ADAPTIVE BILATERAL MATCHING FOR DECODER SIDE MOTION VECTOR REFINEMENT FOR VIDEO CODING

RELATED APPLICATIONS

[0001] This application claims priority to U.S. Patent Application No. 17/929,891, filed September 6, 2022 and U.S. Provisional Application No. 63/245,577, filed September 17, 2021, the entire content of which are both incorporated by reference herein. U.S. Patent Application No. 17/929,891, filed September 6, 2022 claims the benefit of U.S. Provisional Application No. 63/245,577, filed September 17, 2021.

TECHNICAL FIELD

[0002] This disclosure relates to video encoding and video decoding.

BACKGROUND

[0003] Digital video capabilities can be incorporated into a wide range of devices, including digital televisions, digital direct broadcast systems, wireless broadcast systems, personal digital assistants (PDAs), laptop or desktop computers, tablet computers, e-book readers, digital cameras, digital recording devices, digital media players, video gaming devices, video game consoles, cellular or satellite radio telephones, so-called "smart phones," video teleconferencing devices, video streaming devices, and the like. Digital video devices implement video coding techniques, such as those described in the standards defined by MPEG-2, MPEG-4, ITU-T H.263, ITU-T H.264/MPEG-4, Part 10, Advanced Video Coding (AVC), ITU-T H.265/High Efficiency Video Coding (HEVC), ITU-T H.266/Versatile Video Coding (VVC), and extensions of such standards, as well as proprietary video codecs/formats such as AOMedia Video 1 (AV1) that was developed by the Alliance for Open Media. The video devices may transmit, receive, encode, decode, and/or store digital video information more efficiently by implementing such video coding techniques.

[0004] Video coding techniques include spatial (intra-picture) prediction and/or temporal (inter-picture) prediction to reduce or remove redundancy inherent 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 coding tree units (CTUs), coding units (CUs) and/or coding nodes. Video blocks in an intra-coded (I) slice of a picture are encoded using spatial prediction with

respect to reference samples in neighboring blocks in the same picture. Video blocks in an inter-coded (P or B) slice of a picture may use 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 reference pictures may be referred to as reference frames.

SUMMARY

[0005] In general, this disclosure describes techniques related to inter prediction in video codecs. More specifically, this disclosure describes devices and techniques related to decoder side motion vector refinement and early termination of an iterative search process for determining and/or refining a motion vector. A video encoder/decoder may be configured to determine a minimum cost through a previous search iteration and update an overall minimum cost through a current search iteration. The iterative search process may then be terminated early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration. In this way, the search iterative search process may be terminated early when insignificant improvements between successive search iterations are observed. Accordingly, the techniques of this disclosure may reduce computational complexity without materially increasing distortion.

[0006] In one example, this disclosure describes a method of decoding video data, the method comprising determining a motion vector for a block of video data using a decoder side motion vector derivation process that includes an iterative search process, wherein the iterative search process includes: determining a minimum cost through a previous search iteration; updating an overall minimum cost through a current search iteration; and terminating the iterative search process early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and decoding the block of video data using the motion vector.

[0007] In another example, this disclosure describes a method for encoding video data, the method comprising determining a motion vector for a block of video data using a decoder side motion vector derivation process that includes an iterative search process, wherein the iterative search process includes: determining a minimum cost through a previous search iteration; updating an overall minimum cost through a current search

iteration; and terminating the iterative search process early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and encoding the block of video data using the motion vector.

[0008] In another example, a device for decoding video data includes memory configured to store the video data, and processing circuitry configured to determine a motion vector for a block of the video data using a decoder side motion vector derivation process that includes an iterative search process, wherein the iterative search process includes: determining a minimum cost through a previous search iteration; updating an overall minimum cost through a current search iteration; and terminating the iterative search process early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and decode the block of video data using the motion vector.

[0009] In another example, a device for encoding video data includes memory configured to store the video data, and processing circuitry configured to determine a motion vector for a block of the video data using a decoder side motion vector derivation process that includes an iterative search process, wherein the iterative search process includes: determining a minimum cost through a previous search iteration; updating an overall minimum cost through a current search iteration; and terminating the iterative search process early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and encode the block of video data using the motion vector.

[0010] The details of one or more examples are set forth in the accompanying drawings and the description below. Other features, objects, and advantages will be apparent from the description, drawings, and claims.

BRIEF DESCRIPTION OF DRAWINGS

[0011] FIG. 1 is a block diagram illustrating an example video encoding and decoding system that may perform the techniques of this disclosure.

[0012] FIG. 2 illustrates example positions of spatial merge candidates.

[0013] FIG. 3 illustrates an example of motion vector scaling for a temporal merge candidate.

[0014] FIG. 4 illustrates example candidate positions for temporal merge candidates.

[0015] FIG. 5 illustrates an example of bilateral matching.

[0016] FIG. 6 illustrates an example of an extended coding unit region used in bi-directional optical flow.

[0017] FIG. 7 illustrates an example diamond search region for multi-pass decoder side motion refinement.

[0018] FIG. 8 illustrates an example of template matching performed on a search area around an initial motion vector.

[0019] FIG. 9 is a block diagram illustrating an example video encoder that may perform the techniques of this disclosure.

[0020] FIG. 10 is a block diagram illustrating an example video decoder that may perform the techniques of this disclosure.

[0021] FIG. 11 is a flowchart illustrating an example method for encoding a current block in accordance with the techniques of this disclosure.

[0022] FIG. 12 is a flowchart illustrating an example method for decoding a current block in accordance with the techniques of this disclosure.

[0023] FIG. 13 is a flowchart illustrating an example method for terminating an iterative search process early according to one example of the disclosure.

[0024] FIG. 14 is a flowchart illustrating another example method for decoding a current block in accordance with the techniques of this disclosure.

[0025] FIG. 15 is a flowchart illustrating another example method for encoding a current block in accordance with the techniques of this disclosure.

DETAILED DESCRIPTION

[0026] In some example inter-prediction techniques, a video decoder may be configured to perform an iterative search process to determine and/or refine a motion vector. Some example techniques include bilateral matching decoder-side motion vector refinement (DMVR) and template matching. In an example decoder side motion search process, for either bilateral matching based DMVR or template matching, a cost criterion is defined and used to calculate costs for candidate positions within a predefined search area. The motion information associated with the candidate position that has the minimum cost may be the output of the iterative search process. The search process may be conducted in multiple iterations. In each iteration, a subset of candidate positions within a search area are evaluated. Generally, the distance between the initial

position and a candidate position increases after each iteration. In other words, the candidate positions in the current search iteration may be further away from the initial point than the position of the previous iteration.

[0027] A variable minCost may represent the minimum cost calculated during the iterative search process and minCost may represent the minimum cost of the iterative search process determined so far during the iterative search process. Some early termination methods may be applied to end a search process. In some examples, the search process may be terminated if the minCost is less than a threshold, or the search process may be terminated if the best candidate position with the minimum cost was not changed from one subset to the next (e.g., from one iteration to the next).

[0028] However, in some examples, minCost calculated during the iterative search process may remain greater than the threshold while not reducing at a rate to timely decrease below the threshold. Thus, simply comparing the minCost to a threshold may lead to examples of longer decoding times and greater computational complexity in decoder side motion vector derivation while having insignificant improvements in picture distortion.

[0029] To further reduce the computational complexity and improve efficiency in decoder side motion vector derivation, a video encoder/decoder may be configured to determine a minimum cost through a previous search iteration and update an overall minimum cost through a current search iteration. In other words, a video encoder/decoder may determine a minimum cost through a previous search iteration and an overall minimum cost through a current search iteration. The iterative search process may then be terminated early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration. In some examples, determining a minimum cost through a previous search iteration may include determining a minimum cost of all previous search iterations. In some examples, terminating the iterative search process early may mean the iterative search process disclosed herein may have an ability of being terminated earlier than a normal termination of the iterative search process (i.e., the iterative search process is terminated based on a comparison of a minimum cost (for example, minCost as described above) through the iterative search iteration and a threshold). Terminating the iterative search process by comparing the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration may reduce computational complexity without materially increasing distortion.

[0030] In some examples, the iterative search process may be terminated early based on a difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration relative to a difference threshold. In some examples, if the difference is smaller than the difference threshold, the search process is terminated. In some examples, the iterative search process may be terminated early based on a ratio of the minimum cost through the previous search iteration to the overall minimum cost through the current search iteration relative to a ratio threshold. In some examples, the iterative search process may be terminated early based on a ratio of the minimum cost through the previous search iteration to the difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration relative to a ratio threshold. In some examples, the threshold may be based on a characteristic of the block being coded, such as the size of the block being coded.

[0031] In this way, the iterative search process may be terminated early when only insignificant improvements between successive search iterations are observed. By terminating an iterative search process early using the early termination techniques described above, computational complexity in decoder side motion vector derivation may be reduced without materially increasing distortion.

[0032] The early termination techniques described above may be used with any of the decoder side motion vector derivation techniques that involve a search process that has more than two iterations. In one example, the early termination techniques described above may be applied in either the bilateral matching of DMVR or template matching. In other examples. the early termination techniques described above may also be used together with other early termination methods.

[0033] FIG. 1 is a block diagram illustrating an example video encoding and decoding system 100 that may perform the techniques of this disclosure. The techniques of this disclosure are generally directed to coding (encoding and/or decoding) video data. In general, video data includes any data for processing a video. Thus, video data may include raw, unencoded video, encoded video, decoded (e.g., reconstructed) video, and video metadata, such as signaling data.

[0034] As shown in FIG. 1, system 100 includes a source device 102 that provides encoded video data to be decoded and displayed by a destination device 116, in this example. In particular, source device 102 provides the video data to destination device 116 via a computer-readable medium 110. Source device 102 and destination device

116 may comprise any of a wide range of devices, including desktop computers, notebook (e.g., laptop) computers, mobile devices, tablet computers, set-top boxes, telephone handsets such as smartphones, televisions, cameras, display devices, digital media players, video gaming consoles, video streaming device, broadcast receiver devices, or the like. In some cases, source device 102 and destination device 116 may be equipped for wireless communication, and thus may be referred to as wireless communication devices.

[0035] In the example of FIG. 1, source device 102 includes video source 104, memory 106, video encoder 200, and output interface 108. Destination device 116 includes input interface 122, video decoder 300, memory 120, and display device 118. In accordance with this disclosure, video encoder 200 of source device 102 and video decoder 300 of destination device 116 may be configured to apply the techniques for decoder side motion vector derivation. Thus, source device 102 represents an example of a video encoding device, while destination device 116 represents an example of a video decoding device. In other examples, a source device and a destination device may include other components or arrangements. For example, source device 102 may receive video data from an external video source, such as an external camera. Likewise, destination device 116 may interface with an external display device, rather than include an integrated display device.

[0036] System 100 as shown in FIG. 1 is merely one example. In general, any digital video encoding and/or decoding device may perform techniques for decoder side motion vector derivation. Source device 102 and destination device 116 are merely examples of such coding devices in which source device 102 generates coded video data for transmission to destination device 116. This disclosure refers to a "coding" device as a device that performs coding (encoding and/or decoding) of data. Thus, video encoder 200 and video decoder 300 represent examples of coding devices, in particular, a video encoder and a video decoder, respectively. In some examples, source device 102 and destination device 116 may operate in a substantially symmetrical manner such that each of source device 102 and destination device 116 includes video encoding and decoding components. Hence, system 100 may support one-way or two-way video transmission between source device 102 and destination device 116, e.g., for video streaming, video playback, video broadcasting, or video telephony.

[0037] In general, video source 104 represents a source of video data (e.g., raw, unencoded video data) and provides a sequential series of pictures (also referred to as

"frames") of the video data to video encoder 200, which encodes data for the pictures. Video source 104 of source device 102 may include a video capture device, such as a video camera, a video archive containing previously captured raw video, and/or a video feed interface to receive video from a video content provider. As a further alternative, video source 104 may generate computer graphics-based data as the source video, or a combination of live video, archived video, and computer-generated video. In each case, video encoder 200 encodes the captured, pre-captured, or computer-generated video data. Video encoder 200 may rearrange the pictures from the received order (sometimes referred to as "display order") into a coding order for coding. Video encoder 200 may generate a bitstream including encoded video data. Source device 102 may then output the encoded video data via output interface 108 onto computer-readable medium 110 for reception and/or retrieval by, e.g., input interface 122 of destination device 116.

[0038] Memory 106 of source device 102 and memory 120 of destination device 116 represent general purpose memories. In some examples, memories 106, 120 may store raw video data, e.g., raw video from video source 104 and raw, decoded video data from video decoder 300. Additionally or alternatively, memories 106, 120 may store software instructions executable by, e.g., video encoder 200 and video decoder 300, respectively. Although memory 106 and memory 120 are shown separately from video encoder 200 and video decoder 300 in this example, it should be understood that video encoder 200 and video decoder 300 may also include internal memories for functionally similar or equivalent purposes. Furthermore, memories 106, 120 may store encoded video data, e.g., output from video encoder 200 and input to video decoder 300. In some examples, portions of memories 106, 120 may be allocated as one or more video buffers, e.g., to store raw, decoded, and/or encoded video data.

[0039] Computer-readable medium 110 may represent any type of medium or device capable of transporting the encoded video data from source device 102 to destination device 116. In one example, computer-readable medium 110 represents a communication medium to enable source device 102 to transmit encoded video data directly to destination device 116 in real-time, e.g., via a radio frequency network or computer-based network. Output interface 108 may modulate a transmission signal including the encoded video data, and input interface 122 may demodulate the received transmission signal, according to a communication standard, such as a wireless communication protocol. The communication medium may comprise any wireless or wired communication medium, such as a radio frequency (RF) spectrum or one or more

physical transmission lines. The communication medium may form part of a packet-based network, such as a local area network, a wide-area network, or a global network such as the Internet. The communication medium may include routers, switches, base stations, or any other equipment that may be useful to facilitate communication from source device 102 to destination device 116.

[0040] In some examples, source device 102 may output encoded data from output interface 108 to storage device 112. Similarly, destination device 116 may access encoded data from storage device 112 via input interface 122. Storage device 112 may include any of a variety of distributed or locally accessed data storage media such as a hard drive, Blu-ray discs, DVDs, CD-ROMs, flash memory, volatile or non-volatile memory, or any other suitable digital storage media for storing encoded video data.

[0041] In some examples, source device 102 may output encoded video data to file server 114 or another intermediate storage device that may store the encoded video data generated by source device 102. Destination device 116 may access stored video data from file server 114 via streaming or download.

[0042] File server 114 may be any type of server device capable of storing encoded video data and transmitting that encoded video data to the destination device 116. File server 114 may represent a web server (e.g., for a website), a server configured to provide a file transfer protocol service (such as File Transfer Protocol (FTP) or File Delivery over Unidirectional Transport (FLUTE) protocol), a content delivery network (CDN) device, a hypertext transfer protocol (HTTP) server, a Multimedia Broadcast Multicast Service (MBMS) or Enhanced MBMS (eMBMS) server, and/or a network attached storage (NAS) device. File server 114 may, additionally or alternatively, implement one or more HTTP streaming protocols, such as Dynamic Adaptive Streaming over HTTP (DASH), HTTP Live Streaming (HLS), Real Time Streaming Protocol (RTSP), HTTP Dynamic Streaming, or the like.

[0043] Destination device 116 may access encoded video data from file server 114 through any standard data connection, including an Internet connection. This may include a wireless channel (e.g., a Wi-Fi connection), a wired connection (e.g., digital subscriber line (DSL), cable modem, etc.), or a combination of both that is suitable for accessing encoded video data stored on file server 114. Input interface 122 may be configured to operate according to any one or more of the various protocols discussed above for retrieving or receiving media data from file server 114, or other such protocols for retrieving media data.

[0044] Output interface 108 and input interface 122 may represent wireless transmitters/receivers, modems, wired networking components (e.g., Ethernet cards), wireless communication components that operate according to any of a variety of IEEE 802.11 standards, or other physical components. In examples where output interface 108 and input interface 122 comprise wireless components, output interface 108 and input interface 122 may be configured to transfer data, such as encoded video data, according to a cellular communication standard, such as 4G, 4G-LTE (Long-Term Evolution), LTE Advanced, 5G, or the like. In some examples where output interface 108 comprises a wireless transmitter, output interface 108 and input interface 122 may be configured to transfer data, such as encoded video data, according to other wireless standards, such as an IEEE 802.11 specification, an IEEE 802.15 specification (e.g., ZigBee™), a Bluetooth™ standard, or the like. In some examples, source device 102 and/or destination device 116 may include respective system-on-a-chip (SoC) devices. For example, source device 102 may include an SoC device to perform the functionality attributed to video encoder 200 and/or output interface 108, and destination device 116 may include an SoC device to perform the functionality attributed to video decoder 300 and/or input interface 122.

