A VIDEO ENCODER, A VIDEO DECODER AND CORRESPONDING METHODS

TECHNICAL FIELD

[0001] The present disclosure is generally related to video coding, and is specifically related to a flexible video tiling scheme that supports multiple tiles with different resolutions in the same picture.

BACKGROUND

[0002] 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.

[0003] 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] In an embodiment, the disclosure includes a method implemented in an encoder, the method comprising: partitioning, by a processor of the encoder, a picture into a plurality of first level tiles; partitioning, by the processor, a subset of the first level tiles into a plurality of second level tiles; assigning, by the processor, the first level tiles and the second level tiles into one or more tile groups such that each tile group contains a number of first level tiles, one or

more consecutive sequences of second level tiles where each sequence of second level tiles is split from a single first level tile, or combinations thereof; encoding, by the processor, the first level tiles and the second level tiles into a bitstream; and storing, in a memory of the encoder, the bitstream for communication toward a decoder. Certain streaming applications (e.g., virtual reality (VR) and teleconferencing) may be improved if a single image can be sent containing multiple regions encoded at different resolutions. Some slicing and tiling mechanisms, such as raster scan based slicing and/or tiling, may not support such functionality because tiles at different resolutions may be treated differently. For example, a tile at a first resolution may contain a single slice of data while a tile at a second resolution may carry multiple slices of data due to differences in pixel density. A flexible tiling scheme including first level tiles and second level tiles may be employed to support this functionality. Second level tiles are created by partitioning first level tiles. This tiling scheme allows a first level tile to contain one slice of data at a first resolution and a first level tile containing second level tiles to contain a plurality of slices at a second resolution. Such a flexible tiling scheme allows an encoder/decoder (codec) to support a picture containing multiple resolutions, and hence increases the functionality of both the encoder and decoder. The present disclosure describes a mechanism for integrating tile groups into the flexible tiling scheme. A tile group can contain first level tiles and/or complete sets of second level tiles partitioned from one or more first level tiles. This approach prevents second level tiles from a single first level tile from being split into different tile groups. Accordingly, the disclosed mechanism allows flexible tiling scheme tiles to be included in a tile group, which in turn allows coding tools to be applied to various tiles based on tile group. By preventing second level tiles from a single first level tile from being split into different groups, the complexity of the resulting flexible tiling scheme with tile groups is reduced. As such, the present disclosure further increases the functionality of both the encoder and decoder while reducing processor and/or memory resource usage.

[0005] In an embodiment, the disclosure includes a method implemented in an encoder, the method comprising: partitioning, by a processor of the encoder, a picture into a plurality of first level tiles; partitioning, by the processor, a subset of the first level tiles into a plurality of second level tiles; assigning, by the processor, the first level tiles and the second level tiles into one or more tile groups such that all second level tiles created from a single first level tile are assigned to a same tile group; encoding, by the processor, the first level tiles and the second level tiles into a bitstream; and storing, in a memory of the encoder, the bitstream for communication toward a decoder. Certain streaming applications (e.g., VR and teleconferencing) may be improved if a single image can be sent containing multiple regions

encoded at different resolutions. Some slicing and tiling mechanisms, such as raster scan based slicing and/or tiling, may not support such functionality because tiles at different resolutions may be treated differently. For example, a tile at a first resolution may contain a single slice of data while a tile at a second resolution may carry multiple slices of data due to differences in pixel density. A flexible tiling scheme including first level tiles and second level tiles may be employed to support this functionality. Second level tiles are created by partitioning first level tiles. This tiling scheme allows a first level tile to contain one slice of data at a first resolution and a first level tile containing second level tiles to contain a plurality of slices at a second resolution. Such a flexible tiling scheme allows codec to support a picture containing multiple resolutions, and hence increases the functionality of both the encoder and decoder. The present disclosure describes a mechanism for integrating tile groups into the flexible tiling scheme. A tile group can contain first level tiles and/or complete sets of second level tiles partitioned from one or more first level tiles. This approach prevents second level tiles from a single first level tile from being split into different tile groups. Accordingly, the disclosed mechanism allows flexible tiling scheme tiles to be included in a tile group, which in turn allows coding tools to be applied to various tiles based on tile group. By preventing second level tiles from a single first level tile from being split into different groups, the complexity of the resulting flexible tiling scheme with tile groups is reduced. As such, the present disclosure further increases the functionality of both the encoder and decoder while reducing processor and/or memory resource usage.

[0006] Optionally, in any of the preceding aspects, another implementation of the aspect provides, wherein first level tiles outside the subset contain picture data at a first resolution, and wherein second level tiles contain picture data at a second resolution different from the first resolution.

[0007] Optionally, in any of the preceding aspects, another implementation of the aspect provides, wherein each first level tile in the subset of first level tiles includes two or more complete second level tiles.

[0008] Optionally, in any of the preceding aspects, another implementation of the aspect provides, wherein the first level tiles and the second level tiles are encoded according to a scan order, and wherein encoding according to the scan order includes: encoding the first level tiles in raster scan order; when one of the second level tiles is encountered, pause raster scan order encoding of the first level tiles; and encode all consecutive second level tiles in raster scan order prior to continuing the raster scan order encoding of the first level tiles.

[0009] Optionally, in any of the preceding aspects, another implementation of the aspect provides, wherein all second level tiles partitioned from a current first level tile are encoded prior to encoding any second level tiles partitioned from a subsequent second level tile.

[0010] Optionally, in any of the preceding aspects, another implementation of the aspect provides, wherein each of the one or more tile groups are constrained such that all tiles in an assigned tile group cover a rectangular portion of the picture.

[0011] In an embodiment, the disclosure includes a method implemented in a decoder, the method comprising: receiving, by a processor of the decoder via a receiver, a bitstream including a picture partitioned into a plurality of first level tiles, wherein a subset of the first level tiles are further partitioned into a plurality of second level tiles, and wherein the first level tiles and the second level tiles are assigned to one or more tile groups such that each tile group contains a number of first level tiles, one or more consecutive sequences of second level tiles where each sequence of second level tiles is split from a single first level tile, or combinations thereof; decoding, by the processor, the first level tiles and the second level tiles based on the one or more tile groups; and generating, by the processor, a reconstructed video sequence for display based on the decoded first level tiles and the second level tiles. Certain streaming applications (e.g., VR and teleconferencing) may be improved if a single image can be sent containing multiple regions encoded at different resolutions. Some slicing and tiling mechanisms, such as raster scan based slicing and/or tiling, may not support such functionality because tiles at different resolutions may be treated differently. For example, a tile at a first resolution may contain a single slice of data while a tile at a second resolution may carry multiple slices of data due to differences in pixel density. A flexible tiling scheme including first level tiles and second level tiles may be employed to support this functionality. Second level tiles are created by partitioning first level tiles. This tiling scheme allows a first level tile to contain one slice of data at a first resolution and a first level tile containing second level tiles to contain a plurality of slices at a second resolution. Such a flexible tiling scheme allows codec to support a picture containing multiple resolutions, and hence increases the functionality of both the encoder and decoder. The present disclosure describes a mechanism for integrating tile groups into the flexible tiling scheme. A tile group can contain first level tiles and/or complete sets of second level tiles partitioned from one or more first level tiles. This approach prevents second level tiles from a single first level tile from being split into different tile groups. Accordingly, the disclosed mechanism allows flexible tiling scheme tiles to be included in a tile group, which in turn allows coding tools to be applied to various tiles based on tile group. By preventing second level tiles from a single first level tile from being split into

different groups, the complexity of the resulting flexible tiling scheme with tile groups is reduced. As such, the present disclosure further increases the functionality of both the encoder and decoder while reducing processor and/or memory resource usage.

[0012] In an embodiment, the disclosure includes a method implemented in a decoder, the method comprising: receiving, by a processor of the decoder via a receiver, a bitstream including a picture partitioned into a plurality of first level tiles, wherein a subset of the first level tiles are further partitioned into a plurality of second level tiles, and wherein the first level tiles and the second level tiles are assigned to one or more tile groups such that all second level tiles created from a single first level tile are assigned to a same tile group; decoding, by the processor, the first level tiles and the second level tiles based on the one or more tile groups; and generating, by the processor, a reconstructed video sequence for display based on the decoded first level tiles and the second level tiles. Certain streaming applications (e.g., VR and teleconferencing) may be improved if a single image can be sent containing multiple regions encoded at different resolutions. Some slicing and tiling mechanisms, such as raster scan based slicing and/or tiling, may not support such functionality because tiles at different resolutions may be treated differently. For example, a tile at a first resolution may contain a single slice of data while a tile at a second resolution may carry multiple slices of data due to differences in pixel density. A flexible tiling scheme including first level tiles and second level tiles may be employed to support this functionality. Second level tiles are created by partitioning first level tiles. This tiling scheme allows a first level tile to contain one slice of data at a first resolution and a first level tile containing second level tiles to contain a plurality of slices at a second resolution. Such a flexible tiling scheme allows codec to support a picture containing multiple resolutions, and hence increases the functionality of both the encoder and decoder. The present disclosure describes a mechanism for integrating tile groups into the flexible tiling scheme. A tile group can contain first level tiles and/or complete sets of second level tiles partitioned from one or more first level tiles. This approach prevents second level tiles from a single first level tile from being split into different tile groups. Accordingly, the disclosed mechanism allows flexible tiling scheme tiles to be included in a tile group, which in turn allows coding tools to be applied to various tiles based on tile group. By preventing second level tiles from a single first level tile from being split into different groups, the complexity of the resulting flexible tiling scheme with tile groups is reduced. As such, the present disclosure further increases the functionality of both the encoder and decoder while reducing processor and/or memory resource usage.

[0013] Optionally, in any of the preceding aspects, another implementation of the aspect provides, wherein first level tiles outside the subset contain picture data at a first resolution, and wherein second level tiles contain picture data at a second resolution different from the first resolution.

[0014] Optionally, in any of the preceding aspects, another implementation of the aspect provides, wherein each first level tile in the subset of first level tiles includes two or more complete second level tiles.

[0015] Optionally, in any of the preceding aspects, another implementation of the aspect provides, wherein the first level tiles and the second level tiles are decoded according to a scan order, and wherein decoding according to the scan order includes: decoding the first level tiles in raster scan order; when one of the second level tiles is encountered, pause raster scan order encoding of the first level tiles; and encode all consecutive second level tiles in raster scan order prior to continuing the raster scan order decoding of the first level tiles.

[0016] Optionally, in any of the preceding aspects, another implementation of the aspect provides, wherein all second level tiles partitioned from a current first level tile are decoded prior to decoding any second level tiles partitioned from a subsequent second level tile.

[0017] Optionally, in any of the preceding aspects, another implementation of the aspect provides, wherein each of the one or more tile groups are constrained such that all tiles in an assigned tile group cover a rectangular portion of the picture.

[0018] In an embodiment, the disclosure includes a video coding device comprising: a processor, a receiver coupled to the processor, and a transmitter coupled to the processor, the processor, receiver, and transmitter configured to perform the method of any of the preceding aspects.

[0019] In an embodiment, the disclosure includes a non-transitory computer readable medium comprising a computer program product for use by a video coding device, the computer program product comprising computer executable instructions stored on the non-transitory computer readable medium such that when executed by a processor cause the video coding device to perform the method of any of the preceding aspects.

[0020] In an embodiment, the disclosure includes an encoder comprising: a partitioning means for: partitioning a picture into a plurality of first level tiles; and partitioning a subset of the first level tiles into a plurality of second level tiles; an assigning means for assigning the first level tiles and the second level tiles into one or more tile groups such that all second level tiles created from a single first level tile are assigned to a same tile group; an encoding means

for encoding the first level tiles and the second level tiles into a bitstream; and a storage means for storing the bitstream for communication toward a decoder.

[0021] Optionally, in any of the preceding aspects, another implementation of the aspect provides, wherein the encoder is further configured to perform the method of any of the preceding aspects.

[0022] In an embodiment, the disclosure includes a decoder comprising: a receiving means for receiving a bitstream including a picture partitioned into a plurality of first level tiles, wherein a subset of the first level tiles are further partitioned into a plurality of second level tiles, and wherein the first level tiles and the second level tiles are assigned to one or more tile groups such that all second level tiles created from a single first level tile are assigned to a same tile group; a decoding means for decoding the first level tiles and the second level tiles based on the one or more tile groups; and a generating means for generating a reconstructed video sequence for display based on the decoded first level tiles and the second level tiles.

[0023] Optionally, in any of the preceding aspects, another implementation of the aspect provides, wherein the decoder is further configured to perform the method of any of any of the preceding aspects.

[0024] 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.

[0025] 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

[0026] 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.

[0027] FIG. 1 is a flowchart of an example method of coding a video signal.

[0028] FIG. 2 is a schematic diagram of an example coding and decoding (codec) system for video coding.

[0029] FIG. 3 is a schematic diagram illustrating an example video encoder.

[0030] FIG. 4 is a schematic diagram illustrating an example video decoder.

[0031] FIG. 5 is a schematic diagram illustrating an example bitstream containing an encoded video sequence.

[0032] FIGs. 6A-6E illustrate an example mechanism for creating an extractor track for combining sub-pictures of multiple resolutions from different bitstreams into a single picture for use in virtual reality (VR) applications.

[0033] FIG.7 illustrates an example video conferencing application that splices pictures of multiple resolutions from different bitstreams into a single picture for display.

[0034] FIGs. 8A-8D are schematic diagrams illustrating an example flexible video tiling scheme capable of supporting multiple tiles with different resolutions in the same picture.

[0035] FIG. 9 is a schematic diagram of an example video coding device.

[0036] FIG. 10 is a flowchart of an example method of encoding an image by employing a flexible tiling scheme.

[0037] FIG. 11 is a flowchart of an example method of decoding an image by employing a flexible tiling scheme.

[0038] FIG. 12 is a schematic diagram of an example system for coding a video sequence by employing a flexible tiling scheme.

DETAILED DESCRIPTION

[0039] 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.

[0040] Various acronyms are employed herein, such as 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), raw byte sequence payload (RBSP), sequence parameter set (SPS), versatile video coding (VVC), and working draft (WD).

[0041] 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.

[0042] 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). The joint video experts team (JVET) of ITU-T and ISO/IEC has begun developing a video coding standard referred to as Versatile Video Coding (VVC). VVC is included in a Working Draft (WD), which includes JVET-L1001-v5.

