and chroma components tend to cluster in a frame. For example, a patch of green in a portion of a tree tends to be positioned adjacent to similar patches of green. Intra-prediction employs multiple directional prediction modes (e.g., thirty-three in HEVC), a planar mode, and a direct current (DC) mode. The directional modes indicate that a current block is similar/the same as samples of a neighbor block in a corresponding direction. Planar mode indicates that a series of blocks along a row/column (e.g., a plane) can be interpolated based on neighbor blocks at the edges of the row. Planar mode, in effect, indicates a smooth transition of light/color across a row/column by employing a relatively constant slope in changing values. DC mode is employed for boundary smoothing and indicates that a block is similar/the same as an average value associated with samples of all the neighbor blocks associated with the angular directions of the directional prediction modes. Accordingly, intra-prediction blocks can represent image blocks as various relational prediction mode values instead of the actual values. Further, inter-prediction blocks can represent image blocks as motion vector values instead of the actual values. In either case, the prediction blocks may not exactly represent the image blocks in some cases. Any differences are stored in residual blocks. Transforms may be applied to the residual blocks to further compress the file.

**[0080]** At step 107, various filtering techniques may be applied. In HEVC, the filters are applied according to an in-loop filtering scheme. The block based prediction discussed above may result in the creation of blocky images at the decoder. Further, the block based prediction scheme may encode a block and then reconstruct the encoded block for later use as a reference block. The in-loop filtering scheme iteratively applies noise suppression filters, de-blocking filters, adaptive loop filters, and sample adaptive offset (SAO) filters to the blocks/frames. These filters mitigate such blocking artifacts so that the encoded file can be accurately reconstructed. Further, these filters mitigate artifacts in the reconstructed reference blocks so that artifacts are less likely to create additional artifacts in subsequent blocks that are encoded based on the reconstructed reference blocks.

**[0081]** Once the video signal has been partitioned, compressed, and filtered, the resulting data is encoded in a bitstream at step 109. The bitstream includes the data discussed above as well as any signaling data desired to support proper video signal reconstruction at the decoder. For example, such data may include partition data, prediction data, residual blocks, and various flags providing coding instructions to the decoder. The bitstream may be stored in memory for transmission toward a decoder upon request. The bitstream may also be broadcast and/or multicast toward a plurality of

decoders. The creation of the bitstream is an iterative process. Accordingly, steps 101, 103, 105, 107, and 109 may occur continuously and/or simultaneously over many frames and blocks. The order shown in FIG. 1 is presented for clarity and ease of discussion, and is not intended to limit the video coding process to a particular order.

**[0082]** The decoder receives the bitstream and begins the decoding process at step 111. Specifically, the decoder employs an entropy decoding scheme to convert the bitstream into corresponding syntax and video data. The decoder employs the syntax data from the bitstream to determine the partitions for the frames at step 111. The partitioning should match the results of block partitioning at step 103. Entropy encoding/decoding as employed in step 111 is now described. The encoder makes many choices during the compression process, such as selecting block partitioning schemes from several possible choices based on the spatial positioning of values in the input image(s). Signaling the exact choices may employ a large number of bins. As used herein, a bin is a binary value that is treated as a variable (e.g., a bit value that may vary depending on context). Entropy coding allows the encoder to discard any options that are clearly not viable for a particular case, leaving a set of allowable options. Each allowable option is then assigned a code word. The length of the code words is based on the number of allowable options (e.g., one bin for two options, two bins for three to four options, etc.) The encoder then encodes the code word for the selected option. This scheme reduces the size of the code words as the code words are as big as desired to uniquely indicate a selection from a small sub-set of allowable options as opposed to uniquely indicating the selection from a potentially large set of all possible options. The decoder then decodes the selection by determining the set of allowable options in a similar manner to the encoder. By determining the set of allowable options, the decoder can read the code word and determine the selection made by the encoder.

**[0083]** At step 113, the decoder performs block decoding. Specifically, the decoder employs reverse transforms to generate residual blocks. Then the decoder employs the residual blocks and corresponding prediction blocks to reconstruct the image blocks according to the partitioning. The prediction blocks may include both intra-prediction blocks and inter-prediction blocks as generated at the encoder at step 105. The reconstructed image blocks are then positioned into frames of a reconstructed video signal according to the partitioning data determined at step 111. Syntax for step 113 may also be signaled in the bitstream via entropy coding as discussed above.

**[0084]** At step 115, filtering is performed on the frames of the reconstructed video signal in a manner similar to step 107 at the encoder. For example, noise suppression filters, de-blocking filters, adaptive loop filters, and SAO filters may be applied to the frames to remove blocking artifacts. Once the frames are filtered, the video signal can be output to a display at step 117 for viewing by an end user.

**[0085]** FIG. 2 is a schematic diagram of an example coding and decoding (codec) system 200 for video coding. Specifically, codec system 200 provides functionality to support the implementation of operating method 100. Codec system 200 is generalized to depict components employed in both an encoder and a decoder. Codec system 200 receives and partitions a video signal as discussed with respect to steps 101 and 103 in operating method 100, which results in a partitioned video signal 201. Codec system 200 then compresses the partitioned video signal 201 into a coded bitstream when acting as an encoder as discussed with respect to steps 105, 107, and 109 in method 100. When acting as a decoder codec system 200 generates an output video signal from the bitstream as discussed with respect to steps 111, 113, 115, and 117 in operating method 100. The codec system 200 includes a general coder control component 211, a transform scaling and quantization component 213, an intra-picture estimation component 215, an intra-picture prediction component 217, a motion compensation component 219, a motion estimation component 221, a scaling and inverse transform component 229, a filter control analysis component 227, an in-loop filters component 225, a decoded picture buffer component 223, and a header formatting and context adaptive binary arithmetic coding (CABAC) component 231. Such components are coupled as shown. In FIG. 2, black lines indicate movement of data to be encoded/decoded while dashed lines indicate movement of control data that controls the operation of other components. The components of codec system 200 may all be present in the encoder. The decoder may include a subset of the components of codec system 200. For example, the decoder may include the intra-picture prediction component 217, the motion compensation component 219, the scaling and inverse transform component 229, the in-loop filters component 225, and the decoded picture buffer component 223. These components are now described.

**[0086]** The partitioned video signal 201 is a captured video sequence that has been partitioned into blocks of pixels by a coding tree. A coding tree employs various split modes to subdivide a block of pixels into smaller blocks of pixels. These blocks can then be further subdivided into smaller blocks. The blocks may be referred to as nodes on the coding tree. Larger parent nodes are split into smaller child nodes. The number of times a node is subdivided is referred to as the depth of the node/coding tree. The divided blocks can be included in coding units (CUs) in some cases. For example, a CU can be a sub-portion of a CTU that contains a luma block, red difference chroma (Cr) block(s), and a blue difference chroma (Cb) block(s) along with corresponding syntax instructions for the CU. The split modes may include a binary tree (BT), triple tree (TT), and a quad tree (QT) employed to partition a node into two, three, or four child nodes, respectively, of varying shapes depending on the split modes employed. The partitioned video signal 201 is forwarded to the general coder control component 211, the transform scaling and quantization component 213,

the intra-picture estimation component 215, the filter control analysis component 227, and the motion estimation component 221 for compression.

