

REGULAR CODED BIN REDUCTION FOR COEFFICIENT CODING USING THRESHOLD

[0001] This Application claims priority to:

U.S. Patent Application No. 16/560,918, filed 4 September 2019, which claims the benefit of:

U.S. Provisional Patent Application 62/805,625, filed 14 February 2019;

U.S. Provisional Patent Application 62/823,537, filed 25 March 2019;

the entire content of each being hereby incorporated by reference.

TECHNICAL FIELD

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

BACKGROUND

[0003] Digital video capabilities can be incorporated into a wide range of devices, including digital televisions, digital direct broadcast systems, wireless broadcast systems, personal digital assistants (PDAs), laptop or desktop computers, tablet computers, e-book readers, digital cameras, digital recording devices, digital media players, video gaming devices, video game consoles, cellular or satellite radio telephones, so-called “smart phones,” video teleconferencing devices, video streaming devices, and the like. Digital video devices implement video coding techniques, such as those described in the standards defined by MPEG-2, MPEG-4, ITU-T H.263, ITU-T H.264/MPEG-4, Part 10, Advanced Video Coding (AVC), the High Efficiency Video Coding (HEVC) standard, ITU-T H.265/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.

[0004] Video coding techniques include spatial (intra-picture) prediction and/or temporal (inter-picture) prediction to reduce or remove redundancy inherent in video sequences. For block-based video coding, a video slice (e.g., a video picture or a portion of a video picture) may be partitioned into video blocks, which may also be referred to as coding tree units (CTUs), coding units (CUs) and/or coding nodes. Video blocks in an intra-coded (I) slice of a picture are encoded using spatial prediction with respect to reference samples in neighboring blocks in the same picture. Video blocks in

an inter-coded (P or B) slice of a picture may use spatial prediction with respect to reference samples in neighboring blocks in the same picture or temporal prediction with respect to reference samples in other reference pictures. Pictures may be referred to as frames, and reference pictures may be referred to as reference frames.

SUMMARY

[0005] Video coding (e.g., video encoding and/or video decoding) typically involves predicting a block of video data from either an already coded block of video data in the same picture (e.g., intra prediction) or an already coded block of video data in a different picture (e.g., inter prediction). In some instances, the video encoder also calculates residual data by comparing the predictive block to the original block. Thus, the residual data represents a difference between the predictive block and the original block of video data. To reduce the number of bits needed to signal the residual data, the video encoder transforms the residual data into transform coefficients, quantizes the transform coefficients, and signals the transformed and quantized coefficients in the encoded bitstream. The compression achieved by the transform and quantization processes may be lossy, meaning that transform and quantization processes may introduce distortion into the decoded video data. This disclosure describes techniques related to transform coefficient coding.

[0006] In one example, a method for decoding video data includes determining a size of a transform unit, wherein the transform unit comprises N coefficient groups and each coefficient group comprises M coefficients, wherein M and N are integer values; determining a regular bin count threshold for the transform unit based on the size of the transform unit; context decoding syntax elements for the transform unit until the first number of regular coded bins is reached, wherein: the context decoded syntax elements comprise one or more significance flags, one or more parity level flags, and one or more first flags, each of the one or more significance flags indicate if an absolute level for a corresponding coefficient is equal to zero, each of the one or more parity level flags indicates if a corresponding coefficient has an absolute level that is even or odd, each of the one or more first flags indicates if a corresponding coefficient has an absolute level that is greater than 2; in response to reaching the first number of regular coded bins, bypass decoding additional syntax elements of the transform unit; determining values for a first set of coefficients of the transform unit based on the context decoded syntax

elements; and determining values for a second set of coefficients of the transform unit based on the additional syntax elements.

[0007] In another example, a device for decoding video data includes a memory configured to store video data and one or more processors implemented in circuitry and configured to determine a size of a transform unit of the video data, wherein the transform unit comprises N coefficient groups and each coefficient group comprises M coefficients, wherein M and N are integer values; determine a regular bin count threshold for the transform unit based on the size of the transform unit; context decode syntax elements for the transform unit until the first number of regular coded bins is reached, wherein: the context decoded syntax elements comprise one or more significance flags, one or more parity level flags, and one or more first flags, each of the one or more significance flags indicate if an absolute level for a corresponding coefficient is equal to zero, each of the one or more parity level flags indicates if a corresponding coefficient has an absolute level that is even or odd, and each of the one or more first flags indicates if a corresponding coefficient has an absolute level that is greater than 2; in response to reaching the first number of regular coded bins, bypass decode additional syntax elements; determine values for a first set of coefficients of the transform unit based on the context decoded syntax elements; and determine values for a second set of coefficients of the transform unit based on the additional syntax elements.

[0008] In another example, a computer readable medium stores instructions that when executed by one or more processors cause the one or more processors to determine a size of a transform unit of the video data, wherein the transform unit comprises N coefficient groups and each coefficient group comprises M coefficients, wherein M and N are integer values; determine a regular bin count threshold for the transform unit based on the size of the transform unit; context decode syntax elements for the transform unit until the first number of regular coded bins is reached, wherein: the context decoded syntax elements comprise one or more significance flags, one or more parity level flags, and one or more first flags, each of the one or more significance flags indicate if an absolute level for a corresponding coefficient is equal to zero, each of the one or more parity level flags indicates if a corresponding coefficient has an absolute level that is even or odd, and each of the one or more first flags indicates if a corresponding coefficient has an absolute level that is greater than 2; in response to reaching the first number of regular coded bins, bypass decode additional syntax

elements; determine values for a first set of coefficients of the transform unit based on the context decoded syntax elements; and determine values for a second set of coefficients of the transform unit based on the additional syntax elements.

[0009] In another example, an apparatus comprises means for determining a size of a transform unit, wherein the transform unit comprises N coefficient groups and each coefficient group comprises M coefficients, wherein M and N are integer values; means for determining a regular bin count threshold for the transform unit based on the size of the transform unit; means for context decoding syntax elements for the transform unit until the first number of regular coded bins is reached, wherein: the context decoded syntax elements comprise one or more significance flags, one or more parity level flags, and one or more first flags, each of the one or more significance flags indicate if an absolute level for a corresponding coefficient is equal to zero, each of the one or more parity level flags indicates if a corresponding coefficient has an absolute level that is even or odd, each of the one or more first flags indicates if a corresponding coefficient has an absolute level that is greater than 2; means for bypass decoding additional syntax elements of the transform unit in response to reaching the first number of regular coded bins; means for determining values for a first set of coefficients of the transform unit based on the context decoded syntax elements; and means for determining values for a second set of coefficients of the transform unit based on the additional syntax elements.

[0010] In another example, a method for encoding video data includes determining a size of a transform unit, wherein the transform unit comprises N coefficient groups and each coefficient group comprises M coefficients, wherein M and N are integer values; determining a regular bin count threshold for the transform unit based on the size of the transform unit; context encoding syntax elements for the transform unit until the first number of regular coded bins is reached, wherein: the context encoded syntax elements comprises one or more significance flags, one or more parity level flags, and one or more first flags, each of the one or more significance flags indicate if an absolute level for a corresponding coefficient is equal to zero, each of the one or more parity level flags indicates if a corresponding coefficient has an absolute level that is even or odd, each of the one or more first flags indicates if a corresponding coefficient has an absolute level that is greater than 2, and the context encoded syntax elements represent values for a first set of coefficients of the transform unit; and in response to reaching the first number of regular coded bins, bypass encoding additional syntax elements, wherein

the additional syntax elements represent values for a second set of coefficients of the transform unit.

[0011] In another example, a device for encoding video data includes a memory configured to store video data and one or more processors implemented in circuitry and configured to determine a size of a transform unit, wherein the transform unit comprises N coefficient groups and each coefficient group comprises M coefficients, wherein M and N are integer values; determine a regular bin count threshold for the transform unit based on the size of the transform unit; context encode syntax elements for the transform unit until the first number of regular coded bins is reached, wherein: the context encoded syntax elements comprises one or more significance flags, one or more parity level flags, and one or more first flags, each of the one or more significance flags indicate if an absolute level for a corresponding coefficient is equal to zero, each of the one or more parity level flags indicates if a corresponding coefficient has an absolute level that is even or odd, each of the one or more first flags indicates if a corresponding coefficient has an absolute level that is greater than 2, and the context encoded syntax elements represent values for a first set of coefficients of the transform unit; and in response to reaching the first number of regular coded bins, bypass encode additional syntax elements of the transform unit, wherein the additional syntax elements represent values for a second set of coefficients of the transform unit.

[0012] In another example, a computer readable storage medium stores instructions that when executed by one or more processors cause the one or more processors to determine a size of a transform unit of the video data, wherein the transform unit comprises N coefficient groups and each coefficient group comprises M coefficients, wherein M and N are integer values; determine a regular bin count threshold for the transform unit based on the size of the transform unit; context decode syntax elements for the transform unit until the first number of regular coded bins is reached, wherein: the context decoded syntax elements comprise one or more significance flags, one or more parity level flags, and one or more first flags, each of the one or more significance flags indicate if an absolute level for a corresponding coefficient is equal to zero, each of the one or more parity level flags indicates if a corresponding coefficient has an absolute level that is even or odd, and each of the one or more first flags indicates if a corresponding coefficient has an absolute level that is greater than 2; in response to reaching the first number of regular coded bins, bypass decode additional syntax elements; determine values for a first set of coefficients of the transform unit based on

the context decoded syntax elements; and determine values for a second set of coefficients of the transform unit based on the additional syntax elements.

[0013] In another example, an apparatus comprises means for determining a size of a transform unit, wherein the transform unit comprises N coefficient groups and each coefficient group comprises M coefficients, wherein M and N are integer values; means for determining a regular bin count threshold for the transform unit based on the size of the transform unit; means for context encoding syntax elements for the transform unit until the first number of regular coded bins is reached, wherein: the context encoded syntax elements comprises one or more significance flags, one or more parity level flags, and one or more first flags, each of the one or more significance flags indicate if an absolute level for a corresponding coefficient is equal to zero, each of the one or more parity level flags indicates if a corresponding coefficient has an absolute level that is even or odd, each of the one or more first flags indicates if a corresponding coefficient has an absolute level that is greater than 2, and the context encoded syntax elements represent values for a first set of coefficients of the transform unit; and means for bypass encoding additional syntax elements in response to reaching the first number of regular coded bins, wherein the additional syntax elements represent values for a second set of coefficients of the transform unit.

[0014] The details of one or more examples are set forth in the accompanying drawings and the description below. Other features, objects, and advantages will be apparent from the description, drawings, and claims.

BRIEF DESCRIPTION OF DRAWINGS

[0015] FIG. 1 is a block diagram illustrating an example video encoding and decoding system that may perform the techniques of this disclosure.

[0016] FIGS. 2A and 2B are conceptual diagrams illustrating an example quadtree binary tree (QTBT) structure, and a corresponding coding tree unit (CTU).

[0017] FIG. 3 shows an example scan order for syntax elements.

[0018] FIG. 4 shows an illustration of an example template used for selecting probability models.

[0019] FIG. 5 shows an example scan order for syntax elements.

[0020] FIG. 6 shows an example illustrating how a Rice parameter may be used to derive a threshold for switching between bypass coded Golomb-Rice coding and Exponential-Golomb coding.

[0021] FIG. 7 is a block diagram illustrating an example video encoder that may perform the techniques of this disclosure.

[0022] FIG. 8 is a block diagram illustrating an example video decoder that may perform the techniques of this disclosure.

[0023] FIGS. 9A and 9B are conceptual diagrams illustrating a range update process in binary arithmetic coding.

[0024] FIG. 10 is a conceptual diagram illustrating an output process in binary arithmetic coding.

[0025] FIG. 11 is a block diagram illustrating a context adaptive binary arithmetic coding (CABAC) coder in a video encoder.

[0026] FIG. 12 is a block diagram illustrating a CABAC coder in a video decoder.

[0027] FIG. 13 is a flowchart illustrating an example operation of a video encoder.

[0028] FIG. 14 is a flowchart illustrating an example operation of a video decoder.

[0029] FIG. 15 is a flowchart illustrating an example operation of a video encoder.

[0030] FIG. 16 is a flowchart illustrating an example operation of a video decoder.

DETAILED DESCRIPTION

[0031] Video coding (e.g., video encoding and/or video decoding) typically involves predicting a block of video data from either an already coded block of video data in the same picture (e.g., intra prediction) or an already coded block of video data in a different picture (e.g., inter prediction). In some instances, the video encoder also calculates residual data by comparing the predictive block to the original block. Thus, the residual data represents a difference between the predictive block and the original block of video data. To reduce the number of bits needed to signal the residual data, the video encoder transforms and quantizes the residual data and signals the transformed and quantized residual data in the encoded bitstream. The compression achieved by the transform and quantization processes may be lossy, meaning that transform and quantization processes may introduce distortion into the decoded video data.

[0032] A video decoder decodes and adds the residual data to the predictive block to produce a reconstructed video block that matches the original video block more closely

than the predictive block alone. Due to the loss introduced by the transforming and quantizing of the residual data, the reconstructed block may have distortion or artifacts. One common type of artifact or distortion is referred to as blockiness, where the boundaries of the blocks used to code the video data are visible.

[0033] To further improve the quality of decoded video, a video decoder can perform one or more filtering operations on the reconstructed video blocks. Examples of these filtering operations include deblocking filtering, sample adaptive offset (SAO) filtering, and adaptive loop filtering (ALF). Parameters for these filtering operations may either be determined by a video encoder and explicitly signaled in the encoded video bitstream or may be implicitly determined by a video decoder without needing the parameters to be explicitly signaled in the encoded video bitstream.

[0034] As introduced above, a video encoder transforms residual data to produce transform coefficients. Those transform coefficients may additionally be quantized. In this disclosure, the term transform coefficient, or coefficient, may refer to either a quantized transform coefficient or an unquantized transform coefficient. This disclosure describes techniques for signaling the values of transform coefficients, e.g., quantized transform coefficients, from a video encoder to a video decoder. More specifically, this disclosure describes techniques related to an entropy decoding process that converts a binary representation of bits to a series of non-binary valued quantized transform coefficients. The corresponding entropy encoding process, which is generally the reverse process of entropy decoding, is also described in this disclosure.

