(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 November 2015 (26.11.2015)

WIPO | PCT

(10) International Publication Number
WO 2015/179010 A2

(51) International Patent Classification:
H04N 19/60 (2014.01)

(21) International Application Number:
PCT/US2015/020237

(22) International Filing Date:
12 March 2015 (12.03.2015)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
14/216,369 17 March 2014 (17.03.2014) US

(71) Applicant: **QUALCOMM INCORPORATED** [US/US]; Attn: International IP Administration, 5775 Morehouse Drive, San Diego, California 92121-1714 (US).

(72) Inventors: **CHONG, In Suk**; 5775 Morehouse Drive, San Diego, California 92121-1714 (US). **YU, Yang**; 5775 Morehouse Drive, San Diego, California 92121-1714 (US). **WANG, Xianglin**; 5775 Morehouse Drive, San Diego, California 92121-1714 (US). **KARCZEWICZ, Marta**; 5775 Morehouse Drive, San Diego, California 92121-1714 (US).

(74) Agent: **ABUMERI, Mark M.**; Knobbe Martens Olson & Bear LLP, 2040 Main Street, Fourteenth Floor, Irvine, California 92614 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU,

[Continued on next page]

(54) Title: SYSTEMS AND METHODS FOR LOW COMPLEXITY FORWARD TRANSFORMS USING MESH-BASED CALCULATIONS

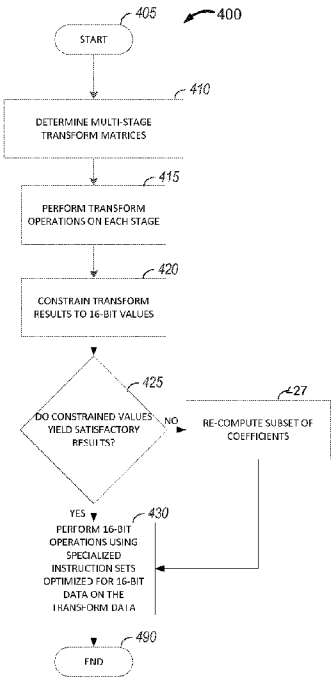

FIG. 4

(57) Abstract: Systems and methods for low complexity forward transforms using mesh-based calculations are described herein. One aspect of the subject matter described in the disclosure provides a video encoder comprising a memory configured to store video information. The video encoder further comprises a processor in communication with the memory. The processor is configured to decompose a transform into multiple transform stages. The processor is further configured to transform the video information using the multiple stages to determine a transform stage output at each transform stage. The processor is further configured to constrain the transform stage output at each transform stage to a predetermined bit depth. The processor is further configured to perform operations on the constrained transform output of a last stage of the multiple stages, wherein the operations are only available for use with data having the predetermined bit depth.

WO 2015/179010 A2

TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

— *without international search report and to be republished upon receipt of that report (Rule 48.2(g))*

SYSTEMS AND METHODS FOR LOW COMPLEXITY FORWARD TRANSFORMS USING MESH-BASED CALCULATIONS

TECHNICAL FIELD

[0001] This disclosure relates to video encoding.

BACKGROUND

[0002] 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, smartphones, 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), High Efficiency Video Coding (HEVC), and extensions of such standards. The video devices may transmit, receive, encode, decode, and/or store digital video information more efficiently by implementing such video coding techniques.

[0003] 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 frame or a portion of a video frame) may be partitioned into video blocks, which may also be referred to as treeblocks, coding units (CUs), and/or coding nodes. CUs may be further partitioned into one or more prediction units (PUs) to determine predictive video data for the CU. The video compression techniques may also partition the CUs into one or more transform units (TUs) of residual video block data, which represents the difference between the video block to be coded and the predictive video data. Linear transforms, such as a two-dimensional discrete cosine transform (DCT), may be applied to a TU to transform the residual video block data from the pixel domain to the frequency domain to achieve further compression. Further, video blocks in an intra-coded (I) slice of a picture may be 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 a reference frames.

[0004] Spatial or temporal prediction results in a predictive block for a block to be coded. Residual data represents pixel differences between the original block to be coded and the predictive block. An inter-coded block is encoded according to a motion vector that points to a block of reference samples forming the predictive block, and the residual data indicating the difference between the coded block and the predictive block. An intra-coded block is encoded according to an intra-coding mode and the residual data. For further compression, the residual data may be transformed from the pixel domain to a transform domain, resulting in residual transform coefficients, which then may be quantized. The quantized transform coefficients, initially arranged in a two-dimensional array, may be scanned in order to produce a one-dimensional vector of transform coefficients, and entropy encoding may be applied to achieve even more compression.

[0005] In older video standards, such as AVC, forward transform and inverse transform size (e.g., 4x4 and 8x8) did not act as a bottleneck for video encoding performance. However, the more modern HEVC standard utilizes up to 16x16 and 32x32 forward transform and inverse transform sizes, which do act as a limiting factor for the HEVC process. The larger transforms require more complexity and cycles to process when transforming from the pixel domain into the coefficient domain. In the interest of coding efficiency, the standard would benefit from a process that decomposes the large forward transform vectors in the video encoder to multiple stages (e.g., "mesh-based method," "Butterfly method" or "Even-Odd Decomposition") and constraining the internal bit depth at each stage. Some advantages of the techniques disclosed herein relate to improving coding efficiency and reducing computational resource requirements during video encoding by decomposing the large forward transform vectors in the video encoder to multiple stages and constraining the internal bit depth at each stage.

SUMMARY

[0006] In general, this disclosure describes techniques related to improving video encoding performance by decomposing large forward transforms into multiple

stages (e.g., mesh-based method of implementing a forward transform) and constraining the internal bit depth at each stage to a level that may accommodate computationally efficient instruction sets. For example, although an input bit depth to a transform may be 9-bits and a starting internal bit depth may be greater than 16-bits, a video encoder may be configured to constrain the internal bit depth of the transform to 16-bits using saturation logic (e.g., clipping the internal values to 16-bits). To prevent a loss of video quality, the video encoder may be further configured to measure a level of deviations (e.g., errors), compare the level to a threshold, and recompute a subset of the transformed coefficients if the level of deviations exceeds the threshold.

[0007] 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 and drawings, and from the claims.

[0008] One aspect of the subject matter described in the disclosure provides a video encoder comprising a memory configured to store video information. The video encoder further comprises a processor in communication with the memory. The processor is configured to decompose a transform into multiple transform stages. The processor is further configured to transform the video information using the multiple stages to determine a transform stage output at each transform stage. The processor is further configured to constrain the transform stage output at each transform stage to a predetermined bit depth. The processor is further configured to perform operations on the constrained transform output of a last stage of the multiple stages, wherein the operations are only available for use with data having the predetermined bit depth.

[0009] Another aspect of the subject matter described in the disclosure provides a method of encoding video. The method includes storing video information. The method further includes decomposing a transform into multiple transform stages. The method further includes transforming the video information using the multiple stages to determine a transform stage output at each transform stage. The method further includes constraining the transform stage output at each transform stage to a predetermined bit depth. The method further includes performing operations on the constrained transform output of a last stage of the multiple stages, wherein the operations are only available for use with data having the predetermined bit depth.

[0010] Another aspect of the subject matter described in the disclosure provides a non-transitory computer-readable medium. The medium comprises code that,

when executed, causes an apparatus to store video information. The medium further comprises code that, when executed, causes an apparatus to decompose a transform into multiple transform stages. The medium further comprises code that, when executed, causes an apparatus to transform the video information using the multiple stages to determine a transform stage output at each transform stage. The medium further comprises code that, when executed, causes an apparatus to constrain the transform stage output at each transform stage to a predetermined bit depth. The medium further comprises code that, when executed, causes an apparatus to perform operations on the constrained transform output of a last stage of the multiple stages, wherein the operations are only available for use with data having the predetermined bit depth.

[0011] Another aspect of the subject matter described in the disclosure provides an apparatus for encoding video. The apparatus comprises means for storing video information. The apparatus further comprises means for decomposing a transform into multiple transform stages. The apparatus further comprises means for transforming the video information using the multiple stages to determine a transform stage output at each transform stage. The apparatus further comprises means for constraining the transform stage output at each transform stage to a predetermined bit depth. The apparatus further comprises means for performing operations on the constrained transform output of a last stage of the multiple stages, wherein the operations are only available for use with data having the predetermined bit depth.

BRIEF DESCRIPTION OF DRAWINGS

[0012] FIG. 1 is a block diagram illustrating an example video encoding and decoding system that may utilize techniques in accordance with aspects described in this disclosure.

[0013] FIG. 2 is a block diagram illustrating an example of a video encoder that may implement techniques in accordance with aspects described in this disclosure.

[0014] FIG. 3 is a block diagram illustrating an example of a video decoder that may implement techniques in accordance with aspects described in this disclosure.

[0015] FIG. 4 illustrates a flowchart of a method of low complexity forward transformation, for example, using mesh-based calculations.

[0016] FIG. 5 illustrates a flowchart of a method for a low complexity forward transformation.

[0017] The various features illustrated in the drawings may not be drawn to scale. Accordingly, the dimensions of the various features may be arbitrarily expanded or reduced for clarity. In addition, some of the drawings may not depict all of the components of a given system, method or device. Finally, like reference numerals may be used to denote like features throughout the specification and figures.

DETAILED DESCRIPTION

[0018] The techniques described in this disclosure generally relate to forward transforms during video encoding, particularly in regards to the High-Efficiency Video Coding (HEVC) standard and its extensions.

[0019] Video coding standards include ITU-T H.261, ISO/IEC MPEG-1 Visual, ITU-T H.262 or ISO/IEC MPEG-2 Visual, ITU-T H.263, ISO/IEC MPEG-4 Visual and ITU-T H.264 (also known as ISO/IEC MPEG-4 AVC), including its Scalable Video Coding (SVC) and Multiview Video Coding (MVC) extensions. In addition, there is a new video coding standard, High Efficiency Video Coding (HEVC), developed by the Joint Collaboration Team on Video Coding (JCT-VC) of ITU-T Video Coding Experts Group (VCEG) and ISO/IEC Motion Picture Experts Group (MPEG).

[0020] As mentioned above, the HEVC standard utilizes forward transforms (e.g., in the HEVC encoder) and inverse transforms of up to 32x32 in size, whereas the AVC standard only utilizes up to an 8x8 transform size. The larger transform sizes increase the coding efficiency of large code blocks in HEVC; however, they also increase the complexity, computing cycles, and processing time as compared to using smaller transform sizes. The methods described in this disclosure may reduce the increased complexity and cycles required when the encoder transforms video information from the pixel domain into the coefficient domain. For example, certain methods include decomposing the large forward transform vectors in the video encoder to multiple stages (e.g., mesh-based method) and constraining the internal bit depth at each stage.

[0021] In some implementations, decomposing large forward transforms into multiple stages (e.g., using the mesh-based, or "Butterfly," method of implementing a forward transform) for purposes of converting from the residual domain to the coefficient domain may result in more efficient processing than using a matrix multiplication method. One example of a mesh-based NxN transform implementation is

illustrated in the code of Appendix A below. In one implementation, a transform unit (e.g., a 16x16 transform) may start with two-hundred-and-fifty-six residual source pixels (e.g., pSrc in Appendix A), each of which may represent a luma value. A processor or encoder (e.g., the transform processing unit 52 of the encoder 20 of FIG. 2) may then determine two-hundred-and-fifty-six output coefficients (e.g., pDst in Appendix A), one for each source pixel. After the first stage, the processor or encoder (e.g., the transform processing unit 52) may then determine one-hundred-and-twenty-eight sums of two pixels and one-hundred-and-twenty-eight differences of two pixels (e.g., nE's and nO's of Appendix A). During a second processing stage, the processor or encoder may use the nE and nO sums and differences to determine a sum of pairs and a sum of differences. After four stages, the processor or encoder may produce the output coefficient (e.g., pDst). Using this mesh-based method (as illustrated in Appendix A), the full-size transform may be decomposed into several smaller, less complex transforms, which if multiplied together would again produce the full-size transform.