**[0087]** The general coder control component 211 is configured to make decisions related to coding of the images of the video sequence into the bitstream according to application constraints. For example, the general coder control component 211 manages optimization of bitrate/bitstream size versus reconstruction quality. Such decisions may be made based on storage space/bandwidth availability and image resolution requests. The general coder control component 211 also manages buffer utilization in light of transmission speed to mitigate buffer underrun and overrun issues. To manage these issues, the general coder control component 211 manages partitioning, prediction, and filtering by the other components. For example, the general coder control component 211 may dynamically increase compression complexity to increase resolution and increase bandwidth usage or decrease compression complexity to decrease resolution and bandwidth usage. Hence, the general coder control component 211 controls the other components of codec system 200 to balance video signal reconstruction quality with bit rate concerns. The general coder control component 211 creates control data, which controls the operation of the other components. The control data is also forwarded to the header formatting and CABAC component 231 to be encoded in the bitstream to signal parameters for decoding at the decoder.

**[0088]** The partitioned video signal 201 is also sent to the motion estimation component 221 and the motion compensation component 219 for inter-prediction. A frame or slice of the partitioned video signal 201 may be divided into multiple video blocks. Motion estimation component 221 and the motion compensation component 219 perform inter-predictive coding of the received video block relative to one or more blocks in one or more reference frames to provide temporal prediction. Codec system 200 may perform multiple coding passes, e.g., to select an appropriate coding mode for each block of video data.

**[0089]** Motion estimation component 221 and motion compensation component 219 may be highly integrated, but are illustrated separately for conceptual purposes. Motion estimation, performed by motion estimation component 221, is the process of generating motion vectors, which estimate motion for video blocks. A motion vector, for example, may indicate the displacement of a coded object relative to a predictive block. A predictive block is a block that is found to closely match the block to be coded, in terms of pixel difference. A predictive block may also be referred to as a reference block. Such pixel difference may be determined by sum of absolute difference (SAD), sum of square difference (SSD), or other difference metrics. HEVC employs several coded objects including a CTU, coding tree blocks (CTBs), and CUs. For example, a CTU can be divided into CTBs, which can then be divided into CBs for inclusion in CUs. A CU can be encoded as a prediction unit (PU) containing prediction data and/or a transform unit (TU) containing transformed

residual data for the CU. The motion estimation component 221 generates motion vectors, PUs, and TUs by using a rate-distortion analysis as part of a rate distortion optimization process. For example, the motion estimation component 221 may determine multiple reference blocks, multiple motion vectors, etc. for a current block/frame, and may select the reference blocks, motion vectors, etc. having the best rate-distortion characteristics. The best rate-distortion characteristics balance both quality of video reconstruction (e.g., amount of data loss by compression) with coding efficiency (e.g., size of the final encoding).

**[0090]** In some examples, codec system 200 may calculate values for sub-integer pixel positions of reference pictures stored in decoded picture buffer component 223. For example, video codec system 200 may interpolate values of one-quarter pixel positions, one-eighth pixel positions, or other fractional pixel positions of the reference picture. Therefore, motion estimation component 221 may perform a motion search relative to the full pixel positions and fractional pixel positions and output a motion vector with fractional pixel precision. The motion estimation component 221 calculates a motion vector for a PU of a video block in an inter-coded slice by comparing the position of the PU to the position of a predictive block of a reference picture. Motion estimation component 221 outputs the calculated motion vector as motion data to header formatting and CABAC component 231 for encoding and motion to the motion compensation component 219.

**[0091]** Motion compensation, performed by motion compensation component 219, may involve fetching or generating the predictive block based on the motion vector determined by motion estimation component 221. Again, motion estimation component 221 and motion compensation component 219 may be functionally integrated, in some examples. Upon receiving the motion vector for the PU of the current video block, motion compensation component 219 may locate the predictive block to which the motion vector points. A residual video block is then formed by subtracting pixel values of the predictive block from the pixel values of the current video block being coded, forming pixel difference values. In general, motion estimation component 221 performs motion estimation relative to luma components, and motion compensation component 219 uses motion vectors calculated based on the luma components for both chroma components and luma components. The predictive block and residual block are forwarded to transform scaling and quantization component 213.

**[0092]** The partitioned video signal 201 is also sent to intra-picture estimation component 215 and intra-picture prediction component 217. As with motion estimation component 221 and motion compensation component 219, intra-picture estimation component 215 and intra-picture prediction component 217 may be highly integrated, but are illustrated separately for conceptual purposes. The intra-picture estimation component 215 and intra-picture prediction component 217 intra-predict a current block relative to blocks in a current frame, as an alternative to the inter-prediction performed

by motion estimation component 221 and motion compensation component 219 between frames, as described above. In particular, the intra-picture estimation component 215 determines an intra-prediction mode to use to encode a current block. In some examples, intra-picture estimation component 215 selects an appropriate intra-prediction mode to encode a current block from multiple tested intra-prediction modes. The selected intra-prediction modes are then forwarded to the header formatting and CABAC component 231 for encoding.

**[0093]** For example, the intra-picture estimation component 215 calculates rate-distortion values using a rate-distortion analysis for the various tested intra-prediction modes, and selects the intra-prediction mode having the best rate-distortion characteristics among the tested modes. Rate-distortion analysis generally determines an amount of distortion (or error) between an encoded block and an original unencoded block that was encoded to produce the encoded block, as well as a bitrate (e.g., a number of bits) used to produce the encoded block. The intra-picture estimation component 215 calculates ratios from the distortions and rates for the various encoded blocks to determine which intra-prediction mode exhibits the best rate-distortion value for the block. In addition, intra-picture estimation component 215 may be configured to code depth blocks of a depth map using a depth modeling mode (DMM) based on rate-distortion optimization (RDO).

**[0094]** The intra-picture prediction component 217 may generate a residual block from the predictive block based on the selected intra-prediction modes determined by intra-picture estimation component 215 when implemented on an encoder or read the residual block from the bitstream when implemented on a decoder. The residual block includes the difference in values between the predictive block and the original block, represented as a matrix. The residual block is then forwarded to the transform scaling and quantization component 213. The intra-picture estimation component 215 and the intra-picture prediction component 217 may operate on both luma and chroma components.

**[0095]** The transform scaling and quantization component 213 is configured to further compress the residual block. The transform scaling and quantization component 213 applies a transform, such as a discrete cosine transform (DCT), a discrete sine transform (DST), or a conceptually similar transform, to the residual block, producing a video block comprising residual transform coefficient values. Wavelet transforms, integer transforms, sub-band transforms or other types of transforms could also be used. The transform may convert the residual information from a pixel value domain to a transform domain, such as a frequency domain. The transform scaling and quantization component 213 is also configured to scale the transformed residual information, for example based on frequency. Such scaling involves applying a scale factor to the residual information so that different frequency information is quantized at different granularities, which may affect final visual quality of the reconstructed video. The transform scaling and quantization component 213 is also

configured to quantize the transform coefficients to further reduce bit rate. The quantization process may reduce the bit depth associated with some or all of the coefficients. The degree of quantization may be modified by adjusting a quantization parameter. In some examples, the transform scaling and quantization component 213 may then perform a scan of the matrix including the quantized transform coefficients. The quantized transform coefficients are forwarded to the header formatting and CABAC component 231 to be encoded in the bitstream.