[0035] This disclosure describes techniques related to transform coefficient coding. Aspects of transform coefficient coding are described in JVET-K0072, H. Schwarz, T. Nguyen, D. Marpe, T. Wiegand, “Non-CE7: Alternative entropy coding for dependent quantization,” JVET document JVET-K0072, Ljubljana, Slovenia, July 2018. The techniques described herein may improve upon the efficiency in encoding and decoding quantized coefficients when compared to the techniques of JVET-K0072.

[0036] This disclosure describes techniques for coding coefficients associated with residual data in a video coding process. In one example, this disclosure describes techniques for determining which syntax elements are context coded and which syntax elements are bypass coded. In another example, this disclosure describes techniques for determining a Rice parameter used to define codes, e.g., Golomb-Rice codes or Exponential-Golomb codes, for coding remaining absolute values of coefficient levels for a block of coefficients where context adaptive binary arithmetic coding (CABAC) is

used to code other indications of significant coefficients, such as coefficient levels greater than 1 and coefficient levels greater than 2. The coefficient levels may be levels of transform coefficients, in the case of lossy coding, or levels of coefficients for which a transform is not applied (i.e., residual pixel values), in the case of lossless coding or lossy coding in transform skip mode.

[0037] The Rice parameter is a tunable value used to select a codeword set from the family of Golomb codes, e.g., Golomb-Rice codes or Exponential-Golomb codes. The codes defined by the Rice parameter may be used to code a remaining absolute value of a coefficient level for at least one coefficient in a transform unit (TU) or a coefficient group (CG), i.e., a block of coefficients. Each of the CGs may be a 4×4 transform block or a 4×4 subblock of a transform block of video data. The CGs may include transform coefficients, in the case of lossy coding, or coefficients for which a transform is not applied, in the case of lossless coding or lossy coding in transform skip mode.

[0038] The techniques of this disclosure may be applied to any of the existing video codecs, such as High Efficiency Video Coding (HEVC), or may be proposed as a promising coding tool for new video coding standards, such as the Versatile Video Coding (VVC) currently being develop or for other future video coding standards.

[0039] FIG. 1 is a block diagram illustrating an example video encoding and decoding system 100 that may perform the techniques of this disclosure. The techniques of this disclosure are generally directed to coding (encoding and/or decoding) video data. In general, video data includes any data for processing a video. Thus, video data may include raw, unencoded video, encoded video, decoded (e.g., reconstructed) video, and video metadata, such as signaling data.

[0040] As shown in FIG. 1, system 100 includes a source device 102 that provides encoded video data to be decoded and displayed by a destination device 116, in this example. In particular, source device 102 provides the video data to destination device 116 via a computer-readable medium 110. Source device 102 and destination device 116 may comprise any of a wide range of devices, including desktop computers, notebook (i.e., laptop) computers, tablet computers, set-top boxes, telephone handsets such as smartphones, televisions, cameras, display devices, digital media players, video gaming consoles, video streaming device, or the like. In some cases, source device 102 and destination device 116 may be equipped for wireless communication, and thus may be referred to as wireless communication devices.

[0041] In the example of FIG. 1, source device 102 includes video source 104, memory 106, video encoder 200, and output interface 108. Destination device 116 includes input interface 122, video decoder 300, memory 120, and display device 118. In accordance with this disclosure, video encoder 200 of source device 102 and video decoder 300 of destination device 116 may be configured to apply the techniques for coefficient coding described herein. Thus, source device 102 represents an example of a video encoding device, while destination device 116 represents an example of a video decoding device. In other examples, a source device and a destination device may include other components or arrangements. For example, source device 102 may receive video data from an external video source, such as an external camera. Likewise, destination device 116 may interface with an external display device, rather than include an integrated display device.

[0042] System 100 as shown in FIG. 1 is merely one example. In general, any digital video encoding and/or decoding device may perform the techniques for coefficient coding described herein. Source device 102 and destination device 116 are merely examples of such coding devices in which source device 102 generates coded video data for transmission to destination device 116. This disclosure refers to a “coding” device as a device that performs coding (encoding and/or decoding) of data. Thus, video encoder 200 and video decoder 300 represent examples of coding devices, in particular, a video encoder and a video decoder, respectively. In some examples, source device 102 and destination device 116 may operate in a substantially symmetrical manner such that each of source device 102 and destination device 116 includes video encoding and decoding components. Hence, system 100 may support one-way or two-way video transmission between source device 102 and destination device 116, e.g., for video streaming, video playback, video broadcasting, or video telephony.

[0043] In general, video source 104 represents a source of video data (i.e., raw, unencoded video data) and provides a sequential series of pictures (also referred to as “frames”) of the video data to video encoder 200, which encodes data for the pictures. Video source 104 of source device 102 may include a video capture device, such as a video camera, a video archive containing previously captured raw video, and/or a video feed interface to receive video from a video content provider. As a further alternative, video source 104 may generate computer graphics-based data as the source video, or a combination of live video, archived video, and computer-generated video. In each case, video encoder 200 encodes the captured, pre-captured, or computer-generated video

data. Video encoder 200 may rearrange the pictures from the received order (sometimes referred to as “display order”) into a coding order for coding. Video encoder 200 may generate a bitstream including encoded video data. Source device 102 may then output the encoded video data via output interface 108 onto computer-readable medium 110 for reception and/or retrieval by, e.g., input interface 122 of destination device 116.

[0044] Memory 106 of source device 102 and memory 120 of destination device 116 represent general purpose memories. In some examples, memories 106, 120 may store raw video data, e.g., raw video from video source 104 and raw, decoded video data from video decoder 300. Additionally or alternatively, memories 106, 120 may store software instructions executable by, e.g., video encoder 200 and video decoder 300, respectively. Although memory 106 and memory 120 are shown separately from video encoder 200 and video decoder 300 in this example, it should be understood that video encoder 200 and video decoder 300 may also include internal memories for functionally similar or equivalent purposes. Furthermore, memories 106, 120 may store encoded video data, e.g., output from video encoder 200 and input to video decoder 300. In some examples, portions of memories 106, 120 may be allocated as one or more video buffers, e.g., to store raw, decoded, and/or encoded video data.

[0045] Computer-readable medium 110 may represent any type of medium or device capable of transporting the encoded video data from source device 102 to destination device 116. In one example, computer-readable medium 110 represents a communication medium to enable source device 102 to transmit encoded video data directly to destination device 116 in real-time, e.g., via a radio frequency network or computer-based network. Output interface 108 may modulate a transmission signal including the encoded video data, and input interface 122 may demodulate the received transmission signal, according to a communication standard, such as a wireless communication protocol. The communication medium may comprise any wireless or wired communication medium, such as a radio frequency (RF) spectrum or one or more physical transmission lines. The communication medium may form part of a packet-based network, such as a local area network, a wide-area network, or a global network such as the Internet. The communication medium may include routers, switches, base stations, or any other equipment that may be useful to facilitate communication from source device 102 to destination device 116.

[0046] In some examples, source device 102 may output encoded data from output interface 108 to storage device 112. Similarly, destination device 116 may access

encoded data from storage device 112 via input interface 122. Storage device 112 may include any of a variety of distributed or locally accessed data storage media such as a hard drive, Blu-ray discs, DVDs, CD-ROMs, flash memory, volatile or non-volatile memory, or any other suitable digital storage media for storing encoded video data.

[0047] In some examples, source device 102 may output encoded video data to file server 114 or another intermediate storage device that may store the encoded video generated by source device 102. Destination device 116 may access stored video data from file server 114 via streaming or download. File server 114 may be any type of server device capable of storing encoded video data and transmitting that encoded video data to the destination device 116. File server 114 may represent a web server (e.g., for a website), a File Transfer Protocol (FTP) server, a content delivery network device, or a network attached storage (NAS) device. Destination device 116 may access encoded video data from file server 114 through any standard data connection, including an Internet connection. This may include a wireless channel (e.g., a Wi-Fi connection), a wired connection (e.g., digital subscriber line (DSL), cable modem, etc.), or a combination of both that is suitable for accessing encoded video data stored on file server 114. File server 114 and input interface 122 may be configured to operate according to a streaming transmission protocol, a download transmission protocol, or a combination thereof.

[0048] Output interface 108 and input interface 122 may represent wireless transmitters/receivers, modems, wired networking components (e.g., Ethernet cards), wireless communication components that operate according to any of a variety of IEEE 802.11 standards, or other physical components. In examples where output interface 108 and input interface 122 comprise wireless components, output interface 108 and input interface 122 may be configured to transfer data, such as encoded video data, according to a cellular communication standard, such as 4G, 4G-LTE (Long-Term Evolution), LTE Advanced, 5G, or the like. In some examples where output interface 108 comprises a wireless transmitter, output interface 108 and input interface 122 may be configured to transfer data, such as encoded video data, according to other wireless standards, such as an IEEE 802.11 specification, an IEEE 802.15 specification (e.g., ZigBee™), a Bluetooth™ standard, or the like. In some examples, source device 102 and/or destination device 116 may include respective system-on-a-chip (SoC) devices. For example, source device 102 may include an SoC device to perform the functionality attributed to video encoder 200 and/or output interface 108, and destination device 116

may include an SoC device to perform the functionality attributed to video decoder 300 and/or input interface 122.

[0049] The techniques of this disclosure may be applied to video coding in support of any of a variety of multimedia applications, such as over-the-air television broadcasts, cable television transmissions, satellite television transmissions, Internet streaming video transmissions, such as dynamic adaptive streaming over HTTP (DASH), digital video that is encoded onto a data storage medium, decoding of digital video stored on a data storage medium, or other applications.

[0050] Input interface 122 of destination device 116 receives an encoded video bitstream from computer-readable medium 110 (e.g., a communication medium, storage device 112, file server 114, or the like). The encoded video bitstream may include signaling information defined by video encoder 200, which is also used by video decoder 300, such as syntax elements having values that describe characteristics and/or processing of video blocks or other coded units (e.g., slices, pictures, groups of pictures, sequences, or the like). Display device 118 displays decoded pictures of the decoded video data to a user. Display device 118 may represent any of a variety of display devices such as a 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.

[0051] Although not shown in FIG. 1, in some examples, video encoder 200 and video decoder 300 may each be integrated with an audio encoder and/or audio decoder, and may include appropriate MUX-DEMUX units, or other hardware and/or software, to handle multiplexed streams including both audio and video in a common data stream. If applicable, MUX-DEMUX units may conform to the ITU H.223 multiplexer protocol, or other protocols such as the user datagram protocol (UDP).

[0052] Video encoder 200 and video decoder 300 each may be implemented as any of a variety of suitable encoder and/or decoder circuitry, such as one or more microprocessors, digital signal processors (DSPs), application specific integrated circuits (ASICs), field programmable gate arrays (FPGAs), discrete logic, software, hardware, firmware or any combinations thereof. When the techniques are implemented partially in software, a device may store instructions for the software in a suitable, non-transitory computer-readable medium and execute the instructions in hardware using one or more processors to perform the techniques of this disclosure. Each of video encoder 200 and video decoder 300 may be included in one or more encoders or

decoders, either of which may be integrated as part of a combined encoder/decoder (CODEC) in a respective device. A device including video encoder 200 and/or video decoder 300 may comprise an integrated circuit, a microprocessor, and/or a wireless communication device, such as a cellular telephone.

[0053] Video encoder 200 and video decoder 300 may operate according to a video coding standard, such as ITU-T H.265, also referred to as High Efficiency Video Coding (HEVC) or extensions thereto, such as the multi-view and/or scalable video coding extensions. Alternatively, video encoder 200 and video decoder 300 may operate according to other proprietary or industry standards, such as the Joint Exploration Test Model (JEM) or ITU-T H.266, also referred to as Versatile Video Coding (VVC). A recent draft of the VVC standard is described in Bross, et al.

“Versatile Video Coding (Draft 6),” Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 15th Meeting: Gothenburg, SE, 3–12 July 2019, JVET-O2001-vE (hereinafter “VVC Draft 6”). The techniques of this disclosure, however, are not limited to any particular coding standard.

[0054] In general, video encoder 200 and video decoder 300 may perform block-based coding of pictures. The term “block” generally refers to a structure including data to be processed (e.g., encoded, decoded, or otherwise used in the encoding and/or decoding process). For example, a block may include a two-dimensional matrix of samples of luminance and/or chrominance data. In general, video encoder 200 and video decoder 300 may code video data represented in a YUV (e.g., Y, Cb, Cr) format. That is, rather than coding red, green, and blue (RGB) data for samples of a picture, video encoder 200 and video decoder 300 may code luminance and chrominance components, where the chrominance components may include both red hue and blue hue chrominance components. In some examples, video encoder 200 converts received RGB formatted data to a YUV representation prior to encoding, and video decoder 300 converts the YUV representation to the RGB format. Alternatively, pre- and post-processing units (not shown) may perform these conversions.

[0055] This disclosure may generally refer to coding (e.g., encoding and decoding) of pictures to include the process of encoding or decoding data of the picture. Similarly, this disclosure may refer to coding of blocks of a picture to include the process of encoding or decoding data for the blocks, e.g., prediction and/or residual coding. An encoded video bitstream generally includes a series of values for syntax elements representative of coding decisions (e.g., coding modes) and partitioning of pictures into

blocks. Thus, references to coding a picture or a block should generally be understood as coding values for syntax elements forming the picture or block.

[0056] HEVC defines various blocks, including coding units (CUs), prediction units (PUs), and transform units (TUs). According to HEVC, a video coder (such as video encoder 200) partitions a coding tree unit (CTU) into CUs according to a quadtree structure. That is, the video coder partitions CTUs and CUs into four equal, non-overlapping squares, and each node of the quadtree has either zero or four child nodes. Nodes without child nodes may be referred to as “leaf nodes,” and CUs of such leaf nodes may include one or more PUs and/or one or more TUs. The video coder may further partition PUs and TUs. For example, in HEVC, a residual quadtree (RQT) represents partitioning of TUs. In HEVC, PUs represent inter-prediction data, while TUs represent residual data. CUs that are intra-predicted include intra-prediction information, such as an intra-mode indication.