[0043] In order to code a video image, the image is first partitioned, and the partitions are coded into a bitstream. Various picture partitioning schemes are available. For example, an image can be partitioned into regular slices, dependent slices, tiles, and/or according to Wavefront Parallel Processing (WPP). For simplicity, HEVC restricts encoders so that only regular slices, dependent slices, tiles, WPP, and combinations thereof can be used when partitioning a slice into groups of CTBs for video coding. Such partitioning can be applied to support Maximum Transfer Unit (MTU) size matching, parallel processing, and reduced end-to-end delay. MTU denotes the maximum amount of data that can be transmitted in a single packet. If a packet payload is in excess of the MTU, that payload is split into two packets through a process called fragmentation.

[0044] A regular slice, also referred to simply as a slice, is a partitioned portion of an image that can be reconstructed independently from other regular slices within the same picture, notwithstanding some interdependencies due to loop filtering operations. Each regular slice is encapsulated in its own Network Abstraction Layer (NAL) unit for transmission. Further, in-picture prediction (intra sample prediction, motion information prediction, coding mode prediction) and entropy coding dependency across slice boundaries may be disabled to support independent reconstruction. Such independent reconstruction supports parallelization. For example, regular slice based parallelization employs minimal inter-processor or inter-core communication. However, as each regular slice is independent, each slice is associated with a separate slice header. The use of regular slices can incur a substantial coding overhead due to the bit cost of the slice header for each slice and due to the lack of prediction across the slice boundaries. Further, regular slices may be employed to support matching for MTU size requirements. Specifically, as a regular slice is encapsulated in a separate NAL unit and can be independently coded, each regular slice should be smaller than the MTU in MTU schemes to avoid breaking the slice into multiple packets. As such, the goal of parallelization and the goal of MTU size matching may place contradicting demands to a slice layout in a picture.

[0045] Dependent slices are similar to regular slices, but have shortened slice headers and allow partitioning of the image treeblock boundaries without breaking in-picture prediction. Accordingly, dependent slices allow a regular slice to be fragmented into multiple NAL units, which provides 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 complete.

[0046] A tile is a partitioned portion of an image created by horizontal and vertical boundaries that create columns and rows of tiles. Tiles may be coded in raster scan order (right to left and top to bottom). The scan order of CTBs is local within a tile. Accordingly, CTBs in

a first tile are coded in raster scan order, before proceeding to the CTBs in the next tile. Similar to regular slices, tiles break in-picture prediction dependencies as well as entropy decoding dependencies. However, tiles may not be included into individual NAL units, and hence tiles may not be used for MTU size matching. Each tile can be processed by one processor/core, and the inter-processor/inter-core communication employed for in-picture prediction between processing units decoding neighboring tiles may be limited to conveying a shared slice header (when adjacent tiles are in the same slice), and performing loop filtering related sharing of reconstructed samples and metadata. When more than one tile is included in a slice, the entry point byte offset for each tile other than the first entry point offset in the slice may be signaled in the slice header. For each slice and tile, at least one of the following conditions should be fulfilled: 1) all coded treeblocks in a slice belong to the same tile; and 2) all coded treeblocks in a tile belong to the same slice.

[0047] In WPP, the image is partitioned into single rows of CTBs. Entropy decoding and prediction mechanisms may use data from CTBs in other rows. Parallel processing is made possible through parallel decoding of CTB rows. For example, a current row may be decoded in parallel with a preceding row. However, decoding of the current row is delayed from the decoding process of the preceding rows by two CTBs. This delay ensures that data related to the CTB above and the CTB above and to the right of the current CTB in the current row is available before the current CTB is coded. This approach appears as a wavefront when represented graphically. This staggered start allows for parallelization with up to as many processors/cores as the image contains CTB rows. Because in-picture prediction between neighboring treeblock rows within a picture is permitted, the inter-processor/inter-core communication to enable in-picture prediction can be substantial. The WPP partitioning does consider NAL unit sizes. Hence, WPP does not support MTU size matching. However, regular slices can be used in conjunction with WPP, with certain coding overhead, to implement MTU size matching as desired.

[0048] Tiles may also include motion constrained tile sets. A motion constrained tile set (MCTS) is a tile set designed such that associated 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. Further, 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. Temporal MCTSs supplemental enhancement information (SEI) messages may be used to indicate the existence of MCTSs in the bitstream and signal the

MCTSs. The MCTSs 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 a MCTS. The information includes a number of extraction information sets, each defining a number of MCTSs and containing raw bytes sequence payload (RBSP) bytes of the replacement video parameter sets (VPSs), sequence parameter sets (SPSs), and picture parameter sets (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) may be rewritten or replaced, and slice headers may be updated because one or all of the slice address related syntax elements (including first_slice_segment_in_pic_flag and slice_segment_address) may employ different values in the extracted sub-bitstream.

[0049] The various tiling schemes may be employed when partitioning a picture for further encoding. As a particular example, tiles can be assigned to tile groups, which can take the place of slices in some examples. In some examples, each tile group can be extracted independently of other tile groups. Accordingly, tile grouping may support parallelization by allowing each tile group to be assigned to a different processor. Tile groups may be assigned in raster scan order or can be constrained to form a rectangular shape of area within a picture. Explicit tile identifier (ID) signaling may be used to support such tile groups. In some systems, a tile ID is always assigned to be the same as the tile index. Explicit tile ID signaling allows a tile ID to be different from a tile index. Having explicit tile ID signalling supports extraction of MCTS from a bitstream without having to update tile group header. It should be noted that signaling of explicit tile IDs and corresponding usage as the address of a tile group may be specific to HEVC style tile structure definitions and signaling. If a tile structure definition and/or signaling are modified, the signaling of tile ID according to explicit tile ID mechanisms may be incorrect and/or inapplicable in some examples. Tile grouping and explicit tile ID signaling can be employed, for example, in cases where a decoder may not wish to decode an entire image. As a particular example, video coding schemes may be employed to support virtual reality (VR) video, which may be encoded according to Omnidirectional Media Application Format (OMAF).

[0050] In VR video, one or more cameras may record the environment around the camera(s). A user can then view the VR video as if the user were present in the same location as the camera. In VR video, a picture encompasses an entire environment around the user. The user then views a sub-portion of the picture. For example, a user may employ a head mounted

display that changes the sub-portion of the picture displayed based on the head movements of the user. The portion of the video being displayed may be referred to as a viewport.

[0051] Accordingly, a distinct feature of omnidirectional video is that only a viewport is displayed at any particular time. This is in contrast to other video applications that may display an entire video. This feature may be utilized to improve the performance of omnidirectional video systems, for example through selective delivery depending on the user's viewport (or any other criteria, such as recommended viewport timed metadata). Viewport-dependent delivery may be enabled, for example, by employing region-wise packing and/or viewport-dependent video coding. The performance improvement may result in lower transmission bandwidth, lower decoding complexity, or both when compared to other omnidirectional video systems when employing the same video resolution/quality.

[0052] An example viewport-dependent operation is an MCTS-based approach for achieving five thousand sample (e.g., 5120×2560 luma samples) resolution (5K) effective equirectangle projection (ERP) resolution with HEVC-based viewport-dependent OMAF video profile. This approach is described in greater detail below. But generally, this approach partitions VR video into tile groups and encodes the video at a plurality of resolutions. The decoder can indicate the viewport currently used by the user during streaming. The video server providing the VR video data can then forward the tile group(s) associated with the viewport at high resolution and forward non-viewed tile groups at lower resolution. This allows the user to view the VR video at a high resolution without requiring the entire picture be sent at high resolution. The non-viewed sub-portions are discarded, and hence the user may be unaware of the lower resolutions. However, the lower resolution tile groups may be displayed to the user if the user changes viewports. The resolution of the new viewport may then be increased as the video proceeds. In order to implement such a system, pictures should be created that contain both the higher resolution tile groups and the lower resolution tile groups.

[0053] In another example, video conferencing applications may be designed to forward pictures including multiple resolutions. For example, a video conference may contain multiple participants. The participant currently speaking may be displayed at a higher resolution and other participants may be displayed at lower resolutions. In order to implement such a system, pictures should be created that contain both the higher resolution tile groups and the lower resolution tile groups.

[0054] Disclosed herein are various flexible tiling mechanisms to support creating a picture with sub-pictures coded at multiple resolutions. For example, a video can be coded at a plurality of resolutions. The video can also be coded by employing sub-pictures at each

resolution. The lower resolution sub-pictures are smaller than the higher resolution sub-pictures. In order to create a picture with multiple resolutions, the picture can be partitioned into first level tiles. The sub-pictures from the highest resolution can be included directly into the first level tiles. Further, the first level tiles can be partitioned into second level tiles that are smaller than the first level tiles. Accordingly, the smaller second level tiles can directly accept the lower resolution sub-pictures. In this way, the slices from each resolution can be compressed into a single picture via a tile index relationship without requiring that different resolution tiles be dynamically readdressed to use a consistent addressing scheme. The first level tiles and second level tiles may be implemented as MCTSs, and hence may accept motion constrained image data at different resolutions. The present disclosure includes many aspects. As a particular example, the first level tiles are split into second level tiles. The first level tiles and second level tiles can then be included in tile groups. The tile groups may be constrained to include an integer number of first level tiles and/or one or more consecutive sequences of second level tiles where each sequence of second level tiles are split from a single first level tile. This approach may ensure that all second level tiles created from a single first level tile are assigned to the same tile group. In another particular example, a scan order is described for coding the flexible tiling scheme. In this example, the first level tiles are coded in raster scan order relative to the picture and/or tile group boundaries. When a second level tile is encountered, the first level tile scanning is paused. A consecutive sequence of second level tiles are then scanned in raster scan order relative to the first level tile from which such second level tiles were partitioned. The scan order then proceeds to the next consecutive sequence of second level tiles, if any. Otherwise, the first level tile scanning is continued. This process continues until the tile group and/or picture is encoded or decoded, depending on the example.

[0055] 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.

[0056] 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.

[0057] 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.

[0058] 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.

[0059] 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.

[0060] 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.

[0061] 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.

[0062] 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.

[0063] 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.

[0064] 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.

[0065] 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.

[0066] 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.

[0067] 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.

[0068] 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.

[0069] 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).

[0070] 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.

[0071] 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.

[0072] 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.

[0073] 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).

[0074] 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.

[0075] 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.

[0076] 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.

[0077] 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.

[0078] 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.

[0079] 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.

[0080] 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.

[0081] 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.

[0082] 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.

[0083] 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.

[0084] 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.

[0085] 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.

[0086] 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.

[0087] The bitstream 500 includes a sequence parameter set (SPS) 510, a plurality of picture parameter sets (PPSs) 512, tile group 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 tiles in corresponding pictures, quantization parameters, offsets, picture specific coding tool parameters (e.g., filter controls), etc. The tile group header 514 contains parameters that are specific to each tile group in a picture. Hence, there may be one tile group header 514 per tile group in the video sequence. The tile group header 514 may contain tile group 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 tile group header 514 as a slice header, and use such information to support slices instead of tile groups.

[0088] 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 is divided into tiles 523. The tiles 523 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. An image/picture can contain one or more tiles 523.

[0089] A tile 523 is a partitioned portion of a picture created by horizontal and vertical boundaries. Tiles 523 may be rectangular and/or square. Specifically, a tile 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 tile 523 may be any rectangular shape, where a square is a special case of a rectangle where all four sides are of equal length. A picture may be petitioned into rows and columns of tiles 523. A tile row is a set of tiles 523 positioned in a horizontally adjacent manner to create a continuous line from the left boundary to the right boundary of a picture (or vice versa). A tile column is a set of tiles 523 positioned in a vertically adjacent manner to create a continuous line from the top boundary to the bottom boundary of the picture (or vice versa). Tiles 523 may or may not allow prediction based on other tiles 523, depending on the example. Each tile 523 may have a unique tile index in the picture. A tile index is a procedurally selected numerical identifier that can be used to distinguish one tile 523 from another. For example, tile indices may increase numerically in raster scan order. Raster scan order is left to right and top to bottom. It should be noted that, in some examples, tiles 523 may also be assigned tile identifiers (IDs). A tile ID is an assigned identifier that can be used to distinguish one tile 523 from another. Computations may employ tile IDs instead of tile indices in some examples. Further, tile IDs can be assigned to have the same values as the tile indices in some examples. Tile indices and/or IDs may be signaled to indicate tile groups containing the tiles 523. For example, the tile indices and/or IDs may be employed to map picture data associated with a tile 523 to a proper position for display. A tile group is a related set of tiles 523 that can be separately extracted and coded, for example to support display of a region of interest and/or to support parallel processing. Tiles 523 in a tile group can be coded without reference to tiles 523 outside of the tile group. Each tile 523 may be assigned to a corresponding tile group, and therefore a picture can contain a plurality of tile groups.

[0090] FIGs. 6A-6E illustrate an example mechanism 600 for creating an extractor track 610 for combining sub-pictures of multiple resolutions from different bitstreams into a single picture for use in virtual reality (VR) applications. Mechanism 600 may be employed to support an example use case of method 100. For example, mechanism 600 can be employed to generate a bitstream 500 for transmission from a codec system 200 and/or an encoder 300 toward a codec system 200 and/or a decoder 400. As a specific example, mechanism 600 can be employed for use in conjunction with VR, OMAF, three hundred sixty degree video, etc.

[0091] In VR, only a portion of video is displayed to a user. For example, VR video may be filmed to include a sphere surrounding a user. The user may employ a head mounted

display (HMD) to view the VR video. The user may point the HMD toward a region of interest. The region of interest is displayed to the user and other video data is discarded. In this way, a user views only a user selected portion of the VR video at any instant. This approach mimics the user's perceptions, and hence causes the user to experience a virtual environment in a manner that mimics a real environment. One of the issues with this approach is that the entire VR video may be transmitted to the user, but only a current viewport of the video is actually used and the remainder is discarded. In order to increase signaling efficiency for streaming applications, the user's current viewport can be transmitted at a higher first resolution and other viewports can be transmitted at a lower second resolution. In this way, the viewports that are likely to be discarded take up less bandwidth than the viewport(s) likely to be viewed by the user. In the event the user selects a new viewport, the lower resolution content can be shown until the decoder can request that a different current viewport be transmitted at the higher first resolution. Mechanism 600 can be employed to create an extractor track 610, as shown in FIG. 6E, to support this functionality. An extractor track 610 is a track of image data that encapsulates a picture at multiple resolutions for use as described above.