**[0096]** The scaling and inverse transform component 229 applies a reverse operation of the transform scaling and quantization component 213 to support motion estimation. The scaling and inverse transform component 229 applies inverse scaling, transformation, and/or quantization to reconstruct the residual block in the pixel domain, e.g., for later use as a reference block which may become a predictive block for another current block. The motion estimation component 221 and/or motion compensation component 219 may calculate a reference block by adding the residual block back to a corresponding predictive block for use in motion estimation of a later block/frame. Filters are applied to the reconstructed reference blocks to mitigate artifacts created during scaling, quantization, and transform. Such artifacts could otherwise cause inaccurate prediction (and create additional artifacts) when subsequent blocks are predicted.

**[0097]** The filter control analysis component 227 and the in-loop filters component 225 apply the filters to the residual blocks and/or to reconstructed image blocks. For example, the transformed residual block from the scaling and inverse transform component 229 may be combined with a corresponding prediction block from intra-picture prediction component 217 and/or motion compensation component 219 to reconstruct the original image block. The filters may then be applied to the reconstructed image block. In some examples, the filters may instead be applied to the residual blocks. As with other components in FIG. 2, the filter control analysis component 227 and the in-loop filters component 225 are highly integrated and may be implemented together, but are depicted separately for conceptual purposes. Filters applied to the reconstructed reference blocks are applied to particular spatial regions and include multiple parameters to adjust how such filters are applied. The filter control analysis component 227 analyzes the reconstructed reference blocks to determine where such filters should be applied and sets corresponding parameters. Such data is forwarded to the header formatting and CABAC component 231 as filter control data for encoding. The in-loop filters component 225 applies such filters based on the filter control data. The filters may include a deblocking filter, a noise suppression filter, a SAO filter, and an adaptive loop filter. Such filters may be applied in the spatial/pixel domain (e.g., on a reconstructed pixel block) or in the frequency domain, depending on the example.

**[0098]** When operating as an encoder, the filtered reconstructed image block, residual block, and/or prediction block are stored in the decoded picture buffer component 223 for later use in

motion estimation as discussed above. When operating as a decoder, the decoded picture buffer component 223 stores and forwards the reconstructed and filtered blocks toward a display as part of an output video signal. The decoded picture buffer component 223 may be any memory device capable of storing prediction blocks, residual blocks, and/or reconstructed image blocks.

**[0099]** The header formatting and CABAC component 231 receives the data from the various components of codec system 200 and encodes such data into a coded bitstream for transmission toward a decoder. Specifically, the header formatting and CABAC component 231 generates various headers to encode control data, such as general control data and filter control data. Further, prediction data, including intra-prediction and motion data, as well as residual data in the form of quantized transform coefficient data are all encoded in the bitstream. The final bitstream includes all information desired by the decoder to reconstruct the original partitioned video signal 201. Such information may also include intra-prediction mode index tables (also referred to as codeword mapping tables), definitions of encoding contexts for various blocks, indications of most probable intra-prediction modes, an indication of partition information, etc. Such data may be encoded by employing entropy coding. For example, the information may be encoded by employing context adaptive variable length coding (CAVLC), CABAC, syntax-based context-adaptive binary arithmetic coding (SBAC), probability interval partitioning entropy (PIPE) coding, or another entropy coding technique. Following the entropy coding, the coded bitstream may be transmitted to another device (e.g., a video decoder) or archived for later transmission or retrieval.

**[00100]** FIG. 3 is a block diagram illustrating an example video encoder 300. Video encoder 300 may be employed to implement the encoding functions of codec system 200 and/or implement steps 101, 103, 105, 107, and/or 109 of operating method 100. Encoder 300 partitions an input video signal, resulting in a partitioned video signal 301, which is substantially similar to the partitioned video signal 201. The partitioned video signal 301 is then compressed and encoded into a bitstream by components of encoder 300.

**[00101]** Specifically, the partitioned video signal 301 is forwarded to an intra-picture prediction component 317 for intra-prediction. The intra-picture prediction component 317 may be substantially similar to intra-picture estimation component 215 and intra-picture prediction component 217. The partitioned video signal 301 is also forwarded to a motion compensation component 321 for inter-prediction based on reference blocks in a decoded picture buffer component 323. The motion compensation component 321 may be substantially similar to motion estimation component 221 and motion compensation component 219. The prediction blocks and residual blocks from the intra-picture prediction component 317 and the motion compensation component 321 are forwarded to a transform and quantization component 313 for transform and quantization of the residual blocks. The transform and quantization component 313 may be substantially similar to

the transform scaling and quantization component 213. The transformed and quantized residual blocks and the corresponding prediction blocks (along with associated control data) are forwarded to an entropy coding component 331 for coding into a bitstream. The entropy coding component 331 may be substantially similar to the header formatting and CABAC component 231.

**[00102]** The transformed and quantized residual blocks and/or the corresponding prediction blocks are also forwarded from the transform and quantization component 313 to an inverse transform and quantization component 329 for reconstruction into reference blocks for use by the motion compensation component 321. The inverse transform and quantization component 329 may be substantially similar to the scaling and inverse transform component 229. In-loop filters in an in-loop filters component 325 are also applied to the residual blocks and/or reconstructed reference blocks, depending on the example. The in-loop filters component 325 may be substantially similar to the filter control analysis component 227 and the in-loop filters component 225. The in-loop filters component 325 may include multiple filters as discussed with respect to in-loop filters component 225. The filtered blocks are then stored in a decoded picture buffer component 323 for use as reference blocks by the motion compensation component 321. The decoded picture buffer component 323 may be substantially similar to the decoded picture buffer component 223.

**[00103]** FIG. 4 is a block diagram illustrating an example video decoder 400. Video decoder 400 may be employed to implement the decoding functions of codec system 200 and/or implement steps 111, 113, 115, and/or 117 of operating method 100. Decoder 400 receives a bitstream, for example from an encoder 300, and generates a reconstructed output video signal based on the bitstream for display to an end user.

**[00104]** The bitstream is received by an entropy decoding component 433. The entropy decoding component 433 is configured to implement an entropy decoding scheme, such as CAVLC, CABAC, SBAC, PIPE coding, or other entropy coding techniques. For example, the entropy decoding component 433 may employ header information to provide a context to interpret additional data encoded as codewords in the bitstream. The decoded information includes any desired information to decode the video signal, such as general control data, filter control data, partition information, motion data, prediction data, and quantized transform coefficients from residual blocks. The quantized transform coefficients are forwarded to an inverse transform and quantization component 429 for reconstruction into residual blocks. The inverse transform and quantization component 429 may be similar to inverse transform and quantization component 329.