[0045] The techniques of this disclosure may be applied to video coding in support of any of a variety of multimedia applications, such as over-the-air television broadcasts, cable television transmissions, satellite television transmissions, Internet streaming video transmissions, such as dynamic adaptive streaming over HTTP (DASH), digital video that is encoded onto a data storage medium, decoding of digital video stored on a data storage medium, or other applications.

[0046] Input interface 122 of destination device 116 receives an encoded video bitstream from computer-readable medium 110 (e.g., a communication medium, storage device 112, file server 114, or the like). The encoded video bitstream may include signaling information defined by video encoder 200, which is also used by video decoder 300, such as syntax elements having values that describe characteristics and/or processing of video blocks or other coded units (e.g., slices, pictures, groups of pictures, sequences, or the like). Display device 118 displays decoded pictures of the decoded video data to a user. Display device 118 may represent any of a variety of display devices such as a liquid crystal display (LCD), a plasma display, an organic light emitting diode (OLED) display, or another type of display device.

[0047] Although not shown in FIG. 1, in some examples, video encoder 200 and video decoder 300 may each be integrated with an audio encoder and/or audio decoder, and may include appropriate MUX-DEMUX units, or other hardware and/or software, to handle multiplexed streams including both audio and video in a common data stream.

[0048] Video encoder 200 and video decoder 300 each may be implemented as any of a variety of suitable encoder and/or decoder circuitry, such as one or more microprocessors, digital signal processors (DSPs), application specific integrated circuits (ASICs), field programmable gate arrays (FPGAs), discrete logic, software, hardware, firmware or any combinations thereof. When the techniques are implemented partially in software, a device may store instructions for the software in a suitable, non-transitory computer-readable medium and execute the instructions in hardware using one or more processors to perform the techniques of this disclosure. Each of video encoder 200 and video decoder 300 may be included in one or more encoders or decoders, either of which may be integrated as part of a combined encoder/decoder (CODEC) in a respective device. A device including video encoder 200 and/or video decoder 300 may comprise an integrated circuit, a microprocessor, and/or a wireless communication device, such as a cellular telephone.

[0049] Video encoder 200 and video decoder 300 may operate according to a video coding standard, such as ITU-T H.265, also referred to as High Efficiency Video Coding (HEVC) or extensions thereto, such as the multi-view and/or scalable video coding extensions. Alternatively, video encoder 200 and video decoder 300 may operate according to other proprietary or industry standards, such as ITU-T H.266, also referred to as Versatile Video Coding (VVC). In other examples, video encoder 200 and video decoder 300 may operate according to a proprietary video codec/format, such as AOMedia Video 1 (AV1), extensions of AV1, and/or successor versions of AV1 (e.g., AV2). In other examples, video encoder 200 and video decoder 300 may operate according to other proprietary formats or industry standards. The techniques of this disclosure, however, are not limited to any particular coding standard or format. In general, video encoder 200 and video decoder 300 may be configured to perform the techniques of this disclosure in conjunction with any video coding techniques that use decoder side motion vector derivation techniques.

[0050] In general, video encoder 200 and video decoder 300 may perform block-based coding of pictures. The term "block" generally refers to a structure including data to be processed (e.g., encoded, decoded, or otherwise used in the encoding and/or decoding

process). For example, a block may include a two-dimensional matrix of samples of luminance and/or chrominance data. In general, video encoder 200 and video decoder 300 may code video data represented in a YUV (e.g., Y, Cb, Cr) format. That is, rather than coding red, green, and blue (RGB) data for samples of a picture, video encoder 200 and video decoder 300 may code luminance and chrominance components, where the chrominance components may include both red hue and blue hue chrominance components. In some examples, video encoder 200 converts received RGB formatted data to a YUV representation prior to encoding, and video decoder 300 converts the YUV representation to the RGB format. Alternatively, pre- and post-processing units (not shown) may perform these conversions.

[0051] This disclosure may generally refer to coding (e.g., encoding and decoding) of pictures to include the process of encoding or decoding data of the picture. Similarly, this disclosure may refer to coding of blocks of a picture to include the process of encoding or decoding data for the blocks, e.g., prediction and/or residual coding. An encoded video bitstream generally includes a series of values for syntax elements representative of coding decisions (e.g., coding modes) and partitioning of pictures into blocks. Thus, references to coding a picture or a block should generally be understood as coding values for syntax elements forming the picture or block.

[0052] HEVC defines various blocks, including coding units (CUs), prediction units (PUs), and transform units (TUs). According to HEVC, a video coder (such as video encoder 200) partitions a coding tree unit (CTU) into CUs according to a quadtree structure. That is, the video coder partitions CTUs and CUs into four equal, non-overlapping squares, and each node of the quadtree has either zero or four child nodes. Nodes without child nodes may be referred to as "leaf nodes," and CUs of such leaf nodes may include one or more PUs and/or one or more TUs. The video coder may further partition PUs and TUs. For example, in HEVC, a residual quadtree (RQT) represents partitioning of TUs. In HEVC, PUs represent inter-prediction data, while TUs represent residual data. CUs that are intra-predicted include intra-prediction information, such as an intra-mode indication.

[0053] As another example, video encoder 200 and video decoder 300 may be configured to operate according to VVC. According to VVC, a video coder (such as video encoder 200) partitions a picture into a plurality of coding tree units (CTUs). Video encoder 200 may partition a CTU according to a tree structure, such as a quadtree-binary tree (QTBT) structure or Multi-Type Tree (MTT) structure. The QTBT

structure removes the concepts of multiple partition types, such as the separation between CUs, PUs, and TUs of HEVC. A QTBT structure includes two levels: a first level partitioned according to quadtree partitioning, and a second level partitioned according to binary tree partitioning. A root node of the QTBT structure corresponds to a CTU. Leaf nodes of the binary trees correspond to coding units (CUs).

[0054] In an MTT partitioning structure, blocks may be partitioned using a quadtree (QT) partition, a binary tree (BT) partition, and one or more types of triple tree (TT) (also called ternary tree (TT)) partitions. A triple or ternary tree partition is a partition where a block is split into three sub-blocks. In some examples, a triple or ternary tree partition divides a block into three sub-blocks without dividing the original block through the center. The partitioning types in MTT (e.g., QT, BT, and TT), may be symmetrical or asymmetrical.

[0055] When operating according to the AV1 codec, video encoder 200 and video decoder 300 may be configured to code video data in blocks. In AV1, the largest coding block that can be processed is called a superblock. In AV1, a superblock can be either 128x128 luma samples or 64x64 luma samples. However, in successor video coding formats (e.g., AV2), a superblock may be defined by different (e.g., larger) luma sample sizes. In some examples, a superblock is the top level of a block quadtree. Video encoder 200 may further partition a superblock into smaller coding blocks. Video encoder 200 may partition a superblock and other coding blocks into smaller blocks using square or non-square partitioning. Non-square blocks may include N/2xN, NxN/2, N/4xN, and NxN/4 blocks. Video encoder 200 and video decoder 300 may perform separate prediction and transform processes on each of the coding blocks.

[0056] AV1 also defines a tile of video data. A tile is a rectangular array of superblocks that may be coded independently of other tiles. That is, video encoder 200 and video decoder 300 may encode and decode, respectively, coding blocks within a tile without using video data from other tiles. However, video encoder 200 and video decoder 300 may perform filtering across tile boundaries. Tiles may be uniform or non-uniform in size. Tile-based coding may enable parallel processing and/or multi-threading for encoder and decoder implementations.

[0057] In some examples, video encoder 200 and video decoder 300 may use a single QTBT or MTT structure to represent each of the luminance and chrominance components, while in other examples, video encoder 200 and video decoder 300 may use two or more QTBT or MTT structures, such as one QTBT/MTT structure for the

luminance component and another QTBT/MTT structure for both chrominance components (or two QTBT/MTT structures for respective chrominance components).

[0058] Video encoder 200 and video decoder 300 may be configured to use quadtree partitioning, QTBT partitioning, MTT partitioning, superblock partitioning, or other partitioning structures.

[0059] In some examples, a CTU includes a coding tree block (CTB) of luma samples, two corresponding CTBs of chroma samples of a picture that has three sample arrays, or a CTB of samples of a monochrome picture or a picture that is coded using three separate color planes and syntax structures used to code the samples. A CTB may be an NxN block of samples for some value of N such that the division of a component into CTBs is a partitioning. A component is an array or single sample from one of the three arrays (luma and two chroma) that compose a picture in 4:2:0, 4:2:2, or 4:4:4 color format or the array or a single sample of the array that compose a picture in monochrome format. In some examples, a coding block is an MxN block of samples for some values of M and N such that a division of a CTB into coding blocks is a partitioning.

[0060] The blocks (e.g., CTUs or CUs) may be grouped in various ways in a picture. As one example, a brick may refer to a rectangular region of CTU rows within a particular tile in a picture. A tile may be a rectangular region of CTUs within a particular tile column and a particular tile row in a picture. A tile column refers to a rectangular region of CTUs having a height equal to the height of the picture and a width specified by syntax elements (e.g., such as in a picture parameter set). A tile row refers to a rectangular region of CTUs having a height specified by syntax elements (e.g., such as in a picture parameter set) and a width equal to the width of the picture.

[0061] In some examples, a tile may be partitioned into multiple bricks, each of which may include one or more CTU rows within the tile. A tile that is not partitioned into multiple bricks may also be referred to as a brick. However, a brick that is a true subset of a tile may not be referred to as a tile. The bricks in a picture may also be arranged in a slice. A slice may be an integer number of bricks of a picture that may be exclusively contained in a single network abstraction layer (NAL) unit. In some examples, a slice includes either a number of complete tiles or only a consecutive sequence of complete bricks of one tile.

[0062] This disclosure may use "NxN" and "N by N" interchangeably to refer to the sample dimensions of a block (such as a CU or other video block) in terms of vertical

and horizontal dimensions, e.g., 16x16 samples or 16 by 16 samples. In general, a 16x16 CU will have 16 samples in a vertical direction (y = 16) and 16 samples in a horizontal direction (x = 16). Likewise, an NxN CU generally has N samples in a vertical direction and N samples in a horizontal direction, where N represents a nonnegative integer value. The samples in a CU may be arranged in rows and columns. Moreover, CUs need not necessarily have the same number of samples in the horizontal direction as in the vertical direction. For example, CUs may comprise NxM samples, where M is not necessarily equal to N.

[0063] Video encoder 200 encodes video data for CUs representing prediction and/or residual information, and other information. The prediction information indicates how the CU is to be predicted in order to form a prediction block for the CU. The residual information generally represents sample-by-sample differences between samples of the CU prior to encoding and the prediction block.

[0064] To predict a CU, video encoder 200 may generally form a prediction block for the CU through inter-prediction or intra-prediction. Inter-prediction generally refers to predicting the CU from data of a previously coded picture, whereas intra-prediction generally refers to predicting the CU from previously coded data of the same picture. To perform inter-prediction, video encoder 200 may generate the prediction block using one or more motion vectors. Video encoder 200 may generally perform a motion search to identify a reference block that closely matches the CU, e.g., in terms of differences between the CU and the reference block. Video encoder 200 may calculate a difference metric using a sum of absolute difference (SAD), sum of squared differences (SSD), mean absolute difference (MAD), mean squared differences (MSD), or other such difference calculations to determine whether a reference block closely matches the current CU. In some examples, video encoder 200 may predict the current CU using uni-directional prediction or bi-directional prediction.

[0065] Some examples of VVC also provide an affine motion compensation mode, which may be considered an inter-prediction mode. In affine motion compensation mode, video encoder 200 may determine two or more motion vectors that represent non-translational motion, such as zoom in or out, rotation, perspective motion, or other irregular motion types.

[0066] To perform intra-prediction, video encoder 200 may select an intra-prediction mode to generate the prediction block. Some examples of VVC provide sixty-seven intra-prediction modes, including various directional modes, as well as planar mode and

DC mode. In general, video encoder 200 selects an intra-prediction mode that describes neighboring samples to a current block (e.g., a block of a CU) from which to predict samples of the current block. Such samples may generally be above, above and to the left, or to the left of the current block in the same picture as the current block, assuming video encoder 200 codes CTUs and CUs in raster scan order (left to right, top to bottom).

[0067] Video encoder 200 encodes data representing the prediction mode for a current block. For example, for inter-prediction modes, video encoder 200 may encode data representing which of the various available inter-prediction modes is used, as well as motion information for the corresponding mode. For uni-directional or bi-directional inter-prediction, for example, video encoder 200 may encode motion vectors using advanced motion vector prediction (AMVP) or merge mode. Video encoder 200 may use similar modes to encode motion vectors for affine motion compensation mode.

[0068] AV1 includes two general techniques for encoding and decoding a coding block of video data. The two general techniques are intra prediction (e.g., intra frame prediction or spatial prediction) and inter prediction (e.g., inter frame prediction or temporal prediction). In the context of AV1, when predicting blocks of a current frame of video data using an intra prediction mode, video encoder 200 and video decoder 300 do not use video data from other frames of video data. For most intra prediction modes, video encoder 200 encodes blocks of a current frame based on the difference between sample values in the current block and predicted values generated from reference samples in the same frame. Video encoder 200 determines predicted values generated from the reference samples based on the intra prediction mode.

[0069] Following prediction, such as intra-prediction or inter-prediction of a block, video encoder 200 may calculate residual data for the block. The residual data, such as a residual block, represents sample by sample differences between the block and a prediction block for the block, formed using the corresponding prediction mode. Video encoder 200 may apply one or more transforms to the residual block, to produce transformed data in a transform domain instead of the sample domain. For example, video encoder 200 may apply a discrete cosine transform (DCT), an integer transform, a wavelet transform, or a conceptually similar transform to residual video data. Additionally, video encoder 200 may apply a secondary transform following the first transform, such as a mode-dependent non-separable secondary transform (MDNSST), a signal dependent transform, a Karhunen-Loeve transform (KLT), or the like. Video

encoder 200 produces transform coefficients following application of the one or more transforms.

[0070] As noted above, following any transforms to produce transform coefficients, video encoder 200 may perform quantization of the transform coefficients. Quantization generally refers to a process in which transform coefficients are quantized to possibly reduce the amount of data used to represent the transform coefficients, providing further compression. By performing the quantization process, video encoder 200 may reduce the bit depth associated with some or all of the transform coefficients. For example, video encoder 200 may round an n-bit value down to an m-bit value during quantization, where n is greater than m. In some examples, to perform quantization, video encoder 200 may perform a bitwise right-shift of the value to be quantized.

[0071] Following quantization, video encoder 200 may scan the transform coefficients, producing a one-dimensional vector from the two-dimensional matrix including the quantized transform coefficients. The scan may be designed to place higher energy (and therefore lower frequency) transform coefficients at the front of the vector and to place lower energy (and therefore higher frequency) transform coefficients at the back of the vector. In some examples, video encoder 200 may utilize a predefined scan order to scan the quantized transform coefficients to produce a serialized vector, and then entropy encode the quantized transform coefficients of the vector. In other examples, video encoder 200 may perform an adaptive scan. After scanning the quantized transform coefficients to form the one-dimensional vector, video encoder 200 may entropy encode the one-dimensional vector, e.g., according to context-adaptive binary arithmetic coding (CABAC). Video encoder 200 may also entropy encode values for syntax elements describing metadata associated with the encoded video data for use by video decoder 300 in decoding the video data.

[0072] To perform CABAC, video encoder 200 may assign a context within a context model to a symbol to be transmitted. The context may relate to, for example, whether neighboring values of the symbol are zero-valued or not. The probability determination may be based on a context assigned to the symbol.

[0073] Video encoder 200 may further generate syntax data, such as block-based syntax data, picture-based syntax data, and sequence-based syntax data, to video decoder 300, e.g., in a picture header, a block header, a slice header, or other syntax data, such as a sequence parameter set (SPS), picture parameter set (PPS), or video parameter set

(VPS). Video decoder 300 may likewise decode such syntax data to determine how to decode corresponding video data.

[0074] In this manner, video encoder 200 may generate a bitstream including encoded video data, e.g., syntax elements describing partitioning of a picture into blocks (e.g., CUs) and prediction and/or residual information for the blocks. Ultimately, video decoder 300 may receive the bitstream and decode the encoded video data.