[0092] The mechanism 600 encodes the same video content at a first resolution 611 and a second resolution 612, as shown in FIGs. 6A and 6B, respectively. As a specific example, the first resolution 611 may be 5120×2560 luma samples and the second resolution 612 may be 2560×1280 luma samples. The pictures of the video may be partitioned into tiles 601 at the first resolution 611 and tiles 603 at the second resolution 612, respectively. In the example shown, the tiles 601 and 603 are each partitioned into a 4x2 grid. Further, an MCTS can be coded for each tile 601 and 603 position. The pictures at the first resolution 611 and the second resolution 612 each result in an MCTS sequence describing video over time at a corresponding resolution. Each coded MCTS sequence is stored as a sub-picture track or a tile track. Mechanism 600 can then use the pictures to create segments to support viewport adaptive MCTS selection. For example, each range of viewing orientations that causes a different selection of high- and low-resolution MCTSs is considered. In the illustrated example, four tiles 601 containing MCTSs at the first resolution 611 and four tiles 603 containing MCTSs at the second resolution 612 are obtained.

[0093] Mechanism 600 can then create an extractor track 610 for each possible viewport-adaptive MCTS selection. FIGs. 6C and 6D illustrate an example viewport-adaptive MCTS selection. Specifically, a set of selected tiles 605 and 607 are selected at the first resolution 611 and the second resolution 612, respectively. The selected tiles 605 and 607 are illustrated in gray shading. In the example shown, the selected tiles 605 are the tiles 601 at the first

resolution 611 that are to be shown to the user and the selected tiles 607 are the tiles 603 at the second resolution 612 that are likely to be discarded but maintained to support display in the event the user selects a new viewport. The selected tiles 605 and 607 are then combined into a single picture containing image data at both the first resolution 611 and the second resolution 612. Such pictures are combined to create an extractor track 610. FIG. 6E illustrates a single picture from a corresponding extractor track 610 for purpose of illustration. As shown, the picture in the extractor track 610 contains the selected tiles 605 and 607 from the first resolution 611 and second resolution 612. As noted above, FIGs. 6C-6E illustrate a single viewport-adaptive MCTS selection. In order to allow for user selection of any viewport, an extractor track 610 should be created for each possible combination of selected tiles 605 and 607.

[0094] In the example shown, each selection of tiles 603 encapsulating content from the second resolution 612 bitstream contains two slices. A RegionWisePackingBox may be included in the extractor track 610 to create a mapping between the packed picture and a projected picture of the equirectangle projection (ERP) format. In the presented example, the bitstreams resolved from the extractor tracks 610 have resolution 3200×2560. Consequently, a four thousand sample (4K)-capable decoder may decode content where the viewport is extracted from a coded bitstream with five thousand sample 5K (5120×2560) resolution.

[0095] As shown, the extractor track 610 contains two rows of high resolution tiles 601 and four rows of low resolution tiles 603. Accordingly, the extractor track 610 contains two slices of high resolution content and four slices of low resolution content. Uniform tiling may not support such a use case. Uniform tiling is defined by a set of tile columns and a set of tile rows. Tile columns extend from the top of a picture to the bottom of the picture. Likewise, tile rows extend from the left of the picture to the right of the picture. While such structure can be simply defined, this structure cannot effectively support advance use cases such as the use case described by mechanism 600. In the example shown, different numbers of rows are employed at different sections of the extractor track 610. If uniform tiling is employed, the tiles at the right side of the extractor track 610 should be rewritten to accept two slices each. This approach is inefficient and computationally complex.

[0096] The present disclosure includes a flexible tiling scheme, as described below, that does not require tiles be rewritten to include different numbers of slices. The flexible tiling scheme allows a tile 601 to contain content at a first resolution 611. The flexible tiling scheme also allows a tile 601 to be partitioned into smaller tiles that can each be directly mapped to tiles 603 at a second resolution 612. This direct mapping is more efficient as such an approach

does not require that the tiles be rewritten/readdressed when different resolutions are combined as described above.

[0097] FIG. 7 illustrates an example video conferencing application 700 that splices pictures of multiple resolutions from different bitstreams into a single picture for display. Application 700 may be employed to support an example use case of method 100. For example, application 700 can be employed at a codec system 200 and/or a decoder 400 to display video content from bitstream 500 from a codec system 200 and/or an encoder 300. The video conferencing application 700 displays a video sequence to a user. The video sequence contains pictures displaying a speaking participant 701 and other participants 703. The speaking participant 701 is displayed at a higher/larger first resolution and the other participants 703 are displayed at a lower/smaller second resolution. In order to code such a picture, the picture should contain a portion with a single row and a portion with three rows. To support such a scenario with uniform tiling, the picture is partitioned into a left and a right tile. The right tile is then rewritten/readdressed to include three rows. Such readdressing results in both a compression and a performance penalty. The flexible tiling scheme described below allows a single tile to be partitioned into smaller tiles and mapped to tiles in sub-picture bitstreams associated with the other participants 703. In this way, the speaking participant 701 can be directly mapped into a first level tile and the other participants 703 can be mapped to second level tiles split from the first tile without such rewriting/readdressing.

[0098] FIGs. 8A-8D are schematic diagrams illustrating an example flexible video tiling scheme 800 capable of supporting multiple tiles with different resolutions in the same picture. The flexible video tiling scheme 800 can be employed to support a more efficient coding mechanism 600 and application 700. Accordingly, the flexible video tiling scheme 800 can be employed as part of method 100. Further, the flexible video tiling scheme 800 can be employed by a codec system 200, an encoder 300, and/or a decoder 400. The results of the flexible video tiling scheme 800 can be stored in a bitstream 500 for transmission between the encoder and the decoder.

[0099] As shown in FIG. 8A, a picture (e.g., frame, image, etc.) can be partitioned into first level tiles 801, also known as level one tiles. As shown in FIG. 8B, the first level tiles 801 can be selectively partitioned to create second level tiles 803, also known as level two tiles. The first level tiles 801 and second level tiles 803 can then be employed to create a picture with sub-pictures coded at multiple resolutions. A first level tile 801 is a tile generated by completely partitioning a picture into a set of columns and a set of rows. A second level tile 803 is a tile generated by partitioning a first level tile 801.

[00100] As described above, in various scenarios a video can be coded at a plurality of resolutions, for example in VR and/or teleconferencing. The video can also be coded by employing slices at each resolution. The lower resolution slices are smaller than the higher resolution slices. In order to create a picture with multiple resolutions, the picture can be partitioned into first level tiles 801. The slices from the highest resolution can be included directly into the first level tiles 801. Further, the first level tiles 801 can be partitioned into second level tiles 803 that are smaller than the first level tiles 801. Accordingly, the smaller second level tiles 803 can directly accept the lower resolution slices. In this way, the slices from each resolution can be compressed into a single picture, for example via a tile index relationship, without requiring that different resolution tiles be dynamically readdressed to use a consistent addressing scheme. The first level tiles 801 and second level tiles 803 may be implemented as MCTSs, and hence may accept motion constrained image data at different resolutions.

[00101] The present disclosure includes many aspects. As a particular example, the first level tiles 801 are split into second level tiles 803. The second level tiles 803 may then be constrained to each contain a single rectangular slice of picture data (e.g., at the lower resolution). A rectangular slice is a slice constrained to maintain a rectangular shape, and is hence coded based on horizontal and vertical picture boundaries. Accordingly, a rectangular slice is not coded based on a raster scan group (which contains CTUs in a line from left to right and top to bottom and may not maintain a rectangular shape). A slice is a spatially distinct region of a picture/frame that is encoded separately from any other region in the same frame/picture. In another example, first level tile 801 can be split into two or more complete second level tiles 803. In such a case, a first level tile 801 may not contain a partial second level tile 803. In another example, a configuration of the first level tiles 801 and second level tiles 803 can be signaled in a parameter set in a bitstream, such as a PPS associated with a picture partitioned to create the tiles. In one example, a split indication, such as a flag, can be coded in a parameter set for each first level tile 801. The indication denotes which first level tiles 801 are further split into second level tiles 803. In another example, the configuration of second level tiles 803 can be signaled as a number of second level tile columns and a number of second level tile rows.

[00102] In another example, the first level tiles 801 and the second level tiles 803 can be assigned into tile groups. Such tile groups can be constrained so that all tiles in a corresponding tile group are constrained to cover a rectangular region of the picture (e.g., in contrast to raster scan). For example, some systems may add tiles to a tile group in raster

scan order. This includes adding an initial tile in a current row, proceeding to add each tile in the row until the left picture boundary of the current row is reached, proceeding to the right boundary of the next row, and adding each tile in the next row, etc. until a final tile is reached. This approach may result in non-rectangular shapes that extend across the picture. Such shapes may not be useful for creating pictures with multiple resolutions as described herein. Instead, the present example may constrain tile groups such that any first level tile 801 and/or second level tile 803 may be added to the tile group (e.g., in any order), but the resulting tile group must be a rectangle or square (e.g., include four sides connected at right angles). This constraint may ensure that second level tiles 803 partitioned from a single first level tile 801 are not placed in different tile groups.

[00103] In another example, data explicitly indicating a number of second level tile columns and a number of second level tile rows can be omitted from a bitstream when a first level tile width is less than twice a minimum width threshold and a first level tile height is less than twice a minimum height threshold. This is because a first level tile 801 meeting such conditions may not be split into more than one column or one row, respectively, and hence such information can be inferred by the decoder. In another example, split indications indicating which first level tiles 801 are partitioned into second level tiles 803 can be omitted from the bitstream for certain first level tiles 801. For example, such data can be omitted when the first level tile 801 has a first level tile width that is less than a minimum width threshold and a first level tile height is less than a minimum height threshold. This is because a first level tile 801 meeting such conditions is too small to be split into second level tiles 803, and hence such information can be inferred by the decoder.

[00104] In another aspect, the flexible video tiling scheme 800 may employ tile groups 805 as shown in FIG. 8C. The tile groups 805 are depicted as bounded by bold lines. The picture is partitioned into first level tiles 801. A subset of the first level tiles 801 are partitioned into second level tiles 803. The first level tiles 801 and the second level tiles 803 can then be assigned to tile groups 805. A tile group 805 is a related set of tiles that can be separately extracted and coded, for example to support display of a region of interest and/or to support parallel processing. A tile group 805 may be generated as a raster scan tile group or a rectangular tile group. A raster scan tile group contains tiles assigned in raster scan order, which proceeds from left to right and top to bottom. For example, raster scan order proceeds from a first tile towards a right picture boundary, then proceeds from a left picture boundary at a next row toward the right picture boundary, etc. until a last tile is reached. In contrast, a

rectangular tile group contains a rectangular sub-portion of the picture. The tile groups 805 are rectangular tile groups, but raster scan tile groups may also be used in some examples.

[00105] In some aspects, the first level tiles 801 and the second level tiles 803 can be assigned into tile groups 805 so that each tile group 805 contains a number of first level tiles 801 or one or more consecutive sequences of second level tiles 803. As used herein, a sequence of second level tiles 803 are a group of second level tiles 803 split from a single first level tile 801, and hence have consecutive tile indices. This approach ensures that all second level tiles 803 created from a single first level tile 801 are assigned to the same tile group 805. In the example depicted in FIG. 8C, one tile group 805 contains four first level tiles 801 and four other tile groups 805 each contain second level tiles 803 split from a single first level tile 801. However, many combinations of tiles and tile groups 805 may be used depending on the type of video to be encoded and decoded.

[00106] In another aspect, the flexible video tiling scheme 800 may employ a scan order 807 as shown in FIG. 8D. A scan order 807 is an order that tiles are encoded at an encoder and/or decoded at a decoder or a hypothetical reference decoder (at an encoder), depending on the example. In the scan order 807 shown, the first level tiles 801 are coded in raster scan order. When one of the second level tiles 803 is encountered, the raster scan order coding of the first level tiles 801 is paused. All consecutive second level tiles 803 are then coded in raster scan order prior to continuing the raster scan order encoding of the first level tiles 801. This process continues until all tiles are coded. In the example shown, a second level tile 803 labeled tile 1 is encountered first. Accordingly, raster scan order coding of the first level tiles 801 is paused and consecutive second level tiles 803 labeled tile 1 and 2 are coded. When all consecutive second level tiles 803 are coded, the raster scan order coding of the first level tiles 801 continues. Accordingly, first level tiles 801 labeled tile 3 and 4 are then coded. A second level tile 803 labeled 5 is encountered. The consecutive second level tiles 803 are then coded. As such, the tiles labeled 5-8 are coded. When all consecutive second level tiles 803 are coded, the raster scan order coding of the first level tiles 801 continues again. This results in coding first level tiles 801 labeled tile 9 and 10. A second level tile 803 labeled 11 is encountered. The consecutive second level tiles 803 are then coded. As such, the tiles labeled 11 and 12 are coded. In more formal terms, the first level tiles 801 are coded in raster scan order relative to boundaries of the picture and/or tile group. Further, all second level tiles 803 partitioned from a current first level tile 801 (e.g., a sequence of consecutive second level tiles 803) are coded prior to encoding any second level tiles 803 partitioned from a subsequent second level tile 803. In addition, all second level tiles 803 partitioned from the

current first level tile 801 are encoded in raster scan order relative to boundaries of the current first level tile 801.

[00107] As described above, a flexible video tiling scheme 800 supports merging sub-pictures from different bitstreams into a picture containing multiple resolutions. The following describes various embodiments that support such functionalities. In general, this disclosure describes methods for signaling and coding of tiles in video coding that partition pictures in a manner that is more flexible that the tiling scheme in the HEVC. More specifically, this disclosure describes some tiling schemes wherein tile columns may not uniformly extend from top to bottom of a coded picture and likewise tile rows may not uniformly extend from left to right of a coded picture.

[00108] For example, based on an HEVC tiling approach, some tiles should be further split into multiple tile rows to support the functionality described in FIGs. 6A-6E and 7. Further, depending on how tiles are positioned, a tile should be further split into tile columns. For example, in FIG. 7 the participants two through four may positioned below participant one in some cases, which could be supported by splitting a tile into columns. To satisfy these scenarios, a first level tile may be split into tile rows and tile columns of second level tiles as described below.

[00109] For example, tile structure can be relaxed as follows. Tiles in the same picture are not required to be a particular number of tile rows. Further, tiles in the same picture are not required to be a particular number of tile columns. For signaling of flexible tiles, the following steps may be used. A first level tile structure may be defined by tile columns and tile rows as defined in HEVC. The tile columns and tile rows may be uniform or not uniform in size. Each of these tiles may be referred to as a first level tile. A flag may be signaled to specify whether or not each first level tile is further divided into one or more tile columns and one or more tile rows. If a first level tile is further divided, the tile columns and tile rows may be either uniform or not uniform in size. New tiles resulting from the division of first level tiles are referred to as second level tiles. The flexible tile structure may be limited to second level tiles only, and hence no further split of any second level tile is allowed in some examples. In other examples, further division of second level tiles can be applied to create subsequent level tiles in a manner similar to the creation of second level tiles from the first level tiles.