**[00105]** The reconstructed residual blocks and/or prediction blocks are forwarded to intra-picture prediction component 417 for reconstruction into image blocks based on intra-prediction operations. The intra-picture prediction component 417 may be similar to intra-picture estimation component 215 and an intra-picture prediction component 217. Specifically, the intra-picture prediction

component 417 employs prediction modes to locate a reference block in the frame and applies a residual block to the result to reconstruct intra-predicted image blocks. The reconstructed intra-predicted image blocks and/or the residual blocks and corresponding inter-prediction data are forwarded to a decoded picture buffer component 423 via an in-loop filters component 425, which may be substantially similar to decoded picture buffer component 223 and in-loop filters component 225, respectively. The in-loop filters component 425 filters the reconstructed image blocks, residual blocks and/or prediction blocks, and such information is stored in the decoded picture buffer component 423. Reconstructed image blocks from decoded picture buffer component 423 are forwarded to a motion compensation component 421 for inter-prediction. The motion compensation component 421 may be substantially similar to motion estimation component 221 and/or motion compensation component 219. Specifically, the motion compensation component 421 employs motion vectors from a reference block to generate a prediction block and applies a residual block to the result to reconstruct an image block. The resulting reconstructed blocks may also be forwarded via the in-loop filters component 425 to the decoded picture buffer component 423. The decoded picture buffer component 423 continues to store additional reconstructed image blocks, which can be reconstructed into frames via the partition information. Such frames may also be placed in a sequence. The sequence is output toward a display as a reconstructed output video signal.

**[00106]** FIG. 5 is a schematic diagram illustrating an example bitstream 500 containing an encoded video sequence. For example, the bitstream 500 can be generated by a codec system 200 and/or an encoder 300 for decoding by a codec system 200 and/or a decoder 400. As another example, the bitstream 500 may be generated by an encoder at step 109 of method 100 for use by a decoder at step 111.

**[00107]** The bitstream 500 includes a sequence parameter set (SPS) 510, a plurality of picture parameter sets (PPSs) 512, slice headers 514, and image data 520. An SPS 510 contains sequence data common to all the pictures in the video sequence contained in the bitstream 500. Such data can include picture sizing, bit depth, coding tool parameters, bit rate restrictions, etc. The PPS 512 contains parameters that are specific to one or more corresponding pictures. Hence, each picture in a video sequence may refer to one PPS 512. The PPS 512 can indicate coding tools available for slices (or tiles) in corresponding pictures, quantization parameters, offsets, picture specific coding tool parameters (e.g., filter controls), etc. The slice header 514 contains parameters that are specific to each slice in a picture. Hence, there may be one slice header 514 per slice in the video sequence. The slice header 514 may contain slice information, picture order counts (POCs), reference picture lists, prediction weights, tile entry points, deblocking parameters, etc. It should be noted that some systems refer to the slice header 514 as a tile group header, and use such information to support tile groups instead of slices.

**[00108]** The image data 520 contains video data encoded according to inter-prediction and/or intra-prediction as well as corresponding transformed and quantized residual data. Such image data 520 is sorted according to the partitioning used to partition the image prior to encoding. For example, the image in the image data 520 includes one or more pictures 521. The sequence or series of pictures 521 may be referred to as a CVS 527. As used herein, the CVS 527 is a sequence of access units (AUs) that include, in decoding order, a coded video sequence start (CVSS) AU, followed by zero or more AUs that are not CVSS AUs, including all subsequent AUs up to but not including any subsequent AU that is a CVSS AU. A CVSS AU is an AU in which there is a prediction unit (PU) for each layer specified by the video parameter set (VPS) and the coded picture in each PU is a coded layer video sequence start (CLVSS) picture. In an embodiment, each picture 521 is disposed within an AU. A PU is a set of NAL units that are associated with each other according to a specified classification rule, are consecutive in decoding order, and contain exactly one coded picture.

**[00109]** The CVS is a coded video sequence for every coded layer video sequence (CLVS) in the video bitstream. Notably, the CVS and the CLVS are the same when the video bitstream includes a single layer. The CVS and the CLVS are only different when the video bitstream includes multiple layers.

**[00110]** Each of the pictures 521 may be divided into subpictures 523. A subpicture 523 is a partitioned portion of a picture created by horizontal and vertical boundaries. Subpictures 523 may be rectangular and/or square. Specifically, a subpicture 523 includes four sides that are connected at right angles. The four sides include two pair of parallel sides. Further, the sides in a parallel side pair are of equal length. As such, a subpicture 523 may be any rectangular shape, where a square is a special case of a rectangle where all four sides are of equal length.

**[00111]** Each subpicture 523 may be partitioned into slices 529. It should be noted, in some applications the slices 529 may be referred to as tiles and/or tile groups. Slices 529 may be rectangular and/or square, meaning that the slices have a rectangle or square shape (as opposed to, for example, raster scan slices that may not have a rectangular or square shape). Specifically, a slice 529 includes four sides that are connected at right angles. The four sides include two pair of parallel sides. Further, the sides in a parallel side pair are of equal length. As such, a slice 529 may be any rectangular shape, where a square is a special case of a rectangle where all four sides are of equal length.

**[00112]** In an embodiment, each subpicture 523 comprises a rectangular region of one or more slices 529 within a picture 521. That is, each subpicture 523 is divided into one or more slices 529, and each picture 521 is divided into one or more subpictures 523.

**[00113]** Slices 529 may or may not allow prediction based on other slices 529, depending on the example. Each slice 529 may have a unique slice index in the subpicture 523. A slice index is a

procedurally selected numerical identifier that can be used to distinguish one slice 529 from another. For example, slice indices may increase numerically from top to bottom or left to right.

**[00114]** It should be noted that, in some examples, slices 529 may also be assigned slice identifiers (IDs). A slice ID is an assigned identifier that can be used to distinguish one slice 529 from another. Computations may employ slice IDs instead of slice indices in some examples. Further, slice IDs can be assigned to have the same values as the slice indices in some examples. Slice indices and/or IDs may be signaled to indicate slices groups containing the slices 529. For example, the slice indices and/or IDs may be employed to map picture data associated with a slice 529 to a proper position for display. A plurality of slices 529 may be referred to as a tile set. A tile set is a related set of slices 529 that can be separately extracted and coded, for example to support display of a region of interest and/or to support parallel processing. Slices 529 in a tile set can be coded without reference to slices 529 outside of the tile group. Each slice 529 may be assigned to a corresponding tile set, and therefore a picture 521 can contain a plurality of tile sets.

**[00115]** The slices 529 are further divided into coding tree units (CTUs). The CTUs are further divided into coding blocks based on coding trees. The coding blocks can then be encoded/decoded according to prediction mechanisms.

**[00116]** In an embodiment, the PPS 512 contains a single slice per subpicture flag 532. In an embodiment, the single slice per subpicture flag 532 has a first value (e.g., one). The first value specifies that each subpicture (e.g., subpicture 523) referring to the PPS 512 (as opposed to another PPS in the bitstream 500) may include one, and only one, rectangular slice (e.g., slice 529). That is, each subpicture includes a single rectangular slice. Thus, the dimensions of the subpicture and the rectangular slice are the same.