[0075] In general, video decoder 300 performs a reciprocal process to that performed by video encoder 200 to decode the encoded video data of the bitstream. For example, video decoder 300 may decode values for syntax elements of the bitstream using CABAC in a manner substantially similar to, albeit reciprocal to, the CABAC encoding process of video encoder 200. The syntax elements may define partitioning information for partitioning of a picture into CTUs, and partitioning of each CTU according to a corresponding partition structure, such as a QTBT structure, to define CUs of the CTU. The syntax elements may further define prediction and residual information for blocks (e.g., CUs) of video data.

[0076] The residual information may be represented by, for example, quantized transform coefficients. Video decoder 300 may inverse quantize and inverse transform the quantized transform coefficients of a block to reproduce a residual block for the block. Video decoder 300 uses a signaled prediction mode (intra- or inter-prediction) and related prediction information (e.g., motion information for inter-prediction) to form a prediction block for the block. Video decoder 300 may then combine the prediction block and the residual block (on a sample-by-sample basis) to reproduce the original block. Video decoder 300 may perform additional processing, such as performing a deblocking process to reduce visual artifacts along boundaries of the block.

[0077] This disclosure may generally refer to "signaling" certain information, such as syntax elements. The term "signaling" may generally refer to the communication of values for syntax elements and/or other data used to decode encoded video data. That is, video encoder 200 may signal values for syntax elements in the bitstream. In general, signaling refers to generating a value in the bitstream. As noted above, source device 102 may transport the bitstream to destination device 116 substantially in real time, or not in real time, such as might occur when storing syntax elements to storage device 112 for later retrieval by destination device 116.

[0078] The techniques of this disclosure relate to inter prediction in video codecs. More specifically, this disclosure describes devices and techniques related to decoder side

motion vector refinement using bilateral matching. In accordance with the techniques of this disclosure, as will be described in more detail below, video decoder 300 may include memory configured to store the video data; and processing circuitry that may be configured to determine a motion vector for a block of video data using a decoder side motion vector derivation process that includes an iterative search process, wherein the iterative search process includes: determining a minimum cost through a previous search iteration; updating an overall minimum cost through a current search iteration; and terminating the iterative search process early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, and decode the block of video data using the motion vector.

[0079] In accordance with the techniques of this disclosure, as will be described in more detail below, video encoder 200 may include memory configured to store the video data; and processing circuitry that may be configured to determine a motion vector for a block of video data using a decoder side motion vector derivation process that includes an iterative search process, wherein the iterative search process includes: determining a minimum cost through a previous search iteration; updating an overall minimum cost through a current search iteration; and terminating the iterative search process early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, and encode the block of video data using the motion vector.

[0080] **Bilateral Matching**

[0081] Bilateral matching (BM) is a technique that refines a pair of initial motion vectors MV0 and MV1. Generally, BM includes searching around the MV0 and MV1 motion vectors to derive refined MVs MV0' and MV1' that minimize the block matching cost. The block matching cost measures the similarity between the two motion compensated predictors generated using the two MVs (some typical criterions are: sum of absolute difference (SAD), sum of absolute transformed difference (SATD), sum of square error (SSE), et al). The block matching cost may also include a regularization term that is derived based on the MV differences between the current MV pair and the initial MV pair. Some constraints may also be applied to the MV differences between MVD0 (MV0'-MV0) and MVD1 (MV1'-MV1). Typically, video coders are configured with the assumption that MVD0 and MVD1 shall be proportional to the temporal distances (TD) between the current picture and the reference pictures

pointed to by the two MVs. However, in some applications, the BM process may be designed with the assumption that MVD0 is equal to – MVD1.

[0082] **Inter Prediction in Versatile Video Coding Standard**

[0083] For inter-predicted CUs in VVC, motion parameters include motion vectors, reference picture indices, and reference picture list usage index, as well as any additional information needed for new coding feature of VVCs. The motion parameters are used for inter-predicted sample generation. The motion parameters can be signalled in an explicit or implicit manner. When a CU is coded with skip mode, the CU is associated with one PU and has no significant residual coefficients, no coded motion vector delta, or reference picture index. A merge mode is specified whereby the motion parameters for the current CU are obtained from neighbouring CUs, including spatial and temporal candidates, as well as additional schedules introduced in VVC. The merge mode can be applied to any inter-predicted CU, not only for skip mode. Some other prediction modes besides merge mode include the explicit transmission of motion parameters, where a motion vector (e.g., a motion vector difference), a corresponding reference picture index for each reference picture list, and a reference picture list usage flag and other needed information are signalled explicitly for each CU.

[0084] Beyond the inter coding features in HEVC, VVC includes a number of new and refined inter prediction coding tools listed as follows:

- Extended merge prediction
- Merge mode with MVD (MMVD)
- Symmetric MVD (SMVD) signalling
- Affine motion compensated prediction
- Subblock-based temporal motion vector prediction (SbTMVP)
- Adaptive motion vector resolution (AMVR)
- Motion field storage: 1/16th luma sample MV storage and 8x8 motion field compression
- Bi-prediction with CU-level weight (BCW)
- Bi-directional optical flow (BDOF)
- Decoder side motion vector refinement (DMVR)
- Geometric partitioning mode (GPM)
- Combined inter and intra prediction (CIIP)

[0085] **Extended Merge Prediction**

[0086] In VVC regular merge mode, the merge candidate list is constructed by including the following five types of candidates, in order:

1) Spatial motion vector predictor (MVP) from spatial neighbour CUs;
2) Temporal MVP from collocated CUs;
3) History-based MVP from a first-in, first-out (FIFO) table;
4) Pairwise average MVP; and
5) Zero motion vectors (MVs).

[0087] The size of the merge list is signalled in a sequence parameter set (SPS) header. In one example, the maximum allowed size of the merge list is 6 candidates. For each CU coded in merge mode, an index of the selected merge candidate is encoded using truncated unary binarization (TU).

[0088] The derivation process of each category of merge candidates is provided in this section. As is done in HEVC, VVC also supports parallel derivation of the merging candidate lists for all CUs within a certain size of area.

[0089] **Spatial Candidates Derivation**

[0090] The derivation of spatial merge candidates in VVC is the same as that in HEVC, except the positions of first two merge candidates are swapped. FIG. 2 illustrates example positions of spatial merge candidates. A maximum of four merge candidates are selected among candidates located in the positions depicted in FIG. 2 relative to current CU 400. The order of derivation is: B_0, A_0, B_1, A_1 and B_2. Position B_2 is considered only when one or more the CUs at positions B_0, A_0, B_1, A_1 are not available (e.g. because the CU belongs to another slice or tile) or the CU is intra coded. After the candidate at position A_1 is added, the addition of the remaining candidates is subject to a redundancy check which ensures that candidates with same motion information are excluded from the list so that coding efficiency is improved.

[0091] **Temporal Candidates Derivation**

[0092] In this technique, only one candidate is added to the list. Particularly, in the derivation of this temporal merge candidate, a scaled motion vector is derived based on co-located CU belonging to the collocated reference picture. FIG. 3 illustrates an example of motion vector scaling for a temporal merge candidate. The reference picture list to be used for derivation of the co-located CU is explicitly signaled in the slice header. The scaled motion vector 412 for temporal merge candidate is obtained as illustrated by the dotted line in FIG. 3, which is scaled from the motion vector of the co-located CU 416 using the picture order count (POC) distances, tb and td. Motion vector

414 is the motion vector between current CU 400 and co-located CU 416. Distance tb is defined to be the POC difference between the reference picture 404 of the current picture and the current picture 406. Distance td is defined to be the POC difference between the reference picture 402 of the co-located picture and the co-located picture 408. The reference picture index of temporal merge candidate is set equal to zero.

[0093] FIG. 4 illustrates example candidate positions for temporal merge candidates. The position for the temporal candidate for current CU 400 is selected between candidates C_0 and C_1, as depicted in FIG 4. If the CU at position C_0 is not available (e.g., is intra coded or is outside of the current row of CTUs), position C_1 is used. Otherwise, position C_0 is used in the derivation of the temporal merge candidate.

[0094] **History-Based Merge Candidates Derivation**

[0095] The history-based MVP (HMVP) merge candidates are added to the merge list after the spatial merge candidates (e.g., spatial MVP) and the temporal merge candidate (e.g., temporal MVP or TMVP). In HMVP, the motion information of a previously coded block is stored in a table and used as MVP for the current CU. The table with multiple HMVP candidates is maintained during the encoding/decoding process. The table is reset (e.g., emptied) when a new CTU row is encountered. Whenever there is a non-subblock inter-coded CU, the associated motion information is added to the last entry of the table as a new HMVP candidate.

[0096] The HMVP table size S is set to be 6, which indicates up to 6 HMVP candidates may be added to the table. When inserting a new motion candidate to the table, a constrained first-in-first-out (FIFO) rule is utilized, wherein a redundancy check is first applied to find whether there is an identical HMVP in the table. If found, the identical HMVP is removed from the table and all the HMVP candidates afterwards are moved forward.

[0097] HMVP candidates may be used in the merge candidate list construction process. The latest several HMVP candidates in the table are checked in order and inserted to the candidate list after the TMVP candidate. A redundancy check is applied on the HMVP candidates to compare the motion information of the HMVP candidate to the spatial or temporal merge candidate.

[0098] To reduce the number of redundancy check operations, the following simplifications may be introduced:

a. The number of HMVP candidates used for merge list generation is set as $(N <= 4)\ ?\ M : (8 - N)$, wherein N indicates number of existing

candidates in the merge list and M indicates number of available HMVP candidates in the table.

b. Once the total number of available merge candidates reaches the maximum allowed merge candidates minus 1, the merge candidate list construction process from HMVP is terminated.

[0099] **Pairwise Average Merge Candidate Derivation**

[0100] Pairwise average candidates are generated by averaging predefined pairs of candidates in the existing merge candidate list. In one example, the predefined pairs are defined as {(0, 1), (0, 2), (1, 2), (0, 3), (1, 3), (2, 3)}, where the numbers denote the merge indices to the merge candidate list. The averaged motion vectors are calculated separately for each reference list. If both motion vectors are available in one list, these two motion vectors are averaged even when both point to different reference pictures; if only one motion vector is available, use the one directly; if no motion vector is available, keep this list invalid.

[0101] When the merge list is not full after pairwise average merge candidates are added, the zero MVPs are inserted in the end of the list until the maximum merge candidate number is reached.

[0102] **Bi-prediction with CU-level weight (BCW)**

[0103] In HEVC, the bi-prediction signal is generated by averaging two prediction signals obtained from two different reference pictures and/or using two different motion vectors. In VVC, the bi-prediction mode is extended beyond simple averaging to allow weighted averaging of the two prediction signals.

$$P_{\text{bi-pred}} = \left((8 - w) * P_0 + w * P_1 + 4\right) \gg 3$$

[0104] Five weights are allowed in the weighted averaging bi-prediction, $w \in \{-2, 3, 4, 5, 10\}$. For each bi-predicted CU, the weight w is determined in one of two ways: 1) for a non-merge CU, the weight index is signalled after the motion vector difference; 2) for a merge CU, the weight index is inferred from neighbouring blocks based on the merge candidate index. BCW is only applied to CUs with 256 or more luma samples (e.g., CU width times CU height is greater than or equal to 256). For low-delay pictures, all 5 weights are used. For non-low-delay pictures, only 3 weights ($w \in \{3,4,5\}$) are used.

– At the encoder, fast search algorithms are applied to find the weight index without significantly increasing the encoder complexity. These algorithms are summarized

as follows. Further details are available from the VVC Test Model (VTM) software and in Yu-Chi Su, et al. "CE4-related: Generalized bi-prediction improvements combined from JVET-L0197 and JVET-L0296," Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 12th Meeting: Macao, CN, 3-12 October 2018 (hereinafter, "JVET-L0646"). When combined with AMVR, unequal weights are only conditionally checked for 1-pel and 4-pel motion vector precisions if the current picture is a low-delay picture.

- When combined with affine motion estimation (ME), affine ME will be performed for unequal weights if and only if the affine mode is selected as the current best mode.
- When the two reference pictures in bi-prediction are the same, unequal weights are only conditionally checked.
- Unequal weights are not searched when certain conditions are met, depending on the POC distance between current picture and its reference pictures, the coding QP, and the temporal level.

[0105] The BCW weight index is coded using one context coded bin followed by bypass coded bins. The first context coded bin indicates if equal weight is used; and if unequal weight is used, additional bins are signalled using bypass coding to indicate which unequal weight is used.

[0106] Weighted prediction (WP) is a coding tool supported by the H.264/AVC and HEVC standards to efficiently code video content with fading. Support for WP was also added into the VVC standard. WP allows weighting parameters (weight and offset) to be signalled for each reference picture in each of the reference picture lists L0 and L1. Then, during motion compensation, the weight(s) and offset(s) of the corresponding reference picture(s) are applied. WP and BCW are designed for different types of video content. In order to avoid interactions between WP and BCW, which will complicate VVC decoder design, if a CU uses WP, then the BCW weight index is not signalled, and w is inferred to be 4 (e.g., equal weight is applied). For a merge CU, the weight index is inferred from neighbouring blocks based on the merge candidate index. This can be applied to both normal merge mode and inherited affine merge mode. For constructed affine merge mode, the affine motion information is constructed based on the motion information of up to 3 blocks. The BCW index for a CU using the constructed affine merge mode is simply set equal to the BCW index of the first control point MV.

[0107] In VVC, CIIP and BCW cannot be jointly applied for a CU. When a CU is coded with CIIP mode, the BCW index of the current CU is set to 2, e.g. equal weight.

[0108] **Decoder -Side Motion Vector Refinement in VVC**

[0109] In VVC, BM based decoder side motion vector refinement (DMVR) may be applied to increase the accuracy of the MVs of a bi-prediction merge candidate. The BM method calculates the sum of absolute differences (SAD) between the two candidate blocks in the reference picture list L0 and list L1. FIG. 5 illustrates an example of bilateral matching. As illustrated in FIG. 5, the SAD between blocks 500 and 502, based on each MV candidate around the initial MV, is calculated. The MV candidate with the lowest SAD becomes the refined MV and used to generate the bi-predicted signal. The SAD of the initial MVs is subtracted by ¼ of the SAD value to serve as regularization term. The temporal distances (e.g., the Picture Order Count (POC) difference) from two reference pictures to the current picture shall be the same, therefore, the MVD0 is just the opposite sign of MVD1.

[0110] The refinement search range is two integer luma samples from the initial MV. The searching includes the integer sample offset search stage and fractional sample refinement stage. 25 points full search is applied for integer sample offset searching. The SAD of the initial MV pair is first calculated. If the SAD of the initial MV pair is smaller than a threshold, the integer sample stage of DMVR is terminated. Otherwise, SADs of the remaining 24 points are calculated and checked in raster scanning order. The point with the smallest SAD is selected as the output of integer sample offset searching stage.

[0111] The integer sample search is followed by fractional sample refinement. To reduce calculational complexity, the fractional sample refinement is derived by using parametric error surface equation, instead of using an additional search with a SAD comparison. The fractional sample refinement is conditionally invoked based on the output of the integer sample search stage. When the integer sample search stage is terminated with the center having the smallest SAD in either the first iteration or the second iteration search, the fractional sample refinement is further applied.

[0112] In parametric error surface based sub-pixel offsets estimation, the center position cost and the costs at four neighboring positions from the center are used to fit a 2-D parabolic error surface equation of the following form:

$$E(x, y) = A(x - x_{min})^2 + B(y - y_{min})^2 + C, \quad (1)$$

where (x_{min}, y_{min}) corresponds to the fractional position with the least cost and C corresponds to the minimum cost value. By solving the above equations using the cost value of the five search points, the (x_{min}, y_{min}) is computed as:

$$x_{min} = (E(-1,0) - E(1,0))/(2(E(-1,0) + E(1,0) - 2E(0,0))) \quad (2)$$

$$y_{min} = (E(0,-1) - E(0,1))/(2((E(0,-1) + E(0,1) - 2E(0,0)) \quad (3)$$

[0113] The value of x_{min} and y_{min} are automatically constrained to be between − 8 and 8 since all cost values are positive and the smallest value is $E(0,0)$. This corresponds to half-pel offset with 1/16th-pel MV accuracy in VVC. The computed fractional (x_{min}, y_{min}) are added to the integer distance refinement MV to get the sub-pixel accurate refinement delta MV.