[0022] After performing a mesh method, such as the mesh method discussed above or any other decomposition method, the methods described in this disclosure may configure a processor or encoder to constrain the internal bit depth to a certain level at each stage to enable the processor or encoder to utilize further computationally efficient instruction sets for that level. Indeed, certain encoders and decoders include computationally efficient instruction sets that are only available for use with inputs having certain bit depths. For example, although an input bit depth to a transform may be 9-bits and a starting internal bit depth may be greater than 16-bits, a video encoder may be configured to constrain the internal bit depth (e.g., the bit depth during the transform operation rather than only the transform output bit depth) of each decomposed stage of the transform to 16-bits. With the internal bit depth of each stage constrained to 16-bits, the processor may utilize computationally efficient instruction sets that are designed specifically for use with 16-bit operations (e.g., ARM architecture, Advanced SIMD (NEON), Digital Signal Processing (DSP), etc.). In one implementation, the video encoder may constrain the internal bit depth at each stage using saturation logic (e.g., clipping the internal values to 16-bits).

[0023] In some cases, constraining the bit depth at the transform stages may result in a decrease of final video quality. To prevent this result, the methods described in this disclosure may measure a level of deviations (e.g., errors) caused by the

constraining process. The methods may then compare the level of deviations to a predetermined threshold (e.g., endurance limit) and then re-compute a subset of the transformed coefficients if the level of deviations exceeds the predetermined threshold. By constraining the bit depth and re-computing a subset of the coefficients in this fashion, the methods described in this disclosure may allow a transform processing unit of a video encoder to use fewer computational resources while also preserving video quality.

[0024] In the video codecs using block-based processing (e.g., HEVC, in which video frames may be partitioned into video blocks or coding units), prediction blocks or prediction units (e.g., from inter or intra prediction) may be subtracted from original pixels. As further explained above, the residual data may then be transformed into residual transform coefficients using forward transforms (e.g., discrete cosine transforms), quantized, and entropy encoded (e.g., to achieve further compression). The entropy encoding may be performed using various entropy coding engines (e.g., CAVLC, CABAC, etc.), which are further described below. Afterwards, and as also further described below, a decoder may then entropy decode, dequantize, and inverse transform the coefficients. Finally, the coefficients may be added back to the prediction blocks to form reconstructed pixels.

[0025] In one embodiment of video coding, an image block may first be predicted using pixels in reconstructed temporally and/or spatially neighboring blocks. The prediction error (sometimes referred to as "residue") may then be transformed and quantized. For example, if S is a residue block of size NxN, the transformed block K can be derived using matrix multiplication as follows:

$$K = A * S * B$$

where K, A, and B are also of size NxN. A is the vertical transform matrix and B is the horizontal transform matrix. In some embodiments, A and B are the transpose of each other, (e.g., B = A' where " ' " means transpose). In other embodiments, A and B are not the transpose of each other. When A and B are the transpose of each other, the previous equation becomes:

$$K = A * S * A'$$

[0026] Each transform (A and B) may include any of a variety of transforms. In some embodiments, the transform includes one of a discrete cosine transform (DCT), a discrete sine transform (DST), a Hadamard transform, a Haar transform, etc.

[0027] In an SVC extension, there may be multiple layers of video information. The bottom layer may serve as a base layer (BL), and the top layer may serve as an enhanced layer (EL) or "enhancement layer." All layers between the top and bottom layers may serve as either or both ELs or BLs. SVC may be used to provide quality scalability (or signal-to-noise ratio, SNR), spatial scalability, and/or temporal scalability. An enhanced layer may have different spatial resolution than a base layer. Prediction of a current block may be performed using the different layers that are provided for SVC. Such prediction may be referred to as inter-layer prediction. Inter-layer prediction methods may be utilized in SVC in order to reduce inter-layer redundancy. Some examples of inter-layer prediction may include inter-layer intra prediction, inter-layer motion prediction, and inter-layer residual prediction. Inter-layer intra prediction uses the reconstruction of co-located blocks in the base layer to predict the current block in the enhancement layer. Inter-layer motion prediction uses motion of the base layer to predict motion in the enhancement layer. Inter-layer residual prediction uses the residue of the base layer to predict the residue of the enhancement layer. One particular coding mode for an enhancement layer called the "Intra BL mode" includes a texture that may be predicted using the texture of the corresponding (sometimes referred to as "co-located", e.g., located at the same spatial location) blocks in base layer.

[0028] In inter-layer residual prediction, the residue of the base layer may be used to predict the current block in the enhancement layer. The residue may be defined as the difference between the temporal prediction for a video unit and the source video unit. In residual prediction, the residue of the base layer is also considered in predicting the current block. For example, the current block may be reconstructed using the residue from the enhancement layer, the temporal prediction from the enhancement layer, and the residue from the base layer. The current block may be reconstructed according to the following equation:

$$\hat{I}e = re + Pe + rb$$

where $\hat{I}e$ denotes the reconstruction of the current block, re denotes the residue from the enhancement layer, Pe denotes the temporal prediction from the enhancement layer, and rb denotes the residue prediction from the base layer.

[0029] For inter coding using difference domain, the current predicted block is determined based on the difference values between the corresponding predicted block

samples in the enhancement layer reference picture and the corresponding predicted block samples in the scaled base layer reference picture. The difference values may be referred to as the difference predicted block. The co-located base layer reconstructed samples are added to the difference predicted block in order to obtain enhancement layer prediction samples.

[0030] The techniques described in this disclosure may address issues relating to complex computational requirements during matrix multiplication of forward transforms in HEVC. The techniques may improve the speed, efficiency, and efficacy at which an encoder and/or a transform processing unit may perform the forward transform matrix multiplication.

[0031] Various aspects of the novel systems, apparatuses, and methods are described more fully hereinafter with reference to the accompanying drawings. This disclosure may, however, be embodied in many different forms and should not be construed as limited to any specific structure or function presented throughout this disclosure. Rather, these aspects are provided so that this disclosure will be thorough and complete, and will fully convey the scope of the disclosure to those skilled in the art. Based on the teachings herein one skilled in the art should appreciate that the scope of the disclosure is intended to cover any aspect of the novel systems, apparatuses, and methods disclosed herein, whether implemented independently of, or combined with, any other aspect of the invention. For example, an apparatus may be implemented or a method may be practiced using any number of the aspects set forth herein. In addition, the scope of the invention is intended to cover such an apparatus or method which is practiced using other structure, functionality, or structure and functionality in addition to or other than the various aspects of the invention set forth herein. It should be understood that any aspect disclosed herein may be embodied by one or more elements of a claim.

[0032] Although particular aspects are described herein, many variations and permutations of these aspects fall within the scope of the disclosure. Although some benefits and advantages of the preferred aspects are mentioned, the scope of the disclosure is not intended to be limited to particular benefits, uses, or objectives. Rather, aspects of the disclosure are intended to be broadly applicable to different wireless technologies, system configurations, networks, and transmission protocols, some of which are illustrated by way of example in the figures and in the following

description of the preferred aspects. The detailed description and drawings are merely illustrative of the disclosure rather than limiting, the scope of the disclosure being defined by the appended claims and equivalents thereof.

[0033] FIG. 1 is a block diagram illustrating an example video encoding and decoding system that may utilize techniques in accordance with aspects described in this disclosure. As shown in FIG. 1, a video encoding and decoding system 10 includes a source device 12 that provides encoded video data to be decoded at a later time by a destination device 14. In particular, the source device 12 provides the video data to the destination device 14 via a computer-readable medium 16. The source device 12 and the destination device 14 may comprise any of a wide range of devices, including desktop computers, notebook (e.g., laptop) computers, tablet computers, set-top boxes, telephone handsets (e.g., smartphones), televisions, cameras, display devices, digital media players, video gaming consoles, video streaming devices, etc. In some cases, the source device 12 and the destination device 14 may be equipped for wireless communication.

[0034] As mentioned above, the destination device 14 may receive the encoded video data to be decoded via the computer-readable medium 16. The computer-readable medium 16 may comprise any type of medium or device capable of moving the encoded video data from the source device 12 to the destination device 14. In one example, the computer-readable medium 16 may comprise a communication medium (not pictured) to enable the source device 12 to transmit encoded video data directly to the destination device 14 in real-time. The encoded video data may be modulated according to a communication standard, such as a wireless communication protocol, and transmitted to the destination device 14. 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 also form part of a packet-based network, such as a local area network, a wide-area network, or a global network (e.g., the Internet). The communication medium may include routers, switches, base stations, or any other equipment that may be useful to facilitate communication from the source device 12 to the destination device 14.

[0035] In some examples, the encoded data may be output from an output interface 22 to a storage device (not pictured). Similarly, the encoded data may be accessed from the storage device by an input interface 28. The storage device 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. In a further example, the storage device may correspond to a file server or another intermediate storage device that may store the encoded video generated by the source device 12. The destination device 14 may access stored video data from the storage device via streaming or download. The file server may be any type of server capable of storing encoded video data and transmitting that encoded video data to the destination device 14. Example file servers include a web server (e.g., for a website), an FTP server, network attached storage (NAS) devices, or a local disk drive. The destination device 14 may access the encoded video data through any standard data connection, including an Internet connection. The data connection may include a wireless channel (e.g., a Wi-Fi connection), a wired connection (e.g., DSL, cable modem, etc.), or a combination of both suitable for accessing encoded video data stored on a file server. The transmission of encoded video data from the storage device may be a streaming transmission, a download transmission, or a combination thereof.

[0036] The techniques of this disclosure are not necessarily limited to wireless applications or settings. The techniques 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. In some examples, the system 10 may be configured to support one-way or two-way video transmission for applications such as video streaming, video playback, video broadcasting, video telephony, etc.

[0037] In the example of FIG. 1, the source device 12 includes a video source 18, a video encoder 20, and the output interface 22. The destination device 14 includes the input interface 28, a video decoder 30, and a display device 32. In accordance with this disclosure, the video encoder 20 of the source device 12 may be configured to apply techniques for coding a bitstream including video data conforming to multiple standards or standard extensions. In other examples, the source device 12 and the destination device 14 may include other components or arrangements. For

example, the source device 12 may receive video data from an external video source, such as an external camera. Likewise, the destination device 14 may interface with an external display device, rather than the integrated display device 32.

[0038] Although the techniques of this disclosure are generally performed by a video encoding device, the techniques may also be performed by a video encoder/decoder, typically referred to as a "CODEC." Moreover, the techniques of this disclosure may also be performed by a video preprocessor. The source device 12 and the destination device 14 are merely examples of such coding devices in which the source device 12 generates coded video data for transmission to the destination device 14. In some examples, the source device 12 and the destination device 14 may operate in a substantially symmetrical manner such that they each include video encoding and decoding components. Hence, the system 10 may support one-way or two-way video transmission between the source device 12 and the destination device 14, e.g., for video streaming, video playback, video broadcasting, video telephony, etc.