**[00117]** In an embodiment, the single slice per subpicture flag 532 has a second value (e.g., zero). The second value specifies that each subpicture (e.g., subpicture 523) referring to the PPS 512 (as opposed to another PPS in the bitstream 500) may include one or more rectangular slices (e.g., slices 529). That is, each subpicture may include a single rectangular slice or may contain a plurality of the slices. When a subpicture includes a plurality of the slices, the dimensions of the subpicture are different than the dimensions of the individual slices. In an embodiment, the single slice per subpicture flag 532 is designated as pps_single_slice_per_subpic_flag.

**[00118]** In an embodiment, the PPS 512 further comprises a number of slices in picture flag 534. The number of slices in picture flag 534 is used to determine the number of rectangular slices (e.g., slices 529) in each picture (e.g., 521) referring to the PPS 512. In an embodiment, the number of slices in picture flag 534 is utilized to signify the number of rectangular slices in each picture referring to the PPS when the single slice per subpicture flag 532 has the second value.

**[00119]** In an embodiment, a value of the number of slices in picture flag 534, plus one, specifies the number of slices in each picture referring to the PPS. For example, when the value of the number of slices in picture flag 534 is three, the total number of slices in the picture is four (e.g., three + one). In an embodiment, the number of slices in picture flag 534 is designated as pps_num_slices_in_pic_minus1.

**[00120]** FIG. 6 is an embodiment of a method 600 of decoding implemented by a video decoder (e.g., video decoder 400). The method 600 may be performed after the decoded bitstream has been directly or indirectly received from a video encoder (e.g., video encoder 300). The method 600 improves the decoding process by using a flag in the bitstream to indicate whether each subpicture of a picture contains one, and only one, rectangular slice. When the flag indicates that each subpicture contains only a single rectangular slice, the syntax elements for signaling the number and layout of subpictures (e.g., the position and size of each subpicture) can be omitted from the bitstream because the subpictures are the same as the slices. That is, only the syntax elements for the slices need to be included in the bitstream when the flag indicates each subpicture contains only a single rectangular slice. This reduces the amount of information carried in the bitstream, reduces redundancy, and increases coding efficiency. Thus, the coder / decoder (a.k.a., "codec") in video coding is improved relative to current codecs. As a practical matter, the improved video coding process offers the user a better user experience when videos are sent, received, and/or viewed.

**[00121]** In block 602, the video decoder receives a bitstream (e.g., bitstream 500) including a PPS (e.g., PPS 512) containing a single slice per subpicture flag (e.g., single slice per subpicture flag 532). In an embodiment, the single slice per subpicture flag is designated as pps_single_slice_per_subpic_flag.

**[00122]** In an embodiment, the PPS further comprises a number of slices in picture flag (e.g., number of slices in picture flag 534). A value of the number of slices in picture flag plus one specifies a number of slices in each picture referring to the PPS. In an embodiment, the number of slices in picture flag is designated as pps_num_slices_in_pic_minus1.

**[00123]** In block 604, the video decoder determines whether the single slice per subpicture flag has a first value or a second value. The first value specifies that each subpicture referring to the PPS includes one and only one rectangular slice. The second value specifies that each subpicture referring to the PPS may include one or more rectangular slices (e.g., one or a plurality of rectangular slices). In an embodiment, the first value is one (1) and the second value is zero (0). In an embodiment, each subpicture comprises a rectangular region of one more slices within a picture.

**[00124]** In block 606, the video decoder decodes the one and only one rectangular slice when the single slice per subpicture flag has the first value and the one or more rectangular slices when

the single slice per subpicture flag has the second value. Once decoded, the one and only one rectangular slice or the one or more rectangular slices, as the case may be, are used to generate or produce an image or video sequence for display to a user on the display or screen of an electronic device (e.g., a smart phone, tablet, laptop, personal computer, etc.).

**[00125]** FIG. 7 is an embodiment of a method 700 of encoding a video bitstream implemented by a video encoder (e.g., video encoder 300). The method 700 may be performed when a picture (e.g., from a video) is to be encoded into a video bitstream and then transmitted toward a video decoder (e.g., video decoder 400). The method 700 improves the encoding process by using a flag in the bitstream to indicate whether each subpicture of a picture contains one, and only one, rectangular slice. When the flag indicates that each subpicture contains only a single rectangular slice, the syntax elements for signaling the number and layout of subpictures (e.g., the position and size of each subpicture) can be omitted from the bitstream because the subpictures are the same as the slices. That is, only the syntax elements for the slices need to be included in the bitstream when the flag indicates each subpicture contains only a single rectangular slice. This reduces the amount of information carried in the bitstream, reduces redundancy, and increases coding efficiency. Thus, the coder / decoder (a.k.a., "codec") in video coding is improved relative to current codecs. As a practical matter, the improved video coding process offers the user a better user experience when videos are sent, received, and/or viewed.

**[00126]** In block 702, the video encoder sets a single slice per subpicture flag (e.g., single slice per subpicture flag 532) to a first value when each subpicture (e.g., subpicture 523) referring to a PPS (e.g., PPS 512) includes one and only one rectangular slice and to a second value when each subpicture referring to the PPS may include one or more rectangular slices.

**[00127]** In an embodiment, the single slice per subpicture flag is designated as pps_single_slice_per_subpic_flag. In an embodiment, the first value is one (1) and the second value is zero (0).

**[00128]** In an embodiment, the PPS further comprises a number of slices in picture flag (e.g., number of slices in picture flag 534). In an embodiment, the video encoder sets the number of slices in picture flag to a value. The value of the number of slices in picture flag plus one specifies a number of slices in each picture referring to the PPS. In an embodiment, the number of slices in picture flag is designated as pps_num_slices_in_pic_minus1. In an embodiment, each subpicture comprises a rectangular region of one more slices within a picture.

**[00129]** In block 704, the video encoder encodes the PPS containing the single slice per subpicture flag into a bitstream (e.g., bitstream 500). In block 706, the video encoder stores the bitstream for communication toward a decoder. The bitstream may be stored in memory until the video bitstream is transmitted toward the video decoder. Once received by the video decoder, the

encoded video bitstream may be decoded (e.g., as described above) to generate or produce an image or video sequence for display to a user on the display or screen of an electronic device (e.g., a smart phone, tablet, laptop, personal computer, etc.).

**[00130]** The following syntax and semantics may be employed to implement the embodiments disclosed herein. The following description is relative to the basis text, which is the latest VVC draft specification, assuming that the signaling of subpicture information is signaled based on the layout of slices. In other words, only the delta is described, while the text in the basis text that are not mentioned below apply as they are.