[0114] In VVC, the resolution of the MVs is 1/16 luma samples. The samples at the fractional position are interpolated using an 8-tap interpolation filter. In DMVR, the search points surround the initial fractional-pel MV with integer sample offset. Therefore, the samples of those fractional position are interpolated for the DMVR search process. To reduce the calculation complexity, the bi-linear interpolation filter is used to generate the fractional samples for the searching process in DMVR. Another effect is that, by using a bi-linear filter with 2-sample search range, the DVMR does not access more reference samples compared to the normal motion compensation process. After the refined MV is attained with DMVR search process, the normal 8-tap interpolation filter is applied to generate the final prediction. In order to not access more reference samples relative to a normal motion compensation process, the samples which are not needed for the interpolation process based on the original MV, but are needed for the interpolation process based on the refined MV, will be padded from those available samples.

[0115] When the width and/or height of a CU are larger than 16 luma samples, the CU will be further split into subblocks with width and/or height equal to 16 luma samples for the DMVR process.

[0116] In VVC, DMVR can be applied for the CUs which are coded with following modes and features:

- CU level merge mode with bi-prediction MV

- One reference picture is in the past and another reference picture is in the future with respect to the current picture
- The distances (e.g., POC difference) from two reference pictures to the current picture are same
- Both reference pictures are short-term reference pictures
- CU has more than 64 luma samples
- Both CU height and CU width are larger than or equal to 8 luma samples
- BCW weight index indicates equal weight
- WP is not enabled for the current block
- CIIP mode is not used for the current block

[0117] **Bi-Directional Optical Flow**

[0118] Bi-directional optical flow (BDOF) is used to refine the bi-prediction signal of luma samples in a CU at the 4×4 sub-block level. As its name indicates, the BDOF mode is based on the optical flow concept, which assumes that the motion of an object is smooth. For each 4×4 sub-block, a motion refinement (v_x, v_y) is calculated by minimizing the difference between the L0 and L1 prediction samples. The motion refinement is then used to adjust the bi-predicted sample values in the 4x4 sub-block. The following steps are applied in the BDOF process.

[0119] First, the horizontal and vertical gradients, $\frac{\partial I^{(k)}}{\partial x}(i,j)$ and $\frac{\partial I^{(k)}}{\partial y}(i,j)$, $k = 0,1$, of the two prediction signals are computed by directly calculating the difference between two neighboring samples, e.g.,

$$\frac{\partial I^{(k)}}{\partial x}(i,j) = \left(I^{(k)}(i+1,j) \gg shift1\right) - \left(I^{(k)}(i-1,j) \gg shift1\right)$$
$$\frac{\partial I^{(k)}}{\partial y}(i,j) = \left(I^{(k)}(i,j+1) \gg shift1\right) - \left(I^{(k)}(i,j-1) \gg shift1\right), \quad (1\text{-}6\text{-}1)$$

where $I^{(k)}(i,j)$ are the sample value at coordinate (i,j) of the prediction signal in list k, $k = 0,1$, and shift1 is calculated based on the luma bit depth, bitDepth, as shift1 is set to be equal to 6.

[0120] Then, the auto- and cross-correlation of the gradients, S_1, S_2, S_3, S_5 and S_6, are calculated as:

$$S_1 = \sum_{(i,j)\in\Omega} \left|\psi_x(i,j)\right|, \quad S_3 = \sum_{(i,j)\in\Omega} \theta(i,j) \cdot \left(-sign\left(\psi_x(i,j)\right)\right) \quad (1\text{-}6\text{-}2)$$

$$S_2 = \sum_{(i,j)\in\Omega} \psi_x(i,j) \cdot sign\left(\psi_y(i,j)\right)$$

$$S_5 = \sum_{(i,j)\in\Omega} \left|\psi_y(i,j)\right| \quad S_6 = \sum_{(i,j)\in\Omega} \theta(i,j) \cdot \left(-sign\left(\psi_y(i,j)\right)\right),$$

where

$$\psi_x(i,j) = \left(\frac{\partial I^{(1)}}{\partial x}(i,j) + \frac{\partial I^{(0)}}{\partial x}(i,j)\right) \gg shift3$$

$$\psi_y(i,j) = \left(\frac{\partial I^{(1)}}{\partial y}(i,j) + \frac{\partial I^{(0)}}{\partial y}(i,j)\right) \gg shift3 \qquad (1\text{-}6\text{-}3)$$

$$\theta(i,j) = \left(I^{(0)}(i,j) \gg shift2\right) - \left(I^{(1)}(i,j) \gg shift2\right),$$

where Ω is a 6×6 window around the 4×4 sub-block, the value of shift2 is set to be equal to 4, and the value of shift3 is set to be equal to 1.

[0121] The motion refinement (v_x, v_y) is then derived using the cross- and auto-correlation terms using the following equations:

$$v_x = S_1 > 0 ? clip3\left(-th'_{BIO}, th'_{BIO}, -\left((S_3 \ll 2) \gg \lfloor \log_2 S_1 \rfloor\right)\right) : 0$$

$$v_y = S_5 > 0 ? clip3\left(-th'_{BIO}, th'_{BIO}, -\left(\left((S_6 \ll 2) - \left((v_x \cdot S_2) \gg 1\right)\right) \gg \lfloor \log_2 S_5 \rfloor\right)\right) : 0, \qquad (1\text{-}6\text{-}4)$$

where, $th'_{BIO} = 1 \ll 4$. $\lfloor \cdot \rfloor$ is the floor function.

[0122] Based on the motion refinement and the gradients, the following adjustment is calculated for each sample in the 4×4 sub-block:

$$b(x,y) = v_x \cdot \left(\frac{\partial I^{(1)}(x,y)}{\partial x} - \frac{\partial I^{(0)}(x,y)}{\partial x}\right) + v_y \cdot \left(\frac{\partial I^{(1)}(x,y)}{\partial y} - \frac{\partial I^{(0)}(x,y)}{\partial y}\right) \qquad (1\text{-}6\text{-}5)$$

[0123] Finally, the BDOF samples of the CU are calculated by adjusting the bi-prediction samples as follows:

$$pred_{BDOF}(x,y) = \left(I^{(0)}(x,y) + I^{(1)}(x,y) + b(x,y) + o_{offset}\right) \gg shift5, \qquad (1\text{-}6\text{-}6)$$

where shift5 is set equal to Max(3, 15 − BitDepth) and the variable o_{offset} is set equal to (1 << (shift5− 1)).

[0124] These values are selected such that the multipliers in the BDOF process do not exceed 15-bits, and the maximum bit-width of the intermediate parameters in the BDOF process is kept within 32-bits.

[0125] In order to derive the gradient values, some prediction samples $I^{(k)}(i,j)$ in list k ($k = 0,1$) outside of the current CU boundaries may be generated. FIG. 6 illustrates an example of an extended coding unit region used in bi-directional optical flow. As depicted in FIG. 6, the BDOF uses one extended row/column around the boundaries of CU 600. In order to control the computational complexity of generating the out-of-boundary prediction samples, prediction samples in the extended area (white positions) are generated by taking the reference samples at the nearby integer positions (using floor() operation on the coordinates) directly without interpolation, and the normal 8-tap motion compensation interpolation filter is used to generate prediction samples within the CU (gray positions). These extended sample values are used in gradient calculation only. For the remaining steps in the BDOF process, if any sample and gradient values outside of the CU boundaries are needed, they are padded (e.g., repeated) from their nearest neighbors.

[0126] BDOF is used to refine the bi-prediction signal of a CU at the 4×4 subblock level. BDOF is applied to a CU if all the following conditions are satisfied:

- The CU is coded using "true" bi-prediction mode, e.g., one of the two reference pictures is prior to the current picture in display order and the other is after the current picture in display order
- The CU is not coded using affine mode or the SbTMVP merge mode
- CU has more than 64 luma samples
- Both CU height and CU width are larger than or equal to 8 luma samples
- BCW weight index indicates equal weight
- WP is not enabled for the current CU
- CIIP mode is not used for the current CU

[0127] **<u>Multi-Pass Decoder Side Motion Refinement</u>**

[0128] In the JVET-V meeting, an enhanced compression model (ECM, https://vcgit.hhi.fraunhofer.de/ecm/VVCSoftware_VTM/-/tree/ECM) was established to study the compression techniques beyond VVC. In ECM, a multi-pass decoder side motion refinement technique, described in Yao-Jen Chang, et al. "Compression efficiency beyond VVC," Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 21st Meeting, by teleconference, 6-15 January 2021

(hereinafter, "JVET_U0100") was adopted to replace the DMVR in VVC. In the first pass, BM (similar to that of DMVR in VVC) is applied to the coding block. In the second pass, BM is applied to each 16x16 subblock within the coding block with the refined MVs in the first pass as initial MVs. In the third pass, an MV in each 8x8 subblock is further refined by applying bi-directional optical flow (BDOF).

[0129] **First Pass – Block-based bilateral matching MV refinement**

[0130] In the first pass, a refined MV is derived by applying BM to a coding block. Similar to decoder side motion vector refinement (DMVR), in bi-prediction operations, a refined MV is searched around the two initial MVs (MV0 and MV1) in the reference picture lists L0 and L1. The refined MVs (MV0_pass1 and MV1_pass1) are derived around the initial MVs based on the minimum bilateral matching cost between the two reference blocks in L0 and L1.

[0131] BM includes performing a local search to derive integer sample precision intDeltaMV. The local search applies a 3×3 square search pattern to loop through the search range [–sHor, sHor] in the horizontal direction and [–sVer, sVer] in the vertical direction, wherein the values of sHor and sVer are determined by the block dimension, and the maximum value of sHor and sVer is 8.

[0132] The bilateral matching cost is calculated as: bilCost = mvDistanceCost + sadCost. When the block size cbW * cbH is greater than 64, a mean-removed SAD (MRSAD) cost function is applied to remove the DC effect of distortion between reference blocks. When the bilCost at the center point of the 3×3 search pattern has the minimum cost, the intDeltaMV local search is terminated. Otherwise, the current minimum cost search point becomes the new center point of the 3×3 search pattern and the video coder continues to search for the minimum cost until the end of the search range is reached.11

[0133] The existing fractional sample refinement is further applied to derive the final deltaMV. The refined MVs after the first pass is then derived as:

- MV0_pass1 = MV0 + deltaMV
- MV1_pass1 = MV1 – deltaMV

[0134] **Second Pass – Subblock based bilateral matching MV refinement**

[0135] In the second pass, a refined MV is derived by applying BM to a 16×16 grid subblock. For each subblock, a refined MV is searched around the two MVs (MV0_pass1 and MV1_pass1), obtained on the first pass, in the reference picture list L0 and L1. The refined MVs (MV0_pass2(sbIdx2) and MV1_pass2(sbIdx2)), wherein

sbIdx2 = 0, …, N- 1, is the index for subblock, are derived based on the minimum bilateral matching cost between the two reference subblocks in L0 and L1.

[0136] For each subblock, BM includes performing a full search to derive integer sample precision intDeltaMV. The full search has a search range [–sHor, sHor] in horizontal direction and [– sVer, sVer] in vertical direction, wherein the values of sHor and sVer are determined by the block dimension, and the maximum value of sHor and sVer is 8.

[0137] The bilateral matching cost is calculated by applying a cost factor to the sum of absolute transformed differences (SATD) cost between two reference subblocks as: bilCost = satdCost * costFactor. FIG. 7 illustrates an example diamond search region 700 for multi-pass decoder side motion refinement. The search area (2*sHor + 1) * (2*sVer + 1) is divided into 5 diamond shaped search regions, as shown in FIG. 7. Each search region is assigned a costFactor, which is determined by the distance (intDeltaMV) between each search point and the starting MV. Each diamond region is processed in the order starting from the center of the search area. I n each region, the search points are processed in the raster scan order starting from the top left going to the bottom right corner of the region. When the minimum bilCost within the current search region is less than a threshold equal to sbW * sbH, the int-pel full search is terminated. Otherwise, the int-pel full search continues to the next search region until all search points are examined.

[0138] The existing VVC DMVR fractional sample refinement is further applied to derive the final deltaMV(sbIdx2) . The refined MVs at second pass is then derived as:

- MV0_pass2(sbIdx2) = MV0_pass1 + deltaMV(sbIdx2)
- MV1_pass2(sbIdx2) = MV1_pass1 – deltaMV(sbIdx2)

[0139] **<u>Third Pass – Subblock based Bi-Directional Optical Flow MV Refinement</u>**

[0140] In the third pass, a refined MV is derived by applying BDOF to an 8×8 grid subblock. For each 8×8 subblock, BDOF refinement is applied to derive scaled Vx and Vy values, without clipping, starting from the refined MV of the parent subblock of the second pass. The derived bioMv(Vx, Vy) is rounded to 1/16 sample precision and clipped between -32 and 32.

[0141] The refined MVs (MV0_pass3(sbIdx3) and MV1_pass3(sbIdx3)) at the third pass are derived as:

- MV0_pass3(sbIdx3) = MV0_pass2(sbIdx2) + bioMv
- MV1_pass3(sbIdx3) = MV0_pass2(sbIdx2) – bioMv

More details of multi-pass DMVR can be found in co-pending U.S. Provisional Patent Application 63/129,221, filed December 22, 2020.

[0142] **Template Matching**

[0143] Template matching (TM) is a decoder-side MV derivation method that refines the motion information of the current CU by finding the closest match between a template (e.g., top and/or left neighbouring blocks of the current CU) in the current picture and a block (e.g., same size as the template) in a reference picture. FIG. 8 illustrates an example of template matching performed on a search area around an initial motion vector. As illustrated in FIG. 8, a better MV is to be searched around the initial motion 800 of the current CU 802 within a [– 8, +8]-pel search range 806 in reference frame 808. With an AMVP candidate selected based on initial matching error, the AMVP MVP is refined by template matching. With a merge candidate indicated by signaled merge index, its merged MVs corresponding to L0 and L1 are refined independently by template matching and then the less accurate motion vector is further refined again with the better motion vector as an initial motion vector (e.g., a prior motion vector).

[0144] **Cost function**: When a motion vector points to a fractional sample position, motion compensated interpolation may be used. To reduce complexity, bi-linear interpolation, instead of regular 8-tap DCT-IF interpolation, is used for template matching to generate templates on reference pictures. The matching cost C of template matching is calculated as follows:

$$C = \mathrm{SAD} + w * (|\,\mathrm{MVx} - \mathrm{MV}^{s}\mathrm{x}\,| + |\,\mathrm{MVy} - MV^{s}y\,|),$$

where w is a weighting factor which can be set to an integer number such as 0, 1, 2, 3 or 4, and MV and MV^{s} indicate the currently tested MV and the initial MV (e.g., a MVP candidate in AMVP mode or merged motion in merge mode), respectively. SAD is used as the matching cost of template matching.

[0145] When TM is used, motion information is refined by using luma samples only. The derived motion information will be used for both luma and chroma for motion compensated inter prediction. After the MV is decided, final motion compensation is performed using an 8-tap interpolation filter for luma components and a 4-tap interpolation filter for chroma components.

[0146] **Search method:** MV refinement is a pattern-based MV search with the criterion of template matching cost and a hierarchical structure. Two search patterns are supported – a diamond search and a cross search for MV refinement. The hierarchical

structure specifies an iterative process to refine the MV, starting at a coarse MVD precision (e.g., quarter-pel) and ending at a fine MVD precision (e.g., 1/8-pel). The MV is directly searched at quarter luma sample MVD precision with a diamond pattern, followed by quarter luma sample MVD precision with a cross pattern. This is followed by one-eighth luma sample MVD refinement with a cross pattern. The search range of MV refinement is set equal to (-8, +8) luma samples around the initial MV. When the current block is coded using bi-prediction, both MVs are refined independently, and then the best of which (in terms of matching cost) is set as a prior MV to further refine the other MV with BCW weight values.

[0147] Specifically, the template matching proposed in Y-W Chen, et. al "Description of SDR, HDR and 360° video coding technology proposal by Qualcomm and Technicolor – low and high complexity versions," Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 10th Meeting: San Diego, US, 10-20 April 2018 (hereinafter, "JVET-J0021") and JVET-U0100 is applied to AMVP mode and merge mode in some example proposals.

[0148] The regular merge candidate list is constructed in the order of the following candidates:

- Spatial MVPs from spatial neighbour CUs: first 4 available candidates are selected among the order of Above neighbour CU (A), Left neighbour CU (L), Above-Right neighbour CU (AR), Left-Below neighbour CU (LB), Left-Above neighbour CU (LA).
- Temporal MVP from collocated CUs: Only one candidate is added.
- History-based MVP: The motion information of a previously coded block is stored in a table and used as MVP for the current CU.
- Pairwise average MVP: Pairwise average candidates are generated by averaging predefined pairs of candidates in the existing regular merge candidate list
- Zero MVs.