[0039] The video source 18 of the source device 12 may include a video capture device (not pictured), such as a video camera, a video archive containing previously captured video, a video feed interface to receive video from a video content provider, etc. As a further alternative, the video source 18 may generate computer graphics-based data or a combination of live video, archived video, and computer-generated video. In some cases, if the video source 18 is a video camera, the source device 12 and the destination device 14 may be camera phones or video phones. In another embodiment, the techniques described in this disclosure may be applicable to video coding in general and may be applied to wireless and/or wired applications. In each case, the captured, pre-captured, or computer-generated video may be encoded by the video encoder 20. The encoded video information may then be output by the output interface 22 onto the computer-readable medium 16.

[0040] The computer-readable medium 16 may include transient media, such as a wireless broadcast or wired network transmission, or storage media (that is, non-transitory storage media), such as a hard disk, flash drive, compact disc, digital video disc, Blu-ray disc, or other computer-readable media. In some examples, a network server (not shown) may receive encoded video data from the source device 12 and provide the encoded video data to the destination device 14, e.g., via network transmission, direct wired communication, etc. Similarly, a computing device of a

medium production facility, such as a disc stamping facility, may receive the encoded video data from the source device 12 and produce a disc containing the encoded video data. Therefore, the computer-readable medium 16 may include one or more computer-readable media of various forms.

[0041] The input interface 28 of the destination device 14 may receive information from the computer-readable medium 16. The information of the computer-readable medium 16 may include syntax information defined by the video encoder 20. The syntax information may also be used by the video decoder 30, which may include syntax elements describing characteristics and/or processing of blocks and other coded units. The display device 32 may display the decoded video data to a user and may comprise any of a variety of display devices such as a cathode ray tube (CRT), a liquid crystal display (LCD), a plasma display, an organic light emitting diode (OLED) display, or another type of display device.

[0042] The video encoder 20 and the video decoder 30 may operate according to a video coding standard, such as the High Efficiency Video Coding (HEVC) standard or any of its variations (e.g., the HEVC Test Model (HM)). Alternatively, the video encoder 20 and the video decoder 30 may operate according to other proprietary or industry standards, such as the ITU-T H.264 standard (MPEG-4), Part 10, Advanced Video Coding (AVC), ITU-T H.263, ITU-T H.262 (ISO/IEC MPEG-2 Visual), ISO/IEC MPEG-1 Visual, ITU-T H.261, or extensions of any such standards. In some aspects, the video encoder 20 and the video decoder 30 may be integrated with an audio encoder, audio decoder, MUX-DEMUX units (not pictured), or other hardware and software to handle encoding of both audio and video in a common data stream or separate data streams.

[0043] The video encoder 20 and the video decoder 30 each may be implemented as any of a variety of suitable encoder 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. If necessary to perform the techniques of this disclosure, the video encoder 20 and/or the video decoder 30 may store instructions for software in a suitable, non-transitory computer-readable medium and execute the instructions in hardware using one or more processors. Each of the video encoder 20 and the video decoder 30 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 the video encoder 20 and/or the video decoder 30 may comprise an integrated circuit, a microprocessor, and/or a wireless communication device, such as a cellphone.

[0044] The HEVC standard specifies that a video frame or picture may be divided into a sequence of treeblocks or largest coding units (LCU) that include both luma and chroma samples. Syntax data within a bitstream may define a size for the LCU, which is a largest coding unit in terms of the number of pixels. A slice includes a number of consecutive treeblocks in coding order. A video frame or picture may be partitioned into one or more slices. Each treeblock may be split into coding units (CUs) according to a quadtree. In general, a quadtree data structure includes one node per CU, with a root node corresponding to the treeblock. If a CU is split into four sub-CUs, the node corresponding to the CU includes four leaf nodes, each of which corresponds to one of the sub-CUs.

[0045] Each node of the quadtree data structure may provide syntax data for the corresponding CU. For example, a node in the quadtree may include a split flag, indicating whether the CU corresponding to the node is split into sub-CUs. Syntax elements for a CU may be defined recursively, and may depend on whether the CU is split into sub-CUs. If a CU is not split further, it is referred as a leaf-CU. In this disclosure, four sub-CUs of a leaf-CU will also be referred to as leaf-CUs even if there is no explicit splitting of the original leaf-CU. For example, if a CU at 16x16 size is not split further, the four 8x8 sub-CUs will also be referred to as leaf-CUs although the 16x16 CU was never split.

[0046] A CU has a similar purpose as a macroblock of the H.264 standard, except that a CU does not have a size distinction. For example, a treeblock may be split into four child nodes (also referred to as sub-CUs), and each child node may in turn be a parent node and be split into another four child nodes. A final, unsplit child node, referred to as a leaf node of the quadtree, comprises a coding node, also referred to as a leaf-CU. Syntax data associated with a coded bitstream may define a maximum number of times a treeblock may be split, referred to as a maximum CU depth, and may also define a minimum size of the coding nodes. Accordingly, a bitstream may also define a smallest coding unit (SCU). This disclosure uses the term "block" to refer to any of a CU, PU, or TU, in the context of HEVC, or similar data structures in the context of

other standards (e.g., macroblocks and sub-blocks thereof in H.264/AVC).

[0047] A CU includes a coding node and prediction units (PUs) and transform units (TUs) associated with the coding node. A size of the CU corresponds to a size of the coding node and must be square in shape. The size of the CU may range from 8x8 pixels up to the size of the treeblock with a maximum of 64x64 pixels or, in some instances, greater. Each CU may contain one or more PUs and one or more TUs. Syntax data associated with a CU may describe, for example, partitioning of the CU into one or more PUs. Partitioning modes may differ between whether the CU is skip or direct mode encoded, intra-prediction mode encoded, or inter-prediction mode encoded. PUs may be partitioned to be non-square in shape. Syntax data associated with a CU may also describe, for example, partitioning of the CU into one or more TUs according to a quadtree. A TU can be square or non-square (e.g., rectangular) in shape.

[0048] The HEVC standard allows for transformations according to TUs, which may be different for different CUs. The TUs are typically sized based on the size of PUs within a given CU defined for a partitioned LCU, although this may not always be the case. The TUs are typically the same size or smaller than the PUs. In some examples, residual samples corresponding to a CU may be subdivided into smaller units using a quadtree structure known as "residual quad tree" (RQT). The leaf nodes of the RQT may be referred to as transform units (TUs). Pixel difference values associated with the TUs may be transformed to produce transform coefficients, which may be quantized.

[0049] A leaf-CU may include one or more prediction units (PUs). In general, a PU represents a spatial area corresponding to all or a portion of the corresponding CU, and may include data for retrieving a reference sample for the PU. Moreover, a PU includes data related to prediction. For example, when the PU is intra-mode encoded, data for the PU may be included in a residual quadtree (RQT), which may include data describing an intra-prediction mode for a TU corresponding to the PU. As another example, when the PU is inter-mode encoded, the PU may include data defining one or more motion vectors for the PU. The data defining the motion vector for a PU may describe, for example, a horizontal component of the motion vector, a vertical component of the motion vector, a resolution for the motion vector (e.g., one-quarter pixel precision or one-eighth pixel precision), a reference picture to which the motion vector points, and/or a reference picture list (e.g., List 0, List 1, or List C) for the motion

vector.

[0050] A leaf-CU having one or more PUs may also include one or more transform units (TUs). The transform units may be specified using an RQT (also referred to as a TU quadtree structure), as discussed above. For example, a split flag may indicate whether a leaf-CU is split into four transform units. Then, each transform unit may be split further into further sub-TUs. When a TU is not split further, it may be referred to as a leaf-TU. Generally, for intra coding, all the leaf-TUs belonging to a leaf-CU share the same intra prediction mode. That is, the same intra-prediction mode is generally applied to calculate predicted values for all TUs of a leaf-CU. For intra coding, a video encoder may calculate a residual value for each leaf-TU using the intra prediction mode, as a difference between the portion of the CU corresponding to the TU and the original block. A TU is not necessarily limited to the size of a PU. Thus, TUs may be larger or smaller than a PU. For intra coding, a PU may be collocated with a corresponding leaf-TU for the same CU. In some examples, the maximum size of a leaf-TU may correspond to the size of the corresponding leaf-CU.

[0051] Moreover, TUs of leaf-CUs may also be associated with respective quadtree data structures, referred to as residual quadtrees (RQTs). That is, a leaf-CU may include a quadtree indicating how the leaf-CU is partitioned into TUs. The root node of a TU quadtree generally corresponds to a leaf-CU, while the root node of a CU quadtree generally corresponds to a treeblock (or LCU). TUs of the RQT that are not split are referred to as leaf-TUs. In general, this disclosure uses the terms CU and TU to refer to leaf-CU and leaf-TU, respectively, unless noted otherwise.

[0052] A video sequence typically includes a series of video frames or pictures. A group of pictures (GOP) generally comprises a series of one or more of the video pictures. A GOP may include syntax data in a header of the GOP, a header of one or more of the pictures, or elsewhere, that describes a number of pictures included in the GOP. Each slice of a picture may include slice syntax data that describes an encoding mode for the respective slice. The video encoder 20 of FIG. 1 may operate on video blocks within individual video slices in order to encode the video data. A video block may correspond to a coding node within a CU. The video blocks may have fixed or varying sizes, and may differ in size according to a specified coding standard.

[0053] HEVC supports prediction in various PU sizes. Assuming that the size of a particular CU is 2Nx2N, HEVC supports intra-prediction in PU sizes of

2Nx2N or NxN, and inter-prediction in symmetric PU sizes of 2Nx2N, 2NxN, Nx2N, or NxN. HEVC also supports asymmetric partitioning for inter-prediction in PU sizes of 2NxnU, 2NxnD, nLx2N, and nRx2N. In asymmetric partitioning, one direction of a CU is not partitioned, while the other direction is partitioned into 25% and 75%. The portion of the CU corresponding to the 25% partition is indicated by an "n" followed by an indication of "Up," "Down," "Left," or "Right." Thus, for example, "2NxnU" refers to a 2Nx2N CU that is partitioned horizontally with a 2Nx0.5N PU on top and a 2Nx1.5N PU on the bottom.

[0054] In this disclosure, "NxN" and "N by N" may be used interchangeably to refer to the pixel dimensions of a video block in terms of vertical and horizontal dimensions, e.g., 16x16 pixels or 16 by 16 pixels. In general, a 16x16 block will have 16 pixels in a vertical direction (y = 16) and 16 pixels in a horizontal direction (x = 16). Likewise, an NxN block may have N pixels in a vertical direction and N pixels in a horizontal direction, where N represents a nonnegative integer value. The pixels in a block may be arranged in rows and columns. Moreover, blocks may not necessarily have the same number of pixels in the horizontal direction as in the vertical direction. For example, blocks may comprise NxM pixels, where M is not necessarily equal to N.

[0055] Following intra-predictive or inter-predictive coding using the PUs of a CU, the video encoder 20 may calculate residual data for the TUs of the CU. The PUs may comprise syntax data describing a method or mode of generating predictive pixel data in the spatial domain (also referred to as the pixel domain) and the TUs may comprise coefficients in the transform domain following application of a transform, e.g., a discrete cosine transform (DCT), an integer transform, a wavelet transform, or a conceptually similar transform to residual video data. The residual data may correspond to pixel differences between pixels of the unencoded picture and prediction values corresponding to the PUs. The video encoder 20 may form the TUs including the residual data for the CU, and then transform the TUs to produce transform coefficients for the CU.

[0056] Following any transforms to produce transform coefficients, the video encoder 20 may perform quantization of the transform coefficients. Quantization is a broad term intended to have its broadest ordinary meaning. In one embodiment, quantization refers to a process in which transform coefficients are quantized to possibly reduce the amount of data used to represent the coefficients, providing further

compression. The quantization process may reduce the bit depth associated with some or all of the coefficients. For example, an n-bit value may be rounded down to an m-bit value during quantization, where n is greater than m.