**[00131]** The PPS syntax is as follows.

| pic_parameter_set_rbsp( ) { | **Descriptor** |
|---|---|
| **...** | |
| **single_tile_in_pic_flag** | u(1) |
| if( !single_tile_in_pic_flag ) { | |
| **...** | |
| **loop_filter_across_bricks_enabled_flag** | u(1) |
| if( loop_filter_across_bricks_enabled_flag ) | |
| **loop_filter_across_slices_enabled_flag** | u(1) |
| } | |
| if( rect_slice_flag ) { | |
| **subpics_present_flag** | u(1) |
| if( subpics_present_flag ) { | |
| if( num_slices_in_pic_minus1 > 0 ) | |
| **single_slice_per_subpic_flag** | u(1) |
| if( !single_slice_per_subpic_flag ) | |
| **num_subpics_minus1** | ue(v) |
| if( num_subpics_minus1 > 0 ) | |
| if( !single_slice_per_subpic_flag ) { | |
| **bottom_right_slice_idx_length_minus1** | ue(v) |
| for( i = 0; i < num_subpics_minus1; i++ ) { | |
| **bottom_right_slice_idx_delta**[ i ] | u(v) |
| if( i > 0 ) | |

| | |
|---|---|
| **slice_idx_delta_sign_flag**[ i ] | u(1) |
| } | |
| } | |
| for( i = 0; i <= num_subpics_minus1; i++ ) { | |
| **subpic_treated_as_pic_flag**[ i ] | u(1) |
| **loop_filter_across_subpic_enabled_flag**[ i ] | u(1) |
| } | |
| } | |
| **subpic_ids_constant_in_clvs_flag** | u(1) |
| **subpic_id_len_minus1** | ue(v) |
| for( i = 0; i <= num_subpics_minus1; i++ ) | |
| **pps_subpic_id**[ i ] | u(v) |
| } | |
| } | |
| **entropy_coding_sync_enabled_flag** | u(1) |
| **...** | |
| rbsp_trailing_bits( ) | |
| } | |

**[00132]** The PPS semantics are as follows.

**[00133]** **brick_idx_delta_sign_flag**[ i ] equal to 1 indicates a positive sign for bottom_right_brick_idx_delta[ i ]. brick_idx_delta_sign_flag[ i ] equal to 0 indicates a negative sign for bottom_right_brick_idx_delta[ i ].

**[00134]** It is a requirement of bitstream conformance that a slice shall include either a number of complete tiles or only a consecutive sequence of complete bricks of one tile.

**[00135]** The variable TopLeftBrickIdx[ i ], BottomRightBrickIdx[ i ], NumBricksInSlice[ i ], BrickToSliceMap[ j ], SliceColBd[ i ], SliceRowBd[ i ], SliceWidth[ i ], and SliceHeight[ i ], which specify the brick index of the brick located at the top left corner of the i-th slice, the brick index of the brick located at the bottom right corner of the i-th slice, the number of bricks in the i-th slice, and the slice index of the slice containing the j-th brick, location of the left boundary of the i-th slice in units of CTBs, location of the top boundary of the i-th slice in units of CTBs, the width of the i-th

slice in units of CTBs, and the height of the i-th slice in units of CTBs, respectively, are derived as follows:

```
for( j = 0; i  = =  0  &&  j < NumBricksInPic; j++ )
   BrickToSliceMap[ j ] = −1
NumBricksInSlice[ i ] = 0
if( i  = =  num_slices_in_pic_minus1 )
   BottomRightBrickIdx[ i ] = NumBricksInPic – 1
else
   BottomRightBrickIdx[ i ] = i  = =  0 ? bottom_right_brick_idx_delta[ i ] ] :
       ( BottomRightBrickIdx[ i − 1 ] + ( brick_idx_delta_sign_flag[ i ] ?
             bottom_right_brick_idx_delta[ i ] : −bottom_right_brick_idx_delta[ i ] ) )
for( j = BottomRightBrickIdx[ i ]; j  >=  0; j− − ) {
   if( BrickColBd[ j ] + BrickWidth[ j ]  <=  BrickColBd[ BottomRightBrickIdx[ i ] ] +
         BrickWidth[ BottomRightBrickIdx[ i ] ]  &&
      BrickRowBd[ j ] + BrickHeight[ j ]  <=  BrickRowBd[ BottomRightBrickIdx[ i ] ] +
         BrickHeight[ BottomRightBrickIdx[ i ] ]  &&
     BrickToSliceMap[ j ]  = =  −1 ) {
     TopLeftBrickIdx[ i ] = j
     NumBricksInSlice[ i ]++
     BrickToSliceMap[ j ] = i
   }
}
SliceColBd[ i ] = BrickColBd[ TopLeftBrickIdx[ i ] ]
SliceRowBd[ i ] = BrickRowBd[ TopLeftBrickIdx[ i ] ]
SliceWidth[ i ] = BrickColBd[ BottomRightBrickIdx[ i ] ] +
             BrickWidth[ BottomRightBrickIdx[ i ] ] − SliceColBd[ i ]
SliceHeight[ i ] = BrickRowBd[ BottomRightBrickIdx[ i ] ] +
             BrickHeight[ BottomRightBrickIdx[ i ] ] − SliceRowBd[ i ]
```

**[00136]** **subpics_present_flag** equal to 1 indicates that subpicture parameters are present in the PPS RBSP syntax. subpics_present_flag equal to 0 indicates that subpicture parameters are not present in the PPS RBSP syntax. When not present, the value of subpics_present_flag is inferred to be equal to 0.

**[00137]** NOTE – When a bitstream is the result of a subpicture based sub-bitstream extraction process wherein the input bitstream contains multiple subpictures per picture and during the extraction VCL NAL units may only be extracted or discarded but not changed, and the bitstream

contains only a true subset of the subpictures of the input bitstream, the value of subpics_present_flag has to be equal to 1 in the RBSP of the PPSs in the bitstream, even when there is only one subpicture in each picture.

**[00138]** **single_slice_per_subpic_flag** equal to 1 specifies that each subpicture in each picture referring to this PPS includes one rectangular slice. single_slice_per_subpic_flag equal to 0 specifies that a subpicture in a picture referring to this PPS may include more than one rectangular slice. When rect_slice_flag is equal to 1 and num_slices_in_pic_minus1 is equal to 0, the value of single_slice_per_subpic_flag is inferred to be equal to 1.

**[00139]** **num_subpics_minus1** plus 1 specifies the number of subpictures in each picture referring to the PPS. num_subpics_minus1 shall be in the range of 0 to num_slices_in_pic_minus1, inclusive. When subpics_present_flag is equal to 0, the value of num_subpics_minus1 is inferred to 0. When not present and single_slice_per_subpic_flag is equal to 1, the value of num_subpics_minus1 is inferred to be equal to num_slices_in_pic_minus1. The variable NumSubPics is derived to be equal to num_subpics_minus1 + 1.

**[00140]** **bottom_right_slice_idx_length_minus1** plus 1 specifies the number of bits used to represent the syntax element bottom_right_slice_idx_delta[ i ]. The value of bottom_right_slice_idx_length_minus1 shall be in the range of 0 to Ceil( Log2( num_slices_in_pic_minus1 + 1 ) ) − 1, inclusive.