[0149] When TM is applied to blocks coded using merge mode, a separate TM merge candidate list is constructed by refining the MVPs of the regular merge candidates based on the template matching.

[0150] In the decoder side motion search process, for either bilateral matching based DMVR or template matching, a cost criterion may be defined and used to calculate costs for candidate positions within a predefined search area. The motion information

associated with the candidate position that has the minimum cost is the output of the iterative search process. The search process may be conducted in multiple iterations. In each iteration, a subset of candidate positions within a search area may be evaluated. Certain patterns for selecting the candidate positions for each subset may be applied. For examples, the square search, diamond search, and the diamond region in FIG. 7. Generally, the distance between the initial position and a candidate position increases after each iteration. In other words, the candidate positions in the current search iteration may be further away from the initial point than the position of the previous iteration.

[0151] A variable minCost may be defined to keep track of the minimum cost. The variable minCost may represent the minimum calculated during the iterative search process. This disclosure may refer to minCost as the minimum cost of the iterative search process determined so far. Some early termination methods may be applied to end a search process. For example, the process is terminated if the minCost is less than a threshold, or the process is terminated if the best candidate position with the minimum cost was not changed from one subset to the next (e.g., from one iteration to the next).

[0152] This disclosure describes techniques for early termination of a search process for decoder side motion vector derivation techniques to reduce the computational complexity of the search process. The techniques of this disclosure may be used with any of the decoder side motion vector derivation techniques that involve a search process. In one example, the techniques of this disclosure may be applied in either the bilateral matching of DMVR or the template matching. In other examples. the techniques of this disclosure may also be used together with other early termination methods, for example, as described above.

[0153] In one example of the disclosure, video encoder 200 and video decoder 300 may calculate the value of a second variable minCostPrev to keep track of the minimum cost in a previous search iteration. In one example, the maximum number of iterations shall be larger than 2 for this example to be applied. In one example, video encoder 200 and video decoder 300 may initialize the value of minCostPrev with a predetermined value, for example, zero.

[0154] Video decoder 300 may determine a minimum cost through a previous search iteration and an overall minimum cost through a current search iteration. In some examples, the previous search iteration may immediately precede the current search iteration. For example, if the current search iteration is a 4^{th} search iteration, the

previous search iteration would be the 3rd search iteration. In this example, a minimum cost through a previous search iteration would be the minimum cost through the 3rd search iteration, while an overall minimum cost through a current search iteration would be the minimum cost through the 4th search iteration.

[0155] Video encoder 200 may determine a minimum cost through a previous search iteration and an overall minimum cost through a current search iteration. In some examples, the previous search iteration may immediately precede the current search iteration. For example, if the current search iteration is a 4th search iteration, the previous search iteration would be the 3rd search iteration. In this example, a minimum cost through the previous search iteration would be the minimum cost through the 3rd search iteration, while an overall minimum cost through a current search iteration would be the minimum cost through the 4th search iteration.

[0156] In another example, a minimum cost of a previous search iteration itself may be used in place of a minimum cost through the previous search iteration. For example, if the current search iteration is a 4th search iteration, the previous search iteration would be the 3rd search iteration. In this example, a minimum cost of the 3rd search iteration itself may be used in place of the minimum cost through the 3rd search iteration.

[0157] In some examples to determine a motion vector for a block of the video data using a decoder side motion vector derivation process, video decoder 300 and/or video encoder 200 may be configured to terminate an iterative search process early in accordance with techniques discussed in this disclosure.

[0158] In the disclosed termination techniques, video encoder 200 and video decoder 300 may be configured to consider minCost and minCostPrev to make a decision whether the iterative search process is terminated. In one example, video encoder 200 and video decoder 300 are configured to compare the difference of minCostPrev – minCost or abs(minCostPrev – minCost) to a threshold, and if the difference is smaller than the threshold, video encoder 200 and video decoder 300 terminates the iterative search process. Upon video encoder 200 and video decoder 300 determining to terminate the iterative search process, video encoder 200 and video decoder 300 may be configured to return the motion vector that has the minimum cost up to that point in the search process.

[0159] Accordingly, in an example, video encoder 200 and video decoder 300 may be configured to terminate the iterative search process early based on a difference of the minimum cost through the previous search iteration and the overall minimum cost

through the current search iteration relative to a difference threshold. For example, if the current search iteration is the 4th search iteration and the previous search iteration is the 3rd search iteration, and if a difference between the minimum cost through the 4th search iteration and the minimum cost through the 3rd search iteration is below a difference threshold, the iterative search process is terminated. In some examples, the difference threshold may be fixed or based on a size of the block of video data. In some examples, the threshold may be set equal to S times the area of the block of video data where S is a scaling factor that may be predefined or signaled in the bitstream.

[0160] In some examples, video encoder 200 and video decoder 300 may be configured to terminate the iterative search process early based on comparing an absolute value of the difference between the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration to the difference threshold.

[0161] In other examples, video encoder 200 and video decoder 300 terminating the iterative search process early when the difference between the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration is below a difference threshold may reduce computation complexity without materially increasing distortion.

[0162] Accordingly, in one example of the disclosure, video decoder 300 may be configured to determine a motion vector for a block of video data using a decoder side motion vector derivation process that includes an iterative search process, wherein the iterative search process includes calculating a minimum cost of a previous search iteration, and terminating the iterative search process early based on the minimum cost through the previous search iteration. Video decoder 300 may then code the block of video data using the motion vector. In a further example, video decoder 300 may update a minimum cost determined for the iterative search process. To terminate the iterative search process early based on the minimum cost through the previous search iteration, video decoder 300 may further calculate a difference or a ratio of the cost through the previous search iteration relative to the minimum cost determined for the iterative search process, and terminate the iterative search process early based on the difference or the ratio relative to a threshold.

[0163] In one example, the disclosed techniques can be described as follows. Video encoder 200 and video decoder 300 may initialize minCostPrev and minCost, for example, to have the same value. In another example, the initialization value may be input from the previous process or set equal to a predefined large value. While the

iterative search process is not terminated and the maximum iterations is not reached, video encoder 200 and video decoder 300 may perform the following iteration:

1) Set minCostPrev equal to minCost.
2) Evaluate the current subset of candidate positions and determine (e.g., update) the minCost.
3) Determine whether to terminate based upon both the values of minCost and minCostPrev.

[0164] Note that some other early termination techniques may also be applied after step 2). For example other early termination techniques may be applied before step 3) or after.

[0165] In one example, step 3) above includes evaluating the difference between minCost and minCostPrev. Video encoder 200 and video decoder 300 may determine to terminate the search process if minCostPrev – minCost is less than a threshold.

[0166] In the above examples, the threshold may be a predefined value, fixed or signaled in the bitstream. In another examples, the threshold may be adaptive based on the size of the input block. For example, the threshold may be set equal to S times the area of the input block, wherein S is a scaling factor and can be predefined or signaled in the bitstream.

[0167] In another example, the difference of the two costs used above may be replaced with the cost ratio. For example, step 3) is conducted by evaluating the relationship between minCost and minCostPrev. The value R is set equal to (minCostPrev – minCost) / minCostPrev. Video encoder 200 and video decoder 300 may determine to terminate the search process if R is less than a predefined threshold.

[0168] Accordingly, in an example, video encoder 200 and video decoder 300 may be configured to terminate the iterative search process early may terminate the iterative search process early based on a ratio of the minimum cost through the previous search iteration to the overall minimum cost through the current search iteration relative to a ratio threshold. For example, if the current search iteration is the 4^{th} search iteration and the previous search iteration is the 3^{rd} search iteration, if ratio of the minimum cost through the 4^{th} search iteration to the minimum cost through the 3^{rd} search iteration is below a ratio threshold, the iterative search process is terminated. In some examples, the ratio threshold may be fixed or based on a size of the block of video data.

[0169] In some examples, terminating the search process early when the ratio of the overall minimum cost through the current search iteration to the minimum cost through

the previous search iteration is below a ratio threshold may reduce computation complexity without materially increasing distortion.

[0170] In some examples, the iterative search process may be terminated early based on a ratio of the minimum cost through the previous search iteration to the difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration relative to a ratio threshold. In some examples, an absolute value of the difference between the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration may be used in the ration. For example, if the current search iteration is the 4^{th} search iteration and the previous search iteration is the 3^{rd} search iteration, a difference between the minimum cost through the 4^{th} search iteration and the minimum cost through the 3^{rd} search iteration is determined, and a ratio of the difference to the minimum cost through the 3^{rd} search iteration is determined. In some examples, if the ratio is below a ratio threshold, the iterative search process is terminated. In some examples, the ratio threshold may be fixed or based on a size of the block of video data.

[0171] In some examples, if the iterative search process is not terminated early and another search iteration occurs, the minimum cost through the current search iteration becomes the minimum cost through the previous search iteration for the next search iteration. For example, if the current search iteration is the 4^{th} search iteration and the previous search iteration is the 3^{rd} search iteration and the iterative search process does not end early, a 5^{th} search iteration is conducted. For the 5^{th} search iteration, the minimum cost through the 4^{th} search iteration becomes the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration is updated to the minimum cost through the 5^{th} search iteration. Accordingly, the overall minimum cost through the current search iteration and the minimum cost through the previous search iteration may be updated for each search iteration.

[0172] In another example, the step 3) is only applied if the number of previous iterations exceeds a certain predefined threshold.

[0173] In another example, the cost of the current search position, denoted as currCost, can be additionally considered with minCost and minCostPrev to make a decision whether to terminate the iterative search process. For example, abs(currCost – minCostPrev) can be evaluated, for example by comparing to thresholds, and if the current cost currCost is not much different from minCostPrev, the search may be terminated. In another example, an additional condition is added where

abs(minCostPrev – minCost) is compared against a threshold, and if less than the threshold and if abs(currCost – minCostPrev) is less than another threshold then the iterative search process is terminated.

[0174] For example, if the current search iteration is the 4th search iteration and the previous search iteration is the 3rd search iteration, the cost of the current search position would be the cost of the 4th search iteration itself, while a minimum cost through a previous search iteration would be the minimum cost through the 3rd search iteration, and an overall minimum cost through a current search iteration would be the minimum cost through the 4th search iteration.

[0175] In some examples, video encoder 200 and video decoder 300 may be configured to compare a difference between a cost of the current search position and the minimum cost through the previous search iteration to a difference threshold. In some examples, video encoder 200 and video decoder 300 may be configured to terminate the iterative search process early if the difference between a cost of the current search position and the minimum cost through the previous search iteration is below a difference threshold.

[0176] In some examples, video encoder 200 and video decoder 300 may be configured to determine a difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration and compare the difference to a difference threshold, as discussed above. In addition, video encoder 200 and video decoder 300 may be configured to compare a difference between a cost of the current search position and the minimum cost through the previous search iteration to a difference threshold. In some examples, video encoder 200 and video decoder 300 may be configured to terminate the iterative search process early when both differences are below a respective difference threshold. In some examples, the respective difference thresholds may be equal. In some examples, the respective difference thresholds may be different and/or independent.

[0177] In some examples, the respective difference thresholds may be fixed or based on a size of the block of video data. In some examples, the respective difference thresholds may respectively be set equal to S times the area of the block of video data where S is a scaling factor that may be predefined or signaled in the bitstream.

[0178] In some examples, video encoder 200 and video decoder 300 terminating an iterative search process early, in accordance with one or more of the examples discussed above, may reduce computation complexity without materially increasing distortion.

[0179] FIG. 9 is a block diagram illustrating an example video encoder 200 that may perform the techniques of this disclosure. FIG. 9 is provided for purposes of explanation and should not be considered limiting of the techniques as broadly exemplified and described in this disclosure. For purposes of explanation, this disclosure describes video encoder 200 according to the techniques of VVC (ITU-T H.266, under development), and HEVC (ITU-T H.265). However, the techniques of this disclosure may be performed by video encoding devices that are configured to other video coding standards and video coding formats, such as AV1 and successors to the AV1 video coding format.

[0180] In the example of FIG. 9, video encoder 200 includes video data memory 230, mode selection unit 202, residual generation unit 204, transform processing unit 206, quantization unit 208, inverse quantization unit 210, inverse transform processing unit 212, reconstruction unit 214, filter unit 216, decoded picture buffer (DPB) 218, and entropy encoding unit 220. Any or all of video data memory 230, mode selection unit 202, residual generation unit 204, transform processing unit 206, quantization unit 208, inverse quantization unit 210, inverse transform processing unit 212, reconstruction unit 214, filter unit 216, DPB 218, and entropy encoding unit 220 may be implemented in one or more processors or in processing circuitry. For instance, the units of video encoder 200 may be implemented as one or more circuits or logic elements as part of hardware circuitry, or as part of a processor, ASIC, or FPGA. Moreover, video encoder 200 may include additional or alternative processors or processing circuitry to perform these and other functions.

[0181] Video data memory 230 may store video data to be encoded by the components of video encoder 200. Video encoder 200 may receive the video data stored in video data memory 230 from, for example, video source 104 (FIG. 1). DPB 218 may act as a reference picture memory that stores reference video data for use in prediction of subsequent video data by video encoder 200. Video data memory 230 and DPB 218 may be formed by any of a variety of memory devices, such as dynamic random access memory (DRAM), including synchronous DRAM (SDRAM), magnetoresistive RAM (MRAM), resistive RAM (RRAM), or other types of memory devices. Video data memory 230 and DPB 218 may be provided by the same memory device or separate memory devices. In various examples, video data memory 230 may be on-chip with other components of video encoder 200, as illustrated, or off-chip relative to those components.

[0182] In this disclosure, reference to video data memory 230 should not be interpreted as being limited to memory internal to video encoder 200, unless specifically described as such, or memory external to video encoder 200, unless specifically described as such. Rather, reference to video data memory 230 should be understood as reference memory that stores video data that video encoder 200 receives for encoding (e.g., video data for a current block that is to be encoded). Memory 106 of FIG. 1 may also provide temporary storage of outputs from the various units of video encoder 200.

[0183] The various units of FIG. 9 are illustrated to assist with understanding the operations performed by video encoder 200. The units may be implemented as fixed-function circuits, programmable circuits, or a combination thereof. Fixed-function circuits refer to circuits that provide particular functionality, and are preset on the operations that can be performed. Programmable circuits refer to circuits that can be programmed to perform various tasks, and provide flexible functionality in the operations that can be performed. For instance, programmable circuits may execute software or firmware that cause the programmable circuits to operate in the manner defined by instructions of the software or firmware. Fixed-function circuits may execute software instructions (e.g., to receive parameters or output parameters), but the types of operations that the fixed-function circuits perform are generally immutable. In some examples, one or more of the units may be distinct circuit blocks (fixed-function or programmable), and in some examples, one or more of the units may be integrated circuits.

[0184] Video encoder 200 may include arithmetic logic units (ALUs), elementary function units (EFUs), digital circuits, analog circuits, and/or programmable cores, formed from programmable circuits. In examples where the operations of video encoder 200 are performed using software executed by the programmable circuits, memory 106 (FIG. 1) may store the instructions (e.g., object code) of the software that video encoder 200 receives and executes, or another memory within video encoder 200 (not shown) may store such instructions.

[0185] Video data memory 230 is configured to store received video data. Video encoder 200 may retrieve a picture of the video data from video data memory 230 and provide the video data to residual generation unit 204 and mode selection unit 202. Video data in video data memory 230 may be raw video data that is to be encoded.

[0186] Mode selection unit 202 includes a motion estimation unit 222, a motion compensation unit 224, and an intra-prediction unit 226. Mode selection unit 202 may

include additional functional units to perform video prediction in accordance with other prediction modes. As examples, mode selection unit 202 may include a palette unit, an intra-block copy unit (which may be part of motion estimation unit 222 and/or motion compensation unit 224), an affine unit, a linear model (LM) unit, or the like.

[0187] Mode selection unit 202 generally coordinates multiple encoding passes to test combinations of encoding parameters and resulting rate-distortion values for such combinations. The encoding parameters may include partitioning of CTUs into CUs, prediction modes for the CUs, transform types for residual data of the CUs, quantization parameters for residual data of the CUs, and so on. Mode selection unit 202 may ultimately select the combination of encoding parameters having rate-distortion values that are better than the other tested combinations.