[0057] Following quantization, the video encoder 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) coefficients at the front of the array and to place lower energy (and therefore higher frequency) coefficients at the back of the array. In some examples, the video encoder 20 may utilize a predefined scan order to scan the quantized transform coefficients to produce a serialized vector that can be entropy encoded. In other examples, the video encoder 20 may perform an adaptive scan. After scanning the quantized transform coefficients to form a one-dimensional vector, the video encoder 20 may entropy encode the one-dimensional vector, e.g., according to context-adaptive variable length coding (CAVLC), context-adaptive binary arithmetic coding (CABAC), syntax-based context-adaptive binary arithmetic coding (SBAC), Probability Interval Partitioning Entropy (PIPE) coding or another entropy encoding methodology. The video encoder 20 may also entropy encode syntax elements associated with the encoded video data for use by the video decoder 30 in decoding the video data.

[0058] The video encoder 20 may further send syntax data, such as block-based syntax data, frame-based syntax data, and GOP-based syntax data, to the video decoder 30, e.g., in a frame header, a block header, a slice header, or a GOP header. The GOP syntax data may describe a number of frames in the respective GOP, and the frame syntax data may indicate an encoding/prediction mode used to encode the corresponding frame.

[0059] FIG. 2 is a block diagram illustrating an example of a video encoder that may implement techniques in accordance with aspects described in this disclosure. One or more of the units of the video encoder 20 may be configured to perform any or all of the techniques of this disclosure. As one example, a transform processing unit 52 may be configured to perform any or all of the transform techniques described in this disclosure. However, aspects of this disclosure are not so limited. In some examples, the techniques described in this disclosure may be shared among the various components of the video encoder 20. In some examples, a processor (not shown) may

be configured to perform any or all of the techniques described in this disclosure.

[0060] The video encoder 20 may perform intra- and inter-coding of video blocks within video slices. Intra-coding relies on spatial prediction to reduce or remove spatial redundancy in video within a given video frame or picture. Inter-coding relies on temporal prediction to reduce or remove temporal redundancy in video within adjacent frames or pictures of a video sequence. Intra-mode (I mode) may refer to any of several spatial based coding modes. Inter-modes, such as uni-directional prediction (P mode) or bi-prediction (B mode), may refer to any of several temporal-based coding modes.

[0061] The video encoder 20 may receive a current video block within a video frame to be encoded. In the example of FIG. 2, the video encoder 20 includes a mode select unit 40, a reference frame memory 64, a summer 50, the transform processing unit 52, quantization unit 54, and an entropy encoding unit 56.. The mode select unit 40 includes a motion estimation unit 42, a motion compensation unit 44, an intra-prediction unit 46, and a partition unit 48. For video block reconstruction, the video encoder 20 may also include an inverse quantization unit 58, an inverse transform unit 60, and a summer 62. A deblocking filter (not pictured) may also be included to filter block boundaries to remove blockiness artifacts from reconstructed video. If desired, the deblocking filter would typically filter the output of the summer 62. Additional filters (in loop or post loop) may also be used in addition to the deblocking filter. Such filters are not shown for brevity, but if desired, may filter the output of the summer 50 (as an in-loop filter).

[0062] During the encoding process, the video encoder 20 may receive a video frame or slice to be coded. The frame or slice may be divided into multiple video blocks. The motion estimation unit 42 and the motion compensation unit 44 may perform inter-predictive coding of the received video block relative to one or more blocks in one or more reference frames to provide temporal prediction. The intra-prediction unit 46 may alternatively perform intra-predictive coding of the received video block relative to one or more neighboring blocks in the same frame or slice as the block to be coded to provide spatial prediction. The video encoder 20 may perform multiple coding passes, e.g., to select an appropriate coding mode for each block of video data.

[0063] Moreover, the partition unit 48 may partition blocks of video data

into sub-blocks, based on evaluation of previous partitioning schemes in previous coding passes. For example, partition unit 48 may initially partition a frame or slice into LCUs, and partition each of the LCUs into sub-CUs based on rate-distortion analysis (e.g., rate-distortion optimization). The mode select unit 40 may further produce a quadtree data structure indicative of partitioning of an LCU into sub-CUs. Leaf-node CUs of the quadtree may include one or more PUs and one or more TUs.

[0064] The mode select unit 40 may select one of the coding modes, intra or inter, e.g., based on error results, and provide the resulting intra- or inter-coded block to the summer 50 to generate residual block data and to the summer 62 to reconstruct the encoded block for use as a reference frame. The mode select unit 40 may also provide syntax elements, such as motion vectors, intra-mode indicators, partition information, and other such syntax information, to the entropy encoding unit 56.

[0065] The motion estimation unit 42 and the motion compensation unit 44 may be highly integrated, but are illustrated separately for conceptual purposes. Motion estimation, performed by the motion estimation unit 42, is the process of generating motion vectors, which estimate motion for video blocks. A motion vector, for example, may indicate the displacement of a PU of a video block within a current video frame or picture relative to a predictive block within a reference frame (or other coded unit) relative to the current block being coded within the current frame (or other coded unit). A predictive block is a block that is found to closely match the block to be coded, in terms of pixel difference, which may be determined by sum of absolute difference (SAD), sum of square difference (SSD), or other difference metrics. In some examples, the video encoder 20 may calculate values for sub-integer pixel positions of reference pictures stored in the reference frame memory 64. For example, the video encoder 20 may interpolate values of one-quarter pixel positions, one-eighth pixel positions, or other fractional pixel positions of the reference picture. Therefore, the motion estimation unit 42 may perform a motion search relative to the full pixel positions and fractional pixel positions and output a motion vector with fractional pixel precision.

[0066] The motion estimation unit 42 calculates a motion vector for a PU of a video block in an inter-coded slice by comparing the position of the PU to the position of a predictive block of a reference picture. The reference picture may be selected from a first reference picture list (List 0) or a second reference picture list (List 1), each of which identify one or more reference pictures stored in the reference frame memory 64.

The motion estimation unit 42 sends the calculated motion vector to the entropy encoding unit 56 and the motion compensation unit 44.

[0067] Motion compensation, performed by the motion compensation unit 44, may involve fetching or generating the predictive block based on the motion vector determined by the motion estimation unit 42. Again, the motion estimation unit 42 and the motion compensation unit 44 may be functionally integrated, in some examples. Upon receiving the motion vector for the PU of the current video block, the motion compensation unit 44 may locate the predictive block to which the motion vector points in one of the reference picture lists. The summer 50 may form a residual video block by subtracting pixel values of the predictive block from the pixel values of the current video block being coded, forming pixel difference values, as discussed below. The motion estimation unit 42 may perform motion estimation relative to luma components, and the motion compensation unit 44 may use motion vectors calculated based on the luma components for both chroma components and luma components. The mode select unit 40 may also generate syntax elements associated with the video blocks and the video slice for use by the video decoder 30 in decoding the video blocks of the video slice. The syntax elements may represent prediction information at one or more of a video sequence level, a video frame level, a video slice level, a video CU level, or a video PU level. For example, the motion compensation unit 44 may generate syntax elements indicating video block information including sizes of CUs, PUs, and TUs, and motion vector information for intra-mode prediction.

[0068] The intra-prediction unit 46 may intra-predict or calculate a current block, as an alternative to the inter-prediction performed by motion estimation unit 42 and motion compensation unit 44, as described above. In particular, intra-prediction unit 46 may determine an intra-prediction mode to use to encode a current block. In some examples, the intra-prediction unit 46 may encode a current block using various intra-prediction modes, e.g., during separate encoding passes, and the intra-prediction unit 46 (or the mode select unit 40, in some examples) may select an appropriate intra-prediction mode to use from the tested modes.

[0069] For example, the intra-prediction unit 46 may calculate rate-distortion values using a rate-distortion analysis for the various tested intra-prediction modes, and select the intra-prediction mode having the best rate-distortion characteristics among the tested modes. Rate-distortion analysis generally determines

an amount of distortion (or error) between an encoded block and an original, unencoded block that was encoded to produce the encoded block, as well as a bitrate (that is, a number of bits) used to produce the encoded block. The intra-prediction unit 46 may calculate ratios from the distortions and rates for the various encoded blocks to determine which intra-prediction mode exhibits the best rate-distortion value for the block.

[0070] After selecting an intra-prediction mode for a block, the intra-prediction unit 46 may provide information indicative of the selected intra-prediction mode for the block to the entropy encoding unit 56. The entropy encoding unit 56 may encode the information indicating the selected intra-prediction mode. The video encoder 20 may include in the transmitted bitstream configuration data, which may include a plurality of intra-prediction mode index tables and a plurality of modified intra-prediction mode index tables (also referred to as codeword mapping tables), definitions of encoding contexts for various blocks, and indications of a most probable intra-prediction mode, an intra-prediction mode index table, and a modified intra-prediction mode index table to use for each of the contexts.

[0071] The video encoder 20 forms a residual video block by subtracting the prediction data from the mode select unit 40 from the original video block being coded. The summer 50 may perform this subtraction operation. The transform processing unit 52 may apply a transform, such as a discrete cosine transform (DCT) or a conceptually similar transform, to the residual block, producing a video block comprising residual transform coefficient values. The transform processing unit 52 may perform other transforms which are conceptually similar to DCT. Wavelet transforms, integer transforms, sub-band transforms or other types of transforms may also be used. The transform processing unit 52 may then apply the transform to the residual block, producing a block of residual transform coefficients. The transform processing unit 52 may convert the residual information from a pixel value domain to a transform domain, such as a frequency domain. More specifically, prior to application of the transform, the TU may comprise residual video data in the pixel domain, and, following application of the transform, the TU may comprise transform coefficients that represent the residual video data in the frequency domain.

[0072] Conventionally, the video encoder 20 maintains separate context models for each of the different sizes of TUs supported by the implemented video

compression standard. For the HEVC standard, additional transform unit sizes, e.g., 32x32 up to 128x128, may be used to improve video coding efficiency, but the additional TU sizes also result in increased memory and computational requirements to maintain the context models for each of the additional transform unit sizes. In some cases, the larger TU sizes may use more contexts, which may result in increased memory and computational requirement to maintain the increased number of contexts for the larger TU sizes. To reduce the effects of this problem, the transform processing unit 52 may further be configured to perform any of the methods described above and below in regards to simplifying the transform (sometimes referred to as a "forward transform") and constraining its bit depth during matrix multiplication (e.g., the internal bit depth constraint methods described below and in regards to FIGS. 4 – 5).

[0073] The transform processing unit 52 may send the resulting transform coefficients to the quantization unit 54. The quantization unit 54 may then quantize the transform coefficients to further reduce the bit rate. The quantization process may reduce the bit depth associated with some or all of the coefficients. The degree of quantization may be modified by adjusting a quantization parameter. In some examples, the quantization unit 54 may then perform a scan of the matrix including the quantized transform coefficients. Alternatively, the entropy encoding unit 56 may perform the scan.

[0074] As an example, the transform processing unit 52 may constrain transform results to a predetermined bit depth value (e.g., a 16-bit bit depth or other bit depth value). In one implementation, the transform processing unit 52 may constrain the transform results at one or more internal transform stages. The video encoder 20 may then utilize specialized instruction sets optimized for the predetermined bit depth value. In this way, the transform processing unit 52 may experience faster processing speeds. This process is further described and demonstrated in FIG. 4.