**[00141]** **bottom_right_slice_idx_delta**[ i ] when i is greater than 0 specifies the difference between the slice index of the slice located at the bottom-right corner of the i-th subpicture and the slice index of the bottom-right corner of the ( i – 1 )-th subpicture. bottom_right_slice_idx_delta[ 0 ] specifies the slice index of the bottom right corner of the 0-th subpicture. When single_slice_per_subpic_flag is equal to 1, the value of bottom_right_slice_idx_delta[ i ] is inferred to be equal to 1. The value of the BottomRightSliceIdx[ num_subpics_minus1 ] is inferred to be equal to num_slices_in_pic_minus1. The length of the bottom_right_slice_idx_delta[ i ] syntax element is bottom_right_slice_idx_length_minus1 + 1 bits.

**[00142]** **slice_idx_delta_sign_flag**[ i ] equal to 1 indicates a positive sign for bottom_right_slice_idx_delta[ i ]. slice_idx_delta_sign_flag[ i ] equal to 0 indicates a negative sign for bottom_right_brick_idx_delta[ i ].

**[00143]** The variable TopLeftSliceIdx[ i ], BottomRightSliceIdx[ i ], NumSlicesInSubpic[ i ], SliceToSubpicMap[ j ], and SliceSubpicToPicIdx[ i ][ k ], which specify the slice index of the slice located at the top left corner of the i-th subpicture, the slice index of the slice located at the bottom right corner of the i-th subpicture, the number of slices in the i-th subpicture, the subpicture index of the subpicture containing the j-th slice, and picture level slice index of the k-th slice in the i-th subpicture, respectively, are derived as follows.

```
if( single_slice_per_subpic_flag ) {
  TopLeftSliceIdx[ i ]  = i
  BottomRightSliceIdx[ i ] = i
  NumSlicesInSubpic[ i ] = 1
  SliceToSubpicMap[ i ] = i
  SliceSubpicToPicIdx[ i ][ 0 ] = 0
} else {
  for( j = 0; i  = =  0  &&  j  <=  num_slices_in_pic_minus1; j++ )
    SliceToSubpicMap[ j ] = −1
  NumSlicesInSubpic[ i ] = 0
  if( i  = =  num_subpics_minus1 )
    BottomRightSliceIdx[ i ] = num_slices_in_pic_minus1
  else
    BottomRightSliceIdx[ i ] = i  = =  0 ? bottom_right_slice_idx_delta[ i ] :
        ( BottomRightSliceIdx[ i−1 ] + ( slice_idx_delta_sign_flag[ i ] ?
            bottom_right_slice_idx_delta[ i ] : −bottom_right_slice_idx_delta[ i ] ) )
  for( j = BottomRightSliceIdx[ i ]; j  >=  0; j− − ) {
    if( SliceColBd[ j ] + SliceWidth[ j ]  <=

SliceColBd[ BottomRightSliceIdx[ i ] ] + SliceWidth[BottomRightSliceIdx[ i ] ]  &&
        SliceRowBd[ j ] + SliceHeight[ j ]  <=

SliceRowBd[ BottomRightSliceIdx[ i ] ] + SliceHeight[BottomRightSliceIdx[ i ] ]  &&
            SliceToSubpicMap[ j ]  = =  −1 ) {
      TopLeftSliceIdx[ i ] = j
      NumSlicesInSubpic[ i ]++
      SliceToSubpicMap[ j ] = i
    }
  }
  for( j = 0, k = 0; j < num_slices_in_pic_minus1  &&  k < NumSlicesInSubpic[ i ]; j++ )
    if( SliceToSubpicMap[ j ]  = =  i )
      SliceSubpicToPicIdx[ i ][ k++ ] = j
}
```

**[00144]** The variables SubpicLeft[ i ], SubpicTop[ i ], SubpicWidth, and SubpicHeight[ i ], which specify the left boundary position, top boundary boundary position, width, and height,

respectively, of the i-th subpicture in unit of CTBs are derived as follows for each value of i in the range of 0 to NumSubPics − 1, inclusive.

SubpicLeft[ i ] = SliceColBd[ TopLeftSliceIdx[ i ] ]
SubpicWidth[ i ] = SliceColBd[ BottomRightSliceIdx[ i ] ] +
SliceWidth[ BottomRightSliceIdx[ i ] ] − SubpicLeft[ i ]
SubpicTop[ i ] = SliceColBd[ TopLeftSliceIdx[ i ] ]
SubpicHeight[ i ] = SliceColBd[ BottomRightSliceIdx[ i ] ] +
SliceHeight[ BottomRightSliceIdx[ i ] ] − SubpicTop[ i ]

**[00145]** **subpic_treated_as_pic_flag**[ i ] equal to 1 specifies that the i-th subpicture of each coded picture in the CVS is treated as a picture in the decoding process excluding in-loop filtering operations. subpic_treated_as_pic_flag[ i ] equal to 0 specifies that the i-th subpicture of each coded picture in the CVS is not treated as a picture in the decoding process excluding in-loop filtering operations. When not present, the value of subpic_treated_as_pic_flag[ i ] is inferred to be equal to 0.

**[00146]** **loop_filter_across_subpic_enabled_flag**[ i ] equal to 1 specifies that in-loop filtering operations may be performed across the boundaries of the i-th subpicture in each coded picture in the CVS. loop_filter_across_subpic_enabled_flag[ i ] equal to 0 specifies that in-loop filtering operations are not performed across the boundaries of the i-th subpicture in each coded picture in the CVS. When not present, the value of loop_filter_across_subpic_enabled_pic_flag[ i ] is inferred to be equal to 1.

**[00147]** It is a requirement of bitstream conformance that the following constraints apply.

**[00148]** - For any two subpictures subpicA and subpicB, when the index of subpicA is less than the index of subpicB, any coded NAL unit of subPicA shall succeed any coded NAL unit of subPicB in decoding order.

**[00149]** The shapes of the subpictures shall be such that each subpicture, when decoded, shall have its entire left boundary and entire top boundary consisting of picture boundaries or consisting of boundaries of previously decoded subpictures.

**[00150]** The values of SubpicLeft[ i ], SubpicTop[ i ], SubpicWidth[ i ] and SubpicHeight[ i ] shall be the same, respectively for each value of i in the range 0 to NumSubPics − 1, inclusive, regardless of which PPS RBSP referenced by the coded slice NAL units of a CLVS they are derived from.

**[00151]** The values of subpic_treated_as_pic_flag[ i ], and loop_filter_across_subpic_enabled_flag[ i ] shall remain the same in all PPS RBSPs referenced by the coded slice NAL units of a CLVS respectively for each value of i in the range of 0 to NumSubPics − 1, inclusive.

**[00152]** **subpic_ids_constant_in_clvs_flag** equal to 1 indicates that subpic_id_len_minus1 and the values of pps_subpic_id[ i ], for each value of i in the range of 0 to num_subpics_minus1, inclusive, remain the same in all PPS RBSPs referenced by the coded slice NAL units of a CLVS. subpic_ids_constant_in_clvs_flag equal to 0 indicates that the values of subpic_id_len_minus1 and the values of pps_subpic_id[ i ] may or may not be constrained.

**[00153]** **subpic_id_len_minus1** plus 1 specifies the length of the pps_subpic_id[ i ] and slice_subpic_id syntax elements in bits. The value of subpic_id_len_minus1 shall be in the range of 3 to 31, inclusive. The values of subpic_id_len_minus1 in the range of 0 to 2, inclusive, are reserved for future use by ITU-T | ISO/IEC.