[0188] Video encoder 200 may partition a picture retrieved from video data memory 230 into a series of CTUs, and encapsulate one or more CTUs within a slice. Mode selection unit 202 may partition a CTU of the picture in accordance with a tree structure, such as the MTT structure, QTBT structure. superblock structure, or the quad-tree structure described above. As described above, video encoder 200 may form one or more CUs from partitioning a CTU according to the tree structure. Such a CU may also be referred to generally as a "video block" or "block."

[0189] In general, mode selection unit 202 also controls the components thereof (e.g., motion estimation unit 222, motion compensation unit 224, and intra-prediction unit 226) to generate a prediction block for a current block (e.g., a current CU, or in HEVC, the overlapping portion of a PU and a TU). For inter-prediction of a current block, motion estimation unit 222 may perform a motion search to identify one or more closely matching reference blocks in one or more reference pictures (e.g., one or more previously coded pictures stored in DPB 218). In particular, motion estimation unit 222 may calculate a value representative of how similar a potential reference block is to the current block, e.g., according to sum of absolute difference (SAD), sum of squared differences (SSD), mean absolute difference (MAD), mean squared differences (MSD), or the like. Motion estimation unit 222 may generally perform these calculations using sample-by-sample differences between the current block and the reference block being considered. Motion estimation unit 222 may identify a reference block having a lowest value resulting from these calculations, indicating a reference block that most closely matches the current block.

[0190] Motion estimation unit 222 may form one or more motion vectors (MVs) that defines the positions of the reference blocks in the reference pictures relative to the position of the current block in a current picture. Motion estimation unit 222 may form a motion vector using any of the techniques for determining a motion vector, as described above. For example, motion estimation unit 222 may be configured to determine a motion vector for a block of video data using a decoder side motion vector derivation process that includes an iterative search process. The iterative search process including determining a minimum cost through a previous search iteration; updating an overall minimum cost through a current search iteration; and terminating the iterative search process early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration. The motion estimation unit 222 may further be configured to encode the block of video data using the motion vector.

[0191] Motion estimation unit 222 may then provide the motion vectors to motion compensation unit 224. For example, for uni-directional inter-prediction, motion estimation unit 222 may provide a single motion vector, whereas for bi-directional inter-prediction, motion estimation unit 222 may provide two motion vectors. Motion compensation unit 224 may then generate a prediction block using the motion vectors. For example, motion compensation unit 224 may retrieve data of the reference block using the motion vector. As another example, if the motion vector has fractional sample precision, motion compensation unit 224 may interpolate values for the prediction block according to one or more interpolation filters. Moreover, for bi-directional inter-prediction, motion compensation unit 224 may retrieve data for two reference blocks identified by respective motion vectors and combine the retrieved data, e.g., through sample-by-sample averaging or weighted averaging.

[0192] When operating according to the AV1 video coding format, motion estimation unit 222 and motion compensation unit 224 may be configured to encode coding blocks of video data (e.g., both luma and chroma coding blocks) using translational motion compensation, affine motion compensation, overlapped block motion compensation (OBMC), and/or compound inter-intra prediction.

[0193] As another example, for intra-prediction, or intra-prediction coding, intra-prediction unit 226 may generate the prediction block from samples neighboring the current block. For example, for directional modes, intra-prediction unit 226 may generally mathematically combine values of neighboring samples and populate these

calculated values in the defined direction across the current block to produce the prediction block. As another example, for DC mode, intra-prediction unit 226 may calculate an average of the neighboring samples to the current block and generate the prediction block to include this resulting average for each sample of the prediction block.

[0194] When operating according to the AV1 video coding format, intra prediction unit 226 may be configured to encode coding blocks of video data (e.g., both luma and chroma coding blocks) using directional intra prediction, non-directional intra prediction, recursive filter intra prediction, chroma-from-luma (CFL) prediction, intra block copy (IBC), and/or color palette mode. Mode selection unit 202 may include additional functional units to perform video prediction in accordance with other prediction modes.

[0195] Mode selection unit 202 provides the prediction block to residual generation unit 204. Residual generation unit 204 receives a raw, unencoded version of the current block from video data memory 230 and the prediction block from mode selection unit 202. Residual generation unit 204 calculates sample-by-sample differences between the current block and the prediction block. The resulting sample-by-sample differences define a residual block for the current block. In some examples, residual generation unit 204 may also determine differences between sample values in the residual block to generate a residual block using residual differential pulse code modulation (RDPCM). In some examples, residual generation unit 204 may be formed using one or more subtractor circuits that perform binary subtraction.

[0196] In examples where mode selection unit 202 partitions CUs into PUs, each PU may be associated with a luma prediction unit and corresponding chroma prediction units. Video encoder 200 and video decoder 300 may support PUs having various sizes. As indicated above, the size of a CU may refer to the size of the luma coding block of the CU and the size of a PU may refer to the size of a luma prediction unit of the PU. Assuming that the size of a particular CU is 2Nx2N, video encoder 200 may support PU sizes of 2Nx2N or NxN for intra prediction, and symmetric PU sizes of 2Nx2N, 2NxN, Nx2N, NxN, or similar for inter prediction. Video encoder 200 and video decoder 300 may also support asymmetric partitioning for PU sizes of 2NxnU, 2NxnD, nLx2N, and nRx2N for inter prediction.

[0197] In examples where mode selection unit 202 does not further partition a CU into PUs, each CU may be associated with a luma coding block and corresponding chroma

coding blocks. As above, the size of a CU may refer to the size of the luma coding block of the CU. The video encoder 200 and video decoder 300 may support CU sizes of 2Nx2N, 2NxN, or Nx2N.

[0198] For other video coding techniques such as an intra-block copy mode coding, an affine-mode coding, and linear model (LM) mode coding, as some examples, mode selection unit 202, via respective units associated with the coding techniques, generates a prediction block for the current block being encoded. In some examples, such as palette mode coding, mode selection unit 202 may not generate a prediction block, and instead generate syntax elements that indicate the manner in which to reconstruct the block based on a selected palette. In such modes, mode selection unit 202 may provide these syntax elements to entropy encoding unit 220 to be encoded.

[0199] As described above, residual generation unit 204 receives the video data for the current block and the corresponding prediction block. Residual generation unit 204 then generates a residual block for the current block. To generate the residual block, residual generation unit 204 calculates sample-by-sample differences between the prediction block and the current block.

[0200] Transform processing unit 206 applies one or more transforms to the residual block to generate a block of transform coefficients (referred to herein as a "transform coefficient block"). Transform processing unit 206 may apply various transforms to a residual block to form the transform coefficient block. For example, transform processing unit 206 may apply a discrete cosine transform (DCT), a directional transform, a Karhunen-Loeve transform (KLT), or a conceptually similar transform to a residual block. In some examples, transform processing unit 206 may perform multiple transforms to a residual block, e.g., a primary transform and a secondary transform, such as a rotational transform. In some examples, transform processing unit 206 does not apply transforms to a residual block.

[0201] When operating according to AV1, transform processing unit 206 may apply one or more transforms to the residual block to generate a block of transform coefficients (referred to herein as a "transform coefficient block"). Transform processing unit 206 may apply various transforms to a residual block to form the transform coefficient block. For example, transform processing unit 206 may apply a horizontal/vertical transform combination that may include a discrete cosine transform (DCT), an asymmetric discrete sine transform (ADST), a flipped ADST (e.g., an ADST in reverse order), and an identity transform (IDTX). When using an identity transform, the

transform is skipped in one of the vertical or horizontal directions. In some examples, transform processing may be skipped.

[0202] Quantization unit 208 may quantize the transform coefficients in a transform coefficient block, to produce a quantized transform coefficient block. Quantization unit 208 may quantize transform coefficients of a transform coefficient block according to a quantization parameter (QP) value associated with the current block. Video encoder 200 (e.g., via mode selection unit 202) may adjust the degree of quantization applied to the transform coefficient blocks associated with the current block by adjusting the QP value associated with the CU. Quantization may introduce loss of information, and thus, quantized transform coefficients may have lower precision than the original transform coefficients produced by transform processing unit 206.

[0203] Inverse quantization unit 210 and inverse transform processing unit 212 may apply inverse quantization and inverse transforms to a quantized transform coefficient block, respectively, to reconstruct a residual block from the transform coefficient block. Reconstruction unit 214 may produce a reconstructed block corresponding to the current block (albeit potentially with some degree of distortion) based on the reconstructed residual block and a prediction block generated by mode selection unit 202. For example, reconstruction unit 214 may add samples of the reconstructed residual block to corresponding samples from the prediction block generated by mode selection unit 202 to produce the reconstructed block.

[0204] Filter unit 216 may perform one or more filter operations on reconstructed blocks. For example, filter unit 216 may perform deblocking operations to reduce blockiness artifacts along edges of CUs. Operations of filter unit 216 may be skipped, in some examples.

[0205] When operating according to AV1, filter unit 216 may perform one or more filter operations on reconstructed blocks. For example, filter unit 216 may perform deblocking operations to reduce blockiness artifacts along edges of CUs. In other examples, filter unit 216 may apply a constrained directional enhancement filter (CDEF), which may be applied after deblocking, and may include the application of non-separable, non-linear, low-pass directional filters based on estimated edge directions. Filter unit 216 may also include a loop restoration filter, which is applied after CDEF, and may include a separable symmetric normalized Wiener filter or a dual self-guided filter.

[0206] Video encoder 200 stores reconstructed blocks in DPB 218. For instance, in examples where operations of filter unit 216 are not performed, reconstruction unit 214 may store reconstructed blocks to DPB 218. In examples where operations of filter unit 216 are performed, filter unit 216 may store the filtered reconstructed blocks to DPB 218. Motion estimation unit 222 and motion compensation unit 224 may retrieve a reference picture from DPB 218, formed from the reconstructed (and potentially filtered) blocks, to inter-predict blocks of subsequently encoded pictures. In addition, intra-prediction unit 226 may use reconstructed blocks in DPB 218 of a current picture to intra-predict other blocks in the current picture.

[0207] In general, entropy encoding unit 220 may entropy encode syntax elements received from other functional components of video encoder 200. For example, entropy encoding unit 220 may entropy encode quantized transform coefficient blocks from quantization unit 208. As another example, entropy encoding unit 220 may entropy encode prediction syntax elements (e.g., motion information for inter-prediction or intra-mode information for intra-prediction) from mode selection unit 202. Entropy encoding unit 220 may perform one or more entropy encoding operations on the syntax elements, which are another example of video data, to generate entropy-encoded data. For example, entropy encoding unit 220 may perform a context-adaptive variable length coding (CAVLC) operation, a CABAC operation, a variable-to-variable (V2V) length coding operation, a syntax-based context-adaptive binary arithmetic coding (SBAC) operation, a Probability Interval Partitioning Entropy (PIPE) coding operation, an Exponential-Golomb encoding operation, or another type of entropy encoding operation on the data. In some examples, entropy encoding unit 220 may operate in bypass mode where syntax elements are not entropy encoded.

[0208] Video encoder 200 may output a bitstream that includes the entropy encoded syntax elements needed to reconstruct blocks of a slice or picture. In particular, entropy encoding unit 220 may output the bitstream.

[0209] In accordance with AV1, entropy encoding unit 220 may be configured as a symbol-to-symbol adaptive multi-symbol arithmetic coder. A syntax element in AV1 includes an alphabet of N elements, and a context (e.g., probability model) includes a set of N probabilities. Entropy encoding unit 220 may store the probabilities as n-bit (e.g., 15-bit) cumulative distribution functions (CDFs). Entropy encoding unit 22 may perform recursive scaling, with an update factor based on the alphabet size, to update the contexts.

[0210] The operations described above are described with respect to a block. Such description should be understood as being operations for a luma coding block and/or chroma coding blocks. As described above, in some examples, the luma coding block and chroma coding blocks are luma and chroma components of a CU. In some examples, the luma coding block and the chroma coding blocks are luma and chroma components of a PU.

[0211] In some examples, operations performed with respect to a luma coding block need not be repeated for the chroma coding blocks. As one example, operations to identify a motion vector (MV) and reference picture for a luma coding block need not be repeated for identifying a MV and reference picture for the chroma blocks. Rather, the MV for the luma coding block may be scaled to determine the MV for the chroma blocks, and the reference picture may be the same. As another example, the intra-prediction process may be the same for the luma coding block and the chroma coding blocks.

[0212] Video encoder 200 represents an example of a device configured to encode video data including a memory configured to store video data, and one or more processing units implemented in circuitry and configured to determine a motion vector for a block of video data using a decoder side motion vector derivation process that includes an iterative search process, wherein the iterative search process includes: calculating a minimum cost of a previous search iteration; and terminating the iterative search process early based on the minimum cost of the previous search iteration; and code the block of video data using the motion vector.

[0213] FIG. 10 is a block diagram illustrating an example video decoder 300 that may perform the techniques of this disclosure. FIG. 10 is provided for purposes of explanation and is not limiting on the techniques as broadly exemplified and described in this disclosure. For purposes of explanation, this disclosure describes video decoder 300 according to the techniques of VVC (ITU-T H.266, under development), and HEVC (ITU-T H.265). However, the techniques of this disclosure may be performed by video coding devices that are configured to other video coding standards.

[0214] In the example of FIG. 10, video decoder 300 includes coded picture buffer (CPB) memory 320, entropy decoding unit 302, prediction processing unit 304, inverse quantization unit 306, inverse transform processing unit 308, reconstruction unit 310, filter unit 312, and decoded picture buffer (DPB) 314. Any or all of CPB memory 320, entropy decoding unit 302, prediction processing unit 304, inverse quantization unit

306, inverse transform processing unit 308, reconstruction unit 310, filter unit 312, and DPB 314 may be implemented in one or more processors or in processing circuitry. For instance, the units of video decoder 300 may be implemented as one or more circuits or logic elements as part of hardware circuitry, or as part of a processor, ASIC, or FPGA. Moreover, video decoder 300 may include additional or alternative processors or processing circuitry to perform these and other functions.

[0215] Prediction processing unit 304 includes motion compensation unit 316 and intra-prediction unit 318. Prediction processing unit 304 may include additional units to perform prediction in accordance with other prediction modes. As examples, prediction processing unit 304 may include a palette unit, an intra-block copy unit (which may form part of motion compensation unit 316), an affine unit, a linear model (LM) unit, or the like. In other examples, video decoder 300 may include more, fewer, or different functional components.

[0216] When operating according to AV1, motion compensation unit 316 may be configured to decode coding blocks of video data (e.g., both luma and chroma coding blocks) using translational motion compensation, affine motion compensation, OBMC, and/or compound inter-intra prediction, as described above. Intra prediction unit 318 may be configured to decode coding blocks of video data (e.g., both luma and chroma coding blocks) using directional intra prediction, non-directional intra prediction, recursive filter intra prediction, CFL, intra block copy (IBC), and/or color palette mode, as described above.

[0217] CPB memory 320 may store video data, such as an encoded video bitstream, to be decoded by the components of video decoder 300. The video data stored in CPB memory 320 may be obtained, for example, from computer-readable medium 110 (FIG. 1). CPB memory 320 may include a CPB that stores encoded video data (e.g., syntax elements) from an encoded video bitstream. Also, CPB memory 320 may store video data other than syntax elements of a coded picture, such as temporary data representing outputs from the various units of video decoder 300. DPB 314 generally stores decoded pictures, which video decoder 300 may output and/or use as reference video data when decoding subsequent data or pictures of the encoded video bitstream. CPB memory 320 and DPB 314 may be formed by any of a variety of memory devices, such as DRAM, including SDRAM, MRAM, RRAM, or other types of memory devices. CPB memory 320 and DPB 314 may be provided by the same memory device or separate memory

devices. In various examples, CPB memory 320 may be on-chip with other components of video decoder 300, or off-chip relative to those components.

[0218] Additionally or alternatively, in some examples, video decoder 300 may retrieve coded video data from memory 120 (FIG. 1). That is, memory 120 may store data as discussed above with CPB memory 320. Likewise, memory 120 may store instructions to be executed by video decoder 300, when some or all of the functionality of video decoder 300 is implemented in software to be executed by processing circuitry of video decoder 300.