[0075] In the example described above, the transform processing unit 52 is configured to constrain the transform results to 16-bit values. In other cases, the transform processing unit 52 may be further configured to determine that the constrained values will yield satisfactory results. This process is further described and demonstrated in FIG. 5.

[0076] Following quantization, the entropy encoding unit 56 may entropy code the quantized transform coefficients. For example, the entropy encoding unit 56

may perform context adaptive variable length coding (CAVLC), context adaptive binary arithmetic coding (CABAC), syntax-based context-adaptive binary arithmetic coding (SBAC), probability interval partitioning entropy (PIPE) coding or another entropy encoding technique. In the case of context-based entropy encoding, context may be based on neighboring blocks. Following the entropy encoding by the entropy encoding unit 56, the encoded bitstream may be transmitted to another device (e.g., the video decoder 30) or archived for later transmission or retrieval.

[0077] The inverse quantization unit 58 and the inverse transform unit 60 may apply inverse quantization and inverse transformation, respectively, to reconstruct the residual block in the pixel domain, e.g., for later use as a reference block. The motion compensation unit 44 may calculate a reference block by adding the residual block to a predictive block of one of the frames of the reference frame memory 64. The motion compensation unit 44 may also apply one or more interpolation filters to the reconstructed residual block to calculate sub-integer pixel values for use in motion estimation. The summer 62 may add the reconstructed residual block to the motion compensated prediction block produced by the motion compensation unit 44 to produce a reconstructed video block for storage in the reference frame memory 64. The reconstructed video block may then be used by the motion estimation unit 42 and the motion compensation unit 44 as a reference block to inter-code a block in a subsequent video frame.

[0078] FIG. 3 is a block diagram illustrating an example of a video decoder that may implement techniques in accordance with aspects described in this disclosure. The techniques described in this disclosure may utilize various components of the video decoder 30. In some examples, a processor (not shown) may be configured to perform any or all of the techniques.

[0079] In the example of FIG. 3, the video decoder 30 includes an entropy decoding unit 70, a prediction unit 81 further including a motion compensation unit 72 and an intra-prediction unit 74, an inverse quantization unit 76, an inverse transformation unit 78, a reference picture (frame) memory 82, and a summer 80. The video decoder 30 may perform a decoding pass generally reciprocal to the encoding pass described with respect to the video encoder 20 (e.g., see FIG. 1 and FIG. 2). The motion compensation unit 72 may generate prediction data based on motion vectors received from the entropy decoding unit 70, while the intra-prediction unit 74 may

generate prediction data based on intra-prediction mode indicators received from the entropy decoding unit 70.

[0080] Conventionally, the video decoder 30 would maintain separate context models for each of the different sizes of TUs supported by the implemented video compression standard. For the HEVC standard, additional transform unit sizes, e.g., 32x32 up to 128x128, may be utilized to improve video coding efficiency, but the additional TU sizes also result in increased memory and computational requirements to maintain the context models for each of the additional transform unit sizes.

[0081] During the decoding process, the video decoder 30 may receive an encoded video bitstream that represents video blocks of an encoded video slice and associated syntax elements from the video encoder 20. The entropy decoding unit 70 of the video decoder 30 entropy decodes the bitstream to generate quantized coefficients, motion vectors or intra-prediction mode indicators, and other syntax elements. The entropy decoding unit 70 may then forward the motion vectors to and other syntax elements to the motion compensation unit 72. The video decoder 30 may receive the syntax elements at the video slice level and/or the video block level.

[0082] When the video slice is coded as an intra-coded (I) slice, the intra prediction unit 74 may generate prediction data for a video block of the current video slice based on a signaled intra prediction mode and data from previously decoded blocks of the current frame or picture. When the video frame is coded as an inter-coded (e.g., B, P or GPB) slice, the motion compensation unit 72 may produce predictive blocks for a video block of the current video slice based on the motion vectors and other syntax elements received from the entropy decoding unit 70. The predictive blocks may be produced from one of the reference pictures within one of the reference picture lists. The video decoder 30 may construct the reference frame lists, List 0 and List 1, using default construction techniques based on reference pictures stored in the reference picture (frame) memory 82. The motion compensation unit 72 may determine prediction information for a video block of the current video slice by parsing the motion vectors and other syntax elements, and use the prediction information to produce the predictive blocks for the current video block being decoded. For example, the motion compensation unit 72 may use some of the received syntax elements to determine a prediction mode (e.g., intra- or inter-prediction) used to code the video blocks of the video slice, an inter-prediction slice type (e.g., B slice, P slice, or GPB slice),

construction information for one or more of the reference picture lists for the slice, motion vectors for each inter-encoded video block of the slice, inter-prediction status for each inter-coded video block of the slice, and other information to decode the video blocks in the current video slice.

[0083] The motion compensation unit 72 may also perform interpolation based on interpolation filters. The motion compensation unit 72 may use interpolation filters as used by the video encoder 20 during encoding of the video blocks to calculate interpolated values for sub-integer pixels of reference blocks. In this case, the motion compensation unit 72 may determine the interpolation filters used by the video encoder 20 from the received syntax elements and use the interpolation filters to produce predictive blocks.

[0084] The inverse quantization unit 76 may inverse quantize, e.g., de-quantize, the quantized transform coefficients provided in the bitstream and decoded by the entropy decoding unit 70. The inverse quantization process may include use of a quantization parameter QPY calculated by the video decoder 30 for each video block in the video slice to determine a degree of quantization and, likewise, a degree of inverse quantization that should be applied.

[0085] The inverse transform unit 78 may apply an inverse transform, e.g., an inverse DCT, an inverse integer transform, or a conceptually similar inverse transform process, to the transform coefficients in order to produce residual blocks in the pixel domain. After the motion compensation unit 72 generates the predictive block for the current video block based on the motion vectors and other syntax elements, the video decoder 30 may form a decoded video block by summing the residual blocks from the inverse transform unit 78 with the corresponding predictive blocks generated by the motion compensation unit 72. The summer 80 may perform this summation operation. A deblocking filter may also be applied to filter the decoded blocks in order to remove blockiness artifacts. Other loop filters (either in the coding loop or after the coding loop) may also be used to smooth pixel transitions, or otherwise improve the video quality. The decoded video blocks in a given frame or picture may then be stored in the reference picture (frame) memory 82, which may store reference pictures used for subsequent motion compensation. The reference picture (frame) memory 82 may also store decoded video for later presentation on a display device, such as the display device 32 of FIG. 1.

[0086] FIG. 4 illustrates a flowchart of a method 400 of low complexity forward transformation, for example, using mesh-based calculations. The method 400 may be performed by a processor or encoder, such as, for example, the encoder 20 of FIG. 2. In one embodiment, the transform processing unit 52 of the encoder 20 of FIG. 2 is configured to perform the method 400. Although the various blocks of the methods are described as being performed by a transform processing unit, it should be understood that the various block of the methods may be performed by other processors, encoders, or units thereof.

[0087] The method 400 allows an encoder to constrain transform results to 16-bit values to enable performing 16-bit operations using specialized instruction sets optimized for 16-bit data (e.g., ARM architecture, Advanced SIMD (NEON), Digital Signal Processing (DSP), etc.). As described above, constraining the bit depth may improve coding efficiency and reduce computational resource requirements during video encoding. However, in some cases, constraining the bit depth may also decrease the quality of the transform output. Therefore, prior to performing the 16-bit operations, the encoder (e.g., transform processing unit 52 of encoder 20) may be further configured to determine that the constrained values will yield satisfactory results and adjust accordingly, which is further described in regards to FIG. 5.

[0088] The method 400 begins at block 405. Then at block 410, the transform processing unit 52 may decompose a full-size forward transform matrix into multiple, less complex stages (e.g., determine multi-stage transform matrices). Decomposition may occur by using a mesh-based decomposition method. In one implementation, rather than using computational resources to decompose the full-size transform matrix, the transform processing unit 52 may instead retrieve the multiple stages from a memory or a coded bitstream. In some implementations, the initial forward transform matrix may be a 32x32 transform matrix. In other implementations, the transform matrix may be of another size.

[0089] Then at block 415, the transform processing unit 52 may perform appropriate transform operations on each of the multiple stages. In some cases, after transform operations have been performed, one or more of the stages may contain transform results that are greater than 16-bits. These transform results may require internal computational operations designed for systems greater than 16-bits (e.g., 32-bit operations), which may decrease the efficiency of the transform processing unit 52.

[0090] Therefore, at block 420, the transform processing unit 52 may constrain the internal bit depth of the transform results at each of the multiple stages to 16-bit values. The transform processing unit 52 may perform this constrained process using saturation logic, e.g., by clipping the internal values to 16-bit values. More specifically, the transform processing unit 52 may constrain the bit depth internally during the transform operation rather than only constraining the transform output bit depth. To accomplish this, the transform processing unit 52 may perform a clipping function at the output of each intermediate transform stage. In one implementation, the transform processing unit 52 may use a clipping function appended to the code illustrated in Appendix A to perform the constrained process, where a and b represent values to be added with the sum constrained to be within the 16-bit range. One example clipping function may be represented as $y = CLIP_3(x, min_{val}, max_{val})$. In this example function, x may represent a number of bytes. If x is less than min_{val}, then y may be set to min_{val}. Further, if x is greater than max_{val}, then y may be set to max_{val}. If x falls between the range of min_{val} and max_{val}, then y may be set to x. A more specific exemplary clipping function may be represented as one of either $(a + b)_{sat} = CLIP_3(-32768, 32767, a + b)$ or $(a_0 * b_0 + a_1 * b_1 + \cdots)_{sat} = CLIP_3(-32768, 32767, a_0 * b_0 + a_1 * b_1 + \cdots)$, which follow similar logic trees as the above clipping example. Other structures of clipping functions may be implemented to perform the constraint process. In any case, for the addition, subtraction, and/or the multiplication and addition operations, the saturation logic may constrain the results to 16-bit.

[0091] Constraining the internal bit depth to 16-bits at each stage may enable the transform processing unit 52 to utilize computationally efficient instruction sets for that level. For example, an input bit depth to a transform may be 9-bits and a starting internal bit depth may be greater than 16-bits. In one example, the transform processing unit 52 may only be capable of performing operations at specific bit-levels (e.g., 16-bit operations, 32-bit operations, or 64-bit operations, etc.). Therefore, when the internal bit depth is greater than 16-bits, the transform processing unit 52 may be required to perform internal operations at at least a 32-bit level. To avoid this result, the transform processing unit 52 may constrain the internal bit depth of each decomposed stage of the transform to 16-bits (e.g., so that 32-bit internal operations are not required). In other words, with the internal bit depth of each stage constrained to 16-

bits, the processor may utilize computationally efficient instruction sets that are designed specifically for use with 16-bit operations (e.g., ARM architecture, Advanced SIMD (NEON), Digital Signal Processing (DSP), etc.).

[0092] Then at optional block 425, the transform processing unit 52 may optionally determine whether the constrained values will yield satisfactory results by comparing the constrained values deviation (e.g., error) at each stage to a threshold. For example, the method 500 may determine whether the constrained values will yield satisfactory results by performing blocks 510 and 520 of the method 500 discussed below with respect to FIG. 5, at block 425. If the constrained values will yield satisfactory results, the method 400 continues to block 430. Otherwise, at block 427, the transform processing unit 52 may re-compute a subset of the coefficients until the deviation is lower than the threshold (e.g., as discussed below with respect to block 530 of method 500). This method 500 may allow the transform processing unit 52 to increase coding efficiency while also preserving video quality. Once the transform processing unit 52 has determined that the constrained values will yield satisfactory results, then at block 430, the transform processing unit 52 may proceed with performing 16-bit operations using specialized instruction sets optimized for 16-bit data (e.g., ARM architecture, Advanced SIMD (NEON), Digital Signal Processing (DSP), etc.) on the transform data. At block 490, the method ends.