[00110] In the preceding examples, a picture may contain zero or more first level tiles that are not further split and zero or more second level tiles. A first level tile that has been further split may only conceptually exist and may not be counted in the total number of tiles in the

picture. An example scan order is specified as follows. For simplicity, a tile group may be constrained to contain either a number of complete first level tiles or a complete subset of a first level tile. First level tiles may be ordered according to a tile raster scan of a picture. When a first level tile that has been further split is referenced, the set of second level tiles resulting from such a split may be collectively referenced. The second level tiles of any current first level tile may be referenced prior to referencing any second level tiles of a subsequent first level tile that succeeds the current first level tile. The second level tiles of a current first level tile are referenced in raster scan order within the current first level tile. CTUs within any current tile may be referenced in CTU raster scan order within the current tile.

[00111] For simplicity, when a first level tile is divided into two or more second level tiles, the division may always use uniform size tile columns and uniform tile rows. In such examples, there may be no need to signal a flag that specifies whether or not level two tile columns and level two tile rows are uniform. Further, there may be no need for syntax elements specifying tile row height and tile row width. In some examples, the first level tiles may be constrained to always use a uniform size for tile columns and tile rows. In such examples, there may be no need to signal a flag that specifies whether or not level one tile columns and level one rows are uniform. Further, there may be no need for syntax elements specifying tile row height and tile row width. In another example, the first level tiles and second level tiles may be constrained to always use uniform size tile columns and tile rows. In such examples, there may be no need to signal a flag that specifies whether or not level one and level two tile columns and rows are uniform. Further, there may be no need for syntax elements specifying tile row height and tile row width.

[00112] Derivation of tile locations, sizes, indices, and scanning order of flexible tiles defined by this approach is described below. In some examples, when such flexible tile structure is used, a tile group may contain only a number of complete first level tiles or only a consecutive sequence of complete second level tiles of one single first level tile. Further, when such flexible tile structure is used, a tile group may be constrained to include of one or more complete first level tiles. In this example, when a tile group contains a second level tile, all second level tiles that originated from the division of the same first level tile should be contained in the tile group. It can further be constrained that when such flexible tile structure is used, a tile group contains one or more tiles and together all tiles belong to a tile group that covers a rectangular region of a picture. In another aspect, when such flexible tile structure is

used, a tile group contains one or more first level tiles and together all tiles belong to a tile group that covers a rectangular region of a picture.

[00113] In an example, signaling of flexible tiles can be as follows. A minimum tile width and minimum tile height are defined values. A first level tile structure can be defined by tile columns and tile rows. The tile columns and tile rows may be uniform or non-uniform in size. Each of these tiles can be referred to as a first level tile. A flag may be signaled to specify whether any of the first level tiles may be further divided. This flag may not be present when the width of each first level tile is not greater than twice of the minimum tile width, and the height of each first level tile is not greater than twice of the minimum tile height. When not present, the value of the flag is inferred to be equal to zero.

[00114] In an example, the following applies for each first level tile. A flag can be signaled to specify whether or not a first level tile is further divided into one or more tile columns and one or more tile rows. The presence of the flag can be constrained as follows. If the first level tile width is greater than the minimum tile width or if the first level tile height is greater than the minimum tile height, the flag is present / signaled. Otherwise, the flag is not present and the value of the flag is inferred to be equal to zero indicating the first level tile is not further divided.

[00115] If a first level tile is further divided, the number of tile columns and the number of tile rows for this division may further be signaled. The tile columns and tile rows may be either uniform or not uniform in size. Tiles resulting from the division of first level tiles are referred to as second level tiles. The presence of the number of tile columns and the number of tile rows can be constrained as follows. When the first level tile width is less than twice of the minimum tile width, the number of tile columns may not be signaled and the number of tile columns value can be inferred to be equal to one. The signaling may employ a _minus1 syntax element so that the signaled syntax element value may be zero and the number of tile columns is the value of the syntax element plus one. This approach may further compress the signaling data. When the first level tile height is less than twice of the minimum tile height, the number of tile rows may not be signaled and the value of the number of tile rows can be inferred to be equal to zero. The signaled syntax element value may be zero and the number of tile rows can be the value of the syntax element plus one to further compress the signaling data. Tiles resulting from the division of first level tiles can be referred to as second level tiles. The flexible tile structure may be limited to second level tiles only, so that no further split of any second level tile is allowed. In other examples, further division of second level tiles can be applied in a similar manner to splitting a first level tile into second level tiles.

[00116] In an example, signaling of a flexible tile structure can be as follows. When a picture contains more than one tile, a signal, such as a flag, can be employed in a parameter set that is directly or indirectly referred to by a corresponding tile group. The flag can specify whether a corresponding tile structure is a uniform tile structure or non-uniform tile structure (e.g., a flexible tile structure as described herein). The flag may be called uniform_tile_structure_flag. When uniform_tile_structure_flag is equal to one, signaling of a HEVC-style uniform tile structure is employed, for example by signaling num_tile_columns_minus1 and num_tile_rows_minus1 to indicate a single level of uniform tiles. When uniform_tile_structure_flag is equal to zero, the following information may also be signaled. The number of tiles in a picture can be signaled by the syntax element num_tiles_minus2, which indicates the number of tiles in the picture (NumTilesInPic) is equal to num_tiles_minus2 + 2. This may result in bit savings during signaling since a picture may be considered to be a tile by default. For each tile, excluding the last one, the addresses of the first coding block (e.g., CTU) and last coding block of the tile are signaled. The address of a coding block can be the index of the block in a picture (e.g., index of CTU in picture). The syntax elements for such coding blocks may be tile_first_block_address[i] and tile_last_block_address[i]. These syntax elements may be coded as ue(v) or u(v). When the syntax elements are coded as u(v), the number of bits used for representing each of the syntax element is ceil(log2(max number of the coding block in a picture)). The addresses of the first and last coding block of the last tile may not be signaled, and may instead be derived based on picture size in luma samples and the aggregation of all other tiles in the picture.

[00117] In an example, for each tile, excluding the last one, instead of signaling the addresses of the first and the last coding block of the tile, the address of the first coding block of the tile, and the width and the height of the tile may be signaled. In another example, for each tile, excluding the last one, instead of signaling the addresses of the first and the last coding block of the tile, the offset of the top left point of the tile relative to the original of the picture (e.g., the top left of the picture) and the width and the height of the tile may be signaled. In yet another example, for each tile, excluding the last one, instead of signaling the addresses of the first and the last coding block of the tile, the following information can be signaled. The width and the height of the tile may be signaled. Also, the location of each tile may not be signaled. Instead a flag may be signaled to specify whether to position the tile immediately to the right or immediately below the previous tile. This flag may be not present if the tile can only be to the right or can only be to the below the previous tile. The top left

offset of the first tile may always be set to be the origin / top left of the picture (e.g., x = zero and y = zero).

[00118] For signaling efficiency, a set of unique tile sizes (e.g., width and height) may be signaled. This list of unique tile sizes may be referred to by index from the loop that includes signaling of each tile size. In some examples, the tile locations and sizes as derived from the signaled tile structure shall constrain the partition to ensure no gap and no overlap occurs between any tiles.

[00119] The following constraints may also apply. Tile shapes may be required to be rectangular (e.g., not raster scan shapes). The unit of tiles in a picture shall cover the picture without any gap and any overlapping among the tiles. When decoding is done with one core only, for coding of a current coding block (e.g., CTU) that are not at the left border of a picture, the left neighbor coding block shall be decoded before the current coding block. When decoding is done with one core only, for coding of a current coding block (e.g., CTU) that are not at the top border of a picture, a top neighbor coding block shall be decoded before the current coding block. When two tiles have tile indices that are next to each other (e.g., idx three and idx four) one of the following is true. The two tiles share vertical border and/or when the first tile has top left location at (Xa, Ya) with size (Wa and Ha representing its width and height) and when the second tile has top left location at (Xb, Yb) then Yb = Ya + Ha.

[00120] The following constraints may also apply. When a tile has more than one left neighbor tile, the height of the tile shall be equal to the sum of heights of all its left neighbor tiles. When a tile has more than one right neighbor tile, the height of the tile shall be equal to the sum of heights of all its left neighbor tiles. When a tile has more than one top neighbor tile, the width of the tile shall be equal to the sum of widths of all its top neighbor tiles. When a tile has more than one bottom neighbor tile, the width of the tile shall be equal to the sum of widths of all its bottom neighbor tiles. In addition, the signaling of tile IDs, including mappings between tile index and tile ID, may be based on the number of tiles in a picture. Accordingly, the mapping may be based on the number of tiles in a picture instead of based on tile columns and tile rows. For example, a loop may be employed for such that each tile index is assigned a tile ID (e.g., from the first index to the last index where the first index is zero and the last index is the number of tiles in a picture minus one).

[00121] The following is a specific example embodiment of the abovementioned aspects. The CTB raster and tile scanning process may be as follows. The list ColWidth[i] for i

ranging from 0 to num_level1_tile_columns_minus1, inclusive, specifying the width of the i-th first level tile column in units of CTBs, can be derived as follows.

```
if( uniform_level1_tile_spacing_flag )
for( i = 0; i <= num_level1_tile_columns_minus1; i++ )
   ColWidth[ i ] = ( ( i + 1 ) * PicWidthInCtbsY ) /
( num_level1_tile_columns_minus1 + 1 ) −
      ( i * PicWidthInCtbsY ) / ( num_level1_tile_columns_minus1 + 1 )
else {
ColWidth[ num_level1_tile_columns_minus1 ] = PicWidthInCtbsY                (6-1)
for( i = 0; i < num_level1_tile_columns_minus1; i++ ) {
   ColWidth[ i ] = tile_level1_column_width_minus1[ i ] + 1
   ColWidth[ num_tile_level1_columns_minus1 ] −= ColWidth[ i ]
}
}
```

[00122] The list RowHeight[j] for j ranging from 0 to num_level1_tile_rows_minus1, inclusive, specifying the height of the j-th tile row in units of CTBs, can be derived as follows:

```
if( uniform_level1_tile_spacing_flag )
for( j = 0; j <= num_level1_tile_rows_minus1; j++ )
   RowHeight[ j ] = ( ( j + 1 ) * PicHeightInCtbsY ) / (
num_level1_tile_rows_minus1 + 1 ) −
                  ( j * PicHeightInCtbsY ) / ( num_level1_tile_rows_minus1 + 1 )
else {
RowHeight[ num_level1_tile_rows_minus1 ] = PicHeightInCtbsY                (6-2)
for( j = 0; j < num_level1_tile_rows_minus1; j++ ) {
   RowHeight[ j ] = tile_level1_row_height_minus1[ j ] + 1
   RowHeight[ num_level1_tile_rows_minus1 ] −= RowHeight[ j ]
}
}
```

[00123] The list colBd[i] for i ranging from 0 to num_level1_tile_columns_minus1 + 1, inclusive, specifying the location of the i-th tile column boundary in units of CTBs, can be derived as follows:

```
for( colBd[ 0 ] = 0, i = 0; i <= num_level1_tile_columns_minus1; i++ )
   colBd[ i + 1 ] = colBd[ i ] + ColWidth[ i ]                (6-3)
```

[00124] The list rowBd[j] for j ranging from 0 to num_level1_tile_rows_minus1 + 1, inclusive, specifying the location of the j-th tile row boundary in units of CTBs, can be derived as follows:

```
for( rowBd[ 0 ] = 0, j = 0; j <= num_level1_tile_rows_minus1; j++ )
rowBd[ j + 1 ] = rowBd[ j ] + RowHeight[ j ]                                    (6-4)
```

[00125] The variable NumTilesInPic, specifying the number of tiles in a picture referring to the PPS, and the lists TileColBd[i], TileRowBd[i], TileWidth[i], and TileHeight[i] for i ranging from 0 to NumTilesInPic − 1, inclusive, specifying the location of the i-th tile column boundary in units of CTBs, the location of the i-th tile row boundary in units of CTBs, the width of the i-th tile column in units of CTBs, and the height of the i-th tile column in units of CTBs, can be derived as follows:

```
for ( tileIdx = 0, i = 0; i < NumLevel1Tiles; i++ ) {
tileX = i % ( num_level1_tile_columns_minus1 + 1 )
tileY = i / ( num_level1_tile_columns_minus1 + 1 )
if ( !level2_tile_split_flag[ i ] ) {                                          (6-5)
   TileColBd[ tileIdx ] = colBd[ tileX ]
   TileRowBd[ tileIdx ] = rowBd[ tileY ]
   TileWidth[ tileIdx ] = ColWidth[ tileX ]
   TileHeight[ tileIdx ] = RowHeight[ tileY ]
   tileIdx++
} else {
   for( k = 0; k <= num_level2_tile_columns_minus1[ i ]; k++ )
      colWidth2[ k ] = ( ( k + 1 ) * ColWidth[ tileX ] ) /
( num_level2_tile_columns_minus1[ i ] + 1 ) −
            ( k * ColWidth[ tileX ] ) / ( num_level2_tile_columns_minus1[ i ] + 1 )
   for( k = 0; k <= num_level2_tile_rows_minus1[ i ]; k++ )
      rowHeight2[ k ] = ( ( k + 1 ) * RowHeight[ tileY ] ) /
( num_level2_tile_rows_minus1[ i ] + 1 ) −
         ( k * RowHeight[ tileY ] ) / ( num_level2_tile_rows_minus1[ i ] + 1 )
   for( colBd2[ 0 ] = 0, k = 0; k <= num_level2_tile_columns_minus1[ i ]; k++ )
      colBd2[ k + 1 ] = colBd2[ k ] + colWidth2[ k ]
   for( rowBd2[ 0 ] = 0, k = 0; k <= num_level2_tile_rows_minus1[ i ]; k++ )
      rowBd2[ k + 1 ] = rowBd2[ k ] + rowHeight2[ k ]
   numSplitTiles = (num_level2_tile_columns_minus1[ i ] + 1) *
```

```
                (num_level2_tile_rows_minus1[ i ] + 1)
        for( k = 0; k < numSplitTiles; k++ ) {
            tileX2 = k % (num_level2_tile_columns_minus1[ i ] + 1)
            tileY2 = k / (num_level2_tile_columns_minus1[ i ] + 1)
            TileColBd[ tileIdx ] = colBd[ tileX ] + colBd2[ tileX2 ]
            TileRowBd[ tileIdx ] = rowBd[ tileY ] + rowBd2[ tileY2 ]
            TileWidth[ tileIdx ] = colWidth2[ tileX2 ]
            TileHeight[ tileIdx ] = rowHeight2[ tileY2 ]
            tileIdx++
        }
    }
    }
    NumTilesInPic = tileIdx
```