[0219] The various units shown in FIG. 10 are illustrated to assist with understanding the operations performed by video decoder 300. The units may be implemented as fixed-function circuits, programmable circuits, or a combination thereof. Similar to FIG. 9, fixed-function circuits refer to circuits that provide particular functionality, and are preset on the operations that can be performed. Programmable circuits refer to circuits that can be programmed to perform various tasks, and provide flexible functionality in the operations that can be performed. For instance, programmable circuits may execute software or firmware that cause the programmable circuits to operate in the manner defined by instructions of the software or firmware. Fixed-function circuits may execute software instructions (e.g., to receive parameters or output parameters), but the types of operations that the fixed-function circuits perform are generally immutable. In some examples, one or more of the units may be distinct circuit blocks (fixed-function or programmable), and in some examples, one or more of the units may be integrated circuits.

[0220] Video decoder 300 may include ALUs, EFUs, digital circuits, analog circuits, and/or programmable cores formed from programmable circuits. In examples where the operations of video decoder 300 are performed by software executing on the programmable circuits, on-chip or off-chip memory may store instructions (e.g., object code) of the software that video decoder 300 receives and executes.

[0221] Entropy decoding unit 302 may receive encoded video data from the CPB and entropy decode the video data to reproduce syntax elements. Prediction processing unit 304, inverse quantization unit 306, inverse transform processing unit 308, reconstruction unit 310, and filter unit 312 may generate decoded video data based on the syntax elements extracted from the bitstream.

[0222] In general, video decoder 300 reconstructs a picture on a block-by-block basis. Video decoder 300 may perform a reconstruction operation on each block individually

(where the block currently being reconstructed, e.g., decoded, may be referred to as a "current block").

[0223] Entropy decoding unit 302 may entropy decode syntax elements defining quantized transform coefficients of a quantized transform coefficient block, as well as transform information, such as a quantization parameter (QP) and/or transform mode indication(s). Inverse quantization unit 306 may use the QP associated with the quantized transform coefficient block to determine a degree of quantization and, likewise, a degree of inverse quantization for inverse quantization unit 306 to apply. Inverse quantization unit 306 may, for example, perform a bitwise left-shift operation to inverse quantize the quantized transform coefficients. Inverse quantization unit 306 may thereby form a transform coefficient block including transform coefficients.

[0224] After inverse quantization unit 306 forms the transform coefficient block, inverse transform processing unit 308 may apply one or more inverse transforms to the transform coefficient block to generate a residual block associated with the current block. For example, inverse transform processing unit 308 may apply an inverse DCT, an inverse integer transform, an inverse Karhunen-Loeve transform (KLT), an inverse rotational transform, an inverse directional transform, or another inverse transform to the transform coefficient block.

Furthermore, prediction processing unit 304 generates a prediction block according to prediction information syntax elements that were entropy decoded by entropy decoding unit 302. For example, if the prediction information syntax elements indicate that the current block is inter-predicted, motion compensation unit 316 may generate the prediction block. In this case, the prediction information syntax elements may indicate a reference picture in DPB 314 from which to retrieve a reference block, as well as a motion vector identifying a location of the reference block in the reference picture relative to the location of the current block in the current picture. Motion compensation unit 316 may generally perform the inter-prediction process in a manner that is substantially similar to that described with respect to motion compensation unit 224 (FIG. 9).

[0225] Motion compensation unit 316 may form a motion vector using any of the techniques for determining a motion vector, as described above. For example, motion compensation unit 316 may be configured to determine a motion vector for a block of video data using a decoder side motion vector derivation process that includes an iterative search process. The iterative search process including determining a minimum

cost through a previous search iteration; updating an overall minimum cost through a current search iteration; and terminating the iterative search process early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration. The motion compensation unit 316 may further be configured to decode the block of video data using the motion vector.

[0226] As another example, if the prediction information syntax elements indicate that the current block is intra-predicted, intra-prediction unit 318 may generate the prediction block according to an intra-prediction mode indicated by the prediction information syntax elements. Again, intra-prediction unit 318 may generally perform the intra-prediction process in a manner that is substantially similar to that described with respect to intra-prediction unit 226 (FIG. 9). Intra-prediction unit 318 may retrieve data of neighboring samples to the current block from DPB 314.

[0227] Reconstruction unit 310 may reconstruct the current block using the prediction block and the residual block. For example, reconstruction unit 310 may add samples of the residual block to corresponding samples of the prediction block to reconstruct the current block.

[0228] Filter unit 312 may perform one or more filter operations on reconstructed blocks. For example, filter unit 312 may perform deblocking operations to reduce blockiness artifacts along edges of the reconstructed blocks. Operations of filter unit 312 are not necessarily performed in all examples.

[0229] Video decoder 300 may store the reconstructed blocks in DPB 314. For instance, in examples where operations of filter unit 312 are not performed, reconstruction unit 310 may store reconstructed blocks to DPB 314. In examples where operations of filter unit 312 are performed, filter unit 312 may store the filtered reconstructed blocks to DPB 314. As discussed above, DPB 314 may provide reference information, such as samples of a current picture for intra-prediction and previously decoded pictures for subsequent motion compensation, to prediction processing unit 304. Moreover, video decoder 300 may output decoded pictures (e.g., decoded video) from DPB 314 for subsequent presentation on a display device, such as display device 118 of FIG. 1.

[0230] In this manner, video decoder 300 represents an example of a video decoding device including a memory configured to store video data, and one or more processing units implemented in circuitry and configured to determine a motion vector for a block of video data using a decoder side motion vector derivation process that includes an

iterative search process, wherein the iterative search process includes: calculating a minimum cost of a previous search iteration; and terminating the iterative search process early based on the minimum cost of the previous search iteration; and code the block of video data using the motion vector.

[0231] FIG. 11 is a flowchart illustrating an example method for encoding a current block in accordance with the techniques of this disclosure. The current block may comprise a current CU. Although described with respect to video encoder 200 (FIGS. 1 and 9), it should be understood that other devices may be configured to perform a method similar to that of FIG. 11.

[0232] In this example, video encoder 200 initially predicts the current block (350). For example, video encoder 200 may form a prediction block for the current block. Video encoder 200 may then calculate a residual block for the current block (352). To calculate the residual block, video encoder 200 may calculate a difference between the original, unencoded block and the prediction block for the current block. Video encoder 200 may then transform the residual block and quantize transform coefficients of the residual block (354). Next, video encoder 200 may scan the quantized transform coefficients of the residual block (356). During the scan, or following the scan, video encoder 200 may entropy encode the transform coefficients (358). For example, video encoder 200 may encode the transform coefficients using CAVLC or CABAC. Video encoder 200 may then output the entropy encoded data of the block (360).

[0233] FIG. 12 is a flowchart illustrating an example method for decoding a current block of video data in accordance with the techniques of this disclosure. The current block may comprise a current CU. Although described with respect to video decoder 300 (FIGS. 1 and 10), it should be understood that other devices may be configured to perform a method similar to that of FIG. 12.

[0234] Video decoder 300 may receive entropy encoded data for the current block, such as entropy encoded prediction information and entropy encoded data for transform coefficients of a residual block corresponding to the current block (370). Video decoder 300 may entropy decode the entropy encoded data to determine prediction information for the current block and to reproduce transform coefficients of the residual block (372). Video decoder 300 may predict the current block (374), e.g., using an intra- or inter-prediction mode as indicated by the prediction information for the current block, to calculate a prediction block for the current block. Video decoder 300 may then inverse scan the reproduced transform coefficients (376), to create a block of quantized

transform coefficients. Video decoder 300 may then inverse quantize the transform coefficients and apply an inverse transform to the transform coefficients to produce a residual block (378). Video decoder 300 may ultimately decode the current block by combining the prediction block and the residual block (380).

[0235] FIG. 13 is a flowchart illustrating an example method for terminating an iterative search process early in accordance with the techniques of this disclosure. The current block may comprise a current CU. The iterative search process is discussed with respect to being performed by video encoder 200 and/or video decoder 300.

[0236] Video encoder 200/decoder 300 may determine a motion vector for a block of video data using a decoder side motion vector derivation process that includes an iterative search process. In an example, in the iterative search process, video encoder 200/decoder 300 may be configured to perform a 1st search iteration and set a minimum cost through a previous search iteration to zero (1300). Video encoder 200/decoder 300 may further be configured to determine a cost through the 1st search iteration and set an overall minimum cost through the current search iteration as the overall cost through the 1st search iteration (1302). Video encoder 200/decoder 300 may further be configured to determine whether to terminate the iterative search process based on comparing the minimum cost through a previous search iteration to the overall minimum cost through the current search iteration (1304). If video encoder 200/decoder 300 determines to terminate the iterative search process early, video encoder 200/decoder 300 is further configured to respectively encode/decode the block of video data using the motion vector having the minimum cost up to that point in the search process (1305).

[0237] If video encoder 200/decoder 300 determines to not terminate the iterative search process early, video encoder 200/decoder 300 is further configured to perform a 2nd search iteration and set a minimum cost through a previous search iteration as the minimum cost through the 1st search iteration (1306). Video encoder 200/decoder 300 may further be configured to update an overall minimum cost through the current search iteration as the overall cost through the 2nd search iteration (1308). Video encoder 200/decoder 300 may further be configured to determine whether to terminate the iterative search process based on comparing the minimum cost through a previous search iteration to the overall minimum cost through the current search iteration (1310). If video encoder 200/decoder 300 determines to terminate the iterative search process early, video encoder 200/decoder 300 is further configured to respectively encode/decode the block of video data using the motion vector having the minimum cost up to that point in the search process (1305).

[0238] If video encoder 200/decoder 300 determines to not terminate the iterative search process early, video encoder 200/decoder 300 is further configured to perform a 3rd search iteration and set a minimum cost through a previous search iteration as the minimum cost through the 2nd search iteration (1312). Video encoder 200/decoder 300 may further be configured to update an overall minimum cost through the current search iteration as the overall cost through the 3rd search iteration (1314). Video encoder 200/decoder 300 may further be configured to determine whether to terminate the iterative search process based on comparing the minimum cost through a previous search iteration to the overall minimum cost through the current search iteration (1316). If video encoder 200/decoder 300 determines to terminate the iterative search process early, video encoder 200/decoder 300 is further configured to respectively encode/decode the block of video data using the motion vector having the minimum cost up to that point in the search process (1305).

[0239] If video encoder 200/decoder 300 determines to not terminate the iterative search process early, video encoder 200/decoder 300 is further configured to perform a next search iteration in accordance with the previous search iteration until video encoder 200/decoder 300 terminate the iterative search process (1318).

[0240] In some examples, terminating the iterative search process early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration may be done based on multiple comparisons.

[0241] FIG. 14 is a flowchart illustrating another example method for decoding a block of video data in accordance with the techniques of this disclosure. The techniques of FIG. 14 may be performed by one or more structural components of video decoder 300, including motion compensation unit 316.

[0242] In one example of the disclosure, video decoder 300 may be configured to determine a motion vector for a block of video data using a decoder side motion vector derivation process that includes an iterative search process (1400). In the iterative search process, video decoder 300 may further be configured to determine a minimum cost through a previous search iteration (1402), update an overall minimum cost through a current search iteration (1404), and terminate the iterative search process early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration (1406). Video decoder 300 may then decode a block of video data using the motion vector (1408).

[0243] In one example, video decoder 300 may be configured to calculate the overall minimum cost through the current search iteration based on at least a current subset of candidate positions.

[0244] In another example, to terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, video decoder 300 may be configured to calculate a difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, and terminate the iterative search process early based on the difference relative to a difference threshold.

[0245] In another example, to terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, video decoder 300 may be configured to calculate a ratio of the minimum cost through the previous search iteration to the overall minimum cost through the current search iteration, and terminate the iterative search process early based on the ratio relative to a ratio threshold.

[0246] In another example, to terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, video decoder 300 may be configured to calculate a difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, calculate a ratio of the minimum cost through the previous search iteration to the difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, and terminate the iterative search process early based on the ratio relative to a ratio threshold.

[0247] In another example, video decoder 300 may be configured to terminate the iterative search process early based on the minimum cost through the previous search iteration, the overall minimum cost through the current search iteration, and a minimum cost of the current search iteration. For example, video decoder 300 may be configured to terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, and a comparison of the minimum cost through the previous search iteration and the minimum cost of the current search iteration.

[0248] FIG. 15 is a flowchart illustrating another example method for encoding a block of video data in accordance with the techniques of this disclosure. The techniques of FIG.

15 may be performed by one or more structural components of video encoder 200, including motion estimation unit 222 and/or motion compensation unit 224.

[0249] In one example of the disclosure, video encoder 200 may be configured to determine a motion vector for a block of video data using a decoder side motion vector derivation process that includes an iterative search process (1500). In the iterative search process, video decoder 300 may further be configured to determine a minimum cost through a previous search iteration (1502), update an overall minimum cost through a current search iteration (1504), and terminate the iterative search process early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration (1506). Video encoder 200 may then encode a block of video data using the motion vector (1508).

[0250] In one example, video encoder 200 may be configured to calculate the overall minimum cost through the current search iteration based on at least a current subset of candidate positions.

[0251] In another example, to terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, video encoder 200 may be configured to calculate a difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, and terminate the iterative search process early based on the difference relative to a difference threshold.

[0252] In another example, to terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, video encoder 200 may be configured to calculate a ratio of the minimum cost through the previous search iteration to the overall minimum cost through the current search iteration, and terminate the iterative search process early based on the ratio relative to a ratio threshold.

[0253] In another example, to terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, video encoder 200 may be configured to calculate a difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, calculate a ratio of the minimum cost through the previous search iteration to the difference of the minimum cost through the previous search iteration and the overall minimum cost through the current

search iteration, and terminate the iterative search process early based on the ratio relative to a ratio threshold.

[0254] In another example, video encoder 200 may be configured to terminate the iterative search process early based on the minimum cost through the previous search iteration, the overall minimum cost through the current search iteration, and a minimum cost of the current search iteration. For example, video decoder 200 may be configured to terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, and a comparison of the minimum cost through the previous search iteration and the minimum cost of the current search iteration.

[0255] Additional method and devices of the disclosure are described in the following aspects.

[0256] Aspect 1A - A method of coding video data, the method comprising: determining a motion vector for a block of video data using a decoder side motion vector derivation process that includes an iterative search process, wherein the iterative search process includes: calculating a minimum cost of a previous search iteration; and terminating the iterative search process early based on the minimum cost of the previous search iteration; and coding the block of video data using the motion vector.

[0257] Aspect 2A - The method of Aspect 1A, further comprising: updating a minimum cost determined for the iterative search process.

[0258] Aspect 3A - The method of Aspect 2A, wherein terminating the iterative search process early based on the minimum cost of the previous search iteration comprises: calculating a difference or a ratio of the minimum cost of the previous search iteration and the minimum cost determined for the iterative search process; and terminating the iterative search process early based on the difference or the ratio relative to a threshold.

[0259] Aspect 4A - The method of Aspect 2A, wherein terminating the iterative search process early based on the minimum cost of the previous search iteration comprises: terminating the iterative search process early based on the minimum cost of the previous search iteration, the minimum cost determined for the iterative search process, and a minimum cost of a current iteration.

[0260] Aspect 5A - The method of Aspect 1A, wherein the iterative search process includes more than two iterations.

[0261] Aspect 6A - The method of Aspect 1A, wherein the iterative search process includes bilateral matching or template matching.

[0262] Aspect 7A - The method of any of Aspects 1A–6A, wherein coding comprises decoding.

[0263] Aspect 8A - The method of any of Aspects 1A–6A, wherein coding comprises encoding.

[0264] Aspect 9A - A device for coding video data, the device comprising one or more means for performing the method of any of Aspects 1A–8A.

[0265] Aspect 10A - The device of Aspect 9A, wherein the one or more means comprise one or more processors implemented in circuitry.

[0266] Aspect 11A - The device of any of Aspects 9A and 10A, further comprising a memory to store the video data.

[0267] Aspect 12A - The device of any of Aspects 9A-11A, further comprising a display configured to display decoded video data.

[0268] Aspect 13A - The device of any of Aspects 9A-12A, wherein the device comprises one or more of a camera, a computer, a mobile device, a broadcast receiver device, or a set-top box.

[0269] Aspect 14A - The device of any of Aspects 9A-13A, wherein the device comprises a video decoder.

[0270] Aspect 15A - The device of any of Aspects 9A-14A, wherein the device comprises a video encoder.

[0271] Aspect 16A - A computer-readable storage medium having stored thereon instructions that, when executed, cause one or more processors to perform the method of any of Aspects 1A–8A.

[0272] Aspect 17A - A device for coding video data, the device comprising: means for determining a motion vector for a block of video data using a decoder side motion vector derivation process that includes an iterative search process, wherein the iterative search process includes: calculating a minimum cost of a previous search iteration; and terminating the iterative search process early based on the minimum cost of the previous search iteration; and means for coding the block of video data using the motion vector.