[0093] FIG. 5 illustrates a flowchart of a method 500 for a low complexity forward transformation. The method 500 may be performed by a processor or encoder, such as the encoder 20 described above with respect to FIG. 2. In one embodiment, a transform processing unit of an encoder (e.g., the transform processing unit 52 of the encoder 20 of FIG. 2) may be used to perform the method 500. Indeed, although the method 500 is described as performed by the transform processing unit 52 of the encoder 20, it should be understood that the method 500 may be performed by a different processor, encoder, or processing unit of an encoder. In one embodiment, the method 500 first determines whether constraining values (e.g., the constrained values from the method of FIG. 4) will yield satisfactory results and adjusts accordingly. As explained above in regards to FIG. 4, when the transform processing unit 52 constrains the internal bit depth, it may also decrease the quality of the transform output. The methods described in regards to FIG. 5 determine to what extent a decrease in quality may occur and adjust accordingly until the deviation is below a predetermined

threshold.

[0094] The method 500 begins at block 505. At the beginning of the method 500, the transform processing unit 52 has already decomposed a full-size forward transform matrix into multiple, less complex stages. For example, the forward transform matrix may have been decomposed according to the methods described in regards to FIG. 4. The internal bit depth of each stage may have already been constrained to a particular bit depth value (e.g., a 16-bit bit depth value), as further discussed in regards to FIG. 4.

[0095] Then at block 510, the transform processing unit 52 may determine a deviation of the constrained values (e.g., the final coefficients) from the original values. As an example, when saturation logic is used, the final coefficient values may deviate from the original values (e.g., contain a level of error).

[0096] After calculating the deviation, at block 520, the transform processing unit 52 may determine whether the deviation is greater than a predetermined threshold limit. If it is, then the method 500 proceeds to block 530. At block 530, the transform processing unit 52 may re-compute a subset of the coefficients (e.g., by partially applying the transform to re-compute a subset of the AC-level coefficients) and replace that same subset of constrained values with the recomputed results. This process may compensate for the deviation that the final coefficient values have from the original values while also minimizing computation requirements (e.g., cycles) and complexity. In one implementation, the subset of the coefficients may contain only the DC-level coefficients, because the DC-level coefficients have larger dynamic ranges than the AC-level coefficients; therefore, they are more likely to deviate from the original values. In other implementations, and to further increase accuracy, the subset of the coefficients may contain AC-level coefficients with values near the DC-level in addition to containing the DC-level coefficients.

[0097] After re-computing the small subset of coefficients as explained above, the transform processing unit may then return to block 510 and check the deviation against the threshold again. Once the transform processing unit 52 has determined that the deviation is less than the predetermined threshold endurance limit (at block 520), the method 500 proceeds to block 590 and ends. At block 590, the transform processing unit 52 may determine that the constrained values will yield satisfactory results, and provide the re-computed subset of coefficients. The transform

processing unit 52 may then utilize the re-computed subset of coefficients at block 427 of the method 400, discussed above with respect to FIG. 4.

[0098] All of the above methods were described in regards to an NxN transform. However, the methods described in this disclosure are not limited to NxN transforms. The methods may also be implemented in regards to any size transforms, regardless of the number of dimensions in the transforms.

[0099] 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.

[00100] 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.

[00101] 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.

[00102] Instructions may be executed by one or more processors, such as one or more digital signal processors (DSPs), general purpose microprocessors, application specific integrated circuits (ASICs), field programmable logic arrays (FPGAs), or other equivalent integrated or discrete logic circuitry. Accordingly, the term "processor," as used herein may refer to any of the foregoing structure or any other structure suitable for implementation of the techniques described herein. In addition, in some aspects, the functionality described herein may be provided within dedicated hardware and/or software modules configured for encoding and decoding, or incorporated in a combined codec. Also, the techniques could be fully implemented in one or more circuits or logic elements.

[00103] 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.

[00104] Various examples have been described. These and other examples are within the scope of the following claims.

Appendix A: Example of a Mesh-Based Implementation of a 16x16 Forward Transform

```
for (j=0; j<16; j++)
{
 /* E and O*/
 for (k=0;k<8;k++)
 {
  nE[k] = pSrc[k] + pSrc[15-k];
  nO[k] = pSrc[k] - pSrc[15-k];
 }
 /* EE and EO */
 for (k=0;k<4;k++)
 {
  nEE[k] = nE[k] + nE[7-k];
  nEO[k] = nE[k] - nE[7-k];
 }
 /* EEE and EEO */
 nEEE[0] = nEE[0] + nEE[3];
 nEEO[0] = nEE[0] - nEE[3];
 nEEE[1] = nEE[1] + nEE[2];
 nEEO[1] = nEE[1] - nEE[2];

 pDst[ 0 ] = (anTransCoef16[ 0][0]*nEEE[0] + anTransCoef16[ 0][1]*nEEE[1] +
   4)>>3;
 pDst[ 8 ] = (anTransCoef16[ 8][0]*nEEE[0] + anTransCoef16[ 8][1]*nEEE[1] +
   4)>>3;
 pDst[ 4 ] = (anTransCoef16[ 4][0]*nEEO[0] + anTransCoef16[ 4][1]*nEEO[1] +
   4)>>3;
 pDst[ 12] = (anTransCoef16[12][0]*nEEO[0] + anTransCoef16[12][1]*nEEO[1] +
   4)>>3;

 for (k=2;k<16;k+=4)
 {
  pDst[ k ] = (anTransCoef16[k][0]*nEO[0] + anTransCoef16[k][1]*nEO[1] +
   anTransCoef16[k][2]*nEO[2] + anTransCoef16[k][3]*nEO[3] + 4)>>3;
 }
```

```
 for (k=1;k<16;k+=2)
 {
  pDst[ k ] = (anTransCoef16[k][0]*nO[0] + anTransCoef16[k][1]*nO[1] +
   anTransCoef16[k][2]*nO[2] + anTransCoef16[k][3]*nO[3] +
    anTransCoef16[k][4]*nO[4] + anTransCoef16[k][5]*nO[5] +
   anTransCoef16[k][6]*nO[6] + anTransCoef16[k][7]*nO[7] + 4)>>3;
 }

 pSrc += 16;
 pDst += 16;

}

pSrc = pCoef;
pDst = pRes;
for (j=0; j<16; j++)
{
 /* E and O*/
 for (k=0;k<8;k++)
 {
  nE[k] = pSrc[k*16] + pSrc[(15-k)*16];
  nO[k] = pSrc[k*16] - pSrc[(15-k)*16];
 }
 /* EE and EO */
 for (k=0;k<4;k++)
 {
  nEE[k] = nE[k] + nE[7-k];
  nEO[k] = nE[k] - nE[7-k];
 }
 /* EEE and EEO */
 nEEE[0] = nEE[0] + nEE[3];
 nEEO[0] = nEE[0] - nEE[3];
 nEEE[1] = nEE[1] + nEE[2];
 nEEO[1] = nEE[1] - nEE[2];
```

```
pDst[ 0    ] = (anTransCoef16[ 0][0]*nEEE[0] + anTransCoef16[ 0][1]*nEEE[1] +
  512)>>10;
pDst[ 8*16 ] = (anTransCoef16[ 8][0]*nEEE[0] + anTransCoef16[ 8][1]*nEEE[1] +
  512)>>10;
pDst[ 4*16 ] = (anTransCoef16[ 4][0]*nEEO[0] + anTransCoef16[ 4][1]*nEEO[1] +
  512)>>10;
pDst[ 12*16] = (anTransCoef16[12][0]*nEEO[0] + anTransCoef16[12][1]*nEEO[1]
  + 512)>>10;

for (k=2;k<16;k+=4)
{
 pDst[ k*16 ] = (anTransCoef16[k][0]*nEO[0] + anTransCoef16[k][1]*nEO[1] +
  anTransCoef16[k][2]*nEO[2] + anTransCoef16[k][3]*nEO[3] + 512)>>10;
}

for (k=1;k<16;k+=2)
{
 pDst[ k*16 ] = (anTransCoef16[k][0]*nO[0] + anTransCoef16[k][1]*nO[1] +
  anTransCoef16[k][2]*nO[2] + anTransCoef16[k][3]*nO[3] +
   anTransCoef16[k][4]*nO[4] + anTransCoef16[k][5]*nO[5] +
  anTransCoef16[k][6]*nO[6] + anTransCoef16[k][7]*nO[7] + 512)>>10;
}

pSrc ++;
pDst ++;

}
```

WHAT IS CLAIMED IS:

1. A video encoder, comprising:

a memory configured to store video information; and

a processor in communication with the memory, the processor configured to:

decompose a transform into multiple transform stages;

transform the video information using the multiple stages to determine a transform stage output at each transform stage;

constrain the transform stage output at each transform stage to a predetermined bit depth; and

perform operations on the constrained transform output of a last stage of the multiple stages, wherein the operations are only available for use with data having the predetermined bit depth.

2. The video encoder of Claim 1, wherein the processor is further configured to decompose the transform using a mesh-based method, a butterfly method, or even-odd decomposition.

3. The video encoder of Claim 1, wherein the processor is further configured to constrain the transform stage output by clipping the transform stage output such that the clipped transform stage output necessarily falls within a predetermined range.

4. The video encoder of Claim 1, wherein said predetermined bit depth is 16-bits.

5. The video encoder of Claim 1, wherein the processor is further configured to determine a deviation between the constrained transform stage output and the transform stage output.

6. The video encoder of Claim 5, wherein the deviation comprises the difference between the constrained transform stage output and the transform stage output.

7. The video encoder of Claim 5, wherein the processor is further configured to re-compute a subset of coefficients within at least one transform stage when the deviation is greater than a predetermined threshold value.

8. The video encoder of Claim 7, wherein the processor is further

configured to transform the video information using at least the re-computed subset of coefficients.

9. A method of encoding video, the method comprising:

storing video information;

decomposing a transform into multiple transform stages;

transforming the video information using the multiple stages to determine a transform stage output at each transform stage;

constraining the transform stage output at each transform stage to a predetermined bit depth; and

performing operations on the constrained transform output of a last stage of the multiple stages, wherein the operations are only available for use with data having the predetermined bit depth.

10. The method of Claim 9, further comprising decomposing the transform using a mesh-based method, a butterfly method, or even-odd decomposition.

11. The method of Claim 9, further comprising constraining the transform stage output by clipping the transform stage output such that the clipped transform stage output necessarily falls within a predetermined range.

12. The method of Claim 9, wherein said predetermined bit depth is 16-bits.

13. The method of Claim 9, further comprising determining a deviation between the constrained transform stage output and the transform stage output.

14. The method of Claim 13, wherein the deviation comprises the difference between the constrained transform stage output and the transform stage output.

15. The method of Claim 13, further comprising re-computing a subset of coefficients within at least one transform stage when the deviation is greater than a predetermined threshold value.

16. The method of Claim 15, further comprising transforming the video information using at least the re-computed subset of coefficients.

17. A non-transitory computer-readable medium comprising code that, when executed, causes an apparatus to:

store video information;

decompose a transform into multiple transform stages;

transform the video information using the multiple stages to determine a transform stage output at each transform stage;

constrain the transform stage output at each transform stage to a predetermined bit depth; and

perform operations on the constrained transform output of a last stage of the multiple stages, wherein the operations are only available for use with data having the predetermined bit depth.