[00126] The list CtbAddrRsToTs[ctbAddrRs] for ctbAddrRs ranging from 0 to PicSizeInCtbsY − 1, inclusive, specifying the conversion from a CTB address in CTB raster scan of a picture to a CTB address in tile scan, can be derived as follows:

```
for (ctbAddrRs = 0; ctbAddrRs < PicSizeInCtbsY; ctbAddrRs++ ) {
tbX = ctbAddrRs % PicWidthInCtbsY
tbY = ctbAddrRs / PicWidthInCtbsYtileFound = FALSE
for ( tileIdx = NumTilesInPic − 1, i = 0; i < NumTilesInPic − 1  &&  !tileFound; i++ )
{                                                                                    (6-6)
   tileFound = tbX  <  (TileColBd[ i ] + TileWidth[ i ] )  &&  tbY  <  ( TileRowBd[ i ]
+ TileHeight[ i ] )
   if (tileFound)
      tileIdx = i
}
CtbAddrRsToTs[ ctbAddrRs ] = 0
for( i = 0; i < tileIdx; i++ )
   CtbAddrRsToTs[ ctbAddrRs ]  +=  TileHeight[ i ] * TileWidth[ i ]
CtbAddrRsToTs[ ctbAddrRs ]  +=
        ( tbY − TileRowBd[ tileIdx ]
) * TileWidth[ tileIdx ] + tbX − TileColBd[ tileIdx ]
}
```

[00127] The list CtbAddrTsToRs[ctbAddrTs] for ctbAddrTs ranging from zero to PicSizeInCtbsY − 1, inclusive, specifying the conversion from a CTB address in tile scan to a CTB address in CTB raster scan of a picture, can bederived as follows.

```
for( ctbAddrRs = 0; ctbAddrRs < PicSizeInCtbsY; ctbAddrRs++ )                    (6-7)
CtbAddrTsToRs[ CtbAddrRsToTs[ ctbAddrRs ] ] = ctbAddrRs
```

[00128] The list TileId[ctbAddrTs] for ctbAddrTs ranging from zero to PicSizeInCtbsY − 1, inclusive, specifying the conversion from a CTB address in tile scan to a tile ID, and the list FirstCtbAddrTs[tileIdx] for tileIdx ranging from zero to NumTilesInPic − 1, inclusive, specifying the conversion from a tile ID to the CTB address in tile scan of the first CTB in the tile can be derived as follows.

```
for( i = 0, tileIdx = 0; i  <=  NumTilesInPic; i++, tileIdx++ ) {
    for( y = TileRowBd[ i ]; y < TileRowBd[ i + 1 ]; y++ )                       (6-8)
        for( x = TileColBd[ i ]; x < TileColBd[ i + 1 ]; x++ )
            TileId[ CtbAddrRsToTs[ y * PicWidthInCtbsY+ x ] ] = tileIdx
    FirstCtbAddrTs[ tileIdx ] =
        CtbAddrRsToTs[ TileColBd[ tileIdx ] ] * PicWidthInCtbsY + TileColBd[ tileIdx ] ]
    }
```

[00129] The list NumCtusInTile[tileIdx] for tileIdx ranging from zero to NumTilesInPic− 1, inclusive, specifying the conversion from a tile index to the number of CTUs in the tile, can be derived as follows.

```
for( i = 0,tileIdx = 0; i < NumTilesInPic; i++, tileIdx++ )                      (6-9)
NumCtusInTile[ tileIdx ] = TileColWidth[ tileIdx ] * TileRowHeight[ tileIdx ]
```

[00130] An example picture parameter set RBSP syntax is as follows.

pic_parameter_set_rbsp( ) {	**Descriptor**
pps_pic_parameter_set_id	ue(v)
pps_seq_parameter_set_id	ue(v)
transform_skip_enabled_flag	u(1)
single_tile_in_pic_flag	u(1)
if(!single_tile_in_pic_flag) {	
num_level1_tile_columns_minus1	ue(v)
num_level1_tile_rows_minus1	ue(v)
uniform_level1_tile_spacing_flag	u(1)
if(!uniform_level1_tile_spacing_flag) {	

for(i = 0; i < num_level1_tile_columns_minus1; i++)	
level1_tile_column_width_minus1[i]	ue(v)
for(i = 0; i < num_level1_tile_rows_minus1; i++)	
level1_tile_row_height_minus1[i]	ue(v)
}	
level2_tile_present_flag	u(1)
for(i = 0; level2_tile_present_flag && i < NumLevel1Tiles; i++) {	
level2_tile_split_flag[i]	u(1)
if(level2_tile_split_flag) {	
num_level2_tile_columns_minus1[i]	ue(v)
num_level2_tile_rows_minus1[i]	ue(v)
}	
}	
if(NumTilesInPic > 1)	
loop_filter_across_tiles_enabled_flag	u(1)
}	
rbsp_trailing_bits( )	
}	

[00131] Example picture parameter set RBSP semantics are as follows. The num_level1_tile_columns_minus1 plus 1 specifies the number of level 1 tile columns partitioning the picture. The num_level1_tile_columns_minus1 shall be in the range of zero to PicWidthInCtbsY − 1, inclusive. When not present, the value of num_level1_tile_columns_minus1 is inferred to be equal to zero. The num_level1_tile_rows_minus1 plus 1 specifies the number of level one tile rows partitioning the picture. num_level1_tile_rows_minus1 shall be in the range of zero to PicHeightInCtbsY − 1, inclusive. When not present, the value of num_level1_tile_rows_minus1 is inferred to be equal to zero. The variable NumLevel1Tiles is set equal to (num_level1_tile_columns_minus1 + 1) * (num_level1_tile_rows_minus1 + 1). When single_tile_in_pic_flag is equal to zero, NumTilesInPic shall be greater than one. The uniform_level1_tile_spacing_flag is set equal to one to specify that level 1 tile column boundaries and likewise level 1 tile row boundaries are distributed uniformly across the picture. The uniform_level1_tile_spacing_flag is equal to zero to specify that level one tile

column boundaries and likewise level one tile row boundaries are not distributed uniformly across the picture but signaled explicitly using the syntax elements level1_tile_column_width_minus1[i] and level1_tile_row_height_minus1[i]. When not present, the value of uniform_level1_tile_spacing_flag is inferred to be equal to one.

[00132] The level1_tile_column_width_minus1[i] plus 1 specifies the width of the i-th level one tile column in units of CTBs. The level1_tile_row_height_minus1[i] plus 1 specifies the height of the i-th tile level one row in units of CTBs. The level2_tile_present_flag specifies that one or more level one tiles are divided into more level two tiles. The level2_tile_split_flag[i] specifies that the i-th level one tile is divided into two or more tiles. The num_level2_tile_columns_minus1[i] plus 1 specifies the number of tile columns partitioning the i-th tile. The num_level2_tile_columns_minus1[i] shall be in the range of zero to ColWidth[i], inclusive. When not present, the value of num_level2_tile_columns_minus1[i] is inferred to be equal to zero. The num_level2_tile_rows_minus1[i] plus 1 specifies the number of tile rows partitioning the i-th tile. The num_level2_tile_rows_minus1[i] shall be in the range of zero to RowHeight[i], inclusive. When not present, the value of num_level2_tile_rows_minus1[i] is inferred to be equal to zero.

[00133] When level2_tile_split_flag[i] is equal to one, the value of (num_level2_tile_columns_minus1[i] + 1) * (num_level2_tile_rows_minus1[i] + 1) shall be greater than one. A picture may contain zero or more level one tiles that have level2_tile_split_flag[i] equal to zero and zero or more level two tiles. A level one tile that has level2_tile_split_flag[i] equal to zero may not be counted in the total number of tiles in a picture. When such tile is referenced, the set of level two tiles resulted from such a split may be collectively referenced.

[00134] The following variables are derived by invoking the CTB raster and tile scanning conversion process: the list ColWidth[i] for i ranging from 0 to num_level1_tile_columns_minus1, inclusive, specifying the width of the i-th level 1 tile column in units of CTBs; the list RowHeight[j] for j ranging from 0 to num_level1_tile_rows_minus1, inclusive, specifying the height of the j-th level 1 tile row in units of CTBs; the variable NumTilesInPic, specifying the number of tiles in a picture referring to the PPS; the list TileWidth[i] for i ranging from 0 to NumTilesInPic, inclusive, specifying the width of the i-th tile in units of CTBs; the list TileHeight[i] for i ranging from 0 to NumTilesInPic, inclusive, specifying the height of the i-th tile in units of CTBs; the list TileColBd[i] for i ranging from 0 to NumTilesInPic, inclusive, specifying the location of the

i-th tile column boundary in units of CTBs; the list TileRowBd[i] for j ranging from 0 to NumTilesInPic, inclusive, specifying the location of the i-th tile row boundary in units of CTBs; the list CtbAddrRsToTs[ctbAddrRs] for ctbAddrRs ranging from 0 to PicSizeInCtbsY − 1, inclusive, specifying the conversion from a CTB address in the CTB raster scan of a picture to a CTB address in the tile scan; the list CtbAddrTsToRs[ctbAddrTs] for ctbAddrTs ranging from 0 to PicSizeInCtbsY − 1, inclusive, specifying the conversion from a CTB address in the tile scan to a CTB address in the CTB raster scan of a picture; the list TileId[ctbAddrTs] for ctbAddrTs ranging from 0 to PicSizeInCtbsY − 1, inclusive, specifying the conversion from a CTB address in tile scan to a tile ID; the list NumCtusInTile[tileIdx] for tileIdx ranging from 0 to PicSizeInCtbsY − 1, inclusive, specifying the conversion from a tile index to the number of CTUs in the tile; and the list FirstCtbAddrTs[tileIdx] for tileIdx ranging from 0 to NumTilesInPic − 1, inclusive, specifying the conversion from a tile ID to the CTB address in tile scan of the first CTB in the tile.

[00135] Example tile group header semantics are as follows. A tile group may contain only a number of complete level one tiles or only a number of complete level two tiles of one single level one tile. A tile_group_address specifies the tile address of the first tile in the tile group, where the tile address is equal to TileId[firstCtbAddrTs] as specified by Equation 6-8 with firstCtbAddrTs being the CTB address in tile scan of the CTBs of the first CTU in the tile group. The length of tile_group_address is Ceil(Log2 (NumTilesInPic)) bits. The value of tile_group_address shall be in the range of zero to NumTilesInPic − 1, inclusive, and the value of tile_group_address shall not be equal to the value of tile_group_address of any other coded tile group NAL unit of the same coded picture. When tile_group_address is not present it is inferred to be equal to zero.

[00136] The following is a second specific example embodiment of the abovementioned aspects. An example CTB raster and tile scanning process is as follows. The variable NumTilesInPic, specifying the number of tiles in a picture referring to the PPS, and the lists TileColBd[i], TileRowBd[i], TileWidth[i], and TileHeight[i] for i ranging from zero to NumTilesInPic − 1, inclusive, specifying the location of the i-th tile column boundary in units of CTBs, the location of the i-th tile row boundary in units of CTBs, the width of the i-th tile column in units of CTBs, and the height of the i-th tile column in units of CTBs, are derived as follows.

```
for ( tileIdx = 0, i = 0; i < NumLevel1Tiles; i++ ) {
tileX = i % ( num_level1_tile_columns_minus1 + 1 )
tileY = i / ( num_level1_tile_columns_minus1 + 1 )
if ( !level2_tile_split_flag[ i ] ) {                                    (6-5)
   TileColBd[ tileIdx ] = colBd[ tileX ]
   TileRowBd[ tileIdx ] = rowBd[ tileY ]
   TileWidth[ tileIdx ] = ColWidth[ tileX ]
   TileHeight[ tileIdx ] = RowHeight[ tileY ]
   tileIdx++
} else {
   if ( uniform_level2_tile_spacing_flag[ i ] ) {
      for( k = 0; k <= num_level2_tile_columns_minus1[ i ]; k++ )
         colWidth2[ k ] = ( ( k + 1 ) * ColWidth[ tileX ] ) /
         ( num_level2_tile_columns_minus1[ i ] + 1 ) −
         ( k * ColWidth[ tileX ] ) / ( num_level2_tile_columns_minus1[ i ] + 1 )
      for( k = 0; k <= num_level2_tile_rows_minus1[ i ]; k++ )
         rowHeight2[ k ] = ( ( k + 1 ) * RowHeight[ tileY ] ) /
         ( num_level2_tile_rows_minus1[ i ] + 1 ) −
         ( k * RowHeight[ tileY ] ) / ( num_level2_tile_rows_minus1[ i ] + 1 )
   } else {
      colWidth2[ num_level2_tile_columns_minus1[ i ] ] = ColWidth[ tileX ] )
      for( k = 0; k <= num_level2_tile_columns_minus1[ i ]; k++ ) {
         colWidth2[ k ] = tile_level2_column_width_minus1[ k ] + 1
         colWidth2[ k ] −= colWidth2[ k ]
      }
      rowHeight2[ num_level2_tile_rows_minus1[ i ] ] = RowHeight[ tileY ] )
      for( k = 0; k <= num_level2_tile_rows_minus1[ i ]; k++ ) {
         rowHeigh2[ k ] = tile_level2_column_width_minus1[ k ] + 1
         rowHeight2[ k ] −= rowHeight2[ k ]
      }
   }
   for( colBd2[ 0 ] = 0, k = 0; k <= num_level2_tile_columns_minus1[ i ]; k++ )
      colBd2[ k + 1 ] = colBd2[ k ] + colWidth2[ k ]
   for( rowBd2[ 0 ] = 0, k = 0; k <= num_level2_tile_rows_minus1[ i ]; k++ )
```

```
            rowBd2[ k + 1 ] = rowBd2[ k ] + rowHeight2[ k ]
        numSplitTiles = (num_level2_tile_columns_minus1[ i ] + 1) *
                (num_level2_tile_rows_minus1[ i ] + 1)
        for( k = 0; k < numSplitTiles; k++ ) {
            tileX2 = k % (num_level2_tile_columns_minus1[ i ] + 1)
            tileY2 = k / (num_level2_tile_columns_minus1[ i ] + 1)
            TileColBd[ tileIdx ] = colBd[ tileX ] + colBd2[ tileX2 ]
            TileRowBd[ tileIdx ] = rowBd[ tileY ] + rowBd2[ tileY2 ]
            TileWidth[ tileIdx ] = colWidth2[ tileX2 ]
            TileHeight[ tileIdx ] = rowHeight2[ tileY2 ]
            tileIdx++
        }
    }
    }
    NumTilesInPic = tileIdx
```