**[00154]** pps_subpic_id[ i ] specifies the identifier of the i-th subpicture.

**[00155]** FIG. 8 is a schematic diagram of a video coding device 800 (e.g., a video encoder 300 or a video decoder 400) according to an embodiment of the disclosure. The video coding device 800 is suitable for implementing the disclosed embodiments as described herein. The video coding device 800 comprises ingress ports 810 and receiver units (Rx) 820 for receiving data; a processor, logic unit, or central processing unit (CPU) 830 to process the data; transmitter units (Tx) 840 and egress ports 850 for transmitting the data; and a memory 860 for storing the data. The video coding device 800 may also comprise optical-to-electrical (OE) components and electrical-to-optical (EO) components coupled to the ingress ports 810, the receiver units 820, the transmitter units 840, and the egress ports 850 for egress or ingress of optical or electrical signals.

**[00156]** The processor 830 is implemented by hardware and software. The processor 830 may be implemented as one or more CPU chips, cores (e.g., as a multi-core processor), field-programmable gate arrays (FPGAs), application specific integrated circuits (ASICs), and digital signal processors (DSPs). The processor 830 is in communication with the ingress ports 810, receiver units 820, transmitter units 840, egress ports 850, and memory 860. The processor 830 comprises a coding module 870. The coding module 870 implements the disclosed embodiments described above. For instance, the coding module 870 implements, processes, prepares, or provides the various codec functions. The inclusion of the coding module 870 therefore provides a substantial improvement to the functionality of the video coding device 800 and effects a transformation of the video coding device 800 to a different state. Alternatively, the coding module 870 is implemented as instructions stored in the memory 860 and executed by the processor 830.

**[00157]** The video coding device 800 may also include input and/or output (I/O) devices 880 for communicating data to and from a user. The I/O devices 880 may include output devices such as a display for displaying video data, speakers for outputting audio data, etc. The I/O devices 880 may also include input devices, such as a keyboard, mouse, trackball, etc., and/or corresponding interfaces for interacting with such output devices.

**[00158]** The memory 860 comprises one or more disks, tape drives, and solid-state drives and may be used as an over-flow data storage device, to store programs when such programs are selected for execution, and to store instructions and data that are read during program execution. The memory 860 may be volatile and/or non-volatile and may be read-only memory (ROM), random access memory (RAM), ternary content-addressable memory (TCAM), and/or static random-access memory (SRAM).

**[00159]** FIG. 9 is a schematic diagram of an embodiment of a means for coding 900. In an embodiment, the means for coding 900 is implemented in a video coding device 902 (e.g., a video encoder 300 or a video decoder 400). The video coding device 902 includes receiving means 901. The receiving means 901 is configured to receive a picture to encode or to receive a bitstream to decode. The video coding device 902 includes transmission means 907 coupled to the receiving means 901. The transmission means 907 is configured to transmit the bitstream to a decoder or to transmit a decoded image to a display means (e.g., one of the I/O devices 880).

**[00160]** The video coding device 902 includes a storage means 903. The storage means 903 is coupled to at least one of the receiving means 901 or the transmission means 907. The storage means 903 is configured to store instructions. The video coding device 902 also includes processing means 905. The processing means 905 is coupled to the storage means 903. The processing means 905 is configured to execute the instructions stored in the storage means 903 to perform the methods disclosed herein.

**[00161]**

**[00162]** While several embodiments have been provided in the present disclosure, it may 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.

**[00163]** 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, components, techniques, or methods without departing from the scope of the present disclosure. Other examples of changes, substitutions, and alterations are ascertainable by one skilled in the art and may be made without departing from the spirit and scope disclosed herein.

## CLAIMS

What is claimed is:

1. A method implemented by a decoder, comprising:

receiving, by the decoder, a bitstream including a picture parameter set (PPS) including a single slice per subpicture flag and num_subpics_minus1, wherein the num_subpics_minus1 plus 1 specifies the number of subpictures in each picture;

determining, by the decoder, whether the single slice per subpicture flag has a first value or a second value, wherein the first value specifies that each subpicture referring to the PPS includes one and only one rectangular slice, and wherein the second value specifies that each subpicture referring to the PPS may include one or more rectangular slices.

2. A method implemented by an encoder, the method comprising:

encoding, by the encoder, the PPS containing the single slice per subpicture flag and num_subpics_minus1 into a bitstream, wherein the num_subpics_minus1 plus 1 specifies the number of subpictures in each picture.

3. A decoding device, comprising:

a receiver configured to receive a bitstream including a picture parameter set (PPS) containing a single slice per subpicture flag and num_subpics_minus1, wherein the num_subpics_minus1 plus 1 specifies the number of subpictures in each picture;

a memory coupled to the receiver, the memory storing instructions; and

a processor coupled to the memory, the processor configured to execute the instructions to cause the decoding device to:

determine whether the single slice per subpicture flag has a first value or a second value, wherein the first value specifies that each subpicture referring to the PPS includes one and only one rectangular slice, and wherein the second value specifies that each subpicture referring to the PPS may include one or more rectangular slices.

4. An encoding device, comprising:

a memory containing instructions;

a processor coupled to the memory, the processor configured to implement the instructions to cause the encoding device to:

set the single slice per subpicture flag to a first value when each subpicture referring to the PPS includes one and only one rectangular slice and to a second value when each subpicture referring to the PPS may include one or more rectangular slices.

5. A coding apparatus, comprising:

a receiver configured to receive a picture to encode or to receive a bitstream to decode;

a transmitter coupled to the receiver, the transmitter configured to transmit the bitstream to a decoder or to transmit a decoded image to a display;

a memory coupled to at least one of the receivers or the transmitter, the memory configured to store instructions; and

a processor coupled to the memory, the processor configured to execute the instructions stored in the memory to perform the method in claim 1 and claim 2.

6. A system, comprising:

an encoder; and

a decoder in communication with the encoder, wherein the encoder or the decoder includes the decoding device, the encoding device, or the coding apparatus of any of claims 1 to 2.

7. An encoder comprising processing circuitry for carrying out the method according to claim 2.

8. A decoder comprising processing circuitry for carrying out the method according to claim 1.

9. A computer program product comprising a program code for performing the method according to any one of claims 1 to 2.

10. A terminal, wherein the terminal comprises one or more processors, a memory, and a communications interface; and the memory and the communications interface are connected to the one or more processors, the terminal communicates with another device through the communications interface, the memory is configured to store computer program code, the computer program code comprises an instruction, and when the one or more processors execute the instruction, the terminal performs the method according to any one of claims 1 to 2.

11. An encoded bitstream, the bitstream comprising a picture parameter set (PPS) including a single slice per subpicture flag and num_subpics_minus1, wherein the num_subpics_minus1 plus 1

specifies the number of subpictures in each picture, and wherein a first value of the single slice per subpicture flag specifies that each subpicture referring to the PPS includes one and only one rectangular slice, and wherein a second value of the single slice per subpicture flag specifies that each subpicture referring to the PPS may include one or more rectangular slices.