18. The computer-readable medium of Claim 17, further comprising code that, when executed, causes the apparatus to decompose the transform using a mesh-based method, a butterfly method, or even-odd decomposition.

19. The computer-readable medium of Claim 17, further comprising code that, when executed, causes the apparatus to constrain the transform stage output by clipping the transform stage output such that the clipped transform stage output necessarily falls within a predetermined range.

20. The computer-readable medium of Claim 17, further comprising code that, when executed, causes the apparatus to determine a deviation between the constrained transform stage output and the transform stage output.

21. The computer-readable medium of Claim 20, wherein the deviation comprises the difference between the constrained transform stage output and the transform stage output.

22. The computer-readable medium of Claim 20, further comprising code that, when executed, causes the apparatus to re-compute a subset of coefficients within at least one transform stage when the deviation is greater than a predetermined threshold value.

23. The computer-readable medium of Claim 22, further comprising code that, when executed, causes the apparatus to transform the video information using at least the re-computed subset of coefficients.

24. An apparatus for encoding video, comprising:

means for storing video information;

means for decomposing a transform into multiple transform stages;

means for transforming the video information using the multiple stages to determine a transform stage output at each transform stage;

means for constraining the transform stage output at each transform stage to a predetermined bit depth; and

means for performing operations on the constrained transform output of a last stage of the multiple stages, wherein the operations are only available for use with data having the predetermined bit depth.

25. The apparatus of Claim 25, further comprising means for decomposing the transform using a mesh-based method, a butterfly method, or even-odd decomposition.

26. The apparatus of Claim 25, further comprising means for constraining the transform stage output by clipping the transform stage output such that the clipped transform stage output necessarily falls within a predetermined range.

27. The apparatus of Claim 25, further comprising means for determining a deviation between the constrained transform stage output and the transform stage output.

28. The apparatus of Claim 27, wherein the deviation comprises the difference between the constrained transform stage output and the transform stage output.

29. The apparatus of Claim 27, further comprising means for re-computing a subset of coefficients within at least one transform stage when the deviation is greater than a predetermined threshold value.

30. The apparatus of Claim 30, further comprising means for transforming the video information using at least the re-computed subset of coefficients.

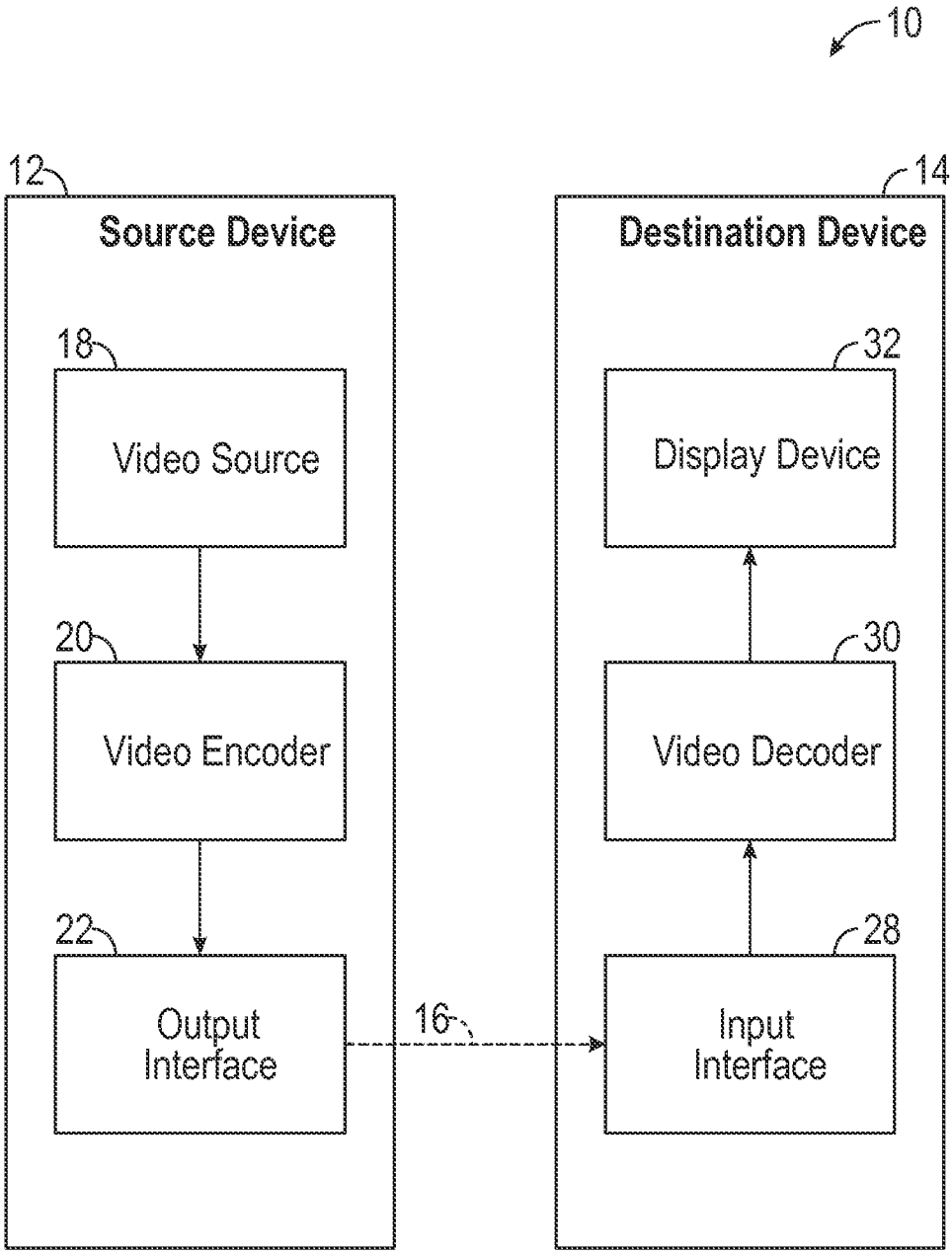
10
12
Source Device
14
Destination Device
18
Video Source
20
Video Encoder
22
Output Interface
16
32
Display Device
30
Video Decoder
28
Input Interface

FIG. 1

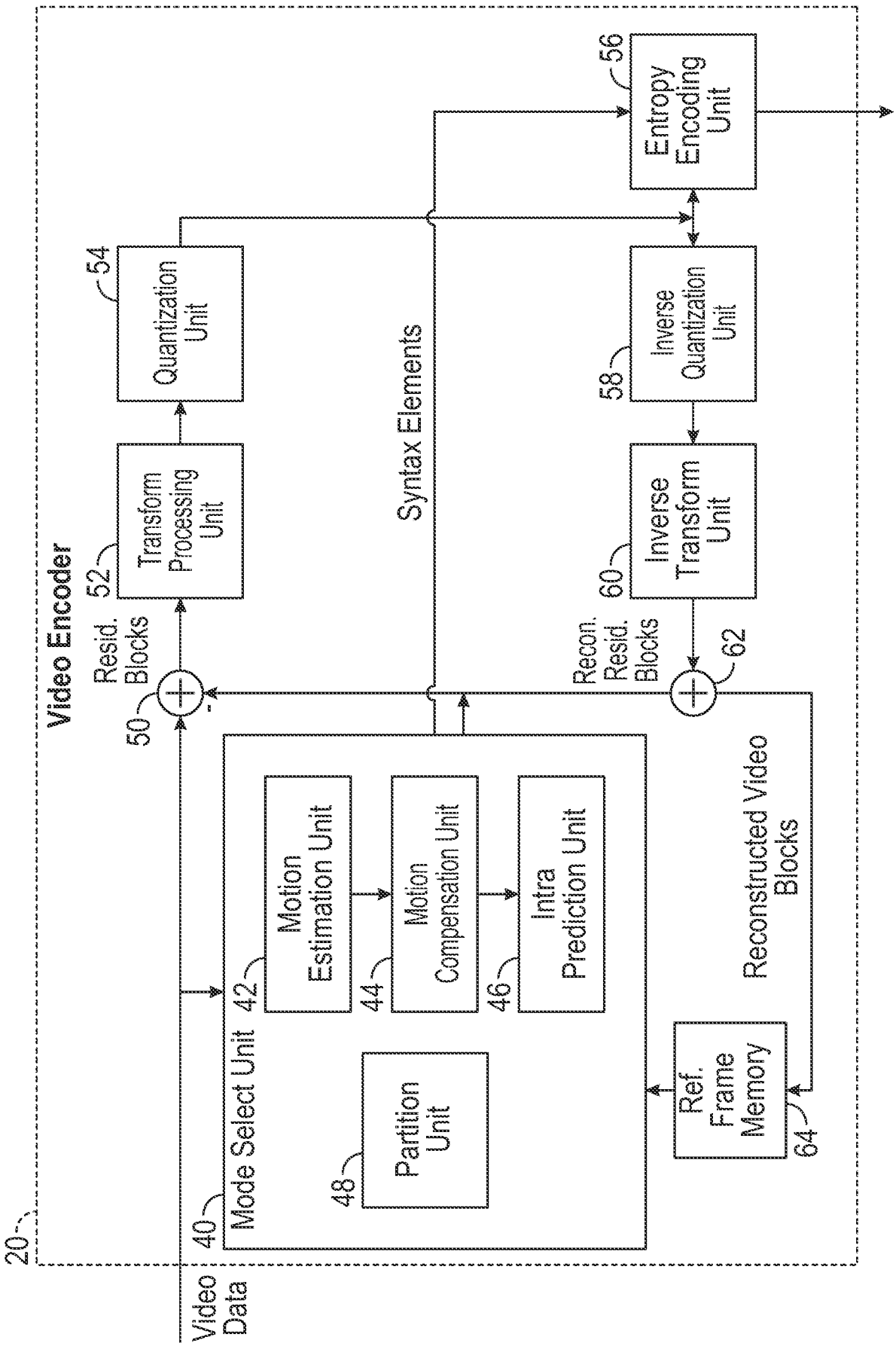
Video Encoder
20
Video Data
Mode Select Unit
40
Motion Estimation Unit
42
Motion Compensation Unit
44
Intra Prediction Unit
46
Partition Unit
48
Resid. Blocks
50
Transform Processing Unit
52
Quantization Unit
54
Syntax Elements
Entropy Encoding Unit
56
Inverse Quantization Unit
58
Inverse Transform Unit
60
Recon. Resid. Blocks
62
Reconstructed Video Blocks
Ref. Frame Memory
64