[00137] An example picture parameter set RBSP syntax is as follows.

pic_parameter_set_rbsp( ) {	Descriptor
pps_pic_parameter_set_id	ue(v)
pps_seq_parameter_set_id	ue(v)
transform_skip_enabled_flag	u(1)
single_tile_in_pic_flag	u(1)
if(!single_tile_in_pic_flag) {	
num_level1_tile_columns_minus1	ue(v)
num_level1_tile_rows_minus1	ue(v)
uniform_level1_tile_spacing_flag	u(1)
if(!uniform_level1_tile_spacing_flag) {	
for(i = 0; i < num_level1_tile_columns_minus1; i++)	
level1_tile_column_width_minus1[i]	ue(v)
for(i = 0; i < num_level1_tile_rows_minus1; i++)	
level1_tile_row_height_minus1[i]	ue(v)
}	
level2_tile_present_flag	u(1)
for(i = 0; level2_tile_present_flag && i < NumLevel1Tiles; i++) {	

level2_tile_split_flag[i]	u(1)
if(level2_tile_split_flag) {	
num_level2_tile_columns_minus1[i]	ue(v)
num_level2_tile_rows_minus1[i]	ue(v)
uniform_level2_tile_spacing_flag[i]	u(1)
if(!uniform_level2_tile_spacing_flag[i]) {	
for(j = 0; j < num_level2_tile_columns_minus1[i]; j++)	
level2_tile_column_width_minus1[j]	ue(v)
for(j = 0; j < num_level2_tile_rows_minus1[i]; j++)	
level2_tile_row_height_minus1[j]	ue(v)
}	
}	
}	
if(NumTilesInPic > 1)	
loop_filter_across_tiles_enabled_flag	u(1)
}	
rbsp_trailing_bits( )	
}	

[00138] Example picture parameter set RBSP semantics are as follows. The uniform_level2_tile_spacing_flag[i] is set equal to one to specify that level two tile column boundaries of the i-th level one tile and likewise level two tile row boundaries of the i-th level one tile are distributed uniformly across the picture. The uniform_level2_tile_spacing_flag[i] can be set equal to zero to specify that level two tile column boundaries of the i-th level one tile and likewise level two tile row boundaries of the i-th level one tile are not distributed uniformly across the picture but signaled explicitly using the syntax elements level2_tile_column_width_minus1[j] and level2_tile_row_height_minus1[j]. When not present, the value of uniform_level2_tile_spacing_flag[i] is inferred to be equal to one. The level2_tile_column_width_minus1[j] plus 1 specifies the width of the j-th level two tile column of the i-th level one tile in units of CTBs. The level2_tile_row_height_minus1[j] plus 1 specifies the height of the j-th tile level two row of the i-th level one tile in units of CTBs.

[00139] The following is a third specific example embodiment of the abovementioned aspects. A CTB raster and tile scanning process is as follows. The list ColWidth[i] for i ranging from zero to num_level1_tile_columns_minus1, inclusive, specifying the width of the i-th first level tile column in units of CTBs, may be derived as follows:

```
for( i = 0; i <= num_level1_tile_columns_minus1; i++ )
    ColWidth[ i ] = ( ( i + 1 ) * PicWidthInCtbsY ) /
( num_level1_tile_columns_minus1 + 1 ) −
                    ( i * PicWidthInCtbsY ) / ( num_level1_tile_columns_minus1 + 1 )
```

[00140] The list RowHeight[j] for j ranging from zero to num_level1_tile_rows_minus1, inclusive, specifying the height of the j-th tile row in units of CTBs may be derived as follows:

```
for( j = 0; j <= num_level1_tile_rows_minus1; j++ )
    RowHeight[ j ] = ( ( j + 1 ) * PicHeightInCtbsY ) / (
num_level1_tile_rows_minus1 + 1 ) −
                    ( j * PicHeightInCtbsY ) / ( num_level1_tile_rows_minus1 + 1 )
```

[00141] An example picture parameter set RBSP syntax is as follows.

pic_parameter_set_rbsp( ) {	Descriptor
pps_pic_parameter_set_id	ue(v)
pps_seq_parameter_set_id	ue(v)
transform_skip_enabled_flag	u(1)
single_tile_in_pic_flag	u(1)
if(!single_tile_in_pic_flag) {	
num_level1_tile_columns_minus1	ue(v)
num_level1_tile_rows_minus1	ue(v)
level2_tile_present_flag	u(1)
for(i = 0; level2_tile_present_flag && i < NumLevel1Tiles; i++) {	
level2_tile_split_flag[i]	u(1)
if(level2_tile_split_flag) {	
num_level2_tile_columns_minus1[i]	ue(v)
num_level2_tile_rows_minus1[i]	ue(v)
}	
}	

if(NumTilesInPic > 1)	
loop_filter_across_tiles_enabled_flag	u(1)
}	
rbsp_trailing_bits( )	
}	

[00142] The following is a fourth specific example embodiment of the abovementioned aspects. An example picture parameter set RBSP syntax is as follows.

pic_parameter_set_rbsp( ) {	Descriptor
pps_pic_parameter_set_id	ue(v)
pps_seq_parameter_set_id	ue(v)
transform_skip_enabled_flag	u(1)
if ((PicWidthInCtbsY * CtbSizeY) >= (2 * MinTileWidth) || (PicHeightInCtbsY * CtbSizeY) >= (2 * MinTileHeight))	
single_tile_in_pic_flag	u(1)
if(!single_tile_in_pic_flag) {	
if (PicWidthInCtbsY * CtbSizeY >= (2 * MinTileWidth))	
num_level1_tile_columns_minus1	ue(v)
if (PicHeightInCtbsY * CtbSizeY >= (2 * MinTileHeight))	
num_level1_tile_rows_minus1	ue(v)
uniform_level1_tile_spacing_flag	u(1)
if(!uniform_level1_tile_spacing_flag) {	
for(i = 0; i < num_level1_tile_columns_minus1; i++)	
level1_tile_column_width_minus1[i]	ue(v)
for(i = 0; i < num_level1_tile_rows_minus1; i++)	
level1_tile_row_height_minus1[i]	ue(v)
}	
if(Level1TilesMayFurtherBeSplit)	
level2_tile_present_flag	u(1)
for(i = 0; level2_tile_present_flag && i < NumLevel1Tiles; i++) {	
if (ColWidth[i] * CtbSizeY > MinTileWidth || RowHeight[i] * CtbSizeY > MinTileHeight)	

level2_tile_split_flag[i]	u(1)
if(level2_tile_split_flag) {	
if (ColWidth[i] * CtbSizeY >= (2 * MinTileWidth))	
num_level2_tile_columns_minus1[i]	ue(v)
if (RowHeight[i] * CtbSizeY >= (2 * MinTileHeight))	
num_level2_tile_rows_minus1[i]	ue(v)
}	
}	
if (NumTilesInPic > 1)	
loop_filter_across_tiles_enabled_flag	u(1)
}	
rbsp_trailing_bits( )	
}	

[00143] An example picture parameter set RBSP semantics is as follows. Bitstream conformance may require that the following constraints apply. The value MinTileWidth specifies the minimum tile width and shall be equal to two hundred fifty six luma samples. The value MinTileHeight specifies the minimum tile height and shall be equal to sixty four luma samples. The value of minimum tile width and minimum tile height may change according to profile and level definition. The variable Level1TilesMayBeFurtherSplit may be derived as follows:

```
Level1TilesMayFurtherBeSplit = 0
for ( i = 0, ! Level1TilesMayFurtherBeSplit && i = 0; i < NumLevel1Tiles; i++ )
if ( ( ColWidth[ i ] * CtbSizeY >= ( 2 * MinTileWidth ) ) ||
        ( RowHeight[ i ] * CtbSizeY >= ( 2 * MinTileHeight ) ))
    Level1TilesMayFurtherBeSplit = 1
```

[00144] The level2_tile_present_flag specifies that one or more level tiles are divided into more tiles. When not present, the value of level2_tile_present_flag is inferred to be equal to zero. The level2_tile_split_flag[i] plus 1 specifies that the i-th level one tile is divided into two or more tiles. When not present, the value of level2_tile_split_flag[i] is inferred to be equal to zero.

[00145] The following is a fifth specific example embodiment of the abovementioned aspects. Each tile location and size may be signaled. The syntax to support such tile structure signaling can be as tabulated below. The tile_top_left_address[i] and

tile_bottom_right_address[i] are the CTU index within the picture that indicate the rectangular area covered by the tile. The number of bits to signal these syntax elements should be enough to represent the max number of CTUs in the picture.

pic_parameter_set_rbsp( ) {	Descriptor
...	.
single_tile_in_pic_flag	u(1)
if(!single_tile_in_pic_flag) {	
tile_size_unit_idc	ue(v)
uniform_tile_flag	u(1)
if(uniform_tile_flag) {	
num_tile_columns_minus1	ue(v)
num_tile_rows_minus1	ue(v)
}	
else {	
num_tiles_minus2	ue(v)
for(i = 0; i < (num_tiles_minus2 + 2); i++) {	
tile_top_left_address[i]	u(v)
tile_bottom_right_address[i]	u(v)
}	
}	
loop_filter_across_tiles_enabled_flag	u(1)
}	
rbsp_trailing_bits( )	
}	

[00146] Each tile location and size may be signaled. The syntax to support such tile structure signaling can be as tabulated below. The tile_top_left_address[i] is the CTU index of the first CTU in the tile in the order of a CTU raster scan of a picture. Tile width and tile height specify the size of the tile. Some bits may be saved when signaling these two syntax elements by first signaling the common tile size unit.

pic_parameter_set_rbsp( ) {	Descriptor
...	.

single_tile_in_pic_flag	u(1)
if(!single_tile_in_pic_flag) {	
tile_size_unit_idc	ue(v)
uniform_tile_flag	u(1)
if(uniform_tile_flag) {	
num_tile_columns_minus1	ue(v)
num_tile_rows_minus1	ue(v)
}	
else {	
num_tiles_minus2	ue(v)
for(i = 0; i < (num_tiles_minus2 + 2); i++) {	
tile_top_left_address[i]	u(v)
tile_width_minus1[i]	ue(v)
tile_height_minus1[i]	ue(v)
}	
}	
loop_filter_across_tiles_enabled_flag	u(1)
}	
rbsp_trailing_bits( )	
}	

[00147] Alternatively, the signaling can be as follows.

pic_parameter_set_rbsp( ) {	Descriptor
...	.
single_tile_in_pic_flag	u(1)
if(!single_tile_in_pic_flag) {	
tile_size_unit_idc	ue(v)
uniform_tile_flag	u(1)
if(uniform_tile_flag) {	
num_tile_columns_minus1	ue(v)
num_tile_rows_minus1	ue(v)
}	

else {	
num_tiles_minus2	ue(v)
for(i = 0; i < (num_tiles_minus2 + 2); i++) {	
tile_x_offset[i]	ue(v)
tile_y_offset[i]	ue(v)
tile_width_minus1[i]	ue(v)
tile_height_minus1[i]	ue(v)
}	
}	
loop_filter_across_tiles_enabled_flag	u(1)
}	
rbsp_trailing_bits( )	
}	

[00148] In another example, each tile size can be signaled as follows. For signaling a flexible tile structure, the location of each tile may not be signaled. Instead a flag may be signaled to specify whether to put the tile immediately to the right of or immediately below the previous tile. This flag may be not present if the tile can only be to the right or can only be below the current tile.

[00149] The values of tile_x_offset[i] and tile_y_offset[i] can be derived by the following ordered steps.

tile_x_offset[0] and tile_y_offset[0] are set equal to 0.

maxWidth is set equal to tile_width[0] and maxHeight is set equal to tile_height[0]

runningWidth is set equal to tile_width[0] and runningHeight is set equal to tile_height[0]

lastNewRowHeight is set equal to 0

TilePositionCannotBeInferred = false

For i > 0, the following applies:

Let the value isRight be set as follows:

if runningWidth + tile_width[i] <= PictureWidth, then isRight = = 1

else, isRight = = 0

Let the value isBelow is set as follows:

if runningHeight + tile_height[i] <= PictureHeight, then isBelow = = 1

else, isBelow = = 0

If isRight = = 1 && isBelow = = 1 then TilePositionCannotBeInferred = true
If isRight = = 1 && isBelow = = 0 then the following apply:
right_tile_flag[i] = 1
tile_x_offset[i] = runningWidth.
tile_y_offset[i] = (runningWidth == maxWidth) ? 0 : lastNewRowHeight
lastNewRowHeight = (runningWidth == maxWidth) ? 0 : lastNewRowHeight
else if isRight = = 0 && isBelow = = 1 then the following apply:
right_tile_flag[i] = 0
tile_y_offset[i] = runningHeight
tile_x_offset[i] = (runningHeight == maxHeight) ? 0 : tile_x_offset[i – 1]
lastNewRowHeight = (runningHeight == maxHeight && runningWidth == maxWidth) ? runningHeight : lastNewRowHeight
else if isRight = = 1 && isBelow = = 1 && right_tile_flag[i] = = 1 then the following apply:
tile_x_offset[i] = runningWidth.
tile_y_offset[i] = (runningWidth == maxWidth) ? 0 : lastNewRowHeight
lastNewRowHeight = (runningWidth == maxWidth) ? 0 : lastNewRowHeight
else (i.e., isRight = = 1 && isBelow = = 1 && right_tile_flag[i] = = 0) then the following apply:
tile_y_offset[i] = runningHeight
tile_x_offset[i] = (runningHeight == maxHeight) ? 0 : tile_x_offset[i – 1]
lastNewRowHeight = (runningHeight == maxHeight && runningWidth == maxWidth) ? runningHeight : lastNewRowHeight
if right_tile_flag[i] = = 1, the following apply:
runningWidth = runningWidth + tile_width[i]
if runningWidth > maxWidth, then set maxWidth equal to runningWidth
runningHeight is equal to tile_y_offset[i] + tile_height[i]
else (i.e., right_tile_flag[i] = = 0), the following apply:
runningHeight = runningHeight + tile_height[i]
if runningHeight > maxHeight, then set maxHeight equal to runningHeight
runningWidth is equal to tile_x_offset[i] + tile_width[i]