[0057] As another example, video encoder 200 and video decoder 300 may be configured to operate according to JEM or VVC. According to JEM or VVC, a video coder (such as video encoder 200) partitions a picture into a plurality of coding tree units (CTUs). Video encoder 200 may partition a CTU according to a tree structure, such as a quadtree-binary tree (QTBT) structure or Multi-Type Tree (MTT) structure. The QTBT structure removes the concepts of multiple partition types, such as the separation between CUs, PUs, and TUs of HEVC. A QTBT structure includes two levels: a first level partitioned according to quadtree partitioning, and a second level partitioned according to binary tree partitioning. A root node of the QTBT structure corresponds to a CTU. Leaf nodes of the binary trees correspond to coding units (CUs).

[0058] In an MTT partitioning structure, blocks may be partitioned using a quadtree (QT) partition, a binary tree (BT) partition, and one or more types of triple tree (TT) (also called ternary tree (TT)) partitions. A triple or ternary tree partition is a partition where a block is split into three sub-blocks. In some examples, a triple or ternary tree partition divides a block into three sub-blocks without dividing the original block through the center. The partitioning types in MTT (e.g., QT, BT, and TT), may be symmetrical or asymmetrical.

[0059] In some examples, video encoder 200 and video decoder 300 may use a single QTBT or MTT structure to represent each of the luminance and chrominance components, while in other examples, video encoder 200 and video decoder 300 may use two or more QTBT or MTT structures, such as one QTBT/MTT structure for the

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

[0060] Video encoder 200 and video decoder 300 may be configured to use quadtree partitioning per HEVC, QTBT partitioning, MTT partitioning, or other partitioning structures. For purposes of explanation, the description of the techniques of this disclosure is presented with respect to QTBT partitioning. However, it should be understood that the techniques of this disclosure may also be applied to video coders configured to use quadtree partitioning, or other types of partitioning as well.

[0061] The blocks (e.g., CTUs or CUs) may be grouped in various ways in a picture. As one example, a brick may refer to a rectangular region of CTU rows within a particular tile in a picture. A tile may be a rectangular region of CTUs within a particular tile column and a particular tile row in a picture. A tile column refers to a rectangular region of CTUs having a height equal to the height of the picture and a width specified by syntax elements (e.g., such as in a picture parameter set). A tile row refers to a rectangular region of CTUs having a height specified by syntax elements (e.g., such as in a picture parameter set) and a width equal to the width of the picture.

[0062] In some examples, a tile may be partitioned into multiple bricks, each of which may include one or more CTU rows within the tile. A tile that is not partitioned into multiple bricks may also be referred to as a brick. However, a brick that is a true subset of a tile may not be referred to as a tile.

[0063] The bricks in a picture may also be arranged in a slice. A slice may be an integer number of bricks of a picture that may be exclusively contained in a single network abstraction layer (NAL) unit. In some examples, a slice includes either a number of complete tiles or only a consecutive sequence of complete bricks of one tile.

[0064] This disclosure may use “NxN” and “N by N” interchangeably to refer to the sample dimensions of a block (such as a CU or other video block) in terms of vertical and horizontal dimensions, e.g., 16x16 samples or 16 by 16 samples. In general, a 16x16 CU will have 16 samples in a vertical direction ($y = 16$) and 16 samples in a horizontal direction ($x = 16$). Likewise, an NxN CU generally has N samples in a vertical direction and N samples in a horizontal direction, where N represents a nonnegative integer value. The samples in a CU may be arranged in rows and columns. Moreover, CUs need not necessarily have the same number of samples in the horizontal direction as in the vertical direction. For example, CUs may comprise NxM samples, where M is not necessarily equal to N.

[0065] Video encoder 200 encodes video data for CUs representing prediction and/or residual information, and other information. The prediction information indicates how the CU is to be predicted in order to form a prediction block for the CU. The residual information generally represents sample-by-sample differences between samples of the CU prior to encoding and the prediction block.

[0066] To predict a CU, video encoder 200 may generally form a prediction block for the CU through inter-prediction or intra-prediction. Inter-prediction generally refers to predicting the CU from data of a previously coded picture, whereas intra-prediction generally refers to predicting the CU from previously coded data of the same picture. To perform inter-prediction, video encoder 200 may generate the prediction block using one or more motion vectors. Video encoder 200 may generally perform a motion search to identify a reference block that closely matches the CU, e.g., in terms of differences between the CU and the reference block. Video encoder 200 may calculate a difference metric using a sum of absolute difference (SAD), sum of squared differences (SSD), mean absolute difference (MAD), mean squared differences (MSD), or other such difference calculations to determine whether a reference block closely matches the current CU. In some examples, video encoder 200 may predict the current CU using uni-directional prediction or bi-directional prediction.

[0067] Some examples of JEM and VVC also provide an affine motion compensation mode, which may be considered an inter-prediction mode. In affine motion compensation mode, video encoder 200 may determine two or more motion vectors that represent non-translational motion, such as zoom in or out, rotation, perspective motion, or other irregular motion types.

[0068] To perform intra-prediction, video encoder 200 may select an intra-prediction mode to generate the prediction block. Some examples of JEM and VVC provide sixty-seven intra-prediction modes, including various directional modes, as well as planar mode and DC mode. In general, video encoder 200 selects an intra-prediction mode that describes neighboring samples to a current block (e.g., a block of a CU) from which to predict samples of the current block. Such samples may generally be above, above and to the left, or to the left of the current block in the same picture as the current block, assuming video encoder 200 codes CTUs and CUs in raster scan order (left to right, top to bottom).

[0069] Video encoder 200 encodes data representing the prediction mode for a current block. For example, for inter-prediction modes, video encoder 200 may encode data

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

[0070] Following prediction, such as intra-prediction or inter-prediction of a block, video encoder 200 may calculate residual data for the block. The residual data, such as a residual block, represents sample by sample differences between the block and a prediction block for the block, formed using the corresponding prediction mode. Video encoder 200 may apply one or more transforms to the residual block, to produce transformed data in a transform domain instead of the sample domain. For example, video encoder 200 may apply a discrete cosine transform (DCT), an integer transform, a wavelet transform, or a conceptually similar transform to residual video data. Additionally, video encoder 200 may apply a secondary transform following the first transform, such as a mode-dependent non-separable secondary transform (MDNSST), a signal dependent transform, a Karhunen-Loeve transform (KLT), or the like. Video encoder 200 produces transform coefficients following application of the one or more transforms.

[0071] As noted above, following any transforms to produce transform coefficients, video encoder 200 may perform quantization of the transform coefficients.

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

[0072] Following quantization, video encoder 200 may scan the transform coefficients, producing a one-dimensional vector from the two-dimensional matrix including the quantized transform coefficients. The scan may be designed to place higher energy (and therefore lower frequency) transform coefficients at the front of the vector and to place lower energy (and therefore higher frequency) transform coefficients at the back of the vector. In some examples, video encoder 200 may utilize a predefined scan order to

scan the quantized transform coefficients to produce a serialized vector, and then entropy encode the quantized transform coefficients of the vector. In other examples, video encoder 200 may perform an adaptive scan. After scanning the quantized transform coefficients to form the one-dimensional vector, video encoder 200 may entropy encode the one-dimensional vector, e.g., according to context-adaptive binary arithmetic coding (CABAC). Video encoder 200 may also entropy encode values for syntax elements describing metadata associated with the encoded video data for use by video decoder 300 in decoding the video data.

[0073] To perform CABAC, video encoder 200 may assign a context within a context model to a symbol to be transmitted. The context may relate to, for example, whether neighboring values of the symbol are zero-valued or not. The probability determination may be based on a context assigned to the symbol.

[0074] Video encoder 200 may further generate syntax data, such as block-based syntax data, picture-based syntax data, and sequence-based syntax data, to video decoder 300, e.g., in a picture header, a block header, a slice header, or other syntax data, such as a sequence parameter set (SPS), picture parameter set (PPS), or video parameter set (VPS). Video decoder 300 may likewise decode such syntax data to determine how to decode corresponding video data.

[0075] In this manner, video encoder 200 may generate a bitstream including encoded video data, e.g., syntax elements describing partitioning of a picture into blocks (e.g., CUs) and prediction and/or residual information for the blocks. Ultimately, video decoder 300 may receive the bitstream and decode the encoded video data.

[0076] In general, video decoder 300 performs a reciprocal process to that performed by video encoder 200 to decode the encoded video data of the bitstream. For example, video decoder 300 may decode values for syntax elements of the bitstream using CABAC in a manner substantially similar to, albeit reciprocal to, the CABAC encoding process of video encoder 200. The syntax elements may define partitioning information of a picture into CTUs, and partitioning of each CTU according to a corresponding partition structure, such as a QTBT structure, to define CUs of the CTU. The syntax elements may further define prediction and residual information for blocks (e.g., CUs) of video data.

[0077] The residual information may be represented by, for example, quantized transform coefficients. Video decoder 300 may inverse quantize and inverse transform the quantized transform coefficients of a block to reproduce a residual block for the

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

[0078] This disclosure may generally refer to “signaling” certain information, such as syntax elements. The term “signaling” may generally refer to the communication of values for syntax elements and/or other data used to decode encoded video data. That is, video encoder 200 may signal values for syntax elements in the bitstream. In general, signaling refers to generating a value in the bitstream. As noted above, source device 102 may transport the bitstream to destination device 116 substantially in real time, or not in real time, such as might occur when storing syntax elements to storage device 112 for later retrieval by destination device 116.

[0079] FIGS. 2A and 2B are conceptual diagram illustrating an example quadtree binary tree (QTBT) structure 130, and a corresponding coding tree unit (CTU) 132. The solid lines represent quadtree splitting, and dotted lines indicate binary tree splitting. In each split (i.e., non-leaf) node of the binary tree, one flag is signaled to indicate which splitting type (i.e., horizontal or vertical) is used, where 0 indicates horizontal splitting and 1 indicates vertical splitting in this example. For the quadtree splitting, there is no need to indicate the splitting type, since quadtree nodes split a block horizontally and vertically into 4 sub-blocks with equal size. Accordingly, video encoder 200 may encode, and video decoder 300 may decode, syntax elements (such as splitting information) for a region tree level of QTBT structure 130 (i.e., the solid lines) and syntax elements (such as splitting information) for a prediction tree level of QTBT structure 130 (i.e., the dashed lines). Video encoder 200 may encode, and video decoder 300 may decode, video data, such as prediction and transform data, for CUs represented by terminal leaf nodes of QTBT structure 130.

[0080] In general, CTU 132 of FIG. 2B may be associated with parameters defining sizes of blocks corresponding to nodes of QTBT structure 130 at the first and second levels. These parameters may include a CTU size (representing a size of CTU 132 in samples), a minimum quadtree size (MinQTSIZE, representing a minimum allowed quadtree leaf node size), a maximum binary tree size (MaxBTSIZE, representing a maximum allowed binary tree root node size), a maximum binary tree depth

(MaxBTDepth, representing a maximum allowed binary tree depth), and a minimum binary tree size (MinBTSize, representing the minimum allowed binary tree leaf node size).

[0081] The root node of a QTBT structure corresponding to a CTU may have four child nodes at the first level of the QTBT structure, each of which may be partitioned according to quadtree partitioning. That is, nodes of the first level are either leaf nodes (having no child nodes) or have four child nodes. The example of QTBT structure 130 represents such nodes as including the parent node and child nodes having solid lines for branches. If nodes of the first level are not larger than the maximum allowed binary tree root node size (MaxBTSize), then the nodes can be further partitioned by respective binary trees. The binary tree splitting of one node can be iterated until the nodes resulting from the split reach the minimum allowed binary tree leaf node size (MinBTSize) or the maximum allowed binary tree depth (MaxBTDepth). The example of QTBT structure 130 represents such nodes as having dashed lines for branches. The binary tree leaf node is referred to as a coding unit (CU), which is used for prediction (e.g., intra-picture or inter-picture prediction) and transform, without any further partitioning. As discussed above, CUs may also be referred to as “video blocks” or “blocks.”

[0082] In one example of the QTBT partitioning structure, the CTU size is set as 128x128 (luma samples and two corresponding 64x64 chroma samples), the MinQTSIZE is set as 16x16, the MaxBTSize is set as 64x64, the MinBTSize (for both width and height) is set as 4, and the MaxBTDepth is set as 4. The quadtree partitioning is applied to the CTU first to generate quad-tree leaf nodes. The quadtree leaf nodes may have a size from 16x16 (i.e., the MinQTSIZE) to 128x128 (i.e., the CTU size). If the leaf quadtree node is 128x128, it will not be further split by the binary tree, since the size exceeds the MaxBTSize (i.e., 64x64, in this example). Otherwise, the leaf quadtree node will be further partitioned by the binary tree. Therefore, the quadtree leaf node is also the root node for the binary tree and has the binary tree depth as 0. When the binary tree depth reaches MaxBTDepth (4, in this example), no further splitting is permitted. When the binary tree node has width equal to MinBTSize (4, in this example), it implies no further horizontal splitting is permitted. Similarly, a binary tree node having a height equal to MinBTSize implies no further vertical splitting is permitted for that binary tree node. As noted above, leaf nodes of the binary tree are

referred to as CUs and are further processed according to prediction and transform without further partitioning.

[0083] JVET-K0072 described techniques for Trellis coded quantization (TCQ). When implementing the TCQ proposal of JVET-K0072, a video coder uses two scalar quantizers switchably for quantization/dequantization. The video coder determines the scalar quantizer to be used on a current transform/quantized coefficient based on the parity (i.e., the least significant bit) of the quantized coefficient that precedes the current transform/quantized coefficient in the scanning order.

[0084] JVET-K0072 also described a coefficient coding scheme coupled with TCQ. In the JVET-K0072 proposal, a video coder performs the context selection for decoding a quantized coefficient based on the quantizer used. Specifically, the significance flag (SIG) of a coefficient indicating the coefficient is zero or non-zero has three sets of context models, and the set selected for a particular SIG depends on the quantizer used for the associated coefficient. Therefore, when starting to encode/decode a value for SIG of a current coefficient, a video coder determines the quantizer for the current coefficient, and thus the context set for that coefficient's SIG, based on the parity of the coefficient in the previous scanning position, which is a known value.

[0085] The video coder may determine the parity in the previous scanning position by, for example, fully reconstructing the previous absolute level (absLevel). This technique, however, potentially limits throughput. In JVET-K0071, an alternative, namely that the parity of each non-zero coefficient is explicitly transmitted in the bitstream and decoded before moving to the next scanning position, was proposed and adopted in an early draft of VVC. More details of JVET-K0071 are provided below.

[0086] A TU is divided into non-overlapped subblocks, called coding groups (CGs), of which the size is usually 4x4. The decoding process described here is for a 4x4 CG but may easily be extended to any other CG sizes.

[0087] Video encoder 200 and video decoder 300 may be configured to process syntax elements in bitstreams. In one example coding scheme, five syntax elements are used to represent a value for absLevel. In this scheme, the syntax element "sig_coeff_flag" is equal to 0 if absLevel is 0; otherwise, the flag is equal to 1. The syntax element "par_level_flag" is present in bitstream if sig_coeff_flag is equal to 1. The syntax element "par_level_flag" is equal to 0 if absLevel is an odd number and is equal to 1 if absLevel is an even number. The syntax element "rem_abs_gt1_flag" is present in bitstream if sig_coeff_flag is equal to 1. The syntax element "rem_abs_gt1_flag" is

equal to 1 if absLevel is greater than 2; otherwise, this flag is equal to 0. The syntax element “rem_abs_gt2_flag” is present in bitstream if rem_abs_gt1_flag is equal to 1. The syntax element “rem_abs_gt2_flag” is equal to 1 if absLevel is greater than 4; otherwise, the flag is equal to 0. The syntax element “abs_remainder” is present in the bitstream if rem_abs_gt2_flag is equal to 1.

[0088] Below and in FIGS. 3 and 5A-5C, sig_coeff_flag, par_level_flag, rem_abs_gt1_flag, rem_abs_gt2_flag, and abs_remainder are denoted as SIG, Par, Gt1, Gt2, and rem, respectively, for the simplicity of description. When not parsed from the bitstream, video decoder 300 may be configured to set any of the above syntax elements to a default value of 0. Given the values of these five syntax elements, video decoder 300 can calculate a value for absLevel as shown in equation (1).

$$\text{absLevel} = \text{SIG} + \text{Par} + (\text{Gt1} \ll 1) + (\text{Gt2} \ll 1) + (\text{rem} \ll 1) \quad (1)$$

All of the five syntax elements are parsed from bitstream in coding scenarios where absLevel is greater than 6.

[0089] FIG. 3 shows an example order for the five syntax elements for all the absLevels values in a CG. The order in FIG. 3 shows the order of the syntax elements representing absLevels in a CG as in JVET-K0072.

[0090] The positions in a CG are scanned in up to three passes. In the first pass (labeled Pass #1 in FIG. 3), the flags SIG, Par, and Gt1 are parsed for each of coefficients 0 through 15. If the SIG flag for a particular coefficient is zero, then that coefficient does not have an associated Par or Gt1 flag. Thus, in the first pass, only non-zero SIG flags are followed by corresponding Par and Gt1 flags. After the first pass, a partial absLevel, denoted as absLevel1, for each position can be determined as shown in (2).

$$\text{absLevel1} = \text{SIG} + \text{Par} + (\text{Gt1} \ll 1) \quad (2)$$

[0091] If there is at least one non-zero Gt1 in the first pass, then the second pass (Pass #2 in FIG. 3) is scanned. In the second pass, Gt2 flags for the coefficients with non-zero Gt1s are parsed. The bins in the first and second passes may all be regular coded, meaning the probability distribution of the bin is modeled by a properly selected context model as described in more detail below. If there is at least one non-zero Gt2 in the second pass, then a third pass (Pass #3 in FIG. 3) is scanned, and rem values of the coefficients with non-zero Gt2s are parsed. As a rem value is not binary, the bins of the binarized version of a rem value may all be bypass-coded, meaning the bins are assumed to be uniformly distributed and no context selection is needed.

[0092] A video coding device, such as source device 102 and video encoder 200 or destination device 116 and video decoder 300, can typically process bypass coded bins much faster than regular coded bins because bypass coded bins can more easily be processed in parallel due to bypass coded bins not having data dependencies related to context selection and because arithmetic coding for bypass bins is generally simpler as it only requires a right shift versus a table look up as required for regular coded bins. Thus, the overall throughput of a video coder can be improved by reducing the number of regular coded bins and using bypass coded bins instead.

[0093] There are at least two potential problems with the coefficient coding scheme proposed in JVET-K0072. A first potential problem is that the number of regular coded bins is increased. Regular coded bins are processed much slower than bypass coded bins due to context selection and interval subdivision calculation. In JVET-K0072, the number of regular coded bins in a CG is up to 80, compared with 25 in JEM or HEVC (16 for SIG, 8 for gt1, and 1 for gt2). In the average case, the total number of bins representing absLevels used in JVET-K0072 is 40% to 90% higher when compared to JEM, depending on different bitrates. Another potential problem is that regular coded bins with different sets of contexts are interleaved. When bins that use the same logic for context selection are grouped together, fewer speculative calculations are needed to process multiple bins per cycle. In JVET-K0072, SIG, Par, and Gt1, which use different sets of contexts, are interleaved per coefficient (as illustrated in FIG. 3).

[0094] Video encoder 200 and video decoder 300 may be configured to perform context modelling. The context modelling used in JVET-K0072 is also briefly introduced here, as it is modified by this disclosure. The term context modelling generally refers to the selection of probability models (a.k.a. contexts) for a bin-to-decode. In JVET-K0072, SIG, Par, Gt1, and Gt2 flags need context modelling, and the selections depend on the values of absLevel1s in a local neighbourhood, denoted as N . In some implementations, positions inside the template but outside the current TU may be excluded from N .

[0095] FIG. 4 shows an illustration of a template that can be used for selecting probability models. Block 150 represents a transform unit. Position X in block 150 represents a current scan position, and positions Y in block 150 represent the local neighbourhood used for determining a probability model for the current scan position.

[0096] For the current scan position (see the black square with the X in FIG. 4), video encoder 200 and video decoder 300 may determine the context indices for the SIG, Par, Gt1, and Gt2 flags, denoted as ctxIdxSig, ctxIdxPar, ctxIdxGt1, and ctxIdxGt2, based

on the Y positions. To determining ctxIdxSig, ctxIdxPar, ctxIdxGt1, and ctxIdxGt2, video encoder 200 and video decoder 300 first determine three variables, numSig, sumAbs1, and d. The term numSig represents the number of non-zero SIGs in N , which is expressed by (3) below.

$$\text{sumSig} = \sum_{i \in N} \text{SIG}(i) \quad (3)$$

[0097] The term sumAbs1 represents the sum of absLevel1 in N , which is expressed by (4) below.

$$\text{sumAbs1} = \sum_{i \in N} \text{absLevel1}(i) \quad (4)$$

[0098] The term d represents the diagonal measure of the current position inside TU, as expressed by (5) below,

$$d = x + y \quad (5)$$

where x and y represent the coordinates of the current position inside TU.

[0099] Given sumAbs1 and d, video encoder 200 and video decoder 300 can determine the context index for decoding SIG as follows:

- For luma, video encoder 200 and video decoder 300 determine ctxIdxSig as follows:

$$\text{ctxIdxSig} = 18 * \max(0, \text{state}-1) + \min(\text{sumAbs1}, 5) + (d < 2 ? 12 : (d < 5 ? 6 : 0)) \quad (6)$$

- For chroma, video encoder 200 and video decoder 300 determine ctxIdxSig as follows:

$$\text{ctxIdxSig} = 12 * \max(0, \text{state}-1) + \min(\text{sumAbs1}, 5) + (d < 2 ? 6 : 0) \quad (7)$$

[0100] In equations (6) and (7) above, the variable “state” represents the current state of the state machine as defined in JVET-K0072.

[0101] Given sumSig, sumAbs1, and d, video encoder 200 and video decoder 300 can determine the context index for decoding Par follows:

- If the current scan position is equal to the position of the last non-zero coefficient, ctxIdxPar is 0.
- Otherwise,
 - For luma, ctxIdxPar is determined by:

$$\text{ctxIdxPar} = 1 + \min(\text{sumAbs1} - \text{numSig}, 4) + (d == 0 ? 15 : (d < 3 ? 10 : (d < 10 ? 5 : 0))) \quad (8)$$

- For chroma, ctxIdxPar is determined by (9)

$$\text{ctxIdxPar} = 1 + \min(\text{sumAbs1} - \text{numSig}, 4) + (d == 0 ? 5 : 0) \quad (9)$$

ctxIdxGt1 and ctxIdxGt2 are set the value of ctxIdxPar.

[0102] Video encoder 200 and video decoder 300 may perform absLevel reconstruction in the same manner as described in JVET-K0072.

[0103] According to a technique of this disclosure, in order to reduce the number of regular coded bins, video encoder 200 and video decoder 300 may use a predetermined number (C1) of regular coded bins in a coefficient group to code the sig-par-gt1 pass (i.e. first pass) of coefficients in a coefficient group until C1 regular coded bin threshold is reached. The techniques of this disclosure will be described with respect to 4x4 CGs, but the techniques of this disclosure are generally applicable to CGs of any size.

[0104] In one example implementation, video encoder 200 and video decoder 300 are configured to code all bins of the last coefficient where the regular bin count threshold C1 is reached as regular bins. After the first pass, which may include only regular coded bins, video encoder 200 and video decoder 300 may code a predetermined number (C2) of gt2 flags (if available) using regular bins. C2 with value equal to 2 is one particular choice that gives good performance-complexity trade off; however, other values of C2 may also be used. After the second pass, all remaining levels of coefficients after the first and second pass may be coded using bypass bins. Specifically, video encoder 200 and video decoder 300 may code remaining levels using the Rice/Golomb coding combination described below following the coding scan order starting with the first coefficient with remaining level info as shown in FIG. 5. For coding of Gt2 flags, the last regular coded Gt2 flag cannot exceed the location of the last regular coded Gt1 flag in first pass, i.e. in FIG. 5, $j \leq K$, even C2 if C2 threshold is not reached.

[0105] FIG. 5 shows an example scan order for syntax elements. In the example of FIG. 5, in a first pass, video encoder 200 and video decoder 300 code SIG, Par, and Gt1 flags until a threshold number of regular coded bins is reached. In the example of FIG. 5, the threshold number of regular coded bins is represented as C1 and occurs at coefficient K. Once the threshold number of regular coded bins is reached, then video encoder 200 and video decoder 300 may code the remaining flags for coefficient K, such as Par and Gt1 if present for coefficient K, using regular coded bins and then the first pass is complete. In a second pass, video encoder 200 and video decoder 300 code Gt2 flags until a second threshold number of regular coded bins is reached. In the example of FIG. 5, the second threshold number of regular coded bins is represented as C2 and occurs at coefficient j. Depending on the particular coding scenario, coefficient

j may be any of coefficients 1 through N. After coding the Gt2 flag for coefficient j, the second pass is complete.

[0106] For the third pass, video encoder 200 and video decoder 300 may perform remaining level coding for those coefficients that have a remaining level to code. For coefficients K+1 through N, the remaining level represents the absolute level of the coefficient. For coefficients 0 through K, the remaining level represents the difference between an intermediate value and the absolute level. The intermediate level for a coefficient is a level determined based on the SIG, Par, and Gt1 flags, and for some coefficients, also based on the Gt2 flag. Video encoder 200 and video decoder 300 may code the remLevel part of the coefficients, whose prefix bins are coded as regular bins in the first and second passes, using the scheme described in JVET-K0072 where a Rice Parameter (ricePar) is determined following the following rule:

- If $\text{sumAbs} - \text{numSig}$ is less than 12, ricePar is set equal to 0;
- Otherwise, if $\text{sumAbs} - \text{numSig}$ is less than 25, ricePar is set equal to 1;
- Otherwise, ricePar is set equal to 2.

[0107] The ricePar is used to derive the threshold to switch between bypass coded Golomb-Rice code and the Exponential-Golomb code JVET-K0072 as illustrated in FIG. 6.

[0108] This disclosure describes techniques for coding remLevel values for coefficients that are not regular bin coded in the first pass, that is, the the remaining coefficients of a CG after reaching the C1 regular coded bin threshold. This disclosure describes techniques for a modified ricePar derivation that depends on the sum of absolute values of neighboring coefficients. The neighboring coefficients may, for example, be the same neighboring coefficients identified in FIG. 4 or may be different coefficients.

[0109] In some examples, video encoder 200 and video decoder 300 may perform a mapping of coefficient levels to codewords to be coded depending on a TCQ state (assumed to be 0 for non-TCQ cases) and the sum of absolute values in the neighborhood template. In addition to the sum of absolute value and the TCQ state dependency, dependencies on the coefficient position such as the diagonal position within a TU may also be used as parameters to define the threshold value (pos0) used for remapping of the coefficient values to codewords.

[0110] Video encoder 200 and video decoder 300 may be configured to derive a ricePar as follows. Video encoder 200 and video decoder 300 calculate a sum of absolute values (i.e., absolute coefficient values) of the coefficients in the context template

$$\text{sumAbsLevel} = \sum_{i \in N} \text{absLevel}(i)$$

[0111] Video encoder 200 and video decoder 300 limit the sumAbsLevel to a maximum of 31 for riceParameter look up from the below table. Video encoder 200 and video decoder 300 determine a value for r based on the lesser of sumAbsLevel or 31. The table below shows an array of 32 values at positions 0 to 31. The value of r is equal to the value at position sumAbsLevel or position 31 if sumAbsLevel is greater than 31.

$r = g_auGoRiceParsCoeff[\min(\text{sumAbsLevel}, 31)]$, where

`const uint32_t g_auiGoRiceParsCoeff[32] = {0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3};`

[0112] Video encoder 200 and video decoder 300 derive a pos0 value to be used for coding of the remaining level as follows:

$\text{pos0} = g_auigoRicePosCoeff0[\max(0, \text{state}-1)][\text{sumAbsLevel}]$, where

`const uint32_t g_auiGoRicePosCoeff0[3][32] = {`
`{0, 0, 0, 0, 0, 1, 2, 2, 2, 2, 3, 3, 4, 4, 4, 4, 4, 4, 5, 5, 6, 7, 7, 8, 8, 8, 8, 8, 8, 9, 9, 9},`
`{1, 1, 1, 1, 2, 3, 4, 4, 5, 6, 6, 7, 8, 8, 8, 9, 10, 11, 12, 12, 12, 12, 14, 15, 15, 15, 16, 16,`
`16, 16, 17, 17},`
`{1, 1, 2, 2, 2, 3, 4, 4, 5, 6, 6, 7, 8, 8, 9, 9, 10, 11, 12, 12, 12, 12, 14, 15, 15, 16, 16, 16,`
`16, 17, 17, 17}`
`};`

[0113] Video encoder 200 and video decoder 300 computes the absLevel0 value to be coded using rice/Exp Golomb coding (with rice Parameter value r computed as described above) for a coefficient with absLevel value as follows:

- absLevel0 is defined as


```

      if (absLevel == 0)
      {
          absLevel0 = pos0;
      }
      else if (absLevel <= pos0)
      {
          absLevel0 = absLevel - 1;
      }
      else
      {

```

```

        absLevel0 = absLevel;
    }

```

[0114] Using this technique, video encoder 200 and video decoder 300 may group regular coded bins and bypass coded bins in a coefficient group which increases throughput for CABAC parsing.

[0115] According to another technique of this disclosure, the parity bit that drives the state machine may not change the state after a certain number of coefficients are coded in a coefficient group. In this case, there is no need to interleave the significance and the parity bins because the state does not need to change for coding efficiency. So, after a certain number of coefficients are decoded in a coefficient group, the coefficient coding scheme can take the form of sig-gt1-gt2-remLevel ordered coding where dependency on the state of TCQ is removed that can be moved to bypass coded bins (reducing regular coded bins) at finer granularity where sig, greater than 1 (gt1), greater than 2 (gt2) regular coded bins can be converted to bypass coded bins.

[0116] Video encoder 200 and video decoder 300 may be configured to implement TU-level regular bin counting. In VVC and HEVC, the number of regular bin count restriction is imposed currently at the coefficient group (CG) level. Coefficient groups are in units of 16 coefficients which are (4x4), 2x8, 8x2, 1x16, 16x1 or sometimes at 2x2 level (for chroma). Currently in VVC, 32 regular coded bins are allowed for 16 coefficient CGs. For 2x2 chroma CGs, 8 regular coded bins are allowed. Basically, it is limited at 2 regular bins per coefficient at CG level. This restriction addresses the worst-case limit for regular bin count for coefficient count by imposing the restrictions at CG level. The restriction can be relaxed by imposing a limit at TU level (TUs are composed of group of CGs for coefficient coding) which would typically be proportional to its size. For example, an 8x8 TU would have four 4x4 CGs. The regular coded bin count for TU can be set to total of 4x32=128 for luma component. Relaxing the CG level limit to TU level would increase the average number of regular coded bins.

[0117] Starting from last position, video encoder 200 and video decoder 300 may be configured to code a predetermined number of regular coded bins for sig, gt1, par, gt3 (abs_level greater than 3 flag) flags (i.e. 4 flags). Video encoder 200 and video decoder 300 may be configured to code these flags in groups of 4 (if present) until the regular coded bin count is reached. So, in this proposed scheme it would be possible to code 16*4=64 regular coded bins per CG (rather than 32 for current case in VVC) until target limit for total number of regular coded bins is reached. After reaching the regular coded

bin count, video encoder 200 and video decoder 300 may represent the rest of the coefficients fully by bypass coding as `dec_abs_level` described in VVC draft 4 and as also described above with respect to remaining level coding.

[0118] The limit for regular bin count may be set by expanding current limit at CG level. So, a TU with N 16- coefficient CGs may have total bit regular bin count limit of $N \times 32$, basically 2 regular bin per coefficient. This can be expanded to other limits, i.e. some regular bin count per coefficient target. In addition, depending on the last past coefficient position, the target can be set according to number of coefficients groups or number of coefficients covered during the scanning of coded coefficients. The target can be set by regular bin count per coefficient for the number of coefficients in the scan covering the coded coefficients.

[0119] The limit can also discount the not coded coefficient groups for setting the target limit or not considering the zero out regions of some types of TU s. Thus, if it is known that no coefficients are present in a CG in advance, then video encoder 200 and video decoder 300 may set a lower total TU level to reduce the average number regular coded bins.

[0120] With this new design, only the last coded coefficient group coding where regular bin count is reached would be similar to the current CG level coefficient coding where you would have mixture of regular coded bins (`sig`, `gt1`, `par`, `gt2`), and fully coded coefficient levels in a CG. There is always, `abs_remainder` [3] codes for all coefficients which have regular coded bin portions for representation of half of the remaining level.

[0121] The target regular bin per coefficient can be made to depend on TU size as well as number of CGs to be coded in a TU. So larger TUs with many coefficients may be assigned a lower regular bin per coefficient target to reduce the average regular bin count.

[0122] Video encoder 200 and video decoder 300 may limit the setting of the TU-level regular bin count for `sig_flag`, `gt1_flag`, `parity_flag`, `gt3_flag` (old `gt2_flag`) in a TU to 28 times the CG count for 16 coefficient CGs and (7 for each of the four coefficient CGs), or 28 for 16 coefficient (7 for each of the 4 coefficient CGs) CGs for Luma and 24 for 16 coefficient chroma and 6 for 4 coefficient chroma CGs.

[0123] In another examples, instead of being dependent on TU-size or number of coded CGs, the TU level limit may be set to a total number (e.g., total number of regularly coded bins) defined by non zeroed-out regions of transform units (currently large transforms dimension greater than 32 get zeroed out to $32 \times N$, $N \times 32$ region, $N \leq 32$)

Some transforms utilizing multiple transform sets (MTS) utilize zero out at the transform level (dimension greater than 16 get zeroed out to $16 \times N$, $N \times 16$ region, $N \leq 16$). Also, secondary transforms (NSST) may utilize a similar zero out scheme at a smaller limit. Accordingly, in some examples, the limit (e.g., limit of regularly coded bins) may be defined by the number of samples in the non-zero out regions times a factor (e.g., 2, 1.75 bins/per sample). A factor of 2 would correspond to 32 regular bins for a 16 coefficient CG. A factor of 1.75 would correspond to 28 bin per 16 coefficient CG. Different limits can be set for chroma TUs. This technique may be combined with discounting the bins allocated for the CGs that are not coded, so a total starting limit is set at TU level, and as the coefficients are decoded, if a skipped or not coded CG is encountered the limit is reduced by the regular bins allocated for the CG. In some examples, this may only be applied to skipped CGs (CGs that are zero from zero out property of some transforms).

[0124] FIG. 7 is a block diagram illustrating an example video encoder 200 that may perform the techniques of this disclosure. FIG. 7 is provided for purposes of explanation and should not be considered limiting of the techniques as broadly exemplified and described in this disclosure. For purposes of explanation, this disclosure describes video encoder 200 in the context of video coding standards such as the HEVC video coding standard and the H.266 video coding standard in development. However, the techniques of this disclosure are not limited to these video coding standards and are applicable generally to video encoding and decoding.

[0125] In the example of FIG. 7, video encoder 200 includes video data memory 230, mode selection unit 202, residual generation unit 204, transform processing unit 206, quantization unit 208, inverse quantization unit 210, inverse transform processing unit 212, reconstruction unit 214, filter unit 216, decoded picture buffer (DPB) 218, and entropy encoding unit 220.

[0126] Video data memory 230 may store video data to be encoded by the components of video encoder 200. Video encoder 200 may receive the video data stored in video data memory 230 from, for example, video source 104 (FIG. 1). DPB 218 may act as a reference picture memory that stores reference video data for use in prediction of subsequent video data by video encoder 200. Video data memory 230 and DPB 218 may be formed by any of a variety of memory devices, such as dynamic random access memory (DRAM), including synchronous DRAM (SDRAM), magnetoresistive RAM (MRAM), resistive RAM (RRAM), or other types of memory devices. Video data

memory 230 and DPB 218 may be provided by the same memory device or separate memory devices. In various examples, video data memory 230 may be on-chip with other components of video encoder 200, as illustrated, or off-chip relative to those components.

[0127] In this disclosure, reference to video data memory 230 should not be interpreted as being limited to memory internal to video encoder 200, unless specifically described as such, or memory external to video encoder 200, unless specifically described as such. Rather, reference to video data memory 230 should be understood as reference memory that stores video data that video encoder 200 receives for encoding (e.g., video data for a current block that is to be encoded). Memory 106 of FIG. 1 may also provide temporary storage of outputs from the various units of video encoder 200.

[0128] The various units of FIG. 7 are illustrated to assist with understanding the operations performed by video encoder 200. The units may be implemented as fixed-function circuits, programmable circuits, or a combination thereof. Fixed-function circuits refer to circuits that provide particular functionality, and are preset on the operations that can be performed. Programmable circuits refer to circuits that can be programmed to perform various tasks and provide flexible functionality in the operations that can be performed. For instance, programmable circuits may execute software or firmware that cause the programmable circuits to operate in the manner defined by instructions of the software or firmware. Fixed-function circuits may execute software instructions (e.g., to receive parameters or output parameters), but the types of operations that the fixed-function circuits perform are generally immutable. In some examples, the one or more of the units may be distinct circuit blocks (fixed-function or programmable), and in some examples, the one or more units may be integrated circuits.

[0129] Video encoder 200 may include arithmetic logic units (ALUs), elementary function units (EFUs), digital circuits, analog circuits, and/or programmable cores, formed from programmable circuits. In examples where the operations of video encoder 200 are performed using software executed by the programmable circuits, memory 106 (FIG. 1) may store the object code of the software that video encoder 200 receives and executes, or another memory within video encoder 200 (not shown) may store such instructions.

[0130] Video data memory 230 is configured to store received video data. Video encoder 200 may retrieve a picture of the video data from video data memory 230 and

provide the video data to residual generation unit 204 and mode selection unit 202.

Video data in video data memory 230 may be raw video data that is to be encoded.

[0131] Mode selection unit 202 includes a motion estimation unit 222, motion compensation unit 224, and an intra-prediction unit 226. Mode selection unit 202 may include additional functional units to perform video prediction in accordance with other prediction modes. As examples, mode selection unit 202 may include a palette unit, an intra-block copy unit (which may be part of motion estimation unit 222 and/or motion compensation unit 224), an affine unit, a linear model (LM) unit, or the like.

[0132] Mode selection unit 202 generally coordinates multiple encoding passes to test combinations of encoding parameters and resulting rate-distortion values for such combinations. The encoding parameters may include partitioning of CTUs into CUs, prediction modes for the CUs, transform types for residual data of the CUs, quantization parameters for residual data of the CUs, and so on. Mode selection unit 202 may ultimately select the combination of encoding parameters having rate-distortion values that are better than the other tested combinations.

[0133] Video encoder 200 may partition a picture retrieved from video data memory 230 into a series of CTUs, and encapsulate one or more CTUs within a slice. Mode selection unit 202 may partition a CTU of the picture in accordance with a tree structure, such as the QTBT structure or the quad-tree structure of HEVC described above. As described above, video encoder 200 may form one or more CUs from partitioning a CTU according to the tree structure. Such a CU may also be referred to generally as a “video block” or “block.”

[0134] In general, mode selection unit 202 also controls the components thereof (e.g., motion estimation unit 222, motion compensation unit 224, and intra-prediction unit 226) to generate a prediction block for a current block (e.g., a current CU, or in HEVC, the overlapping portion of a PU and a TU). For inter-prediction of a current block, motion estimation unit 222 may perform a motion search to identify one or more closely matching reference blocks in one or more reference pictures (e.g., one or more previously coded pictures stored in DPB 218). In particular, motion estimation unit 222 may calculate a value representative of how similar a potential reference block is to the current block, e.g., according to sum of absolute difference (SAD), sum of squared differences (SSD), mean absolute difference (MAD), mean squared differences (MSD), or the like. Motion estimation unit 222 may generally perform these calculations using sample-by-sample differences between the current block and the reference block being

considered. Motion estimation unit 222 may identify a reference block having a lowest value resulting from these calculations, indicating a reference block that most closely matches the current block.

[0135] Motion estimation unit 222 may form one or more motion vectors (MVs) that defines the positions of the reference blocks in the reference pictures relative to the position of the current block in a current picture. Motion estimation unit 222 may then provide the motion vectors to motion compensation unit 224. For example, for uni-directional inter-prediction, motion estimation unit 222 may provide a single motion vector, whereas for bi-directional inter-prediction, motion estimation unit 222 may provide two motion vectors. Motion compensation unit 224 may then generate a prediction block using the motion vectors. For example, motion compensation unit 224 may retrieve data of the reference block using the motion vector. As another example, if the motion vector has fractional sample precision, motion compensation unit 224 may interpolate values for the prediction block according to one or more interpolation filters. Moreover, for bi-directional inter-prediction, motion compensation unit 224 may retrieve data for two reference blocks identified by respective motion vectors and combine the retrieved data, e.g., through sample-by-sample averaging or weighted averaging.

[0136] As another example, for intra-prediction, or intra-prediction coding, intra-prediction unit 226 may generate the prediction block from samples neighboring the current block. For example, for directional modes, intra-prediction unit 226 may generally mathematically combine values of neighboring samples and populate these calculated values in the defined direction across the current block to produce the prediction block. As another example, for DC mode, intra-prediction unit 226 may calculate an average of the neighboring samples to the current block and generate the prediction block to include this resulting average for each sample of the prediction block.

[0137] Mode selection unit 202 provides the prediction block to residual generation unit 204. Residual generation unit 204 receives a raw, uncoded version of the current block from video data memory 230 and the prediction block from mode selection unit 202. Residual generation unit 204 calculates sample-by-sample differences between the current block and the prediction block. The resulting sample-by-sample differences define a residual block for the current block. In some examples, residual generation unit 204 may also determine differences between sample values in the residual block to

generate a residual block using residual differential pulse code modulation (RDPCM). In some examples, residual generation unit 204 may be formed using one or more subtractor circuits that perform binary subtraction.

[0138] In examples where mode selection unit 202 partitions CUs into PUs, each PU may be associated with a luma prediction unit and corresponding chroma prediction units. Video encoder 200 and video decoder 300 may support PUs having various sizes. As indicated above, the size of a CU may refer to the size of the luma coding block of the CU and the size of a PU may refer to the size of a luma prediction unit of the PU. Assuming that the size of a particular CU is $2N \times 2N$, video encoder 200 may support PU sizes of $2N \times 2N$ or $N \times N$ for intra prediction, and symmetric PU sizes of $2N \times 2N$, $2N \times N$, $N \times 2N$, $N \times N$, or similar for inter prediction. Video encoder 20 and video decoder 30 may also support asymmetric partitioning for PU sizes of $2N \times nU$, $2N \times nD$, $nL \times 2N$, and $nR \times 2N$ for inter prediction.

[0139] In examples where mode selection unit does not further partition a CU into PUs, each CU may be associated with a luma coding block and corresponding chroma coding blocks. As above, the size of a CU may refer to the size of the luma coding block of the CU. The video encoder 200 and video decoder 300 may support CU sizes of $2N \times 2N$, $2N \times N$, or $N \times 2N$.

[0140] For other video coding techniques such as an intra-block copy mode coding, an affine-mode coding, and linear model (LM) mode coding, as few examples, mode selection unit 202, via respective units associated with the coding techniques, generates a prediction block for the current block being encoded. In some examples, such as palette mode coding, mode selection unit 202 may not generate a prediction block, and instead generate syntax elements that indicate the manner in which to reconstruct the block based on a selected palette. In such modes, mode selection unit 202 may provide these syntax elements to entropy encoding unit 220 to be encoded.

[0141] As described above, residual generation unit 204 receives the video data for the current block and the corresponding prediction block. Residual generation unit 204 then generates a residual block for the current block. To generate the residual block, residual generation unit 204 calculates sample-by-sample differences between the prediction block and the current block. Thus,

[0142] Transform processing unit 206 applies one or more transforms to the residual block to generate a block of transform coefficients (referred to herein as a “transform coefficient block”). Transform processing unit 206 may apply various transforms to a

residual block to form the transform coefficient block. For example, transform processing unit 206 may apply a discrete cosine transform (DCT), a directional transform, a Karhunen-Loeve transform (KLT), or a conceptually similar transform to a residual block. In some examples, transform processing unit 206 may perform multiple transforms to a residual block, e.g., a primary transform and a secondary transform, such as a rotational transform. In some examples, transform processing unit 206 does not apply transforms to a residual block.

[0143] Quantization unit 208 may quantize the transform coefficients in a transform coefficient block, to produce a quantized transform coefficient block. Quantization unit 208 may quantize transform coefficients of a transform coefficient block according to a quantization parameter (QP) value associated with the current block. Video encoder 200 (e.g., via mode selection unit 202) may adjust the degree of quantization applied to the coefficient blocks associated with the current block by adjusting the QP value associated with the CU. Quantization may introduce loss of information, and thus, quantized transform coefficients may have lower precision than the original transform coefficients produced by transform processing unit 206.

[0144] Inverse quantization unit 210 and inverse transform processing unit 212 may apply inverse quantization and inverse transforms to a quantized transform coefficient block, respectively, to reconstruct a residual block from the transform coefficient block. Reconstruction unit 214 may produce a reconstructed block corresponding to the current block (albeit potentially with some degree of distortion) based on the reconstructed residual block and a prediction block generated by mode selection unit 202. For example, reconstruction unit 214 may add samples of the reconstructed residual block to corresponding samples from the prediction block generated by mode selection unit 202 to produce the reconstructed block.

[0145] Filter unit 216 may perform one or more filter operations on reconstructed blocks. For example, filter unit 216 may perform deblocking operations to reduce blockiness artifacts along edges of CUs. Operations of filter unit 216 may be skipped, in some examples.

[0146] Video encoder 200 stores reconstructed blocks in DPB 218. For instance, in examples where operations of filter unit 216 are not needed, reconstruction unit 214 may store reconstructed blocks to DPB 218. In examples where operations of filter unit 216 are needed, filter unit 216 may store the filtered reconstructed blocks to DPB 218. Motion estimation unit 222 and motion compensation unit 224 may retrieve a reference

picture from DPB 218, formed from the reconstructed (and potentially filtered) blocks, to inter-predict blocks of subsequently encoded pictures. In addition, intra-prediction unit 226 may use reconstructed blocks in DPB 218 of a current picture to intra-predict other blocks in the current picture.

[0147] In general, entropy encoding unit 220 may entropy encode syntax elements received from other functional components of video encoder 200, including the syntax elements described above for coefficient coding. For example, entropy encoding unit 220 may entropy encode quantized transform coefficient blocks from quantization unit 208. As another example, entropy encoding unit 220 may entropy encode prediction syntax elements (e.g., motion information for inter-prediction or intra-mode information for intra-prediction) from mode selection unit 202. Entropy encoding unit 220 may perform one or more entropy encoding operations on the syntax elements, which are another example of video data, to generate entropy-encoded data. For example, entropy encoding unit 220 may perform a context-adaptive variable length coding (CAVLC) operation, a CABAC operation, a variable-to-variable (V2V) length coding operation, a syntax-based context-adaptive binary arithmetic coding (SBAC) operation, a Probability Interval Partitioning Entropy (PIPE) coding operation, an Exponential-Golomb encoding operation, or another type of entropy encoding operation on the data. In some examples, entropy encoding unit 220 may operate in bypass mode where syntax elements are not entropy encoded.

[0148] Video encoder 200 may output a bitstream that includes the entropy encoded syntax elements needed to reconstruct blocks of a slice or picture. In particular, entropy encoding unit 220 may output the bitstream.

[0149] The operations described above are described with respect to a block. Such description should be understood as being operations for a luma coding block and/or chroma coding blocks. As described above, in some examples, the luma coding block and chroma coding blocks are luma and chroma components of a CU. In some examples, the luma coding block and the chroma coding blocks are luma and chroma components of a PU.

[0150] In some examples, operations performed with respect to a luma coding block need not be repeated for the chroma coding blocks. As one example, operations to identify a motion vector (MV) and reference picture for a luma coding block need not be repeated for identifying a MV and reference picture for the chroma blocks. Rather, the MV for the luma coding block may be scaled to determine the MV for the chroma

blocks, and the reference picture may be the same. As another example, the intra-prediction process may be the same for the luma coding blocks and the chroma coding blocks.

[0151] Video encoder 200 represents an example of a device configured to encode video data including a memory configured to store video data, and one or more processing units implemented in circuitry and configured to determine a size of a transform unit, wherein the transform unit comprises N coefficient groups and each coefficient group comprises M coefficients, wherein M and N are integer values; determine a regular bin count threshold for the transform unit based on the size of the transform unit; context encode syntax elements for the transform unit until the first number of regular coded bins is reached; and in response to reaching the first number of regular coded bins, bypass encode additional syntax elements of the transform unit, wherein the additional syntax elements represent values for a second set of coefficients of the transform unit.

[0152] FIG. 8 is a block diagram illustrating an example video decoder 300 that may perform the techniques of this disclosure. FIG. 8 is provided for purposes of explanation and is not limiting on the techniques as broadly exemplified and described in this disclosure. For purposes of explanation, this disclosure describes video decoder 300 is described according to the techniques of JEM and HEVC. However, the techniques of this disclosure may be performed by video coding devices that are configured to other video coding standards.

[0153] In the example of FIG. 8, video decoder 300 includes coded picture buffer (CPB) memory 320, entropy decoding unit 302, prediction processing unit 304, inverse quantization unit 306, inverse transform processing unit 308, reconstruction unit 310, filter unit 312, and decoded picture buffer (DPB) 314. Prediction processing unit 304 includes motion compensation unit 316 and intra-prediction unit 318. Prediction processing unit 304 may include addition units to perform prediction in accordance with other prediction modes. As examples, prediction processing unit 304 may include a palette unit, an intra-block copy unit (which may form part of motion compensation unit 316), an affine unit, a linear model (LM) unit, or the like. In other examples, video decoder 300 may include more, fewer, or different functional components.

[0154] CPB memory 320 may store video data, such as an encoded video bitstream, to be decoded by the components of video decoder 300. The video data stored in CPB memory 320 may be obtained, for example, from computer-readable medium 110 (FIG.

1). CPB memory 320 may include a CPB that stores encoded video data (e.g., syntax elements) from an encoded video bitstream. Also, CPB memory 320 may store video data other than syntax elements of a coded picture, such as temporary data representing outputs from the various units of video decoder 300. DPB 314 generally stores decoded pictures, which video decoder 300 may output and/or use as reference video data when decoding subsequent data or pictures of the encoded video bitstream. CPB memory 320 and DPB 314 may be formed by any of a variety of memory devices, such as dynamic random access memory (DRAM), including synchronous DRAM (SDRAM), magnetoresistive RAM (MRAM), resistive RAM (RRAM), or other types of memory devices. CPB memory 320 and DPB 314 may be provided by the same memory device or separate memory devices. In various examples, CPB memory 320 may be on-chip with other components of video decoder 300, or off-chip relative to those components. **[0155]** Additionally or alternatively, in some examples, video decoder 300 may retrieve coded video data from memory 120 (FIG. 1). That is, memory 120 may store data as discussed above with CPB memory 320. Likewise, memory 120 may store instructions to be executed by video decoder 300, when some or all of the functionality of video decoder 300 is implemented in software to be executed by processing circuitry of video decoder 300.

[0156] The various units shown in FIG. 8 are illustrated to assist with understanding the operations performed by video decoder 300. The units may be implemented as fixed-function circuits, programmable circuits, or a combination thereof. Similar to FIG. 7, fixed-function circuits refer to circuits that provide particular functionality and are preset on the operations that can be performed. Programmable circuits refer to circuits that can be programmed to perform various tasks, and provide flexible functionality in the operations that can be performed. For instance, programmable circuits may execute software or firmware that cause the programmable circuits to operate in the manner defined by instructions of the software or firmware. Fixed-function circuits may execute software instructions (e.g., to receive parameters or output parameters), but the types of operations that the fixed-function circuits perform are generally immutable. In some examples, the one or more of the units may be distinct circuit blocks (fixed-function or programmable), and in some examples, the one or more units may be integrated circuits.

[0157] Video decoder 300 may include ALUs, EFUs, digital circuits, analog circuits, and/or programmable cores formed from programmable circuits. In examples where the

operations of video decoder 300 are performed by software executing on the programmable circuits, on-chip or off-chip memory may store instructions (e.g., object code) of the software that video decoder 300 receives and executes.

[0158] Entropy decoding unit 302 may receive encoded video data from the CPB and entropy decode the video data to reproduce syntax elements, including the syntax elements described above for coefficient coding. Prediction processing unit 304, inverse quantization unit 306, inverse transform processing unit 308, reconstruction unit 310, and filter unit 312 may generate decoded video data based on the syntax elements extracted from the bitstream.

[0159] In general, video decoder 300 reconstructs a picture on a block-by-block basis. Video decoder 300 may perform a reconstruction operation on each block individually (where the block currently being reconstructed, i.e., decoded, may be referred to as a “current block”).

[0160] Entropy decoding unit 302 may entropy decode syntax elements defining quantized transform coefficients of a quantized transform coefficient block, as well as transform information, such as a quantization parameter (QP) and/or transform mode indication(s). Inverse quantization unit 306 may use the QP associated with the quantized transform coefficient block to determine a degree of quantization and, likewise, a degree of inverse quantization for inverse quantization unit 306 to apply. Inverse quantization unit 306 may, for example, perform a bitwise left-shift operation to inverse quantize the quantized transform coefficients. Inverse quantization unit 306 may thereby form a transform coefficient block including transform coefficients.

[0161] After inverse quantization unit 306 forms the transform coefficient block, inverse transform processing unit 308 may apply one or more inverse transforms to the transform coefficient block to generate a residual block associated with the current block. For example, inverse transform processing unit 308 may apply an inverse DCT, an inverse integer transform, an inverse Karhunen-Loeve transform (KLT), an inverse rotational transform, an inverse directional transform, or another inverse transform to the coefficient block.

[0162] Furthermore, prediction processing unit 304 generates a prediction block according to prediction information syntax elements that were entropy decoded by entropy decoding unit 302. For example, if the prediction information syntax elements indicate that the current block is inter-predicted, motion compensation unit 316 may generate the prediction block. In this case, the prediction information syntax elements

may indicate a reference picture in DPB 314 from which to retrieve a reference block, as well as a motion vector identifying a location of the reference block in the reference picture relative to the location of the current block in the current picture. Motion compensation unit 316 may generally perform the inter-prediction process in a manner that is substantially similar to that described with respect to motion compensation unit 224 (FIG. 7).

[0163] As another example, if the prediction information syntax elements indicate that the current block is intra-predicted, intra-prediction unit 318 may generate the prediction block according to an intra-prediction mode indicated by the prediction information syntax elements. Again, intra-prediction unit 318 may generally perform the intra-prediction process in a manner that is substantially similar to that described with respect to intra-prediction unit 226 (FIG. 7). Intra-prediction unit 318 may retrieve data of neighboring samples to the current block from DPB 314.

[0164] Reconstruction unit 310 may reconstruct the current block using the prediction block and the residual block. For example, reconstruction unit 310 may add samples of the residual block to corresponding samples of the prediction block to reconstruct the current block.

[0165] Filter unit 312 may perform one or more filter operations on reconstructed blocks. For example, filter unit 312 may perform deblocking operations to reduce blockiness artifacts along edges of the reconstructed blocks. Operations of filter unit 312 are not necessarily performed in all examples.

[0166] Video decoder 300 may store the reconstructed blocks in DPB 314. As discussed above, DPB 314 may provide reference information, such as samples of a current picture for intra-prediction and previously decoded pictures for subsequent motion compensation, to prediction processing unit 304. Moreover, video decoder 300 may output decoded pictures from DPB for subsequent presentation on a display device, such as display device 118 of FIG. 1.

[0167] In this manner, video decoder 300 represents an example of a video decoding device including a memory configured to store video data, and one or more processing units implemented in circuitry and configured to determine a size of a transform unit of the video data, wherein the transform unit comprises N coefficient groups and each coefficient group comprises M coefficients, wherein M and N are integer values; determine a regular bin count threshold for the transform unit based on the size of the transform unit; context decode syntax elements for the transform unit until the first

number of regular coded bins is reached; in response to reaching the first number of regular coded bins, bypass decode additional syntax elements; determine values for a first set of coefficients of the transform unit based on the context decoded syntax elements; and determine values for a second set of coefficients of the transform unit based on the additional syntax elements.

[0168] FIGS. 9A and 9B show examples of a CABAC process at a bin n . In example 400 of FIG. 9A, at bin n the range at bin 2 includes the RangeMPS and RangeLPS given by the probability of the least probable symbol (LPS) (p_σ) given a certain context state (σ). Example 400 shows the update of the range at bin $n+1$ when the value of bin n is equal to the most probable symbol (MPS). In this example, the low stays the same, but the value of the range at bin $n+1$ is reduced to the value of RangeMPS at bin n . Example 402 of FIG. 9B shows the update of the range at bin $n+1$ when the value of bin n is not equal to the MPS (i.e., equal to the LPS). In this example, the low is moved to the lower range value of RangeLPS at bin n . In addition, the value of the range at bin $n+1$ is reduced to the value of RangeLPS at bin n .

[0169] In one example of the HEVC video coding process, range is expressed with 9 bits and the low with 10 bits. There is a renormalization process to maintain the range and low values at sufficient precision. The renormalization occurs whenever the range is less than 256. Therefore, the range is always equal or larger than 256 after renormalization. Depending on the values of range and low, the binary arithmetic coder (BAC) outputs to the bitstream, a '0,' or a '1,' or updates an internal variable (called BO: bits-outstanding) to keep for future outputs. FIG. 10 shows examples of BAC output depending on the range. For example, a '1' is output to the bitstream when the range and low are above a certain threshold (e.g., 512). A '0' is output to the bitstream when the range and low are below a certain threshold (e.g., 512). Nothing is output to the bitstream when the range and lower are between certain thresholds. Instead, the BO value is incremented and the next bin is encoded.

[0170] In the CABAC context model of H.264/AVC and in some examples of HEVC, there are 128 states. There are 64 possible LPS probabilities (denoted by state σ) that can be from 0 to 63. Each MPS can be zero or one. As such, the 128 states are 64 state probabilities times the 2 possible values for MPS (0 or 1). Therefore, the state can be indexed with 7 bits.

[0171] To reduce the computation of deriving LPS ranges ($rangeLPS_\sigma$), results for all cases may pre-calculated and stored as approximations in a look-up table. Therefore,

the LPS range can be obtained without any multiplication by using a simple table lookup. Avoiding multiplication can be important for some devices or applications, since this operation may cause significant latency in many hardware architectures.

[0172] A 4-column pre-calculated LPS range table may be used instead of the multiplication. The range is divided into four segments. The segment index can be derived by the question $(range \gg 6) \& 3$. In effect, the segment index is derived by shifting and dropping bits from the actual range. The following Table 1 shows the possible ranges and their corresponding indexes.

TABLE 1 – Range Index

Range	256-319	320-383	384-447	448-511
$(range \gg 6) \& 3$	0	1	2	3

[0173] The LPS range table has then 64 entries (one for each probability state) times 4 (one for each range index). Each entry is the Range LPS, that is, the value of multiplying the range times the LPS probability. An example of part of this table is shown in the following Table 2. Table 2 depicts probability states 9-12. In one proposal for HEVC, the probability states may range from 0-63.

TABLE 2 – RangeLPS

Prob State (σ)	RangeLPS			
	Index 0	Index 1	Index 2	Index 3
...	...	...	...	...
9	90	110	130	150
10	85	104	123	142
11	81	99	117	135
12	77	94	111	128
...	...	...	...	...

[0174] In each segment (i.e., range value), the LPS range of each probability state σ is pre-defined. In other words, the LPS range of a probability state σ is quantized into four values (i.e., one value for each range index). The specific LPS range used at a given point depends on which segment the range belongs to. The number of possible LPS

ranges used in the table is a trade-off between the number of table columns (i.e., the number of possible LPS range values) and the LPS range precision. Generally speaking, more columns results in smaller quantization errors of LPS range values, but also increases the need for more memory to store the table. Fewer columns increases quantization errors, but also reduces the memory needed to store the table.

[0175] As described above, each LPS probability state has a corresponding probability. The probability p for each state is derived as follows:

$$p_{\sigma} = \alpha p_{\sigma-1}$$

where the state σ is from 0 to 63. The constant α represents the amount of probability change between each context state. In one example, $\alpha=0.9493$, or, more precisely, $\alpha=(0.01875/0.5)^{1/63}$. The probability at state $\sigma = 0$ is equal to 0.5 (i.e., $p_0=1/2$). That is, at context state 0, the LPS and MPS are equally probable. The probability at each successive state is derived by multiplying the previous state by α . As such, the probability of the LPS occurring at context state $\alpha = 1$ is $p_0 * 0.9493$ ($0.5 * 0.9493 = .47465$). As such, as the index of state α increases, the probability of the LPS occurring goes down.

[0176] CABAC is adaptive because the probability states are updated in order to follow the signal statistics (i.e., the values of previously coded bins). The update process is as follows. For a given probability state, the update depends on the state index and the value of the encoded symbol identified either as an LPS or an MPS. As a result of the updating process, a new probability state is derived, which consists of a potentially modified LPS probability estimate and, if necessary, a modified MPS value.

[0177] In the event of a bin value equaling the MPS, a given state index may be incremented by 1. This is for all states except when an MPS occurs at state index 62, where the LPS probability is already at its minimum (or equivalently, the maximum MPS probability is reached). In this case, the state index 62 remains fixed until an LPS is seen, or the last bin value is encoded (state 63 is used for the special case of the last bin value). When an LPS occurs, the state index is changed by decrementing the state index by a certain amount, as shown in the equation below. This rule applies in general to each occurrence of a LPS with the following exception. Assuming a LPS has been encoded at the state with index $\sigma=0$, which corresponds to the equi-probable case, the state index remains fixed, but the MPS value will be toggled such that the value of the LPS and MPS will be interchanged. In all other cases, no matter which symbol has

been encoded, the MPS value will not be altered. The derivation of the transition rules for the LPS probability is based on the following relation between a given LPS probability p_{old} and its updated counterpart p_{new} :

$$p_{new} = \max(\alpha p_{old}, p_{62}) \quad \text{if a MPS occurs}$$

$$p_{new} = (1 - \alpha) + \alpha p_{old} \quad \text{if a LPS occurs}$$

[0178] With regard to a practical implementation of the probability estimation process in CABAC, it is important to note that all transition rules may be realized by at most two tables each having 63 entries of 6-bit unsigned integer values. In some examples, state transitions may be determined with a single table TransIdxLPS, which determines, for a given state index σ , the new updated state index TransIdxLPS [σ] in case an LPS has been observed. The MPS-driven transitions can be obtained by a simple (saturated) increment of the state index by the fixed value of 1, resulting in an updated state index $\min(\sigma+1, 62)$. Table 3 below is an example of a partial TransIdxLPS table.

TABLE 3 - TransIdxLPS

Prob State (σ)	New State TransIdxLPS [σ]
...	...
9	6
10	8
11	8
12	8
...	...

[0179] The techniques described above with respect to FIGS. 9A, 9B and 10 merely represent one example implementation of CABAC. It should be understood that the techniques of this disclosure are not limited only to this described implementation of CABAC. For example, in older BAC approaches (e.g., the BAC approach used in H.264/AVC), the tables RangeLPS and TransIdxLPS were tuned for low resolution videos, (i.e., common intermediate format (CIF) and quarter-CIF (QCIF) videos). With HEVC and future codecs such as VVC, a large amount of video content is high definition (HD) and, in some cases, greater than HD. Video content that is HD or greater than HD resolution tends to have different statistics than the 10-year-old QCIF sequences used to develop H.264/AVC. As such, the tables RangeLPS and TransIdxLPS from H.264/AVC may cause adaptation between states in a manner that is

too quick. That is, the transitions between probability states, especially when an LPS occurs, can be too great for the smoother, higher resolution content of HD video. Thus, the probability models used according to conventional techniques may not be as accurate for HD and extra-HD content. In addition, as HD video content includes a greater range of pixel values, the H.264/AVC tables do not include enough entries to account for the more extreme values that may be present in HD content.

[0180] As such, for HEVC and for future coding standards such as VVC, the RangeLPS and TransIdxLPS tables may be modified to account for the characteristics of this new content. In particular, the BAC processes for HEVC and future coding standards may use tables that allow for a slower adaptation process and may account for more extreme cases (i.e., skewed probabilities). Thus, as one example, the RangeLPS and TransIdxLPS tables may be modified to achieve these goals by including more probability states and ranges than used in BAC with H.264/AVC or HEVC.

[0181] FIG. 11 is a block diagram of an example entropy encoding unit 220 that may be configured to perform CABAC in accordance with the techniques of this disclosure. A syntax element 418 is input into the entropy encoding unit 220. If the syntax element is already a binary-value syntax element (i.e., a syntax element that only has a value of 0 and 1), the step of binarization may be skipped. If the syntax element is a non-binary valued syntax element (e.g., a syntax element represented by multiple bits, such as transform coefficient levels), the non-binary valued syntax element is binarized by binarizer 420. Binarizer 420 performs a mapping of the non-binary valued syntax element into a sequence of binary decisions. These binary decisions are often called “bins.” For example, for transform coefficient levels, the value of the level may be broken down into successive bins, each bin indicating whether or not the absolute value of coefficient level is greater than some value. For example, bin 0 (sometimes called a significance flag) indicates if the absolute value of the transform coefficient level is greater than 0 or not. Bin 1 indicates if the absolute value of the transform coefficient level is greater than 1 or not, and so on. A unique mapping may be developed for each non-binary valued syntax element.

[0182] Each bin produced by binarizer 420 is fed to the binary arithmetic coding side of entropy encoding unit 220. That is, for a predetermined set of non-binary valued syntax elements, each bin type (e.g., bin 0) is coded before the next bin type (e.g., bin 1). Coding may be performed in either regular mode or bypass mode. In bypass mode, bypass coding engine 426 performs arithmetic coding using a fixed probability model,

for example, using Golomb-Rice or exponential Golomb coding. Bypass mode is generally used for more predictable syntax elements.

[0183] Coding in regular mode involves performing CABAC. Regular mode CABAC is for coding bin values where the probability of a value of a bin is predictable given the values of previously coded bins. The probability of a bin being an LPS is determined by context modeler 422. Context modeler 422 outputs the bin value and the context model (e.g., the probability state σ). The context model may be an initial context model for a series of bins, or may be determined based on the coded values of previously coded bins. As described above, the context modeler may update the state based on whether or not the previously-coded bin was an MPS or an LPS.

[0184] After the context model and probability state σ are determined by context modeler 422, regular coding engine 424 performs BAC on the bin value. According to the techniques of this disclosure, regular coding engine 424 performs BAC using TransIdxLPS table 430 that includes more than 64 probability states σ . In one example, the number of probability states is 128. TransIdxLPS is used to determine which probability state is used for a next bin (bin $n+1$) when the previous bin (bin n) is an LPS. Regular coding engine 424 may also use a RangeLPS table 128 to determine the range value for an LPS given a particular probability state σ . However, according to the techniques of this disclosure, rather than using all possible probability states σ of the TransIdxLPS table 430, the probability state indexes σ are mapped to grouped indexes for use in the RangeLPS table. That is, each index into the RangeLPS table 428 may represent two or more of the total number of probability states. The mapping of probability state index σ to grouped indexes may be linear (e.g., by dividing by two), or may be non-linear (e.g., a logarithmic function or mapping table).

[0185] In other examples of the disclosure, the difference between successive probability states may be made smaller by setting the parameter α to be greater than 0.9493. In one example, $\alpha = 0.9689$. In another example of the disclosure, the highest probability (p_0) of an LPS occurring may be set to be lower than 0.5. In one example, p_0 may be equal to 0.493.

[0186] In accordance with one or more techniques of this disclosure, as opposed to using the same value of a variable used to update a probability state in a binary arithmetic coding process (e.g., one or more of a window size, a scaling factor (α), and a probability updating speed), entropy encoding unit 220 may use different values of the

variable for different context models and/or different syntax elements. For instance, entropy encoding unit 220 may determine, for a context model of a plurality of context models, a value of a variable used to update a probability state in a binary arithmetic coding process and update the probability state based on the determined value.

[0187] FIG. 12 is a block diagram of an example entropy decoding unit 302 that may be configured to perform CABAC in accordance with the techniques of this disclosure.

The entropy decoding unit 302 of FIG. 12 performs CABAC in an inverse manner as that of entropy encoding unit 220 described in FIG. 11. Coded bits from bitstream 448 are input into entropy decoding unit 302. The coded bits are fed to either context modeler 450 or bypass decoding engine 452 based on whether or not the coded bits were entropy coded using bypass mode or regular mode. If the coded bits were coded in bypass mode, bypass decoding engine 452 may, for example, use Golomb-Rice or exponential Golomb decoding to retrieve the binary-valued syntax elements or bins of non-binary syntax elements.

[0188] If the coded bits were coded in regular mode, context modeler 450 may determine a probability model for the coded bits and regular decoding engine 454 may decode the coded bits to produce bins of non-binary valued syntax elements (or the syntax elements themselves if binary-valued). After the context model and probability state σ is determined by context modeler 450, regular decoding engine 454 performs BAC on the bin value. According to the techniques of this disclosure, regular decoding engine 454 performs BAC using TransIdxLPS table 458 that includes more than 64 probability states σ . In one example, the number of probability states is 128, although other numbers of probability states could be defined, consistent with the techniques of this disclosure. TransIdxLPS table 458 is used to determine which probability state is used for a next bin (bin $n+1$) when the previous bin (bin n) is an LPS. Regular decoding engine 454 may also use a RangeLPS table 456 to determine the range value for an LPS given a particular probability state σ . However, according to the techniques of this disclosure, rather than using all possible probability states σ of the TransIdxLPS table 458, the probability state indexes σ are mapped to grouped indexes for use in RangeLPS table 456. That is, each index into RangeLPS table 456 may represent two or more of the total number of probability states. The mapping of probability state index σ to grouped indexes may be linear (e.g., by dividing by two), or may be non-linear (e.g., a logarithmic function or mapping table).

[0189] In other examples of the disclosure, the difference between successive probability states may be made smaller by setting the parameter α to be greater than 0.9493. In one example, $\alpha = 0.9689$. In another example of the disclosure, the highest probability (p_0) of an LPS occurring may be set to be lower than 0.5. In one example, p_0 may be equal to 0.493.

[0190] After the bins are decoded by regular decoding engine 454, a reverse binarizer 460 may perform a reverse mapping to convert the bins back into the values of the non-binary valued syntax elements.

[0191] FIG. 13 is a flowchart illustrating an example operation of a video encoder for encoding a current block of video data. The current block may include a current CU. Although described with respect to video encoder 200 (FIGS. 1 and 7), it should be understood that other devices may be configured to perform an operation similar to that of FIG. 13.

[0192] In this example, video encoder 200 initially predicts the current block (550). For example, video encoder 200 may form a prediction block for the current block. Video encoder 200 may then calculate a residual block for the current block (552). To calculate the residual block, video encoder 200 may calculate a difference between the original, uncoded block and the prediction block for the current block. Video encoder 200 may then transform and quantize coefficients of the residual block (554). Next, video encoder 200 may scan the quantized transform coefficients of the residual block (556). During the scan, or following the scan, video encoder 200 may entropy encode the coefficients (558). For example, video encoder 200 may encode the coefficients using CAVLC or CABAC. Video encoder 200 may then output the entropy coded data of the block (560).

[0193] FIG. 14 is a flowchart illustrating an example operation of a video decoder for decoding a current block of video data. The current block may include a current CU. Although described with respect to video decoder 300 (FIGS. 1 and 8), it should be understood that other devices may be configured to perform an operation similar to that of FIG. 14.

[0194] Video decoder 300 may receive entropy coded data for the current block, such as entropy coded prediction information and entropy coded data for coefficients of a residual block corresponding to the current block (570). Video decoder 300 may entropy decode the entropy coded data to determine prediction information for the current block and to reproduce coefficients of the residual block (572). Video decoder

300 may predict the current block (574), e.g., using an intra- or inter-prediction mode as indicated by the prediction information for the current block, to calculate a prediction block for the current block. Video decoder 300 may then inverse scan the reproduced coefficients (576), to create a block of quantized transform coefficients. Video decoder 300 may then inverse quantize and inverse transform the coefficients to produce a residual block (578). Video decoder 300 may ultimately decode the current block by combining the prediction block and the residual block (580).

[0195] FIG. 15 is a flowchart illustrating an example operation of a video encoder for encoding a current block of video data. The current block may include a current CU. Although described with respect to video encoder 200 (FIGS. 1 and 7), it should be understood that other devices may be configured to perform an operation similar to that of FIG. 15.

[0196] In the example of FIG. 15, video encoder 200 determines a size of a transform unit (600). The transform unit includes N coefficient groups, and each coefficient group includes M coefficients. M and N are integer values. In one example, the transform unit may be a luma transform unit, and M may equal 16. In another example, the transform unit may be a chroma transform unit, and M may equal 4.

[0197] Video encoder 200 determines a regular bin count threshold for the transform unit based on the size of the transform unit (602). To determine the regular bin count threshold for the transform unit based on the size of the transform unit, video encoder 200 may, for example, set the regular bin count threshold for the transform unit equal to $N \times 28$. To determine the regular bin count threshold for the transform unit based on the size of the transform unit, video encoder 200 may, for example, set the regular bin count threshold for the transform unit equal to 1.75 multiplied by a total number of coefficients in the transform unit.

[0198] In one example, to determine the size of the transform unit, video encoder 200 may determine a size of a non-zeroed out region of the transform unit, and to determine the regular bin count threshold for the transform unit based on the size of the transform unit, video encoder 200 may set the regular bin count threshold for the transform unit equal to 28 multiplied by a number of coefficient groups in the non-zeroed out region of the transform unit. In another example, to determine the size of the transform unit, video encoder 200 may determine a size of a non-zeroed out region of the transform unit, and to determine the regular bin count threshold for the transform unit based on the size of the transform unit, video encoder 200 may set the regular bin count threshold for

the transform unit equal to 1.75 multiplied by a total number of coefficients in the non-zeroed out region of the transform unit. In an example where the transform unit is a chroma transform unit, to determine the regular bin count threshold for the transform unit based on the size of the transform unit, video encoder 200 may set the regular bin count threshold for the transform unit equal to $N*7$.

[0199] Video encoder 200 context encodes syntax elements for the transform unit until the first number of regular coded bins is reached (604). The context encoded syntax elements may, for example, include one or more significance flags (e.g., SIG as described above), one or more parity level flags (e.g., Par as described above), and one or more first flags (e.g., Gt1 and/or Gt2 as described above).

[0200] In response to reaching the first number of regular coded bins, video encoder 200 bypass encodes additional syntax elements of the transform unit (606). The additional syntax elements represent values for a second set of coefficients of the transform unit. To bypass encode the additional syntax elements, video encoder 200 may encode the additional syntax elements using one or both of Golomb-Rice coding or Exponential-Golomb coding.

[0201] Video encoder 200 may output, either for transmission or to a storage medium for example, a bitstream of encoded video data that includes context encoded syntax elements and the bypass encoded additional syntax elements.

[0202] FIG. 16 is a flowchart illustrating an example operation of a video decoder for decoding a current block of video data. The current block may include a current CU. Although described with respect to video decoder 300 (FIGS. 1 and 8), it should be understood that other devices may be configured to perform an operation similar to that of FIG. 16.

[0203] Video decoder 300 determines a size of a transform unit (610). The transform unit may include N coefficient groups, and each coefficient group may include M coefficients. M and N are integer values. In one example, the transform unit may be a luma transform unit, and M may equal 16. In another example, the transform unit may be a chroma transform unit, and M may equal 4.

[0204] Video decoder 300 determines a regular bin count threshold for the transform unit based on the size of the transform unit (612). To determine the regular bin count threshold for the transform unit based on the size of the transform unit, video decoder 300 may, for example, set the regular bin count threshold for the transform unit equal to $N*28$. To determine the regular bin count threshold for the transform unit based on the

size of the transform unit, video decoder 300 may, for example, set the regular bin count threshold for the transform unit equal to 1.75 multiplied by a total number of coefficients in the transform unit.

[0205] In one example, to determine the size of the transform unit, video decoder 300 may determine a size of a non-zeroed out region of the transform unit, and to determine the regular bin count threshold for the transform unit based on the size of the transform unit, video decoder 300 may set the regular bin count threshold for the transform unit equal to 28 multiplied by a number of coefficient groups in the non-zeroed out region of the transform unit. In another example, to determine the size of the transform unit, video decoder 300 may determine a size of a non-zeroed out region of the transform unit, and to determine the regular bin count threshold for the transform unit based on the size of the transform unit, video decoder 300 may set the regular bin count threshold for the transform unit equal to 1.75 multiplied by a total number of coefficients in the non-zeroed out region of the transform unit. In an example where the transform unit is a chroma transform unit, to determine the regular bin count threshold for the transform unit based on the size of the transform unit, video decoder 300 may set the regular bin count threshold for the transform unit equal to $N \times 7$.

[0206] Video decoder 300 context decodes syntax elements for the transform unit until the first number of regular coded bins is reached (614). The context decoded syntax elements may, for example, include one or more significance flags (e.g., SIG as described above), one or more parity level flags (e.g., Par as described above), and one or more first flags (e.g., Gt1 and/or Gt2 as described above).

[0207] In response to reaching the first number of regular coded bins, video decoder 300 bypass decodes additional syntax elements of the transform unit (616). To bypass decode the additional syntax elements, video decoder 300 may decode the additional syntax elements using one or both of Golomb-Rice coding or Exponential-Golomb coding.

[0208] Video decoder 300 determines values for a first set of coefficients of the transform unit based on the context decoded syntax elements (618). Video decoder 300 may, for example, determine the first set of coefficients using SIG, Par, Gt1, and Gt2 values, as well as remLevel values as described above. Referring back to FIG. 5, the first set of coefficients may, for example, include coefficients 0 through K.

[0209] Video decoder 300 determines values for a second set of coefficients of the transform unit based on the additional syntax elements (620). Video decoder 300 may,

for example, determine the second set of coefficients using only remLevel values determined in the manner described above. Referring back to FIG. 5, the second set of coefficients may, for example, include coefficients K+1 through N.

[0210] Video decoder 300 may also determine a decoded transform block based on the values for the first set of coefficients and the values for the second set of coefficients; add the decoded transform block to a prediction block to determine a reconstructed block; perform one or more filtering operations on the reconstructed block to determine a decoded block of video data; and output a decoded picture of video data that includes the decoded block of video data.

[0211] 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.

[0212] 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.

[0213] By way of example, and not limitation, such computer-readable storage media can include one or more of 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.

[0214] Instructions may be executed by one or more processors, such as one or more DSPs, general purpose microprocessors, ASICs, FPGAs, or other equivalent integrated or discrete logic circuitry. Accordingly, the 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.

[0215] 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.

[0216] Various examples have been described. These and other examples are within the scope of the following claims.

CLAIMS

1. A method for decoding video data, the method comprising:
 - determining a size of a non-zeroed out region of a transform unit;
 - defining a regular bin count threshold for the transform unit as a number of bins equal to 1.75 multiplied by a total number of coefficients in the non-zeroed out region of the transform unit;
 - context decoding syntax elements in groups of 4 for the transform unit until the regular bin count threshold is reached, wherein:
 - the context decoded syntax elements comprise one or more significance flags and one or more parity level flags,
 - each of the one or more significance flags indicate if an absolute level for a corresponding coefficient is equal to zero,
 - each of the one or more parity level flags indicates if a corresponding coefficient has an absolute level that is even or odd,
 - in response to the context decoded syntax elements reaching the regular bin count threshold, bypass decoding additional syntax elements of the transform unit;
 - determining values for a first set of coefficients of the transform unit based on the context decoded syntax elements; and
 - determining values for a second set of coefficients of the transform unit based on the additional syntax elements.
2. The method of claim 1, wherein the transform unit comprises N coefficient groups and each coefficient group comprises M coefficients, wherein M and N are integer values, wherein the transform unit comprises a luma transform unit and M equals 16.
3. The method of claim 1, wherein bypass decoding the additional syntax elements comprises decoding the additional syntax elements using one or both of Golomb-Rice coding or Exponential-Golomb coding.

4. The method of claim 1, further comprising:
 - determining a decoded transform block based on the values for the first set of coefficients and the values for the second set of coefficients;
 - adding the decoded transform block to a prediction block to determine a reconstructed block;
 - performing one or more filtering operations on the reconstructed block to determine a decoded block of video data; and
 - outputting a decoded picture of video data that includes the decoded block of video data.

5. A method for encoding video data, the method comprising:
 - determining a size of a non-zeroed out region of a transform unit;
 - defining a regular bin count threshold for the transform unit as a number of bins equal to 1.75 multiplied by a total number of coefficients in the non-zeroed out region of the transform unit;
 - context encoding syntax elements in groups of 4 for the transform unit until the regular bin count threshold is reached, wherein:
 - the context encoded syntax elements comprises one or more significance flags and one or more parity level flags,
 - each of the one or more significance flags indicate if an absolute level for a corresponding coefficient is equal to zero,
 - each of the one or more parity level flags indicates if a corresponding coefficient has an absolute level that is even or odd, and
 - the context encoded syntax elements represent values for a first set of coefficients of the transform unit; and
 - in response to the context encoded syntax elements reaching the regular bin count threshold, bypass encoding additional syntax elements for the transform unit, wherein the additional syntax elements represent values for a second set of coefficients of the transform unit.

6. The method of claim 5, wherein the transform unit comprises N coefficient groups and each coefficient group comprises M coefficients, wherein M and N are integer values, wherein the transform unit comprises a luma transform unit and M equals 16.

7. The method of claim 5, wherein bypass encoding the additional syntax elements comprises encoding the additional syntax elements using one or both of Golomb-Rice coding or Exponential-Golomb coding.

8. A device for decoding video data, the device comprising:
 a memory configured to store video data; and
 one or more processors implemented in circuitry and configured to:
 determine a size of a non-zeroed out region of a transform unit;
 define a regular bin count threshold for the transform unit as a number of bins equal to 1.75 multiplied by a total number of coefficients in the non-zeroed out region of the transform unit;
 context decode syntax elements in groups of 4 for the transform unit until the regular bin count threshold is reached, wherein:
 the context decoded syntax elements comprise one or more significance flags and one or more parity level flags,
 each of the one or more significance flags indicate if an absolute level for a corresponding coefficient is equal to zero, and
 each of the one or more parity level flags indicates if a corresponding coefficient has an absolute level that is even or odd, in response to the context decoded syntax elements reaching the regular bin count threshold, bypass decode additional syntax elements of the transform unit;
 determine values for a first set of coefficients of the transform unit based on the context decoded syntax elements; and
 determine values for a second set of coefficients of the transform unit based on the additional syntax elements.

9. The device of claim 8, wherein the transform unit comprises N coefficient groups and each coefficient group comprises M coefficients, wherein M and N are integer values, wherein the transform unit comprises a luma transform unit and M equals 16.

10. The device of claim 8, wherein to bypass decode the additional syntax elements, the one or more processors are configured to decode the additional syntax elements using one or both of Golomb-Rice coding or Exponential-Golomb coding.

11. The device of claim 8, wherein the one or more processors are further configured to:
 - determine a decoded transform block based on the values for the first set of coefficients and the values for the second set of coefficients;
 - add the decoded transform block to a prediction block to determine a reconstructed block;
 - perform one or more filtering operations on the reconstructed block to determine a decoded block of video data; and
 - output a decoded picture of video data that includes the decoded block of video data.

12. A device for encoding video data, the device comprising:
 - a memory configured to store video data; and
 - one or more processors implemented in circuitry and configured to:
 - determine a size of a non-zeroed out region of a transform unit;
 - define a regular bin count threshold for the transform unit as a number of bins equal to 1.75 multiplied by a total number of coefficients in the non-zeroed out region of the transform unit;
 - context encode syntax elements in groups of 4 for the transform unit until the regular bin count threshold is reached, wherein:
 - the context encoded syntax elements comprises one or more significance flags and one or more parity level flags,
 - each of the one or more significance flags indicate if an absolute level for a corresponding coefficient is equal to zero,
 - each of the one or more parity level flags indicates if a corresponding coefficient has an absolute level that is even or odd, and
 - the context encoded syntax elements represent values for a first set of coefficients of the transform unit; and
 - in response to the context encoded syntax elements reaching the regular bin count threshold, bypass encode additional syntax elements for the transform unit, wherein the additional syntax elements represent values for a second set of coefficients of the transform unit.

13. The device of claim 12, wherein the transform unit comprises N coefficient groups and each coefficient group comprises M coefficients, wherein M and N are integer values, wherein the transform unit comprises a luma transform unit and M equals 16.
14. The device of claim 12, wherein to bypass encode the additional syntax elements, the one or more processors are configured to encode the additional syntax elements using one or both of Golomb-Rice coding or Exponential-Golomb coding.