FIG. 2

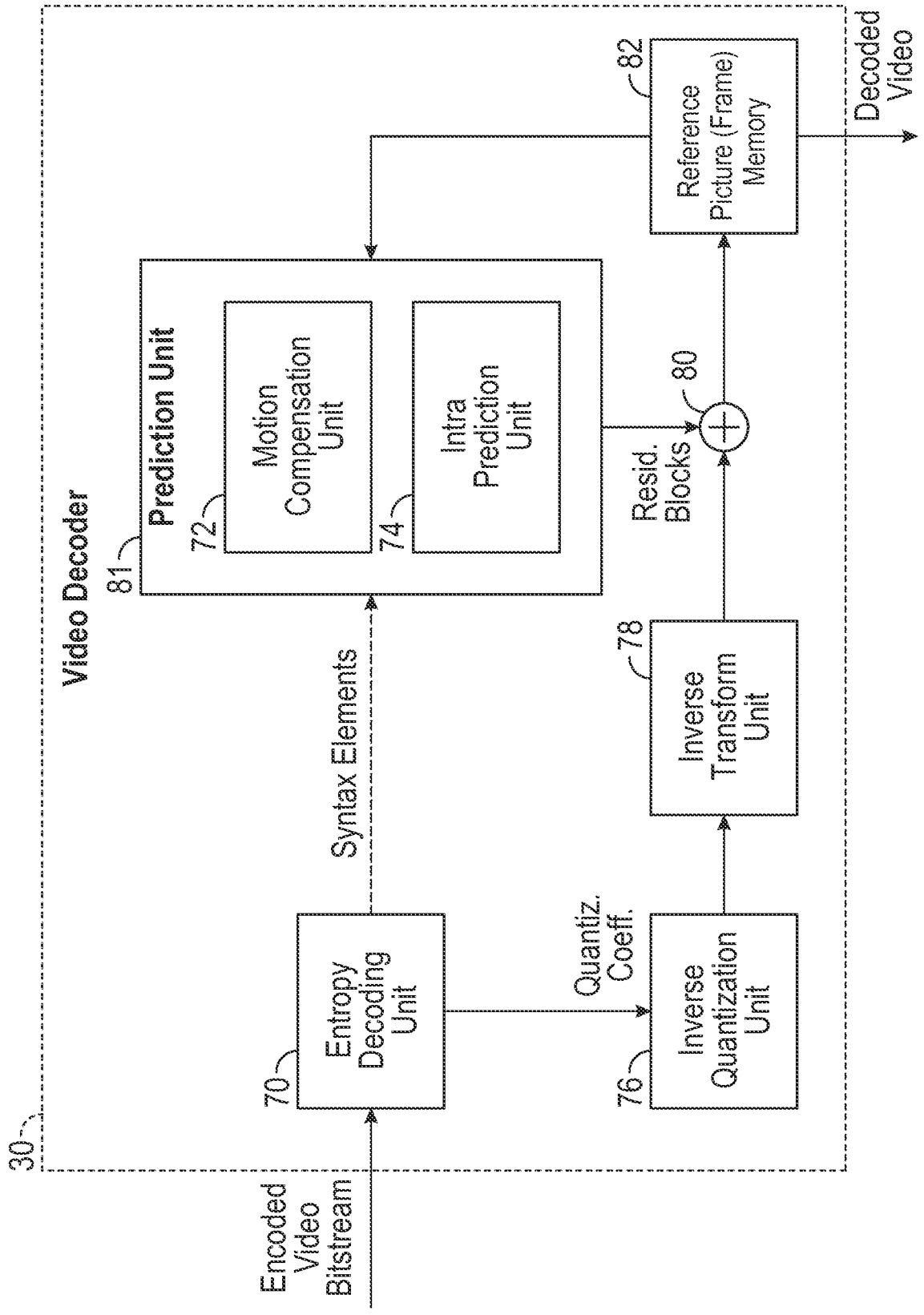
Video Decoder
30
Encoded Video Bitstream
Entropy Decoding Unit
70
Syntax Elements
Prediction Unit
81
Motion Compensation Unit
72
Intra Prediction Unit
74
Quantiz. Coeff.
Inverse Quantization Unit
76
Inverse Transform Unit
78
Resid. Blocks
80
Reference Picture (Frame) Memory
82
Decoded Video

FIG. 3

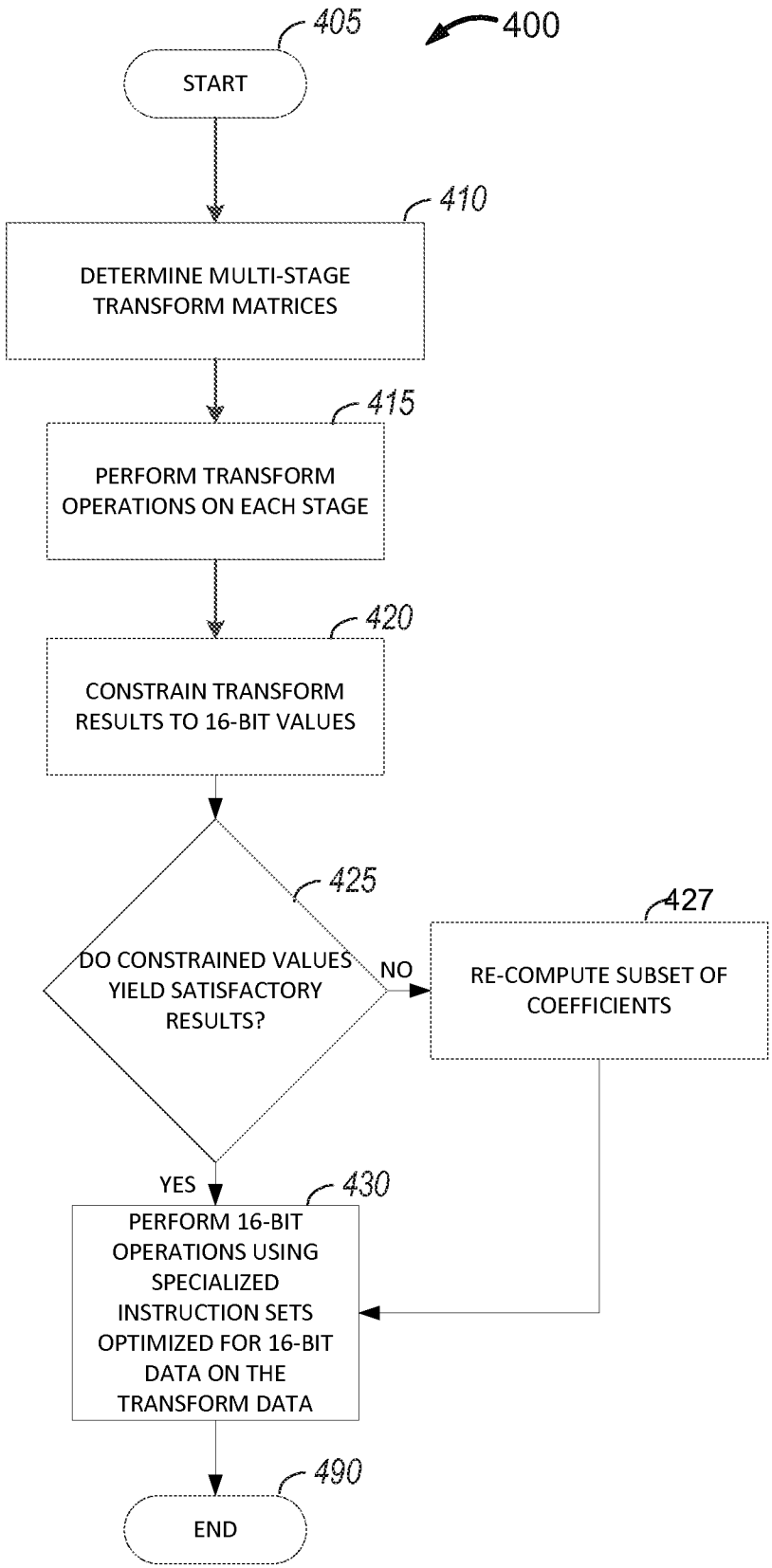
400
405
START
410
DETERMINE MULTI-STAGE TRANSFORM MATRICES
415
PERFORM TRANSFORM OPERATIONS ON EACH STAGE
420
CONSTRAIN TRANSFORM RESULTS TO 16-BIT VALUES
425
DO CONSTRAINED VALUES YIELD SATISFACTORY RESULTS?
NO
427
RE-COMPUTE SUBSET OF COEFFICIENTS
YES
430
PERFORM 16-BIT OPERATIONS USING SPECIALIZED INSTRUCTION SETS OPTIMIZED FOR 16-BIT DATA ON THE TRANSFORM DATA
490
END

FIG. 4

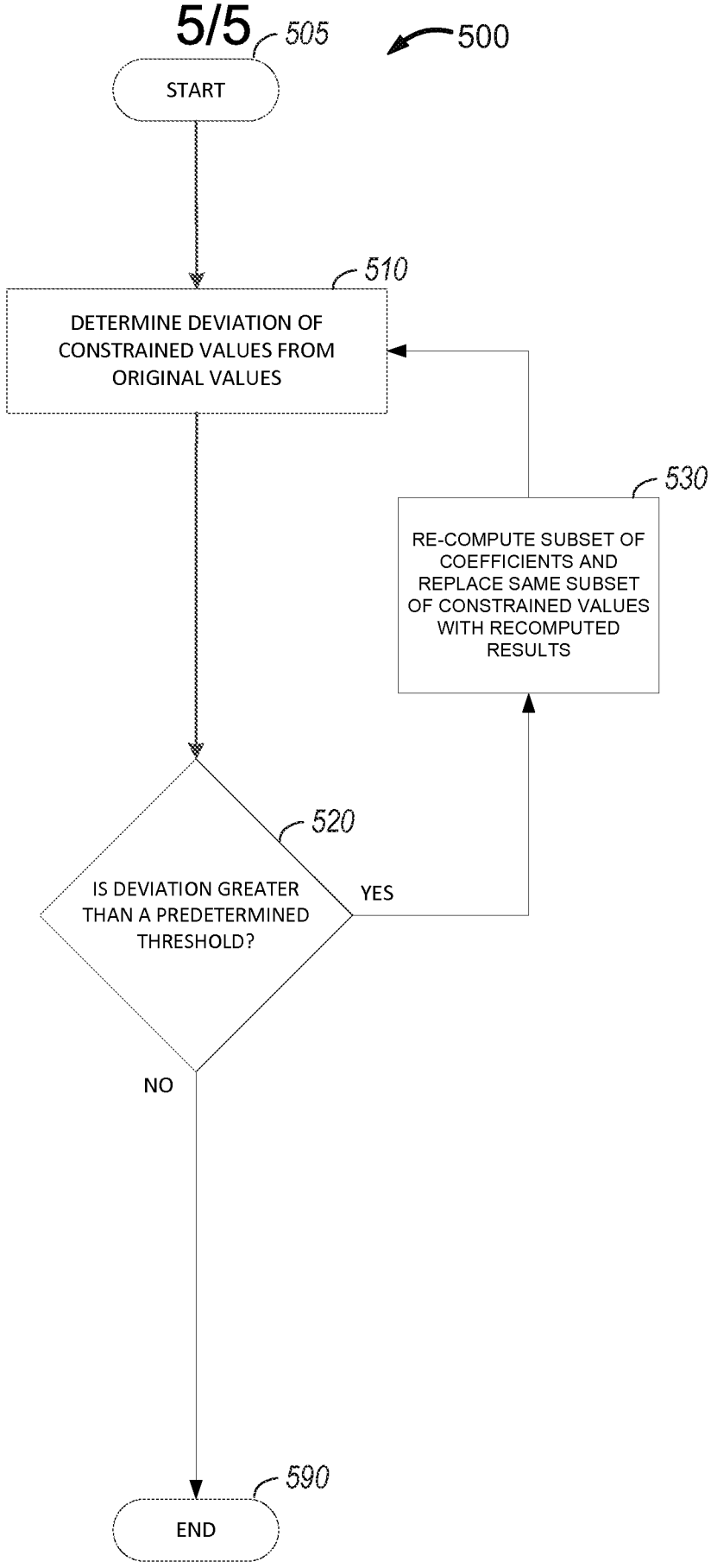
505
500
START
510
DETERMINE DEVIATION OF CONSTRAINED VALUES FROM ORIGINAL VALUES
530
RE-COMPUTE SUBSET OF COEFFICIENTS AND REPLACE SAME SUBSET OF CONSTRAINED VALUES WITH RECOMPUTED RESULTS
520
IS DEVIATION GREATER THAN A PREDETERMINED THRESHOLD?
YES
NO
590
END

FIG. 5