[00150] The preceding can be described in pseudocode as follows.

```
tile_x_offset[0] = 0
tile_y_offset[0] = 0
maxWidth = tile_width[0]
maxHeight = tile_height[0]
runningWidth = tile_width[0]
runningHeight = tile_height[0]
lastNewRowHeight = 0
isRight = false
isBelow = false
TilePositionCannotBeInferred = false
for( i = 1; i < num_tiles_minus2 + 2; i++ ) {
TilePositionCannotBeInferred = false
isRight = ( runningWidth + tile_width[i] <= PictureWidth ) ? true : false
isbelow = ( runningHeight + tile_height[i] <= PictureHeight ) ? true : false
    if (!isRight && !isBelow)
    //Error. This case shall not happen!
if (isRight && isBelow)
    TilePositionCannotBeInferred = true
    if (isRight && !isBelow) {
    right_tile_flag[i] = true
    tile_x_offst[i] = runningWidth
    tile_y_offset[i] = (runningWidth == maxWidth) ? 0 : lastNewRowHeight
    lastNewRowHeight = tile_y_offset[i]
}
else if (!isRight && isBelow) {
    right_tile_flag[i] = false
    tile_y_offset[i] = runningHeight
    tile_x_offset[i] = (runningHeight == maxHeight) ? 0 : tile_x_offset[i – 1]
    lastNewRowHeight = (runningHeight == maxHeight && runningWidth ==
maxWidth) ?
                runningHeight : lastNewRowHeight
}
else if ( right_tile_flag[i] ) {
    tile_x_offst[i] = runningWidth
```

```
        tile_y_offset[i] = (runningWidth == maxWidth) ? 0 : lastNewRowHeight
        lastNewRowHeight = tile_y_offset[i]
    }
    else {
        tile_y_offset[i] = runningHeight
        tile_x_offset[i] = (runningHeight == maxHeight) ? 0 : tile_x_offset[i – 1]
        lastNewRowHeight = (runningHeight == maxHeight && runningWidth ==
    maxWidth) ?
                runningHeight : lastNewRowHeight
    }
    }
    if ( right_tile_flag[i] ) {
    runningWidth += tile_width[i]
    if ( runningWidth > maxWidth ) maxWidth = runningWidth
    runningHeight = tile_y_offset[i] + tile_height[i]
    }
    else {
    runningHeight += tile_height[i]
    if ( runningHeight > maxHeight ) maxHeight = runningHeight
    runningWidth = tile_x_offset[i] + tile_width[i]
    }
```

pic_parameter_set_rbsp( ) {	Descriptor
...	.
single_tile_in_pic_flag	u(1)
if(!single_tile_in_pic_flag) {	
tile_size_unit_idc	ue(v)
uniform_tile_flag	u(1)
if(uniform_tile_flag) {	
num_tile_columns_minus1	ue(v)
num_tile_rows_minus1	ue(v)
}	
else {	

num_tiles_minus2	ue(v)
for(i = 0; i < (num_tiles_minus2 + 2); i++) {	
tile_width_minus1 [i]	ue(v)
tile_height_minus1 [i]	ue(v)
if(TilePositionCannotBeInferred)	
right_tile_flag[i]	u(1)
}	
}	
loop_filter_across_tiles_enabled_flag	u(1)
}	
rbsp_trailing_bits( )	
}	

[00151] The following is an implementation of deriving the size of the last tile in pseudocode.

```
tile_x_offset[0] = 0
tile_y_offset[0] = 0
maxWidth = tile_width[0]
maxHeight = tile_height[0]
runningWidth = tile_width[0]
runningHeight = tile_height[0]
lastNewRowHeight = 0
isRight = false
isBelow = false
TilePositionCannotBeInferred = false
for( i = 1; i < num_tiles_minus2 + 2; i++ ) {
currentTileWidth = ( i == num_tiles_minus2 + 1 ) ? (PictureWidth – runningWidth) %
PictureWidth
                                        : tile_width[i]
currentTileHeight = ( i == num_tiles_minus2 + 1 ) ? (PictureHeight – runningHeight)
% PictureHeight
                                          : tile_Height[i]

isRight = ( runningWidth + currentTileWidth <= PictureWidth ) ? true : false
```

```
isbelow = ( runningHeight + currentTileHeight <= PictureHeight ) ? true : false

if (!isRight && !isBelow)
    //Error. This case shall not happen!
if (isRight && isBelow)
    TilePositionCannotBeInferred = true

if (isRight && !isBelow) {
    right_tile_flag[i] = true
    tile_x_offst[i] = runningWidth
    tile_y_offset[i] = (runningWidth == maxWidth) ? 0 : lastNewRowHeight
    lastNewRowHeight = tile_y_offset[i]
}
else if (!isRight && isBelow) {
    right_tile_flag[i] = false
    tile_y_offset[i] = runningHeight
    tile_x_offset[i] = (runningHeight == maxHeight) ? 0 : tile_x_offset[i – 1]
    lastNewRowHeight = (runningHeight == maxHeight && runningWidth ==
maxWidth) ?
                runningHeight : lastNewRowHeight
}
else if ( right_tile_flag[i] ) {
    tile_x_offst[i] = runningWidth
    tile_y_offset[i] = (runningWidth == maxWidth) ? 0 : lastNewRowHeight
    lastNewRowHeight = tile_y_offset[i]
}
else {
    tile_y_offset[i] = runningHeight
    tile_x_offset[i] = (runningHeight == maxHeight) ? 0 : tile_x_offset[i – 1]
    lastNewRowHeight = (runningHeight == maxHeight && runningWidth ==
maxWidth) ?
                runningHeight : lastNewRowHeight
}
}
```

```
if ( right_tile_flag[i] ) {
runningWidth += currentTileWidth
if ( runningWidth > maxWidth ) maxWidth = runningWidth
runningHeight = tile_y_offset[i] + currentTileHeight
}
else {
runningHeight += currentTileHeight
if ( runningHeight > maxHeight ) maxHeight = runningHeight
runningWidth = tile_x_offset[i] + currentTileWidth
}
```

pic_parameter_set_rbsp( ) {	Descriptor
...	.
single_tile_in_pic_flag	u(1)
if(!single_tile_in_pic_flag) {	
tile_size_unit_idc	ue(v)
uniform_tile_flag	u(1)
if(uniform_tile_flag) {	
num_tile_columns_minus1	ue(v)
num_tile_rows_minus1	ue(v)
}	
else {	
num_tiles_minus2	ue(v)
for(i = 0; i < (num_tiles_minus2 + 1); i++) {	
tile_width_minus1[i]	ue(v)
tile_height_minus1[i]	ue(v)
if(TilePositionCannotBeInferred)	
right_tile_flag[i]	u(1)
}	
}	
loop_filter_across_tiles_enabled_flag	u(1)
}	
rbsp_trailing_bits( )	
}	

[00152] For further signaling bit-saving, the number of unique tile sizes can be signaled to support tabulation of the unit tile size. Tile size can then be referenced by index only.

pic_parameter_set_rbsp( ) {	Descriptor
...	.
single_tile_in_pic_flag	u(1)
if(!single_tile_in_pic_flag) {	
tile_size_unit_idc	ue(v)
uniform_tile_flag	u(1)
if(uniform_tile_flag) {	
num_tile_columns_minus1	ue(v)
num_tile_rows_minus1	ue(v)
}	
else {	
num_tiles_minus2	ue(v)
num_unique_tile_sizes	ue(v)
for(i = 0; i < num_unique_tile_sizes; i++) {	
preset_tile_width_minus1[i]	ue(v)
preset_tile_height_minus1[i]	ue(v)
}	
for(i = 0; i < (num_tiles_minus2 + 2); i++) {	
if (num_unique_tile_sizes)	
tile_size_idx[i]	u(v)
else {	
tile_width_minus1[i]	ue(v)
tile_height_minus1[i]	ue(v)
}	
if(TilePositionCannotBeInferred)	
right_tile_flag[i]	u(1)
}	
}	
loop_filter_across_tiles_enabled_flag	u(1)
}	

rbsp_trailing_bits( )	
}	

[00153] The following is a sixth specific example embodiment of the abovementioned aspects. An example CTB raster and tile scanning process is as follows. The list TileId[ctbAddrTs] for ctbAddrTs ranging from zero to PicSizeInCtbsY − 1, inclusive, specifying the conversion from a CTB address in tile scan to a tile ID, and the list FirstCtbAddrTs[tileIdx] for tileIdx ranging from zero to NumTilesInPic − 1, inclusive, specifying the conversion from a tile ID to the CTB address in tile scan of the first CTB in the tile may be derived as follows.

```
for( i = 0, tileIdx = 0; i <= NumTilesInPic; i++, tileIdx++ ) {
for( y = TileRowBd[ i ]; y < TileRowBd[ i + 1 ]; y++ )                    (6-8)
    for( x = TileColBd[ i ]; x < TileColBd[ i + 1 ]; x++ )
        TileId[ CtbAddrRsToTs[ y * PicWidthInCtbsY+ x ] ] =
                explicit_tile_id_flag ? tile_id_val[ i ] : tileIdx
FirstCtbAddrTs[ tileIdx ] =

CtbAddrRsToTs[ TileColBd[ tileIdx ] ] * PicWidthInCtbsY + TileColBd[ tileIdx ] ]
}
```

[00154] The set TileIdToIdx[tileId] for a set of NumTilesInPic tileId values specifying the conversion from a tile ID to a tile index may be derived as follows.

```
for( ctbAddrTs = 0, tileIdx = 0, tileStartFlag = 1; ctbAddrTs < PicSizeInCtbsY;
ctbAddrTs++ ) {
if( tileStartFlag ) {
    TileIdToIdx[ TileId[ ctbAddrTs ] ] = tileIdx
    tileStartFlag = 0
}
tileEndFlag = ctbAddrTs == PicSizeInCtbsY − 1 || TileId[ ctbAddrTs + 1 ] !=
        TileId[ ctbAddrTs ]
if( tileEndFlag ) {
    tileIdx++
    tileStartFlag = 1
```

}

}

[00155] An example picture parameter set RBSP syntax and semantics is as follows.

pic_parameter_set_rbsp( ) {	Descriptor
pps_pic_parameter_set_id	ue(v)
pps_seq_parameter_set_id	ue(v)
transform_skip_enabled_flag	u(1)
single_tile_in_pic_flag	u(1)
if(!single_tile_in_pic_flag) {	
num_level1_tile_columns_minus1	ue(v)
num_level1_tile_rows_minus1	ue(v)
uniform_level1_tile_spacing_flag	u(1)
if(!uniform_level1_tile_spacing_flag) {	
for(i = 0; i < num_level1_tile_columns_minus1; i++)	
level1_tile_column_width_minus1[i]	ue(v)
for(i = 0; i < num_level1_tile_rows_minus1; i++)	
level1_tile_row_height_minus1[i]	ue(v)
}	
level2_tile_present_flag	u(1)
for(i = 0; level2_tile_present_flag && i < NumLevel1Tiles; i++) {	
level2_tile_split_flag[i]	u(1)
if(level2_tile_split_flag) {	
num_level2_tile_columns_minus1[i]	ue(v)
num_level2_tile_rows_minus1[i]	ue(v)
}	
}	
if(NumTilesInPic > 1)	
loop_filter_across_tiles_enabled_flag	u(1)
}	
tile_id_len_minus1	ue(v)
explicit_tile_id_flag	u(1)
if(explicit_tile_id_flag)	

for(i = 0; i < NumTilesInPic; i++)	
tile_id_val[i]	u(v)
rbsp_trailing_bits( )	
}	

[00156] In some examples, the semantics can be as follows. The tile_id_len_minus1 plus one specifies the number of bits used to represent the syntax element tile_id_val[i] and the syntax element tile_group_address in tile group headers referring to the PPS. The value of tile_id_len_minus1 may be in the range of Ceil(Log2(NumTilesInPic) to fifteen, inclusive. In other examples, the semantics can be as follows. The tile_id_len_minus1 plus one may specify the number of bits used to represent the syntax element tile_id_val[i] and the syntax elements in tile group headers referring to the PPS that refer to tile ID values. The value of tile_id_len_minus1 may be in the range of Ceil(Log2(NumTilesInPic) to fifteen, inclusive.

[00157] The explicit_tile_id_flag may be set equal to one to specify that the tile ID for each tile is explicitliy signaled. The explicit_tile_id_flag may be set equal to zero to specify that tile IDs are not explicitly signaled. The tile_id_val[i] may specify the tile ID of the tile of the i-th tile in the picture. The length of tile_id_val[i] may be tile_id_len_minus1 + 1 bits. For any integer m in the range of zero to NumTilesInPic − 1, inclusive, tile_id_val[i] may not be equal to tile_id_val[j] when i is not equal to j, and tile_id_val[i] may be less than tile_id_val[j] when j is greater than i.

[00158] An example tile group header RBSP syntax and semantics is as follows.

tile_group_header( ) {	Descriptor
tile_group_pic_parameter_set_id	ue(v)
tile_group_address	u(v)
if(NumTilesInPic > 1) {	
num_tiles_in_tile_group_minus1	ue(v)
}	
}	

[00159] The tile_group_address may specify the tile address of the first tile in the tile group. The length of tile_group_address may be tile_id_len_minus1 + 1 bits. The value of tile_group_address may be in the range of 0 to $2^{\text{tile_id_len_minus1} + 1} - 1$, inclusive, and the value of tile_group_address may not be equal to the value of tile_group_address of any other

coded tile group NAL unit of the same coded picture. The tile_group_address may be inferred to be equal to zero when not present.

[00160] FIG. 9 is a schematic diagram of an example video coding device 900. The video coding device 900 is suitable for implementing the disclosed examples/embodiments as described herein. The video coding device 900 comprises downstream ports 920, upstream ports 950, and/or transceiver units (Tx/Rx) 910, including transmitters and/or receivers for communicating data upstream and/or downstream over a network. The video coding device 900 also includes a processor 930 including a logic unit and/or central processing unit (CPU) to process the data and a memory 932 for storing the data. The video coding device 900 may also comprise electrical, optical-to-electrical (OE) components, electrical-to-optical (EO) components, and/or wireless communication components coupled to the upstream ports 950 and/or downstream ports 920 for communication of data via electrical, optical, or wireless communication networks. The video coding device 900 may also include input and/or output (I/O) devices 960 for communicating data to and from a user. The I/O devices 960 may include output devices such as a display for displaying video data, speakers for outputting audio data, etc. The I/O devices 960 may also include input devices, such as a keyboard, mouse, trackball, etc., and/or corresponding interfaces for interacting with such output devices.