[0273] Aspect 18A - A method of coding video data, the method comprising: determining a motion vector for a block of video data using a decoder side motion vector derivation process that includes an iterative search process, wherein the iterative search process includes: calculating a minimum cost of a previous search iteration; and terminating the iterative search process early based on the minimum cost of the previous search iteration; and coding the block of video data using the motion vector.

[0274] Aspect 19A - The method of Aspect 18A, further comprising: updating a minimum cost determined for the iterative search process.

[0275] Aspect 20A - The method of Aspect 19A, wherein terminating the iterative search process early based on the minimum cost of the previous search iteration comprises: calculating a difference or a ratio of the minimum cost of the previous search iteration and the minimum cost determined for the iterative search process; and terminating the iterative search process early based on the difference or the ratio relative to a threshold.

[0276] Aspect 21A - The method of Aspect 19A, wherein terminating the iterative search process early based on the minimum cost of the previous search iteration comprises: terminating the iterative search process early based on the minimum cost of the previous search iteration, the minimum cost determined for the iterative search process, and a minimum cost of a current iteration.

[0277] Aspect 22A - The method of any of Aspects 18A-21A, wherein the iterative search process includes more than two iterations.

[0278] Aspect 23A - The method of any of Aspects 18A-22A, wherein the iterative search process includes bilateral matching or template matching.

[0279] Aspect 24A - A device for coding video data, the device comprising one or more means for performing the method of any of Aspects 18A-23A.

[0280] Aspect 1B - A method of encoding or decoding video data includes determining a motion vector for a block of video data using a decoder side motion vector derivation process that includes performing an iterative search process, wherein performing the iterative search process includes: determining a minimum cost through a previous search iteration; updating an overall minimum cost through a current search iteration; and terminating the iterative search process early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and encoding or decoding the block of video data using the motion vector.

[0281] Aspect 2B - The method of aspect 1B, wherein the previous search iteration immediately precedes the current search iteration in the iterative search process.

[0282] Aspect 3B - The method of aspect 1B, further includes calculating the overall minimum cost through the current search iteration based on at least a current subset of candidate positions.

[0283] Aspect 4B - The method of aspect 1B, wherein terminating the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration comprises: calculating a difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and terminating the iterative search process early based on the difference relative to a difference threshold.

[0284] Aspect 5B - The method of aspect 4B, wherein the difference threshold is based on a size of the block.

[0285] Aspect 6B - The method of aspect 1B, wherein terminating the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration comprises: calculating a ratio of the minimum cost through the previous search iteration to the overall minimum cost through the current search iteration; and terminating the iterative search process early based on the ratio relative to a ratio threshold.

[0286] Aspect 7B - The method of aspect 6B, wherein the ratio threshold is fixed or based on a size of the block.

[0287] Aspect 8B - The method of aspect 1B, wherein terminating the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration comprises: calculating a difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; calculating a ratio of the minimum cost through the previous search iteration to the difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and terminating the iterative search process early based on the ratio relative to a ratio threshold.

[0288] Aspect 9B - The method of aspect 8B, wherein the ratio threshold is fixed or based on a size of the block.

[0289] Aspect 10B - The method of aspect 1B, wherein terminating the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration comprises: terminating the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, and a comparison of the minimum cost through the previous search iteration and a minimum cost of the current search iteration.

[0290] Aspect 11B - The method of aspect 1B, wherein the iterative search process includes more than two iterations.

[0291] Aspect 12B - The method of aspect 1B, wherein performing the iterative search process includes performing bilateral matching or template matching.

[0292] Aspect 13B - A device for processing video data includes memory configured to store the video data; and processing circuitry configured to: determine a motion vector for a block of the video data using a decoder side motion vector derivation process that includes an iterative search process, wherein to perform the iterative search process the processing circuitry is further configured to:: determine a minimum cost through a previous search iteration; update an overall minimum cost through a current search iteration; and terminate the iterative search process early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and encode or decode the block of video data using the motion vector.

[0293] Aspect 14B - The device of aspect 13B, wherein the previous search iteration immediately precedes the current search iteration.

[0294] Aspect 15B - The device of aspect 13B, wherein to perform the iterative search process the processing circuitry is further configured to calculate the overall minimum cost through the current search iteration based on at least a current subset of candidate positions.

[0295] Aspect 16B - The device of aspect 13B, wherein to terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, the processing circuitry is further configured to : calculate a difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and terminate the iterative search process early based on the difference relative to a difference threshold.

[0296] Aspect 17B - The device of aspect 16B, wherein the difference threshold is based on a size of the block.

[0297] Aspect 18B - The device of aspect 13B, wherein to terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, the processing circuitry is further configured to: calculate a ratio of the minimum cost through the previous search iteration to the overall minimum cost through the current

search iteration; and terminate the iterative search process early based on the ratio relative to a ratio threshold.

[0298] Aspect 19B - The device of aspect 18B, wherein the ratio threshold is fixed or based on a size of the block.

[0299] Aspect 20B - The device of aspect 13B, wherein to terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, the processing circuitry is further configured to: calculate a difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; calculate a ratio of the minimum cost through the previous search iteration to the difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and terminate the iterative search process early based on the ratio relative to a ratio threshold.

[0300] Aspect 21B - The device of aspect 20B, wherein the ratio threshold is fixed or based on a size of the block.

[0301] Aspect 22B - The device of aspect 13B, wherein to terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, the processing circuitry is further configured to:: terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, and a comparison of the minimum cost through the previous search iteration and a minimum cost of the current search iteration.

[0302] Aspect 23B - The device of aspect 13B, wherein the iterative search process includes more than two iterations.

[0303] Aspect 24B - The device of aspect 13B, wherein to perform the iterative search process the processing circuitry is configured to perform bilateral matching or template matching.

[0304] Aspect 25B - The device of aspect 13B, further comprising a display configured to display the video data.

[0305] Aspect 26B - The device of aspect 13B, wherein the device comprises one or more of a camera, a computer, a mobile device, a broadcast receiver device, or a set-top box.

[0306] Aspect 27B - The device of aspect 13, wherein the device comprises a camera configured to capture the video data.

[0307] Aspect 1C - A method of encoding or decoding video data includes determining a motion vector for a block of video data using a decoder side motion vector derivation process that includes performing an iterative search process, wherein performing the iterative search process includes: determining a minimum cost through a previous search iteration; updating an overall minimum cost through a current search iteration; and terminating the iterative search process early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and encoding or decoding the block of video data using the motion vector.

[0308] Aspect 2C - The method of aspect 1C, wherein the previous search iteration immediately precedes the current search iteration in the iterative search process.

[0309] Aspect 3C - The method of any of aspects 1C and 2C, further includes calculating the overall minimum cost through the current search iteration based on at least a current subset of candidate positions.

[0310] Aspect 4C - The method of any of aspects 1C through 3C, wherein terminating the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration comprises: calculating a difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and terminating the iterative search process early based on the difference relative to a difference threshold.

[0311] Aspect 5C - The method of aspect 4C, wherein the difference threshold is based on a size of the block.

[0312] Aspect 6C - The method of any of aspects 1C through 5C, wherein terminating the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration comprises: calculating a ratio of the minimum cost through the previous search iteration to the overall minimum cost through the current search iteration; and terminating the iterative search process early based on the ratio relative to a ratio threshold.

[0313] Aspect 7C - The method of aspect 6C, wherein the ratio threshold is fixed or based on a size of the block.

[0314] Aspect 8C - The method of any of aspects 1C through 7C, wherein terminating the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration comprises: calculating a difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; calculating a ratio of the minimum cost through the previous search iteration to the difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and terminating the iterative search process early based on the ratio relative to a ratio threshold.

[0315] Aspect 9C - The method of aspect 8C, wherein the ratio threshold is fixed or based on a size of the block.

[0316] Aspect 10C - The method of any of aspects 1C through 9C, wherein terminating the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration comprises: terminating the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, and a comparison of the minimum cost through the previous search iteration and a minimum cost of the current search iteration.

[0317] Aspect 11C - The method of any of aspects 1C through 10C, wherein the iterative search process includes more than two iterations.

[0318] Aspect 12C - The method of any of aspects 1C through 11C, wherein performing the iterative search process includes performing bilateral matching or template matching.

[0319] Aspect 13C - A device for processing video data includes memory configured to store the video data; and processing circuitry configured to: determine a motion vector for a block of the video data using a decoder side motion vector derivation process that includes an iterative search process, wherein to perform the iterative search process the processing circuitry is further configured to:: determine a minimum cost through a previous search iteration; update an overall minimum cost through a current search iteration; and terminate the iterative search process early based on a comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and encode or decode the block of video data using the motion vector.

[0320] Aspect 14C - The device of aspect 13C, wherein the previous search iteration immediately precedes the current search iteration.

[0321] Aspect 15C - The device of any of aspects 13C and 14C, wherein to perform the iterative search process the processing circuitry is further configured to calculate the overall minimum cost through the current search iteration based on at least a current subset of candidate positions.

[0322] Aspect 16C - The device of any of aspects 13C through 15C, wherein to terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, the processing circuitry is further configured to : calculate a difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and terminate the iterative search process early based on the difference relative to a difference threshold.

[0323] Aspect 17C - The device of aspect 16C, wherein the difference threshold is based on a size of the block.

[0324] Aspect 18C - The device of any of aspects 13C through 17C, wherein to terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, the processing circuitry is further configured to: calculate a ratio of the minimum cost through the previous search iteration to the overall minimum cost through the current search iteration; and terminate the iterative search process early based on the ratio relative to a ratio threshold.

[0325] Aspect 19C - The device of aspect 18C, wherein the ratio threshold is fixed or based on a size of the block.

[0326] Aspect 20C - The device of any of aspects 13C through 19C, wherein to terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, the processing circuitry is further configured to: calculate a difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; calculate a ratio of the minimum cost through the previous search iteration to the difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and terminate the iterative search process early based on the ratio relative to a ratio threshold.

[0327] Aspect 21C - The device of aspect 20C, wherein the ratio threshold is fixed or based on a size of the block.

[0328] Aspect 22C - The device of any of aspects 13C through 21C, wherein to terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, the processing circuitry is further configured to:: terminate the iterative search process early based on the comparison of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration, and a comparison of the minimum cost through the previous search iteration and a minimum cost of the current search iteration.

[0329] Aspect 23C - The device of any of aspects 13C through 22C, wherein the iterative search process includes more than two iterations.

[0330] Aspect 24C - The device of any of aspects 13C through 23C, wherein to perform the iterative search process the processing circuitry is configured to perform bilateral matching or template matching.

[0331] Aspect 25C - The device of any of aspects 13C through 24C, further comprising a display configured to display the video data.

[0332] Aspect 26C - The device of any of aspects 13C through 25C, wherein the device comprises one or more of a camera, a computer, a mobile device, a broadcast receiver device, or a set-top box.

[0333] Aspect 27C - The device of any of aspects 13C through 26C, wherein the device comprises a camera configured to capture the video data.

[0334] It is to be recognized that depending on the example, certain acts or events of any of the techniques described herein can be performed in a different sequence, may be added, merged, or left out altogether (e.g., not all described acts or events are necessary for the practice of the techniques). Moreover, in certain examples, acts or events may be performed concurrently, e.g., through multi-threaded processing, interrupt processing, or multiple processors, rather than sequentially.

[0335] In one or more examples, the functions described may be implemented in hardware, software, firmware, or any combination thereof. If implemented in software, the functions may be stored on or transmitted over as one or more instructions or code on a computer-readable medium and executed by a hardware-based processing unit. Computer-readable media may include computer-readable storage media, which corresponds to a tangible medium such as data storage media, or communication media

including any medium that facilitates transfer of a computer program from one place to another, e.g., according to a communication protocol. In this manner, computer-readable media generally may correspond to (1) tangible computer-readable storage media which is non-transitory or (2) a communication medium such as a signal or carrier wave. Data storage media may be any available media that can be accessed by one or more computers or one or more processors to retrieve instructions, code and/or data structures for implementation of the techniques described in this disclosure. A computer program product may include a computer-readable medium.

[0336] By way of example, and not limitation, such computer-readable storage media can comprise RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage, or other magnetic storage devices, flash memory, or any other medium that can be used to store desired program code in the form of instructions or data structures and that can be accessed by a computer. Also, any connection is properly termed a computer-readable medium. For example, if instructions are transmitted from a website, server, or other remote source using a coaxial cable, fiber optic cable, twisted pair, digital subscriber line (DSL), or wireless technologies such as infrared, radio, and microwave, then the coaxial cable, fiber optic cable, twisted pair, DSL, or wireless technologies such as infrared, radio, and microwave are included in the definition of medium. It should be understood, however, that computer-readable storage media and data storage media do not include connections, carrier waves, signals, or other transitory media, but are instead directed to non-transitory, tangible storage media. Disk and disc, as used herein, includes compact disc (CD), laser disc, optical disc, digital versatile disc (DVD), floppy disk and Blu-ray disc, where disks usually reproduce data magnetically, while discs reproduce data optically with lasers. Combinations of the above should also be included within the scope of computer-readable media.

[0337] Instructions may be executed by one or more processors, such as one or more DSPs, general purpose microprocessors, ASICs, FPGAs, or other equivalent integrated or discrete logic circuitry. Accordingly, the terms “processor” and “processing circuitry,” as used herein may refer to any of the foregoing structures or any other structure suitable for implementation of the techniques described herein. In addition, in some aspects, the functionality described herein may be provided within dedicated hardware and/or software modules configured for encoding and decoding, or incorporated in a combined codec. Also, the techniques could be fully implemented in one or more circuits or logic elements.

[0338] The techniques of this disclosure may be implemented in a wide variety of devices or apparatuses, including a wireless handset, an integrated circuit (IC) or a set of ICs (e.g., a chip set). Various components, modules, or units are described in this disclosure to emphasize functional aspects of devices configured to perform the disclosed techniques, but do not necessarily require realization by different hardware units. Rather, as described above, various units may be combined in a codec hardware unit or provided by a collection of interoperative hardware units, including one or more processors as described above, in conjunction with suitable software and/or firmware.

[0339] Various examples have been described. These and other examples are within the scope of the following claims.

CLAIMS

1. A method of encoding or decoding video data, the method comprising:

determining a motion vector for a block of video data using a decoder side motion vector derivation process that includes performing an iterative search process, wherein performing the iterative search process includes:

determining a minimum cost through a previous search iteration;

updating an overall minimum cost through a current search iteration;

calculating a difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and

terminating the iterative search process early based on the difference relative to a difference threshold; and

encoding or decoding the block of video data using the motion vector.

2. The method of claim 1, wherein the previous search iteration immediately precedes the current search iteration in the iterative search process.

3. The method of claim 1, further comprising:

calculating the overall minimum cost through the current search iteration based on at least a current subset of candidate positions.

4. The method of claim 1, wherein the difference threshold is based on a size of the block.

5. The method of claim 1, wherein the iterative search process includes more than two iterations.

6. The method of claim 1, wherein performing the iterative search process includes performing bilateral matching or template matching.

7. A device for processing video data, the device comprising:

memory configured to store the video data; and

processing circuitry configured to:

determine a motion vector for a block of the video data using a decoder side motion vector derivation process that includes an iterative search process, wherein to perform the iterative search process the processing circuitry is further configured to:

determine a minimum cost through a previous search iteration;

update an overall minimum cost through a current search iteration;

calculate a difference of the minimum cost through the previous search iteration and the overall minimum cost through the current search iteration; and

terminate the iterative search process early based on the difference relative to a difference threshold; and

encode or decode the block of video data using the motion vector.

8. The device of claim 7, wherein the previous search iteration immediately precedes the current search iteration.

9. The device of claim 7, wherein to perform the iterative search process the processing circuitry is further configured to calculate the overall minimum cost through the current search iteration based on at least a current subset of candidate positions.

10. The device of claim 7, wherein the difference threshold is based on a size of the block.

11. The device of claim 7, wherein the iterative search process includes more than two iterations.

12. The device of claim 7, wherein to perform the iterative search process the processing circuitry is configured to perform bilateral matching or template matching.

13. The device of claim 7, further comprising a display configured to display the video data.

14. The device of claim 7, wherein the device comprises one or more of a camera, a computer, a mobile device, a broadcast receiver device, or a set-top box.

15. The device of claim 7, wherein the device comprises a camera configured to capture the video data.