



- (51) International Patent Classification:  
*H04N 19/46* (2014.01)    *H04N 19/80* (2014.01)  
*H04N 19/36* (2014.01)    *H04N 19/85* (2014.01)
- (21) International Application Number: PCT/US2016/052633
- (22) International Filing Date: 20 September 2016 (20.09.2016)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:  
62/221,586    21 September 2015 (21.09.2015)    US  
62/236,804    2 October 2015 (02.10.2015)    US  
62/241,063    13 October 2015 (13.10.2015)    US  
15/269,558    19 September 2016 (19.09.2016)    US
- (71) Applicant: **QUALCOMM INCORPORATED** [US/US];  
ATTN: International IP Administration, 5775 Morehouse  
Drive, San Diego, California 92121-1714 (US).
- (72) Inventors: **RAMASUBRAMONIAN, Adarsh Krishnan**;  
5775 Morehouse Drive, San Diego, California 92121-1714  
(US). **RUSANOVSKYY, Dmytro**; 5775 Morehouse  
Drive, San Diego, California 92121-1714 (US). **SOLE  
ROJALS, Joel**; 5775 Morehouse Drive, San Diego, Cali-  
fornia 92121-1714 (US). **LEE, Sungwon**; 5775 More-  
house Drive, San Diego, California 92121-1714 (US).  
**BUGDAYCI SANSIL, Done**; 5775 Morehouse Drive, San  
Diego, California 92121-1714 (US). **KARCZEWICZ,  
Marta**; 5775 Morehouse Drive, San Diego, California  
92121-1714 (US).
- (74) Agent: **EVANS, Matthew J.**; Shumaker & Sieffert, P.A.,  
1625 Radio Drive, Suite 300, Woodbury, Minnesota 55125  
(US).
- (81) Designated States (*unless otherwise indicated, for every  
kind of national protection available*): AE, AG, AL, AM,  
AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY,  
BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM,  
DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT,  
HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR,

[Continued on next page]

(54) Title: FIXED POINT IMPLEMENTATION OF RANGE ADJUSTMENT OF COMPONENTS IN VIDEO CODING

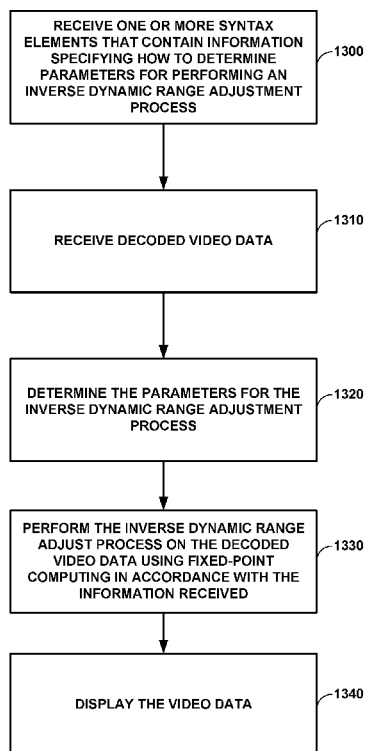


FIG. 13

(57) Abstract: Processing high dynamic range and or wide color gamut video data using a fixed-point implementation. A method of processing video data may include receiving one or more supplemental enhancement information (SEI) messages that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process, receiving decoded video data, and performing the inverse dynamic range adjustment process on the decoded video data using fixed-point computing in accordance with the information in the one or more SEI messages.



KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

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

**(84) Designated States** (*unless otherwise indicated, for every kind of regional protection available*): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ,

**Published:**

— *with international search report (Art. 21(3))*

## **FIXED POINT IMPLEMENTATION OF RANGE ADJUSTMENT OF COMPONENTS IN VIDEO CODING**

**[0001]** This application claims the benefit of U.S. Provisional Application No. 62/221,586, filed September 21, 2015, U.S. Provisional Application No. 62/236,804, filed October 2, 2015, and U.S. Provisional Application No. 62/241,063, filed October 13, 2015, the entire content of each of which is incorporated by reference herein.

### **TECHNICAL FIELD**

**[0002]** This disclosure relates to video processing.

### **BACKGROUND**

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

**[0005]** Spatial or temporal prediction results in a predictive block for a block to be coded. Residual data represents pixel differences between the original block to be coded and the predictive block. An inter-coded block is encoded according to a motion vector that points to a block of reference samples forming the predictive block, and the residual data indicating the difference between the coded block and the predictive block. An intra-coded block is encoded according to an intra-coding mode and the residual data. For further compression, the residual data may be transformed from the pixel domain to a transform domain, resulting in residual transform coefficients, which then may be quantized. The quantized transform coefficients, initially arranged in a two-dimensional array, may be scanned in order to produce a one-dimensional vector of transform coefficients, and entropy coding may be applied to achieve even more compression.

**[0006]** The total number of color values that may be captured, coded, and displayed may be defined by a color gamut. A color gamut refers to the range of colors that a device can capture (e.g., a camera) or reproduce (e.g., a display). Often, color gamuts differ from device to device. For video coding, a predefined color gamut for video data may be used such that each device in the video coding process may be configured to process pixel values in the same color gamut. Some color gamuts are defined with a larger range of colors than color gamuts that have been traditionally used for video coding. Such color gamuts with a larger range of colors may be referred to as a wide color gamut (WCG).

**[0007]** Another aspect of video data is dynamic range. Dynamic range is typically defined as the ratio between the maximum and minimum brightness (e.g., luminance) of a video signal. The dynamic range of common video data used in the past is considered to have a standard dynamic range (SDR). Other example specifications for video data define color data that has a larger ratio between the maximum and minimum brightness. Such video data may be described as having a high dynamic range (HDR).

## SUMMARY

**[0008]** This disclosure describes example techniques and devices for implementing the dynamic range adjustment of components of video data using a fixed-point implementation. The described techniques are applicable to video coding standards, not

limited to H.264/AVC, H.265/HEVC, and other standards, that are configured to encode and decode High Dynamic Range (HDR) content.

**[0009]** In one example of the disclosure, a method of processing video data comprises receiving one or more syntax elements that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process, receiving decoded video data, and performing the inverse dynamic range adjustment process on the decoded video data using fixed-point computing in accordance with the information received.

**[0010]** In another example of the disclosure, an apparatus configured to process video data comprises a memory configured to store decoded video data, and one or more processors configured to receive one or more syntax elements that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process, receive the decoded video data, and perform the inverse dynamic range adjustment process on the decoded video data using fixed-point computing in accordance with the information received.

**[0011]** In another example of the disclosure, an apparatus configured to process video data comprises means for receiving one or more syntax elements that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process, means for receiving decoded video data, and means for performing the inverse dynamic range adjustment process on the decoded video data using fixed-point computing in accordance with the information received.

**[0012]** In another example, this disclosure describes a computer-readable storage medium storing instructions that, when executed, cause one or more processors of a device configured to process video data to receive one or more syntax elements that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process, receive the decoded video data, and perform the inverse dynamic range adjustment process on the decoded video data using fixed-point computing in accordance with the information received.

**[0013]** In another example of the disclosure, a method of processing video data comprises performing a dynamic range adjustment process on video data using fixed-point computing, and generating one or more syntax elements that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process, relative to the dynamic range adjustment process, using fixed-point computing.

**[0014]** In another example of the disclosure, an apparatus configured to process video data comprises a memory configured to store video data, and one or more processors configured to perform a dynamic range adjustment process on the video data using fixed-point computing, and generate one or more syntax elements that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process, relative to the dynamic range adjustment process, using fixed-point computing.

**[0015]** In another example of the disclosure, an apparatus configured to process video data comprises means for performing a dynamic range adjustment process on video data using fixed-point computing, and means for generating one or more syntax elements that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process, relative to the dynamic range adjustment process, using fixed-point computing.

**[0016]** In another example, this disclosure describes a computer-readable storage medium storing instructions that, when executed, cause one or more processors of a device configured to process video data to perform a dynamic range adjustment process on the video data using fixed-point computing, and generate one or more syntax elements that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process, relative to the dynamic range adjustment process, using fixed-point computing.

**[0017]** 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**

**[0018]** FIG. 1 is a block diagram illustrating an example video encoding and decoding system configured to implement the techniques of the disclosure.

**[0019]** FIG. 2 is a conceptual drawing showing a typical structure of a color remapping information (CRI) process.

**[0020]** FIG. 3 is a conceptual drawing illustrating the concepts of HDR data.

**[0021]** FIG. 4 is a conceptual diagram illustrating example color gamuts.

**[0022]** FIG. 5 is a flow diagram illustrating an example of HDR/WCG representation conversion.

[0023] FIG. 6 is a flow diagram illustrating an example of HDR/WCG inverse conversion.

[0024] FIG. 7 is conceptual diagram illustrating example of Electro-optical transfer functions (EOTF) utilized for video data conversion (including SDR and HDR) from perceptually uniform code levels to linear luminance.

[0025] FIG. 8 is a block diagram illustrating an example HDR/WCG conversion apparatus operating according to the techniques of this disclosure.

[0026] FIG. 9 is a block diagram illustrating an example HDR/WCG inverse conversion apparatus according to the techniques of this disclosure.

[0027] FIG. 10 is a block diagram illustrating an example of a video encoder that may implement techniques of this disclosure.

[0028] FIG. 11 is a block diagram illustrating an example of a video decoder that may implement techniques of this disclosure.

[0029] FIG. 12 is a flowchart showing one example video processing technique of the disclosure.

[0030] FIG. 13 is a flowchart showing another example video processing technique of the disclosure.

## DETAILED DESCRIPTION

[0031] This disclosure is related to the processing and/or coding of video data with high dynamic range (HDR) and wide color gamut (WCG) representations. More specifically, the techniques of this disclosure include techniques for performing range adjustment of video data components using fixed point processing operations (e.g., as opposed to floating point processing operations). The techniques and devices described herein may improve compression efficiency of hybrid-based video coding systems (e.g., H.265/HEVC, H.264/AVC, etc.) utilized for coding video data, including HDR and WCG video data.

[0032] Video coding standards, including hybrid-based video coding standards, include ITU-T H.261, ISO/IEC MPEG-1 Visual, ITU-T H.262 or ISO/IEC MPEG-2 Visual, ITU-T H.263, ISO/IEC MPEG-4 Visual and ITU-T H.264 (also known as ISO/IEC MPEG-4 AVC), including its Scalable Video Coding (SVC) and Multi-view Video Coding (MVC) extensions. The design of a new video coding standard, namely High Efficiency Video coding (HEVC, also called H.265), has been finalized by the Joint Collaboration Team on Video Coding (JCT-VC) of ITU-T Video Coding Experts Group

(VCEG) and ISO/IEC Motion Picture Experts Group (MPEG). An HEVC draft specification referred to as HEVC Working Draft 10 (WD10), Bross et al., “High efficiency video coding (HEVC) text specification draft 10 (for FDIS & Last Call),” Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, 12th Meeting: Geneva, CH, 14–23 January 2013, JCTVC-L1003v34, is available from [http://phenix.int-evry.fr/jct/doc\\_end\\_user/documents/12\\_Geneva/wg11/JCTVC-L1003-v34.zip](http://phenix.int-evry.fr/jct/doc_end_user/documents/12_Geneva/wg11/JCTVC-L1003-v34.zip). The finalized HEVC standard is referred to as HEVC version 1.

**[0033]** A defect report, Wang et al., “High efficiency video coding (HEVC) Defect Report,” Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, 14th Meeting: Vienna, AT, 25 July–2 August 2013, JCTVC-N1003v1, is available from [http://phenix.int-evry.fr/jct/doc\\_end\\_user/documents/14\\_Vienna/wg11/JCTVC-N1003-v1.zip](http://phenix.int-evry.fr/jct/doc_end_user/documents/14_Vienna/wg11/JCTVC-N1003-v1.zip). The finalized HEVC standard document is published as ITU-T H.265, Series H: Audiovisual and Multimedia Systems, Infrastructure of audiovisual services – Coding of moving video, High efficiency video coding, Telecommunication Standardization Sector of International Telecommunication Union (ITU), April 2013, and another version of the finalized HEVC standard was published in October 2014. A copy of the H.265/HEVC specification text may be downloaded from <http://www.itu.int/rec/T-REC-H.265-201504-I/en>.

**[0034]** FIG. 1 is a block diagram illustrating an example video encoding and decoding system 10 that may utilize techniques of this disclosure. As shown in FIG. 1, system 10 includes a source device 12 that provides encoded video data to be decoded at a later time by a destination device 14. In particular, source device 12 provides the video data to destination device 14 via a computer-readable medium 16. Source device 12 and destination device 14 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 so-called “smart” phones, so-called “smart” pads, televisions, cameras, display devices, digital media players, video gaming consoles, video streaming devices, broadcast receiver device, or the like. In some cases, source device 12 and destination device 14 may be equipped for wireless communication.

**[0035]** Destination device 14 may receive the encoded video data to be decoded via computer-readable medium 16. Computer-readable medium 16 may comprise any type of medium or device capable of moving the encoded video data from source device 12



to destination device 14. In one example, computer-readable medium 16 may comprise a communication medium to enable source device 12 to transmit encoded video data directly to destination device 14 in real-time. The encoded video data may be modulated according to a communication standard, such as a wired or wireless communication protocol, and transmitted to destination device 14. The communication medium may comprise any wireless or wired communication medium, such as a radio frequency (RF) spectrum or one or more physical transmission lines. The communication medium may 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 12 to destination device 14.

**[0036]** In other examples, computer-readable medium 16 may include non-transitory storage media, such as a hard disk, flash drive, compact disc, digital video disc, Blu-ray disc, or other computer-readable media. In some examples, a network server (not shown) may receive encoded video data from source device 12 and provide the encoded video data to destination device 14, e.g., via network transmission. Similarly, a computing device of a medium production facility, such as a disc stamping facility, may receive encoded video data from source device 12 and produce a disc containing the encoded video data. Therefore, computer-readable medium 16 may be understood to include one or more computer-readable media of various forms, in various examples.

**[0037]** In some examples, encoded data may be output from output interface 22 to a storage device. Similarly, encoded data may be accessed from the storage device by input interface. The storage device may include any of a variety of distributed or locally accessed data storage media such as a hard drive, Blu-ray discs, DVDs, CD-ROMs, flash memory, volatile or non-volatile memory, or any other suitable digital storage media for storing encoded video data. In a further example, the storage device may correspond to a file server or another intermediate storage device that may store the encoded video generated by source device 12. Destination device 14 may access stored video data from the storage device via streaming or download. The file server may be any type of server capable of storing encoded video data and transmitting encoded video data to the destination device 14. Example file servers include a web server (e.g., for a website), an FTP server, network attached storage (NAS) devices, or a local disk drive. Destination device 14 may access the encoded video data 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., DSL, cable modem, etc.), or a combination of both that is suitable for accessing encoded video data stored on a file server. The transmission of encoded video data from the storage device may be a streaming transmission, a download transmission, or a combination thereof.

**[0038]** The techniques of this disclosure are not necessarily limited to wireless applications or settings. The techniques may be applied to video coding in support of any of a variety of multimedia applications, such as over-the-air television broadcasts, cable television transmissions, satellite television transmissions, Internet streaming video transmissions, such as dynamic adaptive streaming over HTTP (DASH), digital video that is encoded onto a data storage medium, decoding of digital video stored on a data storage medium, or other applications. In some examples, system 10 may be configured to support one-way or two-way video transmission to support applications such as video streaming, video playback, video broadcasting, and/or video telephony.

**[0039]** In the example of FIG. 1, source device 12 includes video source 18, video encoding unit 21, which includes video pre-processor unit 19 and video encoder 20, and output interface 22. Destination device 14 includes input interface 28, video decoding unit 29, which includes video post-processor unit 31 and video decoder 30, and display device 32. In accordance with this disclosure, video pre-processor unit 19 and/or video encoder 20 of source device 12 and video post-processor unit 31 and/or video decoder 30 of destination device 14 may be configured to implement the techniques of this disclosure, including signaling and related operations applied to video data in certain color spaces to enable more efficient compression of HDR and WCG video data with a fixed point implementation. In some examples, video pre-processor unit 19 may be separate from video encoder 20. In other examples, video pre-processor unit 19 may be part of video encoder 20. Likewise, in some examples, video post-processor unit 31 may be separate from video decoder 30. In other examples, video post-processor unit 31 may be part of video decoder 30. In other examples, a source device and a destination device may include other components or arrangements. For example, source device 12 may receive video data from an external video source 18, such as an external camera. Likewise, destination device 14 may interface with an external display device, rather than including an integrated display device.

**[0040]** The illustrated system 10 of FIG. 1 is merely one example. Techniques for processing HDR and WCG video data may be performed by any digital video encoding

and/or video decoding device. Moreover, the techniques of this disclosure may also be performed by a video pre-processor and/or video post-processor (e.g., video pre-processor unit 19 and video post-processor unit 31). In general, a video pre-processor may be any device configured to process video data before encoding (e.g., before HEVC encoding). In general, a video post-processor may be any device configured to process video data after decoding (e.g., after HEVC decoding). Source device 12 and destination device 14 are merely examples of such coding devices in which source device 12 generates coded video data for transmission to destination device 14. In some examples, devices 12, 14 may operate in a substantially symmetrical manner such that each of devices 12, 14 include video encoding and decoding components, as well as a video pre-processor and a video post-processor (e.g., video pre-processor unit 19 and video post-processor unit 31, respectively). Hence, system 10 may support one-way or two-way video transmission between video devices 12, 14, e.g., for video streaming, video playback, video broadcasting, or video telephony.

**[0041]** Video source 18 of source device 12 may include a video capture device, such as a video camera, a video archive containing previously captured video, and/or a video feed interface to receive video from a video content provider. As a further alternative, video source 18 may generate computer graphics-based data as the source video, or a combination of live video, archived video, and computer-generated video. In some cases, if video source 18 is a video camera, source device 12 and destination device 14 may form so-called camera phones or video phones. As mentioned above, however, the techniques described in this disclosure may be applicable to video coding and video processing, in general, and may be applied to wireless and/or wired applications. In each case, the captured, pre-captured, or computer-generated video may be encoded by video encoding unit 21. The encoded video information may then be output by output interface 22 onto a computer-readable medium 16.

**[0042]** Input interface 28 of destination device 14 receives information from computer-readable medium 16. The information of computer-readable medium 16 may include syntax information defined by video encoder 20, which is also used by video decoding unit 29, that includes syntax elements that describe characteristics and/or processing of blocks and other coded units, e.g., groups of pictures (GOPs). Display device 32 displays the decoded video data to a user, and may comprise any of a variety of display devices such as a cathode ray tube (CRT), a liquid crystal display (LCD), a plasma

display, an organic light emitting diode (OLED) display, or another type of display device.

**[0043]** As illustrated, video pre-processor unit 19 receives the video data from video source 18. Video pre-processor unit 19 may be configured to process the video data to convert it into a form that is suitable for encoding with video encoder 20. For example, video pre-processor unit 19 may perform dynamic range compacting (e.g., using a non-linear transfer function), color conversion to a more compact or robust color space, and/or floating-to-integer representation conversion. Video encoder 20 may perform video encoding on the video data outputted by video pre-processor unit 19. Video decoder 30 may perform the inverse of video encoder 20 to decode video data, and video post-processor unit 31 may perform the inverse of video pre-processor unit 19 to convert the video data into a form suitable for display. For instance, video post-processor unit 31 may perform integer-to-floating conversion, color conversion from the compact or robust color space, and/or the inverse of the dynamic range compacting to generate video data suitable for display.

**[0044]** Video encoder 20 and video decoder 30 each may be implemented as any of a variety of suitable encoder circuitry, such as one or more microprocessors, digital signal processors (DSPs), application specific integrated circuits (ASICs), field programmable gate arrays (FPGAs), discrete logic, software, hardware, firmware or any combinations thereof. 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 20 and video decoder 30 may be included in one or more encoders or decoders, either of which may be integrated as part of a combined encoder/decoder (CODEC) in a respective device.

**[0045]** Video pre-processor unit 19 and video post-processor unit 31 each may be implemented as any of a variety of suitable encoder circuitry, such as one or more microprocessors, digital signal processors (DSPs), application specific integrated circuits (ASICs), field programmable gate arrays (FPGAs), discrete logic, software, hardware, firmware or any combinations thereof. 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. As discussed above video pre-processor unit 19 and video post-processor unit 31 be separate devices

from video encoder 20 and video decoder 30, respectively. In other examples, video pre-processor unit 19 may integrate with video encoder 20 in a single device and inverse video post-processor unit 31 may be integrated with video decoder 30 in a single device.

**[0046]** In some examples, video encoder 20 and video decoder 30 operate according to a video compression standard, such as ISO/IEC MPEG-4 Visual and ITU-T H.264 (also known as ISO/IEC MPEG-4 AVC), including its Scalable Video Coding (SVC) extension, Multi-view Video Coding (MVC) extension, and MVC-based three-dimensional video (3DV) extension. In some instances, any bitstream conforming to MVC-based 3DV always contains a sub-bitstream that is compliant to a MVC profile, e.g., stereo high profile. Furthermore, there is an ongoing effort to generate a 3DV coding extension to H.264/AVC, namely AVC-based 3DV. Other examples of video coding standards include ITU-T H.261, ISO/IEC MPEG-1 Visual, ITU-T H.262 or ISO/IEC MPEG-2 Visual, ITU-T H.263, ISO/IEC MPEG-4 Visual, and ITU-T H.264, ISO/IEC Visual. In other examples, video encoder 20 and video decoder 30 may be configured to operate according to the HEVC standard.

**[0047]** In HEVC and other video coding standards, a video sequence typically includes a series of pictures. Pictures may also be referred to as “frames.” A picture may include three sample arrays, denoted  $S_L$ ,  $S_{Cb}$ , and  $S_{Cr}$ .  $S_L$  is a two-dimensional array (i.e., a block) of luma samples.  $S_{Cb}$  is a two-dimensional array of Cb chrominance samples.  $S_{Cr}$  is a two-dimensional array of Cr chrominance samples. Chrominance samples may also be referred to herein as “chroma” samples. In other instances, a picture may be monochrome and may only include an array of luma samples.

**[0048]** Video encoder 20 may generate a set of coding tree units (CTUs). Each of the CTUs may comprise a coding tree block of luma samples, two corresponding coding tree blocks of chroma samples, and syntax structures used to code the samples of the coding tree blocks. In a monochrome picture or a picture that has three separate color planes, a CTU may comprise a single coding tree block and syntax structures used to code the samples of the coding tree block. A coding tree block may be an  $N \times N$  block of samples. A CTU may also be referred to as a “tree block” or a “largest coding unit” (LCU). The CTUs of HEVC may be broadly analogous to the macroblocks of other video coding standards, such as H.264/AVC. However, a CTU is not necessarily limited to a particular size and may include one or more coding units (CUs). A slice may include an integer number of CTUs ordered consecutively in the raster scan.

**[0049]** This disclosure may use the term “video unit” or “video block” to refer to one or more blocks of samples and syntax structures used to code samples of the one or more blocks of samples. Example types of video units may include CTUs, CUs, PUs, transform units (TUs) in HEVC, or macroblocks, macroblock partitions, and so on in other video coding standards.

**[0050]** To generate a coded CTU, video encoder 20 may recursively perform quad-tree partitioning on the coding tree blocks of a CTU to divide the coding tree blocks into coding blocks, hence the name “coding tree units.” A coding block is an NxN block of samples. A CU may comprise a coding block of luma samples and two corresponding coding blocks of chroma samples of a picture that has a luma sample array, a Cb sample array and a Cr sample array, and syntax structures used to code the samples of the coding blocks. In a monochrome picture or a picture that has three separate color planes, a CU may comprise a single coding block and syntax structures used to code the samples of the coding block.

**[0051]** Video encoder 20 may partition a coding block of a CU into one or more prediction blocks. A prediction block may be a rectangular (i.e., square or non-square) block of samples on which the same prediction is applied. A prediction unit (PU) of a CU may comprise a prediction block of luma samples, two corresponding prediction blocks of chroma samples of a picture, and syntax structures used to predict the prediction block samples. In a monochrome picture or a picture that have three separate color planes, a PU may comprise a single prediction block and syntax structures used to predict the prediction block samples. Video encoder 20 may generate predictive luma, Cb and Cr blocks for luma, Cb and Cr prediction blocks of each PU of the CU.

**[0052]** Video encoder 20 may use intra prediction or inter prediction to generate the predictive blocks for a PU. If video encoder 20 uses intra prediction to generate the predictive blocks of a PU, video encoder 20 may generate the predictive blocks of the PU based on decoded samples of the picture associated with the PU.

**[0053]** If video encoder 20 uses inter prediction to generate the predictive blocks of a PU, video encoder 20 may generate the predictive blocks of the PU based on decoded samples of one or more pictures other than the picture associated with the PU. Inter prediction may be uni-directional inter prediction (i.e., uni-prediction) or bi-directional inter prediction (i.e., bi-prediction). To perform uni-prediction or bi-prediction, video encoder 20 may generate a first reference picture list (RefPicList0) and a second reference picture list (RefPicList1) for a current slice.

**[0054]** Each of the reference picture lists may include one or more reference pictures. When using uni-prediction, video encoder 20 may search the reference pictures in either or both RefPicList0 and RefPicList1 to determine a reference location within a reference picture. Furthermore, when using uni-prediction, video encoder 20 may generate, based at least in part on samples corresponding to the reference location, the predictive sample blocks for the PU. Moreover, when using uni-prediction, video encoder 20 may generate a single motion vector that indicates a spatial displacement between a prediction block of the PU and the reference location. To indicate the spatial displacement between a prediction block of the PU and the reference location, a motion vector may include a horizontal component specifying a horizontal displacement between the prediction block of the PU and the reference location and may include a vertical component specifying a vertical displacement between the prediction block of the PU and the reference location.

**[0055]** When using bi-prediction to encode a PU, video encoder 20 may determine a first reference location in a reference picture in RefPicList0 and a second reference location in a reference picture in RefPicList1. Video encoder 20 may then generate, based at least in part on samples corresponding to the first and second reference locations, the predictive blocks for the PU. Moreover, when using bi-prediction to encode the PU, video encoder 20 may generate a first motion indicating a spatial displacement between a sample block of the PU and the first reference location and a second motion indicating a spatial displacement between the prediction block of the PU and the second reference location.

**[0056]** After video encoder 20 generates predictive luma, Cb, and Cr blocks for one or more PUs of a CU, video encoder 20 may generate a luma residual block for the CU. Each sample in the CU's luma residual block indicates a difference between a luma sample in one of the CU's predictive luma blocks and a corresponding sample in the CU's original luma coding block. In addition, video encoder 20 may generate a Cb residual block for the CU. Each sample in the CU's Cb residual block may indicate a difference between a Cb sample in one of the CU's predictive Cb blocks and a corresponding sample in the CU's original Cb coding block. Video encoder 20 may also generate a Cr residual block for the CU. Each sample in the CU's Cr residual block may indicate a difference between a Cr sample in one of the CU's predictive Cr blocks and a corresponding sample in the CU's original Cr coding block.

**[0057]** Furthermore, video encoder 20 may use quad-tree partitioning to decompose the luma, Cb and, Cr residual blocks of a CU into one or more luma, Cb, and Cr transform blocks. A transform block may be a rectangular block of samples on which the same transform is applied. A transform unit (TU) of a CU may comprise a transform block of luma samples, two corresponding transform blocks of chroma samples, and syntax structures used to transform the transform block samples. In a monochrome picture or a picture that has three separate color planes, a TU may comprise a single transform block and syntax structures used to transform the transform block samples. Thus, each TU of a CU may be associated with a luma transform block, a Cb transform block, and a Cr transform block. The luma transform block associated with the TU may be a sub-block of the CU's luma residual block. The Cb transform block may be a sub-block of the CU's Cb residual block. The Cr transform block may be a sub-block of the CU's Cr residual block.

**[0058]** Video encoder 20 may apply one or more transforms to a luma transform block of a TU to generate a luma coefficient block for the TU. A coefficient block may be a two-dimensional array of transform coefficients. A transform coefficient may be a scalar quantity. Video encoder 20 may apply one or more transforms to a Cb transform block of a TU to generate a Cb coefficient block for the TU. Video encoder 20 may apply one or more transforms to a Cr transform block of a TU to generate a Cr coefficient block for the TU.

**[0059]** After generating a coefficient block (e.g., a luma coefficient block, a Cb coefficient block or a Cr coefficient block), video encoder 20 may quantize the coefficient block. 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. Furthermore, video encoder 20 may inverse quantize transform coefficients and apply an inverse transform to the transform coefficients in order to reconstruct transform blocks of TUs of CUs of a picture. Video encoder 20 may use the reconstructed transform blocks of TUs of a CU and the predictive blocks of PUs of the CU to reconstruct coding blocks of the CU. By reconstructing the coding blocks of each CU of a picture, video encoder 20 may reconstruct the picture. Video encoder 20 may store reconstructed pictures in a decoded picture buffer (DPB). Video encoder 20 may use reconstructed pictures in the DPB for inter prediction and intra prediction.



**[0060]** After video encoder 20 quantizes a coefficient block, video encoder 20 may entropy encode syntax elements that indicate the quantized transform coefficients. For example, video encoder 20 may perform Context-Adaptive Binary Arithmetic Coding (CABAC) on the syntax elements indicating the quantized transform coefficients. Video encoder 20 may output the entropy-encoded syntax elements in a bitstream.

**[0061]** Video encoder 20 may output a bitstream that includes a sequence of bits that forms a representation of coded pictures and associated data. The bitstream may comprise a sequence of network abstraction layer (NAL) units. Each of the NAL units includes a NAL unit header and encapsulates a raw byte sequence payload (RBSP). The NAL unit header may include a syntax element that indicates a NAL unit type code. The NAL unit type code specified by the NAL unit header of a NAL unit indicates the type of the NAL unit. A RBSP may be a syntax structure containing an integer number of bytes that is encapsulated within a NAL unit. In some instances, an RBSP includes zero bits.

**[0062]** Different types of NAL units may encapsulate different types of RBSPs. For example, a first type of NAL unit may encapsulate a RBSP for a picture parameter set (PPS), a second type of NAL unit may encapsulate a RBSP for a coded slice, a third type of NAL unit may encapsulate a RBSP for Supplemental Enhancement Information (SEI), and so on. A PPS is a syntax structure that may contain syntax elements that apply to zero or more entire coded pictures. NAL units that encapsulate RBSPs for video coding data (as opposed to RBSPs for parameter sets and SEI messages) may be referred to as video coding layer (VCL) NAL units. A NAL unit that encapsulates a coded slice may be referred to herein as a coded slice NAL unit. A RBSP for a coded slice may include a slice header and slice data.

**[0063]** Video decoder 30 may receive a bitstream. In addition, video decoder 30 may parse the bitstream to decode syntax elements from the bitstream. Video decoder 30 may reconstruct the pictures of the video data based at least in part on the syntax elements decoded from the bitstream. The process to reconstruct the video data may be generally reciprocal to the process performed by video encoder 20. For instance, video decoder 30 may use motion vectors of PUs to determine predictive blocks for the PUs of a current CU. Video decoder 30 may use a motion vector or motion vectors of PUs to generate predictive blocks for the PUs.

**[0064]** In addition, video decoder 30 may inverse quantize coefficient blocks associated with TUs of the current CU. Video decoder 30 may perform inverse transforms on the

coefficient blocks to reconstruct transform blocks associated with the TUs of the current CU. Video decoder 30 may reconstruct the coding blocks of the current CU by adding the samples of the predictive sample blocks for PUs of the current CU to corresponding samples of the transform blocks of the TUs of the current CU. By reconstructing the coding blocks for each CU of a picture, video decoder 30 may reconstruct the picture. Video decoder 30 may store decoded pictures in a decoded picture buffer for output and/or for use in decoding other pictures.

**[0065]** Supplemental Enhancement information (SEI) messages are often included in video bitstreams, typically to carry information that is not essential in order to decode the bitstream by the decoder (e.g., video decoder 30). The information contained in an SEI message may be useful in improving the display or processing of the decoded output; e.g. such information could be used by decoder-side entities to improve the viewability of the content. It is also possible that certain application standards could mandate the presence of such SEI messages in the bitstream so that the improvement in quality can be brought to all devices that conform to the application standard (e.g., the carriage of the frame-packing SEI message for frame-compatible plano-stereoscopic 3DTV video format, where the SEI message is carried for every frame of the video, e.g., as described in ETSI – TS 101 547-2, Digital Video Broadcasting (DVB) Plano-stereoscopic 3DTV; Part 2: Frame compatible plano-stereoscopic 3DTV, handling of recovery point SEI message, e.g., as described in 3GPP TS 26.114 v13.0.0, 3rd Generation Partnership Project, Technical Specification Group Services and System Aspects; IP Multimedia Subsystem (IMS); Multimedia Telephony; Media handling and interaction (Release 13), or use of pan-scan scan rectangle SEI message in DVB, e.g., as described in ETSI – TS 101 154, Digital Video Broadcasting (DVB); Specification for the use of Video and Audio Coding in Broadcasting Applications based on the MPEG-2 Transport Stream).

**[0066]** A tone-mapping information SEI message is used to map luma samples, or each of RGB component samples. Different values of tone\_map\_id are used to define different purposes, and the syntax of the tone-map SEI message is also modified accordingly. A value of 1 for the tone\_map\_id allows the SEI message to clip the RGB samples to a minimum and a maximum value. A value of 3 for the tone\_map\_id allows the signaling of a look up table in the form of pivot points. However, when applied, the same values are applied to all RGB components, or only applied to the luma component.

**[0067]** A knee function SEI message is used to indicate the mapping of the RGB components of the decoded pictures in the normalized linear domain. The input and output maximum luminance values are also indicated, and a look-up table maps the input luminance values to the output luminance values. The same look-up table is applied to all the three color components.

**[0068]** A color remapping information (CRI) SEI message defined in the HEVC standard is used to convey information that is used to map pictures in one color space to another. In one example, the syntax of the CRI SEI message includes three parts - first look-up table (Pre-LUT), followed by a 3x3 matrix indicating color remapping coefficients, followed by a second look-up table (Post-LUT). For each color component, e.g., R,G,B or Y,Cb,Cr, independent LUT is defined for both, Pre-LUT and Post-LUT. The CRI SEI message also includes syntax element called `colour_remap_id`, different values of which may be used to indicate different purposes of the SEI message. FIG. 2 shows a typical structure of the color remapping information process specified by a CRI SEI message.

**[0069]** Dynamic range adjustment (DRA) SEI message. The dynamic range adjustment SEI message, e.g., as described in D. Bugdayci Sansli, A. K. Ramasubramonian, D. Rusanovskyy, S. Lee, J. Sole, M. Karczewicz, Dynamic range adjustment SEI message, m36330, MPEG meeting, Warsaw, Poland, 22 – 26 June, 2015, has not been adopted as part of any video coding standard; however, the SEI message includes signaling of one set of scale and offset numbers to map the input samples. The SEI message also allows the signaling of different look-up tables for different components, and also allows for signaling optimization when the same scale and offset are to be used for more than one component. The scale and offset numbers are signaled in fixed length accuracy.

**[0070]** Next generation video applications are anticipated to operate with video data representing captured scenery with HDR and a WCG. Parameters of the utilized dynamic range and color gamut are two independent attributes of video content, and their specification for purposes of digital television and multimedia services are defined by several international standards. For example, ITU-R Rec. BT.709, “Parameter values for the HDTV standards for production and international programme exchange,” and ITU-R Rec. BT.2020, “Parameter values for ultra-high definition television systems for production and international programme exchange,” defines parameters for HDTV (high definition television) and UHDTV (ultra-high definition television), respectively, such as standard dynamic range (SDR) and color primaries that extend beyond the

standard color gamut. Rec. BT.2100, “Image parameter values for high dynamic range television for use in production and international programme exchange” defines transfer functions and representations for HDR television use, including primaries that support wide color gamut representations. There are also other standards developing organization (SDOs) documents that specify dynamic range and color gamut attributes in other systems, e.g., DCI-P3 color gamut is defined in SMPTE-231-2 (Society of Motion Picture and Television Engineers) and some parameters of HDR are defined in SMPTE-2084. A brief description of dynamic range and color gamut for video data is provided below.

**[0071]** Dynamic range is typically defined as the ratio between the maximum and minimum brightness (e.g., luminance) of the video signal. Dynamic range may also be measured in terms of ‘f-stop,’ where one f-stop corresponds to a doubling of a signal’s dynamic range. In MPEG’s definition, content that features brightness variation with more than 16 f-stops is referred as HDR content. In some terms, levels between 10 and 16 f-stops are considered as intermediate dynamic range, but it is considered HDR in other definitions. In some examples of this disclosure, HDR video content may be any video content that has a higher dynamic range than traditionally used video content with a standard dynamic range (e.g., video content as specified by ITU-R Rec. BT.709).

**[0072]** The human visual system (HVS) is capable for perceiving much larger dynamic ranges than SDR content and HDR content. However, the HVS includes an adaptation mechanism to narrow the dynamic range of the HVS to a so-called simultaneous range. The width of the simultaneous range may be dependent on current lighting conditions (e.g., current brightness). Visualization of dynamic range provided by SDR of HDTV, expected HDR of UHD TV and HVS dynamic range is shown in FIG. 3, although the exact range may vary based on each individual and display.

**[0073]** Current video application and services are regulated by ITU Rec.709 and provide SDR, typically supporting a range of brightness (e.g., luminance) of around 0.1 to 100 candelas (cd) per m<sup>2</sup> (often referred to as “nits”), leading to less than 10 f-stops. Some example next generation video services are expected to provide dynamic range of up to 16 f-stops. Although detailed specifications for such content are currently under development, some initial parameters have been specified in SMPTE-2084 and ITU-R Rec. 2020.

**[0074]** Another aspect for a more realistic video experience, besides HDR, is the color dimension. Color dimension is typically defined by the color gamut. FIG. 4 is a

conceptual diagram showing an SDR color gamut (triangle 100 based on the BT.709 color primaries), and the wider color gamut that for UHD TV (triangle 102 based on the BT.2020 color primaries). FIG. 4 also depicts the so-called spectrum locus (delimited by the tongue-shaped area 104), representing the limits of the natural colors. As illustrated by FIG. 3, moving from BT.709 (triangle 100) to BT.2020 (triangle 102) color primaries aims to provide UHD TV services with about 70% more colors. D65 specifies an example white color for the BT.709 and/or BT.2020 specifications.

[0075] Examples of color gamut specifications for the DCI-P3, BT.709, and BT.2020 color spaces are shown in Table 1.

**Table 1 - Color gamut parameters**

RGB color space parameters								
Color space	White point		Primary colors					
	xx <sub>w</sub>	yy <sub>w</sub>	xx <sub>R</sub>	yy <sub>R</sub>	xx <sub>G</sub>	yy <sub>G</sub>	xx <sub>B</sub>	yy <sub>B</sub>
DCI-P3	0.314	0.351	0.680	0.320	0.265	0.690	0.150	0.060
ITU-R BT.709	0.3127	0.3290	0.64	0.33	0.30	0.60	0.15	0.06
ITU-R BT.2020	0.3127	0.3290	0.708	0.292	0.170	0.797	0.131	0.046

[0076] As can be seen in Table 1, a color gamut may be defined by the X and Y values of a white point, and by the x and y values of the primary colors (e.g., red (R), green (G), and blue (B)). The x and y values represent normalized values that are derived from the chromaticity (X and Z) and the brightness (Y) of the colors, as is defined by the CIE 1931 color space. The CIE 1931 color space defines the links between pure colors (e.g., in terms of wavelengths) and how the human eye perceives such colors.

[0077] HDR/WCG video data is typically acquired and stored at a very high precision per component (even floating point), with the 4:4:4 chroma format and a very wide color space (e.g., CIE XYZ). This representation targets high precision and is almost mathematically lossless. However, such a format for storing HDR/WCG video data may include a lot of redundancies and may not be optimal for compression purposes. A lower precision format with HVS-based assumptions is typically utilized for state-of-the-art video applications.

**[0078]** One example of a video data format conversion process for purposes of compression includes three major processes, as shown in FIG. 5. The techniques of FIG. 5 may be performed by source device 12. Linear RGB data 110 may be HDR/WCG video data and may be stored in a floating point representation. Linear RGB data 110 may be compacted using a non-linear transfer function (TF) 112 for dynamic range compacting. Transfer function 112 may compact linear RGB data 110 using any number of non-linear transfer functions, e.g., the PQ TF as defined in SMPTE-2084. In some examples, color conversion process 114 converts the compacted data into a more compact or robust color space (e.g., a YUV or YCrCb color space) that is more suitable for compression by a hybrid video encoder. This data is then quantized using a floating-to-integer representation quantization unit 116 to produce converted HDR' data 118. In this example HDR' data 118 is in an integer representation. The HDR' data is now in a format more suitable for compression by a hybrid video encoder (e.g., video encoder 20 applying HEVC techniques). The order of the processes depicted in FIG. 5 is given as an example, and may vary in other applications. For example, color conversion may precede the TF process. In addition, additional processing, e.g. spatial subsampling, may be applied to color components.

**[0079]** The inverse conversion at the decoder side is depicted in FIG 6. The techniques of FIG. 6 may be performed by destination device 14. Converted HDR' data 120 may be obtained at destination device 14 through decoding video data using a hybrid video decoder (e.g., video decoder 30 applying HEVC techniques). HDR' data 120 may then be inverse quantized by inverse quantization unit 122. Then an inverse color conversion process 124 may be applied to the inverse quantized HDR' data. The inverse color conversion process 124 may be the inverse of color conversion process 114. For example, the inverse color conversion process 124 may convert the HDR' data from a YCrCb format back to an RGB format. Next, inverse transfer function 126 may be applied to the data to add back the dynamic range that was compacted by transfer function 112 to recreate the linear RGB data 128.

**[0080]** The techniques depicted in FIG. 5 will now be discussed in more detail. In general, a transfer function is applied to data (e.g., HDR/WCG video data) to compact the dynamic range of the data such that errors due to quantization are perceptually uniform (approximately) across the range of luminance values. Such compaction allows the data to be represented with fewer bits. In one example, the transfer function may be a one-dimensional (1D) non-linear function and may reflect the inverse of an electro-

optical transfer function (EOTF) of the end-user display, e.g., as specified for SDR in Rec. 709. In another example, the transfer function may approximate the HVS perception to brightness changes, e.g., the PQ transfer function specified in SMPTE-2084 for HDR. The inverse process of the OETF is the EOTF (electro-optical transfer function), which maps the code levels back to luminance. FIG. 7 shows several examples of non-linear transfer function used as EOTFs. The transfer functions may also be applied to each R, G and B component separately.

**[0081]** In the context of this disclosure, the terms “signal value” or “color value” may be used to describe a luminance level corresponding to the value of a specific color component (such as R, G, B, or Y) for an image element. The signal value is typically representative of a linear light level (luminance value). The terms “code level” or “digital code value” may refer to a digital representation of an image signal value. Typically, such a digital representation is representative of a nonlinear signal value. An EOTF represents the relationship between the nonlinear signal values provided to a display device (e.g., display device 32) and the linear color values produced by the display device.

**[0082]** RGB data is typically utilized as the input color space, since RGB is the type of data that is typically produced by image-capturing sensors. However, the RGB color space has high redundancy among its components and is not optimal for compact representation. To achieve more compact and a more robust representation, RGB components are typically converted (e.g., a color transform is performed) to a more uncorrelated color space that is more suitable for compression, e.g., YCbCr. A YCbCr color space separates the brightness in the form of luminance (Y) and color information (CrCb) in different less correlated components. In this context, a robust representation may refer to a color space featuring higher levels of error resilience when compressed at a constrained bitrate.

**[0083]** Following the color transform, input data in a target color space may be still represented at high bit-depth (e.g. floating point accuracy). The high bit-depth data may be converted to a target bit-depth, for example, using a quantization process. Certain studies show that 10-12 bits accuracy in combination with the PQ transfer is sufficient to provide HDR data of 16 f-stops with distortion below the Just-Noticeable Difference (JND). In general, a JND is the amount of something (e.g., video data) must be change in order for a difference to be noticeable (e.g., by the HVS). Data represented with 10-bit accuracy can be further coded with most of the state-of-the-art video coding

solutions. This quantization is an element of lossy coding and is a source of inaccuracy introduced to converted data.

**[0084]** It is anticipated that next generation HDR/WCG video applications will operate with video data captured at different parameters of HDR and CG. Examples of different configuration can be the capture of HDR video content with peak brightness up-to 1000 nits, or up-to 10,000 nits. Examples of different color gamuts may include BT.709, BT.2020 as well SMPTE specified-P3, or others.

**[0085]** It is also anticipated that a single color space, e.g., a target color container, that incorporates (or nearly incorporates) all other currently used color gamuts to be utilized in future. One example of such a target color container is BT.2020. Support of a single target color container would significantly simplify standardization, implementation and deployment of HDR/WCG systems, since a reduced number of operational points (e.g., number of color containers, color spaces, color conversion algorithms, etc.) and/or a reduced number of required algorithms should be supported by a decoder (e.g., video decoder 30).

**[0086]** In one example of such a system, content captured with a native color gamut (e.g. P3 or BT.709) different from the target color container (e.g. BT.2020) may be converted to the target container prior to processing (e.g., prior to video encoding).

Below are several examples of such conversion:

RGB conversion from BT.709 to BT.2020 color container:

$$\begin{aligned} \circ R_{2020} &= 0.627404078626 * R_{709} + 0.329282097415 * G_{709} + 0.043313797587 * B_{709} \\ \circ G_{2020} &= 0.069097233123 * R_{709} + 0.919541035593 * G_{709} + 0.011361189924 * B_{709} \\ \circ B_{2020} &= 0.016391587664 * R_{709} + 0.088013255546 * G_{709} + 0.895595009604 * B_{709} \end{aligned}$$

(1)

RGB conversion from P3 to BT.2020 color container:

$$\begin{aligned} \circ R_{2020} &= 0.753832826496 * R_{P3} + 0.198597635641 * G_{P3} + 0.047569409186 * B_{P3} \\ \circ G_{2020} &= 0.045744636411 * R_{P3} + 0.941777687331 * G_{P3} + 0.012478735611 * B_{P3} \\ \circ B_{2020} &= -0.001210377285 * R_{P3} + 0.017601107390 * G_{P3} + 0.983608137835 * B_{P3} \end{aligned}$$

(2)

**[0087]** During this conversion, the value range occupied by each component (e.g., RGB, YUV, YCrCb, etc.) of a signal captured in P3 or BT.709 color gamut may be reduced in a BT.2020 representation. Since the data is represented in floating point accuracy, there is no loss; however, when combined with color conversion (e.g., a conversion from RGB to YCrCb shown in equation 3 below) and quantization (example



in equation 4 below), the shrinking of the value range leads to increased quantization error for input data.

$$\circ \quad Y' = 0.2627 * R' + 0.6780 * G' + 0.0593 * B'; \quad Cb = \frac{B' - Y'}{1.8814}; \quad Cr = \frac{R' - Y'}{1.4746}$$

(3)

$$\begin{aligned} \circ \quad D_{Y'} &= \left( \text{Round} \left( (1 \ll (\text{BitDepth}_{Y'} - 8)) * (219 * Y' + 16) \right) \right) \\ \circ \quad D_{Cb} &= \left( \text{Round} \left( (1 \ll (\text{BitDepth}_{Cb} - 8)) * (224 * Cb + 128) \right) \right) \\ \circ \quad D_{Cr} &= \left( \text{Round} \left( (1 \ll (\text{BitDepth}_{Cr} - 8)) * (224 * Cr + 128) \right) \right) \end{aligned}$$

(4)

In equation (4)  $D_{Y'}$  is the quantized  $Y'$  component,  $D_{Cb}$  is the quantized  $Cb$  and  $D_{Cr}$  is the quantized  $Cr$  component. The term  $\ll$  represents a bit-wise right shift.  $\text{BitDepth}_{Y'}$ ,  $\text{BitDepth}_{Cr}$ , and  $\text{BitDepth}_{Cb}$  are the desired bit depths of the quantized components, respectively.

**[0088]** In addition, in a real-world coding system, coding a signal with reduced dynamic range may lead to significant loss of accuracy for coded chroma components and would be observed by a viewer as coding artifacts, e.g., color mismatch and/or color bleeding.

**[0089]** To address the problems described above, the following techniques may be considered. One example technique involves HDR coding at the native color space. In such a technique an HDR video coding system would support various types of currently known color gamuts, and allow extensions of a video coding standard to support future color gamuts. This support would not be only limited to support different color conversion transforms, e.g. RGB to YCbCr, and their inverse transforms, but also would specify transform functions that are adjusted to each of the color gamuts. Support of such variety of tools would complex and expensive.

**[0090]** Another example technique includes a color gamut aware video codec. In such a technique, a hypothetical video encoder is configured to estimate the native color gamut of the input signal and adjust coding parameters (e.g., quantization parameters for coded chroma components) to reduce any distortion resulting from the reduced dynamic range. However, such a technique would not be able to recover loss of accuracy, which may happen due to the quantization conducted in equation (4) above, since all input data is provided to a typical codec in integer point accuracy.

**[0091]** This disclosure describes techniques, methods, and apparatuses to perform a dynamic range adjustment (DRA) to compensate dynamic range changes introduced to HDR signal representations by a color gamut conversion. The dynamic range adjustment may help to prevent and/or lessen any distortion caused by a color gamut conversion, including color mismatch, color bleeding, etc. In one or more examples of the disclosure, DRA is conducted on the values of each color component of the target color space, e.g., YCbCr, prior to quantization at the encoder side (e.g., by source device 12) and after the inverse quantization at the decoder side (e.g., by destination device 14).

**[0092]** FIG. 8 is a block diagram illustrating an example HDR/WCG conversion apparatus operating according to the techniques of this disclosure. In FIG. 8, solid lines specify the data flow and dashed lines specify control signals. The techniques of this disclosure may be performed by video pre-processor unit 19 of source device 12. As discussed above, video pre-processor unit 19 may be a separate device from video encoder 20. In other examples, video pre-processor unit 19 may be incorporated into the same device as video encoder 20.

**[0093]** As shown in FIG. 8, RGB native CG video data 200 is input to video pre-processor unit 19. In the context of video preprocessing by video pre-processor unit 19, RGB native CG video data 200 is defined by an input color container. The input color container specifies set of color primaries used to represent video data 200 (e.g., BT. 709, BT. 2020, P3, etc.). In one example of the disclosure, video pre-processor unit 19 may be configured to convert both the color container and the color space of RGB native CB video data 200 to a target color container and target color space for HDR' data 216. Like the input color container, the target color container may specify a set or color primaries used to represent the HDR' data 216. In one example of the disclosure, RGB native CB video data 200 may be HDR/WCG video, and may have a BT.2020 or P3 color container (or any WCG), and be in an RGB color space. In another example, RGB native CB video data 200 may be SDR video, and may have a BT.709 color container. In one example, the target color container for HDR' data 216 may have been configured for HDR/WCG video (e.g., BT.2020 color container) and may use a color space more optimal for video encoding (e.g., YCrCb).

**[0094]** In one example of the disclosure, CG converter 202 may be configured to convert the color container of RGB native CG video data 200 from the input color container (e.g., first color container) to the target color container (e.g., second color

container). As one example, CG converter 202 may convert RGB native CG video data 200 from a BT.709 color representation to a BT.2020 color representation, example of which is shown below.

**[0095]** The process to convert RGB BT.709 samples ( $R_{709}$ ,  $G_{709}$ ,  $B_{709}$ ) to RGB BT.2020 samples ( $R_{2020}$ ,  $G_{2020}$ ,  $B_{2020}$ ) can be implemented with a two-step conversion that involves converting first to the XYZ representation, followed by a conversion from XYZ to RGB BT.2020 using the appropriate conversion matrices.

$$\begin{aligned} X &= 0.412391 * R_{709} + 0.357584 * G_{709} + 0.180481 * B_{709} \\ Y &= 0.212639 * R_{709} + 0.715169 * G_{709} + 0.072192 * B_{709} \\ Z &= 0.019331 * R_{709} + 0.119195 * G_{709} + 0.950532 * B_{709} \end{aligned} \quad (5)$$

**[0096]** Conversion from XYZ to  $R_{2020}G_{2020}B_{2020}$  (BT.2020)

$$\begin{aligned} R_{2020} &= \text{clipRGB}(1.716651 * X - 0.355671 * Y - 0.253366 * Z) \\ G_{2020} &= \text{clipRGB}(-0.666684 * X + 1.616481 * Y + 0.015768 * Z) \\ B_{2020} &= \text{clipRGB}(0.017640 * X - 0.042771 * Y + 0.942103 * Z) \end{aligned} \quad (6)$$

Similarly, the single step and recommended method is as follows:

$$\begin{aligned} R_{2020} &= \text{clipRGB}(0.627404078626 * R_{709} + 0.329282097415 * G_{709} + \\ &\quad 0.043313797587 * B_{709}) \\ G_{2020} &= \text{clipRGB}(0.069097233123 * R_{709} + 0.919541035593 * G_{709} + \\ &\quad 0.011361189924 * B_{709}) \\ B_{2020} &= \text{clipRGB}(0.016391587664 * R_{709} + 0.088013255546 * G_{709} + \\ &\quad 0.895595009604 * B_{709}) \end{aligned} \quad (7)$$

**[0097]** The resulting video data after CG conversion is shown as RGB target CG video data 204 in FIG. 8. In other examples of the disclosure, the color container for the input data and the output HDR' data may be the same. In such an example, CG converter 202 need not perform any conversion on RGB native CG video data 200.

**[0098]** Next, transfer function unit 206 compacts the dynamic range of RGB target CG video data 204. Transfer function unit 206 may be configured to apply a transfer function to compact the dynamic range in the same manner as discussed above with reference to FIG. 5. The color conversion unit 208 converts RGB target CG color data 204 from the color space of the input color container (e.g., RGB) to the color space of the target color container (e.g., YCrCb). As explained above with reference to FIG. 5, color conversion unit 208 converts the compacted data into a more compact or robust color space (e.g., a YUV or YCrCb color space) that is more suitable for compression by a hybrid video encoder (e.g., video encoder 20).

**[0099]** Adjustment unit 210 is configured to perform a dynamic range adjustment (DRA) of the color converted video data in accordance with DRA parameters derived by DRA parameters estimation unit 212. In general, after CG conversion by CG converter 202 and dynamic range compaction by transfer function unit 206, the actual color values of the resulting video data may not use all available codewords (e.g., unique bit sequences that represent each color) allocated for the color gamut of a particular target color container. That is, in some circumstances, the conversion of RGB native CG video data 200 from an input color container to an output color container may overly compact the color values (e.g., Cr and Cb) of the video data such that the resultant compacted video data does not make efficient use of all possible color representations. As explained above, coding a signal with a reduced range of values for the colors may lead to a significant loss of accuracy for coded chroma components and would be observed by a viewer as coding artifacts, e.g., color mismatch and/or color bleeding.

**[0100]** Adjustment unit 210 may be configured to apply DRA parameters to the color components (e.g., YCrCb) of the video data, e.g., RGB target CG video data 204 after dynamic range compaction and color conversion to make full use of the codewords available for a particular target color container. Adjustment unit 210 may apply the DRA parameter to the video data at a pixel level. In general, the DRA parameters define a function that expands the codewords used to represent the actual video data to as many of the codewords available for the target color container as possible.

**[0101]** In one example of the disclosure, the DRA parameters include a scale and offset value that is applied to the components of the video data. In general, the lower the value range of the color components of the video data, the larger a scaling factor may be used. The offset parameter may be used to center the values of the color components to the center of the available codewords for a target color container. For example, if a target color container includes 1024 codewords per color component, an offset value may be chosen such that the center codeword is moved to codeword 512 (e.g., the middle most codeword). In other examples, the offset parameter may be used to provide better mapping of input codewords to output codewords such that overall representation in the target color container is more efficient in combating coding artefacts.

**[0102]** In one example, adjustment unit 210 applies DRA parameters to video data in the target color space (e.g., YCrCb) as follows:

$$- \quad Y'' = \text{scale1} * Y' + \text{offset1}$$

$$\begin{aligned}
 - \quad Cb'' &= scale2 * Cb' + offset2 \\
 - \quad Cr'' &= scale3 * Cr' + offset3
 \end{aligned}
 \tag{8}$$

where signal components  $Y'$ ,  $Cb'$  and  $Cr'$  is a signal produced from RGB to YCbCr conversion (example in equation 3). Note that  $Y'$ ,  $Cb'$  and  $Cr'$  may also be a video signal decoded by video decoder 30.  $Y''$ ,  $Cb''$ , and  $Cr''$  are the color components of the video signal after the DRA parameters have been applied to each color component. As can be seen in the example above, each color component is related to different scale and offset parameters. For example, scale1 and offset1 are used for the  $Y'$  component, scale2 and offset2 are used for the  $Cb'$  component, and scale3 and offset3 are used for the  $Cr'$  component. It should be understood that this is just an example. In other examples, the same scale and offset values may be used for every color component.

**[0103]** In other examples, each color component may be associated with multiple scale and offset parameters. For example, the actual distribution of chroma values for the  $Cr$  or  $Cb$  color components may differ for different partitions or ranges of codewords. As one example, there may be more unique codewords used above the center codeword (e.g., codeword 512) than there are below the center codeword. In such an example, adjustment unit 210 may be configured to apply one set of scale and offset parameters for chroma values above the center codeword (e.g., having values greater than the center codeword) and apply a different set of scale and offset parameters for chroma values below the center codeword (e.g., having values less than the center codeword).

**[0104]** As can be seen in the above example, adjustment unit 210 applies the scale and offset DRA parameters as a linear function. As such, it is not necessary for adjustment unit 210 to apply the DRA parameters in the target color space after color conversion by color conversion unit 208. This is because color conversion is itself a linear process. As such, in other examples, adjustment unit 210 may apply the DRA parameters to the video data in the native color space (e.g., RGB) before any color conversion process. In this example, color conversion unit 208 would apply color conversion after adjustment unit 210 applies the DRA parameters.

**[0105]** In another example of the disclosure, adjustment unit 210 may apply the DRA parameters in either the target color space or the native color space as follows:

$$\begin{aligned}
 - \quad Y'' &= (scale1 * (Y' - offsetY) + offset1) + offsetY; \\
 - \quad Cb'' &= scale2 * Cb' + offset2 \\
 - \quad Cr'' &= scale3 * Cr' + offset3
 \end{aligned}
 \tag{9}$$

In this example, the parameter scale1, scale2, scale3, offset1, offset2, and offset3 have the same meaning as described above. The parameter offsetY is a parameter reflecting brightness of the signal, and can be equal to the mean value of Y'. In other examples, an offset parameter similar to offsetY may be applied for the Cb' and Cr' components to better preserve the mapping of the center value in the input and the output representations.

**[0106]** In another example of the disclosure, adjustment unit 210 may be configured to apply the DRA parameters in a color space other than the native color space or the target color space. In general, adjustment unit 210 may be configured to apply the DRA parameters as follows:

$$\begin{aligned}
 - \quad A' &= \text{scale1} * A + \text{offset1}; \\
 - \quad B' &= \text{scale2} * B + \text{offset2} \\
 - \quad C' &= \text{scale3} * C + \text{offset3}
 \end{aligned}
 \tag{10}$$

where signal components A, B and C are signal components in a color space which is different from target color space, e.g., RGB or an intermediate color space.

**[0107]** In other examples of the disclosure, adjustment unit 210 is configured to apply a linear transfer function to the video to perform DRA. Such a transfer function is different from the transfer function used by transfer function unit 206 to compact the dynamic range. Similar to the scale and offset terms defined above, the transfer function applied by adjustment unit 210 may be used to expand and center the color values to the available codewords in a target color container. An example of applying a transfer function to perform DRA is shown below:

$$\begin{aligned}
 - \quad Y'' &= \text{TF2} (Y') \\
 - \quad Cb'' &= \text{TF2} (Cb') \\
 - \quad Cr'' &= \text{TF2} (Cr')
 \end{aligned}$$

Term TF2 specifies the transfer function applied by adjustment unit 210. In some examples, adjustment unit 210 may be configured to apply different transfer functions to each of the components.

**[0108]** In another example of the disclosure, adjustment unit 210 may be configured to apply the DRA parameters jointly with the color conversion of color conversion unit 208 in a single process. That is, the linear functions of adjustment unit 210 and color conversion unit 208 may be combined. An example of a combined application, where f1 and f2 are a combination of the RGB to YCbCr matrix and the DRA scaling factors, is shown below:

$$Cb = \frac{B' - Y'}{f1} ; Cr = \frac{R' - Y'}{f2}$$

**[0109]** In another example of the disclosure, after applying the DRA parameters, adjustment unit 210 may be configured to perform a clipping process to prevent the video data from having values outside the range of codewords specified for a certain target color container. In some circumstances, the scale and offset parameters applied by adjustment unit 210 may cause some color component values to exceed the range of allowable codewords. In this case, adjustment unit 210 may be configured to clip the values of the components that exceed the range to the maximum value in the range.

**[0110]** The DRA parameters applied by adjustment unit 210 may be determined by DRA parameters estimation unit 212. The frequency and the time instances at which the DRA parameters estimation unit 212 updates the DRA parameters are flexible. For example, DRA parameters estimation unit 212 may update the DRA parameters on a temporal level. That is, new DRA parameters may be determined for a group of pictures (GOP), or a single picture (frame). In this example, the RGB native CG video data 200 may be a GOP or a single picture. In other examples, DRA parameters estimation unit 212 may update the DRA parameters on a spatial level, e.g., at the slice tile, or block level. In this context, a block of video data may be a macroblock, coding tree unit (CTU), coding unit, or any other size and shape of block. A block may be square, rectangular, or any other shape. Accordingly, the DRA parameters may be used for more efficient temporal and spatial prediction and coding.

**[0111]** In one example of the disclosure, DRA parameters estimation unit 212 may derive the DRA parameters based on the correspondence of the native color gamut of RGB native CG video data 200 and the color gamut of the target color container. For example, DRA parameters estimation unit 212 may use a set of predefined rules to determine scale and offset values given a certain native color gamut (e.g., BT.709) and the color gamut of a target color container (e.g., BT.2020).

**[0112]** For example, assume that native color gamut and target color container are defined in the form of color primaries coordinates in xy space and white point coordinates. One example of such information for BT.709 and BT.2020 is shown in Table 2 below.

**Table 2- RGB color space parameters**

RGB color space parameters								
Color space	White point		Primary colors					
	xx <sub>w</sub>	yy <sub>w</sub>	xx <sub>R</sub>	yy <sub>R</sub>	xx <sub>G</sub>	yy <sub>G</sub>	xx <sub>B</sub>	yy <sub>B</sub>
DCI-P3	0.314	0.351	0.680	0.320	0.265	0.690	0.150	0.060
ITU-R BT.709	0.3127	0.3290	0.64	0.33	0.30	0.60	0.15	0.06
ITU-R BT.2020	0.3127	0.3290	0.708	0.292	0.170	0.797	0.131	0.046

**[0113]** In one example, BT.2020 is the color gamut of the target color container and BT.709 is the color gamut of the native color container. In this example, adjustment unit 210 applies the DRA parameters to the YCbCr target color space. DRA parameters estimation unit 212 may be configured to estimate and forward the DRA parameters to adjustment unit 210 as follows:

$$\begin{aligned} \text{scale1} &= 1; & \text{offset1} &= 0; \\ \text{scale2} &= 1.0698; & \text{offset2} &= 0; \\ \text{scale3} &= 2.1735; & \text{offset3} &= 0; \end{aligned}$$

**[0114]** As another example, with BT.2020 being a target color gamut and P3 being a native color gamut, and DRA being applied in YCbCr target color space, DRA parameters estimation unit 212 may be configured to estimate the DRA parameters as:

$$\begin{aligned} \text{scale1} &= 1; & \text{offset1} &= 0; \\ \text{scale2} &= 1.0068; & \text{offset2} &= 0; \\ \text{scale3} &= 1.7913; & \text{offset3} &= 0; \end{aligned}$$

**[0115]** In the examples above, DRA parameters estimation unit 212 may be configured to determine the above-listed scale and offset values by consulting a lookup table that indicates the DRA parameters to use, given a certain native color gamut and a certain target color gamut. In other examples, DRA parameters estimation unit 212 may be configured to calculate the DRA parameters from the primary and white space values of the native color gamut and target color gamut, e.g., as shown in Table 2.

**[0116]** For example, consider a target (T) color container specified by primary coordinates (x<sub>Xt</sub>, y<sub>Xt</sub>), where X stated for R,G,B color components:

$$\text{prime}T = \begin{bmatrix} x_{Rt} & y_{Rt} \\ x_{Gt} & y_{Gt} \\ x_{Bt} & y_{Bt} \end{bmatrix}$$



and native (N) color gamut specified by primaries coordinates ( $x_{Xn}$ ,  $y_{Xn}$ ), where X stated for R,G,B color components:

$$primeN = \begin{bmatrix} x_{Rn} & y_{Rn} \\ x_{Gn} & y_{Gn} \\ x_{Bn} & y_{Bn} \end{bmatrix}$$

The white point coordinate for both gamuts equals  $whiteP = (x_W, y_W)$ . DRA parameters estimation unit 212 may derive the scale2 and scale3 parameters for DRA as a function of the distances between primaries coordinates to the white point. One example of such an estimation is given below:

$$rdT = \sqrt{(primeT(1,1) - whiteP(1,1))^2 + (primeN(1,2) - whiteP(1,2))^2}$$

$$gdT = \sqrt{(primeT(2,1) - whiteP(1,1))^2 + (primeN(2,2) - whiteP(1,2))^2}$$

$$bdT = \sqrt{(primeT(3,1) - whiteP(1,1))^2 + (primeN(3,2) - whiteP(1,2))^2}$$

$$rdN = \sqrt{(primeN(1,1) - whiteP(1,1))^2 + (primeN(1,2) - whiteP(1,2))^2}$$

$$gdN = \sqrt{(primeN(2,1) - whiteP(1,1))^2 + (primeN(2,2) - whiteP(1,2))^2}$$

$$bdN = \sqrt{(primeN(3,1) - whiteP(1,1))^2 + (primeN(3,2) - whiteP(1,2))^2}$$

$$scale2 = bdT/bdN$$

$$scale3 = \sqrt{(rdT/rdN)^2 + (gdT/gdN)^2}$$

**[0117]** In some examples, DRA parameters estimation unit 212 may be configured to estimate the DRA parameters by determining the primaries coordinates in  $primeN$  from the actual distribution of color values in RGB native CG video data 200, and not from the pre-defined primary values of the native color gamut. That is, DRA parameters estimation unit 212 may be configured to analyze the actual colors present in RGB native CG video data 200, and use the primary color values and white point determined from such an analysis in the function described above to calculate DRA parameters. Approximation of some parameters defined above might be used as DRA to facilitate the computation. For instance,  $scale3 = 2.1735$  can be approximated to  $scale3 = 2$ , which allows for easier implementation in some architectures.

**[0118]** In other examples of the disclosure, DRA parameters estimation unit 212 may be configured to determine the DRA parameters based not only on the color gamut of the target color container, but also on the target color space. The actual distributions of values of component values may differ from color space to color space. For example, the chroma value distributions may be different for YCbCr color spaces having a constant luminance as compared to YCbCr color spaces having a non-constant

luminance. DRA parameters estimation unit 212 may use the color distributions of different color spaces to determine the DRA parameters.

**[0119]** In other examples of the disclosure, DRA parameters estimation unit 212 may be configured to derive values for DRA parameters so as to minimize certain cost functions associated with pre-processing and/or encoding video data. As one example, DRA parameters estimation unit 212 may be configured to estimate DRA parameters that minimized quantization errors introduced by quantization unit 214 (e.g., see equation (4)) above. DRA parameters estimation unit 212 may minimize such an error by performing quantization error tests on video data that has had different sets of DRA parameters applied. In another example, DRA parameters estimation unit 212 may be configured to estimate DRA parameters that minimize the quantization errors introduced by quantization unit 214 in a perceptual manner. DRA parameters estimation unit 212 may minimize such an error based on perceptual error tests on video data that has had different sets of DRA parameters applied. DRA parameters estimation unit 212 may then select the DRA parameters that produced the lowest quantization error.

**[0120]** In another example, DRA parameters estimation unit 212 may select DRA parameters that minimize a cost function associated with both the DRA performed by adjustment unit 210 and the video encoding performed by video encoder 20. For example, DRA parameters estimation unit 212 may perform DRA and encode the video data with multiple different sets of DRA parameters. DRA parameters estimation unit 212 may then calculate a cost function for each set of DRA parameters by forming a weighted sum of the bitrate resulting from DRA and video encoding, as well as the distortion introduced by these two lossy process. DRA parameters estimation unit 212 may then select the set of DRA parameters that minimizes the cost function.

**[0121]** In each of the above techniques for DRA parameter estimation, DRA parameters estimation unit 212 may determine the DRA parameters separately for each component using information regarding that component. In other examples, DRA parameters estimation unit 212 may determine the DRA parameters using cross-component information. For example, the DRA parameters derived for a Cr component may be used to derive DRA parameters for a Cb component.

**[0122]** In addition to deriving DRA parameters, DRA parameters estimation unit 212 may be configured to signal the DRA parameters in an encoded bitstream. DRA parameters estimation unit 212 may signal one or more syntax elements that indicate the DRA parameters directly, or may be configured to provide the one or more syntax

elements to video encoder 20 for signaling. Such syntax elements of the parameters may be signaled in the bitstream such that video decoder 30 and/or video post-processor unit 31 may perform the inverse of the process of video pre-processor unit 19 to reconstruct the video data in its native color container. Example techniques for signaling the DRA parameters are discussed below.

**[0123]** In one example, DRA parameters estimation unit 212 may signal one or more syntax elements that in an encoded video bitstream as metadata, in a supplemental enhancement information (SEI) message, in video usability information (VUI), in a video parameter set (VPS), in a sequence parameter set (SPS), in a picture parameter set, in a slice header, in a CTU header, or in any other syntax structure suitable for indicating the DRA parameters for the size of the video data (e.g., GOP, pictures, blocks, macroblock, CTUs, etc.).

**[0124]** In some examples, the one or more syntax elements indicate the DRA parameters explicitly. For example, the one or more syntax elements may be the various scale and offset values for DRA. In other examples, the one or more syntax elements may be one or more indices into a lookup table that includes the scale and offset values for DRA. In still another example, the one or more syntax elements may be indices into a lookup table that specifies the linear transfer function to use for DRA.

**[0125]** In other examples, the DRA parameters are not signaled explicitly, but rather, both video pre-processor unit 19 and video post-processor unit 31 are configured to derive the DRA parameters using the same pre-defined process using the same information and/or characteristics of the video data that are discernible from the bitstream. As one example, video post-processor unit 31 may be configured to indicate the native color container of the video data as well as the target color container of the encoded video data in the encoded bitstream. Video post-processor unit 31 may then be configured to derive the DRA parameters from such information using the same process as defined above. In some examples, one or more syntax elements that identify the native and target color containers are supplied in a syntax structure. Such syntax elements may indicate the color containers explicitly, or may be indices to a lookup table. In another example, video pre-processor unit 19 may be configured to signal one or more syntax elements that indicate the XY values of the color primaries and the white point for a particular color container. In another example, video pre-processor unit 19 may be configured to signal one or more syntax elements that indicate the XY values of the color primaries and the white point of the actual color values (content primaries and

content white point) in the video data based on an analysis performed by DRA parameters estimation unit 212.

**[0126]** As one example, the color primaries of the smallest color gamut containing the color in the content might be signaled, and at video decoder 30 and/or video post-processor unit 31, the DRA parameters are derived using both the container primaries and the content primaries. In one example, the content primaries can be signaled using the x and y components for R, G and B, as described above. In another example, the content primaries can be signaled as the ratio between two known primary sets. For example, the content primaries can be signaled as the linear position between the BT.709 primaries and the BT.2020 primaries:  $x_{r\_content} = \alpha_{r\_r} * x_{r\_bt709} + (1 - \alpha_{r\_r}) * x_{r\_bt2020}$  (with similar equation with  $\alpha_{g\_r}$  and  $\alpha_{b\_r}$  for the G and B components), where parameter  $\alpha_{r\_r}$  specifies a ratio between two known primary sets. In some examples, the signaled and/or derived DRA parameters may be used by video encoder 20 and/or video decoder 30 to facilitate weighted prediction based techniques utilized for coding of HDR/WCG video data.

**[0127]** In video coding schemes utilizing weighted prediction, a sample of currently coded picture  $S_c$  are predicted from a sample (for single directional prediction) of the reference picture  $S_r$  taken with a weight ( $W_{wp}$ ) and an offset ( $O_{wp}$ ) which results in predicted sample  $S_p$ :

$$S_p = S_r * W_{wp} + O_{wp}.$$

**[0128]** In some examples utilizing DRA, samples of the reference and currently coded picture can be processed with DRA employing different parameters, namely {  $scale1_{cur}$ ,  $offset1_{cur}$  } for a current picture and {  $scale1_{ref}$ ,  $offset1_{ref}$  } for a reference picture. In such embodiments, parameters of weighted prediction can be derived from DRA, e.g.:

$$W_{wp} = scale1_{cur} / scale1_{ref}$$

$$O_{wp} = offset1_{cur} - offset1_{ref}$$

**[0129]** After adjustment unit 210 applies the DRA parameters, video pre-processor unit 19 may then quantize the video data using quantization unit 214. Quantization unit 214 may operate in the same manner as described above with reference to FIG. 4. After quantization, the video data is now adjusted in the target color space and target color gamut of the target primaries of HDR' data 216. HDR' data 216 may then be sent to video encoder 20 for compression.

**[0130]** FIG. 9 is a block diagram illustrating an example HDR/WCG inverse conversion apparatus according to the techniques of this disclosure. As shown in FIG. 9, video

post-processor unit 31 may be configured to apply the inverse of the techniques performed by video pre-processor unit 19 of FIG. 8. In other examples, the techniques of video post-processor unit 31 may be incorporated in, and performed by, video decoder 30.

**[0131]** In one example, video decoder 30 may be configured to decode the video data encoded by video encoder 20. The decoded video data (HDR' data 316 in the target color container) is then forwarded to video post-processor unit 31. Inverse quantization unit 314 performs an inverse quantization process on HDR' data 316 to reverse the quantization process performed by quantization unit 214 of FIG. 8.

**[0132]** Video decoder 30 may also be configured to decode and send any of the one or more syntax elements produced by DRA parameters estimation unit 212 of FIG. 8 to DRA parameters derivation unit 312 of video post-processor unit 31. DRA parameters derivation unit 312 may be configured to determine the DRA parameters based on the one or more syntax elements, as described above. In some examples, the one or more syntax elements indicate the DRA parameters explicitly. In other examples, DRA parameters derivation unit 312 is configured to derive the DRA parameters using the same techniques used by DRA parameters estimation unit 212 of FIG. 8.

**[0133]** The parameters derived by DRA parameters derivation unit 312 are sent to inverse adjustment unit 310. Inverse adjustment unit 310 uses the DRA parameters to perform the inverse of the linear DRA adjustment performed by adjustment unit 210. Inverse adjustment unit 310 may apply the inverse of any of the adjustment techniques described above for adjustment unit 210. In addition, as with adjustment unit 210, inverse adjustment unit 310 may apply the inverse DRA before or after any inverse color conversion. As such, inverse adjustment unit 310 may apply the DRA parameter on the video data in the target color container or the native color container. In some examples, inverse adjustment unit 310 may be positioned to apply inverse adjustment before inverse quantization unit 314.

**[0134]** Inverse color conversion unit 308 converts the video data from the target color space (e.g., YCbCr) to the native color space (e.g., RGB). Inverse transfer function 306 then applies an inverse of the transfer function applied by transfer function 206 to uncompact the dynamic range of the video data. In some examples, the resulting video data (RGB target CG 304) is still in the target color gamut, but is now in the native dynamic range and native color space. Next, inverse CG converter 302 converts RGB target CG 304 to the native color gamut to reconstruct RGB native CG 300.

**[0135]** In some examples, additional post-processing techniques may be employed by video post-processor unit 31. Applying the DRA may put the video outside its actual native color gamut. The quantization steps performed by quantization unit 214 and inverse quantization unit 314, as well as the up and down-sampling techniques performed by adjustment unit 210 and inverse adjustment unit 310, may contribute to the resultant color values in the native color container being outside the native color gamut. When the native color gamut is known (or the actual smallest content primaries, if signaled, as described above), then additional process can be applied to RGB native CG video data 304 to transform color values (e.g., RGB or Cb and Cr) back into the intended gamut as post-processing for DRA. In other examples, such post-processing may be applied after the quantization or after DRA application.

**[0136]** As mentioned above, several SEI messages may be used to convey the information regarding dynamic range adjustment information for the various color components of the video data. The component scaling SEI message, such as described above and in more detail below, may convey a set of scale factors, offsets, and ranges (e.g., partitions of codeword values) that can be used to indicate the mapping information for the various color components of the video data. The mapping information may be used to indicate to video decoder 30 and/or video post-processor unit 31 how to expand or shrink the different ranges of sample values in such a way that the overall quality of the reconstructed HDR video data, or also quality of reconstructed SDR video data in some cases where backward compatibility is desired, is improved, or to make the reconstructed output more suitable for display capabilities.

**[0137]** Table 3 below provides one variation of the syntax structure of a component scaling SEI message. Note that although the names of the syntax elements below contain the prefix “hdr\_recon\_” that is different from that described in the examples below, where the names of the syntax elements are prefixed as component\_scaling, the syntax table is otherwise the same.

**[0138] Table 3 – Example Range Adjustment SEI syntax**

hdr_reconstruction_info( payloadSize ) {	Descriptor
<b>hdr_recon_id</b>	ue(v)
<b>hdr_recon_cancel_flag</b>	u(1)
if( !hdr_recon_cancel_flag ) {	
<b>hdr_recon_persistence_flag</b>	u(1)
<b>hdr_recon_transfer_characteristics</b>	u(8)
<b>hdr_recon_default_flag</b>	u(1)
if( !hdr_recon_default_flag ) {	
<b>hdr_recon_scale_bit_depth</b>	u(4)
<b>hdr_recon_offset_bit_depth</b>	u(4)
<b>hdr_recon_scale_frac_bit_depth</b>	u(4)
<b>hdr_recon_offset_frac_bit_depth</b>	u(4)
<b>hdr_recon_num_comps_minus1</b>	ue(v)
}	
for( c = 0; c <= hdr_recon_num_comps_minus1; c++ ) {	
<b>hdr_recon_num_ranges[ c ]</b>	ue(v)
<b>hdr_recon_equal_ranges_flag[ c ]</b>	u(1)
<b>hdr_recon_global_offset_val[ c ]</b>	u(v)
for( i = 0; i <= hdr_recon_num_ranges[ c ]; i++ )	
<b>hdr_recon_scale_val[ c ][ i ]</b>	u(v)
if( !hdr_recon_equal_ranges[ c ] )	
for( i = 0; i <= hdr_recon_num_ranges[ c ]; i++ )	
<b>hdr_recon_range_val [ c ][ i ]</b>	u(v)
}	
}	
}	

[0139] The semantics of the SEI syntax of Table 3 is presented below.

[0140] The mapping process is based on piece-wise linear functions  $\text{map}[c](x)$ , for  $c = 0.. \text{hdr\_recon\_num\_comps\_minus1}$ , that map a value  $x$  in  $[0,1]$  to a value  $y = \text{map}[c](x)$  as follows:

- For  $i$  in the range of 0 to  $\text{hdr\_recon\_num\_ranges}[c] - 1$ , inclusive, the following applies:

- The value  $\text{ScaleValue}[c][i]$  is derived as described in semantics of syntax element  $\text{hdr\_recon\_scale\_val}[c][i]$ .

- The value  $\text{RangeValue}[c][i]$  is derived as described in semantics of syntax element  $\text{hdr\_recon\_range\_val}[c][i]$ .

- The values  $\text{InputRanges}[c][i]$  and  $\text{OutputRanges}[c][i]$ , for  $i$  in the range of 0 to  $\text{hdr\_recon\_num\_ranges}[c] - 1$ , inclusive, are derived as follows:

- If  $i$  is equal to 0, the following applies:

$\text{OutputRanges}[c][i] = - \text{hdr\_recon\_global\_offset\_val}[c] * \text{ScaleValue}[c][i - 1]$  (D-xx)

$\text{InputRanges}[c][i] = 0$  (D-xx)

- Otherwise ( $i$  is not equal to 0), the following applies:

$\text{InputRanges}[c][i] = \text{InputRanges}[c][i - 1] + \text{RangeValue}[c][i - 1]$  (D-xx)

$\text{OutputRanges}[c][i] = \text{OutputRanges}[c][i - 1] +$

$\text{RangeValue}[c][i - 1] * \text{ScaleValue}[c][i - 1]$  (D-xx)

- The values  $\text{OffsetValue}[c][i]$ , for  $i$  in the range of 0 to  $\text{hdr\_recon\_num\_ranges}[c] - 1$ , inclusive, are derived as follows:

$\text{OffsetValue}[c][i] = \text{InputRanges}[c][i + 1] - \text{OutputRanges}[c][i + 1]$  □

$\text{ScaleValue}[c][i - 1]$  (D-xx)

- The parameter  $y = \text{map}[c](x)$  is derived as follows:

- If  $x$  is lower than or equal to  $\text{OutputRanges}[c][0]$ , the following applies:

$y = \text{InputRanges}[c][0]$  (D-xx)

- Otherwise if  $x$  is larger than  $\text{OutputRanges}[c][\text{hdr\_recon\_num\_ranges}[c]]$ , the following applies:

$y = \text{InputRanges}[c][\text{hdr\_recon\_num\_ranges}[c]]$  (D-xx)

- Otherwise, the following applies:

for( $i = 1; i \leq \text{hdr\_recon\_num\_ranges}[c]; i++$ )

if( $\text{OutputRanges}[i - 1] < x \ \&\& \ x \leq \text{OutputRanges}[i]$ )

$y = x \div \text{ScaleValue}[c][i - 1] + \text{OffsetValue}[c][i - 1]$  (D-xx)



**[0141]** Several problems have been identified that are associated with the component scaling information SEI messages, and other parameters that are applicable to adjust the dynamic range of components. In particular, problems have been identified related to the use of floating point numbers to derive scale and offset values, as well as the ranges of codewords for sample values (e.g., RGB values, YCrCb values, YUV values, XYZ values, etc.). For example, the scale values that are signalled in the bitstream are used at the decoder side, for example, by video post-processor 31, to perform an inverse dynamic range adjustment process. However, in order to use the scale values for computing the ranges of sample values, and for computing the mapping process, a reciprocal operation is performed at video post-processor 31. Previous example semantics for a component scaling SEI message specify the use of the reciprocal (e.g., the reciprocal of a scale value, or reciprocal of a scale value and an added offset value) to be multiplied with sample values. Errors introduced in such a reciprocal operation would be more significant than potential errors in a forward operation, as the reciprocal is applied to every sample value generated.

**[0142]** The semantics of the component scaling SEI message indicates the derivation process of the ranges of sample values, and the mapping process (e.g., the application of scale and offset values) to each range of codewords of color components (e.g., sample values) in terms of floating point operations. This could lead to differences in the reconstructed HDR output based on the various floating point arithmetic implementations in different computing systems.

**[0143]** This application describes several techniques to improve the communication of component scaling information using SEI signaling and processing, or other similar signaling techniques which may be specified in video coding standards, such as H.265/HEVC, H.264/AVC, BDA, MPEG or others. It is to be recognized that one or more of the following aspects may be applied independently, or in suitable combination with others of these aspects in any particular example.

**[0144]** In general, this disclosure describes techniques wherein video encoder 20 and/or video pre-processor unit 19 may be configured to signal a scale value for one or more sample value ranges of a component sample values (e.g., color component values). The scale value is specified such that video decoder 30 and video post-processor unit 31 may be configured to perform a mapping process to obtain an output sample value from the input sample value of the component by multiplying the scale value specified for a specific sample value range containing the input sample value with the input sample

value and adding an offset computed based on the parameters as part of the component scaling information.

**[0145]** In another example of the disclosure, rather than using a floating point implementation to compute the size and number of ranges of codewords of a color component, video encoder 20 and/or video pre-processor unit 19 may be configured to derive the size and number of ranges of codewords of the color component using a fixed-point computing implementation. For example, video encoder 20 and/or video pre-processor unit 19 may be configured to use a predetermined number of fractional bits for determining and applying the parameters of the dynamic range adjustment mapping process. Note that the number of fractional bits may be different for each parameter (e.g., range of values for each color component (codeword), scale value, and offset value) of the dynamic range adjustment process.

**[0146]** For example, video pre-processor unit 19 may be configured to perform integer operations on any parameters or syntax element (e.g. `hdr_recon_num_ranges[ c ]`) used to communicate the size and number of ranges of codewords for a color component. Video pre-processor unit 19 may be configured to keep track of the number of bits used by the fractional part of any calculation of the size and number of ranges in the fixed-point implementation used. Video pre-processor unit 19 and/or video encoder 20 may be configured to signal the number of bits used in the fractional part in an SEI message (e.g., `hdr_recon_offset_frac_bit_depth`, `hdr_recon_scale_frac_bit_depth`), or the number of bits used in the fractional part may be a pre-determined value. Video decoder 30 may be configured to decode the syntax elements in the SEI message indicating the number of bits in the fractional part and video post-processor unit 31 may be configured to perform an inverse dynamic range adjustment using the same number of bits in the fractional part for one or more of the parameters of the inverse dynamic range adjustment process.

**[0147]** In one example of the disclosure, when determining the ranges and/or other parameters for the mapping process, video decoder 30 and/or video post-processor unit 31 may be configured to determine such parameters so that, when the signaled fractional bit depths of different parameters are different, the accuracy of the computations performed for the parameters are retained as far as possible. For example, video decoder 30 and/or video post-processor unit 31 may be configured to retain any errors introduced due to rounding to a minimum by accumulating the number of fractional bits in any intermediate calculation steps used to determine a particular parameter. Video

decoder 30 and/or video post-processor unit 31 may then perform a clipping process to bring the final value of a particular parameter to the desired fractional accuracy at the last step of determining and/or calculating a particular parameter. In another example, when the signaled fractional bit depths of the parameters are the same, video decoder 30 and/or video post-processor unit 31 may be configured to accumulate the number of fractional bits in the intermediate steps, and perform clipping to bring the final value of a parameter to the desired accuracy at the last step(s).

**[0148]** In another example, video decoder 30 and/or video post-processor unit 31 may be configured to clip and/or truncate the value of a parameter at one or more intermediate steps of a calculation process or the parameter such that the fractional parts of values obtained for the parameter are reduced to a pre-determined value. That is, rather than waiting until determining a final value for the parameter to perform any clipping, video decoder 30 and/or video post-processor unit 31 may clip intermediate values of calculations performed to determine the parameter. Such clipping or truncation may be based on the number of fractional bits indicated in the SEI message. In another example, video decoder 30 and/or video post-processor unit 31 may be configured to clip and/or truncate intermediate values used when calculating a particular parameter before a particular operation/step when it is determined that, if the operation/step is performed without clipping, the accumulated number of fractional bits would exceed a certain pre-determined value, e.g. the bit depth of the registers used to store the intermediate values.

**[0149]** In another example of the disclosure, video pre-processor unit 19 and/or video post-processor unit 31 may be configured to derive scale, offset and range values according to predetermined sample value ranges based on a defined minimum and maximum values defined for the fixed representation of the color components. For example, a fixed representation of color components may have a plurality of ranges of values defined, e.g., a “standard” range of values, a “full” range of values, and a “restricted” range values. The “full” range of values may have a larger span between the minimum and maximum value of a particular component (e.g., for an 8-bit full-range representation of YCbCr color space, the Y, Cb, and Cr components can take values in the range of 0 to 255, inclusive) as compared to the “standard” range of values (e.g., an 8-bit standard range representation of YCbCr color space, the Y component may take values in the range of 16 to 235, inclusive, and the Cb and Cr components may take values between 16 and 240, inclusive). The “restricted” range of values may

have a smaller span between the minimum and maximum value of a particular component (e.g., for a 10-bit restricted-range representation of YCbCr color space, the Y, Cb, and Cr components may take values in the range of 4 to 1019, inclusive) as compared to the “standard” range of values.

**[0150]** In one example, video encoder 20 and/or video pre-processor unit 19 may be configured to signal a syntax element (e.g., in an SEI message) to indicate to video decoder 30 and/or video post-processor unit 31 the minimum and maximum permitted values of the samples (e.g., color component values) based on what sample range is used (e.g. full, restricted, standard, or others). In another example, video encoder 20 and/or video pre-processor unit 19 may be configured to signal one or more syntax values (e.g., in an SEI message) that indicate the minimum and maximum permitted values of the samples to video decoder based on what sample range is used (e.g. full, restricted, standard). Video decoder 30 and/or video post-processor unit 31 may then determine the range of component values allowed for the inverse dynamic range adjustment process based on the received minimum value and the received maximum value.

**[0151]** In another example, video encoder 20 and/or video pre-processor unit 19 may be configured to signal a flag (e.g., in an SEI message) to indicate whether the scale values are signed or unsigned. In this example, the parsing process of any SEI messages is the same regardless the value of the flag.

**[0152]** The following section includes several examples of embodiments that use example techniques disclosed in the previous section. In this embodiment, the component scaling function is signaled as a lookup table and the number of bits used to signal the points defining the lookup table are also signaled. In one example, the lookup defines a piece-wise linear mapping function. The points for the lookup table correspond to the (x,y) coordinates that define the piece-wise linear mapping. For sample values that do not have explicit points signaled, the value is interpolated based on the neighboring pivot points.

**[0153]** The derivation process of the ranges and the output sample values are defined as below.

**[0154]** The mapping of sample x from component c to sample  $y = \text{map}[c](x)$  is specified as follows:

- Set the value of DefaultPrecShift equal to 9
- Let the variables minSampleVal and maxSampleVal denote the minimum and the maximum sample values as defined by the sample range of the content.
- The variable ScaleValue[ c ][ i ], for i in the range of 0 to  $\text{hdr\_recon\_num\_ranges}[ c ] - 1$ , inclusive, is derived as follows:

```

    SignValue[ c ][ i ] = 0 // 0 for positive, 1 for negative
    hdrReconScaleBitDepth = hdr_recon_scale_bit_depth -
    ( hdr_negative_scales_present_flag ? 1 : 0 )
    if( hdr_negative_scales_present_flag )
        ScaleValue[ c ][ i ] = hdr_recon_scale_val[ c ][ i ] & ( ( 1 <<
        hdrReconScaleBitDepth ) - 1 )      (D-xx)
        SignValue[ c ][ i ] = hdr_recon_scale_val[ c ][ i ] & ( 1 <<
        hdrReconScaleBitDepth )
    else
        ScaleValue[ c ][ i ] = hdr_recon_scale_val[ c ][ i ] (D-xx)
    shiftInvScale = 1 << hdrReconScaleBitDepth
    InvScaleValue[ c ][ i ] = ( 1 << (DefaultPrecShift + hdrReconScaleBitDepth)
    + shiftInvScale ) /
        ScaleValue[ c ][ i ]

```

- The variable RangeValue[ c ][ i ], for i in the range of 0 to  $\text{hdr\_recon\_num\_ranges}[ c ] - 1$ , inclusive, is derived as follows:
  - If  $\text{hdr\_recon\_equal\_ranges\_flag}[ c ]$  is equal to 0, the following applies:
 

```

                        RangeValue[ c ][ i ] = hdr_recon_range_val[ c ][ i ] (D-xx)
                    
```
  - Otherwise (  $\text{hdr\_recon\_equal\_ranges\_flag}[ c ]$  is equal to 1 ), the following applies:
 

```

                        RangeValue[ c ][ i ] = ( ( InputDynamicRangeValue <<
                        hdr_recon_offset_frac_bit_depth ) +
                        ( ( hdr_recon_num_ranges[ c ] + 1 ) >> 1 ) ) /
                        hdr_recon_num_ranges[ c ]      (D-xx)
                    
```

where InputDynamicRangeValue is equal to 1 when the sample range is normalized from 0 to 1.

- The variables InputRanges[ c ][ i ] and OutputRanges[ c ][ i ], for i in the range of 0 to hdr\_recon\_num\_ranges[ c ], inclusive, are derived as follows:

- If i is equal to 0, the following applies:

$$\text{OutputRanges}[c][i] = -\text{hdr\_recon\_global\_offset\_val}[c] * \text{InvScaleValue}[c][i-1] \quad (\text{D-xx})$$

$$\text{InputRanges}[c][i] = 0 \quad (\text{D-xx})$$

- Otherwise (i is not equal to 0), the following applies:

$$\text{InputRanges}[c][i] = \text{InputRanges}[c][i-1] + \text{RangeValue}[c][i-1] \quad (\text{D-xx})$$

$$\begin{aligned} \text{OutputRanges}[c][i] = & \text{OutputRanges}[c][i-1] + \\ & \text{RangeValue}[c][i-1] * \text{InvScaleValue}[c][i-1] \end{aligned} \quad (\text{D-xx})$$

- The parameters OffsetValue[ c ][ i ], for i in the range of 0 to hdr\_recon\_num\_ranges[ c ] – 1, inclusive, are derived as follows:

$$\begin{aligned} \text{precOffsetDeltaBits} &= \text{DefaultPrecShift} + \text{hdr\_recon\_scale\_frac\_bit\_depth} \\ \text{OffsetValue}[c][i] &= \text{InputRanges}[c][i+1] * (1 \ll \text{precOffsetDeltaBits}) \end{aligned}$$

–

$$\text{OutputRanges}[c][i+1] * \text{ScaleValue}[c][i-1] \quad (\text{D-xx})$$

$$\text{OffsetValue}[c][i] = ( ( \text{OffsetValue}[c][i] + ( 1 \ll (\text{BitDepth} - 1) ) ) \gg \text{BitDepth} ) *$$

$$( \text{maxSampleVal} - \text{minSampleVal} )$$

- The parameter  $y = \text{map}[c](x)$  is derived as follows:
  - Variable `bitDepthDelta` is set equal to `DefaultPrecShift +`  
`hdr_recon_offset_frac_bit_depth – BitDepth`
  - If  $(x \ll \text{bitDepthDelta})$  is lower than or equal to `OutputRanges[c][0]`, the following applies:

$$y = \text{InputRanges}[c][0] \quad (\text{D-xx})$$

$$\text{fracBitDepth} = \text{hdr\_recon\_offset\_frac\_bit\_depth}$$

- Otherwise if  $(x \ll \text{bitDepthDelta})$  is larger than `OutputRanges[c][hdr_recon_num_ranges[c]]`, the following applies:

$$y = \text{InputRanges}[c][\text{hdr\_recon\_num\_ranges}[c]](\text{D-xx})$$

$$\text{fracBitDepth} = \text{hdr\_recon\_offset\_frac\_bit\_depth}$$

- Otherwise, the following applies:

$$\text{fracBitDepth} = \text{DefaultPrecShift} + \text{hdr\_recon\_scale\_frac\_bit\_depth} + \text{hdr\_recon\_offset\_frac\_bit\_depth} - \text{BitDepth}$$

$$\text{for}(i = 1; i \leq \text{hdr\_recon\_num\_ranges}[c]; i++)$$

$$\text{if}(\text{OutputRanges}[i-1] < (x \ll \text{bitDepthDelta}) \ \&\&$$

$$(x \ll \text{bitDepthDelta}) \leq \text{OutputRanges}[i]) \{$$

$$\text{rangeBitShift} = \text{DefaultPrecShift} +$$

$$\text{hdr\_recon\_offset\_frac\_bit\_depth} - \text{BitDepth}$$

$$y = (x - \text{minSampleVal}) * \text{ScaleValue}[c][i-1] * (1 \ll \text{rangeBitDepth}) +$$

$$\text{OffsetValue}[c][i-1] +$$

$$\text{minSampleVal} * (1 \ll \text{fracBitDepth}) \ (\text{D-xx})$$

$$\}$$

- $\text{fracShiftOffset} = 1 \ll (\text{fracBitDepth} - 1)$   
 $y = (y + \text{fracShiftOffset}) \gg \text{fracBitDepth}$

Alternatively, the adjustment of the sample range based on `minSampleVal` and `maxSampleVal` are not performed on the `OffsetValue`, but rather on the `InputRanges` and `OutputRanges` as follows:

$$\text{deltaSampleVal} = \text{maxSampleVal} - \text{minSampleVal}$$

```

deltaBitShift = DefaultPrecShift + hdr_recon_offset_frac_bit_depth
sampleShift = ( 1 << ( BitDepth - 1 ) ) + ( minSampleVal << deltaBitShift ) )
OutputRanges[ c ][ i ] = ( ( OutputRanges[ c ][ i ] * deltaSampleVal ) +
sampleShift ) >>

```

BitDepth

```

deltaBitShift = DefaultPrecShift + hdr_recon_offset_frac_bit_depth
sampleShift = ( 1 << ( BitDepth - 1 ) ) + ( minSampleVal << deltaBitShift ) )
InputRanges[ c ][ i ] = ( ( InputRanges[ c ][ i ] * deltaSampleVal ) + sampleShift )
>>

```

BitDepth

**[0155]** This disclosure provides several techniques to improve carriage of component scaling information using SEI signaling and processing or other means which is specified or to be specified in video coding standards, such as H.265/HEVC, H.264/AVC, BDA, MPEG or others. One or more of these techniques may be applied independently, or in combination with others. In addition, the techniques described above for signaling and/or using information in SEI messages for performing a fixed-point implementation of a dynamic range process may utilize one or more of the syntax structures described below for signaling/receiving the information.

**[0156]** In some examples, video encoder 20 may signal one or more SEI messages that include global offset values, including, for each component, a first offset value that determines a first unadjusted component value below which all component values are clipped to the first component value before applying dynamic range adjustment as described in this disclosure. Decoder 30 may receive one or more of such SEI messages, parse and/or decode the information in the SEI messages, and pass the information to the video post-processor 31.

**[0157]** In some examples, for each component, video encoder 20 may signal one or more SEI messages that include a second offset value that specifies the adjusted value to which the first offset value maps to after dynamic range adjustment. Video decoder 30 may receive such SEI messages, parse and/or decode the information, and pass that information to video post-processor 31.

**[0158]** In another example, neither the first global offset value nor the second global offset value is signaled in a SEI message. Instead, decoder 30 assumes that the values



of the first global offset and the second global offset is a constant, predetermined, or signaled value that the decoder 30 either determines per sequence or receives by external means. In another example, video encoder 20 signals the first global offset value in an SEI message, but the second global offset value is not signaled in a SEI message. Instead, video decoder 30 infers that its value is a constant, predetermined, or signaled value that decoder 30 either determines per sequence or received by external means. In a still further example, video encoder 20 signals the second global offset value in an SEI message, but the first global offset value is not signaled in a SEI message. Instead, video decoder 30 infers that the first global offset value is a constant, predetermined, or signaled value that decoder 30 either determines per sequence or received by external means.

**[0159]** In some examples, video encoder 20 may signal offset values that are received by decoder 30, and are used by decoder 30 to derive other global or local parameters, including both global and local scale and offset values, as well as partitions of a range of unadjusted values, and partitions of a range of adjusted values.

**[0160]** In some examples, video encoder 20 may signal one or more SEI messages that include the number of partitions that the range of input representation values (i.e., component values) was divided into during dynamic range adjustment. In one example, the number of partitions may be constrained to be a power of 2 (i.e. 1, 2, 4, 8, 16, etc.) and the number of partitions is signaled as logarithm (e.g. 8 partitions is signaled as  $3 = \log_2 8$ ). Video decoder 30 may receive such SEI messages, parse and/or decode the information, and pass that information to video post-processor 31.

**[0161]** In some examples, the number of partitions for the chroma components may be different from the number of partitions for the luma component. The number of partitions may be constrained to be a power of 2 + 1 and signaled as logarithm and rounding towards minus 0. In this way, pixels with neutral chroma can have their own values and the size of that partition can be smaller than the other partitions. In such an example, neutral chroma may refer to values of chroma around the mid-value (e.g., 0 when the chroma values range between -0.5 and 0.5, or between -512 and 511 in a 10-bit representation). Constraining the number of partitions as a power of 2 may enable the encoder 20 to save bits, because encoder 20 may be able to represent the log of a value with fewer bits than the actual value for integer values. Constraining the number of partitions to a power of 2 + 1 may ensure that at least one partition may be dedicated to the neutral chroma values, and in some examples, the width of the partition

corresponding to the neutral chroma values may be smaller than the rest. In other examples, such a partition may be larger than one or more of the other partitions.

**[0162]** In some examples, decoder 30 may use the signaled number of partitions to derive other global or local parameters, including both global and local scale and offset values, as well as the actual size of the partitions of a range of unadjusted component values and/or the size of the partitions of a range of adjusted component values.

**[0163]** In some examples, encoder 20 may signal one or more SEI messages that may include, for each partition, a local scale and local offset value specifying a range of the input component values and the corresponding mapped output component values. In some examples, encoder 20 may signal an SEI message that includes the number of bits used by the syntax elements to signal the scale and offsets. In other examples, encoder 20 may signal an SEI message that indicates the number of bits that are used to represent the fractional part of the scale and offsets in the syntax elements. In other examples, encoder 20 may signal one or more SEI messages or syntax elements that indicate that the integer part of the scale parameters is signaled in a signed representation. In some examples, the signed representation is two's complement. In other examples, the signed representation is signed magnitude representation. Video decoder 30 may receive such SEI messages and/or syntax elements, parse and/or decode the information, and pass that information to video post-processor 31.

**[0164]** In other examples, encoder 20 may use each offset value successively to first compute the range of adjusted component or representation values, and then using the scale value, compute the corresponding range in the unadjusted representation. For example, one offset value may be used to compute the range of a first partition in the adjusted component using the value of a global offset value derived or signalled for the adjusted component, followed by using the scale value and the range of a first partition of the adjusted representation to derive the range in the corresponding partition of the unadjusted representation and with the respective ranges of the first partition of the adjusted and the corresponding partition of the unadjusted representations, derive a respective value derived for the first partition of the adjusted range and the corresponding partition of unadjusted representations that indicate a boundary of the partitions. Following this, another offset value may be used to compute the range of a second partition in the adjusted component using the boundary value of the first partition in the adjusted component derived in the previous step, followed by using the scale value and the range of a second partition of the adjusted representation to derive

the range of the unadjusted representation, and with the respective ranges of the second partitions of the adjusted representation and corresponding partition of the unadjusted representations, a respective value is derived for the partitions in the adjusted and unadjusted representations that indicate a boundary of the respective representations. This method is repeated until all the ranges and boundaries are derived for all the partitions in the adjusted and unadjusted representations. In another example, encoder 20 may use each offset value successively to first compute the range of unadjusted component or representation values, and then using the scale value, compute the corresponding range in the adjusted representation. In other words, the component or representation to which the scale and offset values are applied could be swapped between unadjusted and adjusted representations.

**[0165]** In some examples, the number of bits used by the syntax elements to signal scale and offset values may depend on the component. In other examples, a default number of bits is defined and used when these numbers are not explicitly signaled.

**[0166]** In some examples, encoder 20 may signal a syntax element indicating whether the length of the partitions of the output representations (i.e., output components) are equal. In such an example, encoder 20 might not signal the offset value for one or more partitions. Decoder 30 may infer the offset values to be equal in some examples. In another example, decoder 30 may assume the partitions are of equal length and may not receive a syntax element so indicating. In some examples, decoder 30 may derive the size of each partition from signaled syntax elements and predefined total dynamical range of the representation.

**[0167]** In other examples, rather than signaling pivot points for each partition as well as scale and offset values for each partition, video encoder 20 may signal one or more SEI messages that indicate derivative or scale value for each partition along with the size of one or more or all partitions. This approach may allow encoder 20 to avoid signaling local offset values for each partition. Instead, in some examples, encoder 20 may be able to signal, in one or more SEI messages, the partition size and scale value (or derivative) for one or more partitions. The local offset value for each partition or partitioning (which may require higher accuracy) may be determined or derived by decoder 30.

**[0168]** In some examples, encoder 20 may signal one or more SEI messages that indicate a mode value that specifies several default values for offset and scale values for

certain partitions. Video decoder 30 may receive such SEI messages, parse and/or decode the information, and pass that information to video post-processor 31.

**[0169]** In some examples, encoder 20 may signal one or more SEI messages that indicate a value defining the persistence of the SEI message such that the persistence of a subset of the components may be defined and component scale values of a subset of the components may be updated. The persistence of an SEI message indicates the pictures to which the values signalled in the instance of the SEI may apply. In some examples, the persistence of the SEI message is defined such that the values signalled in one instance of SEI messages may apply correspondingly to the all components of the pictures to which the SEI message applies. In other examples, the persistence of the SEI message is defined such that the values signalled in one instance of the SEI may be indicated to apply correspondingly to a subset of the components wherein the components to which the values in the instance of the SEI does not apply may either have no values applicable or may have values applicable that are signalled in another instance of the SEI message. Video decoder 30 may receive such SEI messages, parse and/or decode the information, and pass that information to video post-processor 31.

**[0170]** In some examples, encoder 20 may signal one or more SEI messages that include syntax elements indicating the post-processing steps to be performed to the decoded output. Each syntax element may be associated with a particular process (e.g. scaling components, color transforms, up-sampling/down-sampling filters, etc.) and each value of the syntax element may specify that a particular set of parameters associated with the process be used. In some examples, the parameters associated with the process are signaled by video encoder 20 using SEI messages that are part of the bitstream or as metadata that may be transmitted through other means. Video decoder 30 may receive such SEI messages, parse and/or decode the information, and pass that information to video post-processor 31.

**[0171]** In some examples, encoder 20 may signal syntax elements or one or more SEI messages that may be used for describing and/or constructing a piece-wise linear model function for mapping input representations (i.e., input component values) to output representations (i.e., output component values). Video decoder 30 may receive such SEI messages, parse and/or decode the information, and pass that information to video post-processor 31. In other examples, predefined assumptions may be used for describing and/or constructing a piece-wise linear model function for mapping input representations to the output representation.

**[0172]** In some examples, encoder 20 may signal one or more SEI messages that may include one or more syntax elements indicating that the scale and offset parameters signaled in the SEI message represent the variation of the scale to be applied to a first component as a function of different values of a second component.

**[0173]** In some examples, encoder 20 may signal one or more SEI messages indicating offset parameters that are to be or may be applied along with the scale on a first component as a function of different values of a second component. In some examples, encoder 20 may signal one or more SEI messages that may include one or more additional syntax elements that indicating offset parameters that are to be or may be applied along with the scale on a first component as a function of different values of a second component. Video decoder 30 may receive such SEI messages, parse and/or decode the information, and pass that information to video post-processor 31.

**[0174]** In some examples, encoder 20 may signal one or more SEI messages including a first syntax element that indicates a first set of electro-optical transfer function characteristics such that the signaled scale, offset and other dynamic range adjustment parameters the SEI message are applied when the electro-optical transfer function characteristics used on the decoder-side are similar to that first set of electro-optical transfer function characteristics.

**[0175]** In another example, encoder 20 may signal one or more SEI messages indicating that the signaled offset, scale and other dynamic range parameters in the SEI message(s) are to be applied for best reconstruction of the HDR output when the first set of electro-optical transfer function characteristics, or those with similar characteristics, are used by the decoder 30. Video decoder 30 may receive such SEI messages, parse and/or decode the information, and pass that information to video post-processor 31.

**[0176]** In another example, encoder 20 may signal one or more SEI messages indicating that a first set of opto-electronic transfer function characteristics, and the signaled scale, offset and other dynamic range adjustment parameters are applied on by decoder 30 when the corresponding inverse electro-optical transfer function characteristics are applied at the decoder side. Video decoder 30 may receive such SEI messages, parse and/or decode the information, and pass that information to video post-processor 31.

**[0177]** In other examples, encoder 20 may signal a condition such that when more than one SEI message is present indicating different set of electro-optical/opto-electronic characteristics and applicable the current picture, only one SEI message is applied. The encoder may signal different set of electro-optical/opto-electronic characteristics to

satisfy different types of decoders, or decoders with different capabilities. For example, some displays at the decoder side may apply the PT EOTF to convert the coded component values in appropriate domain to linear light, whereas other displays, e.g. legacy displays, may apply the gamma EOTF to convert to linear light. Each SEI with a particular characteristic that the encoder sends may be appropriate or beneficial for certain types of displays and not for other types of displays, e.g. an SEI message with PQ EOTF characteristics may be suitable for displays that apply PQ EOTF to convert the coded video to linear light. The decoder 30 determines which SEI message is to be applied, and makes such a choice based on the application standard, based on the end-user device, based on a signal received, or based on another indication received through external means. For example, decoder 30 may determine that the first syntax element in a first SEI message that applies to a current picture indicates that the SEI message is to be applied with the inverse of PQ OETF and the first syntax element in a second SEI message that applies to a current picture indicates that the SEI message is to be applied with another transfer function (such as BBC, or PH), the decoder 30 or end-user device may choose to apply the parameters in the first SEI message because the device uses PQ EOTF. In some examples, an application standard to which the decoder conforms to may specify that an SEI message with a particular set of characteristics is to be used.

**[0178]** In other examples, encoder 20 may signal an SEI message that carries the parameters corresponding to multiple sets of transfer characteristics. In other examples, encoder 20 may signal different SEI messages for that purpose. Video decoder 30 may receive such SEI messages, parse and/or decode the information, and pass that information to video post-processor 31

**[0179]** In some examples, encoder 20 may signal one or more SEI messages that include a syntax element indicating the applicability of the SEI message. The applicability of the SEI message may include, but is not limited to (1) the components to which the scales and offsets apply, (2) the position at which the component scaling is applied, and/or (3) whether additional scaling parameters are signaled.

**[0180]** As described, encoder 20 may signal one or more SEI messages that include a syntax element indicating the components to which the scales and offsets apply. The following lists several examples of such an application. For example, one value of the syntax element could indicate that signaled parameters for the first component index are to be applied to the RGB components. Another value may indicate that the signaled parameters for the first component index is to be applied to luma component, and those

for the second and third indices are to be applied to the Cb and Cr components. Another value may indicate that the signaled parameters for the first component index is to be applied to R, G and B components, and those for the second and third indices are to be applied to the Cb and Cr components. Another value may indicate that signaled parameters for first three indices are applied to luma, Cb and Cr components, and that corresponding to the remaining indices are applied for color correction. Video decoder 30 may receive such SEI messages, parse and/or decode the information, and pass that information to video post-processor 31.

**[0181]** Also as described, encoder 20 may signal one or more SEI messages including a syntax element indicating the position at which the component scaling is applied.

Several processes occur on the decoder-side, after decoding of the video, and in the video post-processor 31. Signaling of syntax element indicating the position at which the process associated with the SEI is to be applied, in other words indication of any subset of the preceding or succeeding operations of the process associated with using the information in the SEI, would be helpful to the video decoder 30 or the video post-processor 31 to process the video. For example, such a syntax element could indicate the position at which the component scaling is applied, for example to YCbCr components before or after upsampling. In another example, the syntax element could indicate that the component scaling is applied before the quantization on the decoder side. Video decoder 30 may receive such SEI messages, parse and/or decode the information, and pass that information to video post-processor 31.

**[0182]** Also as described, encoder 20 may signal one or more SEI messages that include a syntax element indicating whether an additional set of scaling and parameters, e.g. for color correction, are signaled. The additional set of parameters could be used for color correction to map the color components to fit a particular color gamut, or for correction of component values when a different transfer function is applied than that indicated by the `transfer_characteristics` syntax element in the VUI.

**[0183]** In other examples, encoder 20 may signal different syntax elements to indicate the above aspects; e.g. one syntax element to indicate which component(s) the SEI applies to, one syntax element to indicate whether it applies to HDR-compatible or SDR-compatible content, and one syntax element to indicate the position(s) where the component scaling SEI message is to be applied.

**[0184]** When the number of components to which the component scaling SEI message parameters are applied is more than one, encoder 20 may signal one or more SEI

messages that include a syntax element indicating that application of scale and offset parameters may be done sequentially based on the index of the component. For example, the mapping based on the scale and offset parameters of the first component may be applied, and then the mapping of the second component, which for example uses scale and offset signaled for the second component, may depend on the values of the first component. In some examples, this is indicated by, for example, by syntax element specifying that the mapped values of the first component should be used. Video decoder 30 may receive such SEI messages, parse and/or decode the information, and pass that information to video post-processor 31.

**[0185]** In another example, video encoder 20 may constrain the values signaled in one or more SEI messages, or in the bitstream, in such a way that an HDR10 receiver can decode and show a viewable HDR video even if the SEI post-processing is not applied. The SEI message(s) may include a syntax element to indicate that this is the case (e.g., that the bitstream is an HDR10 backward compatible bitstream).

**[0186]** This section includes several examples that use techniques disclosed in accordance with one or more aspects of the present disclosure.

#### **Example 1**

**[0187]** In this example 1, the component scaling function is signaled as a look-up table and the number of bits used to signal the points defining the look up table are also signaled. For sample values that do not have explicit points signaled, the value is interpolated based on the neighboring pivot points.



### Syntax of the component scaling SEI message

component_scale_info( payloadSize ) {	Descriptor
<b>comp_scale_id</b>	ue(v)
<b>comp_scale_cancel_flag</b>	u(1)
if( !comp_scale_cancel_flag ) {	
<b>comp_scale_persistence_flag</b>	u(1)
<b>comp_scale_num_comps_minus1</b>	ue(v)
<b>comp_scale_input_bit_depth</b>	ue(v)
<b>comp_scale_output_bit_depth</b>	ue(v)
for( c = 0; c <= comp_scale_num_comps_minus1; c++ ) {	
<b>comp_scale_num_points_minus1[ c ]</b>	ue(v)
for( i = 0; i <= comp_scale_num_points_minus1[ c ]; i++ ) {	
<b>comp_scale_input_point[ c ][ i ]</b>	u(v)
<b>comp_scale_output_point[ c ][ i ]</b>	u(v)
}	
}	
}	
}	

### Semantics of the component scaling SEI message

The component scaling SEI message provides information to perform scaling operations on the various components of the decoded pictures. The colour space and the components on which the scaling operations should be performed are determined by the value of the syntax elements signalled in the SEI message.

**comp\_scale\_id** contains an identifying number that may be used to identify the purpose of the component scaling SEI message. The value of **comp\_scale\_id** shall be in the range of 0 to  $2^{32} - 2$ , inclusive. The value of **comp\_scale\_id** may be used to specify the colour space at which the component scaling SEI message, or whether the component scaling SEI message is applied in the linear or the non-linear domain.

Values of **comp\_scale\_id** from 0 to 255, inclusive, and from 512 to  $2^{31} - 1$ , inclusive, may be used as determined by the application. Values of **comp\_scale\_id** from 256 to 511, inclusive, and from  $2^{31}$  to  $2^{32} - 2$ , inclusive, are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore all component scale information SEI messages

containing a value of `comp_scale_id` in the range of 256 to 511, inclusive, or in the range of  $2^{31}$  to  $2^{32} - 2$ , inclusive, and bitstreams shall not contain such values.

NOTE 1 – The `comp_scale_id` can be used to support component scaling processes that are suitable for different display scenarios. For example, different values of `comp_scale_id` may correspond to different display bit depths or different colour spaces in which the scaling is applied.

Alternatively, the `comp_scale_id` may also be used to identify whether the scaling is performed for compatibility to certain types of displays or decoder, e.g. HDR, SDR.

**`comp_scale_cancel_flag`** equal to 1 indicates that the component scaling information SEI message cancels the persistence of any previous component information SEI messages in output order that applies to the current layer. `comp_scale_cancel_flag` equal to 0 indicates that component scaling information follows.

**`comp_scale_persistence_flag`** specifies the persistence of the component scaling information SEI message for the current layer.

`comp_scale_persistence_flag` equal to 0 specifies that the component scaling information applies to the current decoded picture only.

Let `picA` be the current picture. `comp_scale_persistence_flag` equal to 1 specifies that the component scaling information persists for the current layer in output order until any of the following conditions are true:

- A new CLVS of the current layer begins.
- The bitstream ends.
- A picture `picB` in the current layer in an access unit containing a component scaling information SEI message with the same value of `comp_scale_id` and applicable to the current layer is output for which `PicOrderCnt( picB )` is greater than `PicOrderCnt( picA )`, where `PicOrderCnt( picB )` and `PicOrderCnt( picA )` are the `PicOrderCntVal` values of `picB` and `picA`, respectively, immediately after the invocation of the decoding process for picture order count for `picB`.

**`comp_scale_num_comps_minus1`** plus 1 specifies the number of components for which the component scaling function is specified. `comp_scale_num_comps_minus1` shall be in the range of 0 to 2, inclusive.

When `comp_scale_num_comps_minus1` is less than 2 and the component scaling parameters of the  $c$ -th component is not signalled, are inferred to be equal to those of the  $(c - 1)$ -th component.

Alternatively, when `comp_scale_num_comps_minus1` is less than 2, and the component scaling parameters of the *c*-th component is not signalled, the component scaling parameters of the *c*-th component are inferred to be equal to default values such that effectively there is no scaling of that component.

Alternatively, the inference of the component scaling parameters may be specified based on the colour space on which the SEI message is applied.

- When the colour space is YCbCr, and `comp_scale_num_comps_minus1` is equal to 1, the component scaling parameters apply to both Cb and Cr components.
- When the colour space is YCbCr, and `comp_scale_num_comps_minus1` is equal to 2, the first and second component scaling parameters apply to Cb and Cr components.

In one alternative, the different inference is specified based on the value of `comp_scale_id` or on the basis of an explicit syntax element.

Alternatively, a constraint is added as follows:

It is constraint for bitstream conformance that the value of `comp_scale_num_comps_minus1` shall be the same for all the component scaling SEI message with a given value of `comp_scale_id` within a CLVS.

**`comp_scale_input_bit_depth_minus8`** plus 8 specifies the number of bits used to signal the syntax element `comp_scale_input_point[ c ][ i ]`. The value of `comp_scale_input_bit_depth_minus8` shall be in the range of 0 to 8, inclusive.

When component scaling SEI message is applied to an input that is in a normalized floating point representation in the range 0.0 to 1.0, the SEI message refers to the hypothetical result of a quantization operation performed to convert the input video to a converted video representation with bit depth equal to `colour_remap_input_bit_depth_minus8 + 8`.

When component scaling SEI message is applied to a input that has a bit depth not equal to the `comp_scale_input_bit_depth_minus8 + 8`, the SEI message refers to the hypothetical result of a transcoding operation performed to convert the input video representation to a converted video representation with bit depth equal to `colour_remap_input_bit_depth_minus8 + 8`.

**`comp_scale_output_bit_depth_minus8`** plus 8 specifies the number of bits used to signal the syntax element `comp_scale_output_point[ c ][ i ]`. The value of `comp_scale_output_bit_depth_minus8` shall be in the range of 0 to 8, inclusive.

When component scaling SEI message is applied to an input that is in floating point representation, the SEI message refers to the hypothetical result of an inverse quantization operation performed to convert the video representation with a bit depth equal to  $\text{colour\_remap\_output\_bit\_depth\_minus8} + 8$  that is obtained after processing of the component scaling SEI message to a floating point representation in the range 0.0 to 1.0.

Alternatively, the number of bits used to signal  $\text{comp\_scale\_input\_point}[c][i]$  and  $\text{comp\_scale\_output\_point}[c][i]$  are signalled as  $\text{comp\_scale\_input\_bit\_depth}$  and  $\text{comp\_scale\_output\_bit\_depth}$ , respectively, or in other words without subtracting 8.

**comp\_scale\_num\_points\_minus1[c]** plus 1 specifies the number of pivot points used to define the component scaling function.  $\text{comp\_scale\_num\_points\_minus1}[c]$  shall be in the range of 0 to  $(1 \ll \text{Min}(\text{comp\_scale\_input\_bit\_depth\_minus8} + 8, \text{comp\_scale\_output\_bit\_depth\_minus8} + 8)) - 1$ , inclusive.

**comp\_scale\_input\_point[c][i]** specifies the i-th pivot point of the c-th component of the input picture. The value of  $\text{comp\_scale\_input\_point}[c][i]$  shall be in the range of 0 to  $(1 \ll \text{comp\_scale\_input\_bit\_depth\_minus8}[c] + 8) - 1$ , inclusive. The value of  $\text{comp\_scale\_input\_point}[c][i]$  shall be greater than or equal to the value of  $\text{comp\_scale\_input\_point}[c][i - 1]$ , for i in the range of 1 to  $\text{comp\_scale\_points\_minus1}[c]$ , inclusive.

**comp\_scale\_output\_point[c][i]** specifies the i-th pivot point of the c-th component of the output picture. The value of  $\text{comp\_scale\_output\_point}[c][i]$  shall be in the range of 1 to  $(1 \ll \text{comp\_scale\_output\_bit\_depth\_minus8}[c] + 8) - 1$ , inclusive. The value of  $\text{comp\_scale\_output\_point}[c][i]$  shall be greater than or equal to the value of  $\text{comp\_scale\_output\_point}[c][i - 1]$ , for i in the range of 1 to  $\text{comp\_scale\_points\_minus1}[c]$ , inclusive.

The process of mapping an input signal representation x and an output signal representation y, where the sample values for both input and output are in the range of 0 to  $(1 \ll \text{comp\_scale\_input\_bit\_depth\_minus8}[c] + 8) - 1$ , inclusive, and 0 to  $(1 \ll \text{comp\_scale\_output\_bit\_depth\_minus8}[c] + 8) - 1$ , inclusive, respectively, is specified as follows:

```
if( x <= comp_scale_input_point[ c ][ 0 ] )
    y = comp_scale_output_point[ c ][ 0 ]
else if( x > comp_scale_input_point[ c ][ comp_scale_input_point_minus1[ c ] ] )
    y = comp_scale_output_point[ c ][ comp_scale_output_point_minus1[ c ] ]
```

else

for( i = 1; i <= comp\_scale\_output\_point\_minus1[ c ]; i++ )

if( comp\_scale\_input\_point[ i - 1 ] < x && x <=

comp\_scale\_input\_point[ i ] )

y = ( ( comp\_scale\_output\_point[ c ][ i ] - comp\_scale\_output\_point[ c ][ i - 1 ] ) ÷

( comp\_scale\_input\_point[ c ][ i ] - comp\_scale\_input\_point[ c ][ i - 1 ] ) ) \*

( x - comp\_scale\_input\_point[ c ][ i - 1 ] ) +

( comp\_scale\_output\_point[ c ][ i - 1 ] )

**[0188]** In one alternative, input and output pivot points comp\_scale\_input\_point[ c ][ i ] and comp\_scale\_output\_point[ c ][ i ] are coded as difference of adjacent values; e.g., delta\_comp\_scale\_input\_point[ ][ ] and delta\_comp\_scale\_output\_point[ ][ ], and the syntax elements are coded using exponential Golomb codes.

In another alternative, the process of mapping an input and output representation value is specified by other interpolation methods including, but not limited to, splines and cubic interpolation.

## Example 2

**[0189]** This Example 2 shows a different syntax structure compared to the SEI syntax structure described in Example 1. In this syntax structure, the mapping function is described in terms of scales and offsets instead of pivot points.

**Syntax of the component scaling SEI message**

component_scale_info( payloadSize ) {	Descriptor
<b>comp_scale_id</b>	ue(v)
<b>comp_scale_cancel_flag</b>	u(1)
if( !comp_scale_cancel_flag ) {	
<b>comp_scale_persistence_flag</b>	u(1)
<b>comp_scale_num_comps</b>	ue(v)
<b>comp_scale_input_bit_depth</b>	ue(v)
<b>comp_scale_output_bit_depth</b>	ue(v)
<b>comp_scale_bit_depth_scale_val</b>	ue(v)
<b>comp_scale_log2_denom_scale_val</b>	ue(v)
for( c = 0; c < comp_scale_num_comps; c++ ) {	
<b>comp_scale_num_points_minus1[ c ]</b>	ue(v)
<b>comp_scale_global_offset_input_val[ c ]</b>	u(v)
<b>comp_scale_global_offset_output_val[ c ]</b>	u(v)
for( i = 0; i < comp_scale_num_points_minus1[ c ]; i++ ) {	
<b>comp_scale_offset_val[ c ][ i ]</b>	u(v)
<b>comp_scale_val[ c ][ i ]</b>	u(v)
}	
}	
}	
}	

**comp\_scale\_bit\_depth\_scale\_val** specifies the number of bits used to signal the syntax element `comp_scale_val[ c ][ i ]`. The value of `comp_scale_bit_depth_scale_val` shall be in the range of 0 to 24, inclusive.

**comp\_scale\_log2\_denom\_scale\_val** specifies the base 2 denominator of the scale value. The value of `comp_scale_log2_denom_scale_val` shall be in the range of 0 to 16, inclusive.

**comp\_scale\_global\_offset\_input\_val[ c ]** plus 1 specifies the input sample value below which all the input representation values are clipped to

`CompScaleOffsetOutputVal[ c ][ 0 ]`, used to define the component scaling function.

`comp_scale_num_points_minus1[ c ]` shall be in the range of 0 to

$(1 \ll \text{comp\_scale\_input\_bit\_depth}) - 1$ , inclusive. The number of bits used to represent  $\text{comp\_scale\_global\_offset\_input\_val}[c]$  is  $\text{comp\_scale\_input\_bit\_depth}$ .

**comp\_scale\_global\_offset\_output\_val** $[c]$  plus 1 specifies the output sample value to which all the input representation values below

$\text{comp\_scale\_global\_offset\_input\_val}[c]$  are to be clipped.

$\text{comp\_scale\_num\_points\_minus1}[c]$  shall be in the range of 0 to

$(1 \ll \text{comp\_scale\_output\_bit\_depth}) - 1$ , inclusive. The number of bits used to represent  $\text{comp\_scale\_global\_offset\_output\_val}[c]$  is  $\text{comp\_scale\_output\_bit\_depth}$ .

**comp\_scale\_num\_points\_minus1** $[c]$  plus 1 specifies the number of pivot points used to define the component scaling function.  $\text{comp\_scale\_num\_points\_minus1}[c]$  shall be in the range of 0 to  $(1 \ll \text{Min}(\text{comp\_scale\_input\_bit\_depth}, \text{comp\_scale\_output\_bit\_depth}) - 1)$ , inclusive.

The process of mapping an input signal representation  $x$  and an output signal representation  $y$ , where the sample values for both input representation is in the range of 0 to  $(1 \ll \text{comp\_scale\_input\_bit\_depth}) - 1$ , inclusive, and output representation is in the range of 0 to  $(1 \ll \text{comp\_scale\_output\_bit\_depth}) - 1$ , inclusive, is specified as follows:

```

if( x <= CompScaleOffsetInputVal[ c ][ 0 ] )
    y = CompScaleOffsetOutputVal[ c ][ 0 ]
else if( x > CompScaleOffsetInputVal[ c ][ comp_scale_output_point_minus1 ] )
    y = CompScaleOffsetOutputVal[ c ][ comp_scale_output_point_minus1 ]
else
    for( i = 1; i <= comp_scale_output_point_minus1; i++ )
        if( CompScaleOffsetInputVal[ i - 1 ] < x && x <=
CompScaleOffsetInputVal[ i ] )
            y = ( x -
CompScaleOffsetInputVal[ i - 1 ] * ( comp_scale_val[ c ][ i ] +
CompScaleOffsetOutputVal[ c ][ i ]

```

**comp\_scale\_offset\_val** $[c][i]$  specifies the offset value of the  $i$ -th sample value region of the  $c$ -th component. The number of bits used to represent  $\text{comp\_scale\_offset\_val}[c]$  is equal to  $\text{comp\_scale\_input\_bit\_depth}$ .

**comp\_scale\_val** $[c][i]$  specifies the scale value of the  $i$ -th sample value region point of the  $c$ -th component. The number of bits used to represent  $\text{comp\_scale\_val}[c]$  is equal to  $\text{comp\_scale\_bit\_depth\_scale\_val}$ .

The variables `CompScaleOffsetOutputVal[ c ][ i ]` and `CompScaleOffsetInputVal[ c ][ i ]` for `i` in the range of 0 to `comp_scale_num_points_minus1[ c ]`, inclusive, is derived as follows:

```

roundingOffset = (comp_scale_log2_denom_scale_val == 0) ? 0 : (1 <<
                    comp_scale_log2_denom_scale_val - 1)
for( i = 0; i <= comp_scale_num_points_minus1[ c ]; i++ )
    if( i == 0 )
        CompScaleOffsetOutputVal[ c ][ i ] =
comp_scale_global_offset_output_val[ c ]
        CompScaleOffsetInputVal[ c ][ i ] =
comp_scale_global_offset_input_val[ c ]
    else
        CompScaleOffsetOutputVal[ c ][ i ] = CompScaleOffsetOutputVal[ c ][ i -
1 ] +
                    (comp_scale_offset_val[ c ][ i - 1 ] □ comp_scale_val[ c ][ i - 1 ]
                    + roundingOffset ) >>
comp_scale_log2_denom_scale_val
        CompScaleOffsetInputVal[ c ][ i ] = CompScaleOffsetInputVal[ c ][ i - 1 ]
+
                    comp_scale_offset_val[ c ][ i - 1 ]

```

In one alternative, `comp_scale_offset_val[ c ][ i ]` is used to directly calculate

`CompScaleOffsetOutputVal[ ][ i ]` and indirectly calculate

`CompScaleOffsetInputVal[ ][ i ]` for `i` in the range of 0 to

`comp_scale_num_points_minus1[ c ]` as follows:

```

for( i = 0; i < comp_scale_num_points_minus1[ c ]; i++ )
    if( i == 0 )
        CompScaleOffsetOutputVal[ c ][ i ] =
comp_scale_global_offset_output_val[ c ]
        CompScaleOffsetInputVal[ c ][ i ] =
comp_scale_global_offset_input_val[ c ]
    else
        CompScaleOffsetInputVal[ c ][ i ] = CompScaleOffsetInputVal[ c ][ i - 1 ]
+

```



$$\begin{aligned}
 & \text{comp\_scale\_offset\_val}[c][i-1] * \\
 & \text{comp\_scale\_val}[c][i-1] \\
 & \quad + \text{roundingOffset} ) >> \\
 & \text{comp\_scale\_log2\_denom\_scale\_val} ) \\
 & \text{CompScaleOffsetOutputVal}[c][i] = \text{CompScaleOffsetOutputVal}[c][i-1] + \\
 & \quad \text{comp\_scale\_offset\_val}[c][i-1]
 \end{aligned}$$

In one alternative,  $\text{comp\_scale\_offset\_val}[c][i]$  for  $i$  in the range of 0 to  $\text{comp\_scale\_num\_points\_minus1}[c]$ , inclusive, are not signaled, and the values of  $\text{comp\_scale\_offset\_val}[c][i]$  are derived based on  $\text{comp\_scale\_num\_points\_minus1}[c]$  equally spaced intervals for which the scale is specified. The value of  $\text{comp\_scale\_offset\_val}[c][i]$  for  $i$  in the range of 0 to  $\text{comp\_scale\_num\_points\_minus1}[c] - 1$ , inclusive, is derived as follows:

$$\begin{aligned}
 \text{comp\_scale\_offset\_val}[c][i] = & ( (1 \ll \text{comp\_scale\_output\_bit\_depth}) - \\
 & \text{comp\_scale\_global\_offset\_output\_val}[c] ) \div \\
 & ( \text{comp\_scale\_num\_points\_minus1}[c] )
 \end{aligned}$$

In another alternative,  $\text{comp\_scale\_offset\_val}[c][i]$  for  $i$  in the range of 0 to  $\text{comp\_scale\_num\_points\_minus1}[c]$  is calculated as follows:

$$\begin{aligned}
 \text{comp\_scale\_offset\_val}[c][i] = & (1 \ll \text{comp\_scale\_output\_bit\_depth}) \div \\
 & ( \text{comp\_scale\_num\_points\_minus1}[c] )
 \end{aligned}$$

In one alternative, instead of signaling  $\text{comp\_scale\_num\_points\_minus1}[c]$ , the number of pivot points is signaled using  $\text{log2\_comp\_scale\_num\_points}[c]$ , where  $(1 \ll \text{log2\_comp\_scale\_num\_points}[c])$  specifies the number of pivot points for the  $c$ -th component.

Alternatively, each of  $\text{comp\_scale\_offset\_val}[c][i]$  and  $\text{comp\_scale\_val}[c][i]$  is signaled as floating point numbers, or as two syntax elements with exponent and mantissa.

In another alternative, signaling of `comp_scale_val[ c ][ i ]` is replaced by `comp_scale_output_point[ c ][ i ]`.

The semantics of rest of the syntax elements are similar to those described in Example 1.

### **Example 3**

**[0190]** This method described in Example 3 is similar to one of the alternatives described in Example 2, with the exception that the component scaling functions are allowed to be updated independently.

**Syntax of the component scaling SEI message**

component_scale_info( payloadSize ) {	Descriptor
<b>comp_scale_id</b>	ue(v)
<b>comp_scale_cancel_flag</b>	u(1)
if( !comp_scale_cancel_flag ) {	
<b>comp_scale_persistence_flag</b>	u(1)
<b>comp_scale_num_comps</b>	ue(v)
<b>comp_scale_input_bit_depth</b>	ue(v)
<b>comp_scale_output_bit_depth</b>	ue(v)
for( c = 0; c < comp_scale_num_comps; c++ ) {	
<b>comp_scale_persist_component_flag[ c ]</b>	u(1)
if( !comp_scale_persist_component_flag[ c ] )	
<b>comp_scale_num_scale_regions[ c ]</b>	ue(v)
<b>comp_scale_global_offset_input_val[ c ]</b>	u(v)
<b>comp_scale_global_offset_output_val[ c ]</b>	u(v)
for( i = 0; i < comp_scale_num_scale_regions[ c ]; i++ ) {	
<b>comp_scale_offset_val[ c ][ i ]</b>	u(v)
<b>comp_scale_val[ c ][ i ]</b>	u(v)
}	
}	
}	
}	
}	

**Semantics of the component scaling SEI message**

The semantics is similar to Example 2, except for the following syntax elements.

**comp\_scale\_num\_scale\_regions[ c ]** specifies the number of regions for which the syntax element **comp\_scale\_val[ c ][ i ]** is signalled for the c-th component.

**comp\_scale\_num\_scale\_regions[ c ]** shall be in the range of 0 to  $(1 \ll \text{comp\_scale\_input\_bit\_depth}) - 1$ , inclusive.

**comp\_scale\_persist\_component\_flag[ c ]** equal to 0 specifies that component scaling parameters for the c-th component are explicitly signalled in the SEI message.

**comp\_scale\_persist\_component\_flag[ c ]** equal to 1 specifies that component scaling parameters for the c-th component are not explicitly signalled in the SEI message, and it

persists from the component scaling parameters of the c-th component of the component scaling SEI message that applies to previous picture, in output order.

It is a requirement of bitstream conformance that when the component scaling SEI message is present in an IRAP access unit, the value of `comp_scale_persist_component_flag[ c ]`, when present, shall be equal to 0.

Alternatively, the following condition is added:

It is a requirement of bitstream conformance that when the component scaling SEI message is present in an access unit that is not an IRAP access unit and `comp_scale_persist_component_flag[ c ]` is equal to 1, then there is at least one picture that precedes the current picture in output order and succeeds, in output order, the previous IRAP picture in decoding order, inclusive, such that the one picture is associated with a component scaling SEI message with `comp_scale_persistence_flag` equal to 1.

**comp\_scale\_persistence\_flag** specifies the persistence of the component scaling information SEI message for the current layer.

`comp_scale_persistence_flag` equal to 0 specifies that the component scaling information applies to the current decoded picture only.

Let `picA` be the current picture `comp_scale_persistence_flag` equal to 1 specifies that the component scaling information of the c-th component persists for the current layer in output order until any of the following conditions are true:

- A new CLVS of the current layer begins.
- The bitstream ends.
- A picture `picB` in the current layer in an access unit containing a component scaling information SEI message with the same value of `comp_scale_id` and `comp_scale_persist_component_flag[ c ]` equal to 0, and applicable to the current layer is output for which `PicOrderCnt( picB )` is greater than `PicOrderCnt( picA )`, where `PicOrderCnt( picB )` and `PicOrderCnt( picA )` are the `PicOrderCntVal` values of `picB` and `picA`, respectively, immediately after the invocation of the decoding process for picture order count for `picB`.

#### Example 4

**[0191]** In this Example 4, a different method to signal the scale regions is disclosed.

**Changes to component scaling SEI message syntax**

component_scale_info( payloadSize ) {	Descriptor
<b>comp_scale_id</b>	ue(v)
<b>comp_scale_cancel_flag</b>	u(1)
if( !comp_scale_cancel_flag ) {	
<b>comp_scale_persistence_flag</b>	u(1)
<b>comp_scale_num_comps</b>	ue(v)
<b>comp_scale_input_bit_depth</b>	ue(v)
<b>comp_scale_output_bit_depth</b>	ue(v)
for( c = 0; c < comp_scale_num_comps; c++ ) {	
<b>comp_scale_persist_component_flag[ c ]</b>	u(1)
if( !comp_scale_persist_component_flag[ c ] )	
<b>comp_scale_global_offset_input_val[ c ]</b>	u(v)
<b>comp_scale_global_offset_output_val[ c ]</b>	u(v)
<b>comp_scale_num_scale_regions[ c ]</b>	ue(v)
for( i = 0; i < comp_scale_num_scale_regions[ c ]; i++ ) {	
<b>comp_scale_offset_begin_val[ c ][ i ]</b>	u(v)
<b>comp_scale_offset_end_val[ c ][ i ]</b>	u(v)
<b>comp_scale_val[ c ][ i ]</b>	u(v)
}	
}	
}	
}	
}	

**Changes to component scaling SEI message semantics**

The semantics of the syntax elements are similar to those described in previous examples, except for the following:

**comp\_scale\_offset\_begin\_val[ c ][ i ]** specifies the beginning of the sample value range for which the scale value **comp\_scale\_val[ c ][ i ]** is applicable. The number of bits used to represent **comp\_scale\_offset\_begin\_val[ c ]** is equal to **comp\_scale\_input\_bit\_depth**.

**comp\_scale\_offset\_end\_val[ c ][ i ]** specifies the end of the sample value range for which the scale value **comp\_scale\_val[ c ][ i ]** is applicable. The number of bits used to represent **comp\_scale\_offset\_end\_val[ c ]** is equal to **comp\_scale\_input\_bit\_depth**. For regions that are not explicitly specified by **comp\_scale\_offset\_begin\_val** and **comp\_scale\_offset\_end\_val**, the **comp\_scale\_value[ c ][ i ]** for those regions is inferred to be equal to 0.

**[0192]** Alternatively, **comp\_scale\_offset\_end\_val[ c ][ i ]** is not signaled and instead the difference between **comp\_scale\_offset\_end\_val[ c ][ i ]** and **comp\_scale\_offset\_begin\_val[ c ][ i ]** is signaled, and the value of **comp\_scale\_offset\_end\_val[ c ][ i ]** derived at the decoder-side.

**[0193]** In another alternative, the total number of regions in to which the output sample range is split is specified, and the number of regions is signaled for which the scale regions are explicitly signaled.

...	u(v)
<b>comp_scale_global_offset_output_val[ c ]</b>	u(v)
<b>comp_scale_tot_scale_regions[ c ]</b>	ue(v)
<b>comp_scale_num_scale_regions[ c ]</b>	ue(v)
for( i = 0; i < comp_scale_num_scale_regions[ c ]; i++ ) {	
<b>comp_scale_region_idx[ c ][ i ]</b>	u(v)
<b>comp_scale_val[ c ][ i ]</b>	u(v)
}	
...	

**comp\_scale\_tot\_scale\_regions[ c ]** specifies the total number of equal length sample value ranges in to which the sample values are split. The number of bits used to represent **comp\_scale\_tot\_scale\_regions[ c ]** is equal to **comp\_scale\_input\_bit\_depth**. In one alternative, the **comp\_scale\_tot\_scale\_regions[ c ]** sample value ranges may not be exactly equal in length but very nearly equal to account for the integer accuracy of the region lengths.

**comp\_scale\_region\_idx[ c ][ i ]** specifies the index of the sample value range for which the scale value **comp\_scale\_val[ c ][ i ]** is applied. The length of the syntax element **comp\_scale\_region\_idx[ c ]** is  $\text{Ceil}(\text{Log}_2(\text{comp\_scale\_tot\_scale\_regions}[c]))$  bits.

### Alternatives

Alternatively, region around the chroma neutral (511 for 10-bit data) have smaller size, p.e., half the size of the other regions.

### Example 5

#### Syntax of the component scale SEI message

component_scale_info( payloadSize ) {	Descriptor
<b>comp_scale_id</b>	ue(v)
<b>comp_scale_cancel_flag</b>	u(1)
if( !comp_scale_cancel_flag ) {	
<b>comp_scale_persistence_flag</b>	u(1)
<b>comp_scale_scale_bit_depth</b>	u(4)
<b>comp_scale_offset_bit_depth</b>	u(4)
<b>comp_scale_scale_frac_bit_depth</b>	u(4)
<b>comp_scale_offset_frac_bit_depth</b>	u(4)
<b>comp_scale_num_comps_minus1</b>	ue(v)
for( c = 0; c <= comp_scale_num_comps_minus1; c++ ) {	
<b>comp_scale_num_ranges[ c ]</b>	ue(v)
<b>comp_scale_equal_ranges_flag[ c ]</b>	u(1)
<b>comp_scale_global_offset_val[ c ]</b>	u(v)
for( i = 0; i <= comp_scale_num_ranges[ c ]; i++ )	
<b>comp_scale_scale_val[ c ][ i ]</b>	u(v)
if( !comp_scale_equal_ranges[ c ] )	u(v)
for( i = 0; i <= comp_scale_num_ranges[ c ]; i++ )	
<b>comp_scale_offset_val[ c ][ i ]</b>	u(v)
}	
}	

#### Semantics of the component scale SEI message

The component scaling SEI message provides information to perform scaling operations on the various components of the decoded pictures. The colour space and the components on which the scaling operations should be performed are determined by the value of the syntax elements signalled in the SEI message.

**comp\_scale\_id** contains an identifying number that may be used to identify the purpose of the component scaling SEI message. The value of **comp\_scale\_id** shall be in the range of 0 to  $2^{32} - 2$ , inclusive. The value of **comp\_scale\_id** may be used to specify the colour space at which the component scaling SEI message, or whether the component scaling SEI message is applied in the linear or the non-linear domain.

In some examples, **comp\_scale\_id** can specify the configuration of the HDR reconstruction process. In some examples, particular value of **comp\_scale\_id** may be associated with signaling of scaling parameters for 3 components. The scaling of the first components to be applied to samples of R', G', B' color space, and parameters of following 2 components are applied for scaling of Cr and Cb.

For yet another **comp\_scale\_id** value, hdr reconstruction process can utilize parameters for 3 components, and scaling is applied to samples of Luma, Cr and Cb color components.

In yet another **comp\_scale\_id** value, hdr reconstruction process can utilize signaling for 4 components, 3 of which to be applied to Luma, Cr and Cb scaling, and 4th component to bring parameters of color correction.

In some examples, certain range of **comp\_scale\_id** values may be associated with HDR reconstruction conducted in SDR-backward compatible configuration, whereas another range of **comp\_scale\_id** values may be associated with HDR reconstruction conducted to non-backward compatible configuration.

Values of **comp\_scale\_id** from 0 to 255, inclusive, and from 512 to  $2^{31} - 1$ , inclusive, may be used as determined by the application. Values of **comp\_scale\_id** from 256 to 511, inclusive, and from  $2^{31}$  to  $2^{32} - 2$ , inclusive, are reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore all component scale information SEI messages containing a value of **comp\_scale\_id** in the range of 256 to 511, inclusive, or in the range of  $2^{31}$  to  $2^{32} - 2$ , inclusive, and bitstreams shall not contain such values.

NOTE 1 – The **comp\_scale\_id** can be used to support component scaling processes that are suitable for different display scenarios. For example, different values of **comp\_scale\_id** may correspond to different display bit depths or different colour spaces in which the scaling is applied.

Alternatively, the **comp\_scale\_id** may also be used to identify whether the scaling is performed for compatibility to certain types of displays or decoder, e.g. HDR, SDR.

**comp\_scale\_cancel\_flag** equal to 1 indicates that the component scaling information SEI message cancels the persistence of any previous component information SEI



messages in output order that applies to the current layer. `comp_scale_cancel_flag` equal to 0 indicates that component scaling information follows.

**`comp_scale_persistence_flag`** specifies the persistence of the component scaling information SEI message for the current layer.

`comp_scale_persistence_flag` equal to 0 specifies that the component scaling information applies to the current decoded picture only.

Let `picA` be the current picture. `comp_scale_persistence_flag` equal to 1 specifies that the component scaling information persists for the current layer in output order until any of the following conditions are true:

- A new CLVS of the current layer begins.
- The bitstream ends.
- A picture `picB` in the current layer in an access unit containing a component scaling information SEI message with the same value of `comp_scale_id` and applicable to the current layer is output for which `PicOrderCnt( picB )` is greater than `PicOrderCnt( picA )`, where `PicOrderCnt( picB )` and `PicOrderCnt( picA )` are the `PicOrderCntVal` values of `picB` and `picA`, respectively, immediately after the invocation of the decoding process for picture order count for `picB`.

**`comp_scale_scale_bit_depth`** specifies the number of bits used to signal the syntax element `comp_scale_scale_val[ c ][ i ]`. The value of `comp_scale_scale_bit_depth` shall be in the range of 0 to 15, inclusive.

**`comp_scale_offset_bit_depth`** specifies the number of bits used to signal the syntax elements `comp_scale_global_offset_val[ c ]` and `comp_scale_offset_val[ c ][ i ]`. The value of `comp_scale_offset_bit_depth` shall be in the range of 0 to 15, inclusive.

**`comp_scale_scale_frac_bit_depth`** specifies the number of LSBs used to indicate the fractional part of the scale parameter of the *i*-th partition of the *c*-th component. The value of `comp_scale_scale_frac_bit_depth` shall be in the range of 0 to 15, inclusive. The value of `comp_scale_scale_frac_bit_depth` shall be less than or equal to the value of `comp_scale_scale_bit_depth`.

**`comp_scale_offset_frac_bit_depth`** specifies the number of LSBs used to indicate the fractional part of the offset parameter of the *i*-th partition of the *c*-th component and global offset of the *c*-th component. The value of `comp_scale_offset_frac_bit_depth` shall be in the range of 0 to 15, inclusive. The value of `comp_scale_offset_frac_bit_depth` shall be less than or equal to the value of `comp_scale_offset_bit_depth`.

**comp\_scale\_num\_comps\_minus1** plus 1 specifies the number of components for which the component scaling function is specified. **comp\_scale\_num\_comps\_minus1** shall be in the range of 0 to 2, inclusive.

**comp\_scale\_num\_ranges[ c ]** specifies the number of ranges in to which the output sample range is partitioned in to. The value of **comp\_scale\_num\_ranges[ c ]** shall be in the range of 0 to 63, inclusive..

**comp\_scale\_equal\_ranges\_flag[ c ]** equal to 1 indicates that that output sample range is partitioned into **comp\_scale\_num\_ranges[ c ]** nearly equal partitions, and the partition widths are not explicitly signalled. **comp\_scale\_equal\_ranges\_flag[ c ]** equal to 0 indicates that that output sample range may be partitioned into **comp\_scale\_num\_ranges[ c ]** partitions not all of which are of the same size, and the partitions widths are explicitly signalled.

**comp\_scale\_global\_offset\_val[ c ]** is used to derive the offset value that is used to map the smallest value of the valid input data range for the c-th component. The length of **comp\_scale\_global\_offset\_val[ c ]** is **comp\_scale\_offset\_bit\_depth** bits.

**comp\_scale\_scale\_val[ c ][ i ]** is used to derive the offset value that is used to derive the width of the of the i-th partition of the c-th component. The length of **comp\_scale\_global\_offset\_val[ c ]** is **comp\_scale\_offset\_bit\_depth** bits.

The variable **CompScaleScaleVal[ c ][ i ]** is derived as follows :

$$\begin{aligned} \text{CompScaleScaleVal}[c][i] = & (\text{comp\_scale\_scale\_val}[c][i] \gg \\ & \text{comp\_scale\_scale\_frac\_bit\_depth}) + \\ & (\text{comp\_scale\_scale\_val}[c][i] \& \\ & ((1 \ll \text{comp\_scale\_scale\_frac\_bit\_depth}) - 1) \\ & ) \\ & (1 \ll \text{comp\_scale\_scale\_frac\_bit\_depth}) \end{aligned}$$

**comp\_scale\_offset\_val[ c ][ i ]** is used to derive the offset value that is used to derive the width of the of the i-th partition of the c-th component. The length of **comp\_scale\_global\_offset\_val[ c ]** is **comp\_scale\_offset\_bit\_depth** bits.

When **comp\_scale\_offset\_val[ c ][ i ]** is signalled, the value of

**CompScaleOffsetVal[ c ][ i ]** is derived as follows:

$$\begin{aligned} \text{CompScaleOffsetVal}[c][i] = & (\text{comp\_scale\_offset\_val}[c][i] \gg \\ & \text{comp\_scale\_offset\_frac\_bit\_depth}) + \\ & (\text{comp\_scale\_offset\_val}[c][i] \& \\ & ((1 \ll \text{comp\_scale\_offset\_frac\_bit\_depth}) - 1) \end{aligned}$$

)) ÷

$$(1 \ll \text{comp\_scale\_offset\_frac\_bit\_depth})$$

Alternatively, the variable  $\text{CompScaleScaleVal}[c][i]$  and

$\text{CompScaleOffsetVal}[c][i]$  are derived as follows :

$$\text{CompScaleScaleVal}[c][i] = \text{comp\_scale\_scale\_val}[c][i] \&$$

$$(1 \ll \text{comp\_scale\_scale\_frac\_bit\_depth})$$

$$\text{CompScaleOffsetVal}[c][i] = \text{comp\_scale\_offset\_val}[c][i] \div$$

$$(1 \ll \text{comp\_scale\_offset\_frac\_bit\_depth})$$

When  $\text{comp\_scale\_equal\_ranges\_flag}[c]$  is equal to 1,  $\text{comp\_scale\_offset\_val}[c][i]$  is not signalled, and the value of  $\text{CompScaleOffsetVal}[c][i]$  is derived as follows:

$$\text{CompScaleOffsetVal}[c][i] = 1 \div \text{comp\_scale\_num\_ranges}[c]$$

The variable  $\text{CompScaleOutputRanges}[c][i]$  and  $\text{CompScaleInputRanges}[c][i]$  for  $i$  in the range of 0 to  $\text{comp\_scale\_num\_ranges}[c]$  is derived as follows:

for(  $i = 0$ ;  $i \leq \text{comp\_scale\_num\_ranges}[c]$ ;  $i++$  )

if(  $i == 0$  )

$$\text{CompScaleOutputRanges}[c][i] = \text{comp\_scale\_global\_offset\_val}[c] \div$$

$$(1 \ll \text{comp\_scale\_offset\_frac\_bit\_depth})$$

$$\text{CompScaleInputRanges}[c][i] = 0$$

else

$$\text{CompScaleInputRanges}[c][i] = \text{CompScaleOffsetInputRanges}[c][i -$$

1] +

$$(\text{CompScaleOffsetVal}[c][i - 1] *$$

$$\text{CompScaleScaleVal}[c][i - 1]$$

$$\text{CompScaleOutputRanges}[c][i] = \text{CompScaleOutputRanges}[c][i - 1] +$$

$$\text{CompScaleOffsetVal}[c][i - 1]$$

In one alternative, the values of  $\text{CompScaleOutputRanges}[c][i]$  and

$\text{CompScaleInputRanges}[c][i]$  are derived as follows:

for(  $i = 0$ ;  $i \leq \text{comp\_scale\_num\_ranges}[c]$ ;  $i++$  )

if(  $i == 0$  )

$$\text{CompScaleInputRanges}[c][i] = \text{comp\_scale\_global\_offset\_val}[c] \div$$

$$(1 \ll \text{comp\_scale\_offset\_frac\_bit\_depth})$$

$$\text{CompScaleOutputRanges}[c][i] = 0$$

else

$$\text{CompScaleInputRanges}[c][i] = \text{CompScaleOffsetInputRanges}[c][i -$$

1 ] +

(CompScaleOffsetVal[ c ][ i - 1 ] \*

CompScaleScaleVal[ c ][ i - 1 ]

CompScaleOutputRanges[ c ][ i ] = CompScaleOutputRanges[ c ][ i - 1 ] +

CompScaleOffsetVal[ c ][ i - 1 ]

The process of mapping an input signal representation (which may be used to cover both integer as well as floating point)  $x$  and an output signal representation  $y$ , where the sample values for both input representation is normalized in the range of 0 to 1, and output representation is in the range of and 0 to 1, is specified as follows:

if(  $x \leq \text{CompScaleInputRanges}[c][0]$  )

$y = \text{CompScaleOutputRanges}[c][0]$

else if(  $x > \text{CompScaleInputRanges}[c][\text{comp\_scale\_num\_ranges}[c]]$  )

$y = \text{CompScaleOutputRanges}[c][\text{comp\_scale\_num\_ranges}[c]; ]$

else

for(  $i = 1; i \leq \text{comp\_scale\_num\_ranges}[c]; i++$  )

if(  $\text{CompScaleInputRanges}[i - 1] < x \ \&\& \ x \leq$

$\text{CompScaleInputRanges}[i]$  )

$y = (x -$

$\text{CompScaleInputRanges}[i - 1]) \div \text{comp\_scale\_val}[c][i] +$

$\text{CompScaleOutputRanges}[c][i - 1]$

In one alternative, the value of  $\text{CompScaleOutputRanges}[c][0]$  is set based on the permitted sample value range.

Alternatively, the process of mapping an input value  $\text{valIn}$  to output value  $\text{valOut}$  is defined as follows:

$m\_pAtfRangeIn[0] = 0;$

$m\_pAtfRangeOut[0] = -m\_offset2 * m\_pAtfScale2[c][0];$

for(  $\text{int } j = 1; j < m\_atfNumberRanges + 1; j++$  )

{

$m\_pAtfRangeIn[j] = m\_pAtfRangeIn[j - 1] + m\_pAtfDelta[j - 1];$

$m\_pAtfRangeOut[j] = m\_pAtfRangeOut[j - 1] + m\_pAtfScale2[c][j -$

$1] *$

$m\_pAtfDelta[j - 1];$

}

```

    for (int j = 0; j < numRanges && skip == 0; j++)
    {
        if (valIn <= pAtfRangeIn[ j + 1 ])
        {
            valOut = (valIn - pOffset[component][ j ]) *
pScale[ component ][ j ];
            skip = 1;
        }
    }
]

```

In one alternative, `m_offset2` is equal to `comp_scale_global_offset_val[ c ] ÷ ( 1 << comp_scale_offset_frac_bit_depth )`, `m_pAtfScale[ c ][ i ]` is equal to `CompScaleScaleVal[ c ][ i ]` and `m_pAtfDelta[ i ]` is equal to `CompScaleOffsetVal[ c ][ i ]` for the `c`-th component, and `pScale` and `pOffset` are scale and offset parameter derived from `m_AtScale` and `m_pAtfDelta`. An inverse operation would be defined accordingly.

### Example 6

In some examples, some of signaling methods described above, e.g. in example 5, can be utilized as shown in following pseudo code.

`m_atfNumberRanges` is a term for syntax elements `comp_scale_num_ranges[ c ]` for a given `c`, that specifies number of dynamic range partitioning for mapped data.

`m_pAtfRangeIn` is a term for `CompScaleInputRanges`, is an arrays size of `m_atfNumberRanges+1` that includes input sample value specifying the border between two concatenated partitions, e.g., `i` and `i+1`.

`m_pAtfRangeOut` is a term for `CompScaleOutputRanges`, is an arrays size of `m_atfNumberRanges+1` that includes output sample value specifying the border between two concatenated partitions, e.g. `i` and `i+1`.

`m_pAtfScale2` is a term for variable `CompScaleScaleVal[ c ]` is an arrays size of `m_atfNumberRanges` that includes scale values for each partitions.

`m_pAtfOffset2` is an array arrays size of `m_atfNumberRanges` that includes offset values for each partition.

`m_offset2` is a term for `comp_scale_global_offset_val`.

In this example, parameters of piece-wise linear model can be determined form syntax elements as in Algorithm 1:

Algorithm 1:

```

    m_pAtfRangeIn[0] = 0;
    m_pAtfRangeOut[0] = -m_offset2 * m_pAtfScale2[c][0];
    for (int j = 1; j < m_atfNumberRanges + 1; j++)
    {
        m_pAtfRangeIn[j] = m_pAtfRangeIn[j - 1] + m_pAtfDelta[j - 1];
        m_pAtfRangeOut[j] = m_pAtfRangeOut[j - 1] + m_pAtfScale2[c][j - 1]
* m_pAtfDelta[j - 1];
    }

    for (int j = 0; j < m_atfNumberRanges; j++)
    {
        temp = m_pAtfRangeIn[j + 1] - m_pAtfRangeOut[j + 1] /
m_pAtfScale2[c][j];
        m_pAtfOffset2[c][j] = temp;
    }

```

Once determined, piece-wise linear model can be applied to input samples value inValue to determine output sample value outValue as in Algorithm 2:

Algorithm 2:

```

    for (int j = 0; j < m_atfNumberRanges && skip == 0; j++)
    {
        if (inValue <= m_pAtfRangeIn[j + 1])
        {
            outValue = (inValue - m_pAtfOffset2 [j]) *
m_pAtfScale2 [j];
            skip = 1;
        }
    }

```

Inverse process to be conducted as in Algorithm 3:

Algorithm 3:

```

    for (int j = 0; j < m_atfNumberRanges && skip == 0; j++)
    {

```

```

        if (inValue <= m_pAtfRangeOut[j + 1])
        {
            outValue = inValue / m_pAtfScale2[j] + m_pAtfOffset2
[j];

            skip = 1;
        }
    }

```

**[0194]** In some examples, border sample value (an entry of m\_pAtfRangeIn or m\_pAtfRangeOut) between two concatenated partitions i and i+1 can be interpreted differently, as belonging to i+1, instead of belonging to i partition as it is shown in Algorithm 2 and 3.

**[0195]** In some examples, inverse process shown in Algorithm 3, can be implemented with a multiplication by m\_pAtfInverseScale2 value, instead of division by m\_pAtfScale2[j]. In such examples, a value of m\_pAtfScale2[j] is determined from m\_pAtfScale2[j] in advance.

**[0196]** In some examples, m\_pAtfInverseScale2[j] is determined at the decoder side as  $1/m\_pAtfScale2[j]$ .

**[0197]** In some examples, m\_pAtfInverseScale2[j] can be computed at the encoder side, and signalled through bitstream. In such examples, operation given in Algorithms 1, 2 and 3 will be adjusted accordingly.

**[0198]** Various examples

**[0199]** In some examples, proposed signaling mechanism can be used to model a piece-wise function that can be utilized to enable dynamical range adjustment for samples of input signal, e.g. to improve compression efficiency of video coding systems.

**[0200]** In some examples, proposed signaling mechanism can be used to model a piece-wise function that can be applied to codewords (non-linear representation of R,G,B samples) produced by an OETF, e.g. by PQ TF of ST.2084, or others.

**[0201]** In some examples, proposed signaling mechanism can be used to model a piece-wise function that can be applied to samples of YCbCr color representation.

**[0202]** In some examples, proposed signaling mechanism can be used to model a piece-wise function that can be utilized to HDR/WCG solutions with SDR compatibility.

**[0203]** In some examples, proposed signaling mechanism can be used to model a piece-wise function that can be applied to samples in floating point representation. In yet

another example, proposed signaling mechanism and resulting function can be applied to samples in integer representation, e.g. 10 bits.

**[0204]** In some examples, proposed signaling mechanism can be used to model a piece-wise function that can be applied to samples in a form of Look Up Tables. In yet another examples, proposed signaling can be used to model function that can be applied to a sample in a form of multiplier.

#### **Combinations and Extensions**

**[0205]** In the examples above, a linear model is assumed for each region (i.e., scale plus offset); the techniques of this disclosure also may be applicable for higher-order polynomial models, for example, with a polynomial of 2nd degree requiring three parameters instead of two. The signaling and syntax would be properly extended for this scenario.

**[0206]** Combinations of aspects described above are possible and part of the techniques of this disclosure.

**[0207]** Toolbox combination: there are several HDR methods that can target somewhat similar goals to those of the SEIs described in this disclosure. In order to accommodate more than one of them but, at the same time, limiting the number of applicable SEI processing per frame, it is proposed to combine (one or more of) these methods in a single SEI. A proposed syntax element would indicate the specific method to apply in each instance. For example, if there are two possible methods in the SEI, the syntax element would be a flag indicating the one to be used.

#### **Example 7**

**[0208]** In this example, the signaling of scale parameters is modified such that negative scales can be transmitted, and the signaled scale parameters indicate the variation of scale to be applied for different ranges of the various components. The changes with respect to example 5 are below.



**Changes to syntax of the SEI message**

component_scale_info( payloadSize ) {	Descriptor
<b>comp_scale_id</b>	ue(v)
<b>comp_scale_cancel_flag</b>	u(1)
if( !comp_scale_cancel_flag ) {	
<b>comp_scale_persistence_flag</b>	u(1)
<b>comp_scale_scale_bit_depth</b>	u(4)
<b>comp_scale_offset_bit_depth</b>	u(4)
<b>comp_scale_scale_frac_bit_depth</b>	u(4)
<b>comp_scale_offset_frac_bit_depth</b>	u(4)
<b>comp_scale_negative_scales_present_flag</b>	u(1)
<b>comp_scale_dep_component_id</b>	ue(v)
<b>comp_scale_num_comps_minus1</b>	ue(v)
for( c = 0; c <= comp_scale_num_comps_minus1; c++ ) {	
<b>comp_scale_num_ranges[ c ]</b>	ue(v)
<b>comp_scale_equal_ranges_flag[ c ]</b>	u(1)
<b>comp_scale_global_offset_val[ c ]</b>	u(v)
for( i = 0; i <= comp_scale_num_ranges[ c ]; i++ )	
<b>comp_scale_scale_val[ c ][ i ]</b>	u(v)
if( !comp_scale_equal_ranges[ c ] )	u(v)
for( i = 0; i <= comp_scale_num_ranges[ c ]; i++ )	
<b>comp_scale_offset_val[ c ][ i ]</b>	u(v)
}	
}	

**Changes to semantics of the SEI message**

**comp\_scale\_negative\_scales\_present\_flag** equal to 1 specifies that the integer part of the scale parameters derived from `comp_scale_scale_val[ c ][ i ]` is represented as a signed integer. **comp\_scale\_negative\_scales\_present\_flag** equal to 0 specifies that the integer part scale parameters derived from `comp_scale_scale_val[ c ][ i ]` is represented as an unsigned integer.

[0209] In one alternative, another set of offset parameters are signaled along with `comp_scale_scale_val` that are used to define the offset that is applied along with the scale on a first component as a function of the value of a second component.

[0210] The signed-integer representation includes, but is not limited to, twos-complement notation and signed magnitude representation (one bit for sign and the remaining bits in the integer-part). The derivation below is given for the signed magnitude representation. The derivation can be similarly defined for other forms of signed representations.

The variable `CompScaleScaleVal[ c ][ i ]` is derived as follows :

```
compScaleScaleFracPart = ( comp_scale_scale_val[ c ][ i ] &
                          ( (1 << comp_scale_scale_frac_bit_depth) - 1 )
                          )
```

```
(1 << comp_scale_scale_frac_bit_depth)
```

```
if( comp_scale_negative_scales_present_flag ) {
```

```
    compScaleSignPart = comp_scale_scale_val[ c ][ i ] >>
```

```
(comp_scale_scale_bit_depth - 1)
```

```
    compScaleIntegerPart = comp_scale_scale_val[ c ][ i ] - ( compScaleSignPart
                                                             << (comp_scale_scale_bit_depth - 1) )
```

```
    compScaleIntegerVal = ( ( compScaleSignPart == 1 ) : -1 : 1 ) *
```

```
compScaleIntegerPart
```

```
} else
```

```
    compScaleIntegerVal = comp_scale_scale_val[ c ][ i ] >>
```

```
comp_scale_scale_frac_bit_depth
```

```
CompScaleScaleVal[ c ][ i ] = compScaleIntegerVal + compScaleScaleFracPart
```

It is a requirement of bitstream conformance that when

`comp_scale_negative_scale_present_flag` is equal to 1, the value of

`comp_scale_scale_bit_depth` shall be greater than or equal to

`comp_scale_scale_frac_bit_depth`

**comp\_scale\_dependent\_component\_id** specifies the application of scale and offset parameters to the various components of the video. When

`comp_scale_dependent_component_id` is equal to 0, the syntax elements

`comp_scale_global_offset_val[ c ]`, `comp_scale_scale_val[ c ][ i ]` and

`comp_scale_offset_val[ c ][ i ]` are used to identify mapping of input and output values of the *c*-th component. When `comp_scale_dependent_component_id` is greater than 0,

$\text{comp\_scale\_dependent\_component\_id} - 1$  specifies the index of the component such that the syntax elements  $\text{comp\_scale\_global\_offset\_val}[c]$ ,  $\text{comp\_scale\_scale\_val}[c][i]$  and  $\text{comp\_scale\_offset\_val}[c][i]$  specify the mapping of a scale parameter to be applied to the  $c$ -th component of a sample as a function of the value of  $(\text{comp\_scale\_dependent\_component\_id} - 1)$ -th component of the sample.

The rest of the semantics are similar to those described in Example 5.

### Example 8

**[0211]** In this example, the bit depth of the ATF parameters depend on the component. For each component, the bit depth of the syntax elements is explicitly signaled. In addition, there are default bit-depth for those syntax elements. The default value is assigned when the bit depth is not explicitly signaled. A flag might indicate whether the default values are applied or they are explicitly signaled.

**[0212]** The table below shows an example of these concepts. Syntax elements of the ATF parameters are the scale  $\text{hdr\_recon\_scale\_val}[]$  and range  $\text{hdr\_recon\_range\_val}[]$ . The syntax elements indicating the corresponding bit depth (integer and fractional part) are the following ones:

- $\text{hdr\_recon\_scale\_bit\_depth}[c]$ ,
- $\text{hdr\_recon\_offset\_bit\_depth}[c]$ ,
- $\text{hdr\_recon\_scale\_frac\_bit\_depth}[c]$ ,
- $\text{hdr\_recon\_offset\_frac\_bit\_depth}[c]$ ,

where  $c$  is the component index. The default bit-depths for scale and offset (range) can be set to:

- $\text{hdr\_recon\_scale\_bit\_depth}[c] = 8$ ,
- $\text{hdr\_recon\_offset\_bit\_depth}[c] = 8$ ,
- $\text{hdr\_recon\_scale\_frac\_bit\_depth}[c] = 6$ ,
- $\text{hdr\_recon\_offset\_frac\_bit\_depth}[c] = 8$ .

**[0213]** The accuracy of the parameters might also be different for the ATF parameters and the color adjustment parameters. Also, the default might be different per component and for the color adjustment parameters. In this example, the defaults are assumed to be the same.

hdr_reconstruction_info( payloadSize ) {	<b>Descriptor</b>
<b>hdr_recon_id</b>	ue(v)
<b>hdr_recon_cancel_flag</b>	u(1)
if( !hdr_recon_cancel_flag ) {	
<b>hdr_recon_persistence_flag</b>	u(1)
if (hdr_recon_id == 1 ) {	
<b>hdr_output_full_range_flag</b>	
<b>hdr_output_colour primaries</b>	
<b>hdr_output_transfer_characteristics</b>	
<b>hdr_output_matrix_coeffs</b>	
}	
SYNTAX FOR THE MAPPING LUTs	
<b>hdr_recon_num_comps_minus1</b>	ue(v)
for( c = 0; c <= hdr_recon_num_comps_minus1; c++ ) {	
<b>hdr_recon_default_bit_depth [ c ]</b>	u(1)
if ( hdr_recon_default_bit_depth [ c ] == 0 ) {	
<b>hdr_recon_scale_bit_depth[ c ]</b>	u(4)
<b>hdr_recon_offset_bit_depth[ c ]</b>	u(4)
<b>hdr_recon_scale_frac_bit_depth[ c ]</b>	u(4)
<b>hdr_recon_offset_frac_bit_depth[ c ]</b>	u(4)
}	
<b>hdr_recon_num_ranges[ c ]</b>	ue(v)
<b>hdr_recon_equal_ranges_flag[ c ]</b>	u(1)
<b>hdr_recon_global_offset_val[ c ]</b>	u(v)
for( i = 0; i <= hdr_recon_num_ranges[ c ]; i++ )	
<b>hdr_recon_scale_val[ c ][ i ]</b>	u(v)
if( !hdr_recon_equal_ranges[ c ] )	u(v)
for( i = 0; i <= hdr_recon_num_ranges[ c ]; i++ )	
<b>hdr_recon_range_val [ c ][ i ]</b>	u(v)
}	u(v)

SYNTAX FOR THE COLOR CORRECTION PART	
if (hdr_recon_id == 1 ) {	Params related to Colour correction
hdr_color_correction_type	0: on U,V - 1: on R,G,B
hdr_color_accuracy_flag	<b>Syntax for coding the colour</b>
if( ! hdr_recon_color_accuracy_flag ) {	<b>correction LUT</b>
hdr_color_scale_bit_depth	u(4)
hdr_color_offset_bit_depth	u(4)
hdr_color_scale_frac_bit_depth	u(4)
hdr_color_offset_frac_bit_depth	u(4)
}	
color_correction_num_ranges	
color_correction_equal_len_ranges_flag	
color_correction_zero_offset_val	
for( i = 0; i < color_correction_num_ranges; i++ )	
color_correction_scale_val[ i ]	
if( ! color_correction_equal_len_ranges_flag )	
for( i = 0; i < color_correction_num_ranges; i++ )	
color_correction_range_val[ i ]	
}	
}	
}	
}	

**Example 9**

**[0214]** A desirable property of a new HDR solution is that it is backward compatible to previous HDR solutions, like HDR10. A syntax element may indicate that this is the case. This indicates a characteristic of the bitstream, and an HDR decoder might decide not to spend computational resources on the inverse ATF processing under some circumstances if the non ATF version is already viewable.

[0215] In one example, some values of the `hdr_recon_id` syntax element are reserved to indicate HDR10 backward compatibility, or to what degree there is backward compatibility.

[0216] In another example, a flag (`hdr_recon_hdr10_bc`) indicates this situation.

[0217] In one example, the signaled HDR10 backward compatibility indicates that the bitstream is viewable. Alternatively, it might indicate some specific properties of the signaled values: for example, that they are a range of values that guarantees this property. For instance, a constraint could be that the scale is between 0.9 and 1.1.

[0218] FIG. 10 is a block diagram illustrating an example of video encoder 20 that may implement the techniques of this disclosure. Video encoder 20 may perform intra- and inter-coding of video blocks within video slices in a target color container that have been processed by video pre-processor unit 19. Intra-coding relies on spatial prediction to reduce or remove spatial redundancy in video within a given video frame or picture. Inter-coding relies on temporal prediction to reduce or remove temporal redundancy in video within adjacent frames or pictures of a video sequence. Intra-mode (I mode) may refer to any of several spatial based coding modes. Inter-modes, such as uni-directional prediction (P mode) or bi-prediction (B mode), may refer to any of several temporal-based coding modes.

[0219] As shown in FIG. 10, video encoder 20 receives a current video block within a video frame to be encoded. In the example of FIG. 10, video encoder 20 includes mode select unit 40, a video data memory 41, decoded picture buffer 64, summer 50, transform processing unit 52, quantization unit 54, and entropy encoding unit 56. Mode select unit 40, in turn, includes motion compensation unit 44, motion estimation unit 42, intra prediction processing unit 46, and partition unit 48. For video block reconstruction, video encoder 20 also includes inverse quantization unit 58, inverse transform processing unit 60, and summer 62. A deblocking filter (not shown in FIG. 10) may also be included to filter block boundaries to remove blockiness artifacts from reconstructed video. If desired, the deblocking filter would typically filter the output of summer 62. Additional filters (in loop or post loop) may also be used in addition to the deblocking filter. Such filters are not shown for brevity, but if desired, may filter the output of summer 50 (as an in-loop filter).

[0220] Video data memory 41 may store video data to be encoded by the components of video encoder 20. The video data stored in video data memory 41 may be obtained, for example, from video source 18. Decoded picture buffer 64 may be a reference picture

memory that stores reference video data for use in encoding video data by video encoder 20, e.g., in intra- or inter-coding modes. Video data memory 41 and decoded picture buffer 64 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 41 and decoded picture buffer 64 may be provided by the same memory device or separate memory devices. In various examples, video data memory 41 may be on-chip with other components of video encoder 20, or off-chip relative to those components.

**[0221]** During the encoding process, video encoder 20 receives a video frame or slice to be coded. The frame or slice may be divided into multiple video blocks. Motion estimation unit 42 and motion compensation unit 44 perform inter-predictive coding of the received video block relative to one or more blocks in one or more reference frames to provide temporal prediction. Intra prediction processing unit 46 may alternatively perform intra-predictive coding of the received video block relative to one or more neighboring blocks in the same frame or slice as the block to be coded to provide spatial prediction. Video encoder 20 may perform multiple coding passes, e.g., to select an appropriate coding mode for each block of video data.

**[0222]** Moreover, partition unit 48 may partition blocks of video data into sub-blocks, based on evaluation of previous partitioning schemes in previous coding passes. For example, partition unit 48 may initially partition a frame or slice into LCUs, and partition each of the LCUs into sub-CUs based on rate-distortion analysis (e.g., rate-distortion optimization). Mode select unit 40 may further produce a quadtree data structure indicative of partitioning of an LCU into sub-CUs. Leaf-node CUs of the quadtree may include one or more PUs and one or more TUs.

**[0223]** Mode select unit 40 may select one of the coding modes, intra or inter, e.g., based on error results, and provides the resulting intra- or inter-coded block to summer 50 to generate residual block data and to summer 62 to reconstruct the encoded block for use as a reference frame. Mode select unit 40 also provides syntax elements, such as motion vectors, intra-mode indicators, partition information, and other such syntax information, to entropy encoding unit 56.

**[0224]** Motion estimation unit 42 and motion compensation unit 44 may be highly integrated, but are illustrated separately for conceptual purposes. Motion estimation, performed by motion estimation unit 42, is the process of generating motion vectors,

which estimate motion for video blocks. A motion vector, for example, may indicate the displacement of a PU of a video block within a current video frame or picture relative to a predictive block within a reference picture (or other coded unit) relative to the current block being coded within the current picture (or other coded unit). A predictive block is a block that is found to closely match the block to be coded, in terms of pixel difference, which may be determined by sum of absolute difference (SAD), sum of square difference (SSD), or other difference metrics. In some examples, video encoder 20 may calculate values for sub-integer pixel positions of reference pictures stored in decoded picture buffer 64. For example, video encoder 20 may interpolate values of one-quarter pixel positions, one-eighth pixel positions, or other fractional pixel positions of the reference picture. Therefore, motion estimation unit 42 may perform a motion search relative to the full pixel positions and fractional pixel positions and output a motion vector with fractional pixel precision.

**[0225]** Motion estimation unit 42 calculates a motion vector for a PU of a video block in an inter-coded slice by comparing the position of the PU to the position of a predictive block of a reference picture. The reference picture may be selected from a first reference picture list (List 0) or a second reference picture list (List 1), each of which identify one or more reference pictures stored in decoded picture buffer 64. Motion estimation unit 42 sends the calculated motion vector to entropy encoding unit 56 and motion compensation unit 44.

**[0226]** Motion compensation, performed by motion compensation unit 44, may involve fetching or generating the predictive block based on the motion vector determined by motion estimation unit 42. Again, motion estimation unit 42 and motion compensation unit 44 may be functionally integrated, in some examples. Upon receiving the motion vector for the PU of the current video block, motion compensation unit 44 may locate the predictive block to which the motion vector points in one of the reference picture lists. Summer 50 forms a residual video block by subtracting pixel values of the predictive block from the pixel values of the current video block being coded, forming pixel difference values, as discussed below. In general, motion estimation unit 42 performs motion estimation relative to luma components, and motion compensation unit 44 uses motion vectors calculated based on the luma components for both chroma components and luma components. Mode select unit 40 may also generate syntax elements associated with the video blocks and the video slice for use by video decoder 30 in decoding the video blocks of the video slice.



**[0227]** Intra prediction processing unit 46 may intra-predict a current block, as an alternative to the inter-prediction performed by motion estimation unit 42 and motion compensation unit 44, as described above. In particular, intra prediction processing unit 46 may determine an intra-prediction mode to use to encode a current block. In some examples, intra prediction processing unit 46 may encode a current block using various intra-prediction modes, e.g., during separate encoding passes, and intra prediction processing unit 46 (or mode select unit 40, in some examples) may select an appropriate intra-prediction mode to use from the tested modes.

**[0228]** For example, intra prediction processing unit 46 may calculate rate-distortion values using a rate-distortion analysis for the various tested intra-prediction modes, and select the intra-prediction mode having the best rate-distortion characteristics among the tested modes. Rate-distortion analysis generally determines an amount of distortion (or error) between an encoded block and an original, unencoded block that was encoded to produce the encoded block, as well as a bit rate (that is, a number of bits) used to produce the encoded block. Intra prediction processing unit 46 may calculate ratios from the distortions and rates for the various encoded blocks to determine which intra-prediction mode exhibits the best rate-distortion value for the block.

**[0229]** After selecting an intra-prediction mode for a block, intra prediction processing unit 46 may provide information indicative of the selected intra-prediction mode for the block to entropy encoding unit 56. Entropy encoding unit 56 may encode the information indicating the selected intra-prediction mode. Video encoder 20 may include in the transmitted bitstream configuration data, which may include a plurality of intra-prediction mode index tables and a plurality of modified intra-prediction mode index tables (also referred to as codeword mapping tables), definitions of encoding contexts for various blocks, and indications of a most probable intra-prediction mode, an intra-prediction mode index table, and a modified intra-prediction mode index table to use for each of the contexts.

**[0230]** Video encoder 20 forms a residual video block by subtracting the prediction data from mode select unit 40 from the original video block being coded. Summer 50 represents the component or components that perform this subtraction operation. Transform processing unit 52 applies a transform, such as a discrete cosine transform (DCT) or a conceptually similar transform, to the residual block, producing a video block comprising residual transform coefficient values. Transform processing unit 52 may perform other transforms which are conceptually similar to DCT. Wavelet

transforms, integer transforms, sub-band transforms or other types of transforms could also be used. In any case, transform processing unit 52 applies the transform to the residual block, producing a block of residual transform coefficients. The transform may convert the residual information from a pixel value domain to a transform domain, such as a frequency domain. Transform processing unit 52 may send the resulting transform coefficients to quantization unit 54.

**[0231]** Quantization unit 54 quantizes the transform coefficients to further reduce bit rate. The quantization process may reduce the bit depth associated with some or all of the coefficients. The degree of quantization may be modified by adjusting a quantization parameter. In some examples, quantization unit 54 may then perform a scan of the matrix including the quantized transform coefficients. Alternatively, entropy encoding unit 56 may perform the scan.

**[0232]** Following quantization, entropy encoding unit 56 entropy codes the quantized transform coefficients. For example, entropy encoding unit 56 may perform context adaptive variable length coding (CAVLC), context adaptive binary arithmetic coding (CABAC), syntax-based context-adaptive binary arithmetic coding (SBAC), probability interval partitioning entropy (PIPE) coding or another entropy coding technique. In the case of context-based entropy coding, context may be based on neighboring blocks. Following the entropy coding by entropy encoding unit 56, the encoded bitstream may be transmitted to another device (e.g., video decoder 30) or archived for later transmission or retrieval.

**[0233]** Inverse quantization unit 58 and inverse transform processing unit 60 apply inverse quantization and inverse transformation, respectively, to reconstruct the residual block in the pixel domain, e.g., for later use as a reference block. Motion compensation unit 44 may calculate a reference block by adding the residual block to a predictive block of one of the frames of decoded picture buffer 64. Motion compensation unit 44 may also apply one or more interpolation filters to the reconstructed residual block to calculate sub-integer pixel values for use in motion estimation. Summer 62 adds the reconstructed residual block to the motion compensated prediction block produced by motion compensation unit 44 to produce a reconstructed video block for storage in decoded picture buffer 64. The reconstructed video block may be used by motion estimation unit 42 and motion compensation unit 44 as a reference block to inter-code a block in a subsequent video frame.

**[0234]** FIG. 11 is a block diagram illustrating an example of video decoder 30 that may implement the techniques of this disclosure. In particular, video decoder 30 may decode video data into a target color container that may then be processed by video post-processor unit 31, as described above. In the example of FIG. 11, video decoder 30 includes an entropy decoding unit 70, a video data memory 71, motion compensation unit 72, intra prediction processing unit 74, inverse quantization unit 76, inverse transform processing unit 78, decoded picture buffer 82 and summer 80. Video decoder 30 may, in some examples, perform a decoding pass generally reciprocal to the encoding pass described with respect to video encoder 20 (FIG. 10). Motion compensation unit 72 may generate prediction data based on motion vectors received from entropy decoding unit 70, while intra prediction processing unit 74 may generate prediction data based on intra-prediction mode indicators received from entropy decoding unit 70.

**[0235]** Video data memory 71 may store video data, such as an encoded video bitstream, to be decoded by the components of video decoder 30. The video data stored in video data memory 71 may be obtained, for example, from computer-readable medium 16, e.g., from a local video source, such as a camera, via wired or wireless network communication of video data, or by accessing physical data storage media. Video data memory 71 may form a coded picture buffer (CPB) that stores encoded video data from an encoded video bitstream. Decoded picture buffer 82 may be a reference picture memory that stores reference video data for use in decoding video data by video decoder 30, e.g., in intra- or inter-coding modes. Video data memory 71 and decoded picture buffer 82 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 71 and decoded picture buffer 82 may be provided by the same memory device or separate memory devices. In various examples, video data memory 71 may be on-chip with other components of video decoder 30, or off-chip relative to those components.

**[0236]** During the decoding process, video decoder 30 receives an encoded video bitstream that represents video blocks of an encoded video slice and associated syntax elements from video encoder 20. Entropy decoding unit 70 of video decoder 30 entropy decodes the bitstream to generate quantized coefficients, motion vectors or intra-prediction mode indicators, and other syntax elements. Entropy decoding unit 70

forwards the motion vectors to and other syntax elements to motion compensation unit 72. Video decoder 30 may receive the syntax elements at the video slice level and/or the video block level.

**[0237]** When the video slice is coded as an intra-coded (I) slice, intra prediction processing unit 74 may generate prediction data for a video block of the current video slice based on a signaled intra prediction mode and data from previously decoded blocks of the current frame or picture. When the video frame is coded as an inter-coded (i.e., B or P) slice, motion compensation unit 72 produces predictive blocks for a video block of the current video slice based on the motion vectors and other syntax elements received from entropy decoding unit 70. The predictive blocks may be produced from one of the reference pictures within one of the reference picture lists. Video decoder 30 may construct the reference picture lists, List 0 and List 1, using default construction techniques based on reference pictures stored in decoded picture buffer 82. Motion compensation unit 72 determines prediction information for a video block of the current video slice by parsing the motion vectors and other syntax elements, and uses the prediction information to produce the predictive blocks for the current video block being decoded. For example, motion compensation unit 72 uses some of the received syntax elements to determine a prediction mode (e.g., intra- or inter-prediction) used to code the video blocks of the video slice, an inter-prediction slice type (e.g., B slice or P slice), construction information for one or more of the reference picture lists for the slice, motion vectors for each inter-encoded video block of the slice, inter-prediction status for each inter-coded video block of the slice, and other information to decode the video blocks in the current video slice.

**[0238]** Motion compensation unit 72 may also perform interpolation based on interpolation filters. Motion compensation unit 72 may use interpolation filters as used by video encoder 20 during encoding of the video blocks to calculate interpolated values for sub-integer pixels of reference blocks. In this case, motion compensation unit 72 may determine the interpolation filters used by video encoder 20 from the received syntax elements and use the interpolation filters to produce predictive blocks.

**[0239]** Inverse quantization unit 76 inverse quantizes, i.e., de-quantizes, the quantized transform coefficients provided in the bitstream and decoded by entropy decoding unit 70. The inverse quantization process may include use of a quantization parameter  $QP_Y$  calculated by video decoder 30 for each video block in the video slice to determine a degree of quantization and, likewise, a degree of inverse quantization that should be

applied. Inverse transform processing unit 78 applies an inverse transform, e.g., an inverse DCT, an inverse integer transform, or a conceptually similar inverse transform process, to the transform coefficients in order to produce residual blocks in the pixel domain.

**[0240]** After motion compensation unit 72 generates the predictive block for the current video block based on the motion vectors and other syntax elements, video decoder 30 forms a decoded video block by summing the residual blocks from inverse transform processing unit 78 with the corresponding predictive blocks generated by motion compensation unit 72. Summer 80 represents the component or components that perform this summation operation. If desired, a deblocking filter may also be applied to filter the decoded blocks in order to remove blockiness artifacts. Other loop filters (either in the coding loop or after the coding loop) may also be used to smooth pixel transitions, or otherwise improve the video quality. The decoded video blocks in a given frame or picture are then stored in decoded picture buffer 82, which stores reference pictures used for subsequent motion compensation. Decoded picture buffer 82 also stores decoded video for later presentation on a display device, such as display device 32 of FIG. 1.

**[0241]** FIG. 12 is a flowchart showing one example video processing technique of the disclosure. The techniques of FIG. 12 may be performed by video encoder 20 and/or video pre-processor unit 19. In the example of FIG. 12, source device 12 may be configured to capture video data using a camera (1200). Video encoder 20 and/or video pre-processor unit 19 may be configured to perform a dynamic range adjustment process on video data using fixed-point computing (1210). Video encoder 20 and/or video pre-processor unit 19 may be further configured to generate one or more syntax elements that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process, relative to the dynamic range adjustment process, using fixed-point computing (1220).

**[0242]** In one example, video encoder 20 and/or video pre-processor unit 19 may be configured to generate the one or more syntax elements by generating the one or more syntax elements in one or more supplemental enhancement information (SEI) messages. In one example, the parameters comprise one or more of a range parameter, a scale parameter, or an offset parameter. In another example, the information indicates one or more of a first number of fractional bits used for determining the range parameter, a second number of fractional bits used for determining the scale parameter, and a third

number of fractional bits used for determining the offset parameter. In another example, the information includes a minimum value and a maximum value for one or more color components of the video data. In another example, the information includes an index to a predetermined range of sample values of the decoded video data.

**[0243]** FIG. 13 is a flowchart showing another example video processing technique of the disclosure. The techniques of FIG. 13 may be performed by video decoder 30 and/or video post-processor unit 31. In one example of the disclosure, video decoder 30 and/or video post-processor unit 31 may be configured to receive one or more syntax elements that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process (1300), and receive decoded video data (1310).

**[0244]** Video decoder 30 and/or video post-processor unit 31 may be further configured to determine parameters for an inverse dynamic range adjustment process from the received information (1320), and perform the inverse dynamic range adjustment process on the decoded video data using fixed-point computing in accordance with the information received and the determined parameters (1330). Destination device 14 may be further configured to display the decoded video data after performing the inverse dynamic range adjustment process on the decoded video data (1340).

**[0245]** In one example of the disclosure, video decoder 30 and/or video post-processor unit 31 may be configured to receive the one or more syntax elements in one or more supplemental enhancement information (SEI) messages. In one example, the parameters comprise one or more of a range parameter, a scale parameter, or an offset parameter. In another example, the information indicates one or more of a first number of fractional bits used for determining the range parameter, a second number of fractional bits used for determining the scale parameter, and a third number of fractional bits used for determining the offset parameter.

**[0246]** In another example of the disclosure, video decoder 30 and/or video post-processor unit 31 may be configured to determine the parameters, in the case that at least one of the first number of fractional bits, the second number of fractional bits, or the third number of fractional bits is different from one another, by accumulating any fractional bits during any intermediate calculation processes used to determine the parameters, and clip a final result for determining the parameters based on a predetermined fractional accuracy.

**[0247]** In another example of the disclosure, video decoder 30 and/or video post-processor unit 31 may be configured to determine the parameters by truncating any fractional bits over a desired fractional accuracy during all intermediate calculation processes used to determine the parameters.

**[0248]** In another example, the information includes a minimum value and a maximum value for one or more color components of the decoded video data, and video decoder 30 and/or video post-processor unit 31 may be configured to determine the parameters based on the received minimum value and the received maximum value.

**[0249]** In another example, the information includes an index to a predetermined range of sample values for one or more color components of the decoded video data, and video decoder 30 and/or video post-processor unit 31 may be configured to determine a minimum value and a maximum value for the one or more color components of the decoded video data based on the received index, and determine the parameters based on the determined minimum value and the determined maximum value.

**[0250]** In another example of the disclosure, video decoder 30 and/or video post-processor unit 31 may be configured to receive a syntax element indicating if the parameters are signed or unsigned, and perform a parsing process on the information in the SEI message, wherein the parsing process is the same regardless of the value of the syntax element.

**[0251]** Certain aspects of this disclosure have been described with respect to extensions of the HEVC standard for purposes of illustration. However, the techniques described in this disclosure may be useful for other video coding processes, including other standard or proprietary video coding processes not yet developed.

**[0252]** A video coder, as described in this disclosure, may refer to a video encoder or a video decoder. Similarly, a video coding unit may refer to a video encoder or a video decoder. Likewise, video coding may refer to video encoding or video decoding, as applicable.

**[0253]** 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.

**[0254]** 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.

**[0255]** By way of example, and not limitation, such computer-readable storage media can comprise RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage, or other magnetic storage devices, flash memory, or any other medium that can be used to store desired program code in the form of instructions or data structures and that can be accessed by a computer. Also, any connection is properly termed a computer-readable medium. For example, if instructions are transmitted from a website, server, or other remote source using a coaxial cable, fiber optic cable, twisted pair, digital subscriber line (DSL), or wireless technologies such as infrared, radio, and microwave, then the coaxial cable, fiber optic cable, twisted pair, DSL, or wireless technologies such as infrared, radio, and microwave are included in the definition of medium. It should be understood, however, that computer-readable storage media and data storage media do not include connections, carrier waves, signals, or other transitory media, but are instead directed to non-transitory, tangible storage media. Disk and disc, as used herein, includes compact disc (CD), laser disc, optical disc, digital versatile disc (DVD), floppy disk and Blu-ray disc, where disks usually reproduce data magnetically, while discs reproduce data optically with lasers. Combinations of the above should also be included within the scope of computer-readable media.

**[0256]** Instructions may be executed by one or more processors, such as one or more digital signal processors (DSPs), general purpose microprocessors, application specific integrated circuits (ASICs), field programmable logic arrays (FPGAs), or other



equivalent integrated or discrete logic circuitry. Accordingly, the term “processor,” as used herein may refer to any of the foregoing structure or any other structure suitable for implementation of the techniques described herein. In addition, in some aspects, the functionality described herein may be provided within dedicated hardware and/or software modules configured for encoding and decoding, or incorporated in a combined codec. Also, the techniques could be fully implemented in one or more circuits or logic elements.

**[0257]** 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.

**[0258]** Various examples have been described. These and other examples are within the scope of the following claims.

**WHAT IS CLAIMED IS:**

1. A method of processing video data, the method comprising:  
receiving one or more syntax elements that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process;  
receiving decoded video data; and  
performing the inverse dynamic range adjustment process on the decoded video data using fixed-point computing in accordance with the information received.
2. The method of claim 1, wherein receiving the one or more syntax elements comprises receiving the one or more syntax elements in one or more supplemental enhancement information (SEI) messages, and wherein the parameters comprise one or more of a range parameter, a scale parameter, or an offset parameter.
3. The method of claim 2, wherein the information indicates one or more of a first number of fractional bits used for determining the range parameter, a second number of fractional bits used for determining the scale parameter, and a third number of fractional bits used for determining the offset parameter.
4. The method of claim 3, further comprising:  
determining the parameters using the first number of fractional bits, the second number of fractions bits, and the third number of fractional bits.
5. The method of claim 4, wherein determining the parameters further comprises:  
determining the parameters, in the case that at least one of the first number of fractional bits, the second number of fractional bits, or the third number of fractional bits is different from one another, by accumulating any fractional bits during any intermediate calculation processes used to determine the parameters; and  
clipping a final result for determining the parameters based on a predetermined fractional accuracy.

6. The method of claim 4, wherein determining the parameters further comprises:  
determining the parameters by truncating any fractional bits over a desired fractional accuracy during all intermediate calculation processes used to determine the parameters.
7. The method of claim 1, wherein the information includes a minimum value and a maximum value for one or more color components of the decoded video data, the method further comprising:  
determining the parameters based on the received minimum value and the received maximum value.
8. The method of claim 1, wherein the information includes an index to a predetermined range of sample values for one or more color components of the decoded video data, the method further comprising:  
determining a minimum value and a maximum value for the one or more color components of the decoded video data based on the received index; and  
determining the parameters based on the determined minimum value and the determined maximum value.
9. The method of claim 1, further comprising:  
receiving a syntax element indicating if the parameters are signed or unsigned;  
and  
performing a parsing process on the information in the SEI message, wherein the parsing process is the same regardless of the value of the syntax element.
10. The method of claim 1, further comprising:  
displaying the decoded video data after performing the inverse dynamic range adjustment process on the decoded video data.

11. An apparatus configured to process video data, the apparatus comprising:  
a memory configured to store decoded video data; and  
one or more processors configured to:  
    receive one or more syntax elements that contain information specifying  
    how to determine parameters for performing an inverse dynamic range  
    adjustment process;  
    receive the decoded video data; and  
    perform the inverse dynamic range adjustment process on the decoded  
    video data using fixed-point computing in accordance with the information  
    received.
12. The apparatus of claim 11, wherein receiving the one or more processors are  
further configured to receive the one or more syntax elements in one or more  
supplemental enhancement information (SEI) messages, and wherein the parameters  
comprise one or more of a range parameter, a scale parameter, or an offset parameter.
13. The apparatus of claim 12, wherein the information indicates one or more of a  
first number of fractional bits used for determining the range parameter, a second  
number of fractional bits used for determining the scale parameter, and a third number  
of fractional bits used for determining the offset parameter.
14. The apparatus of claim 13, wherein the one or more processors are further  
configured to:  
    determine the parameters using the first number of fractional bits, the second  
    number of fractions bits, and the third number of fractional bits.
15. The apparatus of claim 14, wherein to determine the parameters, the one or more  
processors are further configured to:  
    determine the parameters, in the case that at least one of the first number of  
    fractional bits, the second number of fractional bits, or the third number of fractional  
    bits is different from one another, by accumulating any fractional bits during any  
    intermediate calculation processes used to determine the parameters; and  
    clip a final result for determining the parameters based on a predetermined  
    fractional accuracy.

16. The apparatus of claim 14, wherein to determine the parameters, the one or more processors are further configured to:

determine the parameters by truncating any fractional bits over a desired fractional accuracy during all intermediate calculation processes used to determine the parameters.

17. The apparatus of claim 11, wherein the information includes a minimum value and a maximum value for one or more color components of the decoded video data, and wherein the one or more processors are further configured to:

determine the parameters based on the received minimum value and the received maximum value.

18. The apparatus of claim 11, wherein the information includes an index to a predetermined range of sample values of one or more color components of the decoded video data, and wherein the one or more processors are further configured to:

determine a minimum value and a maximum value for the one or more color components of the decoded video data based on the received index; and

determine the parameters based on the determined minimum value and the determined maximum value.

19. The apparatus of claim 11, wherein the one or more processors are further configured to:

receive a syntax element indicating if the parameters are signed or unsigned; and  
perform a parsing process on the information in the SEI message, wherein the parsing process is the same regardless of the value of the syntax element.

20. The apparatus of claim 11, the apparatus further comprising:

a display configured to display the decoded video data after the one or more processors perform the inverse dynamic range adjustment process on the decoded video data.

21. The apparatus of claim 11, wherein the apparatus comprises one or more of a camera, a computer, a mobile device, a broadcast receiver device, or a set-top box.

22. An apparatus configured to process video data, the apparatus comprising:  
means for receiving one or more syntax elements that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process;  
means for receiving decoded video data; and  
means for performing the inverse dynamic range adjustment process on the decoded video data using fixed-point computing in accordance with the information received.
23. The apparatus of claim 22, wherein the means for receiving the one or more syntax elements comprises means for receiving the one or more syntax elements in one or more supplemental enhancement information (SEI) messages, and wherein the parameters comprise one or more of a range parameter, a scale parameter, or an offset parameter.
24. The apparatus of claim 23, wherein the information indicates one or more of a first number of fractional bits used for determining the range parameter, a second number of fractional bits used for determining the scale parameter, and a third number of fractional bits used for determining the offset parameter.
25. The apparatus of claim 24, further comprising:  
means for determining the parameters using the first number of fractional bits, the second number of fractions bits, and the third number of fractional bits.
26. The apparatus of claim 25, wherein the means for determining the parameters further comprises:  
means for determining the parameters, in the case that at least one of the first number of fractional bits, the second number of fractional bits, or the third number of fractional bits is different from one another, by accumulating any fractional bits during any intermediate calculation processes used to determine the parameters; and  
means for clipping a final result for determining the parameters based on a predetermined fractional accuracy.

27. The apparatus of claim 25, wherein the means for determining the parameters further comprises:

means for determining the parameters by truncating any fractional bits over a desired fractional accuracy during all intermediate calculation processes used to determine the parameters.

28. The apparatus of claim 22, wherein the information includes a minimum value and a maximum value for one or more color components of the decoded video data, the apparatus further comprising:

means for determining the parameters based on the received minimum value and the received maximum value.

29. The apparatus of claim 22, wherein the information includes an index to a predetermined range of sample values or one or more color components of the decoded video data, the apparatus further comprising:

means for determining a minimum value and a maximum value for the one or more color components of the decoded video data based on the received index; and

means for determining the parameters based on the determined minimum value and the determined maximum value.

30. The apparatus of claim 22, further comprising:

means for receiving a syntax element indicating if the parameters are signed or unsigned; and

means for performing a parsing process on the information in the SEI message, wherein the parsing process is the same regardless of the value of the syntax element.

31. The apparatus of claim 22, further comprising:

means for displaying the decoded video data after performing the inverse dynamic range adjustment process on the decoded video data.

32. A computer-readable storage medium storing instructions that, when executed, cause one or more processors of a device configured to process video data to:

receive one or more syntax elements that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process;

receive the decoded video data; and

perform the inverse dynamic range adjustment process on the decoded video data using fixed-point computing in accordance with the information received.

33. The computer-readable storage medium of claim 32, wherein the instructions further cause the one or more processors to receive the one or more syntax elements in one or more supplemental enhancement information (SEI) messages, and wherein the parameters comprise one or more of a range parameter, a scale parameter, or an offset parameter.

34. The computer-readable storage medium of claim 33, wherein the information indicates one or more of a first number of fractional bits used for determining the range parameter, a second number of fractional bits used for determining the scale parameter, and a third number of fractional bits used for determining the offset parameter.

35. The computer-readable storage medium of claim 34, wherein the instructions further cause the one or more processors to:

determine the parameters using the first number of fractional bits, the second number of fractions bits, and the third number of fractional bits.

36. The computer-readable storage medium of claim 35, wherein the instructions further cause the one or more processors to:

determine the parameters, in the case that at least one of the first number of fractional bits, the second number of fractional bits, or the third number of fractional bits is different from one another, by accumulating any fractional bits during any intermediate calculation processes used to determine the parameters; and

clip a final result for determining the parameters based on a predetermined fractional accuracy.



37. The computer-readable storage medium of claim 35, wherein the instructions further cause the one or more processors to:

determine the parameters by truncating any fractional bits over a desired fractional accuracy during all intermediate calculation processes used to determine the parameters.

38. The computer-readable storage medium of claim 32, wherein the information includes a minimum value and a maximum value for one or more color components of the decoded video data, and wherein the instructions further cause the one or more processors to:

determine the parameters based on the received minimum value and the received maximum value.

39. The computer-readable storage medium of claim 32, wherein the information includes an index to a predetermined range of sample values of one or more color components of the decoded video data, wherein the instructions further cause the one or more processors to:

determine a minimum value and a maximum value for the one or more color components of the decoded video data based on the received index; and

determine the parameters based on the determined minimum value and the determined maximum value.

40. The computer-readable storage medium of claim 32, wherein the instructions further cause the one or more processors to:

receive a syntax element indicating if the parameters are signed or unsigned; and  
perform a parsing process on the information in the SEI message, wherein the parsing process is the same regardless of the value of the syntax element.

41. The computer-readable storage medium of claim 32, wherein the instructions further cause the one or more processors to:

display the decoded video data after performing the inverse dynamic range adjustment process on the decoded video data.

42. A method of processing video data, the method comprising:  
performing a dynamic range adjustment process on video data using fixed-point computing; and  
generating one or more syntax elements that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process, relative to the dynamic range adjustment process, using fixed-point computing.
43. The method of claim 42, wherein generating the one or more syntax elements comprises generating the one or more syntax elements in one or more supplemental enhancement information (SEI) messages, and wherein the parameters comprise one or more of a range parameter, a scale parameter, or an offset parameter.
44. The method of claim 43, wherein the information indicates one or more of a first number of fractional bits used for determining the range parameter, a second number of fractional bits used for determining the scale parameter, and a third number of fractional bits used for determining the offset parameter.
45. The method of claim 42, wherein the information includes a minimum value and a maximum value for one or more color components of the video data.
46. The method of claim 42, wherein the information includes an index to a predetermined range of sample values of the decoded video data.
47. The method of claim 42, further comprising:  
capturing the video data with a camera.

48. An apparatus configured to process video data, the apparatus comprising:  
a memory configured to store video data; and  
one or more processors configured to:  
perform a dynamic range adjustment process on the video data using  
fixed-point computing; and  
generate one or more syntax elements that contain information specifying  
how to determine parameters for performing an inverse dynamic range  
adjustment process, relative to the dynamic range adjustment process, using  
fixed-point computing.
49. The apparatus of claim 48, wherein the one or more processors are further  
configured to generate the one or more syntax elements in one or more supplemental  
enhancement information (SEI) messages, and wherein the parameters comprise one or  
more of a range parameter, a scale parameter, or an offset parameter.
50. The apparatus of claim 49, wherein the information indicates one or more of a  
first number of fractional bits used for determining the range parameter, a second  
number of fractional bits used for determining the scale parameter, and a third number  
of fractional bits used for determining the offset parameter.
51. The apparatus of claim 48, wherein the information includes a minimum value  
and a maximum value for one or more color components of the video data.
52. The apparatus of claim 48, wherein the information includes an index to a  
predetermined range of sample values of the decoded video data.
53. The apparatus of claim 48, the apparatus further comprising:  
a camera configured to capture the video data.
54. The apparatus of claim 48, wherein the apparatus comprises one or more of a  
camera, a computer, a mobile device, a broadcast receiver device, or a set-top box.

55. An apparatus configured to process video data, the apparatus comprising:  
means for performing a dynamic range adjustment process on video data using fixed-point computing; and  
means for generating one or more syntax elements that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process, relative to the dynamic range adjustment process, using fixed-point computing.
56. A computer-readable storage medium storing instructions that, when executed, cause one or more processors of a device configured to process video data to:  
perform a dynamic range adjustment process on the video data using fixed-point computing; and  
generate one or more syntax elements that contain information specifying how to determine parameters for performing an inverse dynamic range adjustment process, relative to the dynamic range adjustment process, using fixed-point computing.

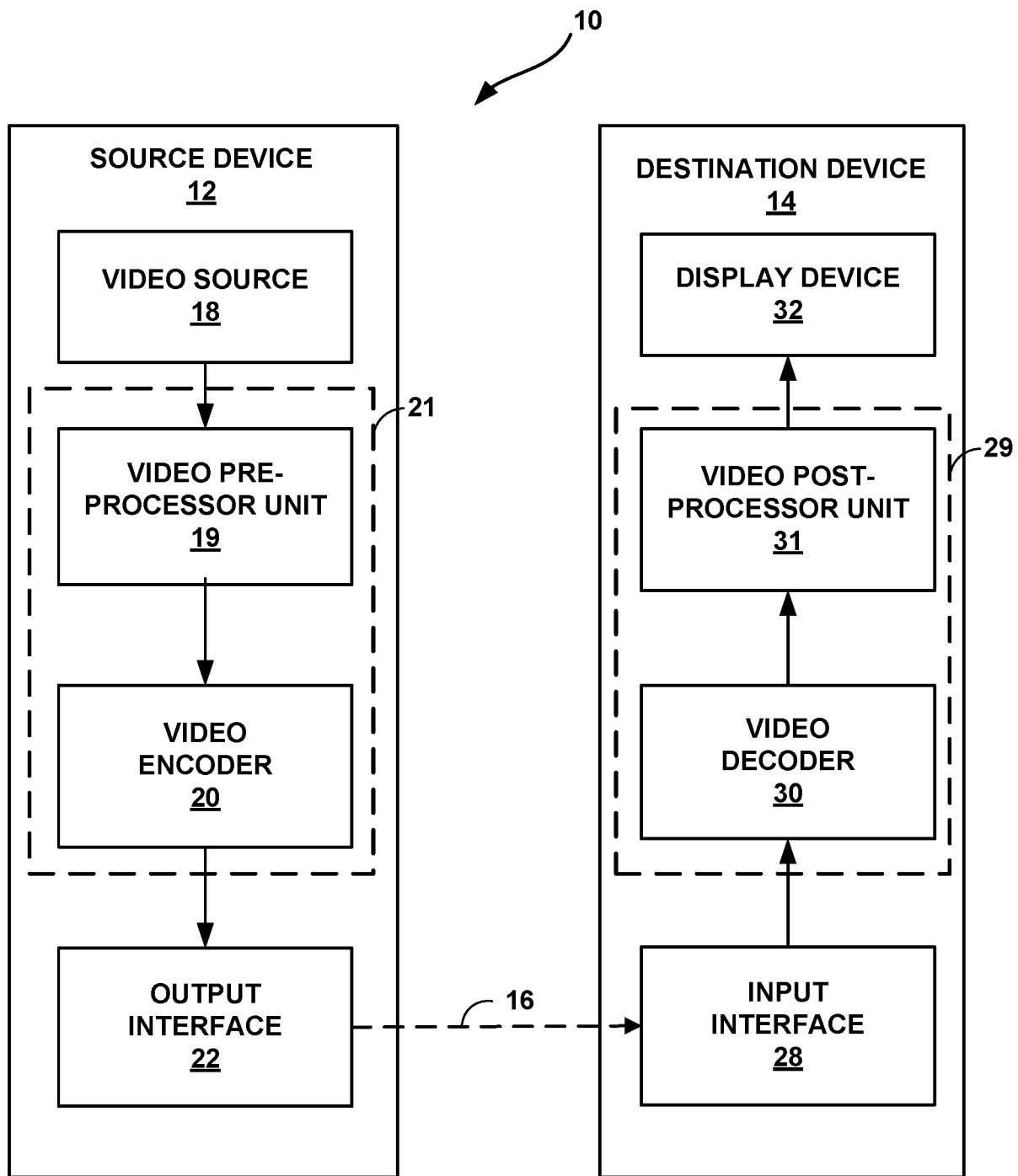


FIG. 1

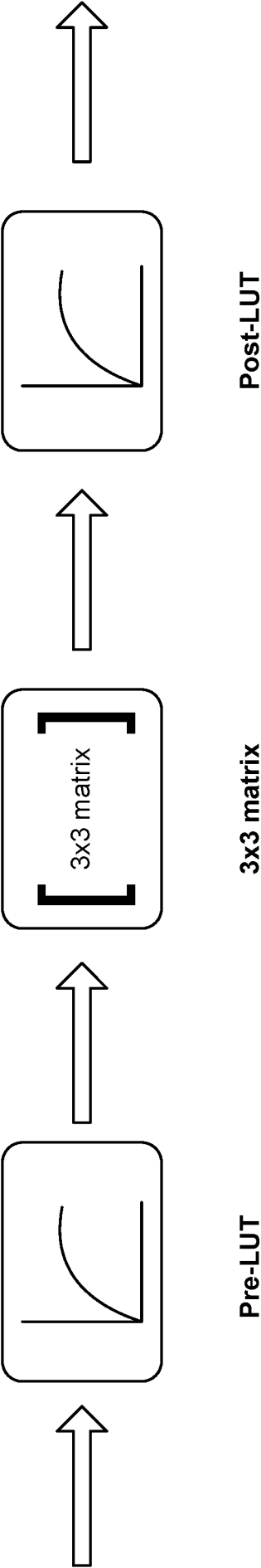


FIG. 2

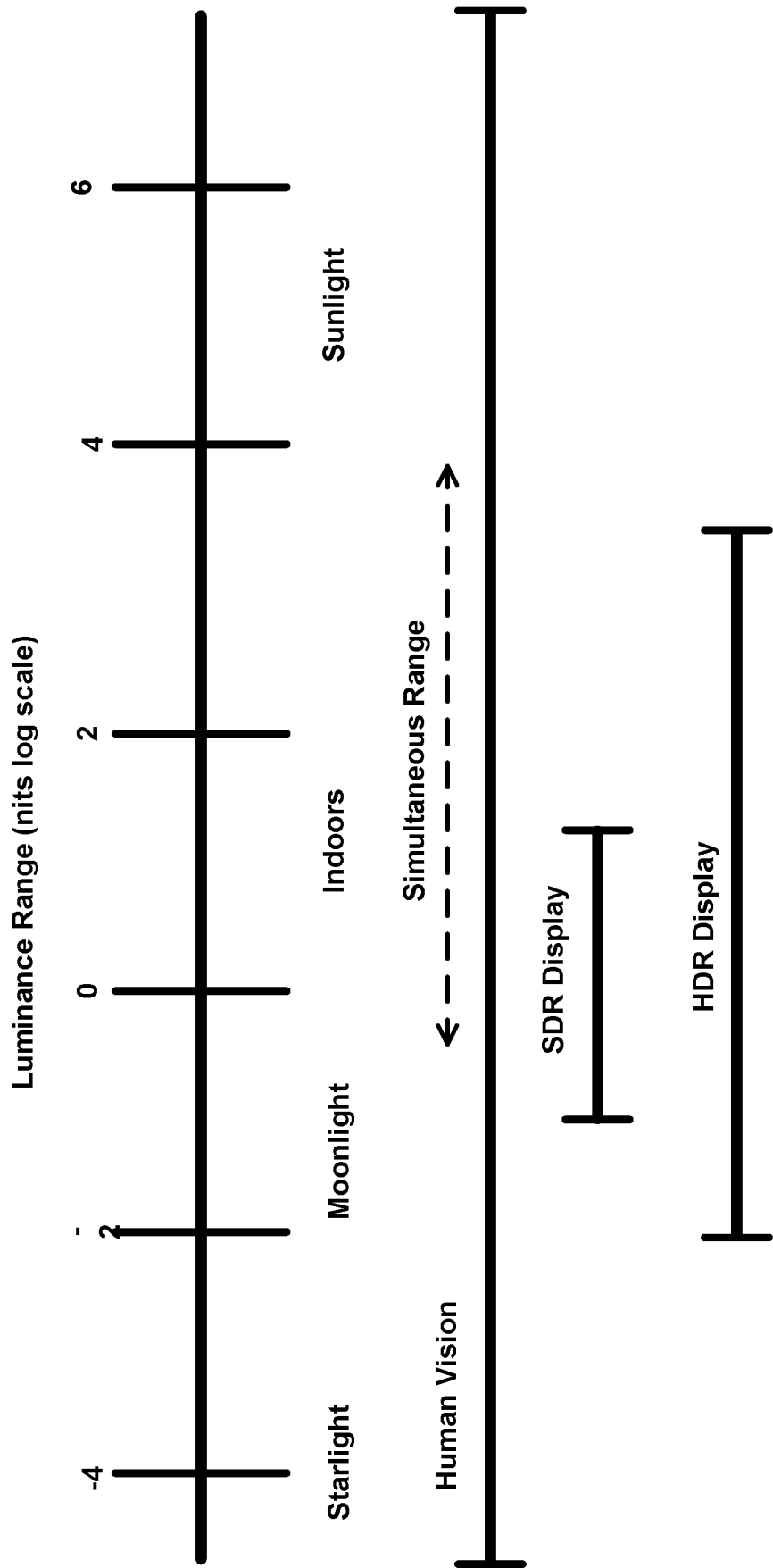


FIG. 3

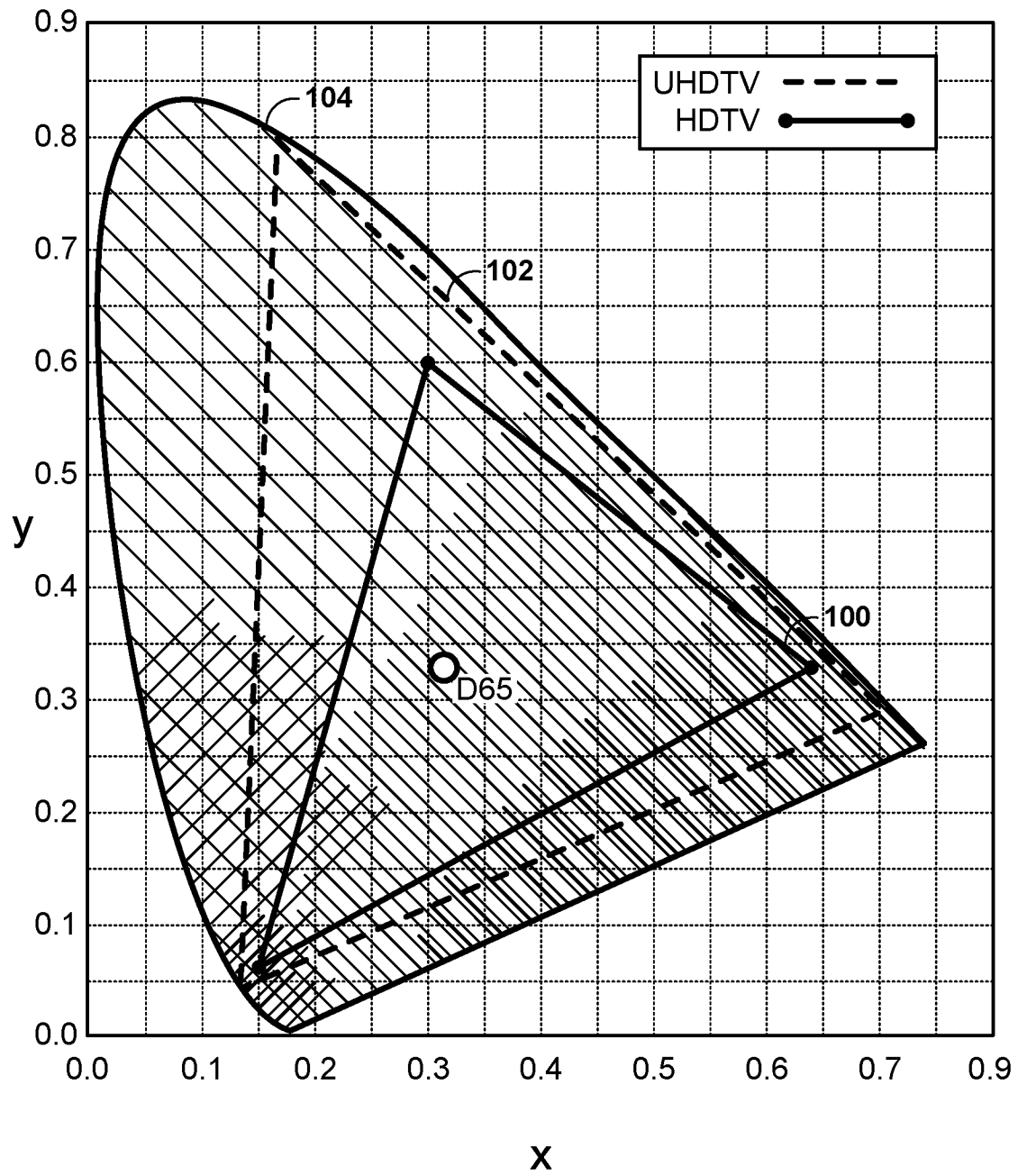


FIG. 4



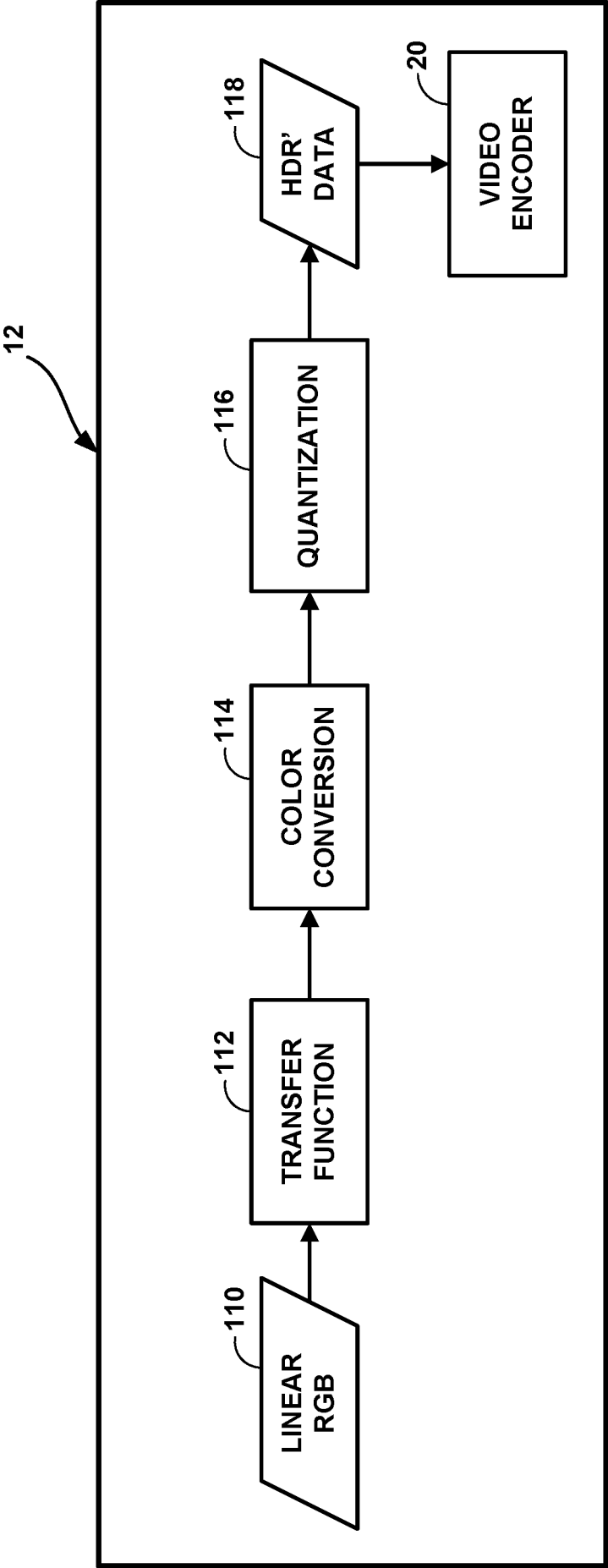


FIG. 5

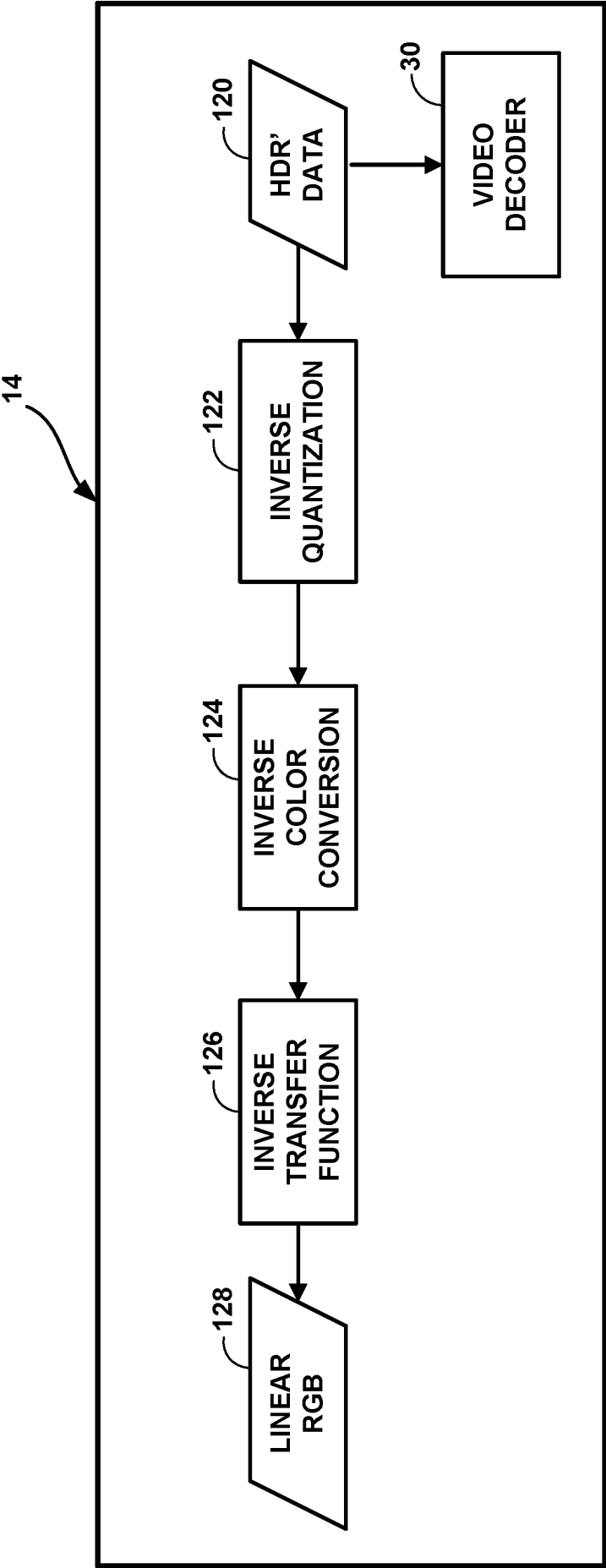
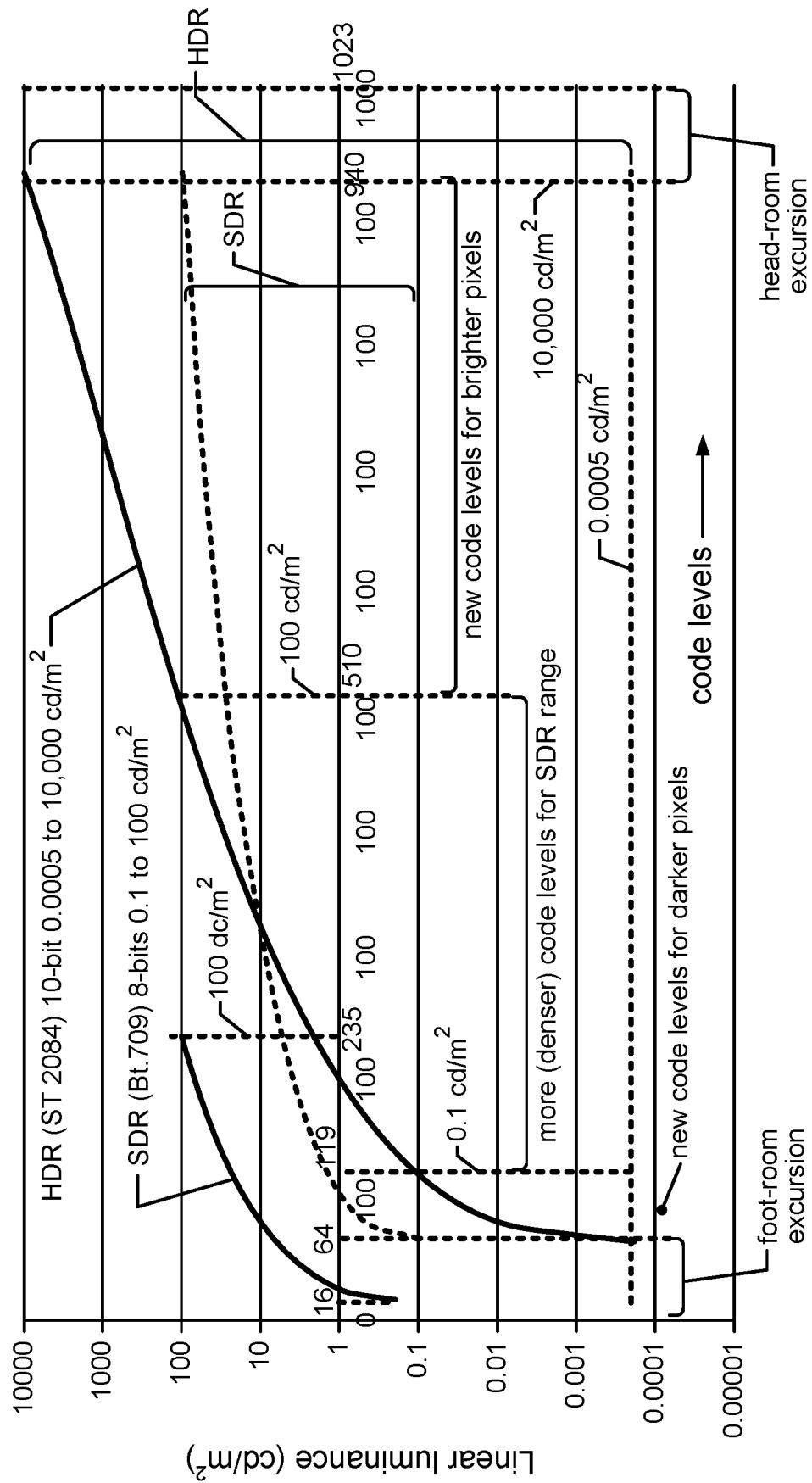


FIG. 6



Example of EOTFs

FIG. 7

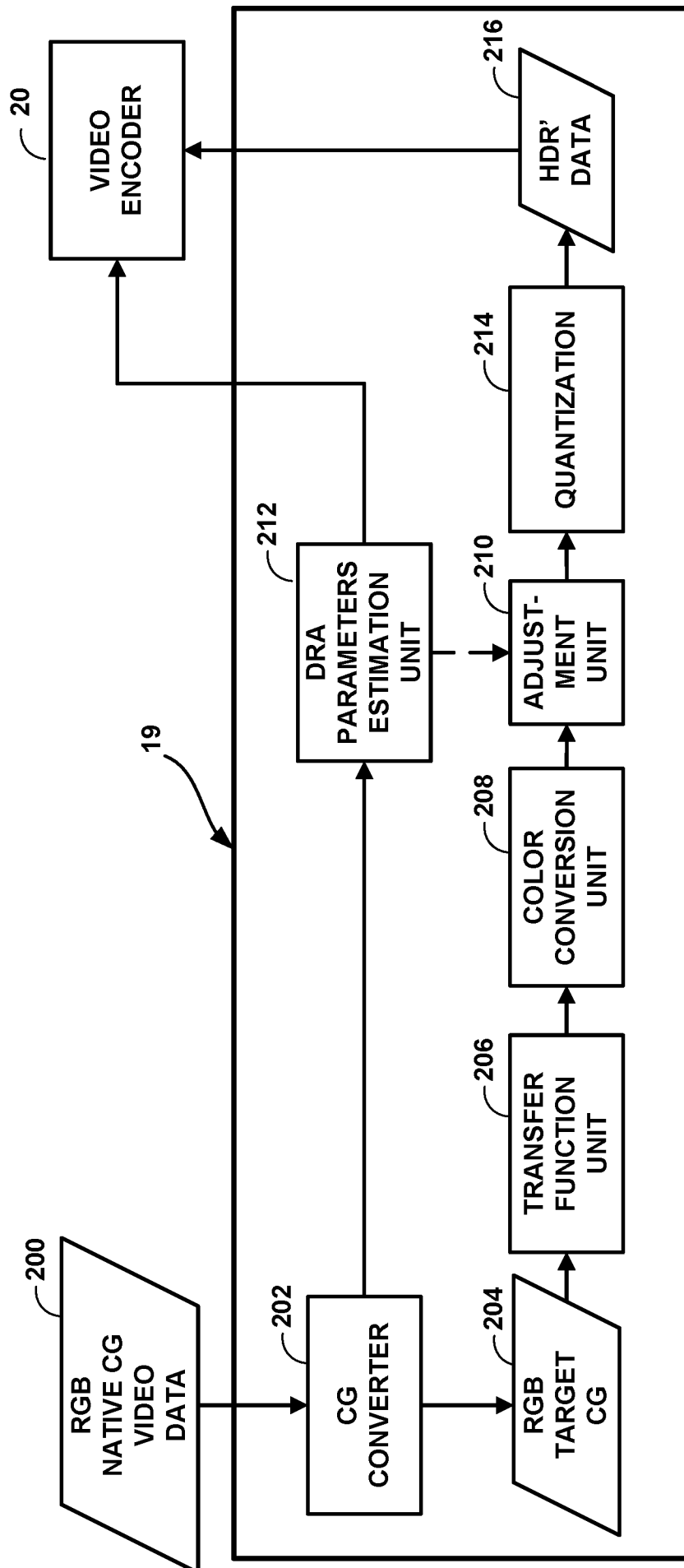


FIG. 8

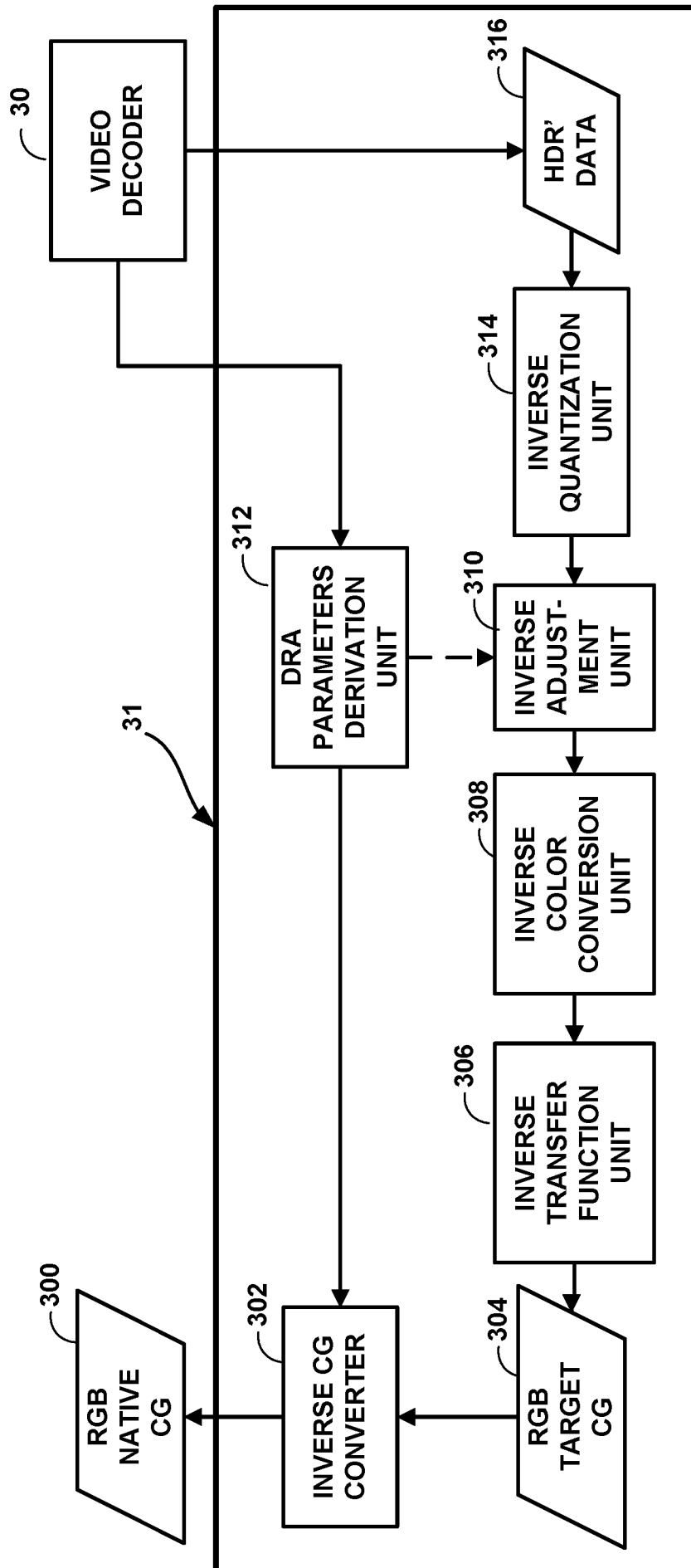
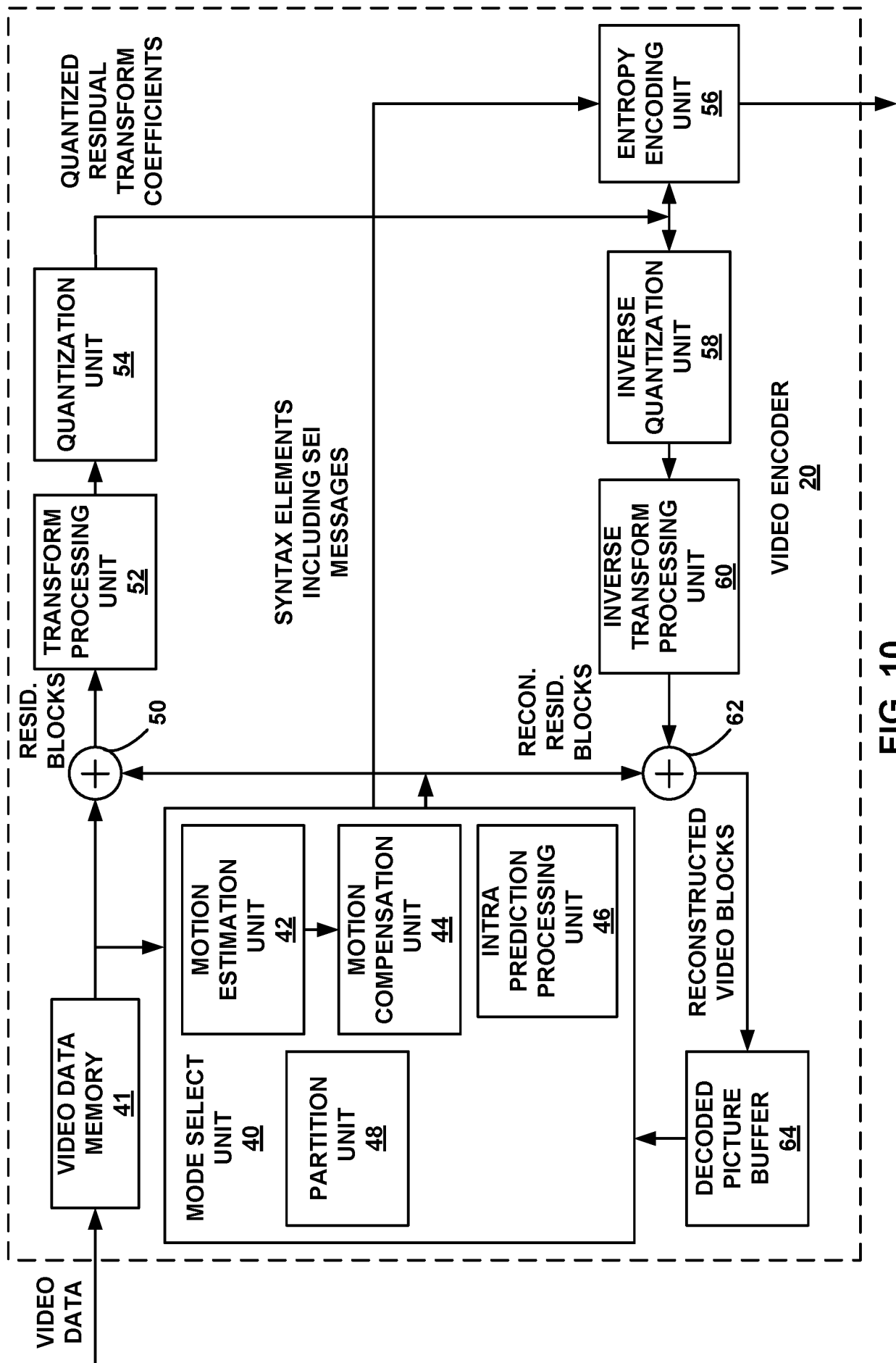


FIG. 9



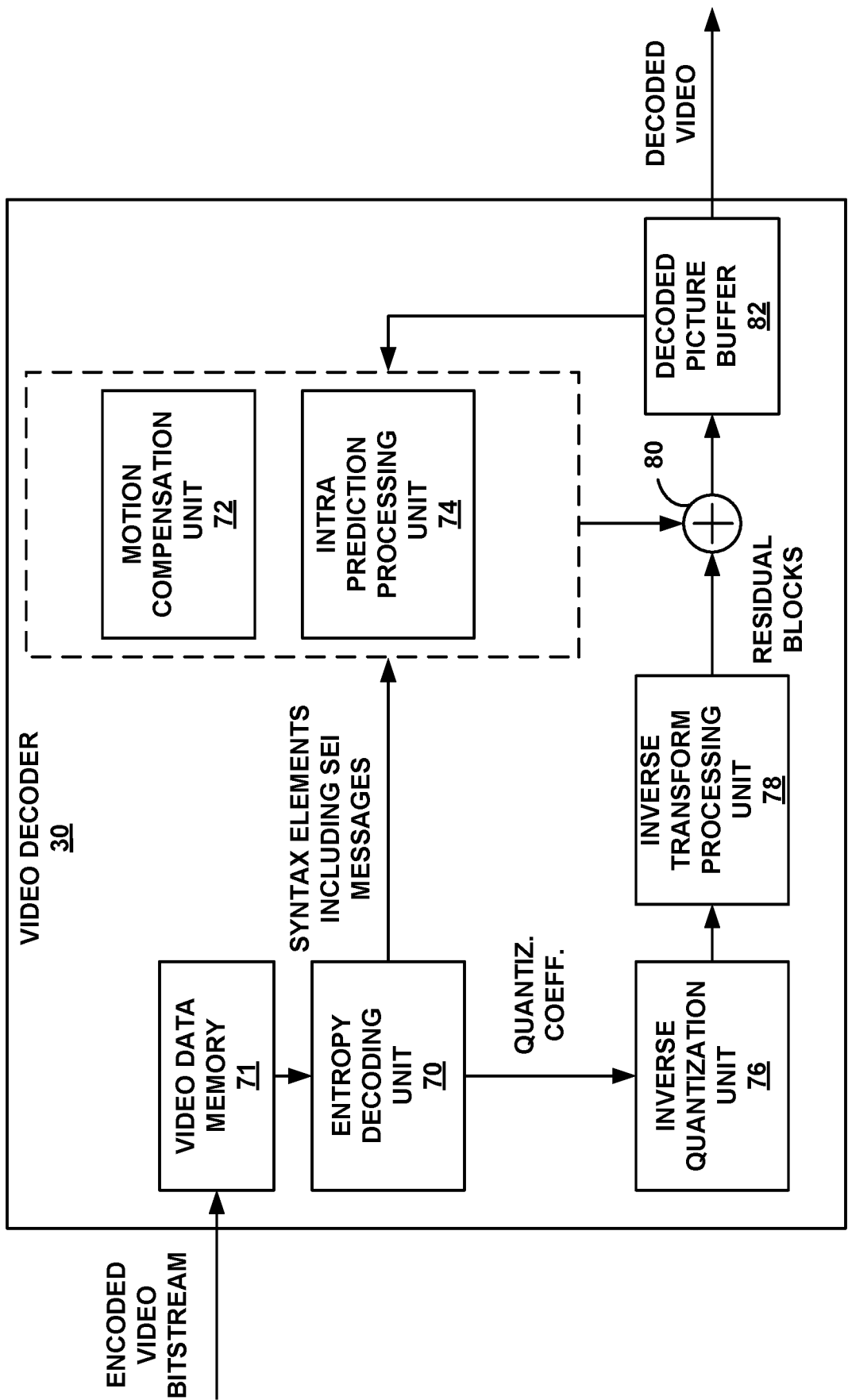


FIG. 11

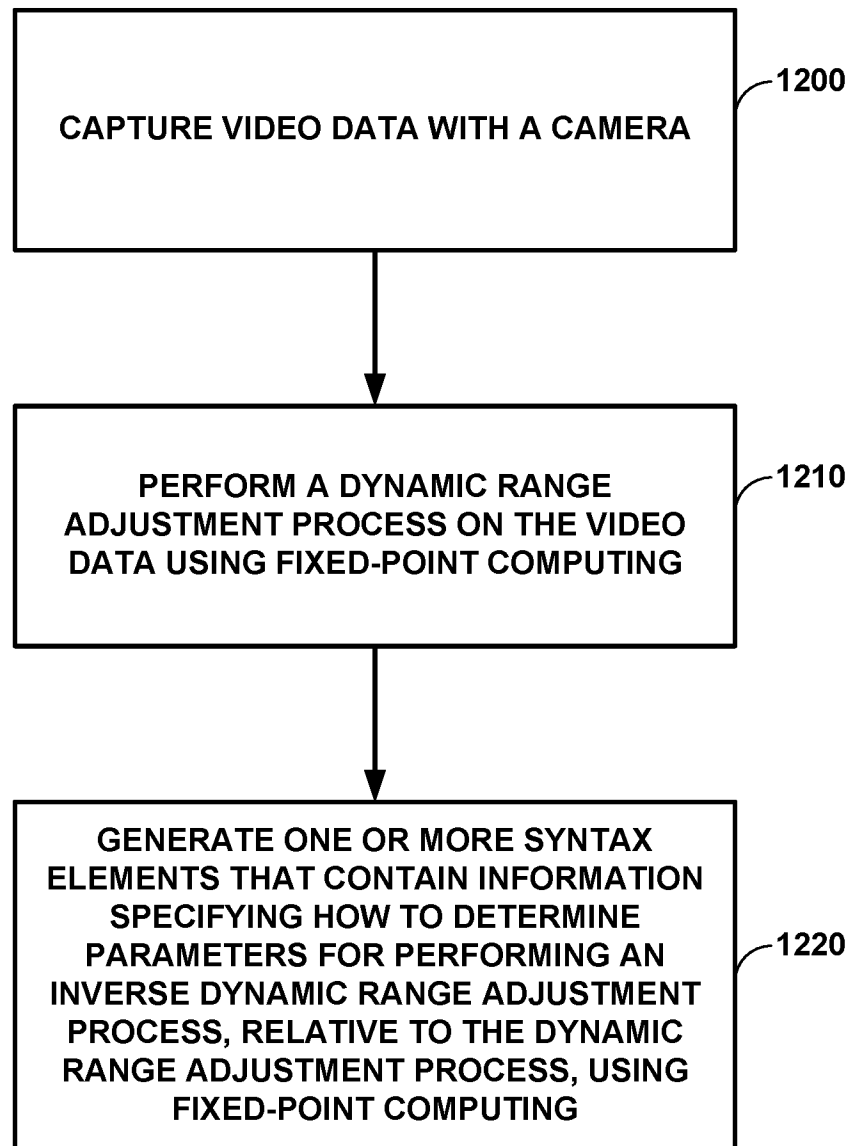


FIG. 12



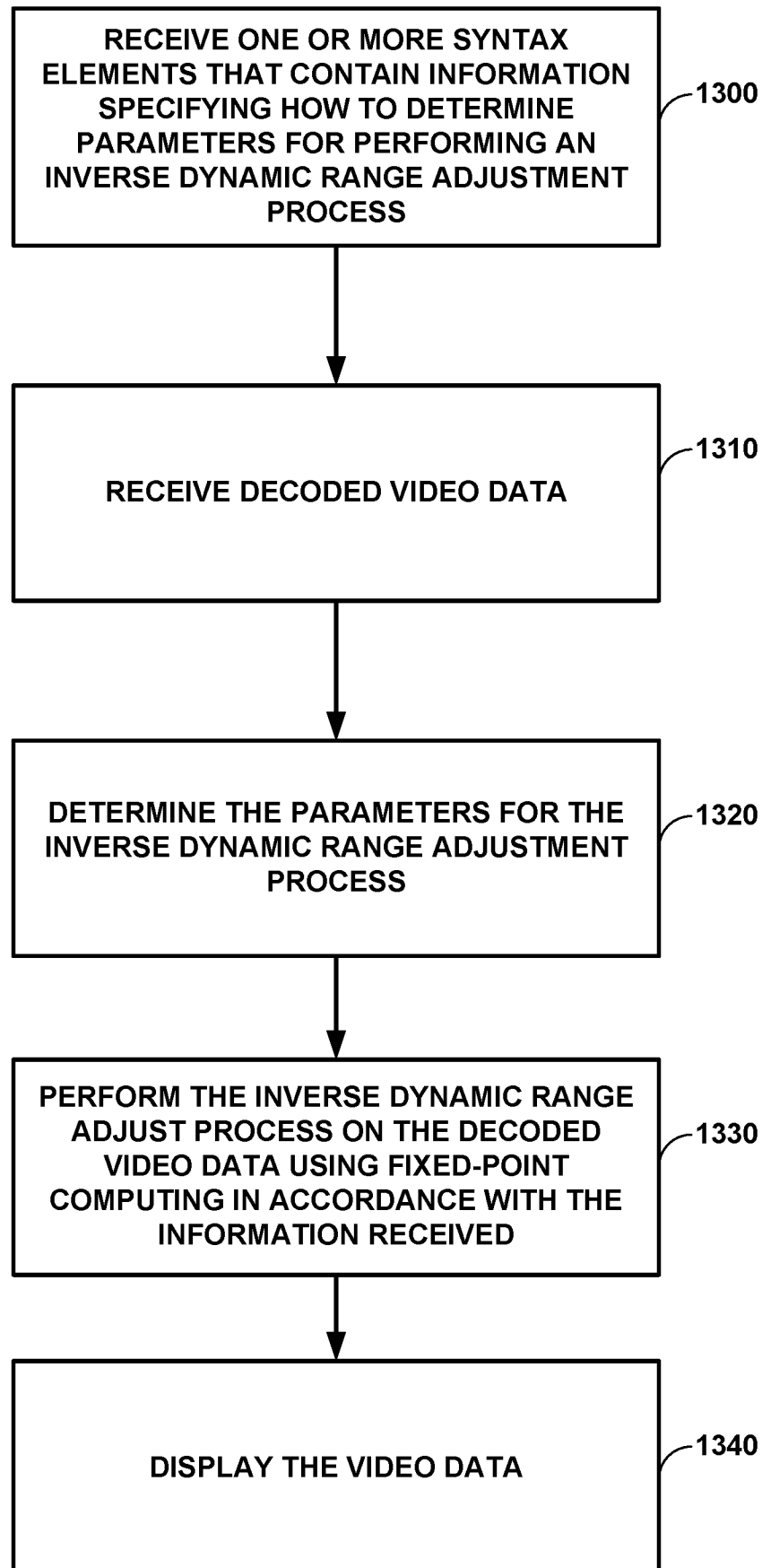


FIG. 13