[00161] The processor 930 is implemented by hardware and software. The processor 930 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 930 is in communication with the downstream ports 920, Tx/Rx 910, upstream ports 950, and memory 932. The processor 930 comprises a coding module 914. The coding module 914 implements the disclosed embodiments described herein, such as methods 100, 1000, and 1100, mechanism 600, and/or application 700 which may employ a bitstream 500 and/or an image partitioned according to flexible video tiling scheme 800. The coding module 914 may also implement any other method/mechanism described herein. Further, the coding module 914 may implement a codec system 200, an encoder 300, and/or a decoder 400. For example, the coding module 914 can partition a picture into first level tiles and partition first level tiles into second level tiles. The coding module 914 can also assign the first level tiles and the second level tiles into tile groups such that each tile group contains a number of first level tiles or a consecutive sequence of second level tiles of a single first level tile so that all second level tiles created from a single first level tile are assigned to the same tile group. The coding module 914 can also encode and/or decode such tiles in a scan order such that first level tiles are coded in raster scan order and second level tiles are coded in

raster scan order within the first level tile from which such second level tiles are partitioned. The coding module 914 further supports employing such mechanisms to combine sub-pictures at different resolutions into a single picture for various use cases as described herein. As such, the coding module 914 improves the functionality of the video coding device 900 as well as addresses problems that are specific to the video coding arts. Further, the coding module 914 effects a transformation of the video coding device 900 to a different state. Alternatively, the coding module 914 can be implemented as instructions stored in the memory 932 and executed by the processor 930 (e.g., as a computer program product stored on a non-transitory medium).

[00162] The memory 932 comprises one or more memory types such as disks, tape drives, solid-state drives, read only memory (ROM), random access memory (RAM), flash memory, ternary content-addressable memory (TCAM), static random-access memory (SRAM), etc. The memory 932 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.

[00163] FIG. 10 is a flowchart of an example method 1000 of encoding an image by employing a flexible tiling scheme, such as flexible video tiling scheme 800. Method 1000 may be employed by an encoder, such as a codec system 200, an encoder 300, and/or a video coding device 900 when performing method 100, mechanism 600, and/or supporting application 700. Further, method 1000 may be employed to generate a bitstream 500 for transmission to a decoder, such as decoder 400.

[00164] Method 1000 may begin when an encoder receives a video sequence including a plurality of images and determines to encode that video sequence into a bitstream, for example based on user input. As an example, the video sequence, and hence the images, can be encoded at a plurality of resolutions. At step 1001, a picture is partitioned into a plurality of first level tiles. A subset of the first level tiles may also be partitioned into a plurality of second level tiles. In some examples, the first level tiles outside the subset may contain picture data at a first resolution. Further, the second level tiles may contain picture data at a second resolution different from the first resolution. In some examples, each first level tile in the subset of first level tiles may include two or more complete second level tiles.

[00165] At step 1003, the first level tiles and the second level tiles are assigned into one or more tile groups. The assignment may be performed such that each tile group contains a number of first level tiles, one or more consecutive sequences of second level tiles where each sequence of second level tiles are split from a single first level tile, or combinations thereof. As a specific example, the assignment may be performed such that all second level

tiles created from a single first level tile are assigned to the same tile group. In some examples, each of the one or more tile groups may be constrained such that all tiles in an assigned tile group cover a rectangular portion of the picture.

[00166] At step 1005, the first level tiles and the second level tiles are encoded into a bitstream. For example, the first level tiles and the second level tiles may be encoded according to a scan order. In a specific example, encoding according to the scan order may include encoding the first level tiles in raster scan order. When one of the second level tiles is encountered, the raster scan order encoding of the first level tiles may be paused. All consecutive second level tiles may be encoded in raster scan order prior to continuing the raster scan order encoding of the first level tiles. For example, all second level tiles partitioned from a current first level tile may be encoded prior to encoding any second level tiles partitioned from a subsequent second level tile. At step 1007, the bitstream may be stored for communication toward a decoder.

[00167] FIG. 11 is a flowchart of an example method 1100 of decoding an image by employing a flexible tiling scheme, such as flexible video tiling scheme 800. Method 1100 may be employed by a decoder, such as a codec system 200, a decoder 400, and/or a video coding device 900 when performing method 100, mechanism 600, and/or supporting application 700. Further, method 1100 may be employed upon receiving a bitstream 500 from an encoder, such as encoder 300.

[00168] Method 1100 may begin when a decoder begins receiving a bitstream of coded data representing a video sequence, for example as a result of method 1000. The bitstream may contain video data from video sequences coded at a plurality of resolutions. At step 1101, a bitstream is received. The bitstream includes a picture partitioned into a plurality of first level tiles. A subset of the first level tiles may be further partitioned into a plurality of second level tiles. In some examples, the first level tiles outside the subset may contain picture data at a first resolution. Further, the second level tiles may contain picture data at a second resolution different from the first resolution. In another example, each first level tile in the subset of first level tiles may include two or more complete second level tiles. The first level tiles and the second level tiles are assigned to one or more tile groups. The assignment may be performed such that each tile group contains a number of first level tiles, one or more consecutive sequences of second level tiles where each sequence of second level tiles are split from a single first level tile, or combinations thereof. As a specific example, the assignment may be performed such that all second level tiles created from a single first level tile are assigned to the

same tile group. In some examples, each of the one or more tile groups may be constrained such that all tiles in an assigned tile group cover a rectangular portion of the picture.

[00169] At step 1105, the first level tiles and the second level tiles may be decoded based on the one or more tile groups. In some examples, the first level tiles and the second level tiles are decoded according to a scan order. In a specific example, decoding according to the scan order may include decoding the first level tiles in raster scan order. When one of the second level tiles is encountered, the raster scan order encoding of the first level tiles may be paused. All consecutive second level tiles may then be encoded in raster scan order prior to continuing the raster scan order decoding of the first level tiles. For example, all second level tiles partitioned from a current first level tile may be decoded prior to decoding any second level tiles partitioned from a subsequent second level tile.

[00170] At step 1107, a reconstructed video sequence may be generated for display based on the decoded first level tiles and the second level tiles.

[00171] FIG. 12 is a schematic diagram of an example system 1200 for coding a video sequence by employing a flexible tiling scheme, such as flexible video tiling scheme 800. System 1200 may be implemented by an encoder and a decoder such as a codec system 200, an encoder 300, a decoder 400, and/or a video coding device 900. Further, system 1200 may be employed when implementing method 100, 1000, 1100, mechanism 600, and/or application 700. System 1200 may also encode data into a bitstream, such as bitstream 500, and decode such a bitstream for display to a user.

[00172] The system 1200 includes a video encoder 1202. The video encoder 1202 comprises a partitioning module 1201 for partitioning a picture into a plurality of first level tiles, and partitioning a subset of the first level tiles into a plurality of second level tiles. The video encoder 1202 further comprises an assigning module 1203 for assigning the first level tiles and the second level tiles into one or more tile groups such that all second level tiles created from a single first level tile are assigned to a same tile group. The video encoder 1202 further comprises an encoding module 1205 for encoding the first level tiles and the second level tiles into a bitstream. The video encoder 1202 further comprises a storing module 1207 for storing the bitstream for communication toward a decoder. The video encoder 1202 further comprises a transmitting module 1209 for transmitting the bitstream toward a decoder. The video encoder 1202 may be further configured to perform any of the steps of method 1000.

[00173] The system 1200 also includes a video decoder 1210. The video decoder 1210 comprises a receiving module 1211 for receiving a bitstream including a picture partitioned into a plurality of first level tiles, wherein a subset of the first level tiles are further partitioned into a

plurality of second level tiles, and wherein the first level tiles and the second level tiles are assigned to one or more tile groups such that all second level tiles created from a single first level tile are assigned to a same tile group. The video decoder 1210 further comprises a decoding module 1213 for decoding the first level tiles and the second level tiles based on the one or more tile groups. The video decoder 1210 further comprises a generating module 1215 for generating a reconstructed video sequence for display based on the decoded first level tiles and the second level tiles. The video decoder 1210 may be further configured to perform any of the steps of method 1100.

[00174] A first component is directly coupled to a second component when there are no intervening components, except for a line, a trace, or another medium between the first component and the second component. The first component is indirectly coupled to the second component when there are intervening components other than a line, a trace, or another medium between the first component and the second component. The term "coupled" and its variants include both directly coupled and indirectly coupled. The use of the term "about" means a range including ±10% of the subsequent number unless otherwise stated.

[00175] It should also be understood that the steps of the exemplary methods set forth herein are not necessarily required to be performed in the order described, and the order of the steps of such methods should be understood to be merely exemplary. Likewise, additional steps may be included in such methods, and certain steps may be omitted or combined, in methods consistent with various embodiments of the present disclosure.

[00176] 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.

[00177] 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 in an encoder, the method comprising:

partitioning, by a processor of the encoder, a picture into a plurality of first level tiles;

partitioning, by the processor, a subset of the first level tiles into a plurality of second level tiles;

assigning, by the processor, the first level tiles and the second level tiles into one or more tile groups such that each tile group contains a number of first level tiles, one or more consecutive sequences of second level tiles where each sequence of second level tiles is split from a single first level tile, or combinations thereof;

encoding, by the processor, the second level tiles into a bitstream.

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

partitioning, by a processor of the encoder, a picture into a plurality of first level tiles;

partitioning, by the processor, a subset of the first level tiles into a plurality of second level tiles;

assigning, by the processor, the first level tiles and the second level tiles into one or more tile groups such that all second level tiles created from a single first level tile are assigned to a same tile group;

encoding, by the processor, the second level tiles into a bitstream.

3. The method of any of claims 1-2, the method further comprising:

storing, in a memory of the encoder, the bitstream for communication toward a decoder.

4. The method of any of claims 1-3, wherein first level tiles outside the subset contain picture data at a first resolution, and wherein second level tiles contain picture data at a second resolution different from the first resolution.

5. The method of any of claims 1-4, wherein each first level tile in the subset of first level tiles includes two or more complete second level tiles.

6. The method of any of claims 1-5, wherein all second level tiles partitioned from a current first level tile are encoded prior to encoding any second level tiles partitioned from a subsequent first level tile.

7. The method of any of claims 1-6, wherein each of the one or more tile groups are constrained such that all tiles in an assigned tile group cover a rectangular portion of the picture.

8. A method implemented in a decoder, the method comprising:

receiving, by a processor of the decoder via a receiver, a bitstream including a picture partitioned into a plurality of first level tiles, wherein a subset of the first level tiles are further partitioned into a plurality of second level tiles, and wherein the first level tiles and the second level tiles are assigned to one or more tile groups such that each tile group contains one or more consecutive sequences of second level tiles where each sequence of second level tiles is split from a single first level tile, or combinations thereof;

decoding, by the processor, the second level tiles based on parameters in the bitstream; and

generating, by the processor, a reconstructed video sequence for display based on the decoded the second level tiles.

9. A method implemented in a decoder, the method comprising:

receiving, by a receiver, a bitstream including a picture partitioned into a plurality of first level tiles, wherein a subset of the first level tiles are further partitioned into a plurality of second level tiles, and wherein the first level tiles and the second level tiles are assigned to one or more tile groups such that all second level tiles created from a single first level tile are assigned to a same tile group;

decoding, by a processor of the decoder, the second level tiles based on parameters in the bitstream; and

generating, by the processor, a reconstructed video sequence for display based on the decoded the second level tiles.

10. The method of any of claims 8-9, wherein first level tiles outside the subset contain picture data at a first resolution, and wherein second level tiles contain picture data at a second resolution different from the first resolution.

11. The method of any of claims 8-10, wherein each first level tile in the subset of first level tiles includes two or more complete second level tiles.

12. The method of any of claims 8-11, wherein all second level tiles partitioned from a current first level tile are decoded prior to decoding any second level tiles partitioned from a subsequent first level tile.

13. The method of any of claims 8-12, wherein each of the one or more tile groups are constrained such that all tiles in an assigned tile group cover a rectangular portion of the picture.

14. A video coding device comprising:

a processor, a receiver coupled to the processor, and a transmitter coupled to the processor, the processor, receiver, and transmitter configured to perform the method of any of claims 1-13.

15. A non-transitory computer readable medium comprising a computer program product for use by a video coding device, the computer program product comprising computer executable instructions stored on the non-transitory computer readable medium such that when executed by a processor cause the video coding device to perform the method of any of claims 1-13.

16. An encoder comprising:

a partitioning unit, configured to:

partitioning a picture into a plurality of first level tiles; and

partitioning a subset of the first level tiles into a plurality of second level tiles;

an assigning unit, configured to assign the first level tiles and the second level tiles into one or more tile groups such that all second level tiles created from a single first level tile are assigned to a same tile group;

an encoding unit, configured to encode the second level tiles into a bitstream.

17. The encoder of claim 16, the encoder further comprising:

a storing unit, configured to store the bitstream for communication toward a decoder.

18. The encoder of any of claims 16-17, wherein the encoder is further configured to perform the method of any of claims 1-8.

19. A decoder comprising:

a receiving unit, configured to receive a bitstream including a picture partitioned into a plurality of first level tiles, wherein a subset of the first level tiles are further partitioned into a plurality of second level tiles, and wherein the first level tiles and the second level tiles are assigned to one or more tile groups such that all second level tiles created from a single first level tile are assigned to a same tile group;

a decoding unit, configured to decode the second level tiles based on parameters in the bitstream; and

a generating unit, configured to generate a reconstructed video sequence for display based on the decoded the second level tiles.

20. The decoder of claim 19, wherein the decoder is further configured to perform the method of any of claims 9-13.

21. A non-transitory computer-readable storage medium storing processor-executable instructions configured to cause a processor to generate an encoded bitstream by performing the method of any of claims 1-7.

22. A device for coding, comprising:

receiving unit, configured to receive a bitstream to decode;

transmission unit coupled to the receiving unit, the transmission unit, configured to transmit a decoded image to a display unit;

storage unit coupled to at least one of the receiving unit or the transmission unit, the storage unit, configured to store instructions; and

processing unit coupled to the storage unit, the processing unit, configured to execute the instructions stored in the storage unit to perform the method in any of claims 1 to 13.

23. An encoder comprising processing circuitry for carrying out the method according to claims 1 to 8.

24. A decoder comprising processing circuitry for carrying out the method according to any of claims 9 to 13.

25. A computer program comprising program code for performing the method according to any of claims 1 to 13 when executed on a computer or a processor.

26. A non-transitory computer-readable medium carrying a program code which, when executed by a computer device, causes the computer device to perform the method of any one of claims 1 